Dalhousie University
Intelligent System Laboratory

ENSTA Bretagne
Robotique Autonome

SURFACE VEHICLE NAVIGATION: ENHANCING THE DYNAMIC WINDOW APPROACH FOR OBSTACLE AVOIDANCE

Victor BELLOT

Supervisor:

Dr. Mae SETO

Dalhousie University

SECOND YEAR INTERNSHIP THESIS - May / August 2023

Contents

Abstract

This thesis presents my internship in maritime robotics at the Dalhousie University, Halifax, Nova Scotia, Canada. It focused on implementing the Dynamic Window Approach algorithm for obstacle avoidance on Clearpath Robotics surface vehicles. The internship involved algorithm development and testing in a simulated environment using ROS and Gazebo, followed by an implementation on a physical Clearpath Kingfisher robot prototype. This multidisciplinary internship encompassed simulation, hardware integration, project management, and collaboration within a research laboratory, enhancing technical skills in Linux system and networking. Finally, it contributed to improve my English language proficiency and presentation skills, and gave me a cross-cultural experience.

Résumé

Cette thèse présente mon stage en robotique maritime à l'Université Dalhousie, à Halifax, en Nouvelle-Écosse, au Canada. Il est axé sur la mise en œuvre de l'algorithme Dynamic Window Approach pour l'évitement d'obstacles sur des véhicules de surface de chez Clearpath Robotics. Le stage a impliqué le développement et les tests de l'algorithme dans un environnement simulé à l'aide de ROS et de Gazebo, suivis de la mise en œuvre sur un prototype physique du robot Clearpath Kingfisher. Ce stage multidisciplinaire a englobé la simulation, l'intégration matérielle, la gestion de projet et la collaboration au sein d'un laboratoire de recherche, renforçant les compétences techniques en Linux et en réseau. Enfin, il a contribué à l'amélioration de ma maîtrise de la langue anglaise, des compétences en présentation et à une expérience interculturelle.

Acknowledgements

I would like to extend my heartfelt gratitude to my tutor Dr Mae SETO, for her dedicated weekly guidance and support during this internship.

I am grateful to Luc JAULIN for his exceptional robotics courses, which gave me the training I needed to fully appreciate this internship.

I would also like to express my appreciation to my fellow interns in the laboratory, whose camaraderie and collaboration made the research environment truly enriching.

I also express my appreciation to the reader for taking the time to engage with this thesis.

Part I

Introduction

1.1 About this internship

ENSTA Bretagne is a French state graduate, post-graduate and research institute. It can be entered after two years of higher education (post high school education) for a three years curriculum. The first year at ENSTA Bretagne is composed of general engineering courses and lead to a Bachelor degree. From the second year on, students have to chose a specialty they will be studying until they graduate from ENSTA Bretagne (Master degree).

In between these two years of specialization, the engineer student must do an at least 12-week long internship. This is an opportunity to:

- better define a professional project
- use new acquired knowledge
- have an immersion in a multicultural environment

I have been passionate for mathematics and computer science since high school, and got an interest for research during my undergraduate years. Thus, I chose to specialize in robotics, a field where research is very active. In robotics at ENSTA Bretagne, I have learned to:

- sense the world (camera, IMU, GNSS, ...)
- communicate data within the robot and with its environment
- simulate a robot by modelling it
- do on-board state estimation (Kalman, interval)
- act on the environment using actuators

As ENSTA Bretagne is located in Brest, Brittany, by the Atlantic Ocean, my robotics formation have been more focused on mobile robot for maritime applications. Thus, I was looking for an internship in robotics research by the ocean. I wanted to go in a place surrounded by nature, where I can improve my English. Therefore, I did my internship in Dalhousie University, Halifax, Nova Scotia, Canada - the Canadian twin of ENSTA Bretagne on the other side of the Atlantic.

My supervisor was Dr. Mae SETO, an associate professor in Dalhousie's Mechanical Engineering Department. She is at the head of the Intelligent System Laboratory (ISL), which aims to make robots intelligent. Her research focus is the development of intelligent autonomous systems, and particularly for deployment in difficult environments like marine and under-ice.

(a) Dalhousie University

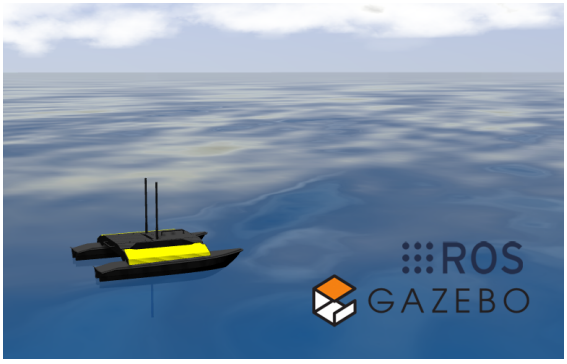
(b) Halifax, Nova Scotia

Figure 1: Pictures of the Dalhousie University in Halifax

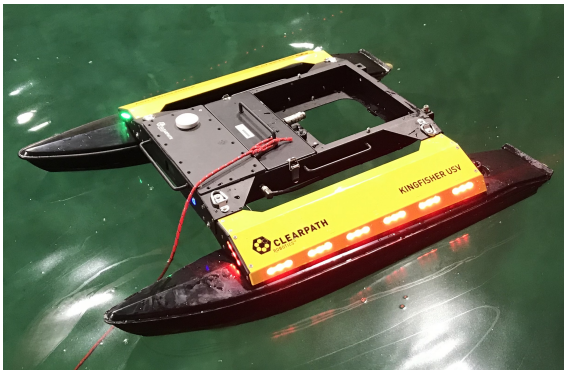
1.2 My project

I wanted to work on obstacle avoidance methods, and the ISL had a surface vehicle available for the summer. Thus, my project was to design an obstacle avoidance algorithm in simulation, and then transfer it on the ISL's Clearpath Kingfisher surface vehicle.

The overall goal was to navigate autonomously toward a target while avoiding obstacles to trow a rescue line. I hadn't the time to work on the line launching part. Thus, this report will focus on the design of the obstacle avoidance algorithm, and on the Clearpath surface vehicles (Kingfisher and Heron).



(a) The Clearpath Heron in Gazebo



(b) The Clearpath Kingfisher at the Aquatron

Figure 2: The Clearpath surface vehicles

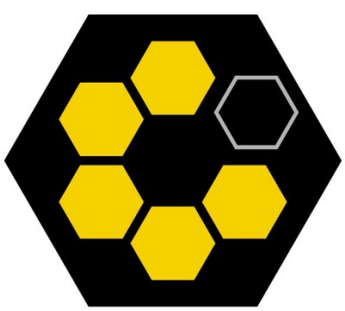
This project is a human experience in the ISL too. I was looking forward to working with others, presenting my work to others and helping each other.

Part II

Simulation

1 Clearpath Robotics, ROS and Gazebo

Clearpath Robotics is a Canadian robot and software manufacturer. Their products aim to make robotics research easier. Since its creation in 2009, Clearpath Robotics uses the ROS abstraction for their robots. Therefore, the company propose a simulation environment of their product in Gazebo.



(a) Clearpath Robotics

(b) ROS

(c) Gazebo

Figure 3: Logo

Thanks to UUV simulator Gazebo plugin [11], the Clearpath Heron gets a brand-new Gazebo simulation [4] in 2019 (4 years after this robot release). Recent maritime robotics simulation improvements [2] helps the creation of the VORC competition [3]. This framework accelerates research for ASV in difficult environment (custom sea state, wind, ...).

I had an introduction course at ENSTA Bretagne on ROS 2. But before this internship, I have never used the Gazebo environment. Thus, I spent the first few weeks learning about the differences between ROS and ROS 2, and playing around with Gazebo.

2 Clearpath Heron simulator

2.1 Robot presentation

The Heron robot is a dual hull surface vehicle equipped with two thrusters at its back. Its dimensions are 1300 mm length x 940 mm width x 340 mm mm height. It weights 20 kg and carry a 9 kg battery.

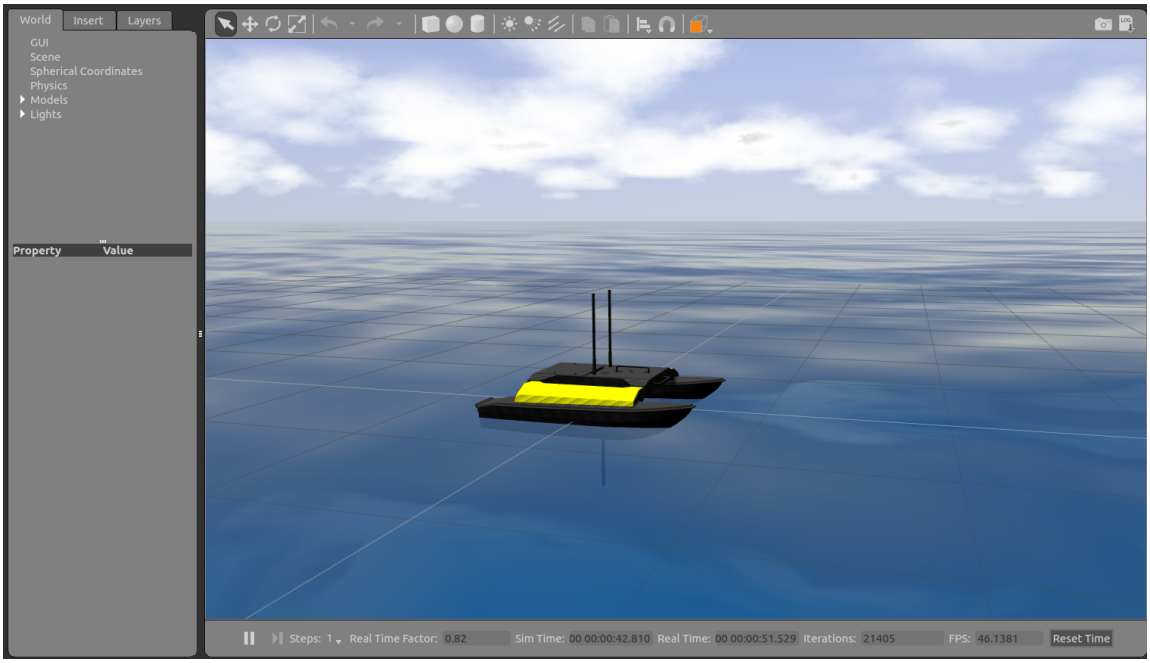


Figure 4: The Gazebo environment

The Heron's basic configuration is composed of a front camera, an IMU, a GNSS (with RTK option), and an internal thermometer. It also carries an embedded computer, and a radio interface for remote control.

To simulate the Heron in Gazebo-classic, I used ROS melodic on an Ubuntu 18.04 machine.

2.2 The Heron's packages

Clearpath Robotics provides a set of ROS packages [5] to describe, simulate, communicate, and control the Heron:

heron_control Manage localization and remote operation
heron_description Load in Gazebo Heron's URDF and publish it
heron_msgs Define custom messages
heron_srvs Define custom services
heron_controller Implement speed and yaw control
heron_viz Manage the RVIZ interface
heron_gazebo Manage the Gazebo physics simulation

The **heron_control** package uses the *navsat_transform_node* to convert GNSS data into local coordinates, and the *ekf_localization_node* for state estimation using an EKF. Both nodes are implemented in the **robot_localization** package.

We suppose the boat to be evolving in a 2D environment, therefore the robot's pose can be described by its x and y coordinates, and its heading ψ. The EKF tries to estimate the robot'pose and its time derivative. It uses the IMU to measure $\dot{\psi}$ and to estimate ψ. x, y and their time derivative are provided by the GNSS sensor.

Let's define $v = \sqrt{\dot{x}^2 + \dot{y}^2}$. The **heron_controller** package allows us to control the Heron's propellers using 5 different topics:

cmd_drive Control left and right propellers in power percents
cmd_wrench Apply force and torque mechanical actions
cmd_helm Command thrust in power percents and $\dot{\psi}$ in rad/s
cmd_vel v and $\dot{\psi}$ commands
cmd_course v and ψ commands

To achieve these control laws, the **heron_controller** package uses PID controllers to regulate ψ, v, and $\dot{\psi}$.

3 First work with the simulation

3.1 PID tuning

By default, the PID constants weren't fine-tuned. Thus, after getting familiar with the **heron** packages, I try to tune them. The PID controllers are implemented by the *Pid* class from the **control_toolbox** ROS package. Five constants can be adjusted :

forward gain Unnecessary for our application
proportional High to ensure quick response (of the order of 100)
integral Middle to overcome water drag (of the order of 10)
derivative Small to stabilize the system (around the unit)
integral bounds Unnecessary for our application

Thanks to these modifications, the system converges to the desired commands in few seconds. We are now ready for higher level control!

3.2 TOGO point mission

To start working with the Heron's simulator, I created a TOGO point mission : the robot tries to go to given surface coordinates at a predefined speed without avoiding obstacles. To manage high level control mission, A *planner* node have been created in a new **heron_mission** package.

With the cmd_course topic, we can control the boat's forward speed v and its heading ψ. Because $\bar{v}$ is chosen by the user, we only have to determine the target $\bar{\psi}$ given the TOGO point.

$$\bar{\psi} = \arctan\left(\frac{\bar{y} - y}{\bar{x} - x}\right)$$

Where $(\bar{x}, \bar{y})$ is the predefined TOGO point, and x & y are the EKF estimation of current robot's coordinates. Therefore, the mission is to follow this rule until the distance between the TOGO point and the estimate robot's position is less than the Heron's size.

$$\text{RUN UNTIL } (\bar{x} - x)^2 + (\bar{y} - y)^2 < heron\_size^2$$

In practice, this simple rule doesn't work very well. Indeed, whenever the boat is close to the goal without facing it, this controller makes the robot turn into circle around the target point. As satellite in orbit around the earth, it will never converge toward its center. This issue occurs because of the latency of the low level control loops, and because the robot is trying to turn while moving at high speed v (comparing to the distance to the target).

One way to fix this problem is to reduce the forward speed command $\bar{v}$ proportionally to the heading error between the boat and the goal point. Put another way, if the robot is not facing the target, we reduce the $\bar{v}$ command to ensure an efficient maneuver.

$$\bar{v} = v_{asked}\left(1 - \min\left(1, \frac{\left|\text{sawtooth}\left(\bar{\psi} - \psi\right)\right|}{turn\_cone}\right)\right)$$

Where v_{asked} is the forward speed asked by the user, ψ is the EKF estimation of the robot's heading, and $turn\_cone$ is an angle in radians (I chose $\pi/4$). If the heading error is greater than $turn\_cone$, $\bar{v}$ is set to zero. Otherwise, the speed command is inversely proportional to the heading error. The *sawtooth* function is used to compute angular differences, and defined by :

$$sawtooth(\theta) = 2\arctan\left(\tan\left(\frac{\theta}{2}\right)\right) = mod(\theta + \pi, 2\pi) - \pi$$

Bellow is a visualization of a GOTO point mission. The target is represented by a red buoy.

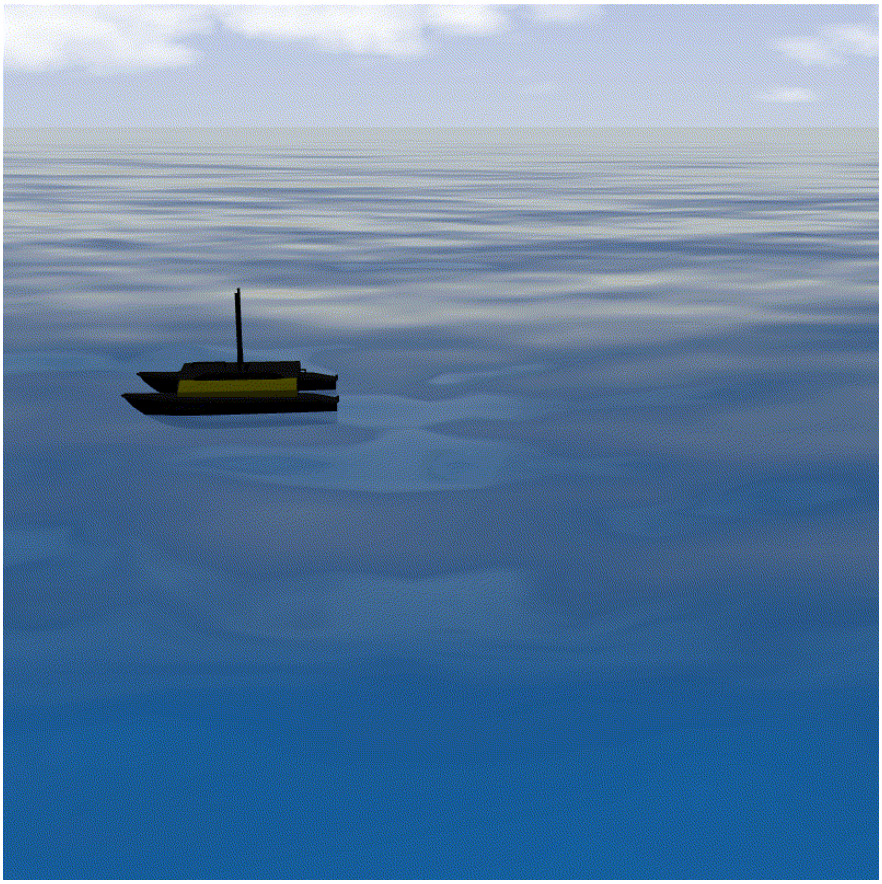


Figure 5: GOTO point mission

3.3 Trajectory mission

To generalize the idea of point following, I created a trajectory mission : the robot tries to follow a predefined trajectory $(\bar{x}(t), \bar{y}(t))$. A new ROS service /planner/traj_mission have been created for interacting with the *planner* node from the **heron_mission** package.

```
rosservice call /planner/traj_mission "{mission_name: 'circle', status: true}"
```

Listing 1: Trajectory mission service call

Providing a good estimate of the x & y robot's coordinates, of its yaw angle ψ and its forward velocity v, we want to control $\mathcal{P} = \begin{pmatrix} x \\ y \end{pmatrix}$.

We suppose the target trajectory $\bar{\mathcal{P}}$, its first and second time derivatives to be well known.

Using the *cmd_wrench* topic, we can act on $u_1 = \dot{\psi}$ and $u_2 = \dot{v}$. Therefore, let's define the error $e = \bar{\mathcal{P}} - \mathcal{P}$ and the controller $\mathcal{U} = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}$.

We want the error to obey the following dynamic : $\ddot{e} + 2\dot{e} + e = 0$. Let's compute the time derivatives of $\mathcal{P}$:

$$\dot{\mathcal{P}} = v \begin{pmatrix} \cos\psi \\ \sin\psi \end{pmatrix} \quad \text{and} \quad \ddot{\mathcal{P}} = A\mathcal{U} \quad \text{where} \quad A = \begin{pmatrix} \cos\psi & -v\sin\psi \\ \sin\psi & +v\cos\psi \end{pmatrix}$$

Thus, in the case where A is invertible (that is to say $v \neq 0$), the controller is :

$$\mathcal{U} = A^{-1} \left(\frac{\mathrm{d}^2}{\mathrm{d}t^2} \bar{\mathcal{P}} + 2\dot{e} + e \right)$$

An issue I encountered with this controller is latency : the boat is following the target from a constant distance apart. To fix this bias, I introduce a drag coefficient c_d such that $\ddot{\mathcal{P}} = A\mathcal{U} - c_d v$. Therefore, the new controller is :

$$\mathcal{U} = A^{-1} \left(\frac{\mathrm{d}^2}{\mathrm{d}t^2} \bar{\mathcal{P}} + 2\dot{e} + e + c_d v \right)$$

That way, the robot counters water's drag : the fastest it tries to go, the more overshot mechanical actions are. It is a better physical model of the Heron on water behavior.

3.4 LIDAR modelling

Before diving into obstacle avoidance simulation, we need to model how the Heron sens the world. As a general approach, we will model a LIDAR to produce a point cloud of the surrounded surface obstacles.

Even if the Heron packages already contain an implementation of the Sick LMS1xx LIDAR [12], I wanted to learn how to model a sensor in the ROS-Gazebo environment. Thus, I design a rotating multibeam LIDAR.

At first, I used SDF to describe the sensor. This description format is closer to Gazebo, therefore it was easy to describe the sensor and its interaction with the environment. The drawback is that it's an old format. So it's quite heavy to write, and most of community's plugins work better with URDF.

So I ended up using URDF. It has the *xacro* functionality, which enable us to write smarter code: variables, functions, computable expressions... I attached the *gazebo_ros_block_laser* Gazebo plugin from the **gazebo_ros** package to my LIDAR description. This plugin simulates the behavior of a comb of lasers, and for each of them estimates the distance toward the closest obstacle. Combine with a plugin making the comb spinning, we get a good idea of what the surrounding looks like.

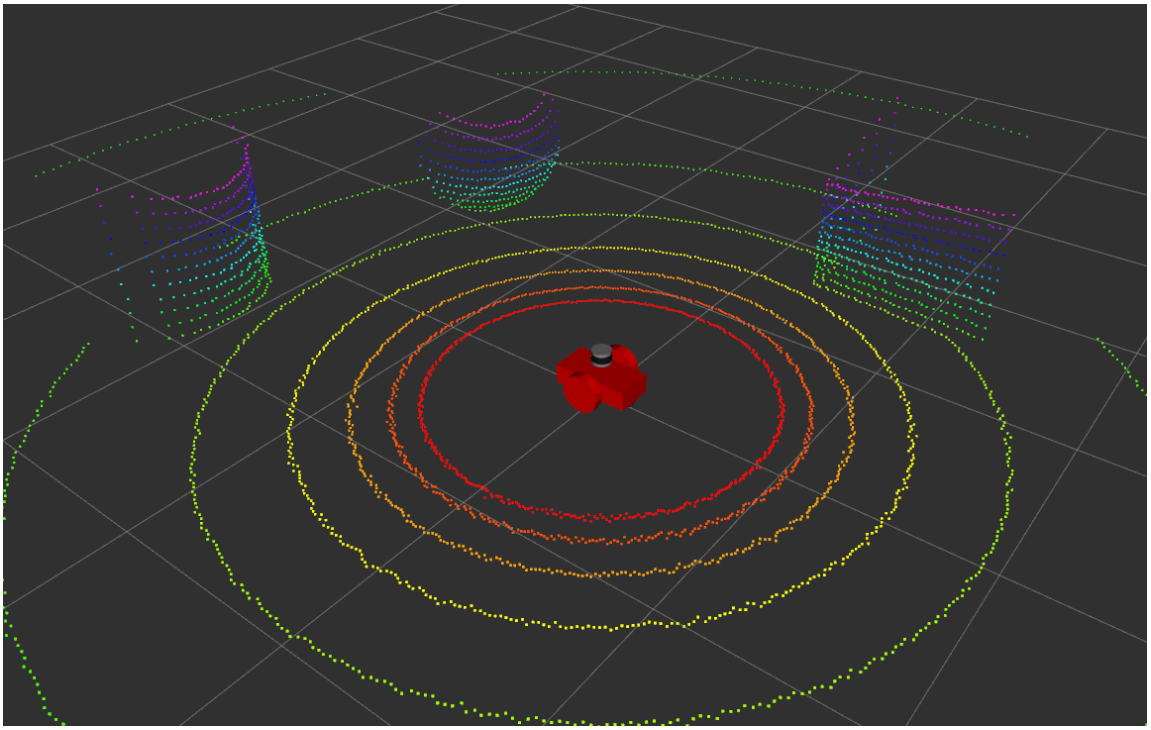


Figure 6: The spinning LIDAR comb in RVIZ

I decided to finally use the LIDAR model provided by Clearpath Robotics. As the Heron the pretty flat, there is no need to have a comb of lasers. Moreover, making the LIDAR spin required a lot of computational resources.

I only add a node using the *LaserGeometry* class from the **ros_perception** package to convert *LaserScan* messages into *PointCloud* messages. That way our obstacle avoidance algorithms will except cloud of points as obstacles representation.

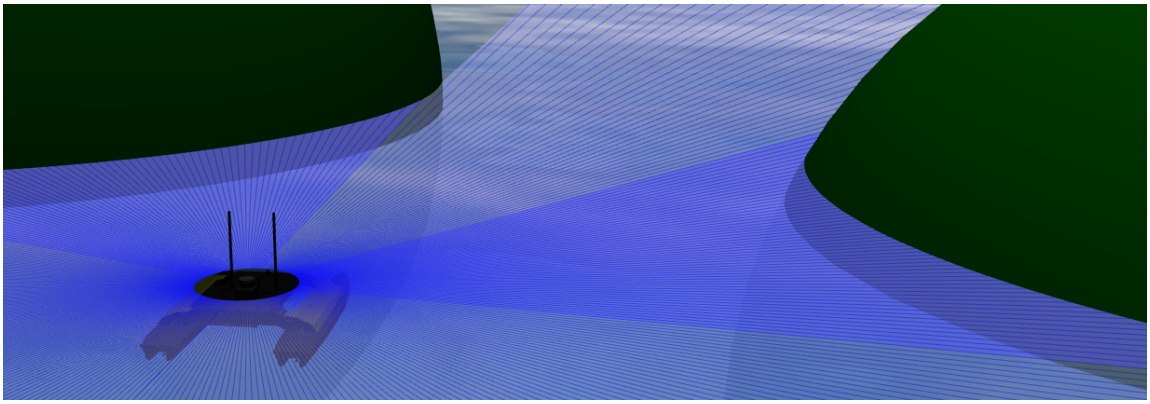


Figure 7: The Sick LMS1xx LIDAR in Gazebo

4 Dynamic Window Approach

4.1 About obstacle avoidance algorithms

We want to create an algorithm to drive the Heron surface vehicle toward a predefined goal while avoiding obstacles. It could be nice if our system can adapt to its surrounding, by taking into account moving objects.

Before introducing the algorithm I implemented, here is an overview of tools and ideas used to create obstacle avoidance systems. I like to distinguish them between three properties:

- TYPE
  Path planning theoretical, discreet & optimal
  Motion planning physical, continuous & adaptive
- ENVIRONMENT
  Static ignore obstacles movement - low reactivity
  Dynamic take obstacles motion into account - high reactivity
- PLANNING HORIZON
  Local compute path as it goes along
  Global compute the entire path leading the target

A* (1968) [8]

A* is a path planning algorithm using a discreet representation of the world. Because it aims to find the shortest path toward the goal, it's quite computationally heavy. Thus, this method can't take into account quick environment changes.

D* (1994) [13]

D* solves the static issue of A* by running partial A* as new obstacles are encountered. It's a path planning algorithm using cellular representation of the world too. Partial A* is based on localization and mapping: the robot needs to have a good idea of what's around it and where it is in order to place newly discovered obstacles.

Potential Field methods (70s) [1]

Potential Field methods work in continuous space. Inspire by physics, the target produces a positive potential and the obstacles a negative one. At each time step, the robot takes a step along the potential's gradient. These methods handle moving goal and obstacles, but can't ensure non collision.

Bug like methods (80s) [14]

Bug like methods are local motion planning algorithms. The robot moves toward the goal until an obstacle is encountered. Then it follows the obstacle's boundary. Different heuristics exist to determine when to move toward the goal again.

Probabilistic RoadMap (1996) [9]

PRM works in the robot's state space by sampling points from the admissible space and connecting them to their closest neighbors. When the graph reaches the desired accuracy, the Dijkstra's algorithm is applied to determine the shortest path toward the desired state. How admissible space is defined, and the point connection policy embody the physical interaction between the robot and the environment.

Rapidly exploring Random Tree (1998) [10]

RRT is a global motion planning algorithm that try to find a path in the robot's state space by building a tree rooted at the robot's starting configuration. For the tree to grow, a point is sampled from the free space to give the direction to go from its closest admissible tree point. This operation is repeated until the goal is considered to be reached.

4.2 DWA presentation

In 1997, Dieter Foxy, Wolfram Burgardy, and Sebastian Thrunyz introduce a new reactive collision avoidance method for mobile robots : the Dynamic Window Approach [7]. This is a motion planning algorithm that is highly responsive to environment changes, that act locally by taking decisions step by step.

At each time step, the DWA algorithm chose a dynamic Δ to follow during the next time step. This method only consider circular trajectories, thus Δ is a pair composed of a forward velocity v and an angular velocity ω. To choose the next dynamic, a search space is computed. The selected dynamic Δ is the element of the search space maximizing the objective function $G(v, \omega)$.

The search space is defined as the intersection of the dynamic window space and the admissible dynamics space. The dynamic window space is composed of the dynamics that can be reached within a short time interval given the limited accelerations of the robot. A dynamic $\Delta = (v, \omega)$ is considered admissible, if the robot is able to stop before it reaches the closest obstacle on the circular trajectory associated with Δ.

In its vanilla version, the objective $G(v, \omega)$ has the following expression:

$$G(v, \omega) = \alpha \cdot \text{heading}(v, \omega) + \beta \cdot \text{clearance}(v, \omega) + \gamma \cdot \text{speed}(v)$$

where α, β and γ are three scaling constants, $\textit{heading}(v, \omega)$ measures the alignment of the robot with the target direction, $\textit{clearance}(v, \omega)$ represents the distance to the closest obstacle that intersects with the curvature, and $\textit{speed}(v)$ determines how fast the robot is about to move.

In their original paper [7], the authors prove that in a first order approximation, a robot following a dynamic (v, ω) describe a circular trajectory of radius v/ω. Therefore, it's quite easy to compute where the robot will be at the next time step. This predicted position is used to compute $\textit{heading}(v, \omega)$ and $\textit{clearance}(v, \omega)$.

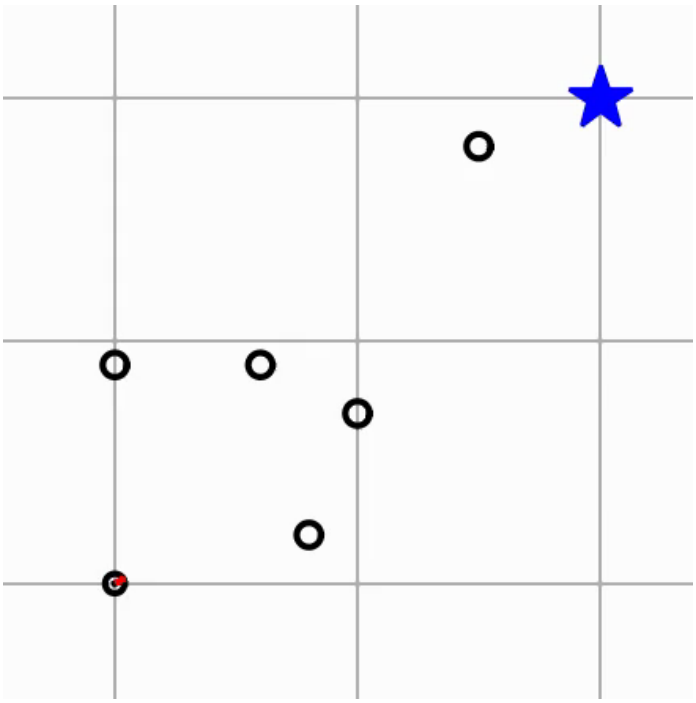


Figure 8: DWA illustration

4.3 Implementation and Limits

Implementation

An implementation of the DWA algorithm can be found in the *dwa_local_planner* package [6] from the **ros-planning/naviguation** project. However, I wanted to learn by doing and to be able to contribute some new features, so I implemented the DWA myself in my *planner* node in the package **heron_mission** using the C++ programming language.

The first thing I had to code, was the computation of the longest distance the robot can travel along a given trajectory without colliding any obstacles. For punctual trajectories, this is trivial. For linear trajectories, I used the Pythagorean theorem. And for circular trajectories, the Al-Kashi theorem. Then to assess potential dynamics, I used the **tf2** package to transpose the point's cloud representation of obstacles into the simulated robot's frame.

Limits

After having implemented the DWA algorithm as described above, I set up a simple test environment in Gazebo to try it. The first robot's mission was to go forward by about 20 m while avoiding a spherical obstacle located 10 m ahead. This very simple mission shows the importance of the dynamic window constrain. In fact if this window is too broad, the robot become undecided for too long. At one time step it decides to turn around the obstacle by the left, and the next one to take it by the right. This behavior appears because once it starts to go left, it can still go right without colliding the obstacle. Changing its mind helps to maximize the *heading* component of the objective function. This is an issue because if the dynamic window is too broad, the time when the robot can't be undecided anymore occurs too late : it is already too close to the obstacle. That way, the dynamic window width is a parameter that can be adjusted to determine how far from an obstacle the robot need to stop being undecided.

The issue I have just mentioned occurs because of the optimistic aspect of the DWA algorithm as well. As a matter of fact, the real trajectory the robot follows is not a circle. Because of the drift effect of the Heron on water, the actual trajectory is not as tight as expected. But the dynamics are chosen to maximize the objective function, that is to say, to take the trajectory the closest to the obstacle in order to optimize the *heading* criterion. Therefore, at first the algorithm chose to go close to the obstacles thinking it won't collide, but because the effective trajectory is worse than expected, the algorithm has to change plan. This can lead to a stuck state - when the search space is only made up of zero forward velocity dynamics.

Another issue the DWA has is that is's a too local method. Because it imagines following a circular trajectory, it can't consider more complex plans. This explains some inconsistent behaviors, such as going round in circles.

4.4 Proposed improvements

To make this method less local, I added the possibility to look several steps ahead. It works like a tree search algorithm : from each possible dynamics, all the children's dynamics are computed to

assess their parent. It's quite computationally heavy, so with in a dynamic space made of 5 possible forward velocities and 9 possible angular velocities, I can't exceed a 3 depth search.

In order to counter the optimistic aspect of the DWA, I extended the definition of a dynamic. Now a dynamic is a triplet Δ_s composed of a forward velocity v, an angular velocity ω and a security factor f_s. If an obstacle point is at a distance less than d_s from a given trajectory, we considered that trajectory to collide this obstacle. The distance d_s is called security distance, it has to be greater than the robot size d_r. Therefore, we have : $d_s = f_s \cdot d_r$. This new parameter has to be optimized by the objective function as well :

$$G(v, \omega, s_f) = \alpha \cdot \text{heading}(v, \omega) + \beta \cdot \text{clearance}(v, \omega) + \gamma \cdot \text{speed}(v) + \delta \cdot \text{security}(s_f)$$

The security factor aims to give some flexibility to the DWA algorithm. Because this method is optimistic, it can't achieve the security distance it plans to ensure. With the security factor variable, the algorithm will try to achieve the greatest s_f possible and if it fails to do that, it can try a less restrictive one without getting stuck. In fact, the security factor creates a compromise between the *heading* and the *security* components of the objective function. By tuning the range in which s_f is picked, we can adjust the amount of risk the algorithm is taking.

Finally, I programmed some limit cases the DWA was handling poorly.

- When the robot gets stuck, it will turn until a free direction is found to escape.
- If there are no obstacles between the robot and the target position, it will act as if it was a GOTO point mission (go in a straight line).
- To prevent inconsistent behaviors, I bounded the search space in such a way that $|v \cdot \omega| < k_d$.

4.5 Result and discussion

I tested this modified version of the DWA in a lot of static environments. It completes its mission each time the mission can be achieved while ensuring at least the smallest security factor authorized. However, in complex environment, the path taken by the robot is far from being optimal. It's still a local algorithm, it's better to use it to react quickly to close obstacles. It needs to be coupled with a global path planning method so that it performs well on the two scales.

Part III

Prototyping

5.1 Kingfisher presentation

I had the Clearpath Robotics Kingfisher surface vehicle at my disposal in the ISL. It's the former version of the Heron I simulated, therefore Clearpath doesn't support it anymore. To be compatible, the Kingfisher's system need to be changed from an Ubuntu 14.04 with ROS Indigo to an Ubuntu 18.04 with ROS Melodic. Because the current configuration had a 32-bits architecture, I changed the embedded computer to a NVIDIA Jetson nano. Everything that has to do with the restoration of the Kingfisher (software parts) can be found on this GitHub Project.

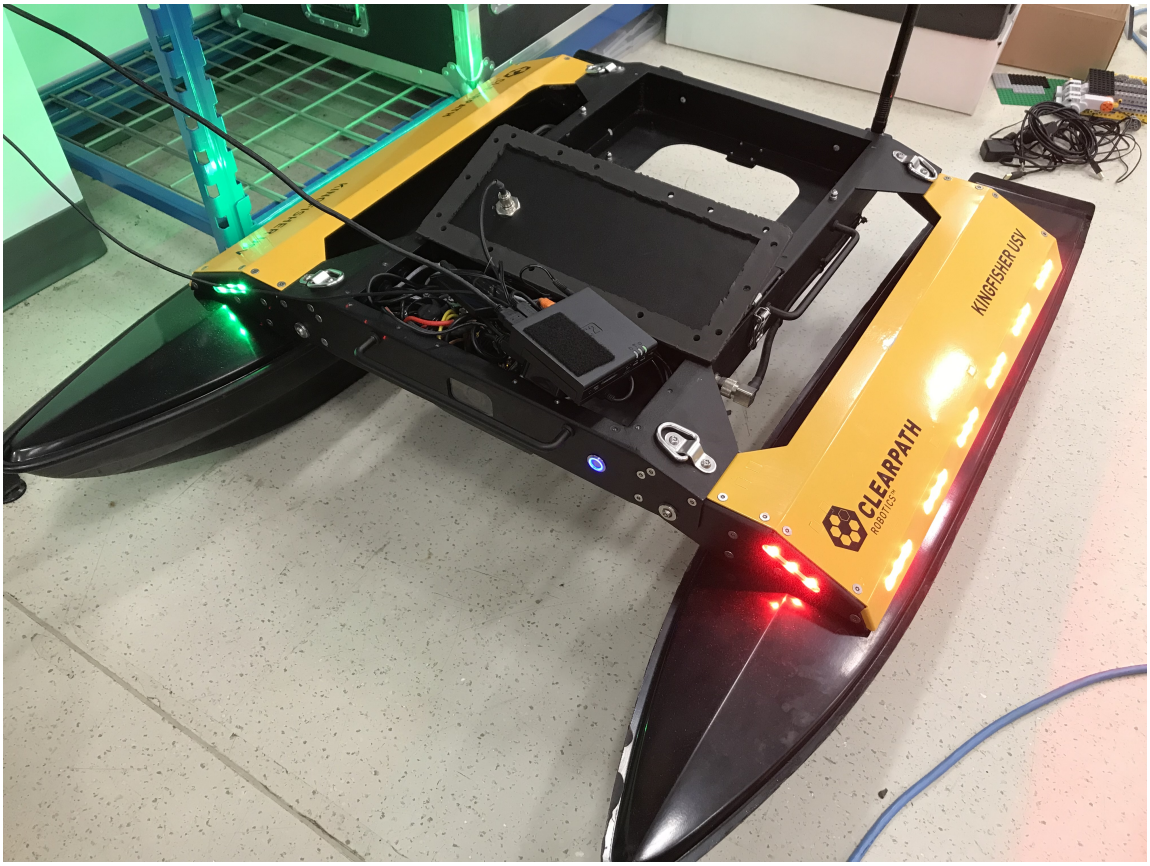


Figure 9: The Kingfisher in the ISL

What's inside now?

Embedded computer NVIDIA Jetson nano

USB hub to connect the I/O to the main robot's computer

Battery pack 14.4V NiMH 40Ah

Front camera I don't know the reference (I hadn't used it)

GNSS NEO-6 U-blox 6 from 2011

IMU CH Robotics UM6 from 2012

Radio antenna to communicate with a radio R/C

Blinking lights red on port side and green on starboard side

2 DC motors for the back thrusters

I used the previously presented ROS packages provided by Clearpath Robotics for the Heron, as they have the same architecture. Only the **heron_robot** package needed to be added to manage I/O with the sensors and the actuators.

5.2 Working with the network

A base station was delivered with the Kingfisher. It aims to create a local network to which the robot can connect. With a PC connected to this same network, instructions can be sent remotely to the Kingfisher.

I configured the MicroHard router of the base station to create the desired network, and I made the Kingfisher automatically connected to this custom network. Then, the Kingfisher's ROS can be accessed by a master PC connected to the same network.

```
1    ssh kingfisler@192.168.1.111
2    export ROS_IP=192.168.1.111
3    export ROS_MASTER_URI=http://$ROS_IP:11311
```

Listing 2: Networking setup for a ssh control

Figure 10: From left tot right : the Kingfisher, a master PC, and a base station.

5.3 Trials

We went to the Aquatron - the Dalhousie University's aquatic research facility - where experiments in biology and robotics are conducted. The goal of this first on water trial was to assess the performance of the thrusters, the robot behavior on water and the IMU data. I recorded ROS bags of the IMU'estimate of orientation and velocities, to analyze them afterward.

To test the GNSS, we went to a park (an open-air place). When the ROS system starts, a local frame is created at the current robot's pose (origin at the robot's position, positive x along the starboard side of the robot and positive y along the front of the robot). This local frame can be defined by hand by giving a GPS position and an orientation.

Part IV

Conclusion

In conclusion, my internship at Dalhousie University has been a remarkable journey of learning, growth, and achievement. Over the course of my internship, I focused on the implementation of the Dynamic Window Approach algorithm for obstacle avoidance on Clearpath Robotics surface vehicles. This experience has provided me with invaluable insights and skills in the field of robotics, particularly in the context of autonomous navigation in maritime environments. One of the significant accomplishments of this internship was the development of an improved version of the DWA algorithm in a simulated environment using ROS and Gazebo. Translating the algorithm from simulation to a physical Clearpath Kingfisher robot prototype was a challenging yet rewarding endeavor. This transition taught me the importance of adaptability, problem-solving, and attention to detail when dealing with the complexities of real-world hardware. Collaborating with a team of students and researchers in a laboratory setting provided me with invaluable insights into teamwork, project management, and the collaborative nature of research in the field of maritime robotics. Throughout my internship, I also honed my skills in Linux, network configuration, and troubleshooting. Additionally, I had the opportunity to improve my English language communication skills, particularly in the context of presenting my work and expressing complex technical concepts, which will undoubtedly be beneficial in my future career. Living and working in a different country has allowed me to develop both personally and culturally.

Glossary

ENSTA École Nationale Supérieure de Techniques Avancées

ISL Intelligent System Laboratory

ROS Robot Operating System

UUV Unmanned Underwater Vehicle

VORC Virtual Ocean Robot Challenge

ASV Autonomous Surface Vehicle

USV Umanned Surface Vehicle

IMU Inertial Measurement Unit

GNSS Global Navigation Satellite System

RTK Real Time Kinematic

URDF Unified Robotics Description Format

RVIZ Ros VISualization

EKF Extended Kalman Filter

PID Proportional Integral Derivative

LIDAR LIght Detection And Ranging

SDF Simulation Description Format

PRM Probabilistic RoadMap

RRT Rapidly exploring Random Tree

DWA Dynamic Window Approach

References

[1] J. Barraquand, B. Langlois, and J.-C. Latombe. "Numerical potential field techniques for robot path planning". In: *IEEE Transactions on Systems, Man, and Cybernetics* 22.2 (1992), pp. 224–241. DOI: 10.1109/21.148426.

[2] Brian Bingham et al. "Toward Maritime Robotic Simulation in Gazebo". In: *OCEANS 2019 MTS/IEEE SEATTLE*. 2019, pp. 1–10. DOI: 10.23919/OCEANS40490.2019.8962724. URL: https://ieeexplore.ieee.org/document/8962724.

[3] Ocean Robotics Challenge. *Virtual Ocean Robotics Challenge*. URL: https://www.oceanroboticschallenge.com.

[4] clearpathrobotics. *Heron USV gets a new simulator*. URL: https://clearpathrobotics.com/blog/2019/01/heron-usv-gets-a-new-simulator/#:~:text=The%20simulator%20relies%20on%20the,compute%20the%20Heron%27s%20water%20physics..

[5] clearpathrobotics. *HERON USV TUTORIALS*. URL: https://www.clearpathrobotics.com/assets/guides/kinetic/heron/index.html.

[6] The ROS community. *DWA local planner*. URL: http://wiki.ros.org/dwa_local_planner.

[7] D. Fox, W. Burgard, and S. Thrun. "The dynamic window approach to collision avoidance". In: *IEEE Robotics and Automation Magazine* 4.1 (1997), pp. 23–33. DOI: 10.1109/100.580977.

[8] 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 (1968), pp. 100–107. DOI: 10.1109/TSSC.1968.300136.

[9] L.E. Kavraki et al. "Probabilistic roadmaps for path planning in high-dimensional configuration spaces". In: *IEEE Transactions on Robotics and Automation* 12.4 (1996), pp. 566–580. DOI: 10.1109/70.508439.

[10] S. LAVALLE. "Rapidly-exploring random trees : a new tool for path planning". In: *Research Report 9811* (1998). URL: https://cir.nii.ac.jp/crid/1573950399665672960.

[11] Musa Morena Marcusso Manhães et al. "UUV Simulator: A Gazebo-based package for underwater intervention and multi-robot simulation". In: *OCEANS 2016 MTS/IEEE Monterey*. IEEE, Sept. 2016. DOI: 10.1109/oceans.2016.7761080. URL: https://doi.org/10.1109%2Foceans.2016.7761080.

[12] Sick. *Sick LMS1xx LIDAR*. URL: https://www.sick.com/it/en/lidar-sensors/2d-lidar-sensors/lms1xx/c/g91901.

[13] A. Stentz. "Optimal and efficient path planning for partially-known environments". In: *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*. 1994, 3310–3317 vol.4. DOI: 10.1109/ROBOT.1994.351061.

[14] Wikipedia. *Bug algorithm*. URL: https://en.wikipedia.org/wiki/Bug_algorithm.

List of Figures

Listings