Comparison of Path Planning Solutions for Autonomous Ground Vehicles in Unstructured Environments

Aimé Zo Randriamoramanana

March 2025

Abstract

Path planning in unstructured environments poses significant challenges for autonomous ground vehicles (AGVs), particularly in terrains affected by heavy rainfall. Such conditions introduce dynamic and complex factors, including reduced traction and obscured navigation landmarks. This paper focuses on the development and evaluation of path planning algorithms adapted to AGVs operating in rain-soaked and muddy environments. A comparative study of existing deterministic and learning-based approaches is conducted, with a focus on their applicability to agricultural robots navigating in muddy soils. The study includes testing and benchmarking different solutions to identify the most effective methods for ensuring safe and reliable navigation in these challenging conditions. By addressing the unique challenges of rain-affected terrains, this work advances the understanding of autonomous navigation for agricultural applications.

1 Introduction

Autonomous Ground Vehicles (AGVs) are increasingly being deployed in agricultural and off-road environments. However, unstructured terrains, particularly those affected by rain, pose significant challenges due to changes in traction and visibility. This paper aims to compare different path planning solutions to enhance AGV navigation in such conditions.

2 Related Work

Several studies have explored the planning of routes for AGVs. Wang et al. (2024) provide a comprehensive survey of approaches in unstructured environments [**?**]. Deterministic methods such as A* and Dijkstra's algorithm provide reliable paths but struggle with dynamic obstacles, while learning-based approaches leverage reinforcement learning for adaptability.

2.1 State of the Art on Terrain Traversability Contact Models

Modeling terrain traversability is a crucial aspect of autonomous ground vehicle navigation, particularly in unstructured environments where soil properties can significantly affect vehicle mobility. Several approaches have been developed to understand and predict the interaction between a vehicle and the underlying terrain.

One of the fundamental methods for modeling terrain interaction is based on soil mechanics, where key parameters such as cohesion, internal friction angle, and soil consistency are used to assess traversability [1]. These properties determine the soil's load-bearing capacity and its deformation behavior under applied stress.

Empirical and semi-empirical models, such as those developed by Bekker, introduce terrain mobility indices by incorporating vehicle-specific parameters with soil characteristics [2]. These models provide a practical approach to estimating vehicle mobility across different terrains but often require experimental calibration.

Numerical approaches, particularly finite element methods (FEM), have been extensively employed to simulate wheel-soil interaction and predict soil deformation under dynamic loading [3]. These models integrate soil rheology, considering both plastic and viscous behaviors, which are essential for understanding the traversability of muddy or loose soils.

Plasticity theory is another critical aspect in modeling terrain deformation, where failure criteria such as Mohr-Coulomb and Drucker-Prager theories are used to predict soil failure under varying loading conditions [4]. These models are particularly useful for evaluating stability in off-road navigation scenarios.

Finally, rheological models describe the flow behavior of complex terrain, especially in muddy conditions, by incorporating yield stress and viscosity parameters [5]. These models allow for a more accurate representation of soil response under vehicle loading and environmental influences.

In order to evaluate the traversability of muddy terrain for autonomous exploration robots, we employ the Bekker model, a well-established approach in terramechanics for estimating soil-vehicle interactions [2]. This model provides a mathematical framework for predicting the pressure exerted by a wheel on deformable terrain, which is crucial for assessing traction and sinkage in muddy conditions.

The normal pressure beneath the wheel is given by:

$$p = k_c \left(\frac{b}{d}\right) + k_\phi \left(\frac{b}{d}\right)^n \tag{1}$$

where k_c and k_ϕ are empirical soil parameters, b is the wheel width, d is the sinkage depth, and n is an exponent dependent on the soil type. These

parameters allow us to model the mechanical response of the muddy terrain under wheel loading.

Additionally, the traction force can be estimated as:

$$F_t = W\mu(1 - e^{-J}) \tag{2}$$

where W represents the vehicle weight, μ is the soil-wheel friction coefficient, and J denotes the slip ratio. This formulation captures the effects of wheel slippage, which is a key factor in muddy terrains where excessive slip can lead to vehicle immobilization.

By integrating the Bekker model into our simulation, we analyze the impact of soil cohesion, friction, and deformability on the mobility of autonomous ground vehicles. The model provides insights into optimal traction control strategies and helps in the selection of suitable wheel configurations for challenging terrains.

2.2 Deterministic Path Planning in Muddy Environments

Path planning for autonomous ground vehicles (AGVs) in unstructured environments is a complex challenge, particularly in muddy terrains where vehicle dynamics and terrain interaction significantly impact navigation efficiency. Deterministic path planning methods, including graph-based and optimization-based approaches, are widely used due to their reliability in structured searches and global path optimization [6].

2.3 Graph-Based Path Planning Methods

Graph-based approaches such as Dijkstra's algorithm and A* are commonly applied in off-road navigation. These methods discretize the environment into a grid or graph representation, allowing the computation of the shortest path based on cost functions that integrate terrain traversability analysis (TTA) [7].

- **Dijkstra's Algorithm:** Guarantees the globally optimal path by exploring all possible routes, making it robust but computationally expensive for large-scale environments [8].
- **A* Algorithm:** Enhances Dijkstra's approach by incorporating a heuristic function to guide the search more efficiently, significantly reducing computational complexity while maintaining optimality [9].

In muddy environments, modifications to these algorithms involve the inclusion of slip ratios, terrain adhesion coefficients, and energy consumption models to account for the difficulty of traversing high-resistance areas [10].

2.4 Optimization-Based Path Planning Methods

Deterministic optimization methods, including Genetic Algorithms (GA) and Particle Swarm Optimization (PSO), have been utilized to improve path selection by minimizing energy costs and maximizing vehicle stability in deformable terrains [11]. These methods optimize path selection by:

- Reducing the risk of vehicle immobilization by incorporating terrain deformation models.
- Considering real-time adaptations to varying soil conditions through physics-based motion models.

Despite their effectiveness, these approaches require high computational resources, making them challenging for real-time applications in dynamically changing environments.

2.5 RRT-Based Path Planning in Muddy Environments

The Rapidly-exploring Random Tree (RRT) algorithm is a popular approach for path planning in high-dimensional spaces, making it well-suited for vehicle navigation in muddy and deformable terrains. RRT operates by incrementally constructing a tree that explores the environment from the initial configuration towards the goal configuration. Each tree node represents a state in the environment, while edges between nodes represent possible transitions or movements.

In muddy environments, modifications to the basic RRT algorithm are necessary to account for dynamic terrain features such as soil stiffness, wheel sinkage, and slip. These adaptations allow for better navigation in areas with high resistance, ensuring that the vehicle can plan a feasible and efficient path.

3 Methodology

This study evaluates 3 differents deterministics algorithms:

- A*
- Dijkstra
- Rapidly-exploring Random Trees (RRT)

Performance is assessed based on computational efficiency, robustness in muddy conditions, and adaptability to dynamic obstacles.

3.1 Terrain Modeling

The terrain is modeled using a two-dimensional grid of size $N \times N$, where each cell represents an area characterized by a difficulty index. The cell values are randomly drawn from the set $\{0, 1, 2, 10\}$ with respective probabilities that reflect the coexistence of easily traversable regions (values 0, 1, 2) and impassable obstacles (value 10). Thus, a cell with a value of 0 or 1 corresponds to an area with low resistance, a value of 2 indicates moderate difficulty, while a value of 10 represents areas that are considered completely untraversable. This approach allows for the evaluation of the performance of trajectory planning algorithms (such as A* or Dijkstra) in reproducing various routing scenarios in heterogeneous environments.

3.2 Vehicle and Muddy Terrain Modeling

The dynamics of the vehicle are modeled by a state vector

$$\mathbf{x} = \begin{pmatrix} x \\ y \\ \theta \\ v \end{pmatrix},$$

where x and y denote the planar position, θ is the orientation, and v represents the vehicle's speed. The state evolution is described by the following differential equations:

$$\begin{aligned} \dot{x} &= v\cos(\theta), \\ \dot{y} &= v\sin(\theta), \\ \dot{\theta} &= \frac{u_r - u_l}{L}, \\ \dot{v} &= a, \end{aligned}$$

where u_r and u_l are the commanded velocities for the right and left wheels respectively, L is the inter-axle distance, and a is the acceleration determined by the available traction force.

In muddy terrain, the terrain is represented by a grid map in which each cell is assigned a value corresponding to its difficulty. Typical values in the map are:

$$T \in \{0,\ 1,\ 2,\ 10\},$$

where lower values (e.g. 0 or 1) indicate relatively easy-to-traverse regions, 2 indicates moderate mud, and 10 represents impassable obstacles. The vehicle experiences a slowdown when traversing muddy terrain. This slowdown is modeled as a multiplicative factor applied to the motor command. Specifically, if the nominal motor command is u, then the effective command u_{eff} is given by:

$$u_{\text{eff}} = s(T)\, u,$$

with the slowdown factor $s(T)$ defined by

$$s(T) = \begin{cases} 1.00, & T = 0, \\ 0.80, & T = 1, \\ 0.50, & T = 2, \\ 0.01, & T = 10. \end{cases}$$

Thus, the vehicle's acceleration is modulated not only by its traction force F (obtained, for example, via the Bekker–Wong model) and mass m, but also by the effective motor command:

$$a = \frac{F}{m} \quad \text{with} \quad F = \text{traction\_force(slip ratio)},$$

and the commanded velocities are reduced by the factor $s(T)$ as the vehicle enters regions of higher mud difficulty. This combined modeling scheme allows us to capture both the nominal dynamics of the vehicle in free running conditions and the degraded performance in muddy or obstructed regions.

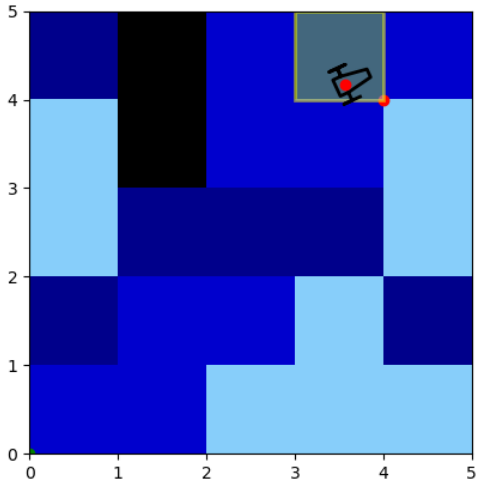


Figure 1: Simulation of a Dubins Car going to the destination point.

4 Experimental Results

Evaluation Metrics for Algorithm Comparison

To comprehensively compare the performance of the planning algorithms, we use the following metrics:

- **Computation Time:** The total time taken by the algorithm to compute a path for each run, which provides insight into its real-time feasibility.
- **Path Length:** The cumulative Euclidean distance between the successive nodes in the computed path. A shorter path generally indicates higher efficiency in terms of travel distance.
- **Energy Cost:** The energy expenditure is estimated by weighting each segment of the path by the terrain difficulty (cell value). This metric indicates how the roughness or "muddy" intensity of the environment increases the energy consumption.
- **Number of Replanifications:** The count of re-computations or retries performed when the algorithm fails to find a path in a given run. This metric reflects the stability and adaptability of the algorithm in dynamic or challenging environments.
- **Success Rate:** The percentage of runs in which a valid path is found, providing a high-level measure of the algorithm's reliability.

These metrics are collected over multiple runs on randomly generated terrain maps. The results are then visualized using plots, allowing for a detailed comparison of computational efficiency, route optimality, energy consumption, and overall robustness of each algorithm.

Metrics Comparison Figures

Figure 4 shows three key performance metrics collected over multiple runs of the planning algorithm. The figures display (a) the computation time (in seconds), (b) the overall path length, and (c) the accumulated energy cost, where the energy cost is calculated by weighting the distance of each path segment with the terrain difficulty.

Conclusion

The simulation results indicate that there is no significant difference in terms of computational (temporal) complexity among the tested approaches. However, a notable divergence is observed in energy consumption. In particular, the RRT algorithm tends to favor shorter paths, which, while reducing travel distance, does not always result in lower total energy consumption. This behavior suggests that, under the current formulation, the energy cost metric for RRT is heavily influenced by the terrain difficulty values rather than solely by the path length. Consequently, further tuning or a multi-objective

[b]0.3

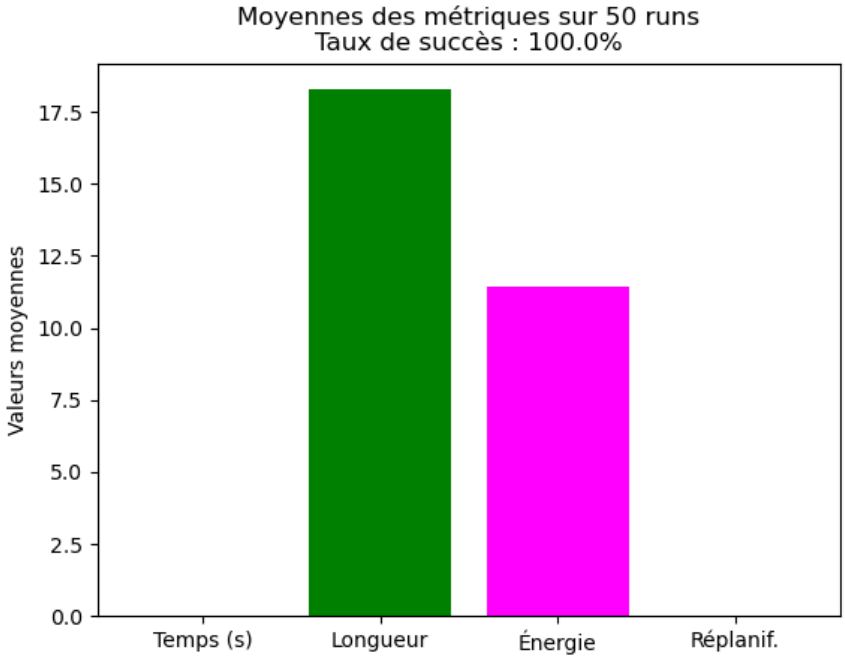


Figure 2: A*

[b]0.3

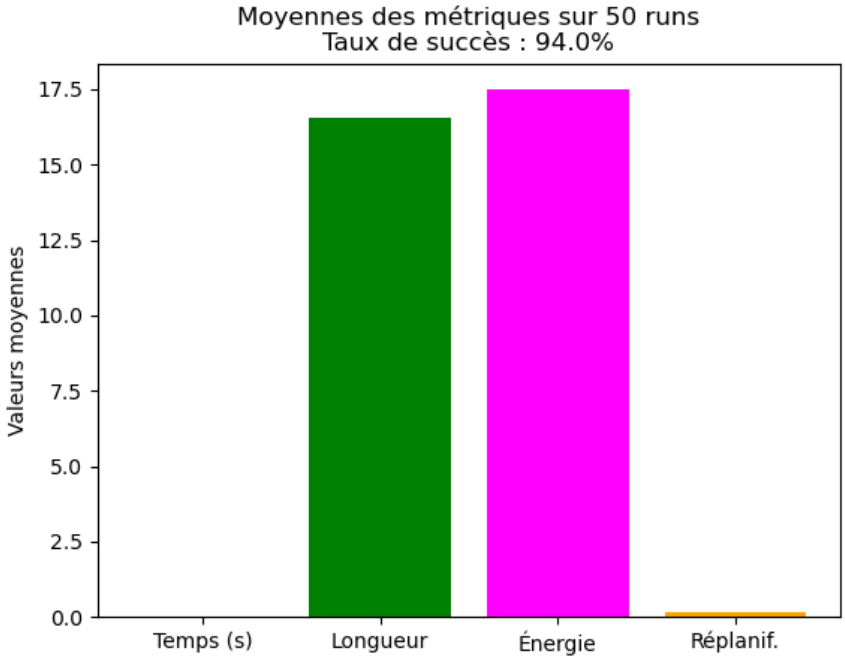


Figure 3: RRT

Figure 4: Comparison of metrics over multiple runs: computation time, path length, and energy cost. These metrics evaluate the efficiency and performance of the planning algorithms.

approach may be required to achieve an optimal balance between minimizing both the path length and the energy expenditure in muddy terrain scenarios.

This comparative study highlights the strengths and weaknesses of different path planning methods for AGVs in challenging environments. To enhance the result a good approach would be to fusion learning algorithm to compare with different approach as long as muddy estimation using sensors data.

References

[1] J. Atkinson and P. Bransby, *The Mechanics of Soils: An Introduction to Critical State Soil Mechanics*. McGraw-Hill, 1978.

[2] M. Bekker, *Introduction to Terrain-Vehicle Systems*. University of Michigan Press, 1969.

[3] L. Zhang and J. Wong, "Modeling of wheel-soil interaction over rough terrain—application to soil parameter identification," *Journal of Terramechanics*, vol. 45, no. 6, pp. 205–221, 2008.

[4] W. Chen, *Limit Analysis and Soil Plasticity*. Elsevier, 1975.

[5] Q. Nguyen and D. Boger, "Measuring the flow properties of yield stress fluids," *Annual Review of Fluid Mechanics*, vol. 24, no. 1, pp. 47–88, 1992.

[6] N. Wang, X. Li, K. Zhang, J. Wang, and D. Xie, "A survey on path planning for autonomous ground vehicles in unstructured environments," *Machines*, vol. 12, no. 1, p. 31, 2024.

[7] Y. Bai, B. Zhang, N. Xu, J. Zhou, J. Shi, and Z. Diao, "Vision-based navigation and guidance for agricultural autonomous vehicles and robots: A review," *Computers and Electronics in Agriculture*, vol. 205, p. 107584, 2023.

[8] S. Chakraborty, D. Elangovan, P. Govindarajan, M. ELnaggar, M. Alrashed, and S. Kamel, "A comprehensive review of path planning for agricultural ground robots," *Sustainability*, vol. 14, no. 9156, 2022.

[9] F. Oliveira, A. Neto, D. Howard, P. Borges, M. Campos, and D. Macharet, "Three-dimensional mapping with augmented navigation cost through deep learning," *Journal of Intelligent and Robotic Systems*, vol. 101, p. 50, 2021.

[10] F. Tian, R. Zhou, Z. Li, L. Li, Y. Gao, D. Cao, and L. Chen, "Trajectory planning for autonomous mining trucks considering terrain constraints," *IEEE Transactions on Intelligent Vehicles*, vol. 6, p. 772–786, 2021.

[11] G. Sakayori and G. Ishigami, "Energy efficient slope traversability planning for mobile robots in loose soil," in *IEEE International Conference on Mechatronics (ICM)*, 2017, p. 99–104.

Adaptive Reinforcement Learning-Driven MPC for Efficient Autonomous Vehicle Navigation

RICOUARD Ambre
3rd year engineering student
ENSTA
29200, Brest, FRANCE
ambre.ricouard@ensta.fr

***Abstract*—Traditional Proportional-Integral-Derivative (PID) controllers are widely used in control systems due to their simplicity and ease of tuning. However, they struggle with complex, multivariable, and constrained environments, limiting their effectiveness in dynamic scenarios. To overcome these limitations, Model Predictive Control (MPC) has emerged as a more advanced approach, capable of handling constraints while optimizing control inputs over a finite horizon. Despite its advantages, MPC requires careful tuning of numerous parameters, which heavily depend on the system dynamics and operating conditions. Improper tuning can lead to suboptimal performance or instability. Reinforcement Learning (RL) offers a promising solution by automating the tuning process, eliminating the need for manual expert adjustments, and enhancing adaptability across varying conditions. This paper investigates the integration of RL with MPC to dynamically optimize controller parameters, ensuring robust and efficient performance. Experimental results demonstrate that RL-enhanced MPC significantly outperforms traditional tuning approaches in terms of adaptability and robustness.**

***Index Terms*—Model Predictive Control, Reinforcement Learning, Autonomous Navigation, Optimization, Robotics**

I. Introduction

Control systems play a crucial role in various engineering fields, ranging from industrial automation to autonomous vehicles. Among the most widely used control strategies, the PID controller stands out due to its simplicity and ease of implementation. It consists of three main components: the proportional term, which reacts to the current error and provides immediate correction; the integral term, which accumulates past errors to eliminate steady-state deviations; and the derivative term, which anticipates future errors based on the rate of change, improving response stability.

Despite its intuitive design, the PID controller has significant limitations. It struggles with multivariable systems, time-varying dynamics, and constraint handling. Moreover, tuning the PID gains requires manual adjustments and empirical methods, making it suboptimal in complex or unpredictable environments. These limitations have led to the development of more advanced control strategies, such as MPC.

MPC is a powerful model-based approach that predicts future system behavior over a finite horizon and optimizes control inputs accordingly, while explicitly considering system constraints. This predictive nature makes MPC more robust and flexible than PID, especially in dynamic and constrained environments. However, MPC has its own challenges. The performance of an MPC controller is highly dependent on numerous tuning parameters, such as the weighting matrices in the cost function, the prediction horizon, and the constraints handling strategy. Determining these parameters manually is complex and time-consuming, often requiring expert knowledge and extensive trial-and-error processes. Furthermore, MPC assumes a relatively accurate model of the system, making it sensitive to modeling inaccuracies and changes in the operating conditions.

To address these challenges, RL has emerged as a promising solution for automating the tuning process of MPC parameters. Instead of relying on expert-defined settings, RL agents learn an optimal control policy through interaction with the environment, continuously adapting to varying conditions. By integrating RL with MPC, it is possible to enhance adaptability and robustness, allowing the controller to self-optimize in real time. This approach eliminates the need for manual tuning and enables the system to generalize better across different operating scenarios.

Several recent studies have explored the combination of RL and MPC to improve autonomous decision-making and control performance. For instance, in [1], the authors propose a weights-varying MPC approach for autonomous vehicle guidance using deep reinforcement learning. Their findings demonstrate that RL-enhanced MPC can significantly improve adaptability in dynamic environments by adjusting the weighting matrices in real time.

This paper explores the integration of RL-based tuning for MPC and demonstrates its effectiveness in improving control performance. We begin by reviewing existing control strategies and their limitations, followed by an implementation of RL-enhanced MPC. Experimental results are presented to highlight the benefits of this approach compared to traditional methods. Finally, we discuss potential improvements and future research directions in this field.

II. Model Predictive Control: Strengths and Limitations

A. Fundamentals of Model Predictive Control

MPC is an advanced control approach that optimizes system inputs over a finite prediction horizon while ensuring compliance with constraints. Unlike classical control methods that

react to current errors, MPC anticipates future system behavior by solving an optimization problem at each control step.

1) Principle of MPC: MPC relies on a dynamic model of the system to predict its future evolution over a prediction horizon N. At each time step t, the controller solves the following quadratic optimization problem [2]:

$$\min_{u_k} \sum_{k=0}^{N} \left(x_k^T Q x_k + u_k^T R u_k \right) \tag{1}$$

subject to the system constraints:

$$x_{\min} \leq x_k \leq x_{\max}, \quad u_{\min} \leq u_k \leq u_{\max}, \tag{2}$$

$$x_{k+1} = f(x_k, u_k) \tag{3}$$

where:

- $x_k \in \mathbb{R}^n$ is the system state at time step k.
- $u_k \in \mathbb{R}^m$ is the control input.
- Q and R are weighting matrices that influence the controller's performance. The matrix Q penalizes deviations of the system state from the desired trajectory, ensuring accurate tracking, while R regulates the control effort, preventing excessive variations in the control inputs.

Once the optimization is solved, only the first control action u_k^* is applied to the system. The process is repeated at the next iteration following a receding horizon approach (1).

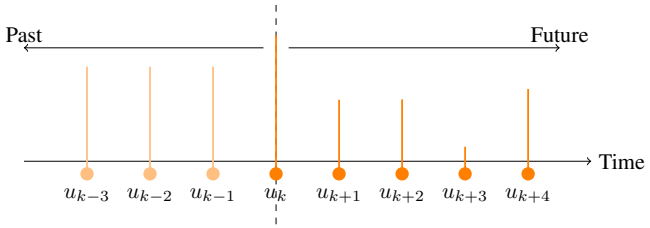


Fig. 1. Illustration of the receding horizon principle in MPC. At each time step, only the first control action u_k is applied, and the horizon shifts forward.

2) MPC Implementation and Performance in Static Environments: Model Predictive Control (MPC) is particularly effective in structured and predictable environments, where system dynamics remain constant over time. This section describes the implementation details of MPC in such conditions and evaluates its performance.

Formulation and Discretization: MPC operates on a discrete-time model of the system, which is typically obtained by discretizing continuous-time dynamics. Given a state-space representation:

$$\dot{x} = Ax + Bu \tag{4}$$

the discrete-time equivalent is:

$$x_{k+1} = A_d x_k + B_d u_k \tag{5}$$

where A_d and B_d are the discrete-time system matrices obtained via numerical integration or exact discretization methods.

At each control step, MPC solves an optimization problem over a finite prediction horizon N, as previously described in Equation (1), subject to the system constraints already stated in Equation (3).

3) Block Diagram Representation: Figure 2 presents the overall structure of an MPC controller operating in a static environment. The system consists of a **Nonlinear Model Predictive Control (NMPC)** loop, where the *desired position* serves as an input to the controller. Within the NMPC, three key components interact:

- **Cost function and constraints:** Define the optimization problem, ensuring the control objectives are met while respecting system limitations.
- **System model:** Predicts future states based on current conditions and control inputs.
- **Dynamic optimizer:** Computes the optimal control inputs by minimizing the cost function over a finite horizon.

The computed *control inputs* are applied to the **plant**, which represents the actual system being controlled. The plant's response, in the form of *system states*, is then fed back into the NMPC loop, allowing real-time adjustments to account for system dynamics and uncertainties.

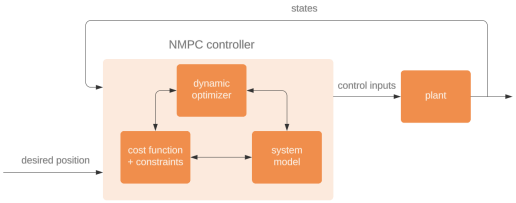


Fig. 2. Block diagram of an MPC controller in a static environment.

B. Performance Evaluation in Static Conditions

In a static environment, the main performance criteria for MPC include:

- **Tracking accuracy**: MPC ensures that the system follows the reference trajectory with minimal steady-state error.
- **Constraint satisfaction**: State and input constraints are respected at all times.
- **Computational efficiency**: Since system dynamics do not change, the optimization problem remains consistent, reducing computation overhead.

For example, in industrial process control applications [**?**], MPC effectively regulates variables such as temperature and pressure by predicting and compensating for slow system responses.

To assess the effectiveness of MPC in a structured environment, we analyze the position tracking results. Figure 3 shows the evolution of the system's position over time compared to the reference trajectory. The system successfully tracks the reference, demonstrating MPC's precision in static conditions. However, small deviations occur due to constraints and numerical optimization approximations.

To quantify the tracking performance, Table I presents the absolute errors at each time step. The errors remain

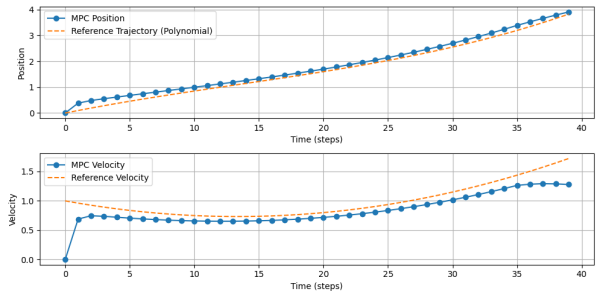


Fig. 3. MPC position tracking performance compared to the reference trajectory and speed.

Time Step	Tracking Error
0	0.00
1	0.28
2	0.29
3	0.27
4	0.25
5	0.23
6	0.21
7	0.19
8	0.17
9	0.16
10	0.15

TABLE I
TRACKING ERRORS BETWEEN MPC AND THE REFERENCE TRAJECTORY.

small, confirming the effectiveness of MPC in a predictable environment.

The tuning of MPC weights (Q and R matrices) was optimized based on knowledge of the reference trajectory and velocity profile. This approach allows the controller to achieve precise tracking in a known environment. However, in dynamic or unknown environments, such tuning is not feasible in real-time, limiting MPC's adaptability. The next section discusses these limitations and how reinforcement learning can enhance MPC's flexibility.

C. Challenges in Dynamic and Uncertain Environments

Model Predictive Control (MPC) performs well in structured environments with known reference trajectories. However, when the reference trajectory itself changes unpredictably, its effectiveness diminishes due to the following challenges:

- **Reference trajectory variation**: MPC relies on a predefined reference. If the trajectory changes unpredictably, the controller struggles to adapt, leading to poor tracking performance.
- **Fixed weight matrices**: The optimization problem is formulated using cost function weights tuned for a specific scenario. A sudden change in reference dynamics can render these weights suboptimal, causing significant deviations.
- **Computation time**: Re-solving the optimization problem at each time step becomes computationally expensive, especially when adapting to unknown changes in the reference trajectory.

To illustrate these limitations, we analyze the case where MPC is optimized to track a polynomial reference trajectory but fails when transitioning to a sinusoidal reference. Figure 4 shows the system's position over time. Initially, the controller accurately follows the polynomial path due to properly tuned weight matrices. However, as the reference shifts to a sinusoidal function, the performance degrades, leading to tracking errors and oscillatory behavior.

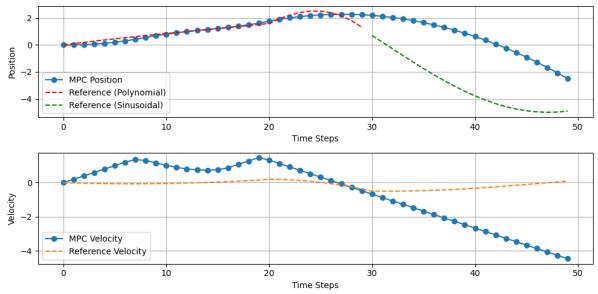


Fig. 4. MPC tracking a polynomial reference initially but failing when the trajectory changes to a sinusoidal function.

To quantify the performance drop, Table II presents the mean and maximum tracking errors before and after the trajectory change. The significant increase in error highlights the limitation of a static MPC formulation in handling varying reference trajectories.

Trajectory Type	Mean Error	Max Error
Polynomial (Static)	0.1635	0.9251
Sinusoidal (Dynamic)	3.4634	4.4499

TABLE II
TRACKING ERRORS BEFORE AND AFTER REFERENCE TRAJECTORY CHANGE.

These results demonstrate that while MPC excels in structured conditions, it struggles to adapt to changing references. To address this limitation, reinforcement learning (RL) techniques can be integrated with MPC, enabling adaptive weight tuning and real-time decision-making. The next section explores how RL can enhance MPC's robustness in uncertain environments.

III. REINFORCEMENT LEARNING FOR MPC TUNING

A. Reinforcement Learning for Adaptive Control

To overcome the limitations of static weight tuning in MPC, reinforcement learning (RL) is employed to dynamically adjust the cost function weights in response to varying system conditions. This approach enhances the controller's adaptability, allowing it to maintain high performance even when reference trajectories or system dynamics change unpredictably.

In this framework, the RL agent operates in a closed-loop with the MPC controller, continuously updating the weighting matrices Q and R via an intermediary component, the **Weights Scheduler**. This scheduler translates the RL agent's actions into real-time modifications of the cost function weights used by the MPC. The key elements of this approach include:

- **State Representation**: The RL agent receives observations from the system, provided by the **Observations and Rewards Generator**, which includes tracking error, control effort, and changes in the reference trajectory.
- **Action Space**: The agent determines adjustments to the elements of Q and R, influencing how the NMPC balances state tracking and control effort.
- **Reward Function**: The reward is designed to minimize tracking error while discouraging excessive control effort or oscillations.
- **Policy Optimization**: The RL agent, implemented as a **Deep Neural Network Policy**, learns an optimal strategy for adjusting MPC weights in real-time. The policy is updated using an RL algorithm based on received rewards.

Figure 5 illustrates this adaptive control framework, where the RL agent refines its policy iteratively to enhance the NMPC's adaptability. This scheme, referred to as **Dynamic Weights-Varying MPC (WMPC)**, enables real-time adjustments to the cost function, improving robustness against system variations.

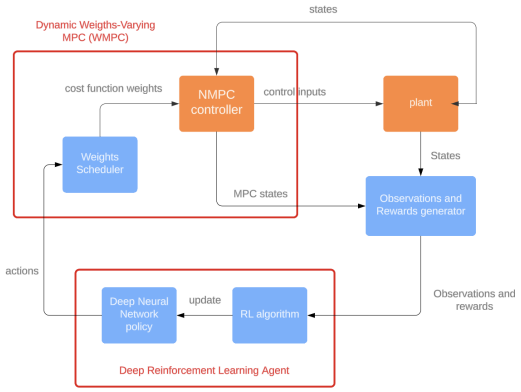


Fig. 5. Reinforcement Learning closed-loop tuning for NMPC (WMPC).

This adaptive control strategy eliminates the need for manual re-tuning of MPC weights, making it well-suited for applications where reference trajectories or operating conditions frequently change. The next section details the training methodology and evaluation metrics used to assess the RL-based MPC tuning performance.

B. Reward Function

Reinforcement learning relies on a well-designed reward function to guide the agent toward optimal behavior. In this study, we employ a *novel general multi-objective cascaded Gaussian (MOCG) reward function*, which offers several advantages over traditional approaches.

1) Motivation for Choosing MOCG: Standard reward functions in optimal control and reinforcement learning often consist of weighted sums of different objectives, such as tracking error and control effort. However, these conventional approaches have limitations:

- A **linear combination** of objectives can make weight tuning difficult and unintuitive.
- **Fixed weighting** does not allow dynamic adaptation to different phases of control.
- **Ignoring correlations** between objectives can lead to suboptimal policies, as interactions between tracking accuracy, control effort, and smoothness are often overlooked.

The *MOCG* function addresses these issues by modeling each criterion using **cascaded multivariate Gaussian distributions**, explicitly capturing the dependencies between key performance metrics. Previous studies, such as [3], have demonstrated the effectiveness of multi-objective reward shaping in reinforcement learning, allowing policies to balance competing objectives more efficiently.

2) Functioning Principle: Instead of treating each objective independently, the MOCG reward function considers the correlation between them. This allows the reward to reflect how variations in one metric (e.g., reducing control effort) influence others (e.g., tracking error). The reward function is defined as:

$$r(x,u) = \alpha \exp\left(-\frac{1}{2}(F(x,u)-\mu)^T \Sigma^{-1}(F(x,u)-\mu)\right) \tag{6}$$

where:

- $F(x,u) = [f_1(x,u), f_2(x,u), ..., f_n(x,u)]^T$ represents the vector of performance metrics (e.g., tracking error, control effort, control smoothness).
- μ is the vector of desired values for these metrics (typically 0 for minimization).
- Σ is the covariance matrix that encodes the interdependence between objectives.
- α is a normalization factor ensuring a well-scaled reward signal.

The covariance matrix Σ is defined as:

$$\Sigma = \begin{bmatrix} \sigma^2_{\text{tracking}} & \rho_{12}\sigma_{\text{tracking}}\sigma_{\text{effort}} & \rho_{13}\sigma_{\text{tracking}}\sigma_{\text{variation}} \\ \rho_{12}\sigma_{\text{tracking}}\sigma_{\text{effort}} & \sigma^2_{\text{effort}} & \rho_{23}\sigma_{\text{effort}}\sigma_{\text{variation}} \\ \rho_{13}\sigma_{\text{tracking}}\sigma_{\text{variation}} & \rho_{23}\sigma_{\text{effort}}\sigma_{\text{variation}} & \sigma^2_{\text{variation}} \end{bmatrix} \tag{7}$$

where:

- $\sigma_{\text{tracking}}, \sigma_{\text{effort}}, \sigma_{\text{variation}}$ are the standard deviations of the tracking error, control effort, and control smoothness.
- $\rho_{12}, \rho_{13}, \rho_{23}$ are the correlation coefficients between these objectives, reflecting how strongly they influence each other.

This formulation provides several benefits:

- **Correlation-aware optimization**: The covariance matrix Σ enables the agent to understand how improving one metric may negatively or positively affect others.
- **Context-dependent weighting**: Instead of using fixed coefficients, the reward dynamically adjusts based on the system's state and the interplay between objectives.

- **Smooth and well-defined gradients**: The Gaussian formulation ensures that the reward landscape remains differentiable, facilitating stable learning.

3) Impact on Learning: By incorporating correlations between performance metrics, the RL agent can learn more efficient trade-offs, leading to improved robustness and adaptability. This eliminates the need for manual fine-tuning of weight coefficients and allows for a more natural adaptation of the control strategy over time.

C. Practical Implementation of RL-Enhanced MPC

The implementation of RL-enhanced MPC involves training an agent to optimize the controller's performance under diverse conditions. This requires defining a structured training process, including environment simulation, policy learning, and evaluation.

The training phase relies on episodic learning, where the RL agent explores different weight configurations and refines its policy through trial and error. Once trained, the agent is deployed in real-time, continuously adjusting Q and R based on system feedback. Performance evaluation is conducted by analyzing tracking accuracy, control effort, and robustness to trajectory changes.

The policy is learned using the Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm, which improves upon standard Deep Deterministic Policy Gradient (DDPG) by reducing overestimation bias and stabilizing training. TD3 is well-suited for this problem as it enables smooth and stable updates of the MPC weights, preventing sudden changes that could degrade control performance.

In deployment, the MPC controller runs at a high frequency with a sampling time of $T_s = 0.04s$, ensuring precise trajectory tracking. Meanwhile, the RL agent updates the cost function weights every $T_{sw} = 10s$, allowing it to adapt gradually without excessive computational overhead.

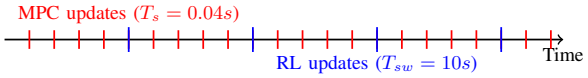


Fig. 6. Timing structure: MPC updates at T_s, RL updates at T_{sw}.

This hierarchical structure ensures that MPC operates with real-time precision, while RL continuously refines its policy based on long-term performance trends.

D. Experimental Evaluation

To evaluate the performance of the RL-MPC approach, we tested the controller on a trajectory tracking task where the reference trajectory was designed to include a transition between two different motion patterns. Initially, the reference follows a polynomial function, representing a smooth and predictable motion. At a predefined time step, the reference gradually shifts towards a sinusoidal trajectory, introducing a sudden but continuous change in motion characteristics. This transition is implemented over a finite time window to ensure a smooth interpolation between the two profiles.

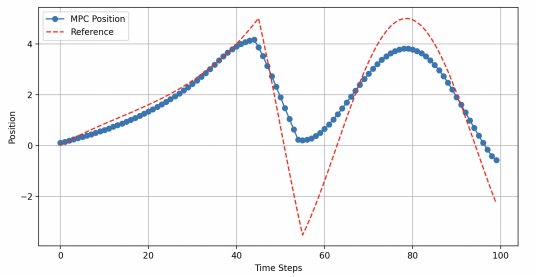


Fig. 7. Trajectory tracking performance of RL-MPC. The blue curve with markers represents the system's actual trajectory under MPC control, while the red dashed curve shows the desired reference trajectory.

As shown in Figure 7, the RL-MPC effectively tracks the reference trajectory throughout the transition. The system initially follows the polynomial trajectory with minimal error. When the reference switches to a sinusoidal profile, the controller adapts and maintains a consistent tracking performance. The smooth transition observed in the controlled trajectory indicates that the RL-adjusted cost function enables the MPC to handle dynamic changes in the reference without excessive overshoot or instability.

These results highlight the adaptability of the RL-MPC approach in scenarios where reference trajectories are subject to dynamic variations, making it suitable for applications requiring real-time responsiveness.

IV. Conclusion

In this study, we proposed an RL-enhanced MPC framework that dynamically adjusts the cost function weights to improve adaptability in trajectory tracking tasks. By integrating a Multi-Objective Cascaded Gaussian (MOCG) reward function, the RL agent learns to optimize tracking accuracy, control effort, and smoothness while considering their interdependencies. The experimental results demonstrated that the proposed approach enables the MPC to effectively handle dynamic changes in reference trajectories, maintaining precise tracking without excessive overshoot or instability.

Compared to traditional fixed-weight MPC, our method provides a more flexible and data-driven adaptation mechanism, reducing the need for manual tuning and improving robustness across varying conditions. The structured update mechanism ensures that the MPC operates in real time while the RL agent continuously refines its policy based on long-term performance trends.

Future work could explore extensions such as incorporating model uncertainty into the learning process, applying the framework to higher-dimensional control problems, or investigating alternative learning architectures for further performance gains. These enhancements could further broaden the applicability of RL-enhanced MPC in complex and dynamic environments.

References

[1] X. Zhang, H. Huang, F. Borrelli, and M. Tomizuka, "Weights-Varying MPC for Autonomous Vehicle Guidance: A Deep Re-

inforcement Learning Approach," *IEEE Transactions on Intelligent Vehicles*, vol. 7, no. 4, pp. 913–924, Dec. 2021. doi: https://doi.org/10.1109/TIV.2021.965504210.1109/TIV.2021.9655042.

[2] J. Maciejowski, *Predictive Control with Constraints*. Prentice Hall, 2002.

[3] T. Brys, A. Harutyunyan, M. Taylor, and A. Nowé, "Multi-Objective Reinforcement Learning Using Reward Shaping," in *IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 282-289, 2014. doi: https://doi.org/10.1109/ICTAI.2014.4710.1109/ICTAI.2014.47.

Comparison of RANSAC and Least Squares Methods for Parameter Estimation in Autonomous Sailboat Dynamics

Main Tihami Ouazzani*

ENSTA, Autonomous Robotics FISE25, France

Abstract

This study compares the performance of the Random Sample Consensus (RANSAC) and classical Least Squares (LS) methods in estimating parameters of the dynamic model for the autonomous sailboat Brave. Real-world sailing conditions often involve substantial noise and outliers, severely impacting traditional parameter estimation techniques. Using experimental data from Guerlédan Lake, we highlight the robustness of RANSAC compared to LS. Results confirm that RANSAC significantly enhances parameter reliability and model fidelity, aligning with recent findings by Djurović (2024).

Keywords: Parameter estimation, RANSAC, Least Squares, Autonomous sailboat, Dynamic modeling, Robust estimation

1. Introduction

Accurate parameter estimation is essential for simulating and controlling autonomous maritime vehicles. Classical methods like Least Squares (LS) are vulnerable to noise and outliers. In contrast, RANSAC has proven robust, effectively managing data inconsistencies [1]. This study compares LS and RANSAC methods using experimental data from the sailboat Brave [2].

*Corresponding author

Email address: Main.tihami@ensta.fr (Main Tihami Ouazzani)

2. Methods

2.1. Dynamic Model

The dynamic equations of Brave are given by:

$$\begin{aligned}
\dot{x} &= v\cos\theta + p_1 a\cos\psi,\\
\dot{y} &= v\sin\theta + p_1 a\sin\psi,\\
\dot{\theta} &= \omega,\\
\dot{v} &= f_s\sin\delta_s - f_r\sin u_1 - p_2 v^2\omega v p_9,\\
\dot{\omega} &= \frac{f_s(p_6 - p_7\cos\delta_s) - p_8 f_r\cos u_1 - p_3\omega v}{p_{10}},\\
f_s &= p_4\|w_{ap}\|^2\sin(\delta_s - \psi_{ap}),\\
f_r &= p_5 v\sin u_1.
\end{aligned} \tag{1}$$

Parameters p_6, p_7, p_8, p_9 are measured experimentally:

- $p_6 = 0.65$ m, $p_7 = 0.06$ m, $p_8 = 0.296$ m, $p_9 = 21.7$ kg.

2.2. Parameter Estimation

The LS method minimizes squared residuals:

$$p = (A^T A)^{-1} A^T Y. \tag{2}$$

RANSAC robustly identifies parameters by selecting inliers:

$$p_{opt} = \arg\max_p |\{i : |y_i - x_i^T p| < t\}|. \tag{3}$$

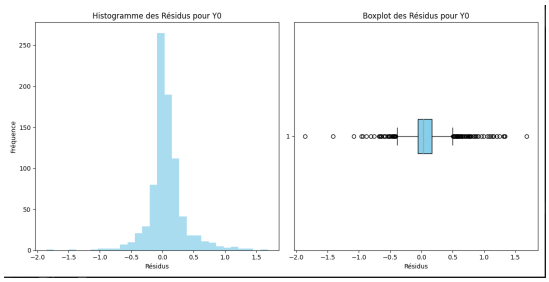
3. Results

Initial LS parameters exhibited sensitivity to outliers:

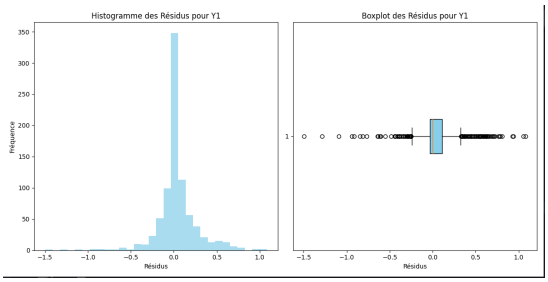
$$p_1 = 0.0147, p_2 = 1.4120, p_3 = -0.0608, p_4 = -0.2426, p_5 = -0.2019, p_{10} = 0.1405.$$

RANSAC produced significantly improved estimates:

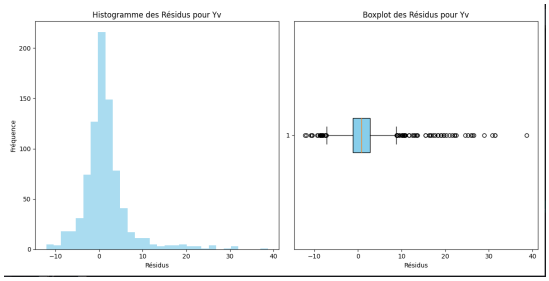
$$p_1 = 0.0887, p_2 = 0.3847, p_3 = -0.00756, p_4 = -0.0937, p_5 = 0.00328, p_{10} = 0.0.$$



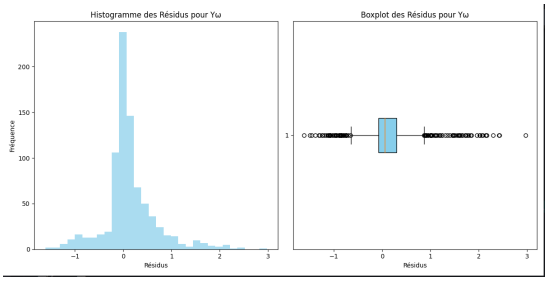
(a) Residual distribution for $\mathbf{Y}_0 = [v_x - v\cos(\theta)]$



(b) Residual distribution for $\mathbf{Y}_1 = [v_y - v\sin(\theta)]$

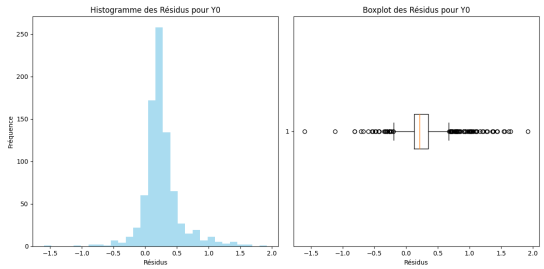


(c) Residual distribution for $\mathbf{Y}_v = [p_9\alpha]$

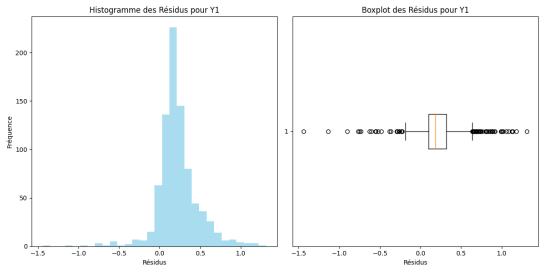


(d) Residual distribution for $\mathbf{Y}_\omega = [0]$

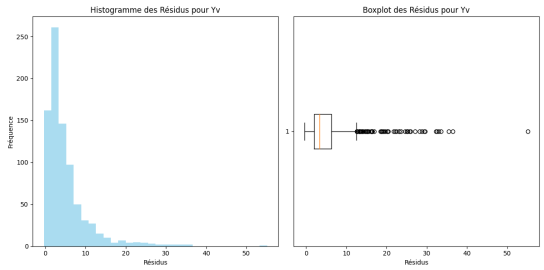
Figure 1: Residual distribution - Least Squares method.



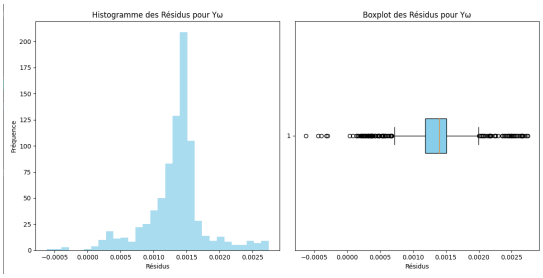
(a) Residual distribution for $\mathbf{Y}_0 = [v_x - v\cos(\theta)]$



(b) Residual distribution for $\mathbf{Y}_1 = [v_y - v\sin(\theta)]$



(c) Residual distribution for $\mathbf{Y}_v = [p_9\alpha]$



(d) Residual distribution for $\mathbf{Y}_\omega = [0]$

Figure 2: Residual distribution - RANSAC

4. Discussion

Residual analysis illustrates RANSAC's superior robustness to outliers. The residual distribution from the LS method showed significant variability and multiple pronounced outliers, indicating that LS was highly sensitive to noise and anomalies present in the experimental data.

In contrast, RANSAC demonstrated notable improvements. The residuals became more symmetrically centered around zero, and the spread significantly reduced, particularly for parameters influencing Y_v and Y_ω. For instance, the residual interval for Y_ω decreased markedly from [-2, 3] with LS to approximately [-0.0005, 0.0025] using RANSAC. Similar improvements, albeit slightly less pronounced, were observed for Y_v, where the tails of the residual distribution were substantially controlled. These results validate RANSAC's enhanced robustness, effectively minimizing the detrimental impact of outliers on parameter estimates. This aligns with Djurović's observations of RANSAC's efficacy under noisy conditions [1].

5. Conclusion

RANSAC substantially improves parameter estimation robustness compared to LS, making it suitable for practical maritime applications involving autonomous sailboats.

Acknowledgments

Thanks to Loïck Degorre, Gabriel Betton and Fabrice Le Bars for their support.

References

[1] I. Djurović, *Random sample consensus algorithm for hyperbolic frequency modulated signals parameters estimation*, Signal Processing, 2024.

[2] V. Adrian, P. Ylona, R. harendra, T. Ouazzani, T. Luc-André, et al., *Commande par jumeau numérique de voilier – Rapport de projet*, ENSTA Bretagne, 2025.

Evaluating Path Planning Algorithms: A Comparison of Dynamic Window and Potential Field Approaches

Terrine Luc-André

March 11, 2025

Contents

1 Introduction

Path planning is a fundamental function in the autonomous navigation of mobile robots. In the context of obstacle avoidance, it plays a crucial role in ensuring efficient and safe movement. Various path planning algorithms exist, each with its own advantages and limitations depending on the application. Among them, the *Potential Field Planning* and the *Dynamic Window Approach (DWA)* are two real-time techniques based on local obstacle avoidance.

In this study, we will analyze the working principles of both algorithms and evaluate their performance in different scenarios. By comparing their behavior, we aim to highlight their strengths and weaknesses in various navigation contexts.

2 Dynamic Window Approach

The Dynamic Window Approach (DWA) is a velocity-space-based local reactive obstacle avoidance technique. Instead of planning an entire trajectory, DWA searches for feasible control commands directly in the velocity space, ensuring real-time adaptability to dynamic environments.

2.1 Principle of Dynamic Window

A robot's trajectory can be described as a sequence of circular and straight-line arcs. The DWA reduces the search space by considering the kinematic and dynamic constraints of the robot. The set of possible velocities, known as the *dynamic window* V_d, is defined as:

$$V_d = \{(v, \omega) \mid v \in [v_c - \dot{v}_b \Delta t, v_c + \dot{v}_a \Delta t], \quad \omega \in [\omega_c - \dot{\omega}_b \Delta t, \omega_c + \dot{\omega}_a \Delta t]\} \tag{1}$$

where v_c and ω_c are the current translational and rotational velocities, and $\dot{v}_a$, $\dot{\omega}_a$, $\dot{v}_b$, and $\dot{\omega}_b$ represent the maximum accelerations and decelerations allowed by the robot's actuators.

2.2 Admissible Velocities and Obstacle Avoidance

A velocity tuple (v, ω) from V_d is considered *admissible* if the robot can stop before colliding with an obstacle. The admissible velocity set V_a is given by:

$$V_a = \{(v, \omega) \mid v, \omega \leq \sqrt{2\rho_{\min}(v, \omega)\dot{v}_b}, \quad \sqrt{2\rho_{\min}(v, \omega)\dot{\omega}_b}\} \tag{2}$$

where $\rho_{\min}(v, \omega)$ represents the distance to the closest obstacle on the corresponding trajectory.

2.3 Objective Function and Trajectory Selection

DWA selects the optimal velocity command by maximizing an objective function $\Gamma(v, \omega)$ that balances obstacle clearance and path alignment:

$$\Gamma(v, \omega) = \lambda \vartheta_{\text{clear}} + (1 - \lambda)\vartheta_{\text{path}} \tag{3}$$

where: - ϑ_{clear} measures the clearance from obstacles. - ϑ_{path} ensures alignment with the planned path. - λ is a weighting factor.

The search for the best velocity is performed iteratively in a reduced two-dimensional search space, making the method computationally efficient for real-time applications.

2.4 Limitations

While DWA is effective for dynamic obstacle avoidance, it can suffer from local minima issues where the robot gets stuck in an oscillatory motion. Integrating global path planning techniques, such as the Focused D* (FD*) algorithm, can help mitigate these problems by providing a broader navigation strategy.

3 Potential Field Approach

The Potential Field Approach (PFA) is a real-time obstacle avoidance technique that models the environment using an artificial force field. This method, originally proposed by Khatib [**?**], treats the goal as an attractive force while obstacles exert repulsive forces, guiding the robot through safe paths without requiring precomputed trajectories.

3.1 Principle of Potential Fields

The core idea behind PFA is to define a potential function $U(x)$ where the robot is influenced by attractive and repulsive forces:

$$U(x) = U_a(x) + U_r(x) \tag{4}$$

where: - $U_a(x)$ is the attractive potential pulling the robot toward the goal. - $U_r(x)$ is the repulsive potential pushing the robot away from obstacles.

The attractive potential is typically defined as:

$$U_a(x) = \frac{1}{2} k_p \|x - x_d\|^2 \tag{5}$$

where k_p is a positive gain and x_d is the goal position. The corresponding attractive force is:

$$F_a(x) = -\nabla U_a(x) = -k_p(x - x_d) \tag{6}$$

For obstacles, a repulsive potential is introduced to prevent collisions:

$$U_r(x) = \begin{cases} \frac{1}{2} k_r \left(\frac{1}{d(x)} - \frac{1}{d_0} \right)^2, & d(x) \leq d_0 \\ 0, & d(x) > d_0 \end{cases} \tag{7}$$

where $d(x)$ is the distance to the nearest obstacle, d_0 is the influence radius, and k_r is a scaling factor. The repulsive force is then:

$$F_r(x) = -\nabla U_r(x) = k_r \left(\frac{1}{d^2(x)} - \frac{1}{d_0^2} \right) \frac{\partial d(x)}{\partial x} \tag{8}$$

3.2 Path Generation and Motion Control

The robot moves by following the resultant force:

$$F(x) = F_a(x) + F_r(x) \tag{9}$$

which determines the robot's velocity and direction. This allows real-time, continuous adaptation to the environment without requiring explicit path planning.

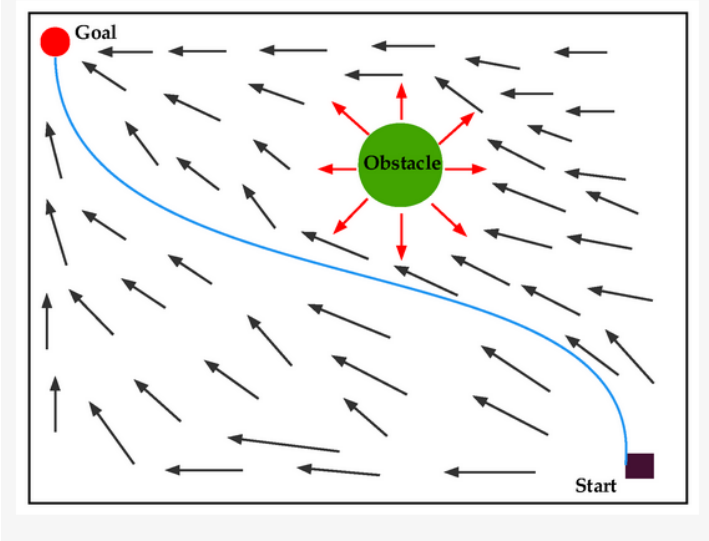


Figure 1: Schematic diagram of the path planning algorithm based on APF.

3.3 Limitations and Challenges

Despite its simplicity and real-time applicability, the Potential Field Approach has several known limitations:

- **Local Minima**: The robot can get trapped in equilibrium points where attractive and repulsive forces cancel out.
- **Oscillations**: In narrow corridors, the repulsive forces can cause unstable behavior, making navigation inefficient.
- **No Consideration of Dynamics**: The approach does not inherently consider the robot's kinematic or dynamic constraints.

Various modifications, such as introducing random perturbations or integrating global path planning methods, have been proposed to mitigate these issues.

4 Comparison of Methods

To evaluate the performance of the algorithms, we conducted tests in different scenarios and compared them based on the following criteria:

- **Mission success**: Whether the algorithm successfully reaches the goal.
- **Path length**: The total distance traveled.
- **Computational time**: The execution time required to compute the trajectory.

4.1 Point-to-Point Navigation Without Obstacles

- **Results:** Both algorithms successfully reach the goal with a similar trajectory.
- **Comparison:** The path length is nearly identical, but the *Dynamic Window Approach (DWA)* requires a longer computation time due to its velocity-space search process.

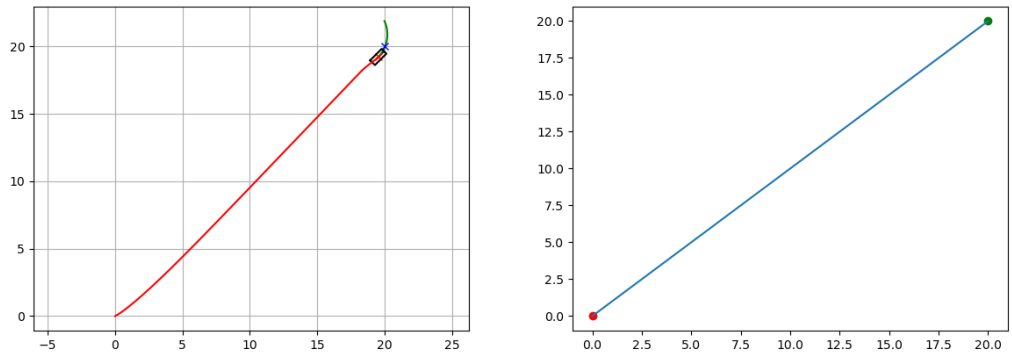


Figure 2: Comparison of DWA (left) and Potential Field (right) without obstacles.

4.2 Navigation With a Single Obstacle

- **Results:** Both algorithms successfully navigate around the obstacle.
- **Comparison:** The path lengths remain similar, but DWA exhibits a higher computational cost.

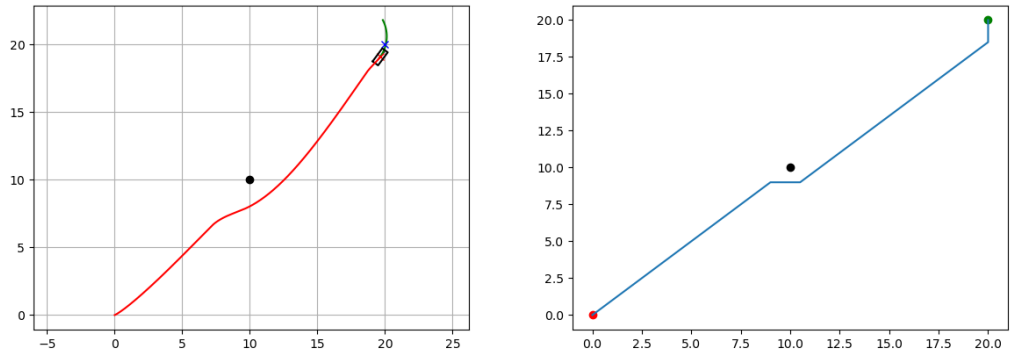


Figure 3: Comparison of DWA (left) and Potential Field (right) with a single obstacle.

4.3 Navigation With Multiple Obstacles

- **Results:**
  - The *Potential Field* method produces a **shorter path** and requires **less computation time**.
  - The *DWA* path is longer due to the constraints of robot dynamics, which prevent it from taking sharp turns.

4.4 Navigation in a Corridor

- **Results:** Both methods successfully reach the goal with similar paths.
- **Comparison:** The path lengths are comparable, but DWA still requires a longer computational time.

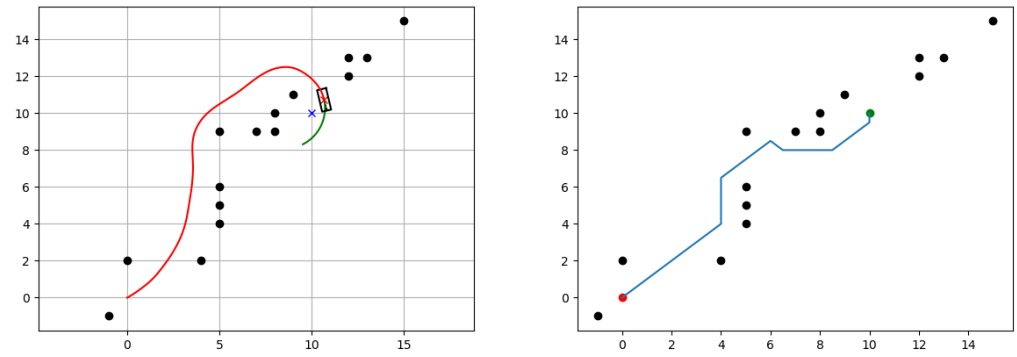


Figure 4: Comparison of DWA (left) and Potential Field (right) in a multi-obstacle environment.

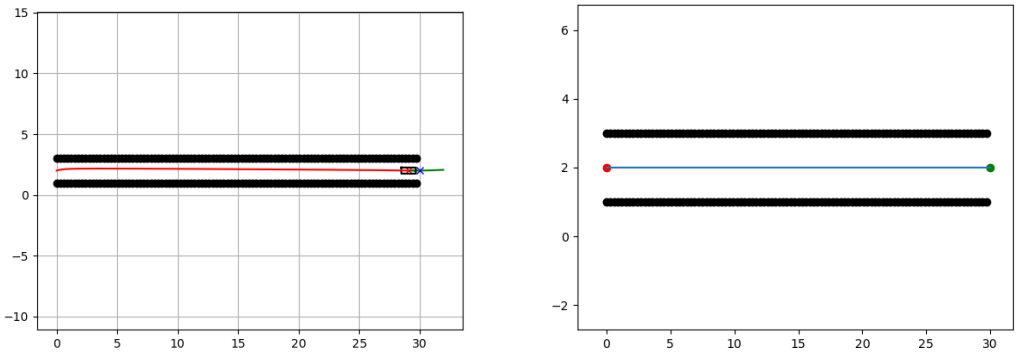


Figure 5: Comparison of DWA (left) and Potential Field (right) in a corridor.

4.5 Navigation in a Maze

- **Results:**
  - The *Potential Field* method **fails** due to oscillations, trapping the robot in local minima.
  - The *DWA* **successfully completes** the task but requires a **significantly longer computation time**.

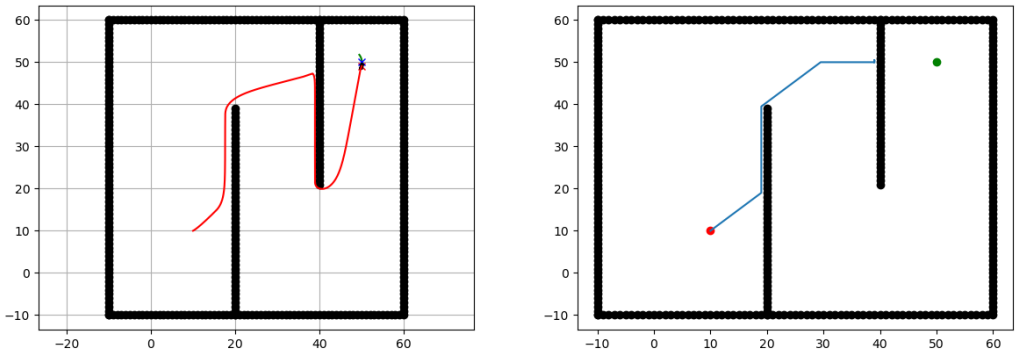


Figure 6: Navigation in a maze: DWA (left) succeeds, while Potential Field (right) fails due to oscillations.

5 Overall Performance Comparison

With the individual test results analyzed, we can now compare the overall efficiency of the two algorithms.

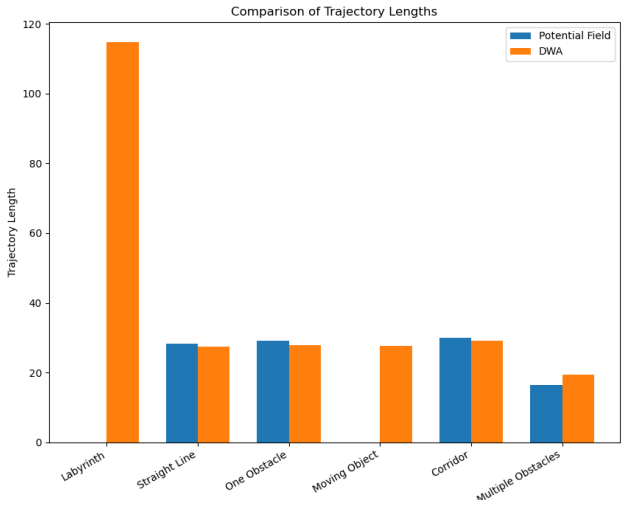


Figure 7: Comparison of path lengths for different scenarios.

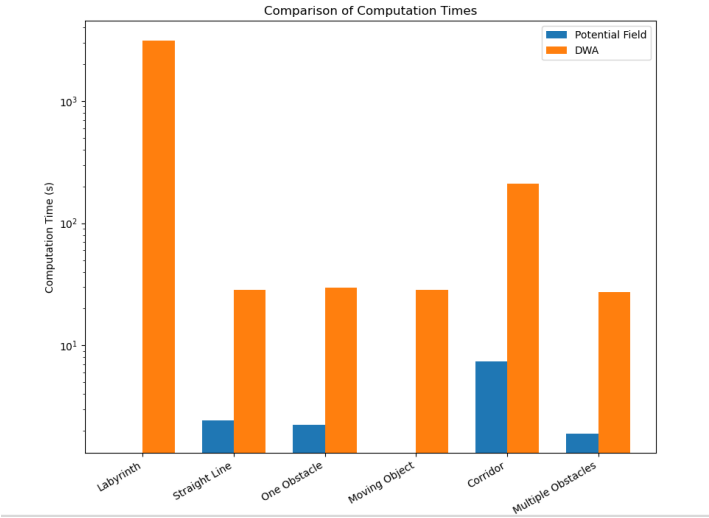


Figure 8: Comparison of computation times for different scenarios.

In general, the difference in path length between the two methods across various test scenarios remains relatively small. Both Dynamic Window Approach (DWA) and Potential Field (PF) generate paths of comparable efficiency in terms of total distance traveled.

However, a significant difference is observed in terms of computational cost. DWA requires substantially more computation time than PF, primarily due to its velocity-space search and real-time trajectory evaluation.

Despite this higher computational cost, DWA offers key advantages:

- It accounts for dynamic models, making it more suitable for environments with moving obstacles.
- It has a much lower risk of getting trapped in local minima, a major issue observed with PF particularly in the maze scenario.

Thus, while PF is computationally more efficient, DWA provides a more robust and reliable navigation strategy, especially in complex and dynamic environments.

6 Conclusion

In this study, we have analyzed and compared two real-time path planning algorithms: the *Dynamic Window Approach (DWA)* and the *Potential Field (PF)* method. Through various experimental scenarios, we evaluated their efficiency based on mission success, path length, and computational cost.

The results show that both methods are capable of successfully navigating in simple environments, with comparable path lengths. However, **DWA consistently requires more computational time** due to its real-time velocity-space search and trajectory evaluation.

Despite this higher computational cost, **DWA demonstrates greater robustness in complex and dynamic environments**, particularly in scenarios involving moving obstacles or labyrinth-like structures. Unlike **PF, which struggles with local minima and oscillatory behavior**, DWA maintains a more stable and adaptable navigation strategy.

On the other hand, **PF proves to be computationally more efficient**, making it a suitable choice for applications where rapid response time is crucial, and the environment is relatively static.

Overall, the choice between these two methods depends on the specific requirements of the application. If computational efficiency is the priority, **PF is a viable option**. However, for more **robust and dynamic navigation**, especially in unpredictable environments, **DWA remains the superior choice** despite its computational cost.

Future work could explore hybrid approaches that combine the **efficiency of PF** with the **robustness of DWA**, potentially leading to an optimized path planning strategy for real-time robotic navigation.

References

[1] Zhao, Y., Ma, T., Liang, X. (2020). The application of drones in precision agriculture: A review. *Drones, 7*(3), 211. Retrieved from https://www.mdpi.com/2504-446X/7/3/211

[2] Borenstein, J., Herve, M., Duffy, M. (1986). The evaluation of robot arm performance. *Journal of Robotic Systems, 5*(1), 95-107. 10.1177/027836498600500106. Retrieved from https://journals.sagepub.com/doi/epdf/10.1177/027836498600500106

[3] De Souza, G. (2007). A new method for solving the inverse kinematics of robot arms using Jacobian matrices. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2007)*, 1765-1770. Retrieved from http://vigir.missouri.edu/~gdesouza/Research/Conference_CDs/IEEE_ICRA_2007/data/papers/1765.pdf

State of the Art on DeepBlueAI in the Underwater World

Arthur Coron

March 10, 2025

Abstract

This state of the art explores DeepBlueAI, an artificial intelligence specialized in underwater applications. It covers its context, technical capabilities, impacts on marine research, as well as its limitations and future perspectives.

1 Introduction

Ocean exploration represents one of the most fascinating and complex frontiers of contemporary scientific research. Covering more than 70 percent of the Earth's surface, these marine areas remain largely unexplored as a result of the technical and environmental challenges they pose.

In this context, DeepBlueAI emerges as a major innovation that combines artificial intelligence (AI) and underwater technologies to push the current boundaries of oceanographic research. By integrating new capabilities in data processing, image recognition, and environmental prediction, DeepBlueAI aims to transform the understanding of marine ecosystems while offering concrete solutions to problems such as biodiversity preservation and adaptation to climate change.

This state of the art explores the foundations of this technology, its applications, and the perspectives it opens in the field of underwater research.

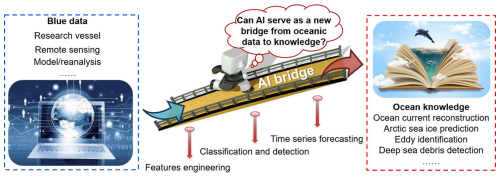


Figure 1: Representation of blue data.

2 Context and Objectives of DeepBlueAI

2.1 Presentation of DeepBlueAI

DeepBlueAI (DBAI) was created to bridge the gap between complex ocean data and actionable scientific knowledge. Blue data, which is all ocean-related datasets collected from satellites, underwater sensors, and autonomous vehicles, represent the foundation for advancing oceanography and are used as a foundation for DBAI.
DBAI leverages AI frameworks, such as deep learning, to extract meaningful patterns and insights from datasets that include satellite imagery, buoy readings, and in-situ observations. These tools are crucial in improving our ability to monitor marine environments, predict phenomena, and support conservation efforts.

2.2 Importance of Marine Technologies

Artificial Intelligence (AI) has revolutionized many fields, and this include marine exploration. For decades, the vast and often inaccessible underwater world has presented numerous challenges to researchers. The oceans, covering over 70 percent of the Earth's surface, remain one of the least explored frontiers, primarily due to their depth, remoteness, and extreme conditions.

AI technologies have the potential to address these challenges by enabling automated, efficient, and precise data collection and analysis, thereby significantly advancing understanding of marine environments. The ability of AI to process and analyze the enormous volumes of data generated by underwater sensors, cameras, and satellites makes it an invaluable tool. For instance, AI-powered systems are used to monitor biodiversity changes or detect pollutants, helping scientists respond to environmental threats.

3 DBAI Integrated Technological Advances

3.1 Applications of Underwater AI

DeepBlueAI has many applications, from environmental monitoring to underwater robotics. It uses real-time analysis and long-term assessment of marine ecosystems through the integration of machine learning algorithms and deep learning models.

One of its applications is in the assessment of coral reef health. By analyzing spectral data from satellite images and underwater photos, DeepBlueAI can classify coral conditions, detect early signs of bleaching, and monitor changes in biodiversity over time. These capabilities allow for improved management of marine protected areas.

3.2 Technologies Utilized

DeepBlueAI integrates recent technologies to enhance underwater research capabilities:

- **Deep Learning Frameworks:** It uses convolutional neural networks (CNNs) and recurrent neural networks (RNNs) that play an key role in image processing and acoustic signal interpretation. CNNs are particularly effective in analyzing seabed images to detect geological formations or human-made objects, while RNNs assist in forecasting of oceanographic variables such as sea surface temperature and wave height.

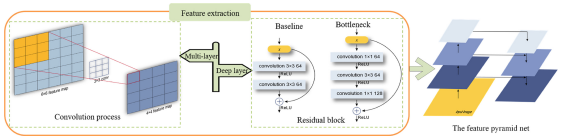


Figure 2: Illustration of convolution processes in feature extraction.

- **Autonomous Underwater Vehicles (AUVs):** Equipped with sonar systems, LiDAR, and cameras, AUVs autonomously navigate and collect spatial and environmental data from previously inaccessible areas.
- **Cloud Computing:** The important volume of oceanographic data requires advanced cloud infrastructure for real-time processing and storage. Computing clusters allow for distributed deep learning training, making it possible to develop AI models capable of identifying different patterns in marine ecosystems.
- **Feature Extraction Techniques:** Advanced feature extraction methods, such as wavelet transforms and principal component analysis (PCA), enable the identification of subtle variations in marine datasets.

3.3 Preprocessing and Data Management

DeepBlueAI success is highly based on effective preprocessing techniques.

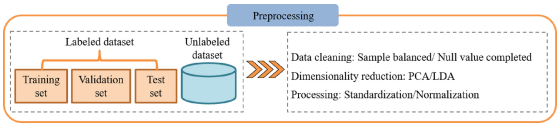


Figure 3: Steps involved in preprocessing labeled and unlabeled datasets.

- **Data Cleaning:** Oceanographic datasets are often incomplete due to sensor malfunctions, transmission errors, or environmental interferences. DeepBlueAI employs imputation methods such as k-nearest neighbors (KNN) and expectation-maximization (EM) algorithms to reconstruct missing data points. This ensures that the datasets remain statistically representative and unbiased.
- **Dimensionality Reduction:** Large marine datasets often contain redundant or highly correlated features, which can introduce noise into machine learning models. Techniques such as Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA) help reduce computational complexity while preserving information and improving classification accuracy.
- **Standardization and Normalization:** Since underwater sensor readings can vary depending on environmental conditions, DeepBlueAI applies normalization techniques such as min-max scaling and Z-score normalization to bring disparate data sources into a common range. This ensures that no single feature influences AI predictions too much, leading to more robust and interpretable model outputs.

4 Current Challenges and Limitations

Despite the potential of this AI, DeepBlueAI still faces challenges that require attention to ensure its effective deployment in underwater research and exploration.

4.1 Technical Constraints

One of the primary technical issues lies in managing and processing massive volumes of data generated from various sources, including satellites, buoys, and AUVs. These datasets are not only immense in size but also diverse in structure. The complexity of integrating and analyzing such data, especially in environments with limited connectivity, often pushes existing frameworks to their limits. Another limitation is the resource-intensive nature of underwater exploration systems. Underwater AI-driven platforms, such as AUVs, rely on finite power supplies, including batteries that are subject to rapid depletion. This limits their operational range and the duration of their missions, particularly in deep-sea environments where recharging or retrieving equipment is challenging. Furthermore, the performance of sensors, cameras, and communication systems in such harsh conditions often deteriorates as a result of variations in pressure, salinity, and temperature.

4.2 Environmental Issues

From an environmental perspective, the footprint of marine exploration activities is a growing concern. Although technologies like DeepBlueAI tend to support sustainable practices, the deployment of equipment in fragile marine ecosystems can disrupt habitats, interfere with marine life, or introduce pollutants. The balance between exploration and conservation remains a big challenge, necessitating the development of ecofriendly materials, quieter operational systems, and minimally invasive exploration methods. Furthermore, significant data gaps persist in remote and extreme marine regions, such as the deep ocean or polar environments, which remain largely inaccessible due to logistical and technological limitations. These regions are crucial for understanding global environmental phenomena, but the lack of comprehensive data impedes the development of accurate models and predictions.

5 Future Perspectives and Conclusion

DeepBlueAI represents a change in our ability to understand and manage the world's oceans. Looking ahead, its potential applications promise to modify marine research, conservation, and development in many ways.

5.1 Potential of DeepBlueAI

One of the most promising aspects of DeepBlueAI is its capacity to support sustainable ocean management through data-driven insights. By analyzing vast datasets, the platform can identify patterns and trends related to marine biodiversity or fishery dynamics. These insights can inform conservation strategies, ensuring that limited resources are directed where they are most needed.

Then, DeepBlueAI improves our ability to respond to climate-related challenges in marine ecosystems. Rising sea temperatures, ocean acidification, and deoxygenation are altering ecosystems at unprecedented rates, and by integrating real-time data with predictive models, DeepBlueAI can forecast the impacts of climate change on marine life, identify areas most at risk, and propose mitigation strategies.

5.2 General Conclusion

In conclusion, DeepBlueAI clearly shows the diversity role of artificial intelligence in addressing some of the most important challenges in marine research and conservation. Using the power of AI to process and analyze blue data, the platform not only expands our understanding of ocean systems, but also provides useful insights to support global sustainability goals. However, realizing the full potential of DeepBlueAI requires concerted efforts from a diverse range of fields. Collaborative initiatives between AI developers, oceanographers, and environmental organizations are essential to overcome many technical and environmental challenges. As the world increasingly turns to the oceans for solutions to challenges such as climate change and loss of biodiversity, the importance of tools such as DeepBlueAI will only continue to grow.

6 references

References

[1] Chen, G., Huang, B., Chen, X., Ge, L., Radenkovic, M., & Ma, Y. (2022). Deep blue AI: A new bridge from data to knowledge for the ocean science. *Deep-Sea Research Part I, 190*, 103886. https://doi.org/10.1016/j.dsr.2022.103886

[2] Amores, A., Jordà, G., Arsouze, T., & Le Sommer, J. (2018). Up to what extent can we characterize ocean eddies using present-day gridded altimetric products? *Journal of Geophysical Research: Oceans, 123*(10), 7220-7236. https://doi.org/10.1029/2018JC014140

[3] Chelton, D. B., Gaube, P., Schlax, M. G., Early, J. J., & Samelson, R. M. (2011). The influence of nonlinear mesoscale eddies on near-surface oceanic chlorophyll. *Science, 334*(6054), 328-332. https://doi.org/10.1126/science.1208897

[4] Chen, X., Chen, G., & Huang, B. (2021). Independent eddy identification with profiling Argo as calibrated by altimetry. *Journal of Geophysical Research: Oceans, 126*(1). https://doi.org/10.1029/2020JC016729

[5] Andersson, T. R., Hosking, J. S., Perez-Ortiz, M., Paige, B., Elliott, A., Russell, C., et al. (2021). Seasonal Arctic sea ice forecasting with probabilistic deep learning. *Nature Communications, 12*(1), 5124. https://doi.org/10.1038/s41467-021-25257-4

[6] Brett, A., Leape, J., & Abbott, M. (2020). Ocean data need a sea change to help navigate the warming world. *Nature, 582*(7811), 181-183. https://doi.org/10.1038/d41586-020-01668-z

Combining Interactive Learning and Deep Reinforcement Learning for Discrete Action Spaces, Comparison Between Two Models for Reinforcement Learning

Romain Bornier

March 2025

Abstract

This paper explores how interactive learning can enhance AI models and their training within a deep reinforcement learning framework. We introduce an interactive AI model based on action guidance, where a human trainer provides suggested actions during the training phase based on the agent's real-time state. The trainer indirectly influences reward assignment by comparing the suggested action with the one chosen by the control policy. This approach aims to reduce the exploratory phase by leveraging human guidance to accelerate learning. Once training is complete, the agent operates autonomously without human intervention.

We evaluate this approach using two deep reinforcement learning algorithms—Proximal Policy Optimization (PPO) and Deep Q-Network (DQN)—in a discrete action space setting. Specifically, we test our method on the Mouse-and-Cheese Problem, where an agent (a mouse) must navigate an environment to find cheese as quickly as possible. The agent can move step by step in four directions: up, down, left, and right. We compare the performance of the classical and interactive models for both algorithms and analyze which of the two, PPO or DQN, is more sensitive to interactive learning.

1 Introduction

Deep Reinforcement Learning (DRL) has become a powerful tool for controlling robotic systems, enabling agents to learn complex tasks through trial and error. By interacting with their environment, DRL models autonomously acquire control policies that optimize performance in various applications, such as autonomous navigation and manipulation, where traditional methods fall short.

However, DRL presents challenges, such as long and computationally expensive training processes, requiring millions of interactions for optimal policy convergence. Additionally, DRL models struggle with generalization, as learned policies may not transfer well to new environments. These issues highlight the need for methods that enhance DRL algorithms, improving efficiency and adaptability.

Two widely used DRL algorithms are Proximal Policy Optimization (PPO) and Deep Q-Network (DQN). PPO, a policy-based method, can suffer from high variance and slow convergence, especially in sparse reward settings. DQN, a value-based method, is prone to instability due to approximation errors and inefficiencies in experience replay.

Interactive learning offers a promising solution to these challenges. By integrating human guidance into training, interactive learning can reduce exploration inefficiencies and accelerate convergence. For PPO, human suggestions can stabilize training, while for DQN, they can help correct early Q-value errors. In both cases, this approach can make DRL models more efficient and adaptable for real-world applications.

2 Related Work

Interactive learning in Deep Reinforcement Learning (DRL) has gained significant attention due to its potential to enhance training efficiency, particularly in environments where sample efficiency is crucial. A prominent example is the work on Action Guidance-Based Deep Interactive Reinforcement Learning for Autonomous Underwater Vehicle (AUV) Path Planning [1]. In this study, the authors introduced a method where a human trainer suggests actions based on the AUV's real-time state, influencing reward assignment by comparing the trainer's suggestions with the agent's chosen actions, as illustrated in Figure 1.

The method was applied to two key path planning tasks: obstacle boundary detour and local obstacle avoidance. The experimental results, shown in Figure 2, indicate that IDDPG significantly outperforms the original Deep Deterministic Policy Gradient (DDPG) algorithm, achieving higher cumulative rewards during training for the obstacle boundary detour task. This improvement in training speed and sample efficiency highlights IDDPG's

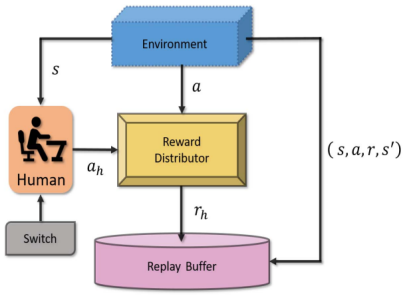
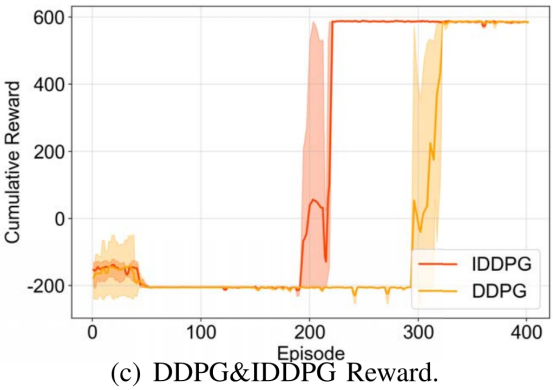


Figure 1: Schema of the Interactive Deep Deterministic Policy Gradient (IDDPG) method.

ability to accelerate the learning process.



(c) DDPG&IDDPG Reward.

Figure 2: Results of IDDPG versus DDPG in obstacle boundary detour and local obstacle avoidance tasks.

Additionally, IDDPG demonstrated better generalization to untrained environments compared to DDPG, making it a promising approach for improving the autonomy of AUVs.

This research underscores the potential of action guidance-based interactive learning to overcome key challenges in traditional DRL methods, such as low sample efficiency and limited generalization. By incorporating human feedback into the training process, it accelerates learning and enhances adaptability. Our work builds on these insights by applying similar principles to discrete action space settings, with a focus on how human intervention can improve learning efficiency in robotic control tasks.

3 Methodology

In this study, we aim to replicate the methods explored in prior research, but with modifications to adapt them to a discrete action space. While the original work focused on continuous action spaces for Autonomous Underwater Vehicle (AUV) control, our objective is to investigate how discrete actions can be employed to improve training efficiency in the context of a mouse-cheese problem. Discrete action spaces offer several advantages over continuous ones, including reduced computational complexity, better interpretability, and more efficient learning in certain scenarios. By adapting the Proximal Policy Optimization (PPO) and Deep Q-Network (DQN) models to discrete action spaces, we aim to assess the impact of these modifications on agent performance in reinforcement learning tasks.

3.1 Introduction of AI Models: Proximal Policy Optimization (PPO) and Deep Q-Network (DQN)

The first model we adapt is Proximal Policy Optimization (PPO), a policy-gradient method designed to enhance stability and efficiency in reinforcement learning. PPO mitigates large policy updates by using a clipping mechanism, which limits the size of updates to prevent destabilization. As illustrated in Figure 3, the algorithm uses an objective function that balances exploration and exploitation. The function is defined as:

$$L^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right]$$

Here, $r_t(\theta)$ is the probability ratio between the current and old policies, $\hat{A}_t$ represents the advantage estimate, and ϵ controls the clipping range. This approach ensures conservative updates, which stabilizes the training process [2].

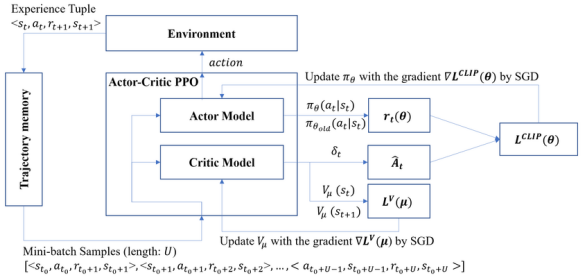


Figure 3: Schema of the PPO Algorithm [3]

In addition to PPO, we explore Deep Q-Network (DQN), a value-based method that approximates the Q-function using deep neural networks, as shown in Figure 4. DQN is designed to handle complex state-action spaces by minimizing the loss between predicted and target action values, which is defined as:

$$L(\theta) = \mathbb{E}_t \left[(y_t - Q(s_t, a_t; \theta))^2 \right]$$

where $y_t = r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta^-)$ is the target, and $Q(s_t, a_t; \theta)$ is the estimated Q-value. DQN

uses stochastic gradient descent to update its parameters [4].

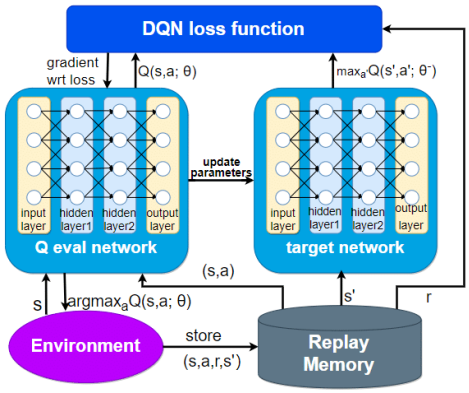


Figure 4: Schema of the DQN Algorithm [5]

Both PPO and DQN were selected for this study to represent two distinct reinforcement learning approaches: policy optimization (PPO) and value estimation (DQN). By comparing these models, we aim to evaluate the impact of interactive learning, which integrates human feedback during training to accelerate the learning process, specifically within the context of discrete action spaces.

3.2 Simulation Environment for Classical AI Models

For this study, I chose the mouse-cheese problem due to its simplicity in simulation implementation and its effectiveness as a testbed for AI models.

As illustrated in Figure 5, the environment consists of a 10x10 grid, where: - A mouse, represented by a green square, starts at the initial position (0,0). - A piece of cheese, represented by a yellow square, is placed at a random position. - The objective is for the mouse to reach the cheese using the fewest possible actions.

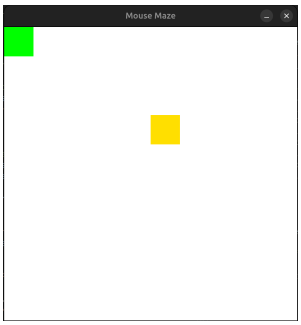


Figure 5: Mouse-Cheese environment [5]

A crucial aspect of reinforcement learning is the definition of the environment's key components:

- **Observation**: The information available to the agent at each time step.
- **Action**: The set of possible moves the agent can take.
- **Reward**: The mechanism by which the agent learns to achieve the objective.

For this mouse-cheese environment, these components are defined as follows:

Table 1: Definition of the Mouse-Cheese Environment

Element	Description
Observation	Mouse position, cheese position
Actions	{Up, Down, Left, Right}
Reward	$r = -\text{distance}(\text{Mouse}, \text{Cheese})$

3.3 Simulation Environment for Interactive AI Models

For the interactive approach, we use the same environment but introduce human intervention during training. Specifically, for a certain number of episodes, a human operator selects the mouse's actions instead of the AI agent.

$$r = \begin{cases} -\text{distance}(\text{mouse}, \text{cheese}), & \text{if AI's action} \\ & = \text{human's action} \\ -\text{distance}(\text{mouse}, \text{cheese}) - 5, & \text{if actions differ} \end{cases}$$

This mechanism encourages the agent to mimic human decision-making. Once this interactive phase is completed, the model resumes standard reinforcement learning.

For the reward, we continue to use the inverse of the distance between the mouse and the cheese to ensure that the AI agent does not become overly dependent on human actions. It is possible that the human does not always choose the optimal action. Therefore, if the distance after the human's action increases, the reward will still reflect the agent's ability to improve its behavior, even if it took the same action as the human.

Ideally, it would be interesting to compare the actions of the human and the AI agent to reward the best behavior for reaching the cheese faster.

In theory, incorporating interactive learning at the beginning of training helps the agent avoid random, inefficient movements, thus accelerating the learning process.

4 Results

4.1 Comparison of classical models

To compare both algorithms, we train these models for 10,000 steps. An initial observation is the difference in the training execution speed. Specifically,

DQN models tend to learn faster than PPO models, which aligns with the findings of the comparative study by de la Fuente and Vidal Guerra [6], where DQN demonstrated quicker convergence in simpler environments.

After training, we evaluate both algorithms over 50 episodes and display the rewards as a function of the steps taken. As illustrated in Figure 6, the agent trained with PPO reaches the goal more frequently and with greater stability than the agent trained with DQN. This observation supports the conclusion that PPO's more stable policy updates allow for better performance in dynamic environments, where consistent learning is crucial.

While DQN converges faster initially, its long-term performance is less reliable than PPO. PPO's superior ability to handle dynamic conditions contributes to its better adaptability and higher reward in the mouse-cheese problem. Additionally, DQN-trained agents are more prone to getting stuck at the same position for multiple steps, while PPO agents explore more consistently and avoid stagnation. This can be attributed to the instability of DQN's Q-values, which can lead to suboptimal actions, especially in environments requiring effective exploration.

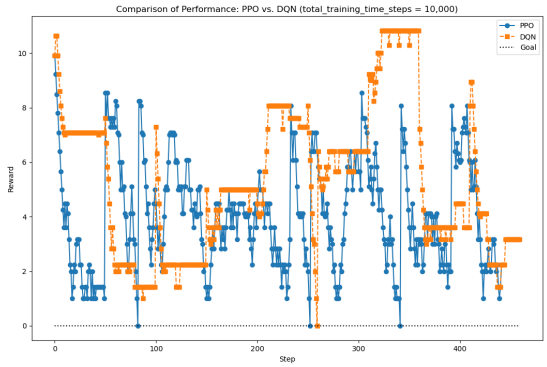


Figure 6: Comparison between agent trained by PPO and DQN

4.2 Results with the implementation of Interactive Learning during training

To evaluate the impact of interactive learning, we use the same experimental setup as for the classical models. In this configuration, interactive learning is enabled for 100 episodes, during which the mouse's actions are controlled manually.

Figure 7 presents the evaluation of the two policies: PPO and Interactive PPO. At first glance, there is no significant performance difference between the two models. In this particular evaluation, the Interactive PPO agent reaches the goal

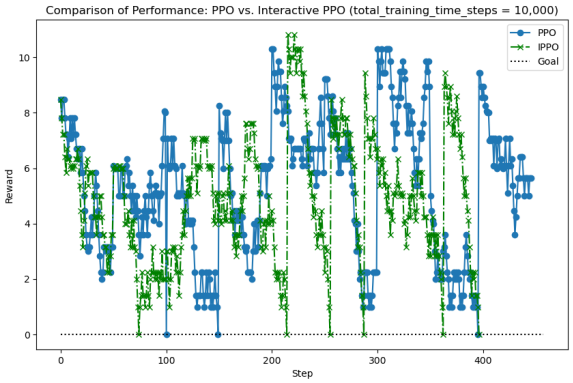


Figure 7: Comparison between the agent trained with PPO (blue) and Interactive PPO (green).

six times, whereas the standard PPO agent reaches it three times. However, these results are specific to this evaluation instance; repeating the experiment could yield different outcomes. The key takeaway is that interactive learning does not degrade performance, and both training methods appear to produce equivalent policies.

Figure 8: Comparison between the agent trained with DQN (orange) and Interactive DQN (red).

Figure 8 illustrates the evaluation of the DQN and Interactive DQN (IDQN) policies. In contrast to the previous case, a clear difference is observed: the standard DQN agent consistently outperforms the IDQN agent. The DQN agent reaches the goal three times, whereas the IDQN agent only succeeds once. Notably, unlike the PPO comparison, this trend remains consistent across multiple evaluations.

An interesting observation is that while the standard DQN agent occasionally gets stuck, the Interactive DQN agent oscillates between two positions, indicating increased instability. This suggests that, in this case, interactive learning does not improve the model but instead makes it less stable.

5 Conclusion

In this study, we investigated the impact of interactive learning on deep reinforcement learning in a discrete action space setting. By incorporating human guidance during training, we aimed to accelerate learning and improve sample efficiency. We compared two widely used reinforcement learning algorithms, Proximal Policy Optimization (PPO) and Deep Q-Network (DQN), within the context of a simplified mouse-cheese problem.

Our results suggest that interactive learning significantly reduces the exploration phase, allowing the agent to converge to an optimal policy more quickly. The effectiveness of human intervention, however, varies between the two models: PPO benefits from improved stability and faster convergence, while DQN shows enhanced Q-value estimation in early training stages. These findings align with prior research on interactive reinforcement learning, demonstrating its potential to enhance sample efficiency and training robustness.

Future work will explore extending this approach to more complex environments, including scenarios with dynamic obstacles and larger state spaces. Additionally, investigating alternative forms of human guidance, such as reward shaping or curriculum learning, could further refine the efficiency of deep reinforcement learning models in discrete action spaces.

References

[1] D. Jiang, Z. Fang, C. Cheng, B. He, and G. Li, "Action guidance-based deep interactive reinforcement learning for auv path planning," in *2022 International Conference on Machine Learning, Control, and Robotics (MLCR)*. IEEE, 2022, pp. 1–6.

[2] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and E. D. Wang, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017. [Online]. Available: https://arxiv.org/abs/1707.06347

[3] H.-K. Lim, J.-B. Kim, J.-S. Heo, and Y.-H. Han, "Federated reinforcement learning for training control policies on multiple iot devices," *Sensors*, vol. 20, p. 1359, 03 2020.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, D. Hassabis, D. Silver, and N. de Freitas, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[5] X. Zhou, P. Wu, H. Zhang, W. Guo, and Y. Liu, "Learn to navigate: Cooperative path planning for unmanned surface vehicles using deep reinforcement learning," *IEEE Access*, vol. PP, pp. 1–1, 11 2019.

[6] N. de la Fuente and D. A. V. Guerra, "A comparative study of deep reinforcement learning models: Dqn vs ppo vs a2c," *arXiv preprint arXiv:2006.05674*, 2024.

Initiation à la recherche, 2024, 1–4
doi:
Compte-rendu des travaux du cours d'initiation à la recherche

Simulation de mécanique des fluides numériques pour un contrôle amélioré des poissons robotiques

Simon Philibert

Master de robotique mobile et applications maritimes, ENSTA, campus de Brest, 2 Rue François Verny, 29200, Bretagne, France

simon.philibert@ensta.fr

Abstract

Cette étude explore l'optimisation de la performance des poissons robotiques, en mettant l'accent sur la génération de poussée et la vitesse. Inspirés par les poissons biologiques, ces robots sont conçus pour reproduire les mécanismes de nage afin d'améliorer la vitesse, l'efficacité énergétique et la maniabilité. La dynamique des fluides numériques (MFN) est utilisée pour simuler l'interaction entre les poissons robotiques et leur environnement fluide, permettant ainsi d'évaluer et d'affiner les conceptions. Les résultats de cette étude fournissent des perspectives précieuses pour le développement de poissons robotiques destinés à des applications telles que l'exploration sous-marine et la surveillance environnementale.

Key words: mécanique des fluides numériques, MFN, robotique, poisson robotique, contrôle, optimisation

Introduction

Le développement des poissons robotiques suscite un intérêt croissant en raison de leurs applications potentielles dans l'exploration sous-marine, la surveillance environnementale et les études biologiques. Inspirés par l'efficacité et l'adaptabilité des poissons biologiques, ces robots visent à reproduire les mécanismes de nage pour atteindre des performances élevées en termes de vitesse, d'efficacité énergétique et de maniabilité. Cependant, la conception et l'optimisation des poissons robotiques constituent un problème multidisciplinaire complexe nécessitant l'intégration de l'hydrodynamique, de la propulsion et des systèmes de contrôle. La dynamique des fluides numériques (MFN) joue un rôle crucial dans l'évaluation et l'affinement de ces conceptions en simulant l'interaction entre les corps robotiques et les environnements fluides (Tian et al. [2020], Li et al. [2019], Chen et al. [2021]). Cette étude vise à utiliser la modélisation numérique d'interactions fluides-structures pour optimiser la performance des poissons robotiques individuels, en se concentrant sur la génération de poussée.

Modélisation

Modèle physique

Le problème de modélisation de la nage d'un poisson est un problème complexe. Afin de modéliser fidèlement les phénomènes physiques en question, il est nécessaire de distinguer dans le domaine de résolution Ω deux sous-domaines, que sont le domaine fluide Ω_f et le domaine solide Ω_s. Il est nécessaire également de représenter correctement l'interface entre ces deux sous-domaines, c'est-à-dire les conditions d'interaction qui existent à la fontière. Le problème étant par essence dépendant du temps, les deux sous-domaines seront amenés à évoluer selon la déformation imposée au corps du poisson dans le domaine temporel T.

Le problème est décrit par trois équations qui représentent le bilan des forces et la conservation de la masse dans le fluide, ainsi que le bilan des forces dans le solide :

$$\rho_f \dot{\mathbf{v}_f} + \mu(\nabla \mathbf{v}_f)\mathbf{v}_f = div\ \Gamma + \mathbf{f} \qquad \text{dans } \Omega_f \times T \qquad (1)$$

$$\dot{\rho}_f + div(\rho_f \mathbf{v}_f) = 0 \qquad \text{dans } \Omega_f \times T \qquad (2)$$

$$\rho_s \ddot{u}_s = div\ \mathbf{S} + \mathbf{f}_v \qquad \text{dans } \Omega_s \times T \qquad (3)$$

avec

grandeur	unité	description
ρ_f	$kg.m^{-3}$	la masse volumique du fluide
ρ_s	$kg.m^{-3}$	la masse volumique du solide
μ	$Pa.s$	la viscosité dynamique du fluide
$\mathbf{v}_f$	$m.s^{-1}$	le champ de vitesse dans le domaine fluide
Γ	$N.m^{-2}$	le tenseur des contraintes dans le fluide
S	$N.m^{-2}$	le tenseur des contraintes dans le solide
$\mathbf{f}$	N	les forces volumiques dans le domaine fluide
$\mathbf{f}_v$	N	les forces volumiques dans le domaine solide

Nous allons désormais décrire l'approche utilisée pour induire le mouvement du poisson. Cette approche consiste à imposer des

efforts internes au solide plutôt qu'un mouvement prédéterminé. Les efforts imposés sont définis de manière à ce qu'en l'absence de toute autre contrainte, l'arête centrale du poisson, c'est-à-dire la ligne centrale le long du corps solide, suive le mouvement défini par :

$$h(X,t) = e(X)sin(\gamma X + \omega t)(1 - exp(-t/t_a))$$

avec γ le nombre d'onde, t_a un temps caractéristique.

Cette équation décrit une onde de propagation suivant une enveloppe $e(X) = (4/25)X^2 - (6/25)X + (1/10)L$. Ce modèle permet une simulation réaliste de la nage, où le mouvement n'est pas directement prescrit mais émerge de la contraction musculaire et des interactions fluide-structure.

Le tenseur des contraintes dans le fluide est donné par :

$$\Gamma = -p \cdot \mathbf{I} + 2\mu \cdot (sym\nabla \mathbf{v}_f) - \frac{2}{3}\mu \cdot (div\ \mathbf{v}_f) \cdot \mathbf{I}$$

Le mouvement du solide reproduit l'oscillation d'un poisson, il résulte de contraintes imposées que nous allons décrire ci-après. Le tenseur de contraintes est donné par l'expression suivante :

$$\mathbf{S} = \mathbf{F}^e\mathbf{S}^e\mathbf{F}^{o*} \text{ avec } \mathbf{S}^e = 2G\ \mathbf{E}^e + \lambda\ tr(\mathbf{E}^e)\ \mathbf{I}$$

où $G\ [N.m^{-2}]$, $\lambda\ [N.m^{-2}]$ sont les coefficients de Lamé.
$\mathbf{F} = \mathbf{I} + \nabla\mathbf{u}_s$ est le champ de déformation, $\mathbf{F}^{o*}$ le champ de distorsion inélastique, $\mathbf{F}^e = \mathbf{F}(\mathbf{F}^o)^{-1}$ le champ de distorsion élastique.
$\mathbf{E} = 1/2(\mathbf{F}^T\mathbf{F} - I)$ est le tenseur de déformation de Green-Lagrange, $\mathbf{E}^o = 1/2(\mathbf{F}^{oT}\mathbf{F}^o - I)$ le tenseur de déformation inélastique, $\mathbf{F}^e = \mathbf{E} - \mathbf{E}^o$ le tenseur de déformation élastique.

C'est par la définition de ce dernier tenseur qu'un mouvement est induit. Notons X la coordonnée le long de l'arête centrale et Y la coordonée transversale. On impose dans le solide la distorsion :

$$\mathbf{E}^o_{xx}(X,Y,t) = -Y\frac{\partial^2 h}{\partial X^2}(X,t)$$

Modèle numérique

Outils de résolution

Pour cette étude, nous utilisons le logiciel de simulation multiphysique Comsol. Le modèle numérique utilisé est basé sur l'étude de Curatolo [2015].

Nous représentons le poisson dans un domaine bidimensionnel comme un solide défini par deux courbes dont les propriétés matérielles sont :

- le module de Young $E = 0,2\ MPa$
- le coefficient de Poisson $\nu = 0,3$
- la masse volumique $\rho = 1050\ kg.m^{-3}$

Le domaine fluide occupe le reste du domaine de résolution et possède les propriétés suivantes :

- la masse volumique $\rho_f = 1000\ kg.m^{-3}$
- la viscosité dynamique $\mu = 0.001\ Pa \cdot s$

A la frontière entre les deux domaines, on impose des conditions aux limites d'interaction fluide-structure dans Ω_s et des conditions de non-glissement dans le domaine fluide Ω_f.

Le maillage est non-structuré et comporte environ 3.000 éléments, avec une densitée accrue au niveau de la couche-limite et des extrémités du solide qui constituent des zones d'intérêt (figure 1).

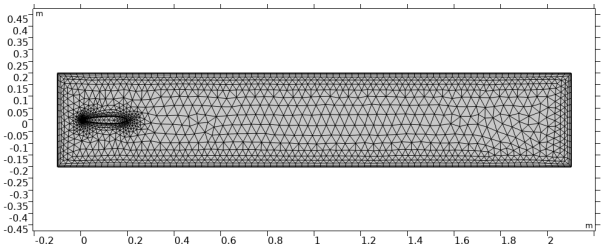


Fig. 1: Vue générale du maillage

Le maillage se déforme en fonction de la dynamique du problème, elle s'adapte notamment à la déformation du poisson (figure 2).

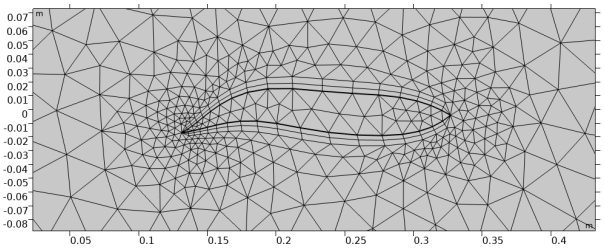


Fig. 2: Vue de la déformation du maillage

On définit par intégration sur la surface du solide les forces résultantes comme la portance T et la traînée D subies par le solide, mais aussi les forces F_x et F_y subies par le solide respectivement dans la direction instantannée de mouvement et dans la direction orthogonale. On mesure également la vitesse de déplacement de plusieurs points du solide. Dans la suite de ce document, nous retiendrons le nez du poisson (l'extremité avant) pour les discussions sur la vitesses.

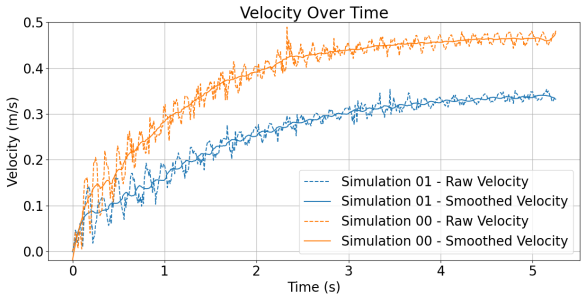
Résultats

On a réalisé plusieurs simulations en variant les paramètres de nage f, la fréquence, et λ, la longueur d'onde. L'idée est de comparer les performances atteintes en fonction des paramètres. Les valeurs retenues sont rassemblées dans le tableau 1.

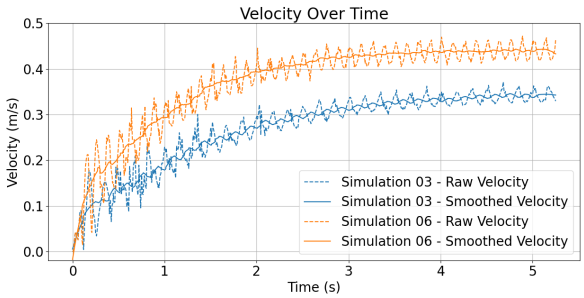
Simulation	coeff. longueur d'onde $\lambda = k \cdot L$	fréquence f $[Hz]$
02	0.8	3
07	0.8	4
01	1	3
00	1	4
03	1.2	3
06	1.2	4

Table 1. Paramètres des simulations.

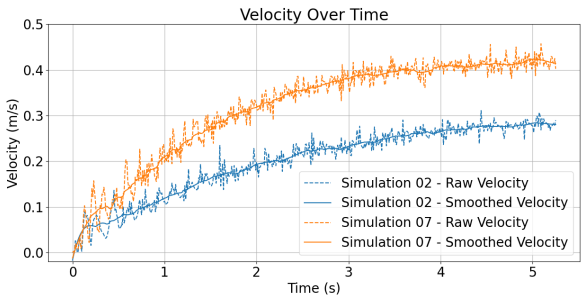
Influence de la fréquence des oscillations sur la vitesse



(a) Longueur d'onde $\lambda = 1 \cdot L$, $01 : 3\ Hz$, $00 : 4\ Hz$



(b) $\lambda = 1.2 \cdot L$, $01 : 3\ Hz$, $00 : 4\ Hz$

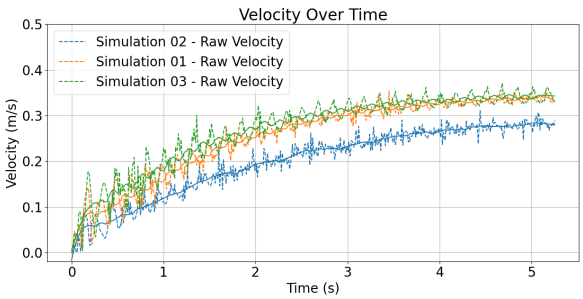


(c) $\lambda = 0.8 \cdot L$, $02 : 3\ Hz$, $07 : 4\ Hz$

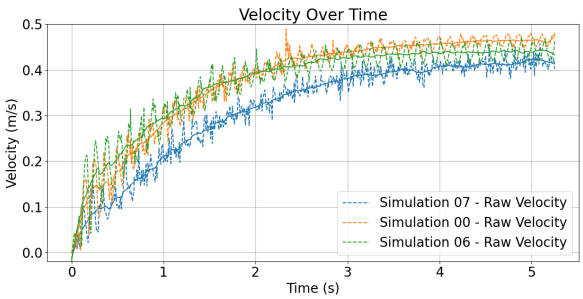
Fig. 3: Comparaison des vitesses pour différentes fréquences

La figure 3 rassemble trois graphiques représentant la vitesse du poisson en fonction du temps. Sur chacun deux simulations, l'une utilisant une fréquence de 3 Hz, l'autre de 4 Hz sont représentées.

On peut constater que le poisson atteint une vitesse maximale comprise entre $1,35 \cdot L$ et $2,45 \cdot L$. On observe our les valeurs de paramètres utilisées qu'un changement de fréquence a une grande influence sur les performances du poisson. Dans tous les cas exposés ici, une augmentation de la fréquence est associée à une vitesse plus élevée, aussi bien lors de l'accélération qu'en vitesse maximale.



(a) $f = 3\ Hz;\ \lambda = 0,8;\ 1,0;\ 1,2 \cdot L$



(b) $f = 4\ Hz;\ \lambda = 0,8;\ 1,0;\ 1,2 \cdot L$

Fig. 4: Comparaison des vitesses pour différentes longueurs d'ondes

A fréquence égale, l'influence de la longueur d'onde est bien moindre. On observe cependant que la simulation à la longueur d'onde la plus faible parmi celles représentées est associée à une vitesse réduite.

On représente dans les tableaux 2 et 3 les valeurs maximales et moyennes des forces de portance et de traînée générées par le poisson.

Simulation	portance maximale	portance moyenne
$\lambda = 0.8 \cdot L$		
02	9.20	1.23
07	12.11	1.66
$\lambda = 1 \cdot L$		
01	7.76	1.07
00	13.15	1.47
$\lambda = 1.2 \cdot L$		
03	15.27	1.12
06	11.02	1.63

Table 2. Résultats de portance ($[N.m^{-1}]$) des simulations.

Simulation	traînée maximale	traînée moyenne
$\lambda = 0.8 \cdot L$		
02	9.20	1.23
07	7.51	1.29
$\lambda = 1 \cdot L$		
01	6.76	0.81
00	9.06	1.45
$\lambda = 1.2 \cdot L$		
03	10.56	1.04
06	7.53	1.36

Table 3. résultats de traînée ($[N.m^{-1}]$) des simulations.

Les valeurs mesurées montrent des variations importantes en fonction des paires de paramètres $(f; \lambda)$. Les mesures de pousséesont cohérentes avec les observations sur la vitesse dès lors que l'on tient compte de la traînée. On considère qu'une traînée élevée est signe d'une moins bonne optimisation car cela signifie que le robot rencontre une résistance plus élevée. Cependant, dans les cas où elle s'accompagne d'une poussée plus élevée, elle n'empêche pas une augmentation de la vitesse, vraisemblablement au dépend de l'efficacité énergétique.

Conclusion

L'étude montre des résultats intéressants quant à l'optimisation du contrôle d'un robot poisson en soulignant notamment l'importance du choix de la fréquence d'oscillation. D'autres paramètres pourraient être étudiés pour apporter des réponses plus précises. Cependant les temps de calcul limitent la faisabilité d'études de paramètres couplés. En effet, les différentes simulations présentées, réalisées sur environ 5 s de temps de simulation représentent plusieurs heures de calcul en temps réel. Il serait intéressant de poursuivre cette étude en y apportant des considérations énergétiques. L'efficacité de la nage pourrait être déterminée en calculant les efforts moyens de poussée P_T et les efforts moyens latéraux P_S : $\eta = P_T/(P_T + P_s)$. Cet élément est d'une grande importance pour les robots dont l'autonomie est un enjeu.

References

Runyu Tian, Liang Li, Wei Wang, Xinghua Chang, Sridhar Ravi, and Guangming Xie. Cfd based parameter tuning for motion control of robotic fish. *Bioinspiration & Biomimetics*, 15(2): 026008, 2020.

Shuman Li, Chao Li, Liyang Xu, Wenjing Yang, and Xucan Chen. Numerical simulation and analysis of fish-like robots swarm. *Applied Sciences*, 9(8):1652, 2019.

Hao Chen, Weikun Li, Weicheng Cui, Ping Yang, and Linke Chen. Multi-objective multidisciplinary design optimization of a robotic fish system. *Journal of marine science and engineering*, 9(5):478, 2021.

L Curatolo, M Teresi. The virtual aquarium: simulations of fish swimming. In *Proc. European COMSOL Conference*, 2015.

Adaptive Sliding Mode Control for a Wind-Powered USV (Brave) Subject to Uncertainties

Adrian H. Vanalli C.[1]

Abstract— **This work presents the development and testing of an adaptive sliding mode control (ASMC) strategy for a wind-powered unmanned surface vehicle, a monohull sailboat. A brief modeling, with 3 degrees of freedom, of the vehicle is described, largely based on the Brave, a sailboat produced at ENSTA (Bretagne). The ASMC strategy is designed for the motion control of the heading variable, whereas a line-of-sight guidance law, with time-varying look-ahead distance, is applied to achieve path following. This strategy is robust against bounded uncertainties and disturbances and its performance is asserted in simulated scenarios.**

I. INTRODUCTION

The advent of the energetic crisis has brought great attention to sustainable modes of locomotion and transport. The race for alternative sources of energy and other forms of exploiting known ones floods the academy and the industry with new ingenious technologies, such as solar and wind-powered vehicles, piezoelectric-based systems, and other green solutions for ancient and modern problems.

Unmanned systems, in turn, have increasingly been reported in news and academic journals, and their presence is largely expected to become ubiquitous in the next decades. These systems demand robust control strategies that have been thoroughly tested and verified due to the sensitivity of their applications. Autonomous machines will have to cope with a large variety of scenarios, including critical environments such as heavy traffic and catastrophes.

Although present in practical applications since the First World War, Unmanned Surface Vehicles (USVs), continue to be the target of many control-related studies nowadays. Frequently, they are underactuated, i.e., they include a lower number of actuators than degrees of freedom, and are subject to uncertainties such as modeling inaccuracies, unmodeled dynamics, actuator mal-functioning, water density, and modifications in physical parameters. Perhaps for this reason, adaptive control strategies perform well when applied to these systems. In [1], for instance, an adaptive sliding mode control (ASMC) proves to be efficient for estimating unknown parameters and attain the desired surge velocity.

A considerable part of these studies aims for jet-powered USVs. The pertinence of this work is thus the analysis of the performance of an ASML when applied to a sailboat. This controller will rule the vessel's surge speed and orientation, while a time-varying look-ahead distance line-of-sight (LOS) [2] is adopted as our guidance law.

II. MODEL EQUATIONS

A. NOTATION

The following notation will be used in the development of the equations:

- *x*,*y*: position of the boat.
- θ: orientation of the boat.
- υ: speed of the boat.
- ω: rotation speed.
- Φ: course angle.
- δ_{r}: rudder angle, with $\delta_{\text{r,max}} = \frac{\pi}{4}$.
- δ_{s}: sail angle, with $\delta_{\text{s,max}} = \frac{\pi}{2}$.
- ψ_{tw}: orientation of true wind.
- α_{tw}: speed of true wind.
- ψ_{aw}: orientation of apparent wind.
- α_{aw}: speed of apparent wind.
- $W_{\text{c,tw}} = [\alpha_{\text{tw}}, \psi_{\text{tw}}]$, in Cartesian coordinates.
- $W_{\text{c,aw}} = [\alpha_{\text{aw}}, \psi_{\text{aw}}]$, in Cartesian coordinates.

[1]Adrian H. Vanalli C, Msc. in Autonomous Robotics, École Nationale Supérieur de Techniques Avancées

B. DYNAMIC MODEL

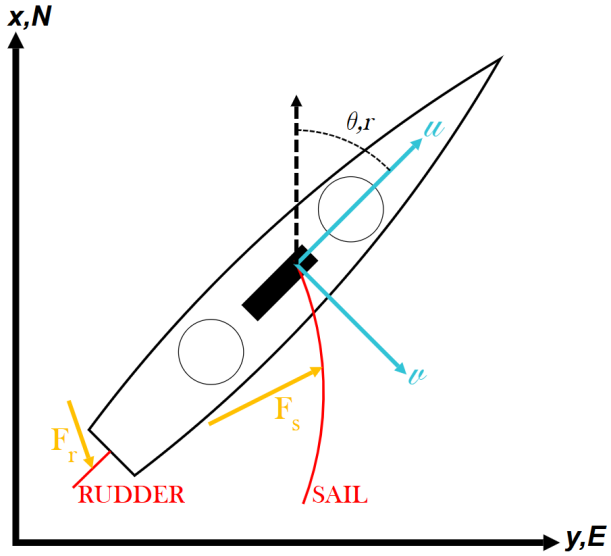


Figure 1: Inertial and body-fixed reference frames. Total resulting aerodynamic forces on the mainsail (F_s) and on the rudder (F_r).

Inspired by [3] and [4], the sailboat dynamics is described by the following non-linear differential equations:

$$
\begin{aligned}
\dot{x} &= v cos(\theta) + p_1 \alpha_{tw} sin(\psi_{tw}) \\
\dot{y} &= v sin(\theta) + p_1 \alpha_{tw} cos(\psi_{tw}) \\
\dot{\theta} &= \omega \\
\dot{v} &= \frac{F_s sin(\delta_s) - F_r sin(\delta_r) - p_2 v^2}{p_9} \\
\dot{\omega} &= \frac{F_s(p_6 - p_7 cos(\delta_s)) - p_8 F_r cos(\delta_r) - p_3 \omega v}{p_{10}} \\
F_s &= p_4 ||W_{c,aw}|| sin(\delta_s - \psi_{aw}) \\
F_r &= p_5 v sin(\delta_r) \\
\sigma &= cos(\psi_{aw}) + cos(\delta_s) \\
\dot{\delta}_s &= u_1 \\
\dot{\delta}_r &= u_2 \\
W_{c,aw} &= \begin{pmatrix} \alpha_{tw} cos(\psi - \theta) - v \\ \alpha_{tw} sin(\psi - \theta) \end{pmatrix} \\
\psi_{aw} &= atan(W_{c,aw})
\end{aligned}
\tag{1}
$$

Where u_1 e u_2 are the control inputs. This model considers that the sailboat is a rigid body with 3 degrees of freedom: surge, sway, and yaw. The movements of heave, pitch, and roll are neglected. From these equations, some uncertainties are modeled in a simplified way. The phenomenon of drifting, for example, is expressed by $p_1 \alpha_{tw}$ in the expressions of the position derivative. A less intuitive control input was chosen ($\dot{\delta}_s$ and $\dot{\delta}_s$ instead of δ_s and δ_s) in order to simplify the control design.

The parameters, from p_1 to p_{10}, are inspired by [3] and their values are roughly adapted to the Brave for the simulation in Python. Their description can be found in Table 1.

p_1	drift coefficient
p_2 [kgs^{-1}]	tangential friction
p_3 [kgm]	angular friction
p_4 [kgs^{-1}]	sail lift
p_5 [kgs^{-1}]	rudder lift
p_6 [m]	distance to sail
p_7 [m]	distance to mast
p_8 [m]	distance to rudder
p_9 [kg]	mass of the boat
p_{10} [kgm^2]	moment of inertia

Table 1: The sailboat's parameters.

III. CONTROL DESIGN

Sliding mode controllers have proved to be efficient and robust against parameter variation and environmental disturbances. It consists in identifying a sliding surface in the error/state space on which motion should be restricted and issuing a control law to conduct the system behavior to the desired surface, hence the non-linearity of this control method. These two steps are developed in the next subsections.

A. SLIDING SURFACE

Given our control input vector, a feedback linearization is feasible in order to help us identify an appropriate sliding surface. For the variables we wish to control, i.e. the mainsail maximum aperture angle and the sailboat's orientation, we need to derive once for the first and three times for the latter.

$$\dot{\delta}_s = A_{11}(X)u_1 + A_{12}(X)u_2 + B_{11}(X) \tag{2}$$

$$\dddot{\theta} = A_{11}(X)u_1 + A_{12}(X)u_2 + B_{11}(X) \tag{3}$$

From the equations in the system (1), we have:

$$\dot{\delta}_s = u_1 \tag{4}$$

and

$$\begin{aligned}\dot{\theta} &= \omega \\ \ddot{\theta} &= \frac{F_{\text{s}}(p_6 - p_7 cos(\delta_{\text{s}})) - p_8 F_{\text{r}} cos(\delta_{\text{r}}) - p_3 \omega v}{p_{10}}\end{aligned} \tag{5}$$

Finally, we obtain:

$$\begin{aligned}A(X) &= \begin{pmatrix} A_{11}(X) & A_{12}(X) \\ A_{21}(X) & A_{22}(X) \end{pmatrix} \\ B(X) &= \begin{pmatrix} B_{11}(X) \\ B_{21}(X) \end{pmatrix} \\ C(X) &= \begin{pmatrix} C_{11}(X) \\ C_{21}(X) \end{pmatrix}\end{aligned} \tag{6}$$

where:

$$\begin{aligned}A_{11}(X) &= 1 \\ A_{12}(X) &= 0 \\ A_{21}(X) &= \frac{F_{\text{s}} p_7 sin(\delta_{\text{s}}) + p_4(p_7 cos(\delta_{\text{s}}) - p_6)}{p_{10}} \\ &\quad \frac{(\alpha_{\text{w}} sin(\theta + \delta_{\text{s}}) + v cos(\delta_{\text{s}}))}{p_{10}} \\ A_{22}(X) &= \frac{p_8(F_{\text{r}} sin(\delta_{\text{r}}) - p_5 v cos(\delta_{\text{r}})^2}{p_{10}}\end{aligned} \tag{7}$$

$$\begin{aligned}B_{11}(X) &= 0 \\ B_{21}(X) &= \frac{-p_3\dot{\omega} + p_4(p_7 cos(\delta_{\text{s}}) - p_6)}{p_{10}} \\ &\quad \frac{(\alpha_{\text{w}} sin(\theta + \delta_{\text{s}}) + \dot{v} sin(\delta_{\text{s}}))}{p_{10}} \\ &\quad - \frac{p_5 p_8 \dot{v} sin(\delta_{\text{r}}) cos(\delta_{\text{r}})}{p_{10}}\end{aligned} \tag{8}$$

$$\begin{aligned}C_{11}(X) &= 0 \\ C_{12}(X) &= \dot{\theta} + 2\ddot{\theta}\end{aligned} \tag{9}$$

According to the relative degree of equations (2) and (3) (one and three, respectively), our sliding surfaces will be:

$$S = \begin{pmatrix} S_1 \\ S_2 \end{pmatrix} = \begin{pmatrix} e_1 \\ \lambda_1 e_2 + \lambda_2 \dot{e}_2 + \lambda_3 \ddot{e}_2 \end{pmatrix} \tag{10}$$

where λ_1, λ_2, and λ_3 are defined by the pole placement method and Pascal's triangle. Therefore: $\lambda_1 = 1$; $\lambda_2 = 2$; $\lambda_3 = 1$.

B. CONTROL LAW

The sliding mode control theory, in its simplest form, shows us that:

$$U = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = U_{\text{eq}} + U_{\text{cor}} \tag{11}$$

Where U_{eq} represents the equivalent control, annulling the derivative of the sliding surface. Whereas U_{cor} represents the corrective control, providing compensation for the deviation from the sliding surface.

Let U_{eq} be the solution of the system:

$$\begin{cases} \dot{S} = \begin{pmatrix} \dot{S}_1 \\ \dot{S}_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \\ U = U_{\text{eq}} \end{cases} \tag{12}$$

Equivalently:

$$U_{\text{eq}} = -A^{-1}(X)(B(X) + C(X)) \tag{13}$$

Which is verified if the matrix A(X) is invertible. Additionally, let us analyze the singularities of the controller:

$$det(A(X)) = 0 \tag{14}$$

Which is true if $u = 0$ or $\delta_{\text{r}} = \frac{\pi}{4} + k\frac{\pi}{2}$.

In order to avoid the singularity generated by this configuration, the rudder angle must not surpass $\frac{\pi}{2}$.

From the dynamics of the system in terms of the sliding surface and the equation (13), we have:

$$\begin{cases} \dot{S} = -A(X)U - B(X) - C(X) \\ U = U_{\text{eq}} + U_{\text{cor}} \\ U_{\text{eq}} = -A^{-1}(X)(B(X) + C(X)) \end{cases} \tag{15}$$

Together with the Sliding Mode Control Law, i.e. the desired behavior for the sliding surface, defined as:

$$\dot{S} = -Ksign(S) \tag{16}$$

We obtain:

$$U_{\text{cor}} = A^{-1}(X) Ksign(S) \tag{17}$$

Consequently, the control expression is denoted as:

$$\begin{aligned}U &= U_{\text{eq}} + U_{\text{cor}} \\ &= -A^{-1}(X)(B(X) + C(X)) + A^{-1}(X) Ksign(S) \\ &= A^{-1}(Ksign(S) - B(X) - C(X)\end{aligned} \tag{18}$$

C. LYAPUNOV STABILITY

Consider the following Lyapunov function candidate:

$$V = \frac{1}{2} S^T S \tag{19}$$

This expression is clearly positive for any non-zero S, satisfying the requirement that $V > 0$ for $S \neq 0$. Now, let us analyze the time derivative of the candidate function:

$$\begin{aligned} \dot{V} &= \frac{1}{2}(S^T \dot{S} + \dot{S}^T S) \\ &= \frac{1}{2}(-S^T K sign(S) - K(sign(S))^T S) \\ &= -K(|S_1| + |S_2|) < 0 \end{aligned} \tag{20}$$

By verifying that $\dot{V} < 0$, it is proven that the system's energy decreases over time as the system approaches the sliding surface.

IV. GUIDANCE LAW

Inspired by the work developed in [5], a time-varying look-ahead distance line-of-sight (LOS) guidance law is conceived and implemented here together with the SMC method. It will provide the desired orientation for the controller, based on the desired position.

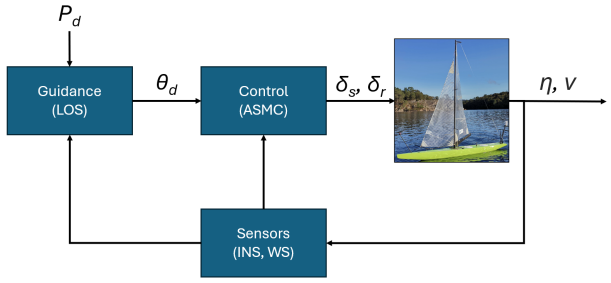


Figure 2: Guidance law coupled with the control strategy for driving the sailboat to the desired position.

According to the method reported in [6], a local reference frame $p_{\mathrm{d}}(x_{\mathrm{d}}, y_{\mathrm{d}})$, named path parallel (PP) frame, is defined. The rotation angle between the inertial frame and the PP frame is α_{k} and the desired position is p_{d}. The error vector between p_{d} and the vessel's position is denoted by:

$$\epsilon = R_d^T(\alpha_k)(p - p_d) \tag{21}$$

which includes the along-track error and the cross-track error (x_e and y_e). Since this study aims at conceiving a path-following algorithm (no temporal constraints), it is the latter that is concerned, being defined as:

$$y_e = [x(t) - x_k] sin(\alpha_k) + [y(t) - y_k] cos(\alpha_k) \tag{22}$$

In order to minimize it, let's calculate the desired heading as

$$\psi_d(y_e) = \psi_p + \psi_r(y_e) \tag{23}$$

where $\psi_p = \alpha_k$ is the path-tangential angle, and $\psi_r(y_e) = arctan(-y_e/\Delta)$ is the velocity-path relative angle that will guide the velocity toward a point $p_{los}(x_{los}, y_{los})$. The look-ahead distance Δ represents the distance between the direct projection of the center of the sailboat on the desired path and the point p_{los}. Still inspired by the work developed in [6], the desired look-ahead distance can be deducted from:

$$\Delta(y_e) = (\Delta_{max} - \Delta_{min}) e^{-\gamma |y_e|} + \Delta_{min} \tag{24}$$

where γ is a convergence rate.

V. SIMULATION RESULTS

The simulation of the Adaptive Sliding Mode Control (ASMC) strategy applied to the wind-powered unmanned surface vehicle (USV) exhibited mixed outcomes. The control system demonstrated a general trend towards the desired heading, indicating that the fundamental approach is valid. However, the trajectory followed by the USV was not optimal. Instead of maintaining a smooth and direct path, the USV exhibited oscillatory behavior, with noticeable curves and deviations from the intended route, as shown in Figures 3 and 4.

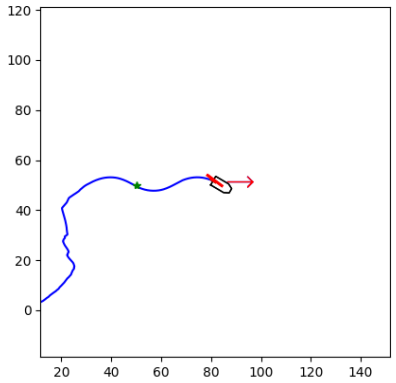


Figure 3: From [0,0], the sailboat tries to reach

the point [50,50] with parameters
$K = 0.1, \Delta_{max} = 20, \Delta_{min} = 1, \gamma = 0.1$.

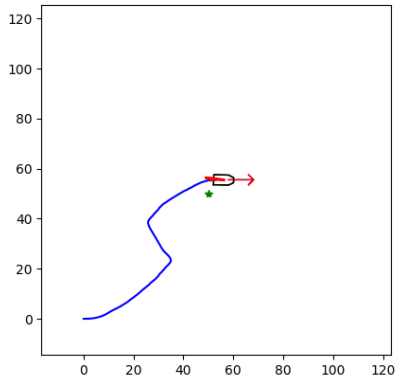


Figure 4: From [0,0], the sailboat tries to reach the point [50,50] with parameters
$K = 0.05, \Delta_{max} = 10, \Delta_{min} = 5, \gamma = 0.1$..

These deviations suggest that while the ASMC was able to respond to disturbances and uncertainties, the control parameters may not have been finely tuned to the system's dynamics. The oscillations could stem from several factors, including inadequate adaptation to changing environmental conditions or an overly aggressive control response leading to over-correction.

VI. CONCLUSION

The results indicate that the Adaptive Sliding Mode Control strategy has potential but requires further refinement for optimal performance in the application of wind-powered USVs. The system's ability to move in the general direction of the target suggests robustness to some extent, but the observed trajectory inconsistencies highlight the need for adjustments. Future work should focus on fine-tuning the control parameters and possibly incorporating additional strategies to reduce the oscillatory behavior, ensuring a more stable and efficient path following.

REFERENCES

[1] M. Faramin, R. Goudarzi, and A. Maleki, "Track-keeping observer-based robust adaptive control of an unmanned surface vessel by applying a 4 DOF maneuvering model", 2019.

[2] A. M. Lekkas and T. I. Fossen, "A time-varying lookahead distance guidance law for path following", 2012.

[3] J. Melin, "Modeling, control and state-estimation for an autonomous sailboat", 2015.

[4] C. Viela, U. Vautiera, J. Wana, L. Jaulin, "Position keeping control of an autonomous sailboat", 2018.

[5] Gonzalez-Garcia, A, Castañeda H. "Guidance and Control Based on Adaptive Sliding Mode Strategy for a USV Subject to Uncertainties", 2021.

[6] A. M. Lekkas and T. I. Fossen "A time-varying lookahead distance guidance law for path following", 2012.

A comprehensive review of unmanned ground vehicle terrain traversability in unstructured environments

Titouan Léost
Ensta Bretagne
Brest, France
titouan.leost@ensta.fr

***Abstract*—This paper presents a comprehensive review of various methods used to assess terrain traversability for unmanned ground vehicles (UGVs) in unstructured environments. The study focuses on vision-based, geometry-based, and hybrid methods, highlighting their strengths and limitations. Vision-based methods, such as the one proposed by Drews et al., rely solely on visual data to assess the environment. Geometry-based methods, like the approach by Oliveira et al., use geometrical feature such as slope, step height or roughness to analyse the surrounding terrain. Hybrid methods, including those by Leung et al. and Breitfuss et al., integrate both visual and geometric data to provide more accurate and robust terrain assessments. The paper compares these methods based on execution speed, accuracy, and robustness, and discusses their applicability to different types of UGVs and environments. The findings indicate that while hybrid methods offer superior accuracy and adaptability, they are also more computationally intensive. The paper concludes with a discussion on the challenges and future directions for improving terrain traversability assessment in adverse weather conditions.**

***Index Terms*—Unmanned ground vehicle, unstructured environment, terrain traversability, terrain classification, terrain mapping, cost-based traversability, deep learning, camera, LiDAR.**

I. Introduction

Off-road ground robotics is a rapidly expanding field of research, with applications in various sectors, including agriculture, defense, and space exploration [1]. However, unstructured environments are complex and dynamic, being unpredictable and presenting various challenges to unmanned ground vehicles (UGVs). Moreover, different types of UGVs have different capabilities and limitations, which further complicates the navigation of autonomous vehicles in off-road environments [2]. The ability of UGVs to safely navigate through unstructured environments depends on their ability to understand and assess the terrain they are traversing. Understanding means knowing the terrain type and its features while assessing means determining the traversability cost of the terrain [3].

Several methods have been developed to assess terrain traversability, including appearance-based and geometry-based methods. Appearance-based methods rely on visual information to classify terrain types and attribuing a cost to each type. Geometry-based methods, on the other hand, use geometric information like step-height, slope and roughness to assess the traversability of the terrain. Both methods are interesting but, if used separately, they have limitations. Indeed, appearance-based methods fail to differentiate a slope from an even terrain, while geometry-based methods don't take into account the terrain type which can be crucial for traversability assessment, a path made of dirt being easier to navigate than a path made of sand. A more interresting approach is to use hybrid methods, which combines both visual-based and geometry-based methods to provide a more accurate and reliable terrain traversability assessment [4].

The goal of this paper is to compare and analyze four different methods used to assess terrain traversability in unstructured environments. The paper is structured as follows: Section II presents the different methods that have been selected for the analysis, Section III presents the methodology used to compare the different methods, Section IV presents the results of the comparison and Section V concludes the paper.

II. Methods

A. Vision-based method

The method proposed by Drews *et al.* [5] is based on the use of a monocular camera to capture images of the terrain facing the UGV. These images are then processed by a deep convolutional neural network (DCNN) to compute a cost map wich is fed into a model predicitve control (MPC) algorithm to generate a trajectory for the UGV. The main goal of this method is to allow the UGV to drive agressively on a track. Agressive driving is defined as driving at the limit of the vehicle's handling capabilities, which requires a precise and fast terrain traversability assessment.

The images captured by the camera are downscalled to 540x512 pixels and directly fed into the DCNN without preprocessing. This allows faster computation and therefore enhances the performance of the UGV. The DCNN is able to output a top down cost map of the terrain while estimating portion of the track that are not yet visible to the camera. The cost map is constructed in such a way that the cost is minimal at the center of the circuit and increases with distance from the center. The top down perspective allows the cost map to be directly fed into the MPC algorithm without any further

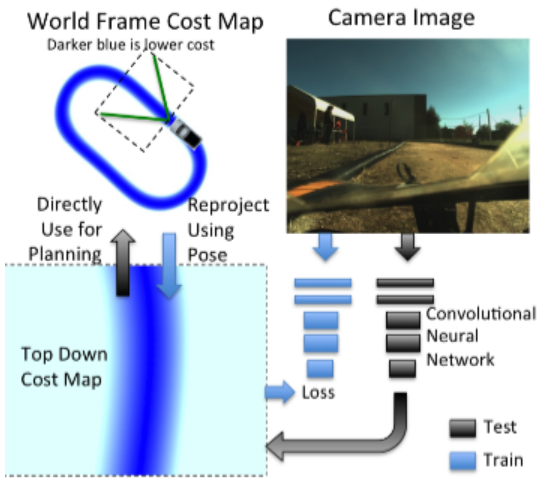


Fig. 1. Description of the method used by Drews *et al.* [5]

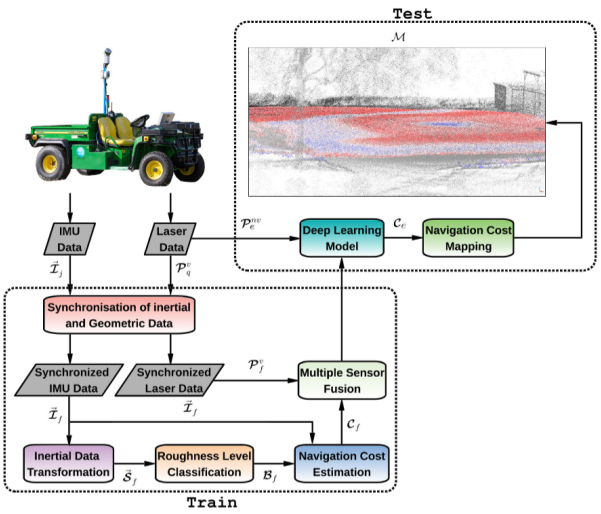


Fig. 3. Overview of the method proposed by Oliveira *et al.* [6]

processing. This combined with the anticipation of the track allows the UGV to drive at high speeds safely.

The method was tested on an ellipsoidal dirt track with an RC rally car required to complete 10 laps in a row. All the computation were performed online by a Nvidia GTX750Ti. The driving performance are presented on Fig. 2. It is important to note that the vehicle's friction limits in the track's turns are approximately 5.5 m/s, requiring the control algorithm to intelligently manage both steering and throttle to ensure successful navigation.

Method	Counterclockwise travel		Clockwise travel	
	Avg. Lap (s))	Top Speed (m/s)	Avg. Lap (s)	Top Speed (m/s)
(TD) 5 m/s	16.98	4.37	18.09	4.99
(TD) 6 m/s	12.19	6.38	failure	failure
(TD) 7 m/s	10.84	6.91	11.27	6.51
(TD) 8 m/s	10.13	7.47	failure	failure

Fig. 2. Testing statistics for top down (TD) network [5]

B. Geometry-based method

The approach described by Oliveira *et al.* [6] is based on the use of a LiDAR and an IMU to gather geometric data of the surrounding terrain to build a cost map. The idea is to combine inertial and LiDAR data with a neural network to compute the cost of a previously unvisited terrain. The method is divised in three main steps:

- Three-dimensional mapping and localization
- Navigation cost estimation using inertial data
- Map augmentation through deep learning

Three-dimensional mapping and localization: For the first step, a C-SLAM algorithm [7] is used to build a 3D map of the terrain with the point clouds provided by the LiDAR. The point clouds is aligned on the trajectory of the UGV thanks to the IMU and is approximated to a 3D grid.

Navigation cost estimation using inertial data: The LiDAR provides information about the geometry of the environment surrounding the robot, while the IMU supplies inertial data for determining the robot's position. IMU data is particularly useful for characterizing terrain roughness, enabling the assignment of a traversability cost. These datasets can be fused by synchronizing them, allowing inertial data to be estimated from LiDAR measurements.

Map augmentation through deep learning: A model is trained to learn the relationship between navigation cost, inertial data, and point cloud information. This process results in the creation of a continuous 3D map representing navigation costs. A DCNN is used to fuse the data, taking as input the discretized data from a 3D point cloud mapped onto a 2D grid of 60 x 40 cells.

Th method was tested in real conditions on an UGV equipped with a Velodyne VLP-16 LiDAR and a Xsens MTi-30 IMU. The classification model predicted the terrain roughness with a mean accuracy of 95.4% with a standard deviation of 0.006 at a speed of 0.6m/s.

C. Hybrid methods

1) Method 1: The method proposed by Leung *et al.* [3] relies on the use of an RGB camera and a LiDAR to take into account both visual and geometric information. The geometric data collected enables the creation of an elevation grid projected onto a 2D cost map, while visual data are processed by a DCNN. This system relies on three main modules operating asynchronously. The first module performs semantic segmentation on image retrieved by the camera to identify terrain types, then projects the terrain types onto a 2D map. The second module uses the LiDAR to generate a 2.5D elevation map centered on the robot. Finally, the third module combines these informations to produce a 2D traversability map.

The generated map is structured into a grid where each cell is associated with a global 3D coordinate system. It contains information about height and traversability costs. The generation of the traversability map is triggered periodically using a timer.

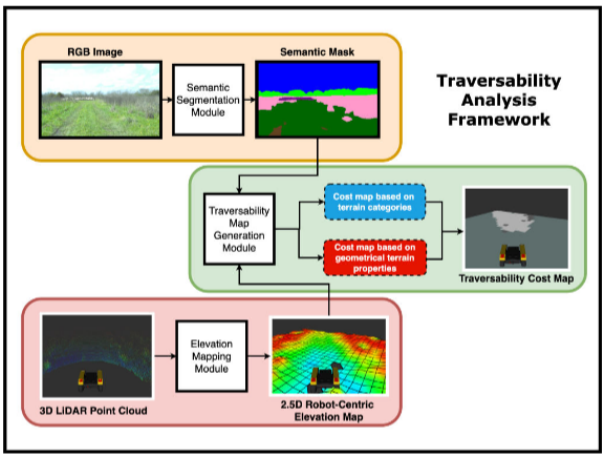


Fig. 4. Overview of the method proposed by Leung *et al.* [3]

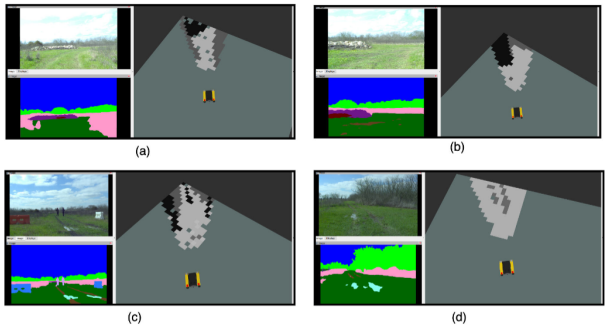


Fig. 5. Visualization of the traversability cost map generated by the framework proposed by Leung *et al.* [3]. In each figure, the image on the top left corner is the RGB image captured by the camera. The image below is the semantic mask output from the semantic segmentation module. The right-hand side of the figure depicts the visualization of the terrain traversability cost map. Grids in deeper colour represent less traversable terrain.

For semantic segmentation, each pixel of the RGB images is assigned a label corresponding to a terrain type. The images, initially at a resolution of 1920x1080, are resized to 640x400 to feed the neural network. Two network architectures were tested: Gated-SCNN and ERFNet. Although ERFNet is slightly less accurate, it was selected for its speed. The training was conducted using the RELLIS-3D dataset, which consists of 5957 images divided into 18 classes.

The elevation mapping is based on a library developed by Fankhauser [8]. Using the 3D point cloud provided by a LiDAR and information about the robot's movement, a 2.5D elevation map is generated. Each cell in the grid contains height data and is positioned in a global coordinate system, facilitating its fusion with semantic data.

The generation of the traversability map incorporates two cost sources. The first source is based on terrain types identified by semantic segmentation. Image pixel data is projected from the 2D camera frame to the global 3D frame, then onto a 2D map centered on the robot. These projections use rotation and translation matrices as well as the Pinhole Camera model. Costs are assigned based on the terrain class associated with each cell in the map. The second cost source is calculated from the terrain's geometric properties, such as slope, roughness, and obstacle height. These costs account for weights assigned to each parameter and critical values related to the vehicle's capabilities. The two cost maps are eventually fused by addition, with specific weighting factors applied to each contribution.

The method was validated by driving a Warthog platform in different off-road environments. The results are presented in Fig. 5 and illustrates good performance. For example, in Fig. 5(a), the robot's sides are covered with bushes, which are correctly identified as obstacles. Indeed, the left and ride side of the cost map are darker, indicating that the terrain is less traversable.

2) Method 2: Similarly to [3], the methods developped by Breitfuss *et al.* [2] proposes to combine geometric and visual data to analyze terrain traversability in real time. This method has been developed with an emphasis on its ability to remain effective for all types of vehicles, especially heavy ones.

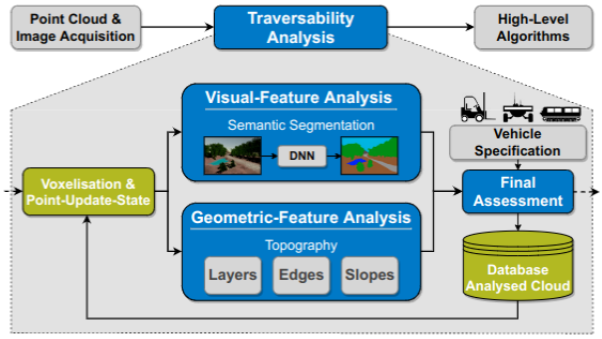


Fig. 6. Overview of the method proposed by Breitfuss *et al.* [2]

Two analyses are performed simultaneously: a 3D point cloud from a LiDAR is used to extract geometric data, and semantic segmentation is applied to images to identify terrain types. Traversability is treated as a function in a multidimensional space that considers the environment's topography (geometric features), terrain type (visual features), and the specifications of the UGV.

The core algorithm involves three main steps. First, an initial estimate of traversability is generated using both geometric and visual data. Second, this estimate is refined by incorporating the vehicle's characteristics. Finally, an iterative data-fusion process combines raw data with point clouds representing traversability calculated in previous iterations. Real-time constraints are a critical challenge, especially when processing large point clouds. This is addressed using voxelization and the Point-Update-State (PUS) principle.

Voxelization overlays a 3D grid of cubes onto the point cloud, merging all points within a cube into a single averaged value. During data fusion, four cases are considered: new points (unexplored areas), old points (previously explored but not revisited areas), overlapping new and old points (unchanged if close, updated if significantly different), and

unchanged points marked for exclusion from further computation using PUS.

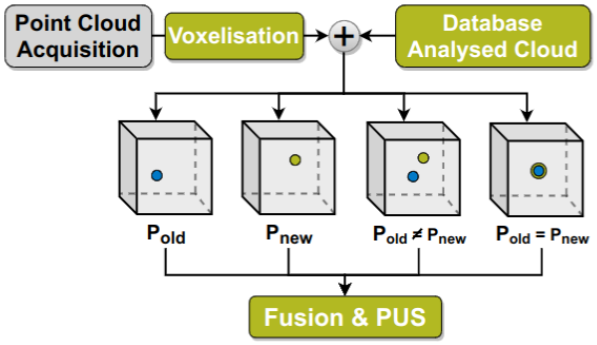


Fig. 7. Data fusion procedure: overlapping new inputs with the previously analysed cloud leads to 1 of 4 cases [2]

Geometric data is processed in three stages. First, overhanging objects are identified by assigning points to primary or secondary layers based on their height relative to the vehicle. Successive points with height differences smaller than the vehicle's height are grouped into the same layer. This process leverages the voxel grid structure, simplifying the classification. Second, edges and obstacle heights are detected by projecting the 3D space onto a 2D grayscale image and applying contour detection (Canny). Obstacle heights are calculated as the height difference between the upper and lower boundary pixels of each contour. Depending on the vehicle's specifications, certain obstacles are marked as impassable and excluded from further analysis. The edge height information is reprojected onto the 3D point cloud. Finally, slope is evaluated for all remaining points. This is achieved using Principal Component Analysis (PCA), which calculates the surface normal at each point by analyzing its local neighborhood within a spherical region.

Visual data is processed using a convolutional neural network based on the UNet architecture, selected for its high precision in semantic segmentation. This allows semantic classes to be accurately projected onto the 3D point cloud. The RUGB dataset, which includes 25 classes representing various objects and terrain types, was used for training.

The final traversability calculation at each point combines terrain type from semantic segmentation with geometric data (slope and obstacle height). These are compared to weights determined by the vehicle's characteristics. The result is a point cloud where each point's value ranges from 0 (impassable) to 1 (ideal terrain). This traversability map can subsequently be used for tasks such as path planning.

The method was tested both in a simulated environment and in real conditions. The results are presented in Fig. 8 and show that the method is able to adapt to different vehicle specifications and terrain types.

III. Methodology

To compare the different methods presented in the previous section, this paper is based on three main criteria: execution speed, accuracy, and robustness. Execution speed refers to the time required to run the method as well as the computational power needed for its operation. This criterion is particularly important for real-time applications where system responsiveness is critical. Accuracy, on the other hand, reflects the method's ability to consider a wide range of parameters to evaluate terrain traversability as precisely as possible. Finally, robustness measures the method's ability to adapt to different environments, weather conditions, vehicle types, and more.

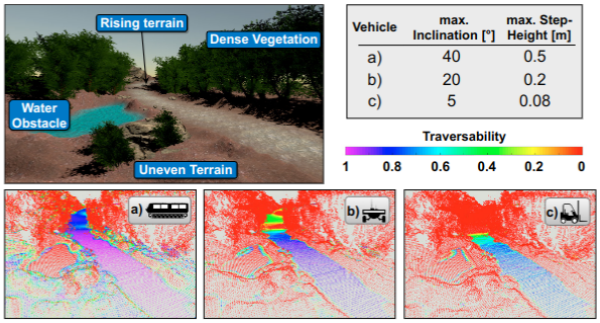


Fig. 8. Results of a simulated outdoor environment for different vehicle specifications depicted in the table [2]

IV. Comparison Results

A. Vision-based method [5]

The method proposed by Drews *et al.* [5] is particularly efficient in terms of execution speed. Indeed, the use of raw images to feed the DCNN makes the method very lightweight. This results in a processing frequency of 40Hz on an Nvidia GTX750Ti, which is very good for real-time applications. Moreover, the capacity of the DCNN to anticipate the track allows the UGV to drive at high speeds, which is a significant advantage for agressive driving. However, the cost map is only based on visual data, thus limiting the accuracy of the terrain traversability assessment. Moreover, the method is well-suited for circuits or dirt tracks, but may not be as effective in more complex environments and does not take into account the characteristics of the vehicule. Therefore, the method is not very robust. In conclusion, [5] is an interesting and very promising method delivering good results in real-world scenarios, but adapted to a very specific use case.

B. Geometry-based method [6]

The method proposed by Oliveira *et al.* [6] is more computationally intensive than the previous one. The best results are obtained for a speed of 0.6m/s, which is relatively slow. Being solely based on geometric data, the method lacks in accuracy. Indeed, the terrain type is not considered, which can be crucial for the traversability assessment. However, the training of the model is done based on the data of an IMU installed on the robot. Thus, the model is automatically adapted to the handling characteristics of the vehicle, which is a significant advantage. The only downside is that the DCNN must be retrained for each new vehicle. Moreover, the LiDAR gives

a 360° view of the environment, allowing the construction of a cost map all around the robot. [6] is better suited to unstructured environments than [5], while being more versatile but less responsive.

C. Hybrid method 1 [3]

The hybrid aspect of the method proposed by Leung *et al.* [3] leads necessarily to a more complex and computationally intensive method. To optimize calculations, the 2.5D elevation map is simplified by superimposing a grid and is calculated only for the line-of-sight portion of the map. The combination of both visual and geometric data allows for a more accurate terrain traversability assessment. The DCNN is trained to classify 18 different terrain types and geometric data includes slope, roughness and step-height. The method is very robust being able to adapt to different environments as well as different vehicle types by taking into account the vehicle's characteristics in the cost map computation. Overall, this method is more comprehensive than the two previous methods, but also more computationally (and materially) expensive. The author expresses the intention to optimize the DCNN to make it more efficient, and to increase the number of classes in the learning set.

D. Hybrid method 2 [2]

The optimization process in the method proposed by Breitfuss *et al.* [2] is based on the voxelization of the 3D point cloud and the use of PUS principle. This allows to reduce the number of points to be processed and thus to speed up the calculations. Nonetheless, the method remains computationally intensive due to the processing of both visual and geometric data. Like [3], the method is very accurate, taking into account the slope, step-height, overhanging objects and terrain types. The DCNN is able to classify 25 different classes and is therefore more accurate than [3]. The method is also highly robust and particularly adaptable to different vehicle types. According to the author, the efficiency of the method is independent of the vehicle type and is well suited for heavy vehicles. Generally, this method is very similar to [3], but has an accent on the adaptability to different vehicle characteristics.

V. Conclusion

The comparison of the four methods presented in this paper shows that each method has its own strengths and weaknesses. The vision-based method proposed by Drews *et al.* [5] is particularly efficient in terms of execution speed and is well suited for agressive driving. However, the method lacks in accuracy and robustness. The geometry-based method proposed by Oliveira *et al.* [6] is more computationally intensive and slower, but is more versatile and automatically adapts to the vehicle handling characteristics. The hybrid methods proposed by Leung *et al.* [3] and Breitfuss *et al.* [2] are more comprehensive and accurate, but also more computationally expensive. The method proposed by Breitfuss *et al.* [2] is particularly well suited for heavy vehicles.

The main limitation of terrain traversability assessment methods is the execution speed and the computational power required. Indeed, the more accurate and comprehensive the method, the more computationally intensive it is. Another important limitation is the use of camera and LiDAR data during rainy or foggy weather. Indeed, the performance of the methods presented in this paper hugely depends on the quality of the data collected. During bad weather conditions, the data collected by camera and LiDAR may be of poor quality, which can lead to inaccurate terrain traversability assessment [9], [10]. Unfortunately, the methods presented in this paper were not tested in bad weather conditions, so it is difficult to assess their performance in such conditions. Some papers are starting to address this issue such as [11] for LiDAR data.

References

[1] P. Papadakis, "Terrain traversability analysis methods for unmanned ground vehicles: A survey," *Engineering Applications of Artificial Intelligence*, vol. 26, no. 4, pp. 1373–1385, 2013.

[2] M. Breitfuß, M. Schöberl, and J. Fottner, "Safety through perception: Multi-modal traversability analysis in rough outdoor environments," *IFAC-PapersOnLine*, vol. 54, no. 1, pp. 223–228, 2021, 17th IFAC Symposium on Information Control Problems in Manufacturing INCOM 2021.

[3] T. H. Y. Leung, D. Ignatyev, and A. Zolotas, "Hybrid terrain traversability analysis in off-road environments," in *2022 8th International Conference on Automation, Robotics and Applications (ICARA)*, 2022, pp. 50–56.

[4] S. Beycimen, D. Ignatyev, and A. Zolotas, "A comprehensive survey of unmanned ground vehicle terrain traversability for unstructured environments and sensor technology insights," *Engineering Science and Technology, an International Journal*, vol. 47, p. 101457, 2023.

[5] P. Drews, G. Williams, B. Goldfain, E. A. Theodorou, and J. M. Rehg, "Aggressive deep driving: Model predictive control with a cnn cost model," *ArXiv*, vol. abs/1707.05303, 2017.

[6] F. Oliveira, M. Campos, and D. Macharet, "Three-dimensional mapping with augmented navigation cost through deep learning," 11 2020, pp. 97–108.

[7] D. Helmick, A. Angelova, and L. Matthies, "Terrain adaptive navigation for planetary rovers," *Journal of Field Robotics*, vol. 26, no. 4, pp. 391–410, 2009.

[8] P. Fankhauser, M. Bloesch, and M. Hutter, "Probabilistic terrain mapping for mobile robots with uncertain localization," *IEEE Robotics and Automation Letters*, vol. 3, pp. 3019–3026, 05 2018.

[9] A. Filgueira, H. González-Jorge, S. Lagüela, L. Díaz-Vilariño, and P. Arias, "Quantifying the influence of rain in lidar performance," *Measurement*, vol. 95, pp. 143–148, 2017.

[10] T. Brophy, D. Mullins, A. Parsi, J. Horgan, E. Ward, P. Denny, C. Eising, B. Deegan, M. Glavin, and E. Jones, "A review of the impact of rain on camera-based perception in automated driving systems," *IEEE Access*, 06 2023.

[11] A. Piroli, V. Dallabetta, J. Kopp, M. Walessa, D. Meissner, and K. Dietmayer, "Energy-based detection of adverse weather effects in lidar data," *IEEE Robotics and Automation Letters*, vol. 8, no. 7, pp. 4322–4329, Jul. 2023.

Adapting Object Detection for water column images from multi-beam sounder

Laura Jouvet
3rd year engineering student
ENSTA Bretagne
29200, Brest, FRANCE
laura.jouvet@ensta.fr

Abstract

Object detection has become a crucial tool in various fields, including autonomous driving, surveillance, and medical imaging. State-of-the-art deep learning models achieve remarkable accuracy in detecting objects in natural images but it has a hard time detecting objects in other types of images such as those with low contrast or high noise levels. Some researchers succeeded in obtaining promising results for medical images by employing fine-tuning techniques but there is still a lack of research with sonar and multi-beam sounder images. This paper explores the challenges and potential solutions for adapting object detection models to water column images, aiming to bridge the gap in research and improve detection capabilities in this domain. First, we create our own dataset with multi-beam sounder images and we evaluate some neural networks on it and then we train some of them on this dataset to obtain significant improvement in the performance of object detection with water column images.

Keywords: ***Object detection, Multi-beam Sounder Image, Neural network***

1 Introduction

Object detection has become a key component in numerous applications, ranging from autonomous driving and security surveillance to medical diagnostics where it helps in identifying key objects or regions of interest (ROIs) from complex data. Recent advancements in deep learning have significantly improved the accuracy and robustness of object detection models, particularly for natural images.

Despite its success in natural images, these neural networks struggles with non-natural imagery, particularly in scenarios where images have low contrast, high noise, or unconventional structures. This limitation becomes evident in specialized domains such as medical imaging, where neural networks' performance often degrades. Similarly, in sonar and multi-beam sounder images, which are used to map underwater environments, their ability to perform object detection is significantly compromised. These images present their own set of challenges, including noise, scattering effects, low resolution, complex patterns and varying acoustic properties of the water column that differ vastly from the more straightforward visual patterns of natural images. These factors make it difficult for conventional neural networks, which are primarily trained on natural image datasets, to generalize effectively to this type of data. Although recent studies have demonstrated progress in adapting object detection models for medical imaging using fine-tuning techniques, research on sonar-based object detection remains scarce. Addressing this gap is crucial for applications such as underwater exploration, marine biology, and submerged infrastructure monitoring.

This paper aims to investigate the adaptation of deep learning-based object detection models for water column images obtained from multi-beam sounders. First, we create our own dataset with multi-beam sounder images because such a dataset does not exist. Then, we evaluate existing neural networks on this dataset to assess their limitations. Finally, we fine-tune selected models on our own dataset to improve detection performance in this challenging domain.

By evaluating neural networks performance and training them with our own dataset, this study contributes to enhancing object detection capabilities for specialized and challenging domains such as underwater imaging. We hope that this work will pave the way for more efficient and accurate analysis of sonar data and other non-traditional images.

2 Related works

2.1 Ever More Efficient Neural Networks for Object Detection

Object detection has seen significant advancements with deep learning, particularly through models like YOLO (You Only Look Once) and Faster R-CNN. YOLO is known for its speed, as it detects objects in real-time by predicting bounding boxes and class labels in a single pass [15]. Recent versions, like YOLOv4 and YOLOv5, have improved accuracy and performance by integrating advanced techniques [1] [8]. Faster R-CNN,

on the other hand, follows a two-stage approach, first generating region proposals with a Region Proposal Network (RPN) and then classifying them [16]. Although slower than YOLO, Faster R-CNN offers higher accuracy, especially in complex detection tasks, and has seen improvements through methods like Feature Pyramid Networks (FPN) [4] [12]. Both architectures continue to push the boundaries of object detection efficiency and accuracy.

2.2 Sonar images

These architectures have been adapted and improved across various domains, but object detection in specific contexts like sonar or underwater images remains challenging due to the low image quality, high noise levels, and complex features.

2.2.1 Side Scan Sonar (SSS) images

The growing marine economy and shipping industry demand improved maritime safety, particularly in detecting underwater targets like shipwrecks and submerged containers. Side-scan sonar (SSS) plays a crucial role in this, providing high-resolution images widely used in maritime search and rescue [3, 11, 21].

Traditionally, SSS object detection relied on manual interpretation, prone to human error [13]. Machine learning (ML) methods leverage artificial features like texture and edges but face challenges in feature engineering and compatibility between extractors and classifiers [6, 7, 19, 22, 23]. Deep learning (DL) offers a more automated approach, yet struggles with limited SSS training data and adapting optical-based models [5, 9, 14].

To overcome these challenges, researchers have explored data augmentation, Generative Adversarial Networks (GANs), and deep learning models such as CNNs and YOLO [2, 5, 10, 18]. However, real-time detection remains difficult due to noise and sparse target features in SSS images. Yu et al. addressed this by integrating a transformer module with YOLOv5s, improving recognition and efficiency through a novel down-sampling strategy [20]. This paper further explores these enhancements, detailing preprocessing, sampling, recognition, and target localization.

2.2.2 Multi-beam sounder (MBES) images

The multi-beam echo sounder is also a valuable tool for detecting objects within the water column, such as fish, marine debris, or other submerged structures. Unlike side-scan sonar, which primarily focuses on seafloor imaging, MBES provides detailed acoustic reflections from the entire water column, enabling the identification of dynamic and suspended elements. This capability makes MBES essential for applications in fisheries research, environmental monitoring, and underwater exploration. However, the detection and classification of objects in the water column remain challenging due to noise, signal attenuation, and the complexity of acoustic scattering. Currently, there is no existing dataset of images from multi-beam echo sounders, and no known object detector has been thoroughly tested or trained on this type of imagery while it would be highly useful to detect objects in real time using such a sounder.

3 Method

In this section, we will present our method for adapting existing object detectors to our images from a multi-beam sounder. First, we created a dataset, then we selected an object detector, and finally, we optimized it for our dataset.

3.1 Creation of a dataset

To create our dataset, we conducted multibeam sonar surveys on lake of Guerlédan to generate training, validation, and test sets for object detectors. This lake is particularly interesting because there are fish, trees, houses, and locks at the bottom. To avoid overlap by using the same images in different sets, we conducted surveys in distinct areas. We then converted this data into images that could be used by object detectors. Finally, we had to annotate all these data to train the object detectors thanks to the software Labelme.

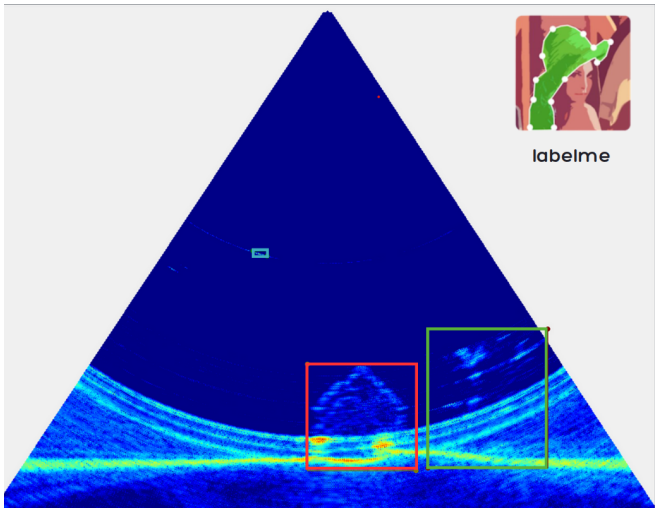


Figure 1: Labelisation on Labelme

In total, we annotated 33.541 images using the Labelme software. Among these images, 54% contained no objects, and the following elements were present:

Fish	3 649
Building	7 266
Tree	6 097

The number of images per class was sufficient and fairly balanced across each set, so we did not need to perform data augmentation.

3.2 Selection of the object detector

To select a detector suitable for our project, we compared about ten neural networks by researching the state of the art and identified several criteria such as output data, processing speeds, memory consumption, and different accuracy levels [17]. The main aspect we focused on was the ability to use the detector in real time since, ultimately, the final goal of our project is to detect objects in real time beneath a boat. To achieve this, we decided to choose a detector capable of processing the entire image to perform detection in a single step rather than in two steps (region separation followed by detection).

The most efficient detector using single-step detection (Single-Shot Detection) is YOLO (You Only Look Once). There are 12 versions of YOLO released between 2015 and 2025. Versions 6 to 8 allow real-time detection, and version 8 is highly efficient, so we selected it. Additionally, for each YOLO version, different models exist, ranging from the smallest N to the largest X. The main difference between these models lies in the number of parameters. The N model has the fewest parameters, making it faster, lighter, and ideal for embedded and real-time systems. Therefore, we will prioritized it, especially as we trained the networks on our personal computers.

3.3 Optimization of the object detector

To optimize YOLOv8, we first had to compare its performance on our images between the pre-trained and non-pre-trained models, then adjust its hyperparameters. Next, we made several adjustments to our datasets to determine which organization led to the best training results. For example, we experimented with the number of empty images, image saturation, and image quality.

4 Experiments and results

We first evaluated the performance of YOLOv8 on our water column images from multibeam sonar. Next, we optimized its parameters, and finally, we modified our dataset to further improve the results.

4.1 Evaluating YOLOv8 on our dataset

In this section, we first apply the original YOLOv8 for the multi-beam sounder images detection task. The model learns well on the training set, with loss functions converging towards 0. Furthermore, the evaluation loss functions converge to 2.2±0.1, but they do not diverge; therefore, we do not have actual overfitting. The mAP50 is 0.46 which is pretty good just like the precision (0.52) and the recall (0.43). However, it can be observed in the confusion matrix that, while the model performs well in locating and identifying trees, it is less effective when it comes to fish and even more so for buildings. Indeed, it is noticeable that when the image is empty, the model has a strong tendency to detect a building or a fish.

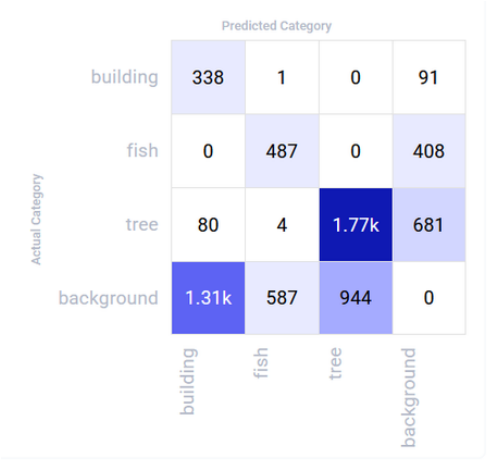


Figure 2: Confusion matrix for untrained YOLOv8

It is still important to note that the model has very rarely confused different classes with each other. Thus, the majority of our areas for improvement and experimentation will focus on handling empty images as well as improving the annotations of our buildings and fish.

4.2 Optimizing YOLOv8 for our dataset

To optimize the performance of YOLOv8 on our dataset, we had to adjust some of its parameters. One of the options offered by YOLO is the use of pre-trained models through fine-tuning. Fine-tuning is a deep learning technique that involves adjusting a pre-trained model on a new, specific dataset. This approach allows for optimal performance without the need to train a model from scratch, significantly reducing computational time and resources. To determine if this method is relevant, we conducted two trainings with equivalent parameters and a dataset, with the number of epochs and batch size set to 50 and 32, respectively. We observed slightly better performance when using a pre-trained model, both in terms of precision metrics and confusion matrix. We also obtained similar loss curves for classification, with the evaluation curve of the pre-trained model converging to a lower value. Therefore, we can conclude that using a pre-trained YOLO model leads to better performance with our dataset, and we decided to keep this model.

Next, we adjusted the hyperparameters of this model to improve its training on our dataset. The hyperparameters include the number of epochs, batch size, and input image size. Regarding the input image size, the optimal recommended size for YOLO is 640, so that is what we used. Regarding the number of epochs, too few epochs may lead to underfitting, while too many epochs can cause overfitting and a longer training duration. For the batch size, it is also necessary to find a balance, as a larger batch size shortens the training time but requires more RAM. Therefore, we had to choose values that were low enough to ensure an acceptable

training time while being high enough for the results to be usable and analyzable.

In the following graph, we can observe the average precision as a function of the number of epochs.

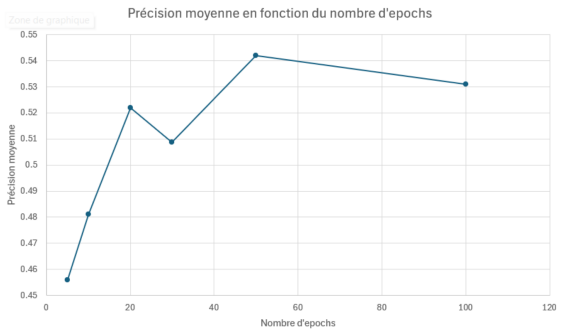


Figure 3: Average precision based on the number of epochs

We can see that the best performance is achieved with 50 epochs. Moreover, with this number of epochs, the loss function converges properly, and the training duration remains acceptable (3 hours and 22 minutes). As for the batch size, the RAM of our GPUs limited it to 32, we therefore performed three trainings with batch sizes of 8, 16, and 32, respectively. Below are the durations and average precisions of these trainings:

Batch size	8	16	32
Training duration	05:36:19	3:45:35	03:22:25
Average precision	0.5251	0.5204	0.542

We can observe that a batch size of 32 provides the best performance while reducing the training duration.

For the rest of our project, we therefore set the number of epochs to 50 and the batch size to 32.

4.3 Optimizing our dataset

After selecting the model and its parameters to achieve the best training performance, we needed to choose our datasets, as the type of data has a significant impact on a model's performance. In this section, we evaluate the influence of certain parameters on the training performance of the selected model.

4.3.1 Percentage of empty images

First, we analyzed the percentage of empty images present in the dataset. To ensure proper generalization, a dataset should include some empty images, but not too many. It is generally recommended to use between 20% and 30% of empty images, whereas our datasets contained 54%. We conducted three tests to evaluate the impact of this percentage: one with all empty images, one after removing all empty images, and one keeping only 20% of the original empty images. By analyzing the results obtained on the same dataset, we observed the importance of the percentage of empty images.

% of empty images retained	0	20	100
Precision	0.61	0.59	0.54
mAP50	0.48	0.46	0.44

When keeping all empty images, the model is trained on too many non-informative samples, which reduces overall precision and weakens object detection. Conversely, removing all empty images significantly increases precision and improves object detection. However, to maintain realism, it is essential to keep a certain number of empty images to ensure the model is exposed to all types of images. Thus, despite slightly lower results, we decided to retain 20% of the empty images in our dataset.

4.3.2 Image saturation

Next, we experimented with image saturation. Indeed, when converting water column data into images, we initially set the saturation limits randomly. Increasing the lower saturation bound helps to homogenize surfaces with low backscatter (BS), such as water. Conversely, decreasing this bound introduces noise in the water, as low BS variations are no longer limited. On the other hand, reducing the upper saturation bound causes trees to appear red, similar to the background, as high BS variations become homogenized. Conversely, increasing this bound allows for greater color diversity among objects with high BS values.

To analyze the impact of these adjustments, we compared training results using two different saturation settings: [-50, 40] and [-64, 10].

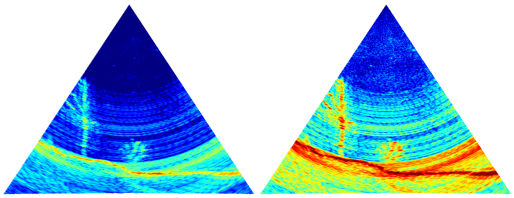


Figure 4: Different saturations

We observed very similar performance between the two, although images with saturation limits set to [-50, 40] yielded slightly better results. The mAP50, confusion matrix, and model precision were all higher for this configuration. Additionally, the loss function curves were smoother, indicating better consistency in training. However, we also noted that the second model with [-64, 10] confused trees and buildings less, likely due to the higher lower saturation bound.

Further testing would be required to determine the optimal saturation limits for this specific application. However, within the scope of this project, we chose to use the [-50, 40] settings.

4.3.3 Difficult images management

After adjusting the percentage of empty images and saturation, our model still struggled to correctly detect

buildings and fish. We then decided to focus on difficult images, specifically those where we were unsure about the annotation. As a result, we modified our building annotations by keeping only those that were clearly identifiable as buildings and ensured that no fish had been left unannotated.

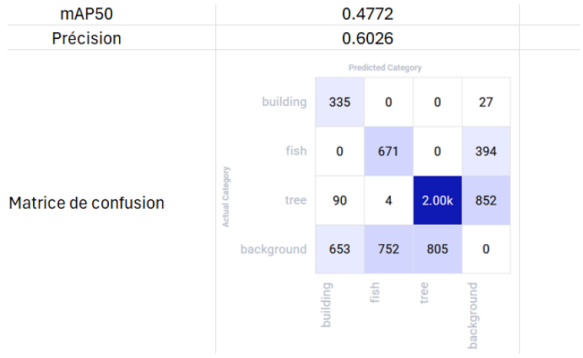


Figure 5: Results after the improvement of the annotations

Although the results are not perfect, we observed a significant improvement, particularly in terms of average precision and the detection of buildings and fish. This highlights the importance of having consistent and precise annotations, which is challenging for a dataset annotated manually by different people. Indeed, these annotations are subject to human errors as well as subjective criteria.

However, we noticed that the model still frequently confuses the background with buildings. This issue arises due to the unique nature of the objects in our images. In fact, most of the annotated buildings are locks, which are excavated into the lakebed. The only distinguishing feature between them and the background is the presence of right angles. Additionally, the variation in image quality within the same survey could further explain the complexity of training for these detections.

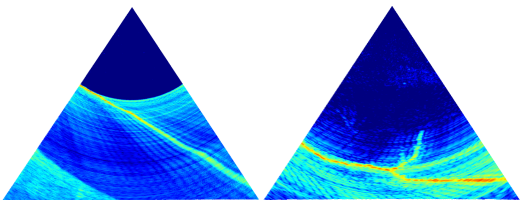


Figure 6: Different images from the same swath

Here, we can observe discrepancies in the presence of noise as well as in the color of the background, even though both images come from the same swath. The composition of the lakebed also adds another layer of difficulty compared to more conventional images. A substrate made of fine particles, such as sand or silt, tends to absorb more acoustic waves than a bottom composed of concrete or rocks.

Some rocks appear with a similar color to buildings and sometimes even a shape resembling locks or lock houses, which could explain this confusion between buildings and the background.

4.4 Testing our final model

We finally tested our final model with pre-trained YOLOv8, an image size of 640, 50 epochs, and a batch size of 32. For our dataset, we used 20% of the total number of empty images and saturation bounds of [-50, 40]. Finally, we used the images we had re-annotated, removing those that were excessively noisy, and we obtained the following results:

Figure 7: Final results

The confusion matrix is much better than the one we had for the untrained model. Indeed, tree detection still performs just as well, while fish and building detection has significantly improved. Moreover, we obtain a mAP of 0.86 and a precision of 0.84, which are very good results and much better than those we had with the untrained YOLOv8.

5 Conclusion

In this study, we propose the adaptation of YOLOv8 for water column images from multibeam sonar. Directly applying YOLOv8 to images from multibeam sonar presents a challenge due to the significant gap between the domain of these images and the training samples typically used for YOLO. To address this issue, we adjusted various model parameters and dataset configurations, and YOLOv8 demonstrates significant improvements in performance.

These performances could be further enhanced by improving image annotations and providing access to neighboring images of the one being processed by YOLO, helping it detect objects in challenging images.

Acknowledgments

This work was realized with Simon Barbarit–Gaboriau, Nicolas Cloarec–Riouat and Arthur Coron with the help of Tyméa Perret and Hughes Moreau.

References

[1] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy for real-time object detection. *arXiv preprint arXiv:2004.10934*, 2020.

[2] N. Bore and J. Folkesson. Modeling and simulation of sidescan using conditional generative adversarial network. *IEEE Journal of Oceanic Engineering*, 46:195–205, 2020.

[3] T. Celik and T. Tjahjadi. A novel method for sidescan sonar image segmentation. *IEEE Journal of Oceanic Engineering*, 36:186–194, 2011.

[4] Jifeng Dai, Kaiming He, and Jian Sun. Instance-aware semantic segmentation via multi-task network cascades. In *CVPR*, 2016.

[5] D. Einsidler, M. Dhanak, and P. Beaujean. A deep learning approach to target recognition in side-scan sonar imagery. In *Proceedings of the MTS/IEEE Charleston OCEANS Conference*, pages 1–4.

[6] E. Fakiris, G. Papatheodorou, M. Geraga, and G. Ferentinos. An automatic target detection algorithm for swath sonar backscatter imagery, using image texture and independent component analysis. *Remote Sensing*, 8:373, 2016.

[7] L. Guillaume and G. Sylvain. Unsupervised extraction of underwater regions of interest in sidescan sonar imagery. *Journal of Oceanic Engineering*, 15:95–108, 2020.

[8] Glenn Jocher et al. Yolov5. `https://github.com/ultralytics/yolov5`, 2020.

[9] J. Kim, J. Choi, H. Kwon, R. Oh, and S. Son. The application of convolutional neural networks for automatic detection of underwater object in side scan sonar images. *Journal of the Acoustical Society of Korea*, 37:118–128, 2018.

[10] S. Lee, B. Park, and A. Kim. Deep learning based object detection via style-transferred underwater sonar images. In *Proceedings of the 12th IFAC Conference on Control Applications in Marine Systems, Robotics, and Vehicles (CAMS 2019)*, pages 152–155.

[11] S. Li, J. Zhao, H. Zhang, Z. Bi, and S. Qu. A novel horizon picking method on sub-bottom profiler sonar images. *Remote Sensing*, 12:3322, 2020.

[12] Tsung-Yi Lin, Piotr Dollár, Ross B. Girshick, Kaiming He, and Bharath Hariharan. Feature pyramid networks for object detection. In *CVPR*, 2017.

[13] D. Neupane and J. Seok. A review on deep learning-based approaches for automatic sonar target recognition. *Electronics*, 9:1972, 2020.

[14] H. Nguyen, E. Lee, and S. Lee. Study on the classification performance of underwater sonar image classification based on convolutional neural networks for detecting a submerged human body. *Sensors*, 20:94, 2019.

[15] Joseph Redmon, Santosh Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CVPR*, 2016.

[16] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NIPS*, 2015.

[17] Saagie. Qu'est-ce que la détection d'objet ?, 2023. Consulté le 20 décembre 2024.

[18] Y. Steiniger, D. Kraus, and T. Meisen. Generating synthetic sidescan sonar snippets using transfer-learning in generative adversarial networks. *Journal of Marine Science and Engineering*, 9:239, 2021.

[19] W. Xiao, J. Zhao, B. Zhu, T. Jiang, and T. Qin. A side scan sonar image target detection algorithm based on a neutrosophic set and diffusion maps. *Remote Sensing*, 10:295, 2018.

[20] Yongcan Yu, Jianhu Zhao, Quanhua Gong, Chao Huang, Gen Zheng, and Jinye Ma. Real-time underwater maritime object detection in side-scan sonar images based on transformer-yolov5. *Remote Sensing*, 13:3555, 2021.

[21] G. Zheng, H. Zhang, Y. Li, and J. Zhao. A universal automatic bottom tracking method of side scan sonar data based on semantic segmentation. *Remote Sensing*, 13:1945, 2021.

[22] L. Zheng and K. Tian. Detection of small objects in sidescan sonar images based on pohmt and tsallis entropy. *Signal Processing*, 142:168–177, 2017.

[23] B. Zhu, X. Wang, Z. Chu, Y. Yang, and J. Shi. Active learning for recognition of shipwreck target in side-scan sonar image. *Remote Sensing*, 11:243, 2019.

Localisation de robot - comparaison des méthodes probabilistes et ensemblistes

Ylona Provot

***Abstract*—La localisation est un des problèmes majeurs en robotique mobile. Ce travail vise à comparer les méthodes ensemblistes et probabilistes , c'est à dire une méthode de localisation par intervalles ou bien grâce à un filtre de Kalman. Cet article s'interesse à un problème de localisation dans un environnement constitué de 3 balises auxquelles le robot mesure sa distance. La résolution d'un tel problème grâce aux intervalles puis à un filtre de Kalman déterminera laquelle des deux méthodes est la plus efficace dans ce contexte précis en tenant compte des incertitudes des capteurs.**

***Index Terms*—Localisation, filtre de Kalman, analyse par intervales.**

I. Introduction

Lors du contrôle d'un système, nous supposons souvent que le vecteur d'état du système est entièrement connu. Ce n'est pourtant pas le cas en réalité car ce vecteur est estimé à partir des capteurs soumis à leurs incertitudes de mesures. Il s'agit donc de filtrer ces mesures afin d'obtenir un vecteur d'état le plus fiable possible.

II. Modélisation du problème

Soit un robot terrestre modélisé par les équations d'état d'une voiture de Dubins :

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{pmatrix} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{v} \end{pmatrix} = \begin{pmatrix} v\cos\theta \\ v\sin\theta \\ u_0 \\ u_1 \end{pmatrix}$$

avec :

- x, y la position du robot
- θ son cap
- v sa vitesse d'avance

Le robot est entouré de 3 landmarks de positions connues et capable de mesurer sa distance à ces 3 landmarks.

On considère que le robot connait son cap à tout instant avec une bonne précision.

III. Contrôle

Appliquons un contrôle par feedback pour que le robot dessine une trajectoire circulaire de rayon R, notée W

$$W = \begin{bmatrix} R\cos(t) \\ R\sin(t) \end{bmatrix}$$

D'après l'équation d'état du système, nous avons la relation :

$$\begin{cases} \ddot{x} = u_0\cos\theta - u_1\sin\theta \\ \ddot{y} = u_0\sin\theta + u_1\cos\theta \end{cases}$$

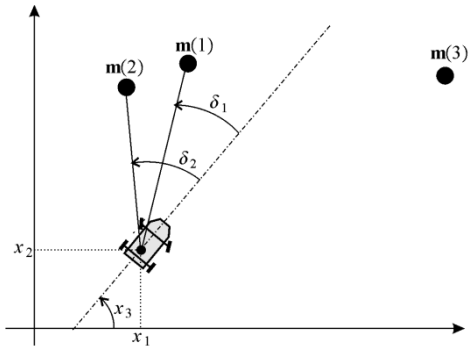


Fig. 1. Représentation du robot et des landmarks

C'est à dire

$$\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \begin{pmatrix} -\sin\theta & \cos\theta \\ \cos\Theta & \sin\Theta \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \end{pmatrix}$$

En prenant comme entrée

$$\begin{pmatrix} u_0 \\ u_1 \end{pmatrix} = \begin{pmatrix} -v\sin\theta & \cos\theta \\ v\cos\theta & \sin\theta \end{pmatrix}^{-1} \cdot \begin{pmatrix} v_0 \\ v_1 \end{pmatrix}$$

où (v_0, v_1) est le nouveau vecteur d'entrées on obtient le système linéaire:

$$\begin{pmatrix} \ddot{x} \\ \ddot{y} \end{pmatrix} = \begin{pmatrix} v_0 \\ v_1 \end{pmatrix}.$$

Plaçons les pôles en -1.

$$\begin{cases} v_0 = \ddot{w}_x - x + 2(\dot{w}_x - \dot{x}) + \ddot{w}_x \\ v_1 = \ddot{w}_y - y + 2(\dot{w}_y - \dot{y}) + \ddot{w}_y \end{cases}$$

et

$$\begin{cases} \ddot{x}_e = \ddot{w}_x - x_e + 2(\dot{w}_x - \dot{x}_e) + \ddot{w}_e \\ \ddot{y} = \ddot{w}_y - y + 2(\dot{w}_y - \dot{y}) + \ddot{w}_y \end{cases}$$

Soit finalement l'expression du vecteur d'entrées

$$\begin{pmatrix} u_0 \\ u_1 \end{pmatrix} = \begin{pmatrix} -v\sin\theta & \cos\theta \\ v\cos\theta & \sin\theta \end{pmatrix}^{-1} \cdot \begin{bmatrix} \ddot{w}_x - x + 2(\dot{w}_x - \dot{x}) + \ddot{w}_x \\ \ddot{w}_y - y + 2(\dot{w}_y - \dot{y}) + \ddot{w}_y \end{bmatrix}$$

IV. Filtre de Kalman

Pour utiliser le filtre de Kalman , on souhaite avoir un système de la forme

$$\begin{cases} \mathbf{x}_{k+1} &= \mathbf{A}_k\mathbf{x}_k + \mathbf{u}_k + \boldsymbol{\alpha}_k \\ \mathbf{y}_k &= \mathbf{C}_k\mathbf{x}_k + \boldsymbol{\beta}_k \end{cases} \tag{1}$$

où $\boldsymbol{\alpha}_k$ et $\boldsymbol{\beta}_k$ sont des bruits blancs gaussiens indépendants dans le temps. [1] C'est à dire que les vecteurs $\boldsymbol{\alpha}_{k_1}$ et $\boldsymbol{\alpha}_{k_2}$ (ou $\boldsymbol{\beta}_{k_1}$ et $\boldsymbol{\beta}_{k_2}$) sont indépendants les uns des autres si $k_1 \neq k_2$. Le filtre opère en deux parties :

- La **correction** :
  Prenons la mesure $\mathbf{y}_k$. Le vecteur représentant l'état est maintenant $\mathbf{x}_{k|k}$, qui est différent de $\mathbf{x}_{k|k-1}$ puisque $\mathbf{x}_{k|k}$ a connaissance de la mesure $\mathbf{y}$. L'espérance $\hat{\mathbf{x}}_{k|k}$ et la matrice de covariance $\Gamma_{k|k}$ associées à $\mathbf{x}_{k|k}$ sont données par les équations suivantes.
  (i) $\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \cdot \tilde{\mathbf{y}}_k$ (estimation corrigée)
  (ii) $\Gamma_{k|k} = \Gamma_{k|k-1} - \mathbf{K}_k \cdot \mathbf{C}_k\Gamma_{k|k-1}$ (covariance corrigée)
  (iii) $\tilde{\mathbf{y}}_k = \mathbf{y}_k - \mathbf{C}_k\hat{\mathbf{x}}_{k|k-1}$ (innovation)
  (iv) $\mathbf{S}_k = \mathbf{C}_k\Gamma_{k|k-1}\mathbf{C}_k^T + \Gamma_{\beta_k}$ (variance de l'innovation)
  (v) $\mathbf{K}_k = \Gamma_{k|k-1}\mathbf{C}_k^T\mathbf{S}_k^{-1}$ (gain de Kalman)
- La **prédiction** qui permet d'obtenir l'état à l'instant $k+1$.

$$\hat{x}_{k+1|k} = A_k\hat{x}_{k|k} + \mathbf{u}_k$$
$$\Gamma_{k+1|k} = A_k \cdot \Gamma_{k|k} \cdot A_k^\top + \Gamma_{\alpha_k}.$$

Pour utiliser le filtre de Kalman, nous avons besoin d'équations linéaires de la forme

$$\mathbf{x}_{k+1} = \mathbf{A}_k\mathbf{x}_k + \mathbf{u}_k + \boldsymbol{\alpha}_k \tag{2}$$

Dans le cas de notre robot, on considère que le cap est connu à tout instant avec une bonne précision. Considérons donc le vecteur d'état réduit :

$$Z = \begin{pmatrix} x_1 \\ x_2 \\ x_4 \end{pmatrix} = \begin{pmatrix} x \\ y \\ v \end{pmatrix}$$

Z vérifie l'équation d'évolution suivante

$$\dot{z} = Az + u$$

où

$$A = \begin{bmatrix} 0 & 0 & \cos(\theta) \\ 0 & 0 & \sin(\theta) \\ 0 & 0 & 0 \end{bmatrix}$$

et

$$u = \begin{pmatrix} 0 \\ 0 \\ u_1 \end{pmatrix}$$

Après discrétisation :

$$\Rightarrow \quad z(k+1) = z(k) + dt\,\dot{z}(k)$$

$$\Rightarrow \quad z(k+1) = \underbrace{\begin{bmatrix} 1 & 0 & dt\cos(\theta) \\ 0 & 1 & dt\sin(\theta) \\ 0 & 0 & 1 \end{bmatrix}}_{A_k} z(k) + \begin{bmatrix} 0 \\ 0 \\ dt.u_1 \end{bmatrix} + \alpha(k)$$

Si on ne connait pas précisément la position de départ :

$$\Gamma_{\dot{z}}(0) = \begin{bmatrix} \infty & & \\ & \infty & \\ & & \infty \end{bmatrix}$$

On considère que le robot mesure sa distance à un des trois landmarks choisi aléatoirement tous les $2.dt$. Le robot est capable de mesurer sa vitesse et sa distance au landmark. En se référant à la figure 1, on obtient le vecteur de mesures

$$y(k) = \underbrace{\begin{bmatrix} 0 & 0 & 1 \\ -\sin(\theta+\delta_{i_1}) & \cos\theta+\delta_{i_1}) & 0 \\ -\sin(\theta+\delta_{i_2}) & \cos(\theta+\delta_{i_2}) & 0 \end{bmatrix}}_{C_k} z(k) + \beta(k)$$

où $\beta(k)$ est un bruit blanc.

Ce modèle établi, nous pouvons à présent calculer l'erreur en position comme la norme entre la moyenne des points de l'ellipsoide de confiance et la position véritable du robot.[2] Nous obtenons ainsi le graphique :

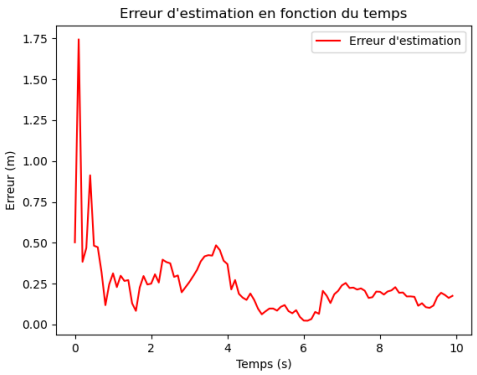


Fig. 2. Erreur en position après filtrage de Kalman

La bonne connaissance du modèle cinématique du robot nous permet donc d'obtenir une bonne estimation de sa position.

V. Intervalles

Notre robot est régi par une équation d'état. L'estimation par intervalles de l'état consiste alors à estimer l'ensemble des trajectoires réalisables qui sont solutions de cette équation . Il s'agit de propager les incertitudes dans le temps et à prendre en compte les observations issues de mesures. Pour cela, nous utilisons la propagation de contraintes sur tubes. [3]
Avant prise en compte des contraintes liées aux landmarks, le tube englobant toutes les trajectoires possibles est représenté figure 3 :

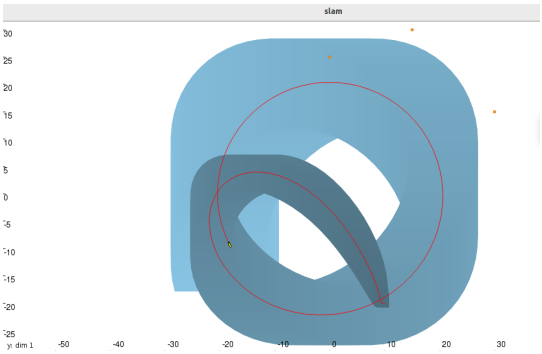


Fig. 3. Avant contraction

On utilise un réseau de contracteurs pour prendre en compte les contraintes liées à la détection de landmarks selon l'algorithme suivant :

Algorithm 1 Boucle de traitement des observations dans le réseau de contracteurs

Input:

- v_m (ensemble des positions des landmarks)
- v_m_boxes (boîtes associées aux landmarks)
- x_truth (trajectoire vraie du robot)
- x, v (états et commandes sous forme de tubes)
- cn (réseau de contracteurs)

Output:

- L'état contracté $x(t)$ sur le domaine, avec les contraintes d'observations intégrées.

Boucle principale :

4) **tant que** $t < tdomain.ub() - 1$ **faire**
   a) **si** $(t - prev\_t\_obs) > 2 \times dt$ **alors** ▷ Nouvelle observation toutes les $2 \cdot dt$
      i) choisir un landmark aléatoirement
      ii) $pos\_x \leftarrow x\_truth(t)[0:2]$ ▷ Position vraie du robot (composantes x et y)
      iii) $pos\_b \leftarrow v\_m[landmark\_id]$ ▷ Position du landmark sélectionné
      iv) $yi \leftarrow \text{Interval}\Big(\text{distance au landmark}\Big)$.
      v) INFLATE(yi, 0.03) ▷ Ajouter une incertitude bornée sur la distance
      vi) $ti \leftarrow \text{Interval}(t)$.
      vii) $xi \leftarrow \text{IntervalVector}(4)$. ▷ Création de variables intermédiaires
      viii) CN.ADD(ctc.eval, [ti, xi, x, v]) ▷ Ajouter la contrainte d'évaluation
      ix) CN.ADD(ctc.dist, [xi[0], xi[1], b[0], b[1], yi]) ▷ Ajouter la contrainte de distance
   b) contraction sur la tranche à l'instant t
5) **fin tant que**
6) CN.CONTRACT(True) ▷ Exécuter les contractions restantes

Un rectangle noir représente l'ensemble des sets du robot à l'instant de la simulation. Les anciens rectangles restent affichés. La figure 4 illustre l'animation au temps $t = tf$.

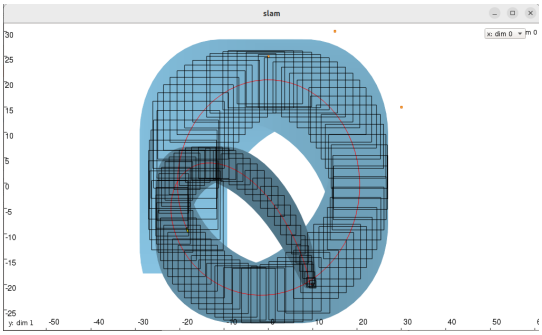


Fig. 4. Après contraction

Calculons maintenant l'erreur de positionnement comme la norme entre la position véritable et le centre de masse du sous-pavage à un instant t.

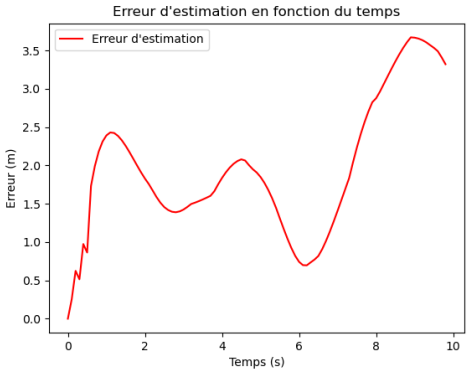


Fig. 5. Erreur de position

Nous constatons donc que l'erreur est deux fois plus importante qu'avec un filtrage de Kalman.

VI. Conclusion et Perspectives

Le modèle cinématique du robot étant connu, le filtre de Kalman est très efficace dans cette situation. En revanche, la méthode par intervalles utilisée peut aussi être adaptée sans connaitre les équations d'état qui régissent le système : elle sera donc beaucoup plus robuste pour des applications mettant en jeu des systèmes plus complexes. Pour cette application toutefois, la méthode probabiliste par filtrage de Kalman permet de minimiser l'erreur en position et se montre plus efficace d'après ce critère.

References

[1] L. Jaulin, *Kalman filter*. 2025.
[2] J. Nicola and L. Jaulin, "Comparison of kalman and interval approaches for the simultaneous localization and mapping of an underwater vehicle,"
[3] L. J. Simon Rohou Benoit Desrochers, "Set-membership state estimation by solving data association," 2020.

Control Strategies for the Autonomous Navigation of a Ducted Fan Flying Robot

Juliette Faury[1]

[1] ENSTA Bretagne

Abstract

Ducted Fan are small and discreet secure platforms, which are able to perform vertical takeoff and landing (VTOL) and stationary flight. A ducted-fan UAV is a UAV that has similar rotors configuration with a coaxial helicopter, but the rotors are mounted within a cylindrical duct. The duct helps to reduce thrust losses of the propellers, and the ducted fans normally have rotational speed. They are of evident interest for civil and military operations in an urban environment (shrouding a rotating propeller protects against blade strikes with objects at low altitude and people around the aircraft). However, this kind of vehicle is unstable and its dynamics along the three axes are strongly coupled. The purpose of this paper is to compare several control strategies for the autonomous navigation of a ducted fan UAV.

Introduction

In recent years, vertical take-off and landing (VTOL) unmanned aerial vehicles (UAVs) have attracted increasing attention in transportation, surveillance, detection and many other areas. A notable example is the ducted fan UAV,with some exemples visibles in Figure 1. It is well known for its compact layout that enables the vehicle to operate safely in crowded urban environments. [1] However, it is sensitive to overcoming wind disturbance, and can be limited by weather condition. [2]. That is why having a robust flight control system is important.

But controling a ducted fan UAV is a challenging task. Indeed, the aerodynamic efforts that apply on the vehicle are complex and hard to estimate and model (the aerodynamic effects are strongly nonlinear and involve flow discontinuities). This type of vehicle is also unstable and its dynamics along the three axes are strongly coupled. [3]

According to recent literature, control techniques can be classified into three categories: linear flight control, nonlinear model-based flight control approach, and learning-based flight control approach [4] (there are described in Table 1). In this paper, we present examples from these categories that have been successfully tested on real robots.

1. Linearisation

The linear controllers can be employed through the linearization by applying relative equilibrium near the operating point. Although the linear controllers are easy to implement and require less computational resources, the control performance deteriorates, when the linear controllers are applied during the transition between horizontal and vertical flight [4].

Proportional-derivative controller

An experimental test using a prototype and this controller on its motor was performed in August 2007 in Canada. Due to the presence of noise, several filters were used. In addition to the filters, saturation was applied to the control effort to limit the movement of the control surfaces.The desired attitude shown was obtained from a pilot input which attempted to minimize the linear motion of the system. Due to physical limitations of the aircraft surroundings, larger deviations in attitude was not possible. The results of the paper show a correlation between the desired and estimated attitude for the system[5]

Linear quadratic regulator

A linear quadratic regulator (LQR) can be shown to be efficient and relatively simpler than classical control system design to apply to the ducted-fan system. Tracking problem is introduced since the desired output is not zero. A linear quadratic tracker (LQT) can be used for the tracking problem, but the steady-state error may occur. To reduce the steady-state error, a linear quadratic tracker with integrator (LQTI) can be proposed for the CNU ducted-fan UAV. The linear quadratic regulator is basic technique by using the optimal control theory. Designing the LQR, the linearized model can be derived from mathematical modeling as the six-degree-of-freedom nonlinear equations of motion such as the Jacobian linearization method [2]

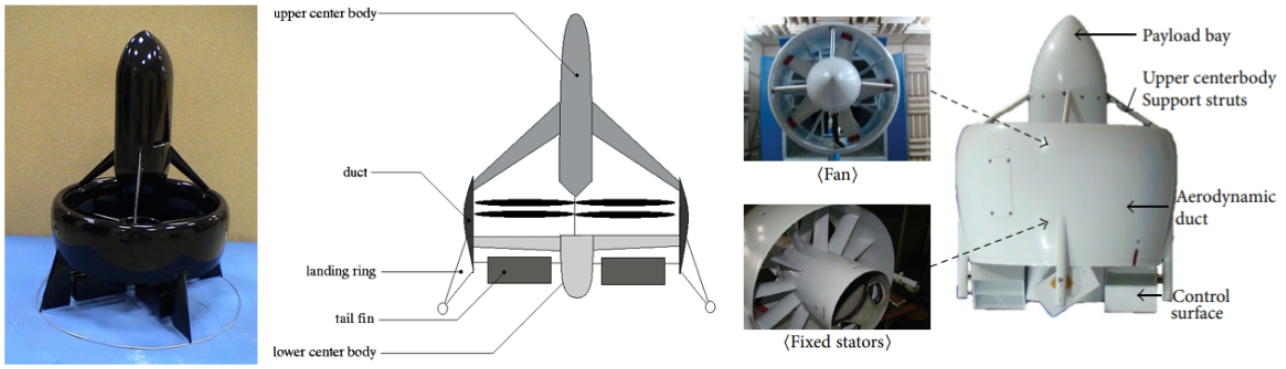


Figure 1: *Exemple of ducted fan: on the left is the Bertin VTOL UAV [3] , while on the right is the Chungnam National University (CNU) ducted-fan UAV [2]*

2. Back-stepping

Hierarchical control strategy

This controller is divided in two parts: the High Level Controller, dedicated to position control in which the magnitude and the direction of the thrust are considered as control inputs and the Low-Level controller which is designed to stabilize the attitude of the vehicle to the direction required by the High Level Controller. The method takes advantage of the connected structure of the system to design a controller dedicated to position control and another one dedicated to attitude stabilization. Global stability of the connected system has been proved, even though only asymptotic stability can be achieved on the rotational dynamics. Both controllers structure can be easily implemented in real time. [6]

Lemma : Let $\{\lambda_1, \lambda_2, \lambda_3\} \in \mathbb{R}_+^*$. Define $\{a, b, c\} \in \mathbb{R}_+^*$ as:

$$a = \sum_i \lambda_i, \quad b = \sum_{i<j} \lambda_i \lambda_j, \quad c = \prod_i \lambda_i$$

We assume that the discriminant $\Delta = b^2 - 4ac$ is positive. Define the gains k_1, k_2, k_F as:

$$k_2 = a, \quad k_1 = \frac{b - \sqrt{\Delta}}{2a}, \quad k_F = \frac{b + \sqrt{\Delta}}{2}$$

And the following error terms:

$$\delta_1 = \xi_D - \xi_s \quad \text{Position error}$$

$$\delta_2 = mk_1\delta_1 + m\nu_D \quad \text{Velocity error}$$

$$\tilde{F}_{\text{ext}} = F_{\text{ext}} - \hat{F}_{\text{ext}} \quad \text{Estimation error}$$

The system is exponentially stabilizable with the control law on the thrust vector:

$$\bar{u}_n d = k_2\delta_2 + \hat{F}_{\text{ext}} + mge_3$$

and the following adaptive filter on F_{ext}:

$$\dot{\hat{F}}_{\text{ext}} = k_F\delta_2$$

Moreover, the adaptive filter converges to the real value of F_{ext}. More precisely, $\xi \to \xi_s$ and $\hat{F}_{\text{ext}} \to F_{\text{ext}}$. [6]

Hovering flight stabilization

The back stepping technique is used to design acontrol law which adapts to incertainties F_{ext} and M_{ext}..We suppose in the control design that the desired thrust is constant. This hypothesis is acceptable because the rotational dynamics is tuned up to be much faster than the translational dynamics.The control law must define magnitude and orientation of the thrust in order to counteract perturbing wind forces. During, the tests, even though the estimated parameters only converge to a neighborhood of the real ones due to unmodelled dynamics, the vehicle stabilizes its position to the desired one. [2]

3.Neural Network

A neural-networks-based controller can be used to learn the system dynamics and compensate for the tracking error between the aircraft dynamics and the desired dynamic performance and achieve a steady transition from hover to high-speed flight. Aircraft dynamics is reconstructed into a better arranged nonlinear cascade form. Then,the proposed NNs-based control scheme on two cascade closedloop systems is applied. With all the efforts, the aircraft is able to track a certain trajectory from hover to high-speed flight. This process is successfully performed by practice flight test.[1]

Technique	Knowledge of the system	Performance in complex systems	Robustness	Real-time Implementation
Linear (PID)	Not Required	Not guaranteed	Acceptable	Easier
Adaptive	Partially Required	Depend on the knowledge of the dynamics	Poor performance in fast adaptation rates	Hard
Robust	Partially Required	Depend on the knowledge of the bounds of uncertainty	Good	Hard
Learning-based	Not Required	Guaranteed	Good	Easy

Table 1: *Comparison between different flight control approaches for DFAV [1].*

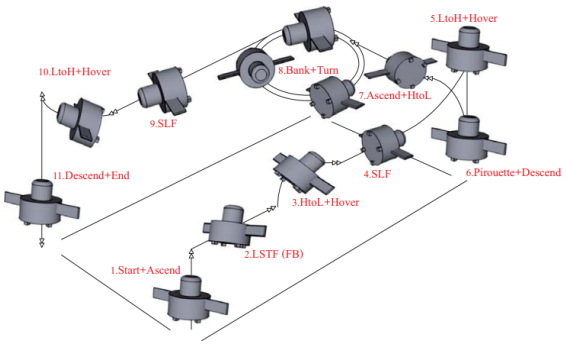


Figure 2: *DFAV flight from ascending to landing*

Conclusion

All the previous controllers can be used,because the control of a ducted fan UAV requires extensive adaptations to different flight configurations. Linearisation is adapted to hovering and cruise mode whereas back-stepping and neural network are also adapted to transition modes. Futhemore, when drones operate in confined spaces, significant proximity effects may interfere with the aerodynamic performance and pose challenges to flight safety. [7].

References

[1] Zihuan Cheng, Hailong Pei, and Shuai Li. "Neural-Networks Control for Hover to High-Speed-Level-Flight Transition of Ducted Fan UAV With Provable Stability". In: *IEEE Access* 8 (2020), pp. 100135–100151. DOI: 10.1109/ACCESS.2020.2997877.

[2] Junho Jeong, Seungkeun Kim, and Jinyoung Suk. "Control System Design for a Ducted-Fan Unmanned Aerial Vehicle Using Linear Quadratic Tracker". In: *International Journal of Aerospace Engineering* (Nov. 2015). DOI: 10.1155/2015/364926.

[3] J.M. Pflimlin, P. Soueres, and T. Hamel. "Hovering flight stabilization in wind gusts for ducted fan UAV". In: *2004 43rd IEEE Conference on Decision and Control (CDC) (IEEE Cat. No.04CH37601)*. Vol. 4. 2004, 3491–3496 Vol.4. DOI: 10.1109/CDC.2004.1429251.

[4] Tayyab Manzoor et al. "Flight control techniques and classification of ducted fan aerial vehicles丨涵道风扇飞行器飞行控制技术与分类 ". In: *Kongzhi Lilun Yu Yinyong/Control Theory and Applications* 39 (Mar. 2022), pp. 201–221. DOI: 10.7641/CTA.2021.00779.

[5] Andrew David Roberts. "Attitude estimation and control of a ducted fan VTOL UAV". en$_U$S. Thesis. Lakehead University, 2007. URL: http://knowledgecommons.lakeheadu.ca/handle/2453/3699.

[6] J.M. Pflimlin et al. "A hierarchical control strategy for the autonomous navigation of a ducted fan flying robot". In: *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* 2006, pp. 2491–2496. DOI: 10.1109/ROBOT.2006.1642076.

[7] Yiwei Luo et al. "Numerical simulation and analysis of a ducted-fan drone hovering in confined environments". In: *Advances in Aerodynamics* 6.1 (July 2024), p. 18. ISSN: 2524-6992. DOI: 10.1186/s42774-024-00179-z. URL: https://doi.org/10.1186/s42774-024-00179-z.

A Comparison of RRT-connect, RRT*-connect and Informed RRT*-connect Path Planning Algorithms

Harendra RANGARADJOU
harendra.rangaradjou@ensta.fr

***Abstract*—Path planning is a critical component in the navigation of autonomous mobile robots, ensuring efficient, collision-free traversal in dynamic and complex environments. Sampling based planning algorithms derived from the Rapidly-exploring Random Tree (RRT) algorithm have been extensively studied in recent years. They are probabilistic complete algorithms and are particularly well-suited for solving high-dimensional complex problems. This paper focuses on the RRT-connect algorithm and two of its evolutions, namely RRT*-connect and informed RRT*-connect. Firstly, a succinct analytical overview of the three algorithms is provided. A performance comparison based on optimality criteria such as path cost, process time, and the total number of vertices in the tree is then conducted through simulation-based experiments, offering insights into their relative strengths and trade-offs.**

***Index Terms*—Path Planning, RRT-Connect, RRT*-Connect, Informed RRT*-Connect, Comparison, Review**

I. Introduction

Path planning has played an essential role in various fields ranging from graphics animation to the navigation of autonomous mobile robots (see [1]–[7] in [1]). Planning algorithms can be broadly classified into two categories, graph-based and sampling-based. These use different strategies to explore the configuration space wherein they operate [2]. On the one hand, graph-based algorithms, like A* [3], fully discretize the configuration before exploring it. This allows them to be resolution complete, meaning they always find a solution if one exists, provided that they are properly tuned. However, this prevents them from being scaled easily to high dimensionality problems. On the other hand, Sampling-Based Planning (SBP) algorithms randomly sample the configuration space. As a result, they are at best probabilistic complete, meaning that they will only find a solution with a probability of 1 for an infinite number of samples. Nevertheless, they are particularly well-suited for high-dimensional and complex problems. In addition, SBP can be further divided into two types: single-query and multi-query [4]. Single-query implies searching for a path between two points in the configuration space while multi-query implies searching for a graph connecting three or more points.

Rapidly-exploring Random Tree (RRT) algorithms are a cornerstone of SBP. The original RRT algorithm was introduced by LaValle [5]. It uses a random tree rooted at a start state in the configuration space, extending it until it reaches a goal state. This solution offers consistent behaviours, probabilistic completeness and scalability but does not guarantee optimality. Extensions of this algorithm were later proposed to account for this drawback. For instance, RRT* [6] ensures the asymptotic optimality of the obtained path. Other solutions, like RRT*-smart [7] and Informed RRT* [8], were later introduced to enhance the convergence speed of RRT*.

Before the introduction of RRT*, RRT-Connect [9] improved on RRT by using a bidirectional tree expansion strategy to accelerate pathfinding. It uses two trees, growing from the start state and goal state respectively, which extend simultaneously until they meet. Building upon this method, RRT*-Connect [10] and Informed RRT*-Connect [1] further ensure asymptotic optimality and higher convergence speeds, by adapting improvements made to the original RRT algorithm.

This paper provides a comparative review of RRT-Connect, RRT*-Connect, and Informed RRT*-Connect algorithms. Simulation-based experiments are used to analyze the performance of these algorithms using criteria such as path cost, process time, and tree density. The trade-offs between computational efficiency and path quality are also discussed.

II. Algorithms overview

A. Problem definition and notations

We define the optimal motion planning problem in a manner similar to [1, 4, 8, 10]. Let X be the configuration space and $X_{\text{obs}} \subseteq X$ be the subset of states that imply collisions with obstacles. We denote $X_{\text{free}} = cl(X \setminus X_{\text{obs}})$ the subset of valid states, where cl is a closed set. Let $x_{start} \in X_{\text{free}}$ be the start state and $X_{\text{goal}} \subset X_{\text{free}}$ be the goal region. We define a tree G over the configuration space as the pair (V, E), where $V \subseteq X$ and $E \subseteq V \times V$ are the set of its vertices and edges, respectively.

We further define a path between two states $x_a, x_b \in X$ as a continuous function $\sigma : [0,1] \rightarrow X$ such that $\sigma(0) = x_a$ and $\sigma(1) = x_b$. Let $\Sigma_{X_{\text{free}}}$ be the set of all

collision-free paths between x_{start} and X_{goal} (*i.e.* $\Sigma_{X_{\text{free}}} = \{\sigma \in \mathcal{C}([0,1], X) \mid \sigma(0) = x_{start}, \sigma(1) \in X_{\text{goal}}$ and $\forall\ s \in [0;1], \sigma(s) \in X_{\text{free}}\}$). We define a cost function $c : \Sigma_{X_{\text{free}}} \to \mathbb{R}_+$ that assigns a cost value to all collision-free paths. Therefore, the optimal motion planning definition is to search for a path $\sigma* \in \Sigma_{X_{\text{free}}}$ minimizing the cost function c, as shown in Equation 1.

$$\sigma* = \underset{\sigma \in \Sigma_{X_{\text{free}}}}{\operatorname{argmin}} \{c(\sigma)\} \tag{1}$$

We denote $X_f \subseteq X$ the subset of the configuration space which can provide a better solution cost than the existing one, c_{best}.

$$X_f = \{x \in X \mid f(x) < c_{best}\} \tag{2}$$

f is defined as the cost of an optimal path passing through x. Planners can therefore increase their convergence rate by limiting their search on states that belong to X_f. However, f in equation 2 is unknown, and it is computationally complicated to be found. Instead, a heuristic function $\hat{f}$ can be considered as an estimation. This heuristic $\hat{f}$ is admissible if it never overestimates the actual value of f.

B. RRT-Connect

The RRT-Connect [9] algorithm is summarized in Algorithm 3. As stated previously, two trees are grown from x_{start} and x_{goal} respectively, and a path forms when the two trees meet. The algorithm uses the same expansion strategy as RRT [5] for each individual tree (*i.e.* EXTEND). Once it obtains a new valid candidate x_{new} for the tree G_a, it attempts to connect the trees. The CONNECT function, detailed in Algorithm 2, extends the tree G_b until it reaches either x_{new} or an obstacle. If an obstacle is reached, the trees are swapped, and the process is repeated until they meet. Nevertheless, like RRT, RRT-Connect does not guarantee path optimality.

The SAMPLE function returns a random state in the configuration space. The EXTEND algorithm returns REACHED when the new state is the goal state, otherwise it returns ADVANCED when the new state is added to the tree (*i.e.* the new state is valid) and TRAPPED when the new state is not added to the tree (*i.e.* the new state is invalid). The NEAREST function returns the closest vertex in the tree to a given state. The STEER function returns a new state that respects a certain set of constraints relative to the nearest vertex in the tree. The COLLISIONFREE function returns whether the path between two states is collision-free.

Algorithm 1 Extend

```
1: in: G = (V, E), x
2: out: S ∈ {Reached, Advanced, Trapped}
3:
4: x_nearest ← Nearest(G, x)
5: x_new ← Steer(x_nearest, x)
6: if CollisionFree(x_nearest, x_new) then
7:     V ← V ∪ {x_new}
8:     E ← E ∪ {(x_nearest, x_new)}
9:     if x_new = x then
10:        return Reached
11:    else
12:        return Advanced
13:    end if
14: end if
15: return Trapped
```

Algorithm 2 Connect

```
1: in: G = (V, E), x
2: out: S ∈ {Reached, Advanced, Trapped}
3:
4: repeat
5:     S ← Extend(G, x)
6: until S ≠ Advanced
7: return S
```

Algorithm 3 RRT-Connect

```
1: in: x_start, x_goal, n
2: out: G_a = (V_a, E_a), G_b = (V_b, E_b)
3:
4: V_a ← {x_start}, E_a ← ∅
5: V_b ← {x_goal}, E_b ← ∅
6: G_a ← (V_a, E_a) G_b ← (V_b, E_b)
7: for i = 1, ..., n do
8:     x_rand ← Sample()
9:     if Extend(G_a, x_rand) ≠ Trapped then
10:        if Connect(G_b, x_new) = Reached then
11:            return G_a, G_b
12:        end if
13:    end if
14:    Swap(G_a, G_b)
15: end for
16: return G_a, G_b
```

C. RRT-Connect*

RRT*-Connect improves upon RRT-Connect by adding a near neighbor search and rewiring steps, first introduced in RRT* [6], to its tree expansion strategy. All the changes are located in the EXTEND* function, presented in Algorithm 4. Once a new candidate x_{new} is obtained via the STEER function, the near neighbor search (lines 9–21) will look for vertices of G which could be used to minimize the cost of x_{new} in G, within a radius r_{RRT^*} of x_{new}. The list of these neighbors is given by the NEAR function. Once the proper edges are added to E, the rewiring process (lines 22–30) will look for vertices of G, in the same neighborhood, that could be reached more efficiently from x_{new} than from their current parent and rewire the tree accordingly.

Algorithm 4 Extend*

in: $G = (V, E)$, x
out: $S \in \{Reached, Advanced, Trapped\}$

$x_{\text{nearest}} \leftarrow \text{NEAREST}(G, x)$
$x_{\text{new}} \leftarrow \text{STEER}(x_{\text{nearest}}, x)$
if $\text{COLLISIONFREE}(x_{\text{nearest}}, x_{\text{new}})$ **then**
  $V \leftarrow V \cup \{x_{\text{new}}\}$
  $x_{\min} \leftarrow x_{\text{nearest}}$
  $X_{\text{near}} \leftarrow \text{NEAR}\left(G, x_{\text{new}}, \min\left(r_{\text{RRT}^*}, \eta\right)\right)$
  $c_{\min} \leftarrow \text{COST}(x_{\text{nearest}}, G) +$
  $\text{COST}(\text{LINE}(x_{\text{nearest}}, x_{\text{new}}))$
  for all $x_{\text{near}} \in X_{\text{near}} \setminus \{x_{\text{nearest}}\}$ **do**
    if $\text{COLLISIONFREE}(x_{\text{near}}, x_{\text{new}})$ **and**
  $\text{COST}(x_{\text{near}}, G) + \text{COST}(\text{LINE}(x_{\text{near}}, x_{\text{new}}))$
  $< c_{\min}$ **then**
      $x_{\min} \leftarrow x_{\text{near}}$
      $c_{\min} \leftarrow \text{COST}(x_{\text{near}}, G) +$
  $\text{COST}(\text{LINE}(x_{\text{near}}, x_{\text{new}}))$
    end if
  end for
  $E \leftarrow E \cup \{(x_{\min}, x_{\text{new}})\}$
  for all $x_{\text{near}} \in X_{\text{near}} \setminus \{x_{\min}\}$ **do**
    if $\text{OBSTACLEFREE}(x_{\text{new}}, x_{\text{near}})$ **and**
  $\text{COST}(x_{\text{new}}, G) + \text{COST}(\text{LINE}(x_{\text{new}}, x_{\text{near}}))$
  $< \text{COST}(x_{\text{near}}, G)$ **then**
      $x_{\text{parent}} \leftarrow \text{PARENT}(x_{\text{near}}, G)$
      $E \leftarrow E \setminus \{(x_{\text{parent}}, x_{\text{near}})\}$
      $E \leftarrow E \cup \{(x_{\text{new}}, x_{\text{near}})\}$
    end if
  end for
  if $x_{\text{new}} = x$ **then**
    return REACHED
  else
    return ADVANCED
  end if
end if
return TRAPPED

Algorithm 5 Connect*

in: $G = (V, E)$, x
out: $S \in \{Reached, Advanced, Trapped\}$

repeat
  $S \leftarrow \text{EXTEND}^*(G, x)$
until $S \neq \text{ADVANCED}$
return S

Algorithm 6 RRT-Connect*

in: x_{start}, x_{goal}, n
out: $G_a = (V_a, E_a)$, $G_b = (V_b, E_b)$

$V_a \leftarrow \{x_{\text{start}}\}$, $E_a \leftarrow \emptyset$
$V_b \leftarrow \{x_{\text{goal}}\}$, $E_b \leftarrow \emptyset$
$G_a \leftarrow (V_a, E_a)$ $G_b \leftarrow (V_b, E_b)$
for $i = 1, \ldots, n$ **do**
  $x_{\text{rand}} \leftarrow \text{SAMPLE}()$
  if $\text{EXTEND}^*(G_a, x_{\text{rand}}) \neq \text{TRAPPED}$ **then**
    if $\text{CONNECT}^*(G_b, x_{\text{new}}) = \text{REACHED}$ **then**
      return G_a, G_b
    end if
  end if
  $\text{SWAP}(G_a, G_b)$
end for
return G_a, G_b

D. Informed RRT-Connect*

Lastly, Informed RRT*-Connect [1] expands upon RRT*-Connect by introducing informed sampling [8], presented in Algorithm 8, to further improve efficiency and path quality. The algorithm initially operates like RRT*-Connect. Once an initial feasible path is found, sampling is restricted to an ellipsoidal region defined by the current best path cost and the positions of x_{start} and x_{goal}. This focused exploration reduces sampling in irrelevant regions of the configuration space. Additionally, Informed RRT*-Connect retains the bidirectional search and optimal path refinement capabilities of its predecessors, achieving high-quality solutions while maintaining asymptotic optimality.

The ROTATIONTOWORLDFRAME function returns a rotation matrix that aligns the informed sampling ellipsoid with the best path. The SAMPLEUNITBALL function returns a random point in the unit ball. The CALCULATESHORTESTPATHLENGTH function returns the length of the shortest path found so far. Note that tree pruning step, shown in lines 12–14 of Algorithm 9 and detailed in Algorithm 7, is not present in the implementation used in the benchmarks but is included here for completeness.

Algorithm 7 PruneTree

in: $V \subseteq X$, $E \subseteq V \times V$, $c_{best} \in \mathbb{R}_+$
out: V, E

repeat
  $V_{\text{prune}} \leftarrow \{v \in V \mid \hat{f}(v) > c_{\text{best}}$ and $\forall$
  $w \in V, (v, w) \notin E\}$
  $E \leftarrow \{(u, v) \in E \mid v \in V_{\text{prune}}\}$
  $V \leftarrow V_{\text{prune}}$
until $V_{\text{prune}} = \emptyset$

Algorithm 8 InformedSample

in: x_{start}, x_{goal}, $c_{\max}$
out: x_{rand}

if $c_{\max} < \infty$ **then**
  $c_{\min} \leftarrow \|x_{\text{goal}} - x_{\text{start}}\|_2$
  $x_{\text{center}} \leftarrow \frac{x_{\text{start}} + x_{\text{goal}}}{2}$

$C \leftarrow$ RotationToWorldFrame$(x_{\text{start}}, x_{\text{goal}})$
$r_1 \leftarrow \frac{c_{\max}}{2}$
$\{r_i\}_{i=2,\ldots,n} \leftarrow \sqrt{\frac{c_{\max}^2 - c_{\min}^2}{2}}$
$L \leftarrow \text{diag}\{r_1, r_2, \ldots, r_n\}$
$x_{\text{ball}} \leftarrow$ SampleUnitBall()
$x_{\text{rand}} \leftarrow (CLx_{\text{ball}} + x_{\text{center}}) \cap X$
else
$x_{\text{rand}} \sim \mathcal{U}(X)$
end if
return x_{rand}

Algorithm 9 Informed RRT-Connect*

in: x_{start}, x_{goal}, n
out: $G_a = (V_a, E_a), G_b = (V_b, E_b)$

$V_a \leftarrow \{x_{\text{start}}\}$, $E_a \leftarrow \emptyset$
$V_b \leftarrow \{x_{\text{goal}}\}$, $E_b \leftarrow \emptyset$
$G_a \leftarrow (V_a, E_a)$, $G_b \leftarrow (V_b, E_b)$
$X_{\text{soln}} \leftarrow \emptyset$
$c_{\text{best}} \leftarrow \infty$
for $i = 1$ to n **do**
$c_{\text{best,prev}} \leftarrow c_{\text{best}}$
$c_{\text{best}} \leftarrow$ CalculateShortestPathLength(X_{soln})
if $c_{\text{best}} < c_{\text{best,prev}}$ **then**
PruneTree(V, E, c_{best})
end if
$x_{\text{rand}} \leftarrow$ InformedSample$(x_{\text{start}}, x_{\text{goal}}, c_{\text{best}})$
if Extend*$(G_a, x_{\text{rand}}) \neq$ Trapped **then**
Connect*(G_b, x_{new})
end if
Swap(G_a, G_b)
if isSolutionFound(x_{new}) **then**
$X_{\text{soln}} \leftarrow X_{\text{soln}} \cup \{x_{\text{new}}\}$
end if
end for
return G_a, G_b

III. Experiments

Two distinct 2D scenarios were used to evaluate the performance of the algorithms. Both scenarios were simulated over 20 independent runs of 3000 iterations each. All benchmarks were executed using Python 3.10.9 on a 64-bit Ubuntu 20.04.6 system, running on an AMD Ryzen 7 5800H CPU (base frequency of 3.2 GHz, with Turbo Boost up to 4.4 GHz) and 16 GB of memory. The unoptimized implementation was shared as much as possible between the three algorithms to ensure a fair relative comparison. All code is available on GitHub [11].

A. Single obstacle map

In the first scenario, a simple environment featuring a single obstacle is used, as shown in Figure 1. The configuration space is defined as $X = [0, 100] \times [0, 100]$. The start state x_{start} is placed at coordinates $(15, 50)$ and the goal state x_{goal} at coordinates $(85, 50)$. A single 40×40 square obstacle is placed at coordinates $(50, 50)$, resulting in $X_{obs} = [30, 70] \times [30, 70]$.

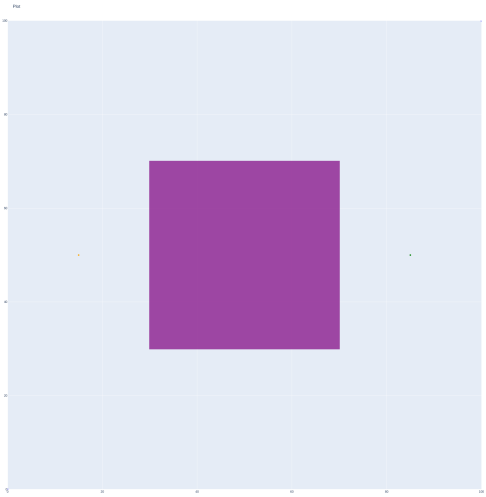


Figure 1: *Map used in the single obstacle scenario.*

B. Cluttered map

In the second scenario, a cluttered map containing multiple randomly generated obstacles provided a more challenging testbed, as illustrated by Figure 2. The configuration space is once again defined as $X = [0, 100] \times [0, 100]$. The start state x_{start} is placed at coordinates $(0, 0)$ and the goal state x_{goal} at coordinates $(100, 100)$. Forty rectangular obstacles are placed randomly in the configuration space without overlapping and are then left unchanged throughout all runs. The edge lengths of each obstacle are randomly set between 1 and 10% of the configuration space size along the corresponding axis (*e.g.* between 1 and 10 along both the x and y axes in our case).

Figure 2: *Map used in the cluttered scenario.*

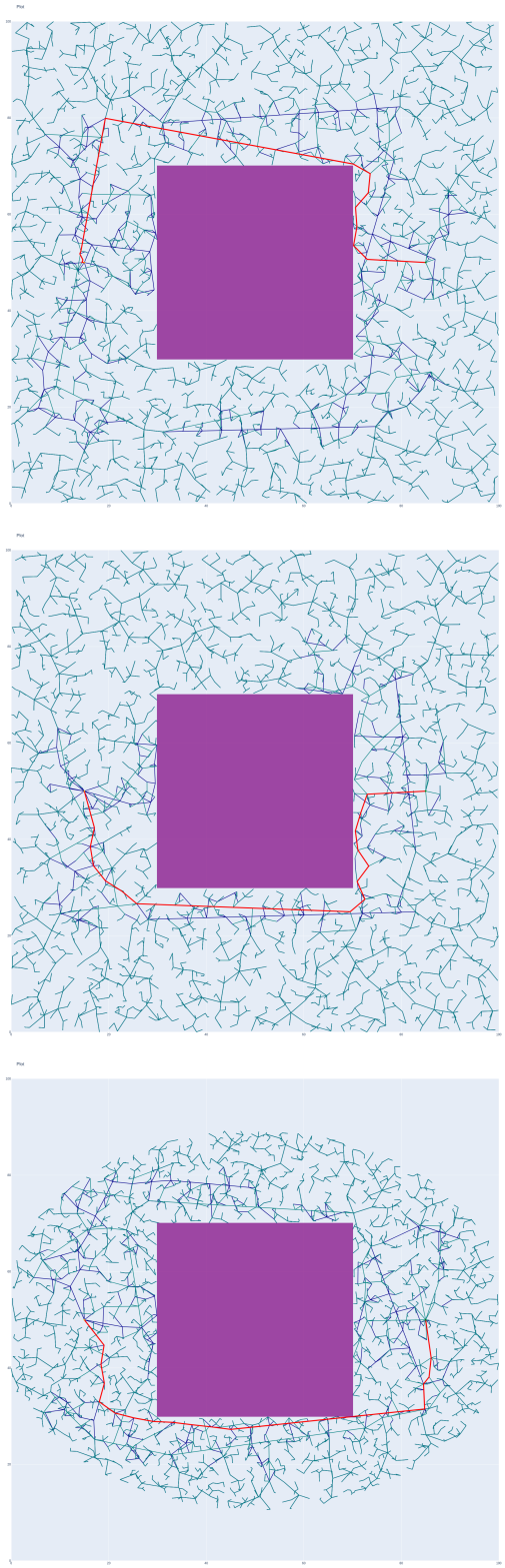


Figure 3: *Sample run for single obstacle scenario. Top: RRT-Connect. Middle: RRT*-Connect. Bottom: Informed RRT*-Connect.*

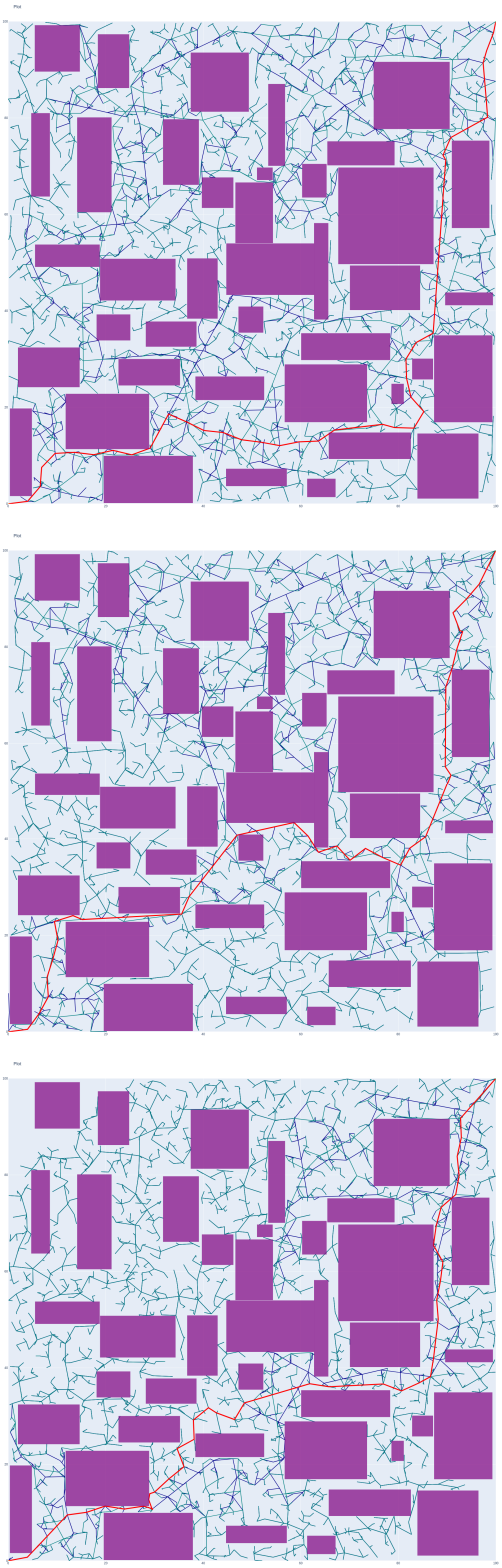


Figure 4: *Sample run for cluttered scenario. Top: RRT-Connect. Middle: RRT*-Connect. Bottom: Informed RRT*-Connect.*

IV. Results

The performance metrics recorded in our experiments include the CPU times, cost of the best path, and tree density, as well as the iteration count at which the first solution was found. Figures 5 and 6 summarize the process times and path costs for the cluttered and single obstacle scenarios, respectively, while Table I compiles the median scores for each algorithm across all measured metrics.

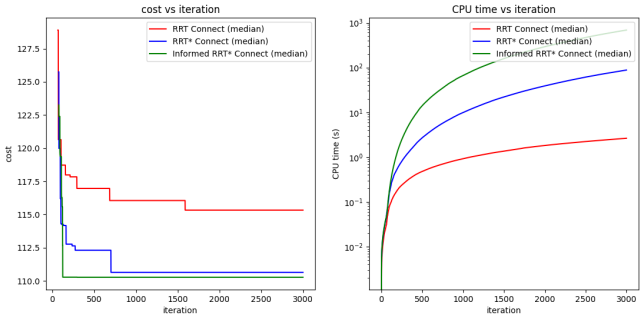


Figure 5: *Median costs and CPU times for single obstacle scenario*

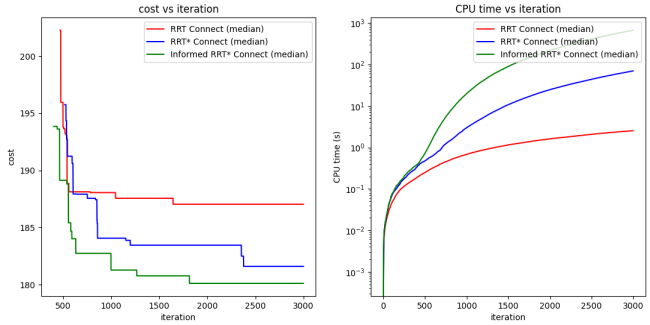


Figure 6: *Median costs and CPU times for cluttered scenario*

Scenario	Algorithm	Success rate	Total CPU time
Single obstacle	RRT-connect	1.0	2.64 ± 0.11
	RRT*-connect	1.0	87.92 ± 3.76
	Informed RRT*-connect	1.0	685.81 ± 73.59
Cluttered obstacle	RRT-connect	1.0	2.54 ± 0.18
	RRT*-connect	1.0	70.09 ± 10.28
	Informed RRT*-connect	1.0	673.22 ± 142.68

Iteration of first solution	Best path cost	Tree density		
		start tree	goal tree	total
66 ± 17	115 ± 9	2983 ± 12	2986 ± 8	5969 ± 19
73 ± 17	111 ± 8	2981 ± 10	2977 ± 12	5956 ± 21
74 ± 22	110 ± 7	2984 ± 16	2984 ± 16	5967 ± 30
454 ± 129	187 ± 9	2416 ± 125	2417 ± 121	4834 ± 234
491 ± 195	182 ± 7	2343 ± 184	2350 ± 148	4664 ± 325
365 ± 247	180 ± 8	2457 ± 262	2448 ± 206	4905 ± 466

Table I: *Comparison of median scores for all scenarios and all algorithms with associated standard deviation*

RRT-Connect consistently achieved the fastest runtimes in the single obstacle scenario, reflecting its lightweight expansion strategy. Although incurring higher runtimes, RRT*-Connect successfully converged to lower-cost paths compared to RRT-Connect, confirming that the additional computation invested in near neighbor searches and rewiring operations directly contributes to improved path quality. Lastly, while exhibiting significantly higher computational costs, Informed RRT*-Connect returned both the best path quality and the highest convergence rate, underlining the effectiveness of informed sampling. Similar trends were observed in the cluttered scenario.

While expectedly higher than that of RRT-Connect, the computational cost per iteration of RRT*-Connect after finding a first solution was significantly higher than anticipated, even for suboptimal implementations. This discrepancy is even more pronounced for the gap between RRT*-Connect and Informed RRT*-Connect. Indeed, the gaps in process times shown in Figure 5 are akin to a whole degree of polynomial complexity going from RRT-Connect to RRT*-Connect or from RRT*-Connect to Informed RRT*-Connect. As a result, no process time based review can be conducted in a meaningful way, even relatively. We suspect that the implementation of the method used to return the neighbourhood used for the near neighbor search and the rewiring operations is the main culprit.

V. Conclusion

This study provided a comparative review of RRT-Connect, RRT*-Connect, and Informed RRT*-Connect algorithms for path planning. Simulation-based experiments were conducted in two 2D scenarios–a simple single obstacle map and a more complex cluttered map–where performance was evaluated in terms of path cost, runtime, and tree density.

While the experiments demonstrated the expected relative strengths of each algorithm, the exceedingly high process times prevented any related evaluation of the algorithms, even relative. Future work should therefore focus on code optimization, specifically on the analysis of the data structures used and associated methods. Despite this unexpected setback, the results in this paper still provide some insights into the qualitative performance differences between RRT-Connect, RRT*-Connect, and Informed RRT*-Connect.

***Note*—This paper was written as an assignment for an introductory course on research methodology [12] and should be treated as such. The contents of this paper were heavily inspired by existing literature [13] and should not be considered original or novel work.**

References

[1] Reza Mashayekhi et al. 'Informed RRT*-Connect: An Asymptotically Optimal Single-Query Path Planning Method'. In: *IEEE Access* 8 (2020), pp. 19842–19852. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.2969316.

[2] Steven M. LaValle. *Planning Algorithms*. en. Cambridge University Press, May 2006. ISBN: 978-1-139-45517-6.

[3] Peter E. Hart, Nils J. Nilsson and Bertram Raphael. 'A Formal Basis for the Heuristic Determination of Minimum Cost Paths'. In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (July 1968). Conference Name: IEEE Transactions on Systems Science and Cybernetics, pp. 100–107. ISSN: 2168-2887. DOI: 10.1109/TSSC.1968.300136.

[4] Sertac Karaman and Emilio Frazzoli. 'Sampling-based algorithms for optimal motion planning'. en. In: *The International Journal of Robotics Research* 30.7 (June 2011). Publisher: SAGE Publications Ltd STM, pp. 846–894. ISSN: 0278-3649. DOI: 10.1177/0278364911406761.

[5] S. LAVALLE. 'Rapidly-exploring random trees : a new tool for path planning'. In: *Research Report 9811* (1998). Publisher: Department of Computer Science, Iowa State University.

[6] Sertac Karaman et al. 'Anytime Motion Planning using the RRT*'. In: *2011 IEEE International Conference on Robotics and Automation*. ISSN: 1050-4729. May 2011, pp. 1478–1483. DOI: 10.1109/ICRA.2011.5980479.

[7] Jauwairia Nasir et al. 'RRT*-SMART: A Rapid Convergence Implementation of RRT*'. en. In: *International Journal of Advanced Robotic Systems* 10.7 (July 2013). Publisher: SAGE Publications, p. 299. ISSN: 1729-8806. DOI: 10.5772/56718.

[8] Jonathan D. Gammell, Siddhartha S. Srinivasa and Timothy D. Barfoot. 'Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic'. In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. ISSN: 2153-0866. Sept. 2014, pp. 2997–3004. DOI: 10.1109/IROS.2014.6942976.

[9] J.J. Kuffner and S.M. LaValle. 'RRT-connect: An efficient approach to single-query path planning'. In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*. Vol. 2. ISSN: 1050-4729. Apr. 2000, 995–1001 vol.2. DOI: 10.1109/ROBOT.2000.844730.

[10] Sebastian Klemm et al. 'RRT*-Connect: Faster, asymptotically optimal motion planning'. In: *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. Zhuhai, China: IEEE, Dec. 2015, pp. 1670–1677. ISBN: 978-1-4673-9675-2. DOI: 10.1109/ROBIO.2015.7419012.

[11] Harendra Rangaradjou. *ReveR42/rrt-algorithms*. 3rd Mar. 2025. URL: https://github.com/ReveR42/rrt-algorithms (visited on 03/03/2025).

[12] Luc Jaulin. *Initiation à la recherche 2024-25*. 2024. URL: https://www.ensta-bretagne.fr/jaulin/learnsearch2024.html.

[13] Iram Noreen, Amna Khan and Zulfiqar Habib. 'A Comparison of RRT, RRT* and RRT*-Smart Path Planning Algorithms'. en. In: (2016).

Comparison between interactive robots and screens in social assistance

ALIX AGNES[1]

[1] ROB 2025, Initiation à la recherche, alix.agnes@ensta.fr

The goal of socially assistive robots is to provide assistance to humans in need of their services, and this assistance is carried out via social interaction. These services can be used in many settings : education, healthcare, entertainment... But as their use becomes more and more widespread, the question of their improved usefulness as opposed to screens or tablets accomplishing the same actions is an important one. This article intents to compare the benefits and drawbacks of both robots and screens in social settings. There are many factors that come into play when it comes to this comparison, such as the cost of production, its ease of use, the perception users have of these interfaces, its flexibility... The end goal of this article is to determine which of these two types of human-machine interfaces (HMI) is more relevant for users.

INTRODUCTION

In the late 1990s, the then-doctoral student at MIT Cynthia Breazeal developed Kismet, the first robot designed to interact in a natural and expressive way and to be able to recognize the emotions of the people using it.[1]. Since this original breakthrough, the field of Social Robotics has seen tremendous growth, with many more social robots being created in the past three decades.

The goal of socially assistive robots is to provide assistance to humans in need of their services, and this assistance is carried out via social interaction. Socially assistive robots are meant to be used in people's lives in various ways. Some are used to help teach, notably small children, others can be used to improve day to day life by helping people adopt better behaviors. Helping people in this way is something that is also possible via tablets, using various apps and tools. The goal of this article is to compare these two types of interfaces, and see which is best for users.

1. AN OVERVIEW OF SOCIAL ROBOTS AND SCREENS

A. Definitions and key differences

Robots and tablets share many similarities: both are machines that can interact with humans in various ways, such as through visual, tactile, and auditory interactions. Although some robots do not include a screen, many have a graphical user interface (GUI) that functionally operates like a tablet. They usually can both connect to the Internet and utilize artificial intelligence in the form of chatbots and personalized treatment based on user feedback. They are commonly used in healthcare, education, and entertainment for their assistive functions.

However, there are several key differences between robots and screens :

- **Physical Embodiment**: The most obvious difference between robots and tablets lies in their general shape. A robot will have a physical body that can move and interact in 3D space, while a tablet is a flat, stationary device with no movement.
- **Design complexity**: Robot hardware will include more elements, such as actuators, motors, and sensors than tablets. This also enables them to perform actions autonomously (such as physical gestures like waving, blinking, walking, sitting..) while tablets will usually require a user action.
- **Interactions**: Users can interact with robots via touch, unlike tablets that are restricted to visual and audio communication.

B. State of the art

B.1. A timeline of social robots

Following the creation of Kismet, several social robots were designed and commercialized in the late 1990s and early 2000s. In May 1999, Sony first released their dog-like robot AIBO, presenting it as the first robot pet to be widely available to the public. AIBOs can move around, see their environment and recognize voice commands. They can learn and mature under the guidance of their owner, through stimuli from their environment or from other robots. Furthermore, AIBOs were designed to be temperamental and do not systematically obey orders.[2]

In 2000, Honda began producing the ASIMO (Advanced Step in Innovative Mobility) humanoid robots. Although never commercialized, it was used for eighteen years to further the development of bipedal social robots. ASIMO robots can recognize moving objects, gestures, sounds, faces, and their environment, enabling human interaction. They can interpret voice

commands and gestures and distinguish those commands from other sounds, responding to handshakes, waves, and pointing. It answers questions verbally or with gestures and can recognize and address about ten different people by name.[3]


Fig. 1. The AIBO (left) and ASIMO (right) social robots

With time, more specific social robots were progressively designed, such as the PARO robot commercialized in 2003 in Japan. The goal of this seal-shaped robot was to soothe and elicit emotional responses in elderly patients suffering from cognitive disorders such as dementia and Alzheimer's. It also showed positive results with young autistic children.[4]

The 2010s saw the creation of ever more evolved social robots, using AI to better understand and react to social interactions. One such example is the robot Pepper, commercialized in 2014 by Aldebaran Robotics. This robot does not have a particular domestic purpose, and has for its only goal to connect with people.[5] It is able to recognize emotion based on detection and analysis of facial expressions and voice tones, and is one of the best selling social robot with around 27 000 units sold worldwide. [6]

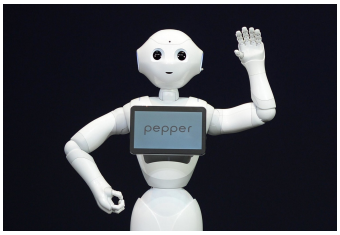



Fig. 2. The PARO (left) and Pepper (right) social robots

B.2. A timeline of digital screens

In 1961, the RAND tablet was created and is credited as the first digital graphic device. Combining a tablet interface and a monitor, users could write and draw on the tablet, and the input would be displayed on the monitor. [7] This technology was refined over the following decade, leading to the release of the Apple II tablet in 1979, although this release was not commercially successful.

The first commercial success in tablet technology was the release in 1989 of the GRiDPad, a pen-enabled tablet by Grid Systems Corporation. Several other models followed suit in the late 1990's and early 2000's from companies such as Apple and Hewlett-Packard (with the HP Compaq TC1100 model in 2001).

The next major breakthrough was the release of the iPad by Apple in 2010. Although not a new creation, this tablet pushed touch-screen tablets into the mainstream, and led to a rise in affordable and user-friendly devices.[8] Similar products were quickly released by market competitors such as Samsung with its Galaxy Tab, also released in 2010.

The rise of tablets in mainstream culture led to many new possibilities in the field of human-machine interactions. Their user-friendliness and intuitive use made it easy to integrate them in industries such as healthcare, business and education as a learning or monitoring aid. It also encouraged app development dedicated to tablets, leading to a wave of tablet-specific games and tools (such as productivity tools).

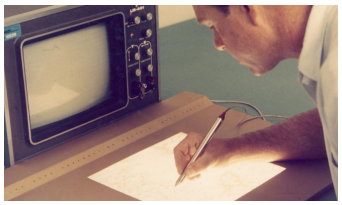
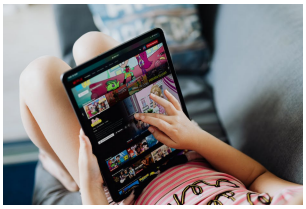



Fig. 3. From the RAND to the iPad : tablets over the years

2. COMPARATIVE ANALYSIS OF THE TWO INTERFACES

Robots and tablets are complex machines: to compare the two, several performance axes need to be used to ensure a comprehensive comparison.

A. Cost

The first criteria one can study when it comes to comparing products is the price. There are many different robots and tablets: the selection here takes a range of product to ensure the comparison remains fair.

For the robots, the models chosen are three widely commercialized models (PARO, Nao and iRobiQ), and an experimental model called Philos that aims to be a "low-cost" social robot [9] : this model will thus be considered a good representative of the cheapest a social robot can get.

Category	Model	Price (in €)
Robots	PARO	6000 [9]
	Nao	10000[9]
	iRobiQ	4000 (15 000 in 2008)[10]
	Philos	3000 [9]
Tablets	Ipad (2024 models)	200-800
	Samsung (2024 models)	250-900
	Refurbished tablets	100-200

Table 1. Comparison of Robot and Tablet Prices

On average, robot prices tend to be around ten times higher than tablet prices, and this amount doesn't account for eventual price decrease. Indeed, tablet prices are reduced drastically the longer they have been produced, and tablets that have the necessary requirements to be used in a social context can be easily found for less than 200€. On the other hand, robots can also lose value (like the iRobiQ model used in Korean schools,

which now retails at 4000€ instead of 15000€ at its launch) but most will remain close to their original price for much longer than tablets (here, the Nao and PARO models).

This price difference can be explained by the manufacturing cost and the generally more complex mechanics and electronics of robots : nevertheless, this difference favors tablets, as it makes it easier to buy them in high volume. If one wants to provide a social interface for a significant number of people, they will prefer having a larger amount of tablets than an insufficient amount of robots.

B. Flexibility and adaptability

Another factor to account for is the flexibility and adaptability of the device chosen. Both tablets and robots can accomplish a variety of actions and can be reprogrammed to adapt to a new mission, but robots are usually designed for a specific set of objectives. This specialization shows in the physical design of the robot: a robot designed for education like iRobiQ and one for play like AIBO will not look the same, and no amount of reprogramming will change its outward appearance to fit a new purpose.

Fig. 4. iRobiQ (left) and AIBO (right), two very different robots

The physical presence of robots also impacts their flexibility of use. Although some robots can be quite small, tablets are designed to be compact and portable, while robots are not. In many cases such as in schools, hospitals or research labs, their size do not stop robots from accomplishing their objectives, but in public or at home tablets remain the most practical option.

C. Ease of use and longevity

One of the leading domain of use for social robots and tablets is healthcare, and specifically care for elderly people : as the life expectancy continues to rise globally, the number of people in need of support is quickly exceeding the number of caretakers currently trained to help them. Using digital tools can help alleviate the pressure put on healthcare workers, and improve elderly people's experience with medical care. Studies have shown that using robots has a more lasting effect than tablets, by helping them improve their cognitive functioning and overall well-being[11]. As opposed to tablets, the physical component in robots such as gestures and embodied voice help them interact better with people unfamiliar with technology, as the use is simpler and more human-like than a 2D interface.

For longevity, although both tools have electrical and mechanical components that need maintenance, robots are generally made to last longer than tablets. The rise of tablets in the mainstream has lead to reduced prices compared to robots, but it has also lead to shorter lifespans : tablet are rarely expected to last longer than 5 years, while robots are a longer term investment. Unlike tablets, robots are also usually seen as a way to reduce human workforce : most of the time, firms investing in a robot will factor in the savings made thanks to this into the cost of a robot. A robot might cost 16 000€, but help save more than 80 000€ every year once it is in place[10].

D. User perception and experience

The final criteria to take into account is a crucial one : what is the user perception of these tools ? Do people consistently prefer one over the over, and does one perform significantly better than the other when it comes to its goals ?

Several studies have been performed in different environments to see the difference between tablet use and robot use.

First, studies have been done to compare tablet and robot uses in the workforce. A 2020 experiment aiming to test the efficiency of a socially assistive robot compared to a regular tablet buzzer in the context of break-taking at work[12].

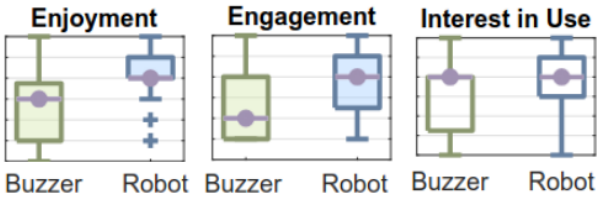


Fig. 5. Results of post-experiment survey. Purple line represents the median.

Across several metrics linked to user perception, the overall results show people favor using the robot, and find it to be a much more pleasant experience.

In the healthcare sector, the trend remains the same. In experiments carried out to see how people interacted with either a robot or a tablet computer delivering healthcare instructions, results showed participants had more positive interactions with the robot compared to the computer tablet, like increased speech and positive emotion such as smiling. After the experiment, the robot was rated higher on scales of trust, enjoyment, and desire for future interaction[13].

Another experiment comparing a robotic weight loss coach and a regular computer showed that participants developed a close relationship with the robot and ended up tracking their calorie consumption and exercise for nearly twice as long when using the robot than with the other methods[14].

Finally, studies conducted on students, and notably young children, show that robots are also favored in education. In a 2020 Kazak study, children were taught a new script and its handwriting system in three conditions: a robot and a tablet, a tablet only, and a teacher. Children's ratings and positive mood change scores demonstrated significant benefits favoring the robot over a traditional teacher and tablet only approaches[15].

Although those studies seem to point to a total robot superiority in practice, the difference between performance and user perception is important to point out. In most studies, the actual difference in performance tends to remain very close between a tablet and a robot : children learning a new script gained similar knowledge in all three experimental conditions[15], and workers taking a break did not take a significantly higher amount with the social robot[12].

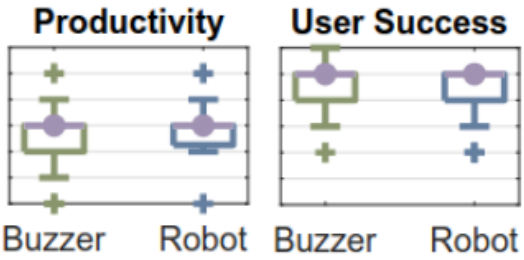


Fig. 6. Results of post-experiment survey. Purple line represents the median.

What differs with robots is the user perception : even without significant improvement in the performance, people feel better when they use a robot and enjoy their experience much more. Enjoyment plays an important role in learning new skills and keeping up positive habits : it is an important parameter to take into account when comparing different tools such as robots and tablets.

CONCLUSION

Tablets and robots share many similarities, but robots tend to perform better in the context of social assistance when it comes to user experience. The main drawbacks of a robot over a tablet are its cost, lack of flexibility and eventually size. For small scale use, such as in research labs, in classrooms in addition to a teacher, or specific retirement homes, they remain the best choice. As of now, for wider scale, public, or individual use, tablets remain the more versatile and logical choice, although new breakthroughs leading to a drastic reduction of robot prices could contest their current domination.

REFERENCES

1. C. Breazeal, “Kismet: Overview,” https://web.media.mit.edu/~cynthiab/research/robots/kismet/overview/overview.html (2025).
2. S. Corporation, “Aibo - the companion robot,” https://us.aibo.com/ (2025).
3. M. Eaton, *Evolutionary Humanoid Robotics*, SpringerBriefs in Intelligent Systems (Springer Berlin, Heidelberg, 2015), 1st ed.
4. PARO Robots, “Paro - therapeutic robot,” http://www.parorobots.com/index.asp (2025).
5. Aldebaran Robotics, “Pepper - the humanoid robot,” https://aldebaran.com/en/pepper/ (2024).
6. Reuters, “Exclusive: Softbank shrinks robotics business, stops pepper production - sources,” (2021).
7. M. Davis and T. O. Ellis, *The RAND Tablet: A Man-Machine Graphical Communications Device* (RAND Corporation, Santa Monica, CA, 1964).
8. D. Nations, “First-generation ipad: Hardware specs and features,” https://www.lifewire.com/first-generation-ipad-hardware-specs-1999509 (2024).
9. C. Puehn, T. Liu, K. Hornfeck, and K. Lee, “Design of a low-cost social robot: Towards personalized human-robot interaction,” (2014), pp. 704–713.
10. V. Young, “Is that a dalek in the surgery, dr ropata?” National Bus. Rev. (2013).
11. B. Sawik, S. Tobis, E. Baum *et al.*, “Robots for elderly care: Review, multi-criteria optimization model and qualitative case study,” Healthc. (Basel) **11**, 1286 (2023).
12. B. J. Zhang, R. Quick, A. Helmi, and N. T. Fitter, “Socially assistive robots at work: Making break-taking interventions more pleasant, enjoyable, and engaging,” in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS),* (2020), pp. 11292–11299.
13. J. A. Mann, B. A. MacDonald, I.-H. Kuo, *et al.*, “People respond better to robots than computer tablets delivering healthcare instructions,” Comput. Hum. Behav. **43**, 112–117 (2015).
14. C. D. Kidd and C. Breazeal, “Robots at home: Understanding long-term human-robot interaction,” in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems,* (2008), pp. 3230–3235.
15. Z. Zhexenova, A. Amirova, M. Abdikarimova, *et al.*, “A comparison of social robot to tablet and teacher in a new script learning context,” Front. Robotics AI **7** (2020).

Comparative Study of Swarm Robotics Control Methods

Antoine MORVAN
Filière Robotique 3e année
ENSTA Bretagne
Brest, France
antoine.morvan2@ensta.fr

***Abstract*—Swarm robotics control plays a crucial role in enabling large groups of autonomous agents to achieve collective behaviors. This paper presents a comparative analysis of different control strategies, including leader-based, decentralized, and bio-inspired methods. By reviewing key contributions from recent literature, we evaluate their efficiency in terms of convergence time, robustness, and scalability. Our analysis highlights the strengths and limitations of each approach, providing insights into their applicability in real-world swarm robotics applications.**

***Index Terms*—agent, swarm, robotics, control, bio-inspired, centralized, decentralized**

I. Introduction

Swarm robotics has gained significant attention due to its potential in applications such as search and rescue, surveillance, and environmental monitoring. The challenge lies in developing control mechanisms that ensure coordination while maintaining adaptability and scalability. Various control strategies have been proposed, ranging from centralized leader-based methods to fully decentralized approaches inspired by biological swarms. This paper compares these strategies based on their effectiveness in different swarm conditions.

In the near future, swarms of robots may replace humans and single robots for a number of common tasks. Because swarm systems are composed of multiple interacting robots, they can offer robustness, redundancy, and scalability, making them suitable for applications such as exploration and foraging [10], construction, and hazardous environment operations like firefighting or HAZMAT interventions. Indeed, recent advancements [8] have transitioned swarm robotics from theoretical models to real-world implementations in laboratory environments.

The fundamental principle behind swarm robotics lies in the use of simple, scalable control laws applied uniformly across all agents. These rules, often inspired by biological swarms, lead to emergent collective behaviors such as flocking, coordinated navigation, and task allocation. For instance, flocking can emerge from a few simple rules: maintaining a safe distance from nearby agents, moving toward distant neighbors, and aligning movement direction with local peers. This balance of attractive and repulsive forces allows a swarm to move cohesively and adapt dynamically to changing environments.

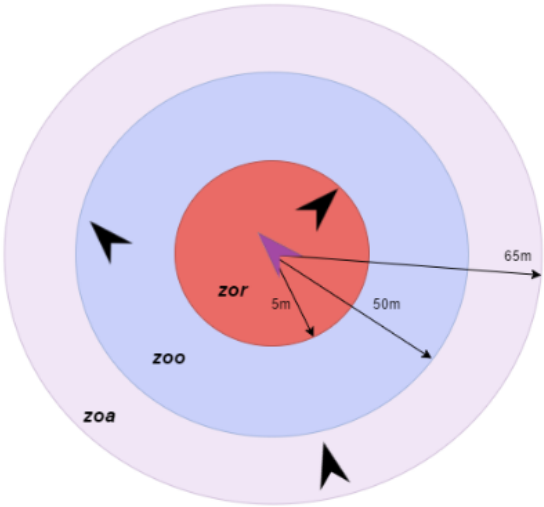


Fig. 1. Example of the zones of an agent – zone of repulsion (zor), zone of orientation (zoo), and zone of attrac- tion (zoa) [1]

While fully autonomous swarms are envisioned for many applications, in several scenarios, human intervention remains essential. Operators may need to influence swarm behavior in tasks such as large-scale area surveillance, search-and-rescue missions, or coordinated UAV operations. However, direct teleoperation of individual robots in large-scale swarms is impractical due to cognitive and bandwidth constraints. Consequently, various control strategies have been developed to inject human influence into swarms, ranging from global approaches—where a central command influences the entire swarm—to bottom-up leader/follower approaches, where specific agents guide the swarm's movements.

This study examines these control strategies, assessing their efficiency, robustness, and applicability to different operational contexts. By understanding the trade-offs between leader-based and decentralized control, we aim to highlight the best practices for designing effective swarm systems in real-world environments.

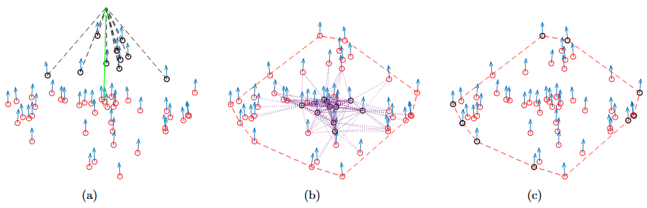


Fig. 2. Front leaders are selected as the closest agents to a reference point which is generated at 1000 m away along the direction of the average heading angle of the swarm, b) Middle leaders are selected after removing agents from the convex hull, and those that have maximum connectivity with the remaining agents (purple dotted lines show neighbours of the leaders), and c) Periphery leaders are chosen from the convex hull. The number of leaders selected are about 20% leaders of the swarm and they are black in color. Leaders are selected at t = 0. [1]

II. Leader Based Swarm Control

Leader-based control relies on a small subset of agents influencing the movement of the entire swarm. The study by Tiwari et al. (2017) [1] explores the effect of number and placement of leaders within a swarm, demonstrating that middle and periphery positions lead to faster convergence than front placements. While leader-based control offers a structured approach, it faces challenges in robustness, particularly when leaders are lost or compromised.

A key advantage of leader-based control is its ability to introduce directed movement and structured behavior in a swarm. Leaders act as reference points that dictate the swarm's global behavior, making it easier to execute predefined formations and coordinated maneuvers. This is especially useful in applications where precision is crucial, such as military operations, search-and-rescue missions, and coordinated drone formations. By strategically placing leaders within the swarm, operators can optimize performance in terms of efficiency and response time.

However, this method also has significant drawbacks. The reliance on specific agents means that if leaders fail or are removed, the entire swarm may experience disorganization or drift. The effectiveness of leader-based control is also highly dependent on the percentage of leaders within the swarm. Studies suggest that an optimal balance exists, where too few leaders result in inefficient guidance, while too many may lead to excessive control efforts and reduced swarm flexibility [1](Fig. 4). Moreover, leader-based control typically requires robust communication between leaders and followers, which can be disrupted in environments with high interference or limited connectivity.

To mitigate these challenges, hybrid strategies have been proposed, integrating leader-based control with decentralized mechanisms to improve resilience. One approach involves dynamic leader selection, where new leaders are assigned based on predefined rules or emergent behaviors when existing leaders are lost. Another method introduces secondary layers of communication to enhance redundancy, ensuring that information can still propagate effectively within the swarm.

Overall, leader-based control provides a strong foundation for structured swarm coordination but must be carefully designed to ensure adaptability and robustness, particularly in unpredictable environments.

III. Decentralized and Bio-Inspired Control

Decentralized methods, such as those proposed by Koifman et al. (2024) [2], leverage local agent interactions to achieve emergent behaviors. These approaches mimic natural systems, ensuring flexibility and robustness without relying on predefined leaders. Similarly, Vega et al. (2023) [3] introduce indirect control strategies that modify environmental conditions to influence swarm behavior, reducing the need for explicit communication among agents.

Bio-inspired control strategies take inspiration from natural collective behaviors observed in flocks of birds [7], schools of fish, and colonies of insects. These strategies rely on simple local rules followed by individual agents, which result in complex global behaviors without requiring central coordination [9]. For example, pheromone-based communication, inspired by ant colonies, allows robots to deposit virtual markers in an environment that guide the swarm toward a collective objective. Another approach mimics the alignment and cohesion rules observed in bird flocking behavior to maintain formation integrity while allowing adaptability to environmental changes.

One of the key benefits of decentralized and bio-inspired control is scalability. Since agents interact based on local information, these methods naturally extend to large swarms without the communication bottlenecks associated with leader-based approaches. Additionally, decentralized control enhances robustness, as no single agent is critical to the swarm's function. If individual agents fail, the overall system can still function effectively by redistributing influence among remaining members.

However, decentralized control also presents challenges. Achieving precise global coordination can be difficult since decision-making is distributed among agents. Convergence time may vary depending on environmental complexity and communication constraints. Furthermore, ensuring that local behaviors lead to a desired global outcome requires careful tuning of interaction rules, often requiring extensive simulations or real-world experimentation.

Overall, decentralized and bio-inspired methods offer promising alternatives to leader-based control by enhancing flexibility and robustness. Future research aims to refine these approaches to improve predictability and control in dynamic environments.

IV. Performance Comparison

We compare these control methods based on key performance metrics:

A. Convergence Time:

Leader-based control achieves faster convergence in structured environments, while decentralized methods excel in dynamic settings. The structured nature of leader-based control

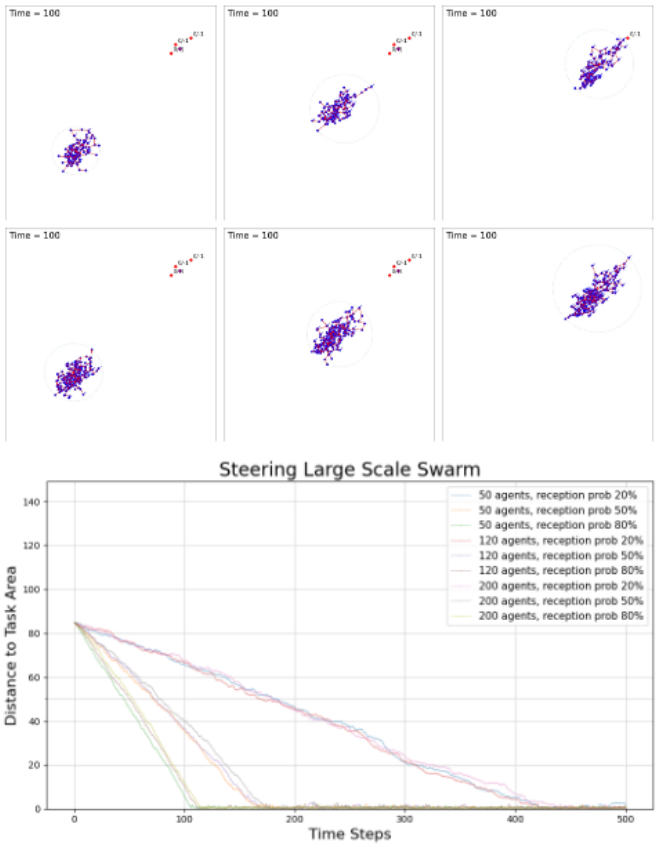


Fig. 3. The influence of various reception probabilities and swarm sizes on the swarm mobility towards the task area. The figure shows two rows of images taken at time=100. The top row, from left to right, displays 120 agents' positions with a reception probability of 20%, 50%, and 80%. The bottom row from, left to right, displays 200 agents with a reception probability of 20%, 50%, and 80% from right to left. The comparative graph depicts distance to task area of 50, 120, 200 agents with reception probability of 20%, 50%, and 80% [2].

allows for quicker information dissemination among agents, reducing the time required for the swarm to adjust its trajectory. On the other hand, decentralized control methods can adapt more effectively to unexpected environmental changes, albeit sometimes at the cost of increased convergence time due to the reliance on local interactions.

B. Robustness:

Decentralized and bio-inspired approaches demonstrate higher fault tolerance, as they do not depend on specific agents. In a leader-based system, if key leaders are removed or malfunction, the entire swarm can become disorganized. By contrast, decentralized methods distribute control responsibilities among all agents, ensuring that failures of individual units have minimal impact on the overall system performance.

C. Scalability:

Decentralized methods scale more efficiently as they eliminate the need for global coordination. As swarm size increases, communication overhead and decision-making

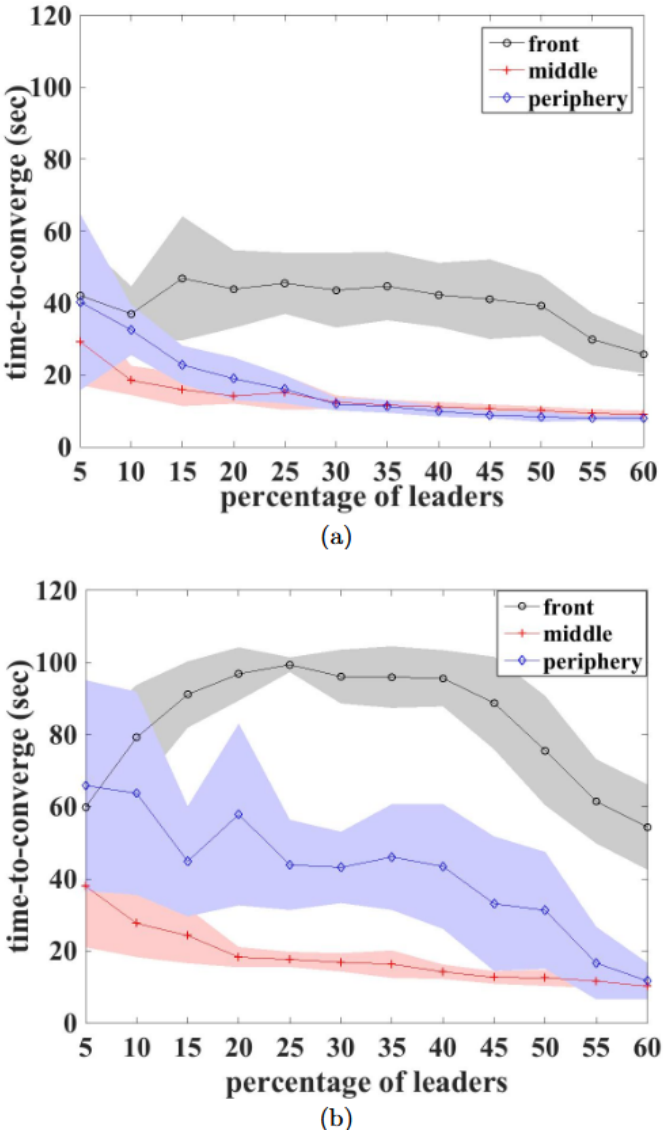


Fig. 4. Variation of the time-to-converge with percentage of leaders for a) 50 agents and b) 100 agents. The lower bound denotes the time to con- verge for a single leader with no influence [1].

bottlenecks become prominent in leader-based approaches. Decentralized approaches avoid these issues by relying on local agent interactions, enabling seamless scalability even in very large groups.

Hybrid models are being explored to balance the advantages of both paradigms. Some systems use leader agents for high-level guidance while allowing decentralized behaviors for adaptability. This combination enhances efficiency while maintaining resilience against failures. Research by Koifman et al. (2024) [2] supports the notion that distributed decision-making can complement leader-based strategies by dynamically adjusting the role of leaders based on environmental conditions and mission objectives.

Overall, the choice of control strategy depends on the

application requirements. Tasks requiring high precision may benefit from leader-based control, while those demanding adaptability and robustness favor decentralized approaches.

V. Applications and Case Studies

Leader-based control is well-suited for structured missions requiring precise coordination, such as drone formations. In contrast, decentralized methods are preferable for exploration tasks where adaptability is crucial. Case studies in search-and-rescue operations demonstrate the superiority of hybrid approaches that combine leader guidance with decentralized adaptation [11].

In disaster response scenarios, decentralized swarms have been used to map unknown environments and identify survivors autonomously. For example, the work by Dirafzoon and Lobaton (2013) [4] illustrates how topological mapping using robotic swarms can enhance navigation in complex environments. Similarly, the research by Han, Rossi, and Shen (2007) [5] shows that covert leader-based strategies enable efficient swarm guidance with minimal external control.

Industrial applications also benefit from swarm robotics, particularly in warehouse automation and agricultural monitoring. Swarms of autonomous drones have been deployed for crop surveillance, leveraging decentralized coordination to cover large areas efficiently. Studies such as those by PheroCom (2022) [6] emphasize the potential of bio-inspired communication methods, such as virtual pheromones, to enhance swarm collaboration in real-world settings.

VI. Conclusion

Swarm robotics control remains an active research area with diverse approaches catering to different operational needs. While leader-based methods offer structured control, decentralized approaches provide robustness and scalability. Future research should explore hybrid models that integrate the strengths of both paradigms to enhance swarm efficiency in complex environments. Further exploration of AI-driven control mechanisms could help refine decentralized strategies by allowing agents to dynamically adapt their behaviors based on real-time environmental feedback.

Another promising direction involves enhancing swarm resilience in dynamic environments through adaptive learning techniques. Machine learning models can be integrated into swarm control frameworks to allow robots to improve decision-making based on previous experiences. Additionally, increasing research into human-swarm interaction could help refine ways for operators to efficiently influence swarm behavior without reducing autonomy. By addressing these challenges, swarm robotics can be further optimized for real-world deployment across various industries and scientific domains.

References

[1] R. Tiwari, P. Jain, S. Butail, et al., "Effect of Leader Placement on Robotic Swarm Control," AAMAS, 2017.

[2] Y. Koifman, A. Barel, A. M. Bruckstein, "Distributed and Decentralized Control and Task Allocation for Flexible Swarms," arXiv, 2024.

[3] R. Vega et al., "Indirect Swarm Control: Characterization and Analysis of Emergent Swarm Behaviors," arXiv, 2023.

[4] A. Dirafzoon, E. Lobaton, "Topological Mapping of Unknown Environments Using an Unlocalized Robotic Swarm," IROS, 2013.

[5] X. Han, L. F. Rossi, C.-C. Shen, "Autonomous Navigation of Wireless Robot Swarms with Covert Leaders," IEEE Press, 2007.

[6] C. R. Tinoco, G. M. B. Oliveira, "PheroCom: Decentralised and Asynchronous Swarm Robotics Coordination," arXiv, 2022.

[7] M. Nagy, Z. Ákos, D. Biro, and T. Vicsek, "Hierarchical group dynamics in pigeon flocks," Nature, 464(7290), 2010.

[8] L. Parker, "Multiple mobile robot systems," Springer Handbook of Robotics, 2008.

[9] I. D. Couzin, J. Krause, R. James, et al., "Collective memory and spatial sorting in animal groups," Journal of Theoretical Biology, 2002.

[10] S. Nunnally, P. Walker, A. Kolling, et al., "Human influence of robotic swarms with bandwidth and localization issues," IEEE International Conference on Systems, Man, and Cybernetics, 2012.

[11] R. Bähnemann et al., L'article de Bähnemann et al. (2017) s'intitule "A Decentralized Multi-Agent Unmanned Aerial System to Search, Pick Up, and Relocate Objects", arXiv, 2017.

Numerical modelling of optical tweezers applied to mobile micro-robots automatization

Gaétan PEREZ

UE 5.1 – Initiation à la recherche

March 5, 2025

Abstract

Optical tweezers are contactless tools that enable precise manipulation of micron-sized particles. Discovered in 1987, this technology can be integrated into robotic systems for enhanced performance. This project aims to develop a numerical simulation to model the interactions between optical tweezers and optical receptors, facilitating the automated control of microrobots. Optical tweezers operate using a Gaussian light beam focused on a microparticle, where the beam exerts a force similar to a harmonic oscillator, depending on the particle's equilibrium position within the beam. The precise and automated control of such microrobots has great potential across various fields, especially in biology, where accurate manipulation is crucial for the success of delicate operations.

1 Introduction

Micro-scale mobile robotics, meaning actuation of sub-millimeter components, is a wide field of mobile robotics but still a bit behind classical mobile robotics due to its numerous challenges. Indeed, components can be hard to manufacture because of its size and micro robots often need very specific installations in order to be able to observe it. However, progresses in the fabrication of micro components, and scientific discoveries enabled this field of mobile robotics to grow and become a relevant solution to loads of applications such as biology, surgery, micro manufacturing [1].

Thanks to recent progresses, many actuation technics of micro sized robotics has been developed such as using electromagnetic fields, piezoelectric components or lasers for example [1] [2]. In this article, the focus will be on optical trapping (also called optical tweezers) technic applied to mobile micro robotics. Optical tweezers enable to leverage the momentum transfer between the cell and a laser beam to trap the cell within the beam, enabling precise control over its position and movement through the laser beam. Optical trapping was first discovered in the 80s and was used to manipulate cells and make them "levitate" to be able visualize them from every angle [3]. Many examples prove that optical trapping is of great importance for biology and cell manipulation [4][5][6], and therefore it can also be applied to robotics for automation. Currently, there are two potential methods for actuation using optical trapping. The first involves directly actuating a micro-sized robot via receptors. Indeed, a micro-robot controlled by optical trapping was introduced in 2019, demonstrating the ability to move across six degrees of liberty (three translational and three rotational) only through optical trapping [7]. Another way is to trap a cell on an actuated platform and to move the platform while the cell is not moving. As a result, the cell will

move on the platform layout [8].
However, one of the principal issue with optical tweezers, and more generally micro-sized robotics is its difficult control. Indeed because of its size, it is complicated to visualize precisely the moves of the robots, and to automatize its control. To complete the robot presented in[7], an manual haptic solution was proposed to control the micro robot with great precision. Nevertheless, another step for automation would be a system fully autonomous using guidance algorithm to control the robot. Such program could enable to accelerate cells preparation in biology or automatized micro-sized manufacturing. Then in this context, this study will be about developing guidance algorithm to automatize the control of the micro-sized robot presented in [7]. A dynamic python simulation will be presented to illustrate this guidance program. However, this simulation will not take in account the changes of the evolution of systems at very low Reynold number which happen at such small scale.

2 The optical Tweezers

An optical tweezers are a tool enabling trapping and manipulation of specific targets such as cells, or micro-sized particles, in air or in a liquid [9]. This tool can also be used to create "optical levitation" when only using the laser beam's force, the target is held in air without any external help.
This trap created by the laser beam interacting with the target is induced by the dipolar forces linked to the Gaussian structure of the beam. Indeed, ideal Gaussian beams has a specific electromagnetic amplitude envelope following a Gaussian curve in the transverse plan. Using lens and mirrors, a high precision focus on the target can be done as presented in figure 1

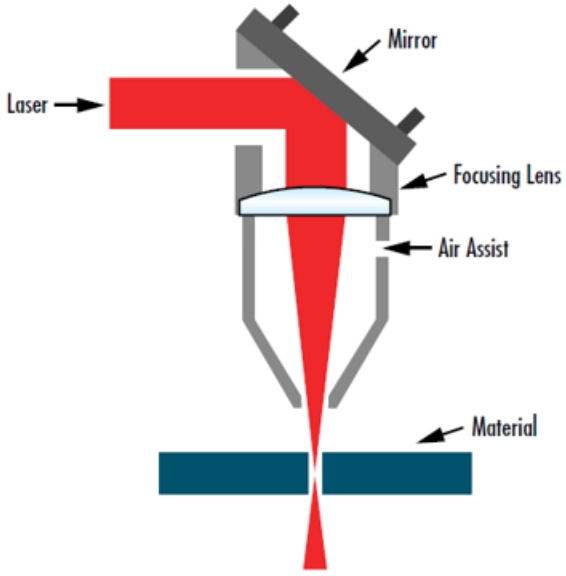


Figure 1: Laser focus using lens and mirror [10]

The interaction between the beam and the target can be expressed using the dipolar forces which represent the movement of a system impacted by a laser, linked with the coupling energy between the system and the beam:

$$\bar{U} = -\alpha * \frac{E(r,z)^2}{2}$$

where $E(r,z)^2$ is the amplitude of the electric field and α is the polarizability of the target [11]. Then, depending on the coefficient α, the target will be attracted by the minimal or maximums of intensity of the electrical field induced by the laser beam. Then, the dipolar force can be expressed:

$$F_{dip}(r,z) = \alpha * \frac{\nabla(E(r,z)^2)}{2}$$

and α can be expressed like this for micro-sized targets [11]:

$$\alpha = n_e^2 * ((\frac{n_t}{n_e})^2 - 1) * \frac{a^3}{\frac{{n_t}^2}{n_e} + 2}$$

with n_e and n_t the optical indices of the environment and the target. Since, a Gaussian laser beam is used, the amplitude of the electrical field is expressed like this:

$$E(r,z) = E_0 * \frac{\omega(0)}{\omega(z)} * e^{-\frac{r^2}{\omega(z)^2}} * e^{-i(kz + \frac{r^2}{2R(z)} - \Phi(z))}$$

where $\Phi(z) = arctan(\frac{\lambda z}{\pi\omega^2(0)})$, with λ the wavelength of the laser,
$k = 2\pi\frac{\nu}{v}$, with ν the temporal frequency of laser and v the speed of propagation of the beam (here the speed of light).
$r^2 = x^2 + y^2$, the "width" of the beam,
$R(z) = z(1 + (\frac{\pi\omega^2(0)}{\lambda z})^2)$, the bending radius of the beam,
$\omega(z) = \omega^2(0)(1 + (\frac{\lambda z}{\pi\omega^2(0)})^2)$, the beam waist as it is shown in figure 2.

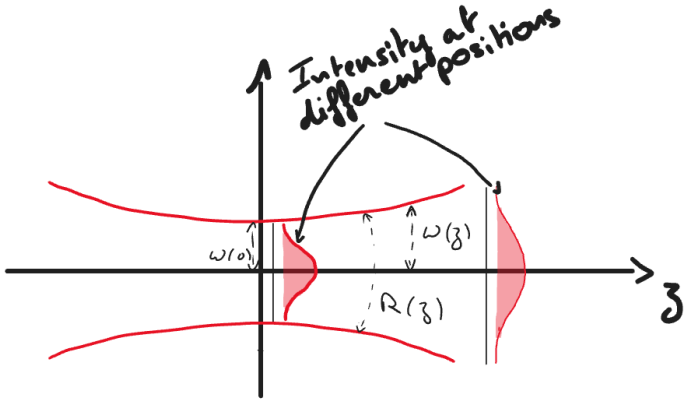


Figure 2: Scheme of a Gaussian beam profile focused on a point z=0

Using sympy, it is possible to obtain the expressions for F_x, F_y and F_z and
Taylor's development around (0, 0), we get these simplified formula of the forces, similar to an harmonic oscillator:

$$F_{x,\text{Taylor}} = -\frac{4\pi^4 E_0^2 w_0^6 x\alpha}{(\pi^2 w_0^4 + z^2\lambda^2)^2}$$

$$F_{y,\text{Taylor}} = -\frac{4\pi^4 E_0^2 w_0^6 y\alpha}{(\pi^2 w_0^4 + z^2\lambda^2)^2}$$

$$F_{z,\text{Taylor}} = -\frac{\pi^2 E_0^2 w_0^4 z\alpha\lambda^2}{(\pi^2 w_0^4 + z^2\lambda^2)^2}$$

.
Then, with $k_x(z) = \frac{4\pi^4 E_0^2 w_0^6\alpha}{(\pi^2 w_0^4 + z^2\lambda^2)^2}$, $k_y(z) = \frac{4\pi^4 E_0^2 w_0^6\alpha}{(\pi^2 w_0^4 + z^2\lambda^2)^2}$ and $k_z(z) = \frac{\pi^2 E_0^2 w_0^4\alpha\lambda^2}{(\pi^2 w_0^4 + z^2\lambda^2)^2}$ we get:

$$F_{x,\text{Taylor}} = -k_x(z)x \tag{1}$$

$$F_{y,\text{Taylor}} = -k_y(z)y \tag{2}$$

$$F_{z,\text{Taylor}} = -k_z(z)z \tag{3}$$

With x, y, z the positions relatively to the focus. Thanks to these simplifications, it will be easier to include the optical forces applied to the system in the simulation.

3 Simulation

In this section, a dynamic simulated micro robot based on the vehicle presented in [7] will be done. The simulation will follow the block diagram presented in figure 3, based on Lapierre's theory [12].

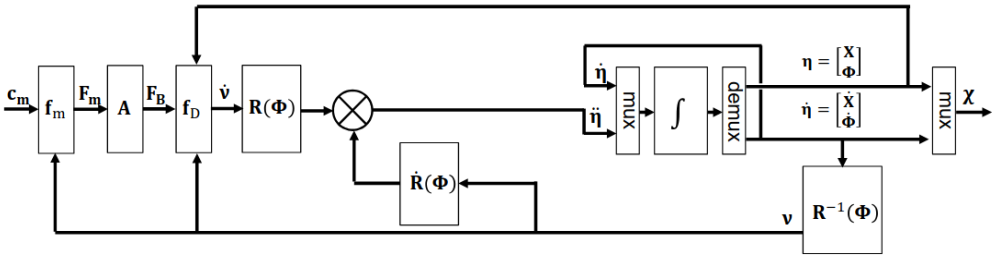


Figure 3: Block diagram of a dynamic simulation using Euler Angles [12]

3.1 Kinematic model

First, lets write the kinematic model of the robot which links the velocity and the angular speed in the global frame, with the velocity and angular speed in the body frame. The model will use the Euler angles as the gimbal lock should never be :
Then, we get $\dot{\eta} = R_{\text{cin}}^{eul} \cdot \nu$, with the Euler matrix:

$$\mathbf{R}_{\text{cin}}^{\text{eul}} = \begin{bmatrix} \cos\psi \cdot \cos\theta & \cos\psi \cdot \sin\theta \cdot \sin\phi - \sin\psi \cdot \cos\phi & \cos\psi \cdot \sin\theta \cdot \cos\phi + \sin\psi \cdot \sin\phi & 0 & 0 & 0 \\ \sin\psi \cdot \cos\theta & \sin\psi \cdot \sin\theta \cdot \sin\phi + \cos\psi \cdot \cos\phi & \sin\psi \cdot \sin\theta \cdot \cos\phi - \cos\psi \cdot \sin\phi & 0 & 0 & 0 \\ -\sin\theta & \cos\theta \cdot \sin\phi & \cos\theta \cdot \cos\phi & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \sin\phi \cdot \tan\theta & \cos\phi \cdot \tan\theta \\ 0 & 0 & 0 & 0 & \cos\phi & -\sin\phi \\ 0 & 0 & 0 & 0 & \cos\theta & \cos\theta \end{bmatrix}$$

, $\eta = \begin{bmatrix} X \\ Q \end{bmatrix}$ the the position and orientation of the system in the world frame, and $\nu = \begin{bmatrix} V_r \\ \omega_r \end{bmatrix}$ the velocity and angular speed in the body frame.

3.2 Dynamic model

In order to make the dynamic simulation, a dynamic model is necessary to link the forces at the entry of the system and the kinematic model. As previously said in the section 1, an this dynamic model does not consider the effect of having a very low Reynolds number. Indeed E.M Purcell explains in [13] that at such scale, physics behaves differently. We will approximate this behavior with a strong depreciation factor. In the figure 3, the dynamic model is represented by the function f_d. Since the robot is considered in levitation thanks to the optical tweezers [7], we can apply an aerodynamical model to our robot. Because the robot is constraint by the 4 laser beams, we can neglect the Coriolis forces into our model and thus do not consider the potential sliding effect when the robot is rotating. Then, here is the dynamic model chosen:

$$\begin{bmatrix} F_u \\ F_v \\ F_w \\ \Gamma_p \\ \Gamma_q \\ \Gamma_r \end{bmatrix} = \begin{bmatrix} X_{u|u|} \cdot u|u| + X_{\dot{u}} \cdot \dot{u} \\ X_{v|v|} \cdot v|v| + X_{\dot{v}} \cdot \dot{v} \\ X_{w|w|} \cdot w|w| + X_{\dot{w}} \cdot \dot{w} \\ K_{p|p|} \cdot p|p| + K_{\dot{p}} \cdot \dot{p} \\ K_{q|q|} \cdot q|q| + K_{\dot{q}} \cdot \dot{q} \\ K_{r|r|} \cdot r|r| + K_{\dot{r}} \cdot \dot{r} \end{bmatrix}$$

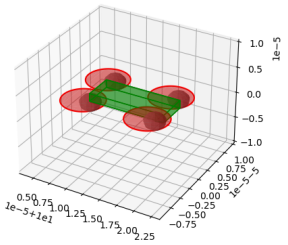
Where $X_{u|u|}$ is the fluid friction coefficient in the direction of u and $X_{\dot{u}}$ is the coefficient representing the inertia of the system in the direction of u. Then, if we reverse this model, we can obtain the derivative of the system's state

in the body frame $\dot{\nu}$:

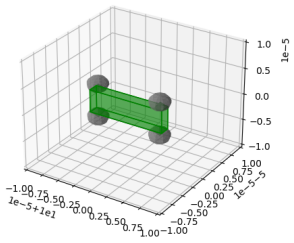
$$\dot{\nu} = \begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \\ \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} \frac{1}{X_{\dot{u}}}(F_u - X_{u|u|} \cdot u|u|) \\ \frac{1}{X_{\dot{v}}}(F_v - X_{v|v|} \cdot v|v|) \\ \frac{1}{X_{\dot{w}}}(F_w - X_{w|w|} \cdot w|w|) \\ \frac{1}{K_{\dot{p}}}(\Gamma_p - K_{p|p|} \cdot p|p|) \\ \frac{1}{K_{\dot{q}}}(\Gamma_q - K_{q|q|} \cdot q|q|) \\ \frac{1}{K_{\dot{r}}}(\Gamma_r - K_{r|r|} \cdot r|r|) \end{bmatrix}$$

Where $F_{(x)}$ and $\Gamma_{(x)}$ are the force and the torque in the x direction. These are computed using the formulas (1), (2), and (3) expressed in the body frame. The coefficients $X_{(x)|(x)|}$, $X_{(\dot{x})}$ can be estimated or experimentally found. In this paper, they will be estimated for the simulation since no experimental data on this particular micro-sized robot was found by the author. However, we know that in such scale, the friction are much bigger than the inertial effects.

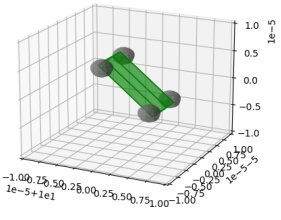
Thus, having the kinematic model and the dynamic model, the simulation can now be implemented. Using the data of the laser, and of the shape of the robot from [7], we can create an accurate simulation with coherent values of the forces induced by the laser beams on the receptors. Figure 4 shows the visual of the robot in the simulation, in different orientations.



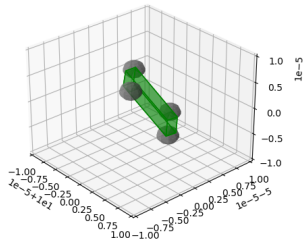
(a) Visual of the simulated robot, with the laser beams



(b) Simulated robot oriented: $\phi = \pi/2$



(c) Simulated robot oriented: $\theta = \pi/4$

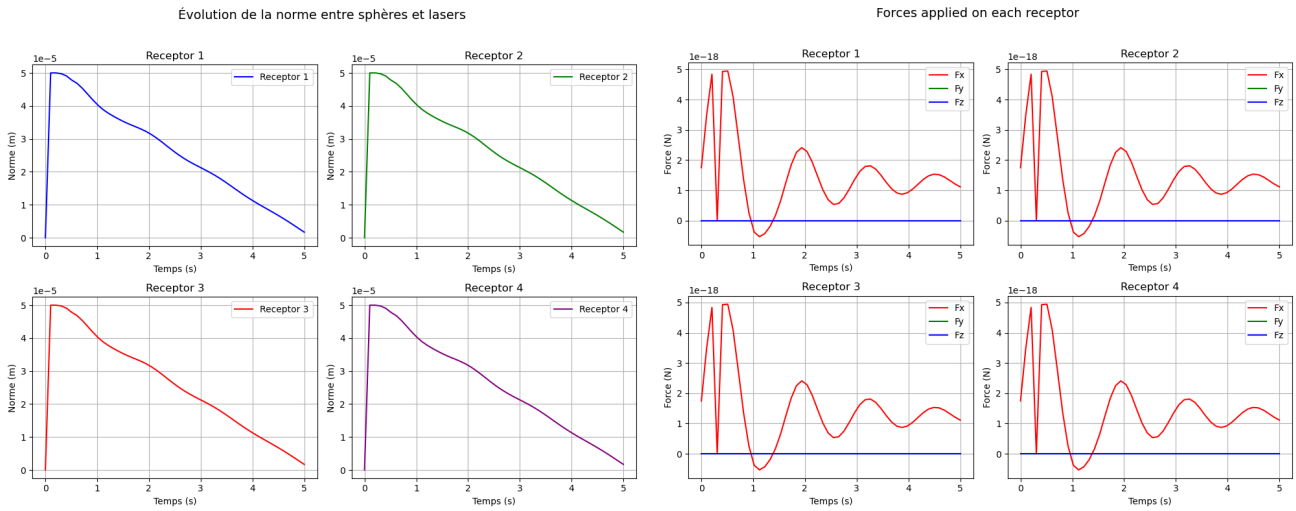


(d) Simulated robot oriented: $\theta = \pi/4$, $\psi = \pi/2$

Figure 4: Simulated robot in different positions

4 Guidance and control

Now that we have a working simulation, it would be interesting to see how it can be controlled. Indeed, first very simple tests shows that the system responds correctly to the laser beams in the simulation, but the response can be optimized. Without guidance and control system, the command introduced in the system are too abrupt making the response oscillating for a long time as it is shown in figure 5.



(a) The norm between each sphere and its corresponding laser beam in function of the time

(b) The forces applied by the laser beams on each corresponding sphere in function of the time

Figure 5: Forces and norm between the laser beams and the spheres in the simulation

Our objective in this section is to remove the oscillation so that the system converges directly to the wanted state, and if it exists, we should try to remove the static error. To understand how the guidance and the control programs impact the system, we should complete figure 3 and add the guidance blocks. To simplify the scheme, all the blocks represented in figure 3 will be transformed in one block named "system" as it is a model of the real system.

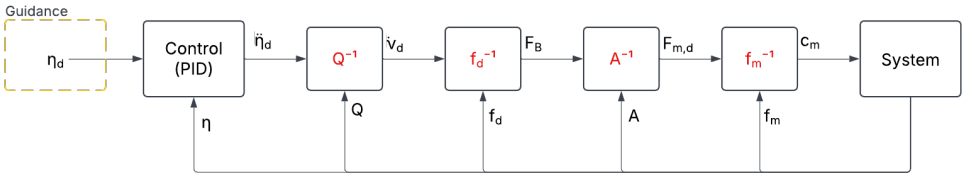


Figure 6: Block diagram representing the implementation of guidance and control in the pipeline [12]

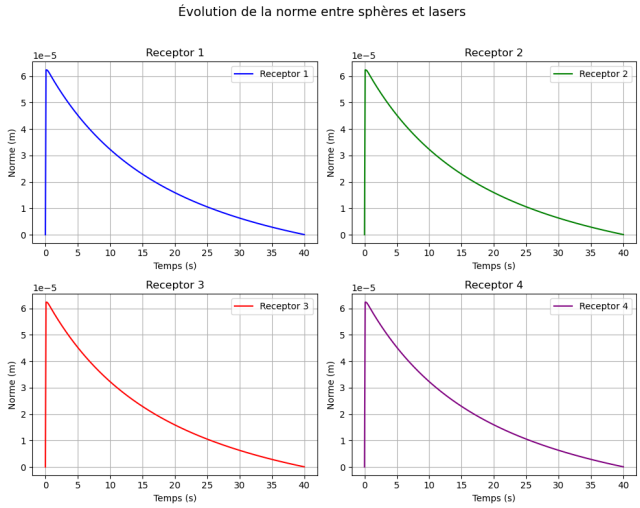
As presented in figure 6, the guidance program gives a desired state of the robot (this can be for example its position and orientation) that will be introduced in the control program. This control program is responsible of removing the oscillation and errors in the response of the robot. The control can be for example a proportional–integral-derivative controller (PID) but other types of controllers exist. In order to test the controller, we will implement a simple stabilization program consisting in reaching a fixed point in the 3D space. However, for now, the third dimension will be omitted since we can split the evolution in 3D in 2 steps:

- **Evolution in the plane** : the robots moves in the 2D space (x, y) to go to its objective;

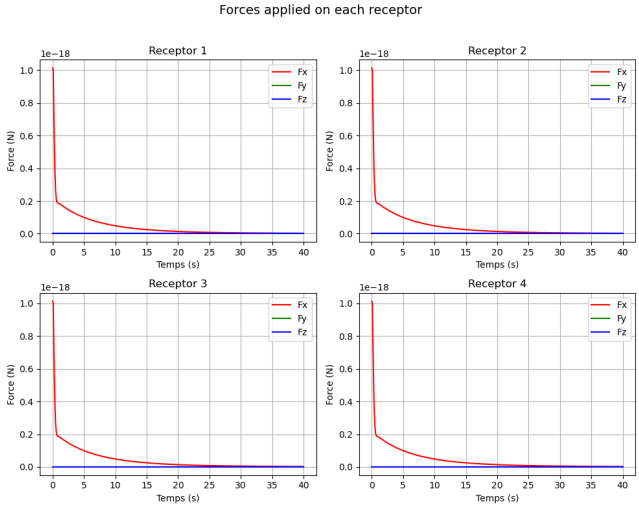
- **Change of plane** : if the robots needs to move according to z axis, then the motion according to (x, y) stops, and only the movement in the z axis is enabled.

Then, in this context, we define the 2D objective this way:
$e_0 = \begin{bmatrix} x_d - x \\ y_d - y \\ \psi_d - \psi \end{bmatrix}$, the error expressed in the global frame, and $e_B = R_{eul} \cdot e_0$, the error expressed in the body frame.
Then, simply we express the wanted state in the body frame like this (in 2 dimensions so ν_d is a 3 dimensional vector):
$$\nu_d = \begin{bmatrix} K_u \cdot tanh(K_x \cdot e_{B,x} + K_{\dot{x}} \cdot \dot{e_{B,x}}) \\ K_v \cdot tanh(K_y \cdot e_{B,y} + K_{\dot{y}} \cdot \dot{e_{B,y}}) \\ tan(K_\psi \cdot e_{B,\psi} + K_{\dot{\psi}} \cdot \dot{e_{B,\psi}}) \end{bmatrix}.$$

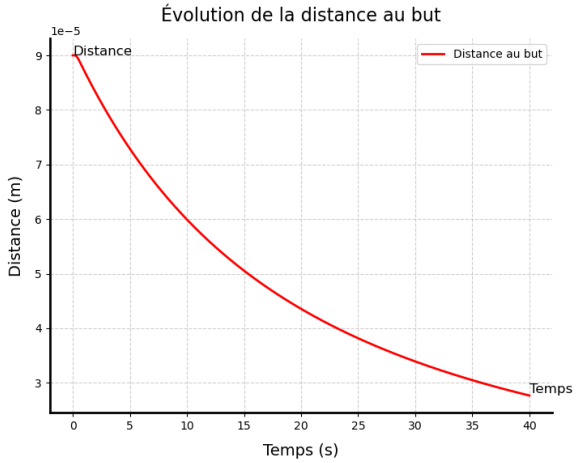
Here we used a simple proportional controller saturated using a *tanh* function, but we could put also a complete PID inside it to be more precise. Here the objective is only to show that any kind of controller could be used with this simulation, quite easily. So, now that we have ν_d, we can follow the chain represented in figure 6 and get to the wanted input.



(a) The norm between each sphere and its corresponding laser beam in function of the time



(b) The forces applied by the laser beams on each corresponding sphere in function of the time



(c) The distance between the objective and the robot in function of the time

Figure 7: Forces and norm between the laser beams and the spheres in the simulation using a guidance program

The figure 7 shows the the same plots as in figure 5 but with the previously explained guidance program implemented. We can see that the oscillations disappeared as expected. However, on plot 7c, we can see that the robot seems to never achieve its objective (I let the simulation during a long time and it never achieves its goal). In order to remove this static error, an integral term should be added in the controller.

5 Conclusion

This work presents a dynamic simulation of the micro-sized robots described in [7]. The ultra-precise control provided by optical tweezers makes these micro-robots highly promising for various applications. However, to fully automate their operation, the development of guidance programs is essential.
In this context, the simulator enables easy testing of guidance algorithms without requiring complex infrastructure or high computational power. It provides a flexible and accessible environment to design and refine control strategies before deploying them in real-world experiments. However, a strong hypothesis has been made when not considering the effects of a low Reynolds number of the environment on the system.

Keywords : Optical Tweezers, Gaussian Beam

References

[1] Eric Diller and M. Sitti. Micro-scale mobile robotics. *Foundations and Trends in Robotics*, 2:143–259, 01 2013. 1

[2] Nicolas Andreff Stéphane Régnier Ali Oulmas, Johan E Quispe. Comparing swimming performances of flexible and helical magnetic swimmers. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2019. 1

[3] D. E. Smalley, E. Nygaard, K. Squire, J. Van Wagoner, J. Rasmussen, S. Gneiting, K. Qaderi, J. Goodsell, W. Rogers, M. Lindsey, K. Costner, A. Monk, M. Pearson, B. Haymore, and J. Peatross. A photophoretic-trap volumetric display. *Nature*, 553(7689):486–490, January 2018. 1

[4] David G. Grier. A revolution in optical manipulation. *Nature*, 424(6950):810–816, August 2003. 1

[5] Carlos Bustamante, Steven B Smith, Jan Liphardt, and Doug Smith. Single-molecule studies of dna mechanics. *Current Opinion in Structural Biology*, 10(3):279–285, 2000. 1

[6] A. Ashkin. History of optical trapping and manipulation of small-neutral particle, atoms, and molecules. *IEEE Journal of Selected Topics in Quantum Electronics*, 6(6):841–856, 2000. 1

[7] Stéphane Régnier Edison Gerena and Sinan Haliyo. High-bandwidth 3d multi-trap actuation technique for 6-dof real-time control of optical robots. *IEEE Robotics and Automation Letters*, 2019. 1, 2, 4, 5, 8

[8] Alioune Badara Diouf, Ferhat Sadak, Edison Gerena, Abdelkrim Mannioui, Daniela Zizioli, Irene Fassi, Mokrane Boudaoud, Giovanni Legnani, and Haliyo Sinan. Robotic sorting of zebrafish embryos. 02 2024. 2

[9] A. Ashkin. Observation of a single-beam gradient force optical trap for dielectric particles. *Optica Publishing Group*, 1986. 2

[10] EdmundOptics. https://www.edmundoptics.com/knowledge-center/application-notes/lasers/gaussian-beam-propagation/?srsltid=AfmBOoovtUaSrxpq3LexgJNhlWgyFS1KVkd7cLDUn9EO4xLVmwiPfsqa. 2

[11] S.KIELICH J.R. LALANNE, A. DUCASSE. *Interaction Laser Molécule: Physique du laser et optique non linéaire moléculaire*. 1994. 2, 3

[12] Lionel Lapierre. *Teaching*. https://www.ensta-bretagne.fr/lapierre/Teaching/. 4, 6

[13] E.M Purcell. Life at low reynolds number. *Physics and our world: a symposium in honor of Victor F. Weisskopf*, 1976. 4

Interval Approach for Marine Robot Localization Without GPS or LIDAR

Salah El Din SEKAR[1]

***Abstract*— This paper addresses the localization of a marine robot in an environment where access to received information is limited, within a critical situation where GPS is unavailable. The robot must localize itself relying on Inertial Measurement Unit (IMU), a camera and a Doppler Velocity Log (DVL), based on pre-defined beacons on a map. To solve this problem, we propose a method based on interval computation [2] and contractors [3] to reduce uncertainties in the various possible regions. This approach is particularly well-suited when the initial position of the robot is unknown. We implement our approach using Codac[1] and evaluate its performance on simulated data**

I. INTRODUCTION

In this paper, we address the problem of robot localization using bearing measurements. We consider the following assumptions:

- The initial position of the robot is known.
- The position of the landmarks is known.
- The landmarks are distinguishable.
- The map is known.

The problem of localization has always been a major challenge in robotics. Numerous solutions exist, each with different implementations, and the choice of method depends on the available resources and the specific constraints of the environment. In cases where GPS is unavailable, maintaining an accurate estimate of the robot's position becomes a critical issue, requiring alternative approaches [5].

To perform localization, we rely on sensor data to estimate the robot's position and orientation. The following measurements are used:

- **Velocity**: Measured using a Doppler Velocity Log (DVL).
- **Bearing angles**: Obtained from cameras and LiDAR.
- **Heading (Yaw/Cap)**: Provided by an Inertial Measurement Unit (IMU).

[1]Salah El Din SEKAR, Msc. in Autonomous Robotics, École Nationale Supérieur de Techniques Avancées

II. MATHEMATICAL FOUNDATION

Let us consider the robot in Figure 1 moving on a plane. We call **bearing** the angle α_i between the axis of the robot and the vector pointing towards the landmark.

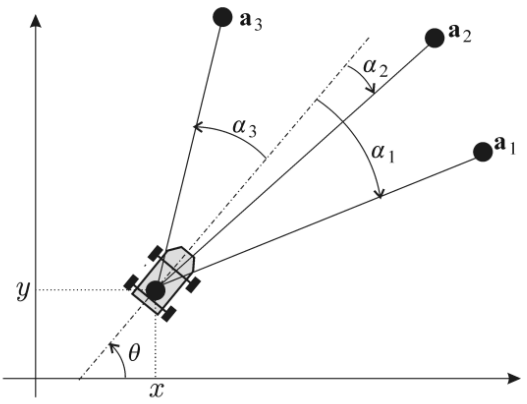


Fig. 1: A robot moving on a plane, measuring angles to locate itself.

Goniometric localization [4] determines the position of a robot by using bearing angles measured from known landmarks. The fundamental equation governing this technique is:

$$(x_i - x)\sin(\theta + \alpha_i) - (y_i - y)\cos(\theta + \alpha_i) = 0 \quad (1)$$

where:

- (x_i, y_i) are the known coordinates of the landmark,
- (x, y) is the unknown position of the robot,
- θ is the robot's heading angle,

- α_i is the measured bearing angle to the landmark.

Given multiple landmarks, the system of nonlinear equations can be solved to estimate the robot's position. However, in real-world scenarios, measurements contain uncertainty due to sensor noise, which can lead to errors in position estimation.

To address this issue, interval analysis is introduced to model the uncertainty by representing bearing measurements as intervals rather than exact values. This ensures a more robust estimation process in the presence of measurement noise [7].

III. Constraint Network

the evolution function and the observation function that are shown in the equations 2 will define the constaraint for the problem we are facing. therefore, the contactors that i will use are related to those constraint

$$\begin{cases} \dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)) \quad \text{(evolution eq.)} \\ \mathbf{g}(\mathbf{x}(t), \mathbf{m}^i) = 0 \quad \text{(observation eq.)} \end{cases} \tag{2}$$

where m_i represents the known position of the landmark

the evolution function that will represnet the dynamic of the robot in my simulation will be presented by the following equations:

$$f(x, u) = \begin{pmatrix} 10\cos(x_3) \\ 10\sin(x_3) \\ u + n_u \end{pmatrix} \tag{3}$$

The observation function represents the measured bearing angle , which can be obtained using various sensors such as cameras and LiDAR

$$g(x, m) = \tan^{-1}\left(\frac{y_i - y}{x_i - x}\right) - \theta \tag{4}$$

those functions can be summed to obtain the following constrains [9] that will be used in our contractor

$$\begin{cases} (i) & \mathbf{v}(\cdot)(t_i) = \mathbf{v}(t_i) \\ (ii) & \mathbf{x}_\theta(\cdot)(t_i) = \theta(t_i) \\ (iii) & \dot{\mathbf{x}}(\cdot) = \mathbf{v}(\cdot) \\ (iv) & \mathbf{p}^i = \mathbf{x}(t_i) \\ (v) & \mathbf{a}^i = \theta(t_i) \\ (vi) & \mathbf{p}^i = \mathbf{g}(\mathbf{y}^i, \mathbf{m}^i) \end{cases} \tag{5}$$

We define the right domain for each variable so we can apply the contractors on them. We define the tubes $[\mathbf{X_{xy}}](\cdot)$, $[\mathbf{X}_\theta](\cdot)$, that contain the trajectories of the position (x, y) and the heading θ. we also define the tubes $[\mathbf{v}](\cdot)$,$[\mathbf{u}](\cdot)$. As for the boxes we have $[\mathbf{m_i}]$,$[\mathbf{p_i}]$, and for the Intervals we have $[\mathbf{y_i}]$,$[\mathbf{a_i}]$.

to solve the problem of those constrains we apply the right contractors on thses domain in an interative way. the procedure stops when the contractors are no more efficient. we we show you next the contractos used for each constrains most of those constrains are similare to the one that exists in the article of data association [10].

- *(i),(ii)*: adds continuous data to a tube at a time t. the contractor used is C_{add_data}. In this case, the tube represents an enclosure of a trajectory of measurements that are given in realtime.
- *(iii)*: this constrain bind the trajectory $[\mathbf{x}](\cdot)$ to the corresponding derivative $[\mathbf{v}](\cdot)$. The related contractor $C_{d/dt}$ allows contractions on the tube [x](·) to preserve only trajectories consistent with the derivatives enclosed in the tube [v](·).
- *(iv),(v)*: $p^i = x(t_i)$ is a constraint that links the vectors p^i and a^i to the evaluation of the trajectory $x(\cdot)$ and $\theta(\cdot)$ at time t_i . A dedicated contractor C_{eval} has been provided. It will allow the correction of the positions of the robot.

As for the final constraint, it represents the bearing constrain and for this one there is no defined contractor to use so we had to create our own contractor that is called gonio contractor. in this following section we will talk about this contractor

IV. Gonio contractor

The contractor enforces the bearing constraints derived from goniometric equations that was mention in the mathematical foundation section. for better understanding, we will take an example of a robot with bearing with two different landmarks. the system will be represeneted by the following matrix equation

$$\mathbf{A}(\theta, \alpha_1, \alpha_2) \cdot \begin{pmatrix} x \\ y \end{pmatrix} = \mathbf{b}(\theta, \alpha_1, \alpha_2, x_1, y_1, x_2, y_2).$$

with:

$$\mathbf{A} = \begin{pmatrix} \sin(\theta + \alpha_1) & -\cos(\theta + \alpha_1) \\ \sin(\theta + \alpha_2) & -\cos(\theta + \alpha_2) \end{pmatrix}$$

and

$$\mathbf{b} = \begin{pmatrix} x_1 \sin(\theta + \alpha_1) - y_1 \cos(\theta + \alpha_1) \\ x_2 \sin(\theta + \alpha_2) - y_2 \cos(\theta + \alpha_2) \end{pmatrix}$$

i.e.,

$$\begin{pmatrix} x \\ y \end{pmatrix} = \mathbf{A}^{-1}(\theta, \alpha_1, \alpha_2) \cdot \mathbf{b}(\theta, \alpha_1, \alpha_2, x_1, y_1, x_2, y_2).$$

This system is solved iteratively using backward propagation, which reduces uncertainty in position estimation. we will show you an example of a static situation where we have a robot in a position that is given

To illustrate the application of goniometric localization, we present an example in a **static scenario**. We consider a **robot** with a true state of:

$$x_{\text{truth}} = (-3, -2, \frac{\pi}{4})$$

where the robot's position is unknown. However, we assume that the robot is equipped with sensors that measure the **bearing angles** from known **landmarks**.

The **two landmarks** in this scenario are located at: $(0, 6)$ and $(4, 0)$

Since measurements in real-world applications are subject to **uncertainties**, we introduce **error intervals** to account for potential inaccuracies:

TABLE I: Uncertainties on Data Used for the example.

Data	Description	Uncertainty	Unit
m^i	Landmark position	$[-0.05, 0.05]$	m
y^i	Measured bearing angle	$[-0.05, 0.05]$	rad
θ	Heading angle	Known	rad
(x, y)	Robot position	Unknown	m

Given these uncertainties, we apply the **Goniometric Contractor** to estimate the **robot's position** using **only bearing measurements**. The estimation is performed using a **paving algorithm**, specifically **SIVIA (Set Inversion Via Interval Analysis)**, as implemented in the **CODAC library**.

Below, we provide an illustration of this example, showing the **computed position estimations** based on the interval constraints.

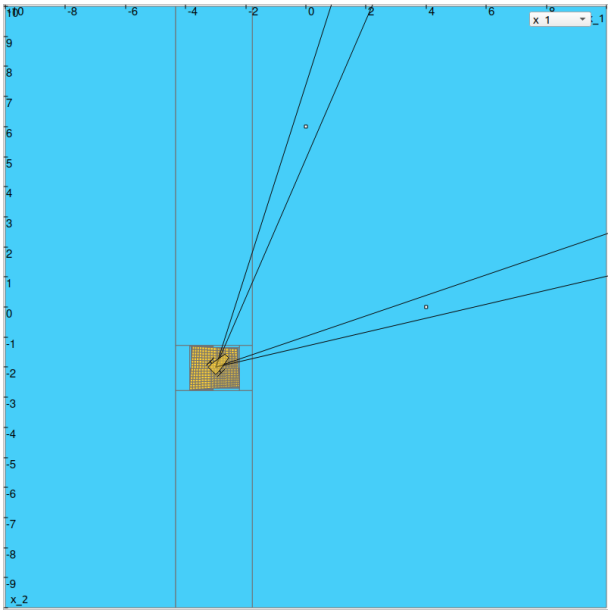


Fig. 2: Illustration of the Goniometric Localization Example using SIVIA [8] Algorithm.

V. Application

Now that we have explored all the constraints necessary to solve our localization problem and demonstrated the use of the goniometric contractor, we will move on to solving a real-time dynamic localization problem.

In this scenario, we assume that the robot starts with a known initial position but eventually loses its localization. Our goal is to use the previously defined constraints to continuously track and refine the robot's estimated position over time.

To achieve this, we will introduce landmarks on the map that the robot can detect and identify. Using bearing measurements, we will progressively refine the robot's estimated position, improving localization accuracy as more landmarks are observed.

This simulation will demonstrate how we can apply the goniometric contractor in a real-time

setting, integrating dynamic state updates, sensor measurements, and interval-based reasoning to maintain an accurate estimate of the robot's position. Finally, we will present an example to illustrate the approach in action.

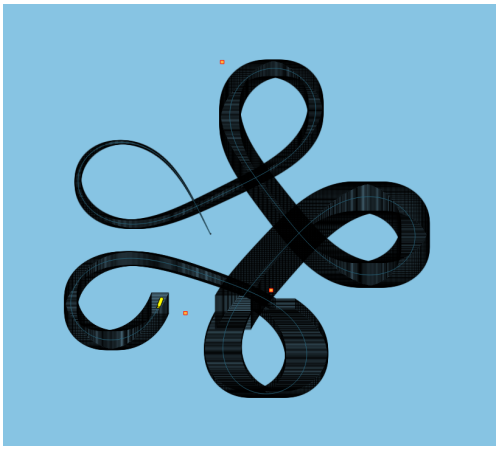
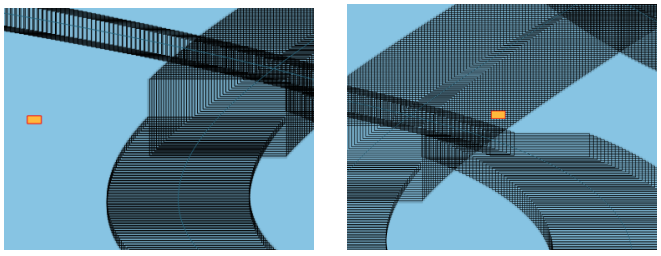


Fig. 3: Illustration of a live contraction for the position of the robot using the network that was defined at the start.

In the example, we assume that the initial position is known with $x_{\text{truth}} = (0, 0, 2)$ and the evolution function is as defined in Equation 3 In the example we shown orange boxes represnts the landmarks that are predefined on our map and they are precieved as follows:

TABLE II: Landmark Positions and Their Time of Perception

Time (t)	**Landmark Position** m^i
$t = 5$	$(1, 15)$
$t = 10$	$(-2, -7)$
$t = 12$	$(5, -5)$



(a) (b)

Fig. 4: Illustration of goniometric contraction for an evolving system.

in the 2 figures we zoom in to 2 landmarks that gives a good data for our contractor network to gives more precision for our location, we can see after the perception of each landmark how the position box is being contracted gives a better result that englob the real position of our system, and the boxes with black borders

VI. Conclusion

In this work, we demonstrated how goniometric localization combined with interval analysis provides a robust approach to estimating a robot's position under uncertainty. By leveraging contractor-based methods, we ensured that the feasible region for the robot's location was rigorously bounded, unlike traditional probabilistic approaches. This method offers guaranteed consistency, resilience to measurement noise, and computational efficiency compared to classical techniques like Extended Kalman Filters (EKF) [12] or Particle Filters (PF) [11]. The results highlight the potential of interval-based localization as a reliable alternative for real-world robotic applications, where precise state estimation is crucial. In future work, we aim to apply this method in scenarios where the initial position of the robot is unknown, no LiDAR is available for localization, and the robot must identify landmarks autonomously.

References

[1] https://codac.io/

[2] C. Jermann, G. Trombettoni, B. Neveu, and P. Mathis. Decomposition of geometric constraint systems: a survey. International Journal of Computational Geometry & Applications, 16(05n06):379–414, 2006.

[3] Chabert, G., Jaulin, L. "Contractor Programming." Artificial Intelligence Journal, 2009.

[4] Jaulin, L. "kalmooc" https://www.ensta-bretagne.fr/jaulin/kalmooc.html

[5] L. Jaulin. Mobile Robotics. ISTE editions, 2015.

[6] Briechle, K., & Hanebeck, U. D. (2002). Localization of a Mobile Robot using Relative Bearing Measurements. IEEE Transactions on Robotics and Automation, 18(6), 963–972.

[7] Jaulin, Luc & Walter, Eric & Meizel, Dominique. (2000). Robust Autonomous Robot Localization Using Interval Analysis. Reliable Computing. 6. 337-362. 10.1023/A:1009990700281.

[8] V. Drevelle and P. Bonnifait. Localization confidence domains via set inversion on short-term trajectory. IEEE Transactions on Robotics, 29:1244–1256, 2013.

[9] Rocca, P., Anselmi, N., Benoni, A., & Massa, A. (2021). Probabilistic Interval Analysis for the Analytic Prediction of the Pattern Tolerance Distribution in Linear Phased Arrays With Random Excitation Errors. arXiv preprint arXiv:2102.02255

[10] Rohou, Simon, Benoît Desrochers, and Luc Jaulin. "Set-membership state estimation by solving data association." 2020 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2020.

[11] D. Talwar and S. Jung, "Particle Filter-based Localization of a Mobile Robot by Using a Single Lidar Sensor under SLAM in ROS Environment," 2019 19th International Conference on Control, Automation and Systems (ICCAS), Jeju, Korea (South), 2019, pp. 1112-1115, doi: 10.23919/ICCAS47443.2019.8971555. keywords: ROS;localization;autonomous driving;AMCL;SLAM,

[12] Eman, Alhamdi, and Hedjar Ramdane. "Mobile robot localization using extended Kalman filter." 2020 3rd International Conference on Computer Applications Information Security (ICCAIS). IEEE, 2020.

Basile Mollard
Ensta Bretagne
Etudiant en robotique

Positionnement par mesures goniométrique

Résumé

This paper addresses the localization of a boat using only its camera. In military contexts, the first action in the event of war between major powers would be to destroy the adversary's satellites, thus disabling GNSS-based localization. To counter this, alternative methods for localizing military assets must be developed. This work employs goniometry, using angular measurements between notable objects and the vehicle's heading for localization. The camera serves as the sensor for detecting and distinguishing objects, though object detection itself is beyond the scope of this paper. Once the angles are obtained, interval theory and the CODAC [1] library developed by ENSTA Bretagne are used for localization. Interval theory offers a rigorous approach to handling measurement uncertainties, which is particularly useful when working with noisy sensor data. The CODAC library, which manipulates intervals and multidimensional boxes, allows for reliable uncertainty propagation. Additionally, a goniometric contractor is introduced to refine localization by reducing uncertainty bounds while maintaining the inclusion of all valid solutions. This method is integrated with velocity-based localization when no objects are detectable, utilizing a DVL (Doppler Velocity Log) to measure the boat's speed. The combination of goniometric contractors and velocity integration provides a robust localization solution even in the absence of GNSS signals, as demonstrated through simulations. However, challenges such as the necessity of an initial position and the degradation of accuracy over time suggest the need for a SLAM algorithm to further refine the boat's location estimation.

Introduction

Dans ce papier, nous allons aborder la localisation d'un bateau avec seulement l'aide de sa caméra. En effet, dans le cadre militaire, la première action, en cas de guerre entre deux grandes puissances, sera de détruire les satellites adverses et donc la localisation GNSS. Dans ce contexte, il est essentiel de développer d'autres méthodes afin de localiser les différentes armes de notre armée. Ici, nous utiliserons donc la goniométrie, c'est-à-dire que nous nous aiderons de la mesure des angles pour nous localiser. Nous mesurons les angles entre les différents objets remarquables et le cap du véhicule, et pour cela, nous utiliserons comme capteur la caméra. Il faut donc un algorithme pour détecter et différencier les objets, mais cela n'est pas le sujet de ce papier. Enfin, une fois les angles obtenus, nous utiliserons la théorie des intervalles, des contracteurs et la librairie CODAC associée, développée par l'ENSTA Bretagne.

Théorie des intervalles

Comme dit précédemment, nous utiliserons la théorie des intervalles. Cette dernière est une approche mathématique permettant de manipuler des incertitudes et des erreurs de mesure de manière rigoureuse. Contrairement à l'arithmétique classique, où les valeurs sont représentées par des nombres réels précis, la théorie des intervalles représente les grandeurs par des bornes inférieure et supérieure,

formant ainsi un intervalle contenant la valeur réelle inconnue. Cette approche est particulièrement utile dans notre cas, car nos données sont entachées d'incertitudes dues à des erreurs de capteurs, des approximations numériques ou des phénomènes non modélisés. En goniométrie, l'utilisation de la théorie des intervalles permet d'améliorer la robustesse des méthodes de localisation en tenant compte des incertitudes angulaires et positionnelles, garantissant ainsi que la solution obtenue est cohérente avec l'ensemble des mesures disponibles.

Afin d'utiliser cette théorie, nous nous aiderons de la bibliothèque CODAC. Cette librairie utilise, premièrement, les intervalles, qui sont représentés sous la forme d'ensembles bornés $[x_{\min}, x_{\max}]$, où $x_{\min}$ et $x_{\max}$ définissent respectivement les limites inférieure et supérieure de l'incertitude sur une grandeur donnée. CODAC permet également de manipuler des boîtes, qui sont des ensembles multidimensionnels formés de plusieurs intervalles, facilitant ainsi la modélisation des incertitudes sur plusieurs variables simultanément. Grâce à ces outils, il est possible d'effectuer des opérations d'intersection, d'union et de propagation d'incertitudes de manière fiable et rigoureuse.

Enfin, un contracteur est un opérateur de la librairie CODAC permettant de réduire un ensemble d'intervalles tout en garantissant que la solution correcte demeure incluse dans l'ensemble réduit. Il repose sur le principe de contraction des domaines définis par des contraintes, ce qui est particulièrement utile pour traiter des problèmes de localisation et de filtrage en présence d'incertitudes. Un contracteur agit donc comme un filtre en supprimant les valeurs impossibles sans exclure les solutions valides.

Contracteur goniométrique [2]

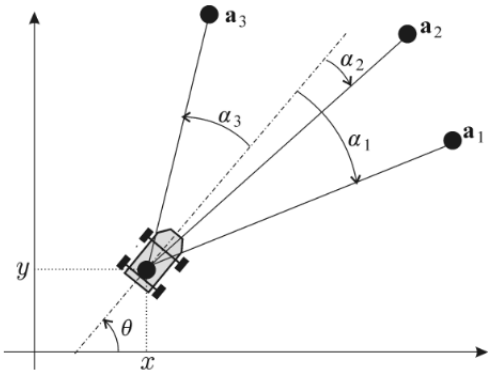


FIGURE 1 – Véhicule sur un plan mesurant des angles pour se localiser

Sur ce schéma, on voit trois bouées repérées à l'aide d'angles par rapport au cap du bateau. On utilisera seulement deux d'entre elles.

Pour commencer, nous avons les vecteurs suivants :

$$\begin{bmatrix} x_i - x \\ y_i - y \end{bmatrix} \text{ et } \begin{bmatrix} \cos(\theta + \alpha_i) \\ \sin(\theta + \alpha_i) \end{bmatrix}$$

qui sont colinéaires, donc on a :

$$\det\left(\begin{bmatrix} x_i - x \\ y_i - y \end{bmatrix}, \begin{bmatrix} \cos(\theta + \alpha_i) \\ \sin(\theta + \alpha_i) \end{bmatrix}\right) = 0$$

On a finalement :

$$(x_i - x)\sin(\theta + \alpha_i) - (y_i - y)\cos(\theta + \alpha_i) = 0$$

Donc d'après la relation suivante on obtient ce système :

$$\begin{cases} (x_1 - x)\sin(\theta + \alpha_1) - (y_1 - y)\cos(\theta + \alpha_1) & = 0 \\ (x_2 - x)\sin(\theta + \alpha_2) - (y_2 - y)\cos(\theta + \alpha_2) & = 0 \end{cases}$$

On peut donc mettre le problème sous la forme matricielle :

$$AX = b$$

avec :

$$A = \begin{bmatrix} \sin(\theta + \alpha_1) & -\cos(\theta + \alpha_1) \\ \sin(\theta + \alpha_2) & -\cos(\theta + \alpha_2) \end{bmatrix}$$

et

$$b = \begin{bmatrix} y_1\cos(\theta + \alpha_1) - x_1\sin(\theta + \alpha_1) \\ y_2\cos(\theta + \alpha_2) - x_2\sin(\theta + \alpha_2) \end{bmatrix}$$

Enfin, il nous reste plus à implémenter ceci avec l'aide de la version 2 de codac :

```python
def contract(self, x):
    n = len(self.m)
    A = c2.IntervalMatrix(n, 2) #c2 est codac2
    b = c2.IntervalVector(n)
    p = c2.IntervalVector([x[0], x[1]])

    for i in range(n):
        A[i, 0] = c2.sin(x[2] + self.y[i])
        A[i, 1] = -c2.cos(x[2] + self.y[i])
        b[i] = self.m[i][0] * c2.sin(x[2] + self.y[i])
        - self.m[i][1] * c2.cos(x[2] + self.y[i])

    c2.MulOp.bwd(b, A, p)
    x.put(0, p)
    return x
```

FIGURE 2 – Implémentation du contracteur en python et avec codac2

Repérage sans observation

Maintenant que nous avons ce contracteur, nous pouvons nous localiser lorsque nous observons au moins deux bouées dont nous connaissons la position. Cependant, il est fort probable qu'en mer, il y ait des moments où l'on ne détecte aucun objet. Or, dans notre configuration, dès que l'on ne voit rien sur les caméras, nous ne pouvons pas nous situer dans l'espace. Ainsi, nous allons utiliser le cap

et la vitesse pour nous localiser en attendant de revoir des objets remarquables. En effet, nous pouvons facilement mesurer la vitesse d'un bateau sans satellite grâce au DVL. Le DVL est un capteur de vitesse qui utilise l'effet Doppler avec le fond marin pour mesurer la vitesse d'un robot.

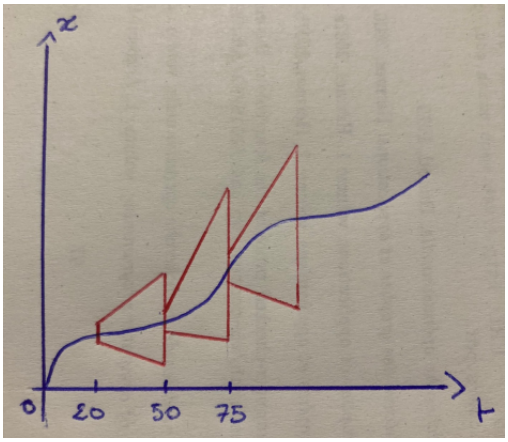


FIGURE 3 – Illustration de la méthode

Sur ce schéma est représentée la position en x en fonction du temps. On suppose qu'aux temps 20, 50 et 75, on utilise le contracteur présenté ci-dessus. Entre-temps, on intègre la vitesse projetée en x et y pour continuer à se localiser.

Ainsi, il faut définir une façon d'intégrer la vitesse afin de conserver l'avantage des intervalles. On pourrait utiliser la méthode des rectangles, par exemple, mais on perdrait alors le côté pessimiste des intervalles. En effet, ce côté pessimiste nous permet d'assurer que la position réelle du robot se trouve bien dans l'intervalle trouvé. C'est pour cela que nous allons faire l'union [3] de la vitesse et du cap du robot à chaque instant. Puis, toutes les secondes, nous allons intégrer cette union de vitesses selon x et y. Ainsi, en formalisant, cela donne à chaque instant :

$$\begin{cases} v_{rslice} = v_{rslice} \cup v_t \\ cap_{slice} = cap_{slice} \cup cap_t \end{cases}$$

Et toutes les secondes nous projetons la vitesse sur les axes x et y :

$$\begin{cases} v_{xslice} = v_{rslice} * \cos(cap_{slice}) \\ v_{yslice} = v_{rslice} * \sin(cap_{slice}) \end{cases}$$

Ensuite on regroupe dans une boxe v_{slice} les deux vitesses. Et puis on integre :

$$p_{slice} = p_{slice} + dt \cdot v_{slice}$$

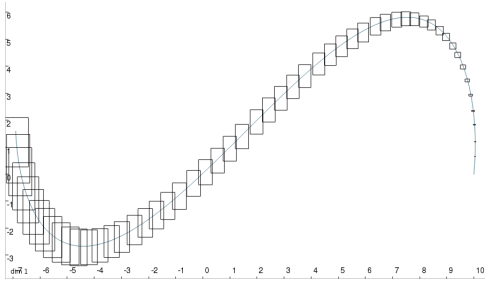


FIGURE 4 – Implémentation de l'intégration de la vitesse

Si on implémente cette méthode pour visualiser ce que cela donne sur une trajectoire quelconque, on obtient la figure ci-dessus. La ligne bleue est la trajectoire théorique, tandis que les différentes boîtes représentent les positions du robot avec l'intégration de la vitesse. On voit donc que la boîte grossit en fonction du temps tout en gardant la trajectoire théorique en son sein. On vient, au final, de reconstituer les tubes de CODAC.

L'avantage de cette méthode par rapport aux tubes est qu'ici, on n'a pas à décider d'un temps de mission. De plus, on n'est pas obligé de garder en mémoire les intervalles passés.

Union des deux méthodes

Maintenant, il faut regrouper le contracteur goniométrique et l'intégration de la vitesse. Pour cela, nous décidons qu'au moment d'intégrer la vitesse, nous vérifions si des observations d'objets remarquables sont disponibles. Si c'est le cas, nous contractons la boîte x_{slice} avec le contracteur goniométrique.

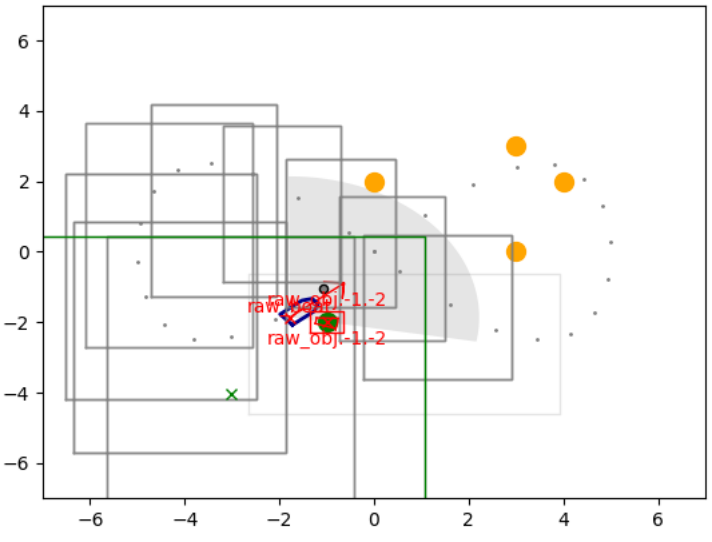


FIGURE 5 – Simulation finale

Sur cette simulation [4], on voit le bateau ainsi que les bouées qu'il peut observer. On dis-

tingue également les boxes de localisation du robot : celles en gris correspondent aux estimations passées, tandis que celle en vert représente l'estimation actuelle. On observe que les boxes suivent la trajectoire du robot tout en se contractant lorsque celui-ci perçoit les bouées. À l'inverse, elles s'élargissent lorsque le bateau est en aveugle et que l'estimation repose uniquement sur la vitesse et le cap.

Conclusion

Pour conclure, nous avons mis au point un contracteur goniométrique qui collabore avec une intégration de la vitesse. Ceci permet de se localiser dans son repère en cas de perte de signal GNSS. Grâce à une simulation, nous pouvons valider notre implémentation et notre théorie.

Cependant, cette méthode comporte plusieurs limites. Tout d'abord, une position initiale est nécessaire pour se repérer dans le monde ainsi que de connaitre celles des amers. Une solution serait de considerer seulement la distance qui sépare le bateau et les bouées [5]. De plus, en mer, les amers sont peu nombreux, ce qui oblige à s'appuyer principalement sur la vitesse. Le problème est qu'après un long laps de temps, la précision de la position du bateau se dégrade considérablement. Il serait donc nécessaire de mettre en place un algorithme de SLAM afin d'affiner l'estimation de la localisation du robot.

Références

[1] https://codac.io/ ENSTA.

[2] *Kalmooc* Luc Jaulin. 2023.

[3] *Analyse par intervalles pour la détection de boucles dans la trajectoire d'un robot mobile* Clément Aubry. 2014.

[4] *https ://github.com/AA-Katsrg/Guerledan-projet-gonio* 2025.

[5] *Pure range-only slam with indistinguishable marks. Constraints, 21(4) :557–576,* Luc Jaulin. 2016.

Vidéo youtube du simulateur final, utilisant l'intégration de la vitesse et le contracteur gogniométrique :

https://youtu.be/S2m_YWOU58o

Metric-Semantic Localization and Mapping: Exploring the Integration of Semantics in Visual SLAM

Barbarit–Gaboriau Simon
Autonomous Robotics
ENSTA Bretagne
Brest, France

***Abstract*—This paper explores the integration of semantic information into traditional visual simultaneous localization and mapping (SLAM) algorithms, aiming to enhance environmental understanding and improve robot perception. We begin with a detailed examination of the standard visual SLAM pipeline, outlining its key components and challenges. Next, we introduce the concept of semantic visual SLAM, where semantic segmentation networks are incorporated into the SLAM framework to enable object recognition and scene understanding. This integration enhances localization accuracy and mapping efficiency, particularly in dynamic and unstructured environments. Finally, we compare visual SLAM and semantic visual SLAM using public datasets and libraries. By analyzing the benefits, implementation challenges, and potential applications of semantic integration, this paper provides insights into the future of intelligent localization and mapping systems.**

***Index Terms*—Computer vision, SLAM, visual SLAM, robotics**

I. Introduction

Simultaneous Localization and Mapping (SLAM) is a fundamental capability for robotics in general, enabling robots to construct maps of their surroundings while determining their own position within those maps. Visual SLAM (vSLAM), which relies on camera-based perception, has become a widely used approach due to its cost-effectiveness and ability to work in GPS-denied environments. Traditional vSLAM methods primarily focus on geometric features such as points, lines, and planes to represent the environment. However, these purely geometric maps lack higher-level scene understanding, which can limit their effectiveness in complex, dynamic environments where recognizing objects and semantic elements is crucial.

To address this limitation, Semantic Visual SLAM (Semantic vSLAM) has emerged as an extension of traditional vSLAM by incorporating semantic information into the localization and mapping process. By leveraging deep learning-based semantic segmentation, Semantic vSLAM allows robots to not only localize themselves but also recognize and label objects in the environment, leading to richer and more informative maps. This integration enhances navigation, decision-making, and interaction with dynamic and unstructured scenes, making it particularly useful in applications such as autonomous driving, service robotics, and augmented reality.

This paper explores the incorporation of semantics into vSLAM and its impact on localization and mapping. We begin with a comprehensive review of the standard visual SLAM pipeline, outlining its key components and challenges. Next, we introduce Semantic vSLAM and discuss how semantic segmentation can be integrated into the SLAM framework. Finally, we provide a comparative analysis of vSLAM and Semantic vSLAM using public datasets and libraries to evaluate their performance and advantages. Through this exploration, we highlight the potential of semantic integration in advancing the field of intelligent mapping and localization systems.

II. The traditional visual SLAM pipeline

In monocular visual SLAM, there are two main approches which are the EKF approach and the Keyframe approach with the latter being more accurate given the same computational effort [1]. We will then focus on the Keyframe approach, illustrating it using a simplified ORB-SLAM C++ code on the TUM-RGBD dataset.

The complete and simplified ORB-SLAM systems are in figure 1:

The simplified version simply focuses on two main aspects: Tracking and Mapping.

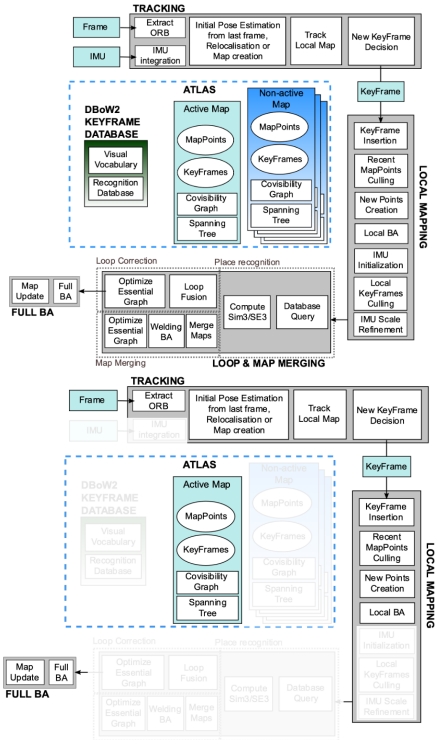


Fig. 1. Complete and simplified ORB-SLAM systems [2]

A. Tracking

As its name implies, the tracking thread's end goal is to track the position of the camera. However, doing so requires feature extraction and matching, as well as camera calibration.

a) Feature extraction: Feature extraction is done using the ORB (Oriented FAST and Rotated BRIEF) [4] algorithm, which is a combination of the FAST (Features from Accelerated Segment Test) [3] feature extractor and the BRIEF (Binary Robust Independent Elementary Features) [5] feature descriptor.

The FAST algorithm finds a corner by identifying a pixel p surrounded by n consecutive pixels, all brighter (or darker) than p by a threshold t:

$I_i \geq I_p + t$ (brighter region) or $I_i \leq I_p - t$ (darker region)

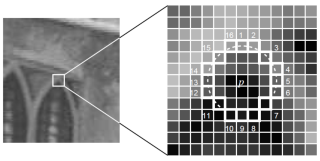


Fig. 2. FAST corner detector [3]

The original FAST algorithm evaluates all 16 pixels in the circular neighborhood, which can be computationally expensive. To improve efficiency, an optimized approach leverages a decision tree classifier to reduce the number of intensity checks required for corner detection.

To minimize unnecessary computations, an initial check is performed on four strategically selected pixels at positions 1, 5, 9, and 13 in the circle. If at least three out of these four pixels fail to meet the corner condition, the evaluation is terminated early, eliminating the need for further intensity comparisons.

To further enhance performance, a machine learning-based decision tree classifier is trained using real-world corner data. Instead of systematically evaluating all 16 pixels, the classifier predicts whether a candidate pixel is a corner using a reduced number of comparisons. This approach significantly accelerates the corner detection process while maintaining detection accuracy.

Once the feature is found, the SLAM algorithm stil needs to give it a vector called a descriptor to identify it. This is done using the BRIEF descriptor which compares the intensity of pixels in the immediate vincinity of the feature using a diagram. The descriptor is then created as a binary string containing these comparisons:

$$d_p = \{b_1, b_2, \ldots, b_k\} \text{ with } b_i = \begin{cases} 1 & \text{if } I(p_i) > I(p_j) \\ 0 & \text{else} \end{cases}$$

With p the feature, I the light intensity and p_i the pixels adjacent to p. With this, the feature extraction is done:

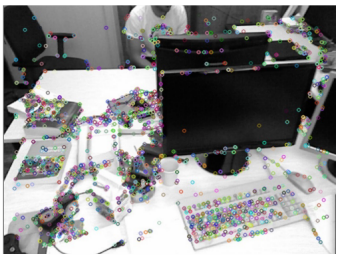


Fig. 3. Feature extraction of the simplified ORB

b) Feature matching: Feature matching in ORB is a critical step in establishing correspondences between keypoints (features) detected in consecutive frames. It involves comparing the extracted binary descriptors between two sets of keypoints, typically from the current and previous frames. The binary descriptors are compared using the Hamming distance, which measures the number of differing bits between two binary strings. Specifically, given two descriptors d_1 and d_2, the Hamming distance is defined as:

$$\text{Hamming distance}(d_1, d_2) = \sum_{i=1}^{k} |b_{1i} - b_{2i}|$$

where k is the number of bits in each descriptor, and b_{1i} and b_{2i} are the individual bits of descriptors d_1 and d_2, respectively. The lower the Hamming distance, the more similar the two descriptors, indicating a match between the corresponding features.

For example, consider two binary descriptors:

$$d_1 = \{1, 0, 1, 1, 0, 1\}, \quad d_2 = \{0, 0, 1, 0, 0, 1\}$$

The Hamming distance between d_1 and d_2 is calculated by counting the number of differing bits:

$$\text{Hamming distance}(d_1, d_2) = 2$$

Thus, the Hamming distance between these two descriptors is 2, indicating that they differ in two bit positions.

There are two primary methods for performing feature matching in ORB: brute force matching and guided matching.

1. Brute force matching: In this method, each descriptor from the current frame is compared to every descriptor in the previous frame using the Hamming distance. This approach, while simple, can be computationally expensive, especially for large sets of descriptors.

2. Guided matching: To improve efficiency, ORB often uses more advanced matching techniques, such as FLANN (Fast Library for Approximate Nearest Neighbors). These methods reduce the computational complexity by indexing the descriptors and efficiently finding the closest matches, improving both speed and accuracy.

In our implementation, we decided to use a guided matching algorithm that is a direct improvement over the brute force approach. Instead of finding the shortest Hamming distance for a keypoint, we select the two best corresponding keypoints by Hamming distance and then add the match with the feature having the shortest distance only if the second-best match is far enough (using a simple threshold on the percentage of the distance). In case the second-best match is too close to the first, we resolve this uncertainty by detecting and removing this keypoint:

Fig. 4. Matching between the first two frames for a treshold of 0.9 and 0.7 respectively

Before doing the camera pose tracking, it is important to calibrate our camera by establishing its intrinsic parameters. The general equation for camera calibration is based on the pinhole camera model, where the 3D world coordinates are projected into 2D image coordinates.

The projection equations are given as:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Where:

- (X, Y, Z) are the 3D coordinates of a point in space.
- (u, v) are the 2D image coordinates.
- f_x and f_y are the focal lengths of the camera along the x- and y-axes, respectively.
- c_x and c_y represent the principal point (the point where the optical axis intersects the image plane).
- $\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$ is the calibration matrix

Once the initial matches are established and the camera is calibrated, the algorithm performs pose estimation using the matches. Pose estimation involves determining the relative camera motion (rotation and translation) between consecutive frames within one scale factor, typically using RANSAC (Random Sample Consensus) to robustly estimate the transformation while rejecting outliers. RANSAC is an iterative algorithm that selects random subsets of correspondences, computes a potential transformation for each subset, and evaluates its quality based on the consensus set (i.e., how many correspondences agree with the transformation). By iterating and refining the transformation based on the largest consensus, RANSAC effectively handles the presence of noisy or incorrect matches, ensuring that the pose estimate is robust to outliers. This process is part of short-term data association (DA), where the goal is to correct the current pose estimate based on the matches from the previous frame, enabling real-time tracking of the camera.

In addition to short-term tracking, ORB-SLAM also performs mid-term data association (DA) by tracking the local map over several frames. The local map is composed of keyframes and their corresponding 3D points. During mid-term DA, the system refines the current camera pose by adjusting it relative to this local map, ensuring consistency over time. Pose correction is applied to maintain the accuracy of the camera trajectory in the global map.

Both short-term and mid-term tracking benefit from several optimization techniques to improve their accuracy and robustness. The key optimizations include:

- Bundle Adjustment: A global optimization technique that refines the camera poses and 3D point positions by minimizing the reprojection error across all frames and keypoints.

- Loop closure detection: A mechanism to detect when the camera revisits a previously mapped area, triggering global pose correction and reducing drift in the map over time. We will not be implementing this optimization method in the simplified ORB-SLAM however.

These optimizations, particularly bundle adjustment, help maintain accurate and robust tracking of the camera's trajec-

tory, enabling ORB-SLAM to perform effectively in real-time applications while also providing precise 3D mapping.

B. Mapping

In ORB-SLAM, the map has two elements: map points which are 3D landmarks and keyframes which are the most important frames.

First, we need to choose a criterion to add keyframes to the map. The most important ones are: the minimum number of frames since the last keyframe that we set at 5 and the number of features matched with the last keyframe which must be greater than 30. This criterion is lax because we want to insert a lot of keyframes to make the tracking robust to fast camera motions, and redundant keyframes will be removed later.

Once a keyframe is added to the map, it brings with it new information, we must therefore add the new corresponding map points. The keypoints can only become map points if they are matched with points from previous keyframes. However, before doing this matching, we first need to compute the essential matrix between the new keyframe and the older one we are trying the matches on:

$$\mathbf{E} = \mathbf{t}_{\times}\mathbf{R} \tag{1}$$

Where:
- E is the essential matrix
- R is the rotation matrix
- t is the translation vector
- $\mathbf{t}_{\times}$ is the antisymmetric matrix (or cross-product matrix) associated with t, defined as :

$$\mathbf{t}_{\times} = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix} \tag{2}$$

This new keyframe (kf1) is then compared with co-visible older ones (kf2) in order to match keypoints and triangulate new map points. For each keypoint from kf1, we look for a match among kf2's keypoints using the guided matching approach described earlier. Using this approach, we can count the number of matches between the current keyframe and the older ones. We then triangulate new map points using the keyframes that have at least 20 matching keypoints.

Once a match is found, we check that the keypoint in kf2 is not associated with a map point before computing the normalized projection ray of both keypoints and verifying the coplanarity constraint using the essential matrix. Thanks to the two projection rays, we can then solve the triangulation equation and obtain the 3D coordinates of the map points.

However, before adding the latter to the map, we establish a few tests that it must pass to reduce the number of corrupted map points:

- Parallax angle test: We check the parallax angle between both projection rays, defined as:

$$\theta = \cos^{-1}\left(\mathbf{x}_1 \cdot \mathbf{x}_2\right) \tag{3}$$

After testing, we found that if the parallax is less than 1 rad, it means that the camera did not rotate enough between the two frames, making the depth measurement unreliable.

- Depth consistency check: The 3D point must be correctly positioned in front of the camera in both cases.

- Reprojection error test: The 3D point is reprojected into the images and compared with its actual position to compute the reprojection error. If this error is too high, the point is not added.

$$e = \left\|\mathbf{u} - K\begin{bmatrix} R & t \end{bmatrix}\mathbf{X}\right\|^2 \tag{4}$$

C. Map managing and Evaluation

To make our algorithm lighter, we need to implement a map points and keyframes culler:

- A MapPoint is considered unused if it has not been seen in the 2 following KeyFrames after its triangulation or if it has not been matched in at least 25 frames that should see the point

- A KeyFrame is considered redundant if at least a 90% of its matched MapPoints are seen by at least 3 other KeyFrames in the local map.

To evaluate our SLAM algorithm we decided to use the Root Mean Square Error (RMSE) Absolute Translation Error (ATE). The RMSE is a commonly used metric in visual odometry and SLAM (Simultaneous Localization and Mapping) to evaluate the accuracy of estimated trajectories compared to ground truth trajectories.

RMSE ATE measures how far the estimated trajectory is from the ground truth trajectory in terms of absolute position differences. It quantifies the global drift of the trajectory without considering orientation errors.

Given:

- $P = \{p_1, p_2, ..., p_N\}$ as the estimated trajectory (3D positions).
- $Q = \{q_1, q_2, ..., q_N\}$ as the ground truth trajectory (3D positions).

The Absolute Translation Error (ATE) for each timestamp is:

$$e_i = \|p_i - q_i\| \tag{5}$$

where $\|\cdot\|$ is the Euclidean norm.

The RMSE ATE is then computed as:

$$RMSE_{ATE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} e_i^2} \tag{6}$$

where N is the number of trajectory points.

III. Integrating Semantics into Visual SLAM

The integration of semantic information into the SLAM pipeline aims to overcome the limitations of purely geometric methods by adding higher-level cues that enable robust perception in dynamic and complex environments. In short, it aims to replace classical geometrical features and informations with

semantiac information which contains the position, orientation, colour, texture, shape, and specific attributes of objects in the environment. For instance, the Kimera library [7] demonstrates a successful fusion of visual-inertial odometry with dense 3D semantic reconstruction.

A. Semantic information extraction

Semantic information is typically extracted using deep learning techniques by following one the three main approaches:

- **Object Detection:** Detectors like YOLO or Faster R-CNN are employed to localize objects in images. These bounding-box detections provide a first-level semantic cue that can be used to replace features or be an addition to it. Moreover, the recent versions of yolo can now detect and identify even partially hidden objects, adding robustness to semantic SLAM. Unfortunately [8] tells us that most simple semantic SLAM algorithms are not robust to dynamic environments. To correct this, an other layer of object classification is added on the more recent algorithms. This additionnal layer aims to identify potentially moving objects such as people, vehicules and animals.
- **Semantic Segmentation:** Semantic segmentation is the cornerstone technology of image understanding, which can give the exact pixels corresponding to each type of object but cannot distinguish different individuals of the same type. It is pivotal in autonomous driving and UAVs. Approaches such as U-Net or PSPNet can deliver pixel-wise labels, enabling a dense classification of scene elements. In Kimera [7], 2D semantic segmentation outputs are combined with depth estimates (from dense stereo) to label reconstructed 3D points for exemple.
- **Instance Segmentation:** This approach is the perfect mixt of the previous two as it allows object detection while also achieving pixel-level object separation. Contrary to the semantic segmentation, this approach can distinguish different individuals of the same type. However, its computing cost is too high to allow for real-time SLAM.

Integrating semantics into a SLAM system is not limited to the extraction of semantic information (via detection, segmentation, or instance segmentation) but also requires the correct association of semantic measurements with persistent 3D landmarks, referred to as semantic landmarks, especially for the localization. This association step, which is critical for robustness in dynamic environments, ensures that observations of the same object across multiple images and time instants are grouped together. Another approach described in [9] aims to replace the semantic landmarks with 3D bounding boxes.

The most important aspect of semantic SLAM however is the mapping part of SLAM. The objective of semantic mapping in SLAM is not only to enable robust localization by building a map that the robot can reuse without reconstructing it at each mission, but also to enrich the geometric map with high-level semantic information. Traditional vSLAM systems generate maps that can be sparse, semi-dense, or dense. While dense maps provide a detailed 3D reconstruction suitable for localization, navigation, and obstacle avoidance, they often lack the semantic labels needed for intelligent human–machine interaction and complex task execution.

Early approaches to semantic mapping often relied on a priori CAD model databases to annotate 3D reconstructions. However, these methods were limited by the predefined set of objects available in the database. Later works integrated dense vSLAM with 2D semantic segmentation labels to build static semantic maps, and some methods even used instance-aware segmentation to differentiate between background, moving, or potentially moving objects. Unfortunately, such approaches sometimes struggle to achieve real-time performance. To address this, several researchers have proposed constructing sparse semantic maps in real time using frameworks like ORB-SLAM2, where semantic objects are directly fused into sparse 3D maps.

An illustrative example of these advancements is **Kimera** ([7]) which can create maps of various difficulties depending on the usage:

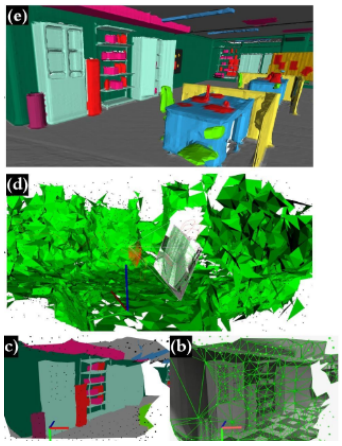


Fig. 5. Illustration semantic mapping using Kimera

IV. Conclusion

The incorporation of semantic information into visual SLAM has been shown to provide significant improvements over traditional, geometry-only methods. Traditional visual SLAM systems rely solely on low-level features such as corners and edges. While effective in static environments, these systems are vulnerable to errors in dynamic or texture-less scenes. In contrast, semantic visual SLAM enhances localization by:

- Providing Contextual Constraints: Semantic labels help distinguish static landmarks from dynamic objects. For example, Kimera uses semantic cues to filter out feature points on moving objects (e.g., vehicles or pedestrians), leading to lower drift and more stable loop closures.
- Improving Data Association: Object-level information facilitates more robust feature matching across frames.

Traditional SLAM generates point clouds or sparse maps that lack high-level information about scene objects. Semantic SLAM, by contrast, produces maps where landmarks are enriched with semantic labels and allows for dense environment reconstruction or simpler semantic mesh reconstructions. These dense semantic 3D maps are a lot more suited for obstacle avoidance and human-robot interaction. Semantic visual SLAM can also be used for visual navigation and allow robot control tu use simple images as position goals [10].

While the integration of semantic information introduces additional processing stages, recent implementations demonstrate that this overhead is mostly manageable. Kimera for exemple runs entirely on a CPU, with its semantic module (Kimera-Semantics) incurring an extra cost that does not prevent real-time operation. The modular architecture of Kimera allows for the system to fall back to a traditional solution if semantic labels are unavailable, offering flexibility based on computational resources. However, the dense environment reconstruction can only be done in post-processing as it cannot be done in real-time.

In summary, while traditional visual SLAM systems are efficient and simpler to deploy, they lack the capacity for rich scene understanding. Semantic visual SLAM, as exemplified by Kimera, achieves:

- Lower localization drift through semantic data association.
- Superior map quality with semantically annotated 3D reconstructions.
- Robust performance in dynamic environments by filtering out non-static features.

These benefits justify the additional computational complexity, particularly in applications requiring high-level environmental understanding and robust operation in real-world, dynamic settings.

REFERENCES

[1] J. M. M. Montiel, Andrew J. Davison, Real-time Monocular SLAM: Why Filter?. IEEE Int. Conf. Robotics and Automation, ICRA 2010.

[2] Raul Mur-Artal, J. M. M. Montiel, and Juan D. Tardos, ORB-SLAM: a Versatile and Accurate Monocular SLAM System. IEEE Int. Conf. Robotics and Automation, 2015

[3] E Rosten, T Drummond , Machine learning for high-speed corner detection, European Conf. on Computer Vision 2006

[4] Rublee, E., Rabaud, V., Konolige, K., and Bradski, G. ORB: an efficient alternative to SIFT or SURF, ICCV 2011

[5] Calonder, M., Lepetit, V., Strecha, C., and Fua, P. (2010). BRIEF: Binary Robust Independent Elementary Features. European Conference on Computer Vision (ECCV 2010), pp. 778–792.

[6] Kaiqi Chen, Jianhua Zhang, Jialing Liu, Qiyi Tong, Ruyu Liu, Shengyong Chen. Semantic Visual Simultaneous Localization and Mapping: A Survey. JOURNAL OF LATEX CLASS FILES, VOL. 14, NO. 8, AUGUST 2021

[7] Antoni Rosinol, Marcus Abate, Yun Chang, Luca Carlone. Kimera: an Open-Source Library for Real-Time Metric-Semantic Localization and Mapping. IEEE Int. Conf. Robot. Autom. (ICRA), 2020.

[8] F. Zhong, S. Wang, Z. Zhang, and Y. Wang, "Detect-slam: Making object detection and slam mutually beneficial," in 2018 IEEE Winter Conference on Applications of Computer Vision (WACV). IEEE, 2018, pp. 1001–1010.

[9] S. Yang and S. Scherer, "Cubeslam: Monocular 3-d object slam," IEEE Transactions on Robotics, vol. 35, no. 4, pp. 925–938, 2019.

[10] Devendra Singh Chaplott, Ruslan Salakhutdinov, Abhinav Gupta1, Saurabh Gupta. Neural Topological SLAM for Visual Navigation. Facebook AI Research, 3 UIUC

Swarm robotics: Analysis of social learning methods

Camilo Ortiz
camilo.ortiz@ensta.fr

March 2025

Abstract

During a mission in an unknown environment, a swarm of robots needs to have a high potential of adaptivity to realize the mission in the most effective way possible. Swarms are well known for their ability to perform complex tasks, despite each individual robot having few control options. Through simulations of swarm learning, we demonstrate that a swarm of robots can, on average, learn and optimize its controller faster than an individual robot learning alone. In this paper, we discuss the forms of swarm learning that exist in the literature and try to present, through a simulation, the different learning rates between those forms of learning.

Contents

1 Introduction : Swarm robotics

The idea of swarm robotics [8] was first inspired by the study of social insects such as ants, bees, wasps, etc. These insects are individually incapable of performing complex tasks, but as a cohesive group, they accomplish remarkable feats that benefit the entire swarm. Ants build bridges and collect food, wasps build nests with complex geometrical properties. What's interesting about this is how complex behaviors can emerge from a group of animals that individually have limited capabilities. Swarm robotics, as described by Beni [1], is the study of how to coordinate large groups of relatively simple robots through the use of local rules. In [11], Sahin outlines the property of the emergence of complex synchronization as a key point to swarm robotics:

> Swarm robotics is the study of how large number of relatively simple physically embodied agents can be designed such that a desired collective behavior emerges from the local interactions among agents and between the agents and the environment.

To become a coherent swarm of robots, automatically synchronized and working towards a same goal, the robots must individually follow some general rules:

1. The robots of the swarm must be autonomous robots, able to sense and actuate in a real environment.
2. The swarm must consist of a large number of robots, or at least be governed by control rules that allow scalability.
3. Robots should be homogeneous. While different types of robots may exist within the swarm, their diversity should be limited.
4. The robots must collaborate to ensure success and enhance the overall performance of the swarm.
5. Robots have only local communication and sensing capabilities. It ensures the coordination is distributed, so scalability becomes one of the properties of the system.

1.1 Swarm systems' properties

In [7], Isaeva demonstrated that social insects act in a decentralized way, which means that they have no leader or human operator sending information orders. All the robots are hierarchically equal, yet their system and task repartition is robust, flexible and scalable. Those properties are the reason why swarms of robots can be efficient for specific tasks [11].

1. Robustness measures the adaptability of the system in case of failures of one or more robots, or disturbances in the environment. Swarms' properties make the systems robust by essence. First, the redundancies in the system. The swarm possesses more robots than needed, thanks to the low fabrication cost of 'simple' robots. This makes the robots dispensable. In addition, the decentralized property makes the swarm robust to the loss of any robot, whereas losing a leader in a centralized system would be bothersome. Secondly, the multiplicity of sensing, in a system in which robots communicate their vision of the environment, can help increase the total signal-to-noise ratio of the system.
2. Flexibility requires the swarm robotic system to have the ability to generate proficient solutions to different tasks, provided they have the time to learn how as a group.
3. Scalability requires that a swarm robotic system should be able to operate under a wide range of group sizes. That is, the coordination mechanisms that ensure the operation of the swarm should be relatively undisturbed by changes in the group sizes.

Swarm robotics is considered a promising approach for a wide range of applications. The list being inexhaustible, we can mention agriculture, rescue mission, foraging, exploration and environmental monitoring. We can describe in the grand lines the characteristics of the tasks in which swarms can be efficient.

1.2 Types of tasks in which swarm robotics can bring value

1. In any task that require a repartition over a field, We can simply deploy a swarm of robots with a homogeneous repartition, in such a way that each robot can communicate with the robots nearby. This allows for an extremely fast transfer of information across a large field. In case of a rescue mission for instance, once one robot finds the target, information can spread and all the robots will converge to the location and help in the rescue. Such strategy is efficient thanks to the creation of a network of sensors.
2. For high-risk tasks, the expendability of individual robots makes swarm robotics a suitable solution. For instance, clearing a path through a mined field can be accomplished quite efficiently by a swarm of "suicidal" robots. This is possible because the swarm can be deployed with more robots than strictly necessary.
3. Tasks that scale-up or scale-down in time can be handled efficiently with a swarm. For instance, an oil leakage from a sinking ship can increase due to the tanks slowly breaking down. When the problem's urgency is low, only a few robots are enough to handle the task. And with time, as the leakage becomes more important, we can add robots little by little to help fixing the issue.

4. Finally, tasks that require redundancy are handled thanks to the redundancy property of the swarm. This redundancy allows the swarm robotic system to degrade in a non blocking way, making the system less prone to catastrophic failures. For instance, swarm robotic systems can create dynamic communication networks in the battlefield. Such networks can enjoy the robustness achieved through the reconfiguration of the communication nodes when some of the nodes are hit by enemy fire.

Now that we have discussed the use cases of swarm robotics, we have to think about how to recreate the insect-like behaviors that we observe in nature. In this paper we will talk about social learning, i.e. how robots can evolve while accomplishing a mission, and converge together towards a positive solution. Section 2 will explain the concepts of social learning, section 3 will display a simulation of those concepts in foraging tasks, section 4 will be an analysis of diverse techniques of swarm learning found in the literature, and finally section 5 will talk about the future of social learning in swarm robotics.

2 Social learning in swarm robotics

Social learning as presented in this article is a sub category of machine learning, that uses gradient free methods [9]. This means that the robots will never use the derivatives of the fitness function to improve their performances. Instead, they will communicate and exchange their own experiments, their local information of the environment, they will transfer information that spreads among the whole swarm, or more simply, they will share their own control functions. As explained in [2], social learning in swarm robotics is framed by constraints of locality.

1. Local interaction: All the robots can see the world only through their own sensors, thus experiencing their own local version of what is surrounding them. This can lead to detrimental behaviors, such as robots hindering or degrading each other. But it can also lead to unexpected collective behaviors, such as robots aligning with one another, unifying their motion. Whatever behavior may emerge from the learning process, the robots only perceive what's near them. They don't possess wide knowledge of the environment.

2. Local communication: Communication between robots is limited in radius (either by choice, or because of technical limitations, which we'll talk about in section 3). Besides, the information transferred can be limited in size and in speed. This implies that diffusion over the swarm can take time even if all the robots are packed together. But as mentioned previously, information transmission would still remain faster than having one robot physically move to be in the communication range of another robot.

3. Local performance self-assessment: The swarm is autonomous, both as a group and individually. This means that there is no operator computing the contributions of each robot. The robots of the swarm need to have its own self-assessment method. The job of the human operator, before a mission, would be to provide this self-assessment function. This is a central nerve in the learning process of the swarm, because this function needs to compute the benefits brought by one robot to the whole swarm. For instance, in a foraging mission, gathering objects is obviously a contribution to the swarm, but operating too close to other robots will reduce efficiency due to an overcrowding effect.

The two first rules allow for different methods of social learning : imitation through observation, or through communication of the control function that maps the sensors to the actuators of the robot. The third aspect of locality is where the artificial diverges from the natural. While in nature the performance of an individual is evaluated in terms of its ability to survive and reproduce, this is not the case here. In swarm robotics, performance assessment is explicitly calculated by a fitness function designed by a human supervisor. Hence, a robot may be measured as efficient with respect to its participation to the accomplishment of the task, without its own integrity being taken into account. This has a major implication for the dynamics of social learning, as the very definition of successful behavioral strategies now depends both on their ability to diffuse over the population, and their ability to perform well on a user-defined task.

3 Simulating foraging tasks

In this section, we will develop a simulation for a practical case of social learning. We will implement robots with the task of foraging.

3.1 The environment

The robots resemble Dubins' cars. They have two actuators, left wheel and right wheel. Nine distance sensors are attached to the robot, uniformly distributed around the z-axis. They are limited in range, which makes the robots only aware of what's in a circle of radius max_range. The function that maps the sensors output to the actuators (the motors of each wheel) is a neural network called a perceptron. In our case, the perceptron has no hidden layer, it maps 9 sensors into 2 outputs, hence the number of parameters is 18. The robots' objective is to find objects. Those objects are represented as green disks. In order to validate the task, the robot must be at a certain distance from the object. There are 150 objects in the world, distributed randomly, and in order to keep the number of objects

constant, an object will re-spawn immediately in the world, at a new random location.

3.2 Sensors simulation

The sensors placed around the robot allow it to detect the distance between the sensor and the object. In order to manage this in a computational point of view, we need to solve a geometrical intersection problem between a ray and a circle. Our goal is to determine the intersection points between the ray and the circle. A ray can be expressed parametrically as:

$$x = x_0 + t \cdot dx$$

$$y = y_0 + t \cdot dy$$

A circle is defined by:

$$(x - x_c)^2 + (y - y_c)^2 = r^2$$

This means all points (x, y) on the circle satisfy this equation. Replacing x and y in the circle equation using the ray equations gives:

$$(x_0 + t \cdot dx - x_c)^2 + (y_0 + t \cdot dy - y_c)^2 = r^2$$

Developing this equation gives a polynomial equation for t:

$$At^2 + Bt + C = 0$$

where:

$$A = dx^2 + dy^2$$

$$B = 2(dx \cdot (x_0 - x_c) + dy \cdot (y_0 - y_c))$$

$$C = (x_0 - x_c)^2 + (y_0 - y_c)^2 - r^2$$

The quadratic formula solves for t:

$$t = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}$$

If the solutions exist, it means the ray crosses the circle at least once. Among the two solutions, we only keep the positives ones, as we are looking for a distance. And if both are positive, we keep the smaller one. We repeat this operation for all 9 sensors.

3.3 Information transfer

In this simulation, robots learn from each other by transferring their own perceptron to other robots they meet along the mission. When two robots are within communication range, a Horizontal Information Transfer algorithm is triggered [5]. First, the two robots compare their self-assessed score. The robot with the highest score will transfer its perceptron. In the literature, a common procedure is to only transfer a percentage α of the perceptron, in which case the transferred parameters are chosen randomly. This is due to a simulation of any communication failure that might happen. In real life applications, the distance between two robots, the harsh environment, the cheap computers embedded in each robot, are all factors that might create failures in the information transfer. But in this simulation, we will purposely use this parameter to create new genotype combinations, hopefully resulting in new positive behaviors.

3.4 Fitness function

To assess the performances of the robots, we will use a simple function that computes the number of objects found within the last N steps of the simulation.

$$f = \sum_{n=CurrentStep-N}^{CurrentStep} ObjectsFound \qquad (1)$$

By computing the number of objects found within a precise amount of time, we can have a rough idea of how efficient the robot is compared to its peers. At the beginning of the simulation, we let some time before beginning the mating processes, so the robots can have a good, full evaluation of their perceptron. Once the a robot self-assessed its performance, it becomes open to meeting other robots. We then use the HIT algorithm as explained in 2, the robot with the least good performance copies a fraction of the other's perceptron. Then his score is resetted to 0, and it returns to an evaluation phase. Once a robot has been self-evaluating during enough time and becomes open to communication, its fitness function keeps running, and we only keep the objects found during the last N steps. This allows to reduce the score of a robot that was lucky at the beginning of its assessment but that ended up stuck in a corner of the field for instance.

3.5 Results

The simulation was tested with various parameters as previously explained (see 1).

My simulation didn't work as expected, it lacks some adjustments for the social learning to be effective. One way to improve it would be to introduce to swarm some agents with a perceptron of our choice, one that would be effective. Doing so wouldn't prove the creation of new behavior but the perceptron introduced would normally be able to spread to the whole swarm.

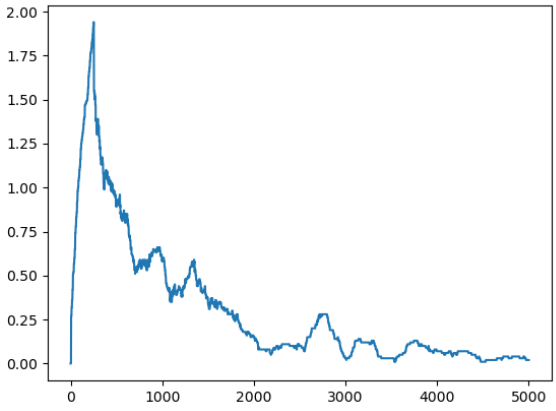


Figure 2: The average score of the swarm of robots with respect to the step number, with $\alpha = 0.9$

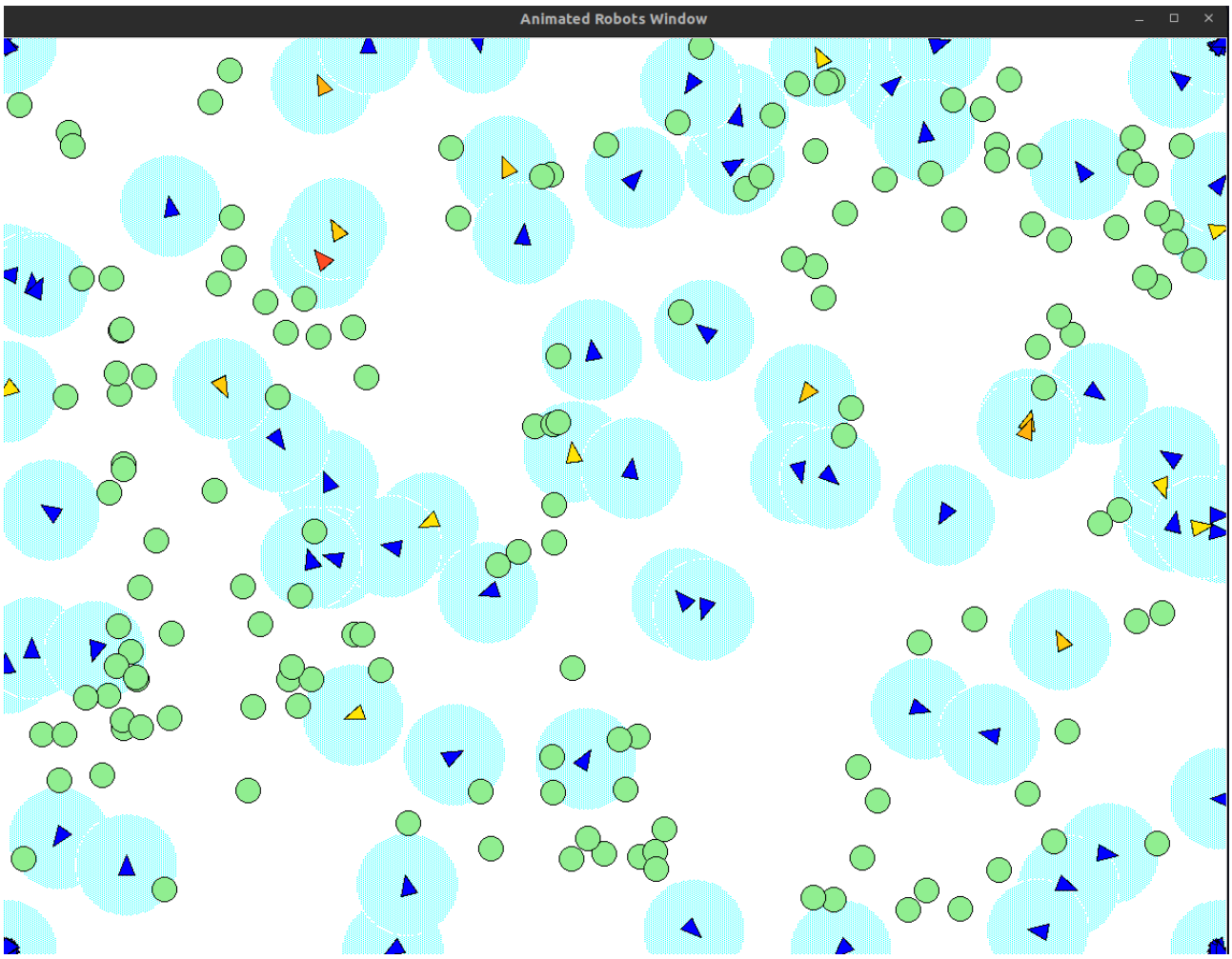


Figure 1: An image of the simulation. Green dots represent the objects to forage. The triangles are the robots, at the center of a blue circle representing their circle of detection. Blue robots have found zero objects. A gradient of color from yellow to red display the number of objects found by each robot

?? Shows that the simulation starts with a peak of performance, which corresponds to the randomness of the system's initialization. Then quickly after, the robots end up stuck in the corners of the map and the performance decreases rapidly.

In a simulation similar to the one implemented, Bredeche [2] proved that it is possible for the robot's perceptrons to converge towards one genotype that renders good performance for the whole swarm. The fact that one genotype manages to emerge from the initialization batch proves that its performance is consistent and does not depend on luck or circumstance. In most cases, its performance will surpass that of the others.

One advantage of gradient-free machine learning methods is their ease of implementation and low computational requirements. It makes the methods suitable for real-time learning in simulations and even possible to run on interpreted languages such as Python. However, if we add more than 100 robots, the simulation will slow down considerably. The first improvement would be to implement the same code in C++, or to use simulators already implemented in the literature, such as e-puck [4].

Another improvement, to help converge towards an effective controller, would be to combine the concepts of social learning with other gradient-free methods of machine learning. We can mention genetic algorithms, or imitation through observation of the other robots, self learning through experimentation. In the next section, we will analyze how to implement these methods and how beneficial they can be in solving the problem of creating a collective intelligence of the swarm.

4 Comparison between swarm adaptation methods

In this section we present two other approaches to evolving robot controllers in a swarm during a mission (we talk about on-line learning). While social learning has proven to be a robust solution, exploring alternative approaches in the literature is worthwhile. The first method is an on-board learning one. [3] aims to evolve robot controllers on-line using only the robots' computational resources. The key focus is on optimizing obstacle avoidance and movement efficiency through phenotype evolution. The core machine learning approach used in this study is a (1+1)-

```
for evaluation ← 0 to N do
    if random() ≤ P_reevaluate then
        Recover(Champion)
        Fitness_Champion ← RunAndEvaluate(Champion)
    end
    else
        Challenger ← Champion + 𝒩(0, σ) Recover(Challenger)
        Fitness_Challenger ← RunAndEvaluate(Challenger)
        if Fitness_Challenger ≥ Fitness_Champion then
            Champion ← Challenger
            Fitness_Champion ← Fitness_Challenger
            σ ← σ_min
        end
        else
            σ ← σ × 2
        end
    end
end
```

Algorithm 1: (1+1)-ONLINE Evolutionary Algorithm

ONLINE Evolutionary Algorithm. Like social learning, this method involves the learning of a perceptron.

This algorithm is mutation-based and noise-smoothing. The evolutionary strategy relies on Gaussian mutation by a small step σ. When a mutated controller outperforms its predecessor, the mutation rate is reduced, and it increases if no improvement occurs. The algorithm includes a reevaluation step, which allows to reduce the noise of "lucky" perceptrons. Due to noisy fitness evaluation (changing environmental conditions), successful controllers are retested at random intervals to ensure robustness. A fitness function is used to evaluate the performance of the perceptrons. In this experience, the function was chosen so that straight movements with minimal collisions are appreciated.

The study demonstrated that on-line evolution is feasible, but with certain concessions. Evolution gradually improved the performance of the controller, with robots having increasingly optimized obstacle avoidance behaviors. But sometimes performance would regress because the controllers got stuck in local optima and adaptation was slow because the system relied solely on mutation without an additional learning mechanism. One of the biggest values of the algorithm is the reevaluation that helps mitigate overfitting. In this way, we ensure that selected controllers remain effective over multiple tests. This method proves to be generally feasible. The (1+1)-Online approach favors stability but is limited in its elasticity. When confronted with a whole new environment, robots have difficulties adapting.

—

In [6], a new approach is considered: the idea of cross-implementing various gradient-free machine learning methods. This allows to speed up the optimization of the perceptrons. In this article, the "Three-fold adaptivity" is a combination of three concepts. Genetic algorithm, Individual learning (like the previous study) and social learning (like the previous simulation). As we already explained the concepts of social and individual learning, we will now focus on Genetic algorithm [10]. This method requires that a ranking of the more efficient individuals (always according to their fitness function). The first batch of robots (the most efficient) remain the same, and the second batch will undergo a genetic transformation based on the barycenter of two genotypes chosen among the first batch. It is also possible, as a variant method, to add some randomness with a Gaussian mutation during the transformation process. These algorithms are renowned for their fast convergence, making the learning method very suitable for online learning. As expected, the robots' adaptation in Heinerman's study was significantly faster compared to the example in [3] due to the combination of evolution, individual learning, and social learning. Social learning had the greatest impact in homogeneous robot groups (groups of robots that had the same sensory layout), while its effect was reduced in heterogeneous groups. As a plus, evolution favored more efficient designs, reducing computational complexity by limiting the number of active sensors. Finally the study made clear that a hybrid approach combining genetic evolution with individual and social learning is more effective than relying on mutation alone.

5 Conclusion

Obtaining complex behaviors from a swarm of robots on the fly is not easy but not impossible thanks to gradient-free methods of machine learning. Every parameter of the robot, ranging from its perceptron to its configuration (i.e. which sensors is the robot allowed to use) can evolve, be learned/taught and transferred. Three-fold adaptivity may be a subject of further study to implement in more complex tasks. Whether it is foraging or obstacle avoidance, those tasks do not require

much synchronization from the swarm. In future research it may be interesting to focus on swarm tasks that require the robots to work together (moving a heavy object, having a synchronous locomotion...)

References

[1] Gerardo Beni. From swarm intelligence to swarm robotics. In *International Workshop on Swarm Robotics*, pages 1–9. Springer, 2004.

[2] Nicolas Bredeche and Nicolas Fontbonne. Social learning in swarm robotics. *Philosophical Transactions of the Royal Society B*, 377(1843):20200309, 2022.

[3] Nicolas Bredeche, Evert Haasdijk, and Agoston E Eiben. On-line, on-board evolution of robot controllers. In *Artifical Evolution: 9th International Conference, Evolution Artificielle, EA, 2009, Strasbourg, France, October 26-28, 2009. Revised Selected Papers 9*, pages 110–121. Springer, 2010.

[4] Christopher M Cianci, Xavier Raemy, Jim Pugh, and Alcherio Martinoli. Communication in a swarm of miniature robots: The e-puck as an educational tool for swarm robotics. In *Swarm Robotics: Second International Workshop, SAB 2006, Rome, Italy, September 30-October 1, 2006, Revised Selected Papers 2*, pages 103–115. Springer, 2007.

[5] Nicolas Fontbonne, Olivier Dauchot, and Nicolas Bredeche. Distributed on-line learning in swarm robotics with limited communication bandwidth. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, 2020.

[6] Jacqueline Heinerman, Dexter Drupsteen, and Agoston Endre Eiben. Three-fold adaptivity in groups of robots: the effect of social learning. In *Proceedings of the 2015 annual conference on genetic and evolutionary computation*, pages 177–183, 2015.

[7] Valeria Isaeva. Self-organization in biological systems. *Izvestiia Akademii nauk. Seriia biologicheskaia / Rossiĭskaia akademiia nauk*, 39:144–53, 04 2012.

[8] Iñaki Navarro and Fernando Matía. An introduction to swarm robotics. *International Scholarly Research Notices*, 2013(1):608164, 2013.

[9] John C. Raisbeck. On the term “gradient-free” in machine learning, 2024. Consulté le 3 mars 2025.

[10] Nesma M Rezk, Yousra Alkabani, Hassan Bedor, and Sherif Hammad. A distributed genetic algorithm for swarm robots obstacle avoidance. In *2014 9th International Conference on Computer Engineering & Systems (ICCES)*, pages 170–174. IEEE, 2014.

[11] Erol Şahin. Swarm robotics: From sources of inspiration to domains of application. In *International workshop on swarm robotics*, pages 10–20. Springer, 2004.

A Lightweight Approach to Efficient Multimodal 2D Navigation and Mapping: Unified Laser-Scans as an Alternative to 3D Methods

Ocean Noel[1,*], Rafael Cisneros-Limón[1], Kenji Kaneko[1] and Fumio Kanehiro[1]

***Abstract*— In this paper, we propose a novel approach for efficient 2D navigation using a multimodal sensor fusion technique. Our method focuses on merging data from multiple sensors, such as LiDARs, cameras, and ultrasonic sensors, into a unified Laser-Scan, which serves as a foundation for faster and more lightweight navigation. By fusing sensor data at the Laser-Scan level, our approach enables the use of basic 2D Simultaneous Localization And Mapping (SLAM) algorithms for mapping tasks, or any others Laser-Scan based features, while still benefiting from the rich information provided by multimodal 3D inputs. This results in a more computationally efficient solution compared to traditional 3D methods that rely on depth points or full multimodal SLAM systems. Our experimental results demonstrate that the proposed approach achieves comparable accuracy in mapping and localization while significantly reducing computational complexity and processing time. This research offers a promising alternative for real-time 2D navigation in resource-constrained autonomous systems, such as drones or any small unmanned vehicles.**

***Index Terms*— Sensor fusion, Multimodal sensors, SLAM, Navigation, 2D mapping, Real-time processing.**

I. Introduction

Autonomous navigation in complex and dynamic environments is a critical challenge for various applications, including robotics, self-driving vehicles, and drones. Accurate mapping and efficient localization are essential components of any navigation system, enabling autonomous agents to understand their surroundings and make informed decisions. For addressing these challenges, Simultaneous Localization and Mapping (SLAM) has emerged as a popular solution, particularly with the advent of 3D SLAM techniques that leverage depth information from sensors like 3D LiDARs, stereo cameras, and RGB-D cameras [1]. However, 3D SLAM methods as well as most current navigation features considering 3D obstacles, often require substantial computational resources [2], which can be a limiting factor for resource-constrained autonomous systems. Moreover, processing data from multiple sensors in real-time can be challenging, leading to delays and reduced performance [2]. To address these issues, we propose a lightweight alternative to 3D methods such as those using depth points. Throughout this paper, the term 'Laser-Scan' will refer to the LaserScan message in the Robotic Operating System (ROS) [3][4], a standard message type used to publish 2D LiDAR data. This format typically stores data in a simple list containing the ranges (distances) of detected obstacles or INFINITY value if no obstacle is detected. The position of a range in this list allows to know the direction of the obstacle, based on the Field of View (FoV) of the sensor specified in the Header of the LaserScan message. The LaserScan message is widely used in robotics applications for tasks such as mapping, localization, and obstacle detection, due to its ability to provide a detailed representation of the environment surrounding the robot [5]. Our method focuses on fusing sensors data at the Laser-Scan level, creating a unified Laser-Scan that serves as a foundation for faster and more lightweight 2D navigation.

When navigating in a 2D space, it is still important to consider 3D obstacles, as their shape might not be detected by classical 2D LiDARs and the robot might collide with them. This is particularly true for objects with a shape that changes with the height (i.e a chair or a person), since 2D LIDAR only detects a single layer at a time, whereas a camera can yield 3D information. This allows the system to consider shapes and obstacles outside the 2D LiDAR's plane, addressing the problem that the robot may be taller than the clearance under certain objects. Therefore, we aim to integrate these 3D data into our 2D mapping and navigation system.

Several methods are possible for this integration. We can use usual 3D mapping/navigation algorithms like `RTAB-Map` [6] or `ORB-SLAM3` [7] to represent the environment and project the map [8][2]. However, this can add unnecessary computing time, as Visual SLAM has higher computational requirements than LiDAR SLAM [2]. Also considering 2D navigation, using 3D LiDAR based algorithms like `SC-LeGO-LOAM` [9] may also lead to unnecessary 3D data computing and storage. Other solutions have already been developed for 2D navigation considering the 3D environment [10][11]. Compared to those solutions, that will be detailed in Section II, we propose an easy-to-plug modular package implemented in ROS2 Humble [12][13]. It takes an unrestricted amount of inputs formatted as Laser-Scans, such that any robot can easily plug and play this package into their current mapping, or navigation, system using several different sensors. For example, the result of `SC-LeGO-LOAM` [9] can easily be adapted and used as inputs to our package for lighter high-level computations. Also, the Laser-Scan called "virtual 2D scan" generated from Wulf's system [10][11] can be used with our package and fuse with "other sensor data.

Laser-Scans are the lightest 360-degree representation of the current closest environment around the robot. By converting everything into Laser-Scan prior to any mapping or navigation algorithm, we can avoid the need of heavy 3D approaches. Also, many Laser-Scan-based algorithms already

[1] CNRS-AIST JRL (Joint Robotics Laboratory), IRL, Tsukuba, Japan.
* Corresponding author E-mail: ocean.noel.jp@gmail.com

exist in the literature for mapping [14][15][16][17][18], or for navigation purposes [19][20][21][22][23][24][25]. Thus, having a rich Laser-Scan message containing 3D-sourced data can leverage all the existing features using Laser-Scans and lighten the entire navigation stack.

The remainder of this paper is organized as follows:

- Section II provides an overview of the related work.
- Section III details the problem statement; that is, the current limitations.
- Section IV proposes a methodology for fusing sensor data at the Laser-Scan level.
- Section V presents and discusses the experimental results and evaluations.
- Section VI concludes the paper and outlines future work.

II. Related work

A. Sensor Fusion

Sensor fusion is a crucial technique in robotics that combines data from multiple sensors to enhance feedback accuracy and reliability. By integrating complementary information from diverse sensors, such as LiDARs (Light Detection and Ranging), cameras, and IMUs (Inertial Measurement Unit), sensor fusion mitigates individual sensor limitations and uncertainties. This process often employs probabilistic robotics principles, using Bayesian inference and statistical methods to estimate the system's state. Fusing data reduces noise, handles sensor failures gracefully, and improves confidence in the estimated state, leading to more reliable performance in complex environments. Traditional methods like Kalman filters, particle filters, and Bayesian networks have been extensively used to integrate data from various sensors. For example, Thrun et al. [26] provide a comprehensive overview of probabilistic methods for sensor fusion in robotics.

Recent advancements in deep learning have introduced neural network-based approaches for sensor fusion, offering improved performance in complex environments. For instance, Chen et al. [27] demonstrate the use of Convolutional Neural Networks (CNNs) for fusing LiDAR and camera data to improve object detection and localization. Similarly, Wang et al. [28] employ deep learning to fuse data from multiple sensors for enhanced navigation and obstacle avoidance.

B. 2D Navigation Using 3D Sensors Data

Multimodal mapping involves creating maps using data from various sensors, leveraging each sensor's strengths to build a comprehensive and accurate representation of the environment. A LiDAR provides distance measurements with centimeter-level accuracy, operates effectively in low-light conditions, and offers a wide FoV, making it ideal for precise obstacle detection and mapping. Cameras offer rich visual information, including color and texture, crucial for identifying landmarks and understanding the environment's context. However, cameras may struggle in low-light conditions and can be affected by environmental factors such as fog or rain. Combining these sensors enhances the robot's ability to navigate accurately and safely in diverse conditions. Techniques such as probabilistic mapping, occupancy grids, and feature-based mapping are commonly employed to integrate multimodal data. The work by Wulf et al. [10] presents a method for integrating 3D data into 2D navigation by projecting 3D point clouds onto a 2D plane, creating a colored occupancy grid map. While this approach allows the robot to consider 3D obstacles while navigating in a 2D space, it relies solely on a 3D LiDAR and does not directly allow merging data from different sensors.

Similarly, the ROS2 plugin `Navigation2` allows the generation of 2D costmaps from 3D depth points and Laser-Scan inputs [19]. These costmaps are then used to perform path planning and navigation. However, this method does not efficiently consider useful-only data (basically the inner part of obstacles can be ignored), making it computationally costly when several points are considered.

III. Problem Statement

While current 3D multimodal mapping and navigation systems offer high accuracy and robustness, they often require substantial computational resources and can be challenging to implement in real-time on resource-constrained platforms. The processing of high-dimensional data from multiple sensors can lead to delays and reduced performance, making these methods less suitable for applications where computational efficiency is critical.

Moreover, most existing optimized methods focus on a single type of sensor. For example, LiDAR-based SLAM methods, such as `LOAM` [29], are highly effective for 3D mapping but do not efficiently integrate data from other sensors like cameras. Similarly, visual SLAM methods, such as `ORB-SLAM` [30], focus on using monocular camera images. This specialization limits the flexibility and adaptability of these systems in diverse and dynamic environments and on various robot's systems.

Currently, there is a lack of ROS packages that allow for an accurate and efficient fusion of data from multiple sensor types using a standard shared representation. Existing solutions often require custom integration and optimization, which can be time-consuming and complex. For instance, the work by Cadena et al. [1] highlights the challenges in integrating different sensor modalities and the need for more robust and generalized fusion techniques.

In summary, the primary challenges in the current state of the art for 2D Navigation include: 1. High computational requirements for processing high-dimensional data from multiple sensors. 2. Lack of efficient and accurate methods for fusing data from different sensor types. 3. The need for a standardized and modular approach to sensor fusion that can be easily integrated into existing robotic systems.

IV. Methodology

In this section, we present the methodology for converting and merging sensor data into a unified Laser-Scan format, which is essential for efficient and accurate navigation and mapping. We have divided this section into subsections to

provide a comprehensive overview of our approach. The first subsection, "All Sensors to Laser-Scan," details the process of converting data from various sensors, such as depth cameras, into a common Laser-Scan format. This focuses on the specific method for transforming a point cloud of depth points to Laser-Scan, addressing the challenges posed by moving cameras in the robot's frame. The subsequent subsection, "Laser-Scan Merger," focuses on the merging of Laser-Scan data from multiple sensors. This involves transforming the data to a common reference frame, synchronizing the data using odometry interpolation, and performing fusion to create a unified representation of the environment. Each subsection provides detailed steps and considerations to ensure the accuracy and reliability of the merged data. Our method is ready-to-use and available on the public GitHub repository `multi-laserscan-toolbox-ros2` [12].

A. All Sensors to Laser-Scan

Our approach involves converting data from various sensors into a unified Laser-Scan format. For instance, depth information from cameras can be transformed into a Laser-Scan-like representation. This conversion simplifies the data processing pipeline and reduces the computational load.

1) Laser-Scan format: The desired Laser-Scan is represented in polar coordinates, where each point is defined by an angle a and a range r. Mathematically, a Laser-Scan can be described as a set of ordered pairs (a_i,r_i), where a_i represents the angle of the $i-th$ point relative to a reference direction, and r_i represents the distance from the sensor to the detected obstacle at that angle. The angle a typically ranges from 0 to 2π radians (or 0 to 360°) for a full 360° scan. This representation is particularly useful for navigation and mapping tasks, as it provides a straightforward way to represent the environment in terms of distances to obstacles in various directions.

2) Depth Points to Laser-Scan: The output of depth cameras is usually a point cloud, and we need to convert it to a Laser-Scan. The point cloud is represented by a list of points, that we call depth points all along this paper, defined by their position x, y, and z in the camera's frame. The safest way of conversion for navigation is to keep the closest points to the robot from the point cloud projected on the floor plan, and retain this data in the Laser-Scan.

Some algorithms are available to perform such a conversion [31][32]. However, to our best knowledge, none of these algorithms can consider a moving camera in the robot's frame when filtering the depth points. For example, our robot, described in the following Section V, is equipped with two cameras, each with a 52° vertical FoV. These cameras are mounted on servomotors, allowing for tilt movement, which enables them to sweep up and down to gather more information. This dynamic capability should not negatively impact the navigation quality. However, existing algorithms impose constraints to the camera specified in the initial or fixed configuration of the camera. A major limitation of this approach is that it prevents the specification of desired obstacle filtering constraints in the world frame.

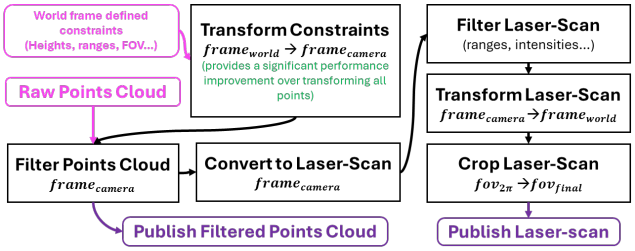


Fig. 1: Point Cloud to Laser-Scan algorithm's Logic.

For instance, consider a scenario where our robot needs to navigate through a doorway with a high overhead clearance. If the camera is tilted downwards to scan the floor, the algorithms may interpret the floor as an obstacle due to the fixed reference configuration. Similarly, if the camera is tilted upwards to scan the ceiling, high objects like door frames might be detected as obstacles, even though the robot could easily pass beneath them. To address this, we developed our own point cloud to Laser-Scan converter that takes into consideration relative motion. Our algorithm allows specifying constraints in the world frame, and using the camera-robot transformation, it automatically extracts the desired filtered point cloud. This allows to consistently filter the wanted depths points even when the camera is tilted up or down. Additionally, our package includes the capability to detect holes and treat them as obstacles. A concise overview of the algorithmic strategy employed by this package is presented in Fig. 1. Although further details cannot be provided in this paper, the package is freely available in our repository [33].

B. Laser-Scan Merger

Once our sensor's data is lightened by being converted into Laser-Scans, they can be used as inputs to our Laser-Scan Merger. An unrestricted number of inputs can be set to the Laser-Scan merger, and each input can be configured individually by specifying the desired FoV, ranges limits (minimum or maximum distances to consider for obstacles), and more. The result is a Laser-Scan containing the synchronized merged information from each sensor. This output can also be further transformed to have a specific FoV or specific ranges. The following subsections will detail the steps involved in this process, including the transformation to a virtual common standard 360° Laser-Scan, the synchronization using odometry interpolation and the global fusion of the data.

1) Transformation to virtual common standard 360° Laser-Scan: Before merging Laser-Scans from multiple sensors, our algorithm standardizes their formats by aligning the resolution (number of points), the FoV, and the reference frame to a common format. Specifically, the common format includes the final desired resolution specified by the user for the output Laser-Scan, a FoV of 360° to encompass data from all possible sensors around the robot, and the output frame specified by the user. The steps involved in this process are illustrated in Fig. 2 and will be detailed in this subsection.

The first step allows to take into consideration Laser-Scans with different resolutions. Indeed, for example, in our case, camera data contains many points, so we decided to convert it into a Laser-Scan with 1440 points, whereas our two LiDARs only have 360 points each. A simple embedded remapping program allows converting a low-resolution Laser-Scan to a higher one and vice-versa. This remapping program also converts the Laser-Scan to a 360° format by adding INFINITY values for out of FoV values, and filter the values according to the constraints fixed by the user (Maximum range, minimum range, specific FoV). The implementation logic for those steps is illustrated in Fig. 2 with, respectively, the names "Resolution conversion", "FoV conversion" and "User preferences Filtering". For the resolution conversion, Eq. (1) is used to determine the new position i_{new} of a polar point in the final Laser-Scan point list of resolution n_{new}, based on its position in the initial Laser-Scan point list i_{init} with n_{init} points.

$$i_{\text{new}} = \frac{n_{\text{new}}}{n_{\text{init}}} \cdot i_{\text{init}} \tag{1}$$

In the case of low-resolution data extended to a higher one, the gaps between the known data are filled with INFINITE range values. These INFINITE values are replaced during the global fusion if another sensor has more data here due to a better resolution or a different origin.

Secondly, merging Laser-Scans from several sensors placed at different origins on the robot requires knowing their relative positions. The algorithm extracts those relative positions from the Laser-Scans directly, as they contain the frame name in which the data are expressed, and transform all the 360° formatted Laser-Scans into virtual common origin Laser-Scans. The transformation involves aligning the data from each sensor to a common reference frame, ensuring that the Laser-Scans are spatially consistent. This means that the positions of obstacles detected by different sensors are correctly mapped relative to each other and to the robot's position, creating a coherent and consistent representation of the surroundings. This step is crucial for accurate fusion of the sensor data. We first transform each Laser-Scan's points from polar coordinates to Cartesian coordinates ((x,y) in 2D). For that, using the definition detailed in Subsection IV.A.1 above, we use Eq. 2.

$$x = r \cdot \cos(a) \qquad y = r \cdot \sin(a) \tag{2}$$

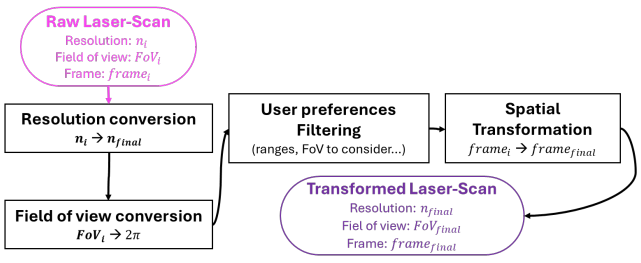


Fig. 2: Algorithm's logic that converts any Laser-Scan to the wanted standard format.

Then, using homogeneous coordinates [34], we transform each Cartesian point to the wanted common virtual frame using the relation Eq. 3, with:

- $A_{|1}$: The homogeneous coordinates of the point A in reference frame R_1.
- $A_{|2}$: The homogeneous coordinates of the point A in reference frame R_2.
- T_{2-1}: The linear homogeneous transformation matrix from R_2 to R_1.
- R_{2-1}: The angular homogeneous transformation matrix from R_2 to R_1.

$$A_{|2} = (T_{2-1}R_{2-1})A_{|1} \tag{3}$$

And then, we transform each Cartesian point in R_2 back to polar coordinates, this time with respect to the new frame, using the relations 4.

$$r^2 = x^2 + y^2 \qquad \tan(\alpha) = \frac{y}{x} \tag{4}$$

This step corresponds to the block named "Spatial Transformation" in Fig. 2.

Finally, the overall algorithm to have compatible Laser-Scans matching the user preferences is as shown in Fig. 2.

2) Laser-Scan Synchronization Using Odometry Interpolation: Format compatibility problems have been solved as described in the previous subsection. However, each sensor might publish its data asynchronously with different rates and with some delay. So the data are not time-compatible and cannot yet be merged as they are. To perform an accurate and consistent fusion, we need to synchronize all the data on a given timestamp. This is a well-known problem when dealing with distributed sensors [35].

In order to deal with that, our algorithm is able to apply a transformation on the received Laser-Scans. If the Laser-Scan message is late, it will apply the correct transformation to it so that even if the data is old, we can extrapolate it to the current timestamp. For that, the algorithm uses the robot's odometry to estimate the motion between the data's timestamp and the current timestamp. However, using the odometry data as a Laser-Scan synchronizer can be problematic. Indeed, as it is also published asynchronously, we might not have the robot's state at the exact timestamp of the sensors' data. Thus, our algorithm contains an embedded odometry interpolator to estimate the robot's state at the time of the sensor data, knowing the state before and after. This interpolator uses the algorithm Slerp [36] for quaternion interpolation for orientation and linear interpolation for position. Let's say we have two odometries $Odom_1$ and $Odom_2$, we define x_i, y_i, q_i, t_i, respectively, the position on the x-axis, the position on the y-axis, the Quaternion and the timestamp of $Odom_i$, which represents the robot's state at time t_i. If the package receives a Laser-Scan at a timestamp t_a with $t_1 < t_a < t_2$, our algorithm will estimate the robot's

state (position, orientation) at t_a with Eq. (5):

$$
\begin{aligned}
u &= \frac{t_a - t_1}{t_2 - t_1} \\
x_a &= x_1(1-u) + x_2 u \\
y_a &= y_1(1-u) + y_2 u \\
q_a &= \text{Slerp}(q_1, q_2, u)
\end{aligned}
\tag{5}
$$

The last available odometry will be used if the wanted state is in the future (i.e., when the sensor's data is available before the odometry at the current timestamp). For state requests older than the oldest available odometry, the oldest data will be used. Our algorithm allows the odometry queue size to be specified. Additionally, a sensor data timeout can be set, which determines the maximum allowable delay between the sensor data and the current time, to avoid using too old data.

It is worth to note that the odometry data are usually not fully accurate enough for integration; indeed, small errors in speed measurements can accumulate over time, leading to drift in the estimated robot position and orientation. This makes the odometry data unsuitable for determining the robot's true current state without additional correction methods. But, this problem doesn't affect our Laser-Scan merger as it is based on odometry differences. The error that is introduced is therefore the error between two consecutive odometry data, and this is usually small enough to have good accuracy when transforming Laser-Scans. The full code logic to synchronise a Laser-Scan on a wanted timestamp is as shown in Fig. 3.

3) Global Fusion: Finally, we perform the global fusion of the aligned and synchronized Laser-Scans to create the wanted unified representation of the 3D environment in a 2D Laser-Scan. This step allows combining the data from all sensors to build a rich Laser-Scan that can be used for navigation or mapping. The fusion simply consists of comparing all Laser-Scans and keep the smallest ranges, which represent the closest obstacles. The overall algorithm merging the Laser-Scans from an unrestricted amount of sensors is as described in Fig. 4.

V. Experiments and Results

Our algorithm is implemented as a ROS2 package, making it easily integrable with any existing robot already using ROS2[4]. In our case, we integrated our ROS2 package with the navigation system of an omnidirectional robot called CALL-M. This robot is designed to collect cardboards and trash bags in a shopping mall, requiring it to navigate in an environment filled with complex-shaped objects and dynamic obstacles.

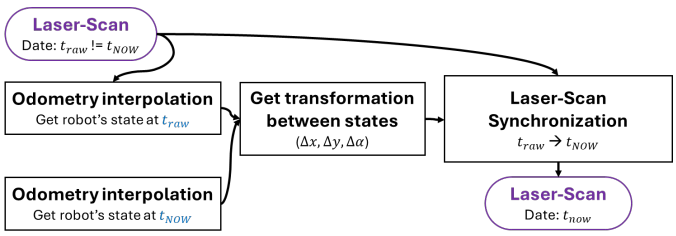


Fig. 3: Algorithm's logic to synchronize a Laser-Scan with current robot's state.

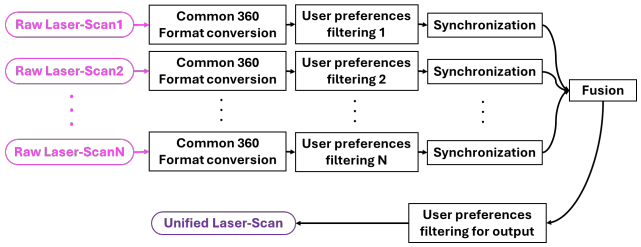


Fig. 4: Algorithm's logic for a consistent Laser-Scans fusion.

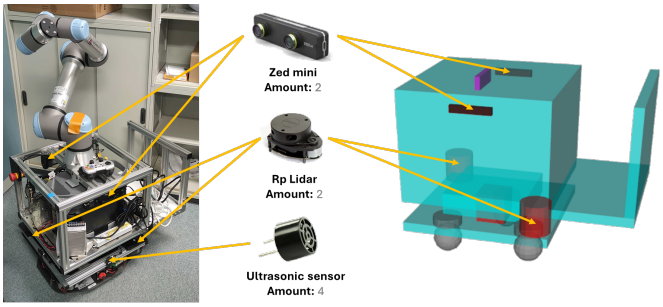


Fig. 5: CALL-M robot's sensors.

To achieve this, CALL-M is equipped with two 2D RpLidars SLAMTEC A1[1], two Zed-Mini[2] cameras, and four Ultrasonic sensors Maxbotic MB1403[3]. Also, the two cameras are mounted on servomotors allowing them to sweep up and down. The Ultrasonic sensors were added later and are not yet implemented in the ROS2 navigation system, so they were not used in the experiments presented here. The sensors were arranged as shown in Fig. 5, with the hardware on the left and the corresponding simulated robot model for the mobile base only (without the Ultrasonic sensors) on the right. Table I outlines how each sensor enhances the robot's environmental awareness and navigation capabilities.

The robot is also equipped with two computers, a Nvidia JETSON Orin NX 16 Gb[4] to manage GPU related packages (especially for the Zed-Minis) and a NUC 13 Pro Kit[5] to run other programs related to control. We use a TriOrb[6] mobile base allowing for omnidirectionnal motion, and UR5e robotic arm[7] for further grasping tasks. The robot is around 85 Kg (with the UR5e), has a rectangle footprint of 0.48 m x 0.74 m, and an height of 0.60 m without the UR5e on top.

The inputs to our algorithm are unrestricted but need to be Laser-Scans. Therefore, the data from the Depth Cameras have been converted to Laser-Scan using our developed

[1]https://www.slamtec.ai/product/slamtec-rplidar-a1/
[2]https://www.stereolabs.com/en-fr/store/products/zed-mini
[3]https://maxbotix.com/products/mb1403
[4]https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/
[5]https://www.asus.com/displays-desktops/nucs/nuc-mini-pcs/asus-nuc-13-pro/techspec/
[6]https://triorb.co.jp/en/
[7]https://www.universal-robots.com/products/ur5-robot/

LiDARs	Provide accurate 2D obstacle detection, robust to luminosity variations. The two LiDARs offer a 360° FoV.
Cameras	Allow 3D detection of obstacles, considering the height of obstacles. However, the two cameras combined cover only a 164° FoV. Also, data is sensitive to luminosity variations, fog, and other environmental factors.
Ultrasonic sensor	Currently not in use, but intended for detection of glasses and mirrors. They detect obstacles accurately within a certain range, though the position accuracy is limited.

TABLE I: Advantages and drawbacks of mounted LiDARs, Cameras and Ultrasonic sensors.

Specification	LiDAR	Camera
Laser-Scan Publish Rate (Hz)	8	14
Laser-Scan Resolution (number of points)	360	1440
Laser-Scan FoV (°)	360	82

TABLE II: LiDARs and Cameras ROS2 specifications.

(a) Simulated environment. (b) Map using LiDARs (c) Map using LiDARs and cameras

Fig. 6: Mapping using LiDARs Only and with Depth Cameras' data integration.

package presented in Section IV. This package appeared to be efficient and could run at more than 300 Hz[8] when processing 52000 depth points from each camera on a CPU i7-7700HQ. Additionally, given the LiDARs' arrangement, we needed to filter their outputs to prevent the robot's body from being detected and also to merge their data. As the LiDARs' outputs are Laser-Scans, we use them directly as inputs to our algorithm. So we can filter and fuse them with the cameras' Laser-Scans into a virtual LiDAR at the center of the robot. Also, our package facilitates this process of individually customizing each source easily through a .yaml ROS2 configuration file.

The performance specifications of each sensor are presented in Table II.

To validate our algorithm efficiency when merging data from two LiDARs and two Cameras, two experiments are presented in this paper. First, we demonstrate how our algorithm enhances 2D mapping by considering 3D obstacles with the cameras' data. Then, we show how our algorithm allows to lighten an existing navigation system while maintaining the navigation quality considering 3D obstacles. Both experiments presented here have been done in a simulated environment in Gazebo. Complications with the hardware of the mobile base prevented us to conduct those final experiments on the real robot. Those experiments also demonstrate that our package seamlessly enables the combination and customization of data from multiple sources.

A. Light mapping with LiDARs and Cameras Using ROS2

Fig. 6 shows that we could correctly fuse cameras' 3D data and 2D LiDARs' data into a rich unified Laser-Scan from which a rich map containing also the 3D obstacles information can be generated through a classic 2D SLAM. In our case `slam-toolbox` package has been used [14]. The left part shows the 3D scene used to perform the mapping. The middle part shows the map before integrating cameras, where we can note that only the footprints of tables and shelves appear on the map. However, considering the robot height and the arm that is mounted on it, we want to avoid the robot to go below these objects; thus, we had to consider 3D camera's data. The right part shows the map resulting from this integration. We can see that this map contains the appropriate obstacles that should be considered according to our robot's height, improving the environment knowledge and proving that we could correctly enhance the classical 2D SLAM using our package. The mapping of the shelves in the top left corner is incomplete because the robot did not navigate between each shelf to perform a thorough mapping during this experiment.

We noted that the generated map when integrating cameras contains thicker obstacles than the one using LiDARs. This effect doesn't appear from our package merging the Laser-Scans, but from our depth to Laser-Scan package converting camera's data to Laser-Scan. This is due to the high resolution of the cameras' Laser-Scan (1440 points like specified in Table II). Due to this high resolution, some inside points like holes between boxes or shelves' support might be considered as the closest points. But this effect doesn't add wrong points and the amount of inner points considered is negligible compared to the true closest point detected. This effect wasn't a problem for accurate mapping and navigation in our case. Anyway, our Depth to Laser-Scan converter package can be adjusted, and the resolution can be reduced to avoid this effect if needed.

B. Light navigation in dynamic Environment with LiDARs and Cameras Using ROS2

We also tested our approach in a dynamic environment where the robot needs to navigate considering 3D data from two cameras and 2D data from two LiDARs. To measure how our strategy allows to lighten the navigation, we used `Navigation2` ROS2 plugin (`NAV2`) [19]. This plugin contains several ready-to-use 2D path planning and controllers. And NAV2 can take as inputs Laser-Scans and point clouds, so we wanted to compare the performance when using our LiDARs' Laser-Scans and cameras' depth points directly as inputs to the default system of `Navigation2`, and when using only our rich Laser-Scan that contains all

[8]The measured frequency is based on the processing time for 52000 depth points. During runtime, the package fits to the maximum rate of the cameras' publications which was 14 Hz in our case.

Model	Acer Predator G9-793
Processor	Intel® Core™ i7-7700HQ CPU @ 2.80 GHz × 8
Memory	20 Gb
Graphic Card	NVIDIA GeForce GTX 1060 Mobile 6 Gb
OS	Ubuntu 22.04.4 LTS

TABLE III: Experiment's computer details.

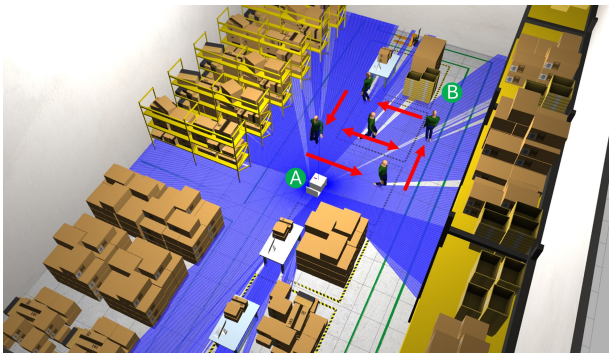


Fig. 7: Simulated world for the Navigation experiment.

the information. We operate the same navigation simulation through the same obstacles and compare the navigation system load and the duration to reach the goal. As shown Fig. 7, the experiment consists of asking the robot to go from point A to point B while navigating through the moving obstacles, represented by simulated humans. The details of the computer used to perform the experiment can be found in Table III.

We operate the experiment several times with different amounts of depth points from the cameras, and for each following method:

- Raw inputs: The two LiDARs and two cameras are directly given as inputs to `NAV2` navigation system.
- Merged Laser-Scan: The two LiDARs and cameras are merged into one Laser-Scan using our algorithm, and only this Laser-Scan is given to the navigation system.

Fig. 8 shows the measured duration to reach the goal, and the different loads (CPU and Memory) for each case.

First, we observe that the navigation duration is unaffected regardless of the method used. This demonstrates that our method maintains good navigation quality. Then, as more depth points are considered, the CPU usage of the default `NAV2` navigation system increases, while our algorithm allows to maintain a constant CPU load. This is because the default system only computes one Laser-Scan when using our method. And the `multi-laserscan-toolbox-ros2` package, processes fixed-size Laser-Scan inputs and has also a constant CPU Load. However, our approach requires an additional conversion of depth points to Laser-Scan using the `depth-filter-scan-converter` package described in Section IV.A.2.

As shown in Fig. 9, we also monitor the overall computer's system loads. A large amount of the measured loads are due to the simulated environment but we can still note a difference when using our method or the default `NAV2` navigation system. Especially with large amounts of depth points, using the default NAV2 system slows down the computer, with a simulation running at 40 FPS rather than the usual 61 FPS, because of the large CPU Load reaching 100%, but our algorithm still allows the computer to run properly considering the same sensors with a maximum load around 97%.

Those results demonstrate the efficiency of merging all sensors into a rich Laser-Scan using our algorithm for a lighter and enhanced mapping or navigation. Our rich Laser-Scan will enhance any higher-level algorithm using it for mapping or navigation.

VI. Conclusion

In this paper, we presented a lightweight approach to efficient 2D navigation using multimodal sensor fusion. Our method is based on converting data from various sensors into a standardized Laser-Scan format and performing fusion to create a rich Laser-Scan for mapping or any Laser-Scan based navigation system. Experiments using ROS2 demonstrated the effectiveness of our approach, showing similar quality navigation and significant reduced computational load compared to traditional 2D navigation methods considering 3D obstacles. Additionally, our approach proved to be fast enough for robustness and efficiency in handling dynamic environments and for enhancing the mapping of a common 2D SLAM with 3D obstacles data.

Our proposed method aims to provide a lightweight and modular ROS2 solution for fusing sensor data at the Laser-Scan level from an unrestricted amount of sensors, enabling efficient and accurate 2D navigation or mapping while considering 3D obstacles. It does not aim to replace existing methods for 2D mapping and navigation but rather seeks to enhance them by reducing their computational load, through the efficient formatting of sensor data into the Laser-Scan format and integrating data from multiple 3D and 2D sensors.

Future work will integrate ultrasonic sensors into the navigation system, requiring further study to convert their imprecise range data into Laser-Scan format. Additionally, future research will explore synergy with the robotic arm to enhance navigation by considering its motion.

References

[1] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Transactions on robotics*, vol. 32, no. 6, pp. 1309–1332, 2016.

[2] Y.-L. Zhao, Y.-T. Hong, and H.-P. Huang, "Comprehensive Performance Evaluation between Visual SLAM and LiDAR SLAM for Mobile Robots: Theories and Experiments," *Applied Sciences*, vol. 14, no. 9, p. 3945, 2024.

[3] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng *et al.*, "ROS: an open-source Robot Operating System," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[4] ROS, "ROS: An open-source robot operating system," https://www.ros.org/.

[5] ——, "Introduction to Working with Laser Scanner Data," http://wiki.ros.org/laser_pipeline/Tutorials/IntroductionToWorkingWithLaserScannerData.

[6] M. Labbé and F. Michaud, "RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation," *Journal of field robotics*, vol. 36, no. 2, pp. 416–446, 2019.

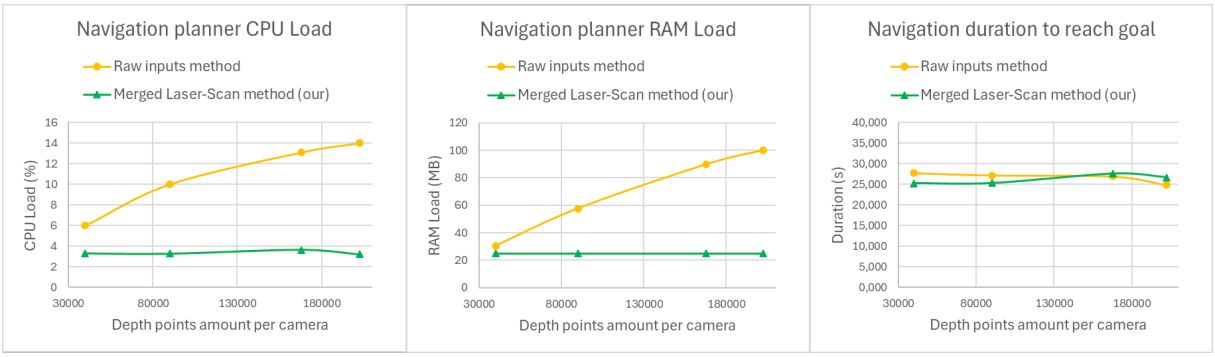


Fig. 8: Duration and Navigation system load for different cameras' amount of points and for each method.

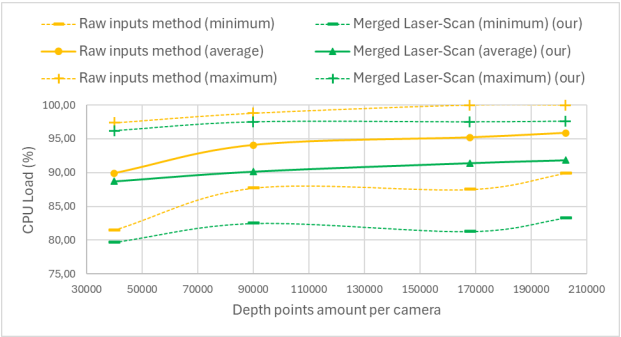


Fig. 9: Overall CPU load during Navigation for different amount of depth points and for each method.

[7] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. Montiel, and J. D. Tardós, "Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap SLAM," *IEEE Transactions on Robotics*, vol. 37, no. 6, pp. 1874–1890, 2021.

[8] Y. Nitta, D. Bogale, Y. Kuba, and Z. Tian, "Evaluating SLAM 2d and 3d mappings of indoor structures," in *ISARC. Proceedings of the international symposium on automation and robotics in construction*, vol. 37. IAARC Publications, 2020, pp. 821–828.

[9] T. Shan and B. Englot, "Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 4758–4765.

[10] O. Wulf, C. Brenneke, and B. Wagner, "Colored 2D maps for robot navigation with 3D sensor data," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, vol. 3. IEEE, 2004, pp. 2991–2996.

[11] O. Wulf, K. O. Arras, H. I. Christensen, and B. Wagner, "2D mapping of cluttered indoor environments by means of 3D perception," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, vol. 4. IEEE, 2004, pp. 4204–4209.

[12] O. Noel, "multi-laserscan-toolbox-ros2," https://github.com/Noceo200/multi-laserscan-toolbox-ros2.

[13] ROS, "ROS 2 Humble Documentation," https://docs.ros.org/en/humble/index.html.

[14] S. Macenski and I. Jambrecic, "SLAM Toolbox: SLAM for the dynamic world," *Journal of Open Source Software*, vol. 6, no. 61, p. 2783, 2021, http://wiki.ros.org/slam_toolbox.

[15] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE transactions on Robotics*, vol. 23, no. 1, pp. 34–46, 2007, http://docs.ros.org/en/hydro/api/gmapping/html/index.html.

[16] S. Kohlbrecher, O. Von Stryk, J. Meyer, and U. Klingauf, "A flexible and scalable SLAM system with full 3D motion estimation," in *2011 IEEE international symposium on safety, security, and rescue robotics*. IEEE, 2011, pp. 155–160, http://wiki.ros.org/hector_slam.

[17] W. Hess, D. Kohler, H. Rapp, F. Andert, and W. Burgard, "Real-Time Loop Closure in 2D LIDAR SLAM," https://google-cartographer-ros.readthedocs.io.

[18] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "Monte carlo localization for mobile robots," in *Proceedings 1999 IEEE international conference on robotics and automation (Cat. No. 99CH36288C)*, vol. 2. IEEE, 1999, pp. 1322–1328, http://wiki.ros.org/amcl.

[19] S. Macenski, F. Martín, R. White, and J. Ginés Clavero, "The Marathon 2: A Navigation System," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, https://docs.nav2.org/. [Online]. Available: https://github.com/ros-planning/navigation2.

[20] O. Noel, "ROS2 Vector Field Controller," https://github.com/Noceo200/ros2-vector-field-controller.

[21] R. N. S. Maintainers, "Costmap 2D Documentation," http://wiki.ros.org/costmap_2d.

[22] R. P. Maintainers, "Voxel Grid Documentation," http://wiki.ros.org/voxel_grid.

[23] R. N. S. Maintainers, "Move Base Documentation," http://wiki.ros.org/move_base.

[24] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997, http://wiki.ros.org/dwa_local_planner.

[25] C. Rösmann, F. Hoffmann, and T. Bertram, "Integrated online trajectory planning and optimization in distinctive topologies," *Robotics and Autonomous Systems*, vol. 88, pp. 142–153, 2017, http://wiki.ros.org/teb_local_planner.

[26] S. Thrun, "Probabilistic robotics," *Communications of the ACM*, vol. 45, no. 3, pp. 52–57, 2002.

[27] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3d object detection network for autonomous driving," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2017, pp. 1907–1915.

[28] Z. Wang, Y. Wu, and Q. Niu, "Multi-sensor fusion in automated driving: A survey," *Ieee Access*, vol. 8, pp. 2847–2868, 2019.

[29] J. Zhang, S. Singh *et al.*, "LOAM: Lidar odometry and mapping in real-time." in *Robotics: Science and systems*, vol. 2, no. 9. Berkeley, CA, 2014, pp. 1–9.

[30] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "ORB-SLAM: a versatile and accurate monocular SLAM system," *IEEE transactions on robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[31] R. P. Maintainers, "depthimage_to_laserscan," https://wiki.ros.org/depthimage_to_laserscan.

[32] ——, "pointcloud_to_laserscan," https://github.com/ros-perception/pointcloud_to_laserscan.

[33] O. Noel, "Depth point filter-converter to Scan," https://github.com/Noceo200/depth-filter-scan-converter.

[34] J. Bloomenthal and J. Rokne, "Homogeneous coordinates," *The Visual Computer*, vol. 11, pp. 15–26, 1994.

[35] M. Kam, X. Zhu, and P. Kalata, "Sensor fusion for mobile robot navigation," *Proceedings of the IEEE*, vol. 85, no. 1, pp. 108–119, 1997.

[36] V. E. Kremer, "Quaternions and SLERP," in *Embots. dfki. de/doc/seminar ca/Kremer Quaternions. pdf*, 2008.

Comparative Evaluation of Path Planning Algorithms for Autonomous Vessels in Dynamic Maritime Environments

Marie Dubromel, *Member, ENSTA, Brest, France,*

***Abstract*—This paper presents a comprehensive comparison of path planning algorithms for autonomous vessels, focusing on their efficiency and adaptability in dynamic maritime environments. Building on a previously developed Python-based simulator using the Artificial Potential Field (APF) method, this work extends the simulator by integrating four additional path planning algorithms: A* (A Star), D* Lite (for dynamic environments), Ant Colony Optimization, and Particle Swarm Optimization. The primary objective is to evaluate these algorithms based on their ability to plan efficient, collision-free paths while navigating complex maritime scenarios, including encounters with both manned and unmanned vessels. The performance of each algorithm is assessed through multiple criteria, including path length, computational runtime, and their ability to replan in real-time during dynamic vessel encounters. This paper provides a detailed analysis of how each algorithm performs in fixed and dynamic environments, offering valuable insights for enhancing the autonomy and safety of maritime navigation in compliance with the International Regulations for Preventing Collisions at Sea (COLREGs).**

***Index Terms*—path planning, USV, anti-collision, autonomous, simulator, maritime.**

I. Introduction

AUTONOMOUS vessels have emerged as a transformative innovation in maritime operations, offering significant advantages across commercial, tourism, and defense sectors. By eliminating the need for onboard crew, these vessels remove humans from potentially hazardous situations and enable advanced operations in increasingly hostile environments. However, achieving full autonomy for marine vessels requires robust and reliable control and guidance systems capable of handling diverse encounters with both manned and unmanned vessels. These systems must operate effectively under various sea conditions while guaranteeing safety and respecting the International Regulations for Preventing Collisions at Sea (COLREGs).

COLREGs [1], initially conceived in the 19th century, were designed with human sailors in mind, relying on their interpretation and execution of ambiguous language. Adapting these regulations for autonomous systems presents significant challenges, particularly in mixed scenarios involving both manned and unmanned vessels. Human navigators' often unpredictable behavior further complicates the task of designing collision avoidance systems. While traditional model-based approaches can address certain situations, their complexity makes them inadequate for the vast range of potential encounters and environmental conditions.

To address these challenges, modern advancements in autonomous navigation have explored various methodologies, including the development of path planning algorithms with avoiding-collision system. Last year, a Python-based simulator was developed to model and test path planning algorithms [2] , serving as a foundation for this work. The simulator initially used the Artificial Potential Field (APF) method to ensure a collision-free navigation. Using this simulator as a start, it has since been enhanced to incorporate and compare additional algorithms, including A* (A Star), D* Lite, Ant Colony Optimization, and Particle Swarm Optimization. Here is a list of the relevant COLREGs rules which have been implemented in the inital COLSim Simulator:

- Rule 8 - *Actions to avoid collision*: if there is sufficient sea-room, alteration of course alone may be most effective. Reduce speed, stop or reverse only if necessary.
- Rule 13 - *Overtaking*: Any vessel overtaking any other shall keep out of the way of the vessel being overtaken.
- Rule 14 - *Head-on*: Each head-on vessel shall alter her course to starboard so that each shall pass on the port side of the other.
- Rule 15 - *Crossing*: The vessel which has the other on her own starboard side shall keep out of the way.
- Rule 16 - *Actions by give-way vessel*: Take early and substantial action to keep well clear.
- Rule 17 - *Actions by stand-on vessel*: Keep her course and speed but may take action to avoid collision if the other vessel is not taking a COLREGs-compliant action.

This paper presents a comparative study conducted with the help of Tiphaine Calvier-Moisson of these algorithms, focusing on their efficiency, adaptability, and compliance with COLREGs. The performance of each algorithm is evaluated in both static and dynamic maritime environments, with metrics such as path length, computational runtime, and real-time collision avoidance capabilities. By analyzing these approaches, this work contributes to the development of adaptive and efficient guidance systems for autonomous vessels, paving the way for safer and more reliable maritime operations.

M. Shell was with the Department of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, 30332 USA e-mail: (see http://www.michaelshell.org/contact.html).

J. Doe and J. Doe are with Anonymous University.

Manuscript received April 19, 2005; revised August 26, 2015.

II. COLSim Simulator : Potential Artificial Field Algorithm

A. State of the art

Before the implementation of the COLSim, two other simulators have been studied in order to identify possible solutions for similar problem around USV in a simulator. The Fossen Simulator [3] is designed to simulate the behavior of different types of vehicles in a 3D simulation environment. It takes into account different simulation parameters such as gravity, friction, air resistance, and vehicle dynamics to simulate the movement and behavior of vehicles in real- time. Users can adjust simulation parameters to represent different types of vehicles and environments. Each vehicle is modeled as an object in Python and the vehicle class has methods for guidance, navigation and control. However, it does not take into account the concept of collision with other boats, and because of the accuracy of the simulation, the complexity of the program is a bit too high to be run a thousand times in an AI training context. However, the state equations chosen to implement a maritime environment such as waves and currents could be useful to upgrade the COLSim if needed in the future. // The UTSeaSim simulator [4] is a multi-agent simulation environment for underwater robotics research. It allows users to simulate underwater vehicles and their interactions with the environment, as well as communication between vehicles and with a surface station. The simulation environment includes several modules, such as a physics engine, a sensor module, a communication module, and a behavior module. The physics engine simulates the dynamics and kinematics of the underwater vehicles, while the sensor module simulates various sensors such as sonar and vision sensors. The communication module simulates acoustic and radio communication between vehicles and with the surface station. The behavior module is responsible for controlling the behavior of the vehicles in the simulation. It also uses an RTT algorithm to avoid obstacles wich in the context of the COLSim may not work if the obstacles are close.

B. Simulator A

To effectively evaluate COLREGs scenarios involving multiple vessels, both manned and unmanned, a simple yet realistic tool was needed, as existing complex tools like the ones presented in the state of the art, were not well-suited for this purpose. The COLSim is intended to support the development of reinforcement learning-based path planning algorithms while ensuring compliance with COLREGs. It also provides a benchmark for comparing the performance and efficiency of different path planning approaches, different than the APF initially implemented [5]. The Simulator A can be run in two different ways :

- The simulation with the scene with the different USV displayed
- The simulation without any display, but with the USV's following information : MMSI number, x, y, theta, v saved in a .csv file

The simulator also features four types of agents representing a variety of manned or unmanned surface vehicles, ranging from small boats to large commercial ships, along with dynamic objects like marine animals or drifting debris with unpredictable behaviors. Additionally, it includes a static agent to represent non-navigable areas, such as coastlines, on the navigation map. This diverse set of agents creates a comprehensive framework for testing and improving collision avoidance strategies. In real-world scenarios, vessels often come from different operators and follow distinct procedures, reflecting the simulator's design where no communication occurs between agents. This highlights the importance of each agent independently navigating and avoiding collisions.

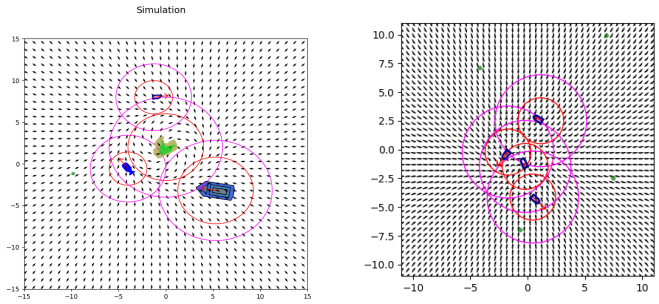


Fig. 1: Illustrations of different simulation runned with different types of agents

C. Simulator B : AIS data

The second simulator introduced offers the capability to recreate real-life scenarios using AIS data from various registered boats and add multiple ASV in the scene without causing any disturbance. The datasets are sourced from the AISHub website, providing an authentic and dynamic backdrop for simulation scenarios. With this simulator, users can immerse themselves in maritime environments, and understand the actual vessel movements and interactions. This simulator also takes into account the rules of the COLREGs to avoid any kind of collision.

Both simulators operate on a 2D map, simplifying scenarios by excluding variables like weather and sea conditions. This controlled planar environment enables a focused analysis of the effectiveness and efficiency of different path planning algorithms, free from the added complexity of environmental factors.

III. Comparison of different path planning algorithms

A. Local or global path planning

In their review of path planning algorithms for Maritime Autonomous Surface Ships (MASS) with a focus on navigation safety [6], researchers categorized various algorithms based on their suitability for local or global path planning and whether they employ heuristic or artificial intelligence-based methods. On the one hand, global path planning relies on prior knowledge of the environment, including obstacle positions and the final goal, and is typically designed for fully known environments. And on the other hand, local path planning deals with

unknown or partially known environments, requiring reactive strategies to adapt in real time. Traditional global approaches often use methods like cell decomposition and potential fields, which, while effective, lack the advanced adaptability of more intelligent systems. Local strategies, being reactive, are better suited to dynamically evolving scenarios and are capable of autonomously adjusting plans.

Similarly, a review of path planning strategies for mobile robot navigation [7] offers a detailed summary of popular algorithms. Drawing from over 200 papers, it highlights their applications in navigating static obstacles, dynamic obstacles, and dynamic goals, as well as their use in hybrid systems or multi-robot environments. While fewer studies focus on dynamic environments, the review concluded that reactive strategies generally outperform classical approaches due to their superior ability to manage environmental uncertainties.

For the simulator COLSim of the study, the constraints and requirements are clearly known. The simulation environment represents a **local navigation scenario** where obstacles are only known within the map, and while obstacles may be dynamic, the goal remains fixed. The algorithms are not intended for multi-robot systems and must generate efficient, smooth trajectories without unnecessary backtracking. Based on these criteria, five algorithms have been selected, each reflecting a distinct approach to path planning, to evaluate their performance in this controlled environment. These algorithms have been added to the COLSim [8], but unlike the APF, the COLRegs have been added into the new algorithms

B. Path planning algorithms

1) APF: APF follows potential lines and is the only algorithm that includes directives for adhering to the COLREGs (Convention on the International Regulations for Preventing Collisions at Sea), the sea navigation rules, among the implemented algorithms in the COLSim. When there is no nearby obstacle, a move_straight() function directs the agent towards its goal at each iteration. If the vessel is in a collision situation (an object is in his safety zone), the vessel will try to avoid the obstacle or other agents by following the COLREGs rules [2]. APF works by treating the goal as an attractive force and obstacles as repulsive forces, creating a potential field that guides the agent [5].

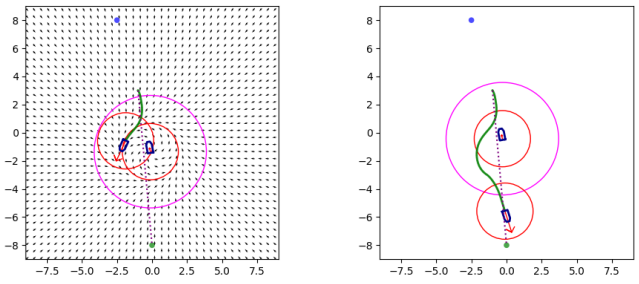


Fig. 2: Illustration of Rule 14 - Head-on situation from the COLREGs implemented in the simulator.

2) A Star: A* (A-star) is an efficient and widely used pathfinding algorithm that finds the shortest path between a start and goal node on a grid or graph as demonstrated in the article [9]. It is commonly applied in robotics, video games, and navigation systems. The algorithm balances two factors:

- g(n): The cost from the start node to the current node.
- h(n): A heuristic estimate of the cost from the current node to the goal.
- f(n) = g(n) + h(n): The total estimated cost of the path through node n.

A* iteratively selects the node with the lowest f(n) value by computing their g, h, and f values, and adds them to the open list if they haven't been explored., expands its neighbors, and updates their costs until it reaches the goal. If no path exists, it determines that no feasible route is available. If a neighbor is the goal node, the path is reconstructed by backtracking from the goal to the start. If the open list is emptied without reaching the goal, it indicates that no valid path exists. The simulator of this study requires an efficient local path-planning algorithm that operates in a dynamic 2D environment where obstacles and other vessels are present, therefore, this type of path planning algorythm could be interesting as it's :

- Optimized for Shortest Paths : A* guarantees finding the shortest path if the heuristic function h(n) is admissible (so it never overestimates the true cost). This is interesting in a real life scenario where saving fuel is environmentally and economically crucial.
- Handles Static and Dynamic Obstacles : While A* is traditionally used for static environments, it can be adapted for dynamic obstacles by recalculating the path when new obstacles appear. This makes it effective in maritime navigation, where other vessels may change course unpredictably.

3) D-Star-Lite: D* Lite (Dynamic A*) is an efficient incremental path-planning algorithm designed for environments where obstacles or terrain conditions can change over time. It is an improved version of the D* algorithm but is simpler to implement while maintaining the ability to efficiently replan paths in real-time when the environment changes as illustrated in the article [10]. Like A*, D* Lite uses a heuristic function to estimate the best path, but instead of computing the shortest path from start → goal, it works backward (goal → start) and updates only the necessary parts of the path when changes occur. This allows it to adapt dynamically without re-computing everything from scratch. If an obstacle appears or disappears, instead of recomputing the entire path, D* Lite only updates the affected regions of the previous path. This real-time adaptability is very useful considering the context of the study, where the simulation does not allow direct agent communication, each vessel must independently adjust its path when encountering new obstacles. It significantly reduces computational overhead, making it ideal for real-time applications. D* Lite will allow autonomous vessels to navigate dynamically without prior knowledge of all obstacles, improving collision avoidance.

4) ACO : Ant Colony Optimization: Ant Colony Optimization (ACO) is a bio-inspired algorithm used for solving complex optimization problems, espacially pathfinding and graph traversal. It is inspired by how real ants all work together to find the shortest route between their nest and a food source using pheromones. Instead of directly computing the shortest path like A* or D* Lite, ACO simulates a swarm of artificial ants that explore multiple paths and improve the best ones over time. Therefore, the 'ants' will move from the start to the end, leaving pheromones on their paths. The probability of choosing a path increases with the pheromone concentration and a heuristic value calculated using the potential field method. Over time, pheromones evaporate, preventing premature convergence on suboptimal solutions, and balancing exploration and exploitation. The process iterates until the best path is found or the maximum iterations are reached. Therefore, the shortest path is the one corresponding to the highest pheromone concentration and the best heuristic value.

5) PSO : Particle Swarm Optimization : This algorithm works for a known environment [11] and is also based on the concept of social interaction among animals searching for food or the social behavior of bird flocking or fish schooling. In PSO, each "particle"—explores the search space by adjusting its position and velocity based on its own experience (its best-known position) and the experience of the entire swarm (the global best-known position). Particles are initialized with random positions and velocities, and updated using formulas that incorporate:

- Their current velocity.
- The difference between their personal best and current position.
- The difference between the global best and current position.

Over time, this guides them toward more promising regions in the search space. They converge toward optimal or near-optimal solutions by sharing information and adapting their trajectories accordingly until a stopping criteria, such as the number of iteration in the case of this simulator. The PSO could work for the COLSim as its ability to explore multiple solutions concurrently means it can adapt to changes in real time by continuously updating the particles' positions. This would be very useful to avoid moving obstacles. Moreover, PSO is relatively simple to implement compared to other optimization algorithms. Its straightforward update rules make it easy to adapt for path planning purposes, allowing you to incorporate various constraints, such as adherence to COLREGs.
Another interesting aspect of this algorithm, is that it has a multi-agent approach, so this could to lend to scenarios where each vessel can be modeled as a particle or where a swarm of particles collectively explores the solution space for optimal navigation strategies.

C. Performance comparison

A* and D* Lite are easy-to-implement, grid-based algorithms, meaning movement is restricted to the eight neighboring cells (up, down, left, right, or diagonally to the four corners). This limitation can lead to less smooth and suboptimal paths, as vessels cannot move in a direct line unless it aligns with the grid. Reducing the grid step size can improve accuracy but increases computational cost.

Therefore, to address this, a path smoothing function was introduced by Tiphaine CALVIER-MOISSON. It removes unnecessary waypoints while ensuring the path remains obstacle-free. Using Bresenham's line algorithm, the function skips intermediate points if a direct path is clear, resulting in a more efficient and realistic trajectory. Although it adds a slight computational overhead, it significantly improves navigation quality.

To compare the algorithms various situations are gonna be analyzed, with USV with the same level of privilege:

- The trajectory of one vessel in a environment with one or two obstacles
- Crossing situations (Red to Red)
- Overtaking situations

The criteria to consider are whether the vessel reached the goal without any failures (not crossing the Distance to the Closest Point of approach DCPA displayed in red in *Fig 1 and 2*, the duration of simulation and length of the path.

A very important aspect to keep in mind during this analysis is that the APF algorithm can't be directly compared to the other algorithm as unlike the others, it already complies to the COLREGs. Therefore, its algorithm is more complex and can't be less effective on specific criteria than the A*, D*Lite, ACO, and PSO which were implemented in the simulator, with pre-existing versions adapted and improved to suit the simulator without complying to the COLREGs at all.

1) One vessel and one or two obstacles: The trajectories observed here with A* (orange curve, beneath the red curve in env. 2) and ACO (red curve) produce very appealing trajectories thanks to the path smoother. For PSO (purple curve), the use of splines creates a direct and visually pleasing path as well. D* Lite (green curve), on the other hand, may start with a good trajectory but often ends up stopping far from the goal.

According to these plots in Figure 3, it can be noted that the APF can get "stuck" in a potential minima zone, such as in environment 3, , blue curve. This occurs when the agent reaches a point where the combined attractive and repulsive forces balance out, resulting in no net force to push the agent towards the goal. Essentially, the agent gets "trapped" in a local minimum of the potential field, unable to move towards the goal or away from obstacles, as the algorithm perceives it as already being in a balanced state.

For the time and distance to reach the goal APF appears to be the best algorithm as illustrated in 4, as it is the fastest, and the priority here is to comply to the COLREGs, which explains why it could use a longer path to reach its goal.

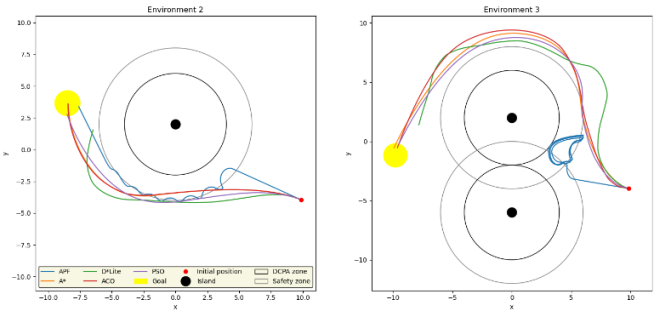


Fig. 3: Trajectories after simulations with different algorithms and obstacles

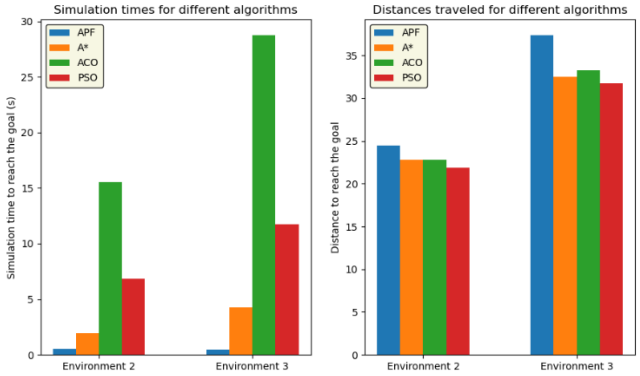


Fig. 4: Comparison of simulation times and distances traveled for different algorithms

2) Crossing situations (Red to Red): The results in the Appendix A (Fig5) show that some algorithms perform better than others because they ”anticipate” the behavior of the other vessels. Even without knowing the future trajectory of the USVs, certain algorithms, such as A*, ACO and PSO, planned to avoid the other USV from the very first step, resulting in a much smoother path. This proactive approach to path planning demonstrates the effectiveness of algorithms that can account for potential future movements, even in uncertain and dynamic conditions. With APF, both ships realized they were in a collision situation and entered each other's safety zone (marked as a collision, but is not a proper collision). The APF followed the COLREGs with the 'Red to Red' rule or Rule 14: for a head-on, each head-on vessel shall alter her course to starboard so that each shall pass on the port side of the other, unlike the others, which just chose the most suitable path for them according to their implementation.

3) Overtaking situations: According to the COLREGs, an ideal overtaking maneuver should occur as follows: the overtaking vessel can pass on either side and must keep clear of the vessel being overtaken. The vessel being overtaken has priority and should be able to maintain its course without altering it. APF still shows a sharp trajectory due to its lack of anticipation. In contrast, A*, ACO, and PSO achieve smooth trajectories, allowing the blue ship to overtake the orange one without altering its course. However, D*Lite fails to reach the goal, resulting in the two ships remaining in collision.

IV. Conclusion

This study compared several path planning algorithms for autonomous maritime navigation, highlighting their respective strengths and limitations. A* is easy to implement and efficient in static environments, producing smooth trajectories when combined with a path-smoothing function, though its grid-based limitations can lead to suboptimal paths without post-processing. Also, to deal with a more dynamic environment, modifications in the algorithm would be needed. For D Lite* it works very well in dynamic environments by replanning incrementally, but it sometimes fails to reach its goal, especially in complex scenarios. Ant Colony Optimization (ACO) offers smooth trajectories through probabilistic exploration, though its performance can be inconsistent due to its stochastic nature and sensitivity to parameter tuning. Similarly, Particle Swarm Optimization (PSO) produces direct and visually appealing paths using splines but shares ACO's variability across runs. Artificial Potential Field (APF), while not directly comparable due to its inherent COLREGs compliance, demonstrated strong performance in head-on and overtaking situations but often got trapped in local minima and produced less smooth paths. In the end, while A*, ACO, and PSO offer smoother and more anticipatory paths in crossing and overtaking situations, their consistency can be an issue. And D* Lite's adaptability makes it suitable for dynamic environments but requires refinements to ensure goal completion.

Therefore, this paper highlights the direction to take in order to keep enhancing the COLSim Simulator to have simulations of complex martime scenrios with autonomous and safe navigation path planning methods in compliance with the COLREGs. Indeed, future work should focus on hybrid approaches, and not only one path planning algorithm, that combine D* Lite's real-time replanning with the smoothness of A*, ACO, or PSO and keep a strong algorithm like APF, guaranteeing a safe path and adaptability in its code to achieve both COLREGs compliance and efficient, adaptable navigation in complex maritime scenarios for both manned and unmanned vessels.

APPENDIX A
TRAJECTORIES AFTER SIMULATION WITH DIFFERENT ALGORITHMS. TWO BOATS FACING EXACTLY EACH OTHERS (RED TO RED RULE FROM THE COLREGS)

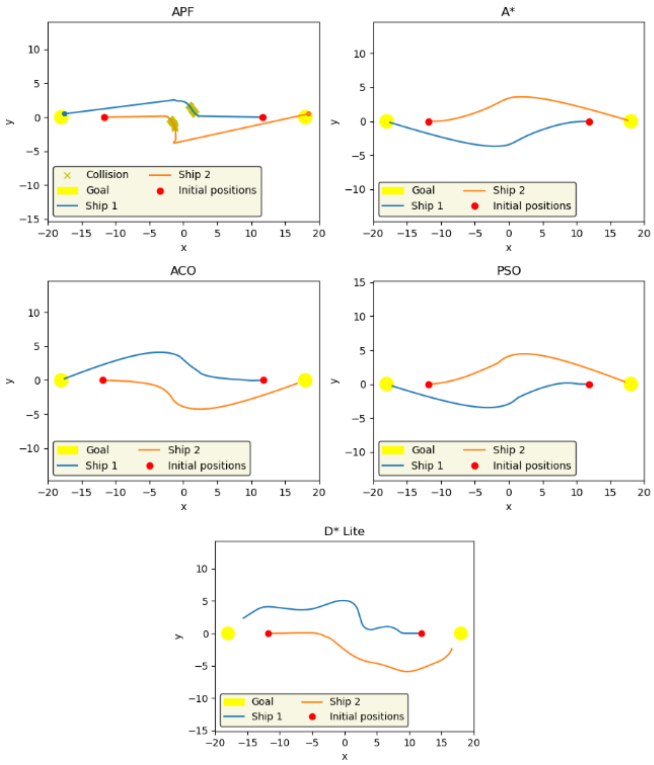


Fig. 5: Comparison of simulation times and distances traveled for different algorithms

APPENDIX B
TRAJECTORIES AFTER SIMULATION WITH DIFFERENT ALGORITHMS. ONE BOAT OVERTAKING ANOTHER BOAT

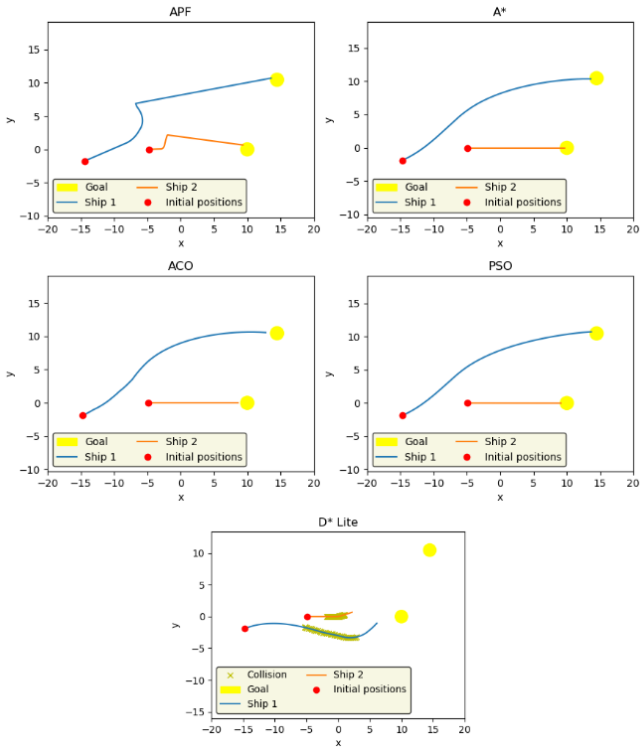


Fig. 6: Comparison of simulation times and distances traveled for different algorithms

REFERENCES

[1] A. Cockcroft and J. Lameijer, *A Guide to the Collision Avoidance Rules (Seventh Edition)*. Oxford: Butterworth-Heinemann, 2012.

[2] B. Clement, M. Dubromel, P. Santos, K. Sammut, M. Oppert, and F. Dayoub, "Hybrid navigation acceptability and safety," *Proceedings of the AAAI Symposium Series*, vol. 2, pp. 11–17, 01 2024.

[3] T. I. Fossen, "Python vehicule simulator," 2013. [Online]. Available: https://www.cs.utexas.edu/ UTSeaSim/

[4] E. Crase, C. Wideman, M. Noble, and A. Tarantola, "Utseasim documentation," p. 10, 2013. [Online]. Available: https://www.cs.utexas.edu/ UTSeaSim/download/1.0/Oct2013Documentation.pdf

[5] L. Jaulin, "Mobile robotics: Guidance," ENSTA Bretagne, Tech. Rep., 2023. [Online]. Available: https://www.ensta-bretagne.fr/jaulin/guidage.html

[6] Ülkü Öztürk, M. Akdağ, and T. Ayabakan, "A review of path planning algorithms in maritime autonomous surface ships: Navigation safety perspective," *Ocean Engineering*, vol. 251, p. 111010, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0029801822004334

[7] B. Patle, G. Babu L, A. Pandey, D. Parhi, and A. Jagadeesh, "A review: On path planning strategies for navigation of mobile robot," *Defence Technology*, vol. 15, no. 4, pp. 582–606, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214914718305130

[8] B. C. Marie DUBROMEL, Tiphaine CALVIER-MOISSON, "Colsim simulator," Tech. Rep., 2024. [Online]. Available: https://github.com/clemenbe/COLSim

[9] G. Tang, C. Tang, C. Claramunt, X. Hu, and P. Zhou, "Geometric a-star algorithm: An improved a-star algorithm for agv path planning in a port environment," *IEEE Access*, vol. 9, pp. 59 196–59 210, 2021.

[10] K. Al-Mutib, M. AlSulaiman, M. Emaduddin, H. Ramdane, and E. Mattar, "D* lite based real-time multi-agent path planning in dynamic environments," in *2011 Third International Conference on Computational Intelligence, Modelling Simulation*, 2011, pp. 170–174.

[11] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95 - International Conference on Neural Networks*, vol. 4, 1995, pp. 1942–1948 vol.4.