Generation of 2D polar sonar images and 3D underwater structures reconstruction

Estelle ARRICAU ESTELLE.ARRICAU@ENSTA-BRETAGNE.ORG
student of ENSTA Bretagne

Abstract

This paper presents a method to represent sonar data from a FLS (Forward Looking Sonar) in two different ways. The first one consists in generating 2D polar coordinates sonar images. The second one consists in generating a 3D point-cloud of the underwater structures detected by the FLS. The purpose of this article is to provide relevant sonar information to a SLAM algorithm to correct the AUV's 3D position in its environment. A point matcher algorithm such as ICP (Iterative Closest Point) can be used to associate the 3D point-cloud generated to the underwater structures' 3D model which localisation in the environment is known.

1. INTRODUCTION

Seafloor investigation is one of the major tasks of AUVs (Autonomous Underwater Vehicles). Indeed, as survey boats have already covered the majority of the low-depth oceans' floor, AUVs are now used to explore inaccessible areas such as deeper seafloors, narrow underwater breaches, or the interior of wrecks.

To do so, many sensors can be used depending on the specifications of the mission. Almost all of them use an INS (Inertial Navigation Unit) for proprioceptive motions, a pressure sensor to estimate the immersion depth, a 2D or 3D sonar or camera to get observations of the underwater environment, and a GNSS (Global Navigation Satellite System) receiver to adjust the estimated position of the AUV when it surfaces.

With such sensors, a mission in an unexplored environment can easily lead to a SLAM (Simultaneous Localisation And Mapping) application. But as the majority of oceans have been covered, a partial or approximate map of the environment can ease the localisation process. The recognition of specific landmarks or structures is thus a major asset to improve location accuracy.

In this context, the 3D projection of 2D underwater structures observed with a sonar or a camera compared with a DTM (Digital Terrain Model) or a 3D model can be very useful. If the structures match enough, the AUV's position is readjusted.

2. RELATED WORK

Regardless of the visual sensor used, the first step is to represent the data observed. Here, a 2D FLS (Forward-Looking Sonar) or FSS (Forward-Scan Sonar) is used. Thus, 2D images can be generated from the sonar's raw data in cartesian or polar coordinates (2).

Then, different image processing methods can be applied to detect structures of interest (particular geometry of artifical underwater objects or naturally-textured seafloor structures)(4). Optical flow methods can also track the objects of interest through time.

Now that objects are clearly identified on 2D images, they can be projected in a 3D frame thanks to the AUV's forward motion which provides information about how the sonified object evolves over time in the image. The idea is to track the amount of shadow over time to estimate the elevation of the object. The object can then be represented in three dimensions(3)(1).

3D objects can be compared to 3D existing models or put into a NN (Neural Network)(5) to be identified. This identification process enables to locate the AUV accurately or to classify the mapped environment.

This information can thus be integrated into a full SLAM application for AUV equipped using inertial and acoustic data (6).

3. IMPLEMENTED ALGORITHM

3.1. Generation of 2D polar coordinates sonar images

The FLS emits multiple acoustic beams in a fan shape in its forward direction. In the spherical coordinates, the beams span the field of view as shown in Figure1.

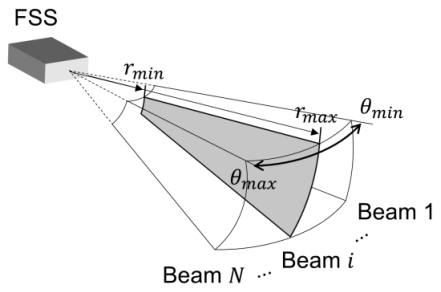


Figure 1: Acoustic beam configuration of a FLS

Once a beam is emitted, it is reflected by both the seafloor and the surface of the object. The FLS measures the reflection intensities over time : first from the seabed region between the sonar and the object, then from the object (see Figure 2).

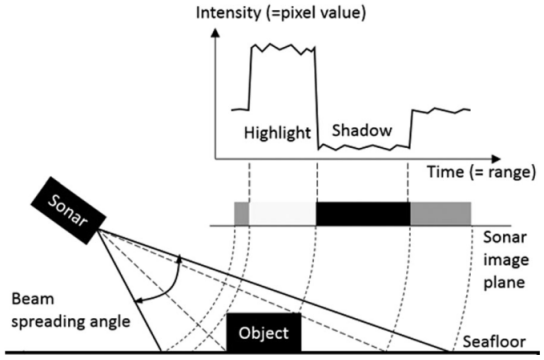


Figure 2: Acoustic beam reflections over time

In general, the reflection intensity from the surface of the object is stronger than that of other reflections. Therefore, the object generates a highlight region in the sonar image. After strong reflections, no reflections are measured because the emitted acoustic beam is blocked by the object. This creates the shadow region in the sonar image. The last measurements correspond to the reflection from the seafloor beyond the object.

The time-of-flight values for each reflection can be converted into range values by multiplying with the speed of sound. Moreover, the intensities of each reflection can be converted into grayscale pixel values. If the number of sampled reflections is assumed to be M and the number of acoustic beams is assumed to be N, an $M \times N$ sonar image can be generated.

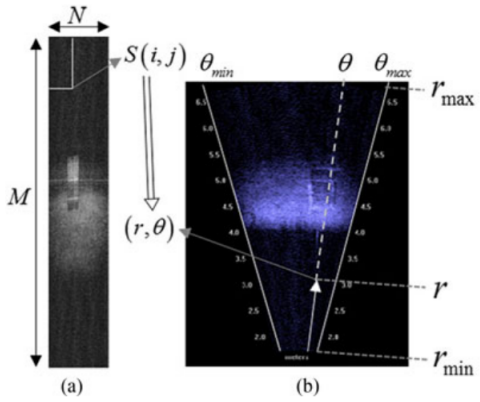


Figure 3: Cartesian and polar representation of a FLS image

Figure 3(a) shows the sonar image $S(i,j)$ whose dimensions are $M \times N$ for $1 \leq i \leq M$ and $1 \leq j \leq N$. To explicitly show the acoustic beam configuration, the sonar image is expressed in a polar-coordinate form, rather than as a rectangular 2-D array, as shown in Figure 3(b).

According to sonar geometry, a polar coordinate (r,θ) can be assigned to each pixel index (i,j). As shown in Fig. 3(b), the field-of-view of the sonar image can be defined as θ_{min} and θ_{max} . In addition, the minimum and maximum ranges of the sonar image can be defined as r_{min} and r_{max}, respectively. In this situation, a pixel index (i,j) can be converted into (r,θ) using the following equations:

$$r = r_{min} + \epsilon_r(i - 0.5) \quad (1)$$

$$\theta = \theta_{min} + \epsilon_\theta(j - 0.5) \quad (2)$$

where ϵ_r and ϵ_θ are the radial and angular resolutions, respectively, and are defined as:

$$\epsilon_r = \frac{r_{max} - r_{min}}{M} \tag{3}$$

$$\epsilon_\theta = \frac{\theta_{max} - \theta_{min}}{N} \tag{4}$$

To create the sonar image in the polar form, a new grid of coordinates (x, y) with $-r_{max} \leq x \leq r_{max}$ and $0 \leq y \leq r_{max}$. Then, the corresponding polar coordinates are computed using the following equations:

$$r = \sqrt{x^2 + y^2} \tag{5}$$

$$\theta = \arctan\left(\frac{y}{x}\right) \tag{6}$$

To make sure that all the (r, θ) tuples produced are within the sonar fan-shaped field of view, only the tuples verifying $\theta_{min} \leq \theta \leq \theta_{max}$ and $r_{min} \leq r \leq r_{max}$ are kept. Finally, the image indexes are interpolated to the nearest (r, θ) tuple indexes to put the intensity values at the right place.

3.2. Generation of 3D point-cloud from 3D line scanning

3.2.1. Preliminary concepts

Loss of elevation angle

The elevation angle ϕ cannot be obtained from the pixel index (i, j) because the sonar image is a mapping of 3D spherical coordinates to 2D polar coordinates, which are defined by $(r,)$.

In Fig. 4, there are three positions that are located at the same range but different elevation angles. Based on sonar geometry, the three points are mapped to the same pixel in the sonar image. Therefore, the elevation angle ϕ is lost which means the height of the object is unknown in Cartesian coordinates.

Due to this loss of the elevation angle ϕ and the 3D projection process, real curved objects are turned into flat ones in the sonar images, and conversely, as shown in Fig. 5.

Effective region

The acoustic beams of a FLS have a finite vertical beam spreading angle s, as shown in Fig. 6. If the altitude h and tilt angle t of the FLS are also given,

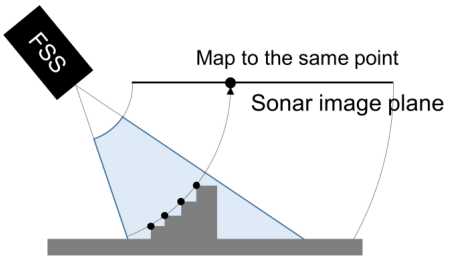


Figure 4: Loss of the elevation angle with a FLS

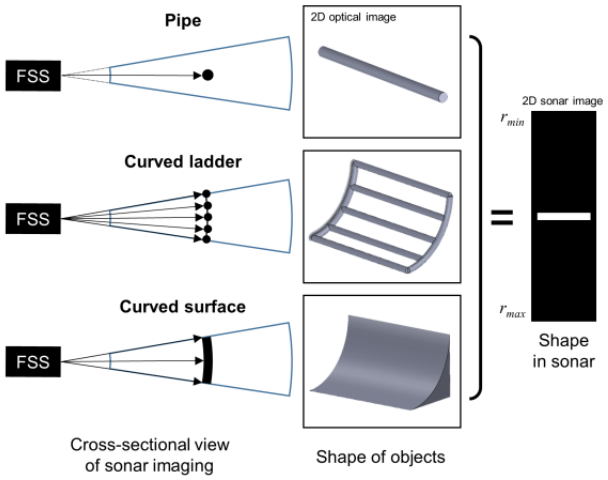


Figure 5: Shape ambiguity of objects observed with a 2D sonar due to 3D reprojection

the region in which the acoustic beam is actually reflected is bounded by the ranges r_{emin} and r_{emax}, which are calculated as:

$$r_{emin} = \frac{h}{\sin(t + 0.5s)} \tag{7}$$

$$r_{emax} = \frac{h}{\sin(t - 0.5s)} \tag{8}$$

In this study, the region between these limits is called the **effective region**. In general, the effective region has brighter pixels than the shadow region because of the acoustic beam reflection from the seafloor.

Highlight extension

Assume that an object is located on the seafloor

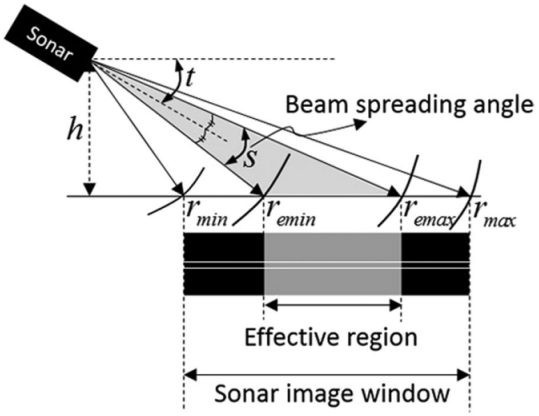


Figure 6: Effective region of an FLS

and that the FLS device approaches the object.

In the situation shown by Fig. 7(b) the highlight region of the object extends beyond the effective region. The surface of the object at which the acoustic reflection occurs is closer than r_{emin} ; therefore, the highlight of the object is mapped to a closer range than the boundary between the effective and ineffective regions. In this study, this situation is called **highlight extension**.

When the FLS device moves further, the amount of the highlight extension increases, and eventually, the front part of the object meets one of the edges of the spreading acoustic beam of the FLS device, as shown in Fig. 7(c). In this study, this location is called the **critical point**.

As long as the ALMS device moves in the forward direction, the FLS must meet the critical point, and highlight extension will be constant after the critical point. The amount of highlight extension can be converted into the height of the object.

3.2.2. Detection of highlight extension

As mentioned above, the ineffective region has low pixel values and appears as a shadow region. On the other hand, the extended highlight has high pixel values. Because of their high contrast, the highlight extension causes extreme pixel value increases in the ineffective region. Therefore, highlight extension can be readily detected by applying difference filters, which yield a peak at the point at which extreme

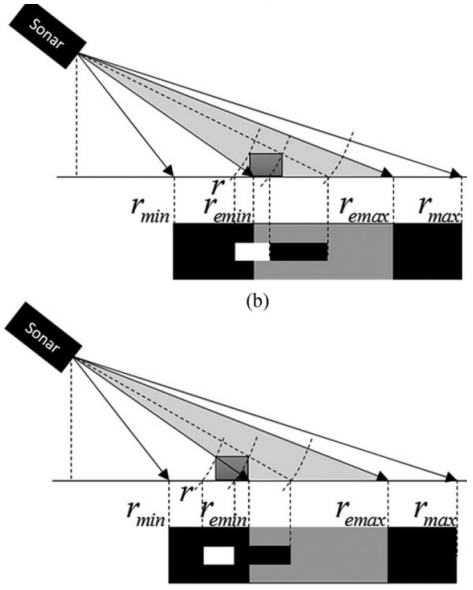


Figure 7: Highlight extension of an object observed by a FLS

pixel value variation occurs.

When the number of row indices of the sonar image is M, r_{min} and r_{max} correspond to the row pixel indices 1 and M , respectively. Therefore, the row indices corresponding to r_{emin} and r_{emax}, which are denoted as M_{emin} and M_{emax}, respectively, can be calculated as:

$$M_{emin} = M\frac{r_{emin} - r_{min}}{r_{max} - r_{min}} \tag{9}$$

$$M_{emax} = M\frac{r_{emax} - r_{min}}{r_{max} - r_{min}} \tag{10}$$

Based on M_{emin} and M_{emax} , the sonar image $S(i, j)$ can be divided into three subparts, i.e., $S_1(i, j)$, $S_2(i, j)$, and $S_3(i, j)$ whose row index ranges satisfy $1 \leq iM_{emin}$, $M_{emin} \leq i \leq M_{emax}$, and $M_{emax}i \leq M$, respectively, as shown in Fig. 8.

Because the highlight extension occurs in $S_1(i, j)$, the difference filter is applied only to $S_1(i, j)$ rather than the entire sonar image. As shown in Fig. 9, the difference filter is defined using a 3 × N kernel $D(i, j)$ whose values are set to 1, 0, and 1 for i = 1, i = 0, and i = 1, respectively, for all j. Using the

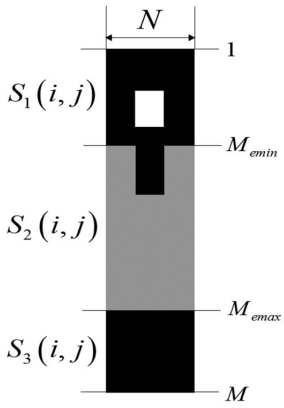


Figure 8: Sonar image separation

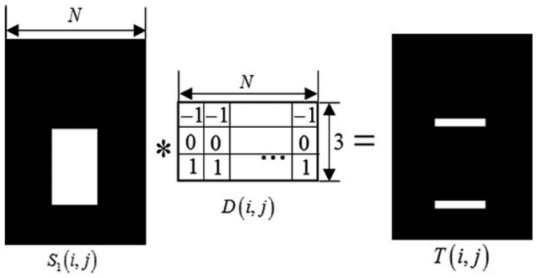


Figure 9: Difference filter for detecting the extended highlight

correlation operation between $S_1(i,j)$ and $D(i,j)$, the filtered image, $T(i,j)$, can be derived as:

$$T(i,j) = \sum_{k=-1}^{1} S_1(i+k,j)D(k,j) \qquad (11)$$

Ideally, one of the columns of $T(i,j)$ that includes the extended highlight has one positive peak, which corresponds to the rising edge of the extended highlight of $S_1(i,j)$. A negative peak can also appear, and it corresponds to the falling edge of the extended highlight. Therefore, the index of the positive peak can be selected as the front end of the highlight extension.

In practical cases, the filtered image $T(i,j)$ may have many peaks that are not caused by the extended highlight but by undesired noise, as shown in Fig. 10(b). Applying a smoothing filter like a Gaussian blur is one solution for mitigating noise effects.

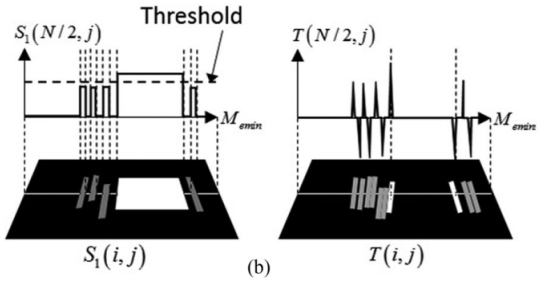


Figure 10: Extended highlight front-end threshold

3.2.3. Elevation angle calculation

The highlight detection process is executed for all N columns of $T(i,j)$. As a result, we can obtain N indices for each front end of the highlight extension for each column. We denote the array of indices as $m(j)$ for $1 \le j \le N$. If highlight extension is detected in the jth column, $m(j)$ is set to the index of the front end of the highlight. If no peak is detected in the jth column, $m(j)$ is set to M_{emin} , which implies that the jth column does not include an object higher than the seafloor.

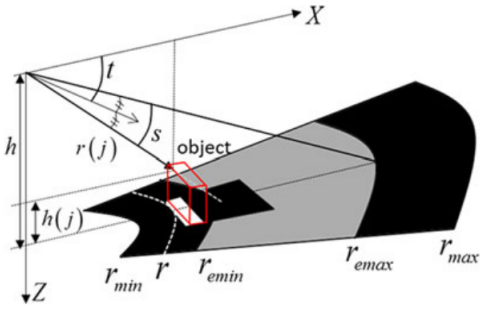


Figure 11: Geometry used for height calculation

The range $r(j)$ corresponding to $m(j)$ can be calculated as:

$$r(j) = r_{min} + (r_{max} - r_{min})\frac{m(j)}{M} \qquad (12)$$

In the spherical coordinate system, the elevation angle $\phi(j)$ can be calculated as:

$$\phi(j) = \phi = t + 0.5s \tag{13}$$

The azimuth angle of the pixels of $S(m(j), j)$ can be calculated as:

$$\theta(j) = \theta_{min} + \epsilon_\theta(j - 0.5) \tag{14}$$

As a result, the 3D coordinate corresponding to the pixel $S(m(j), j)$ can be expressed as $(r(j), (j), (j))$ in spherical coordinates. This coordinate can also be converted into Cartesian coordinates $(x(j), y(j), z(j))$ using coordinate conversion equations:

$$x(j) = r(j)\cos(\phi(j))\cos(\theta(j)) \tag{15}$$

$$y(j) = r(j)\cos(\phi(j))\sin(\theta(j)) \tag{16}$$

$$z(j) = r(j)\sin(\phi) \tag{17}$$

3.2.4. 3D POINT-CLOUD GENERATION

One pixel in each column of a sonar image is converted to 3D coordinates by the proposed method. Therefore, the number of 3D coordinates is N and the coordinates constitute line scan data of the seafloor. If a FLS-equipped AUV successively obtains the line scan data from each sonar image, the 3D point cloud of the seafloor can be generated by accumulating line scan data in the same coordinate system, as shown in Fig. 12.

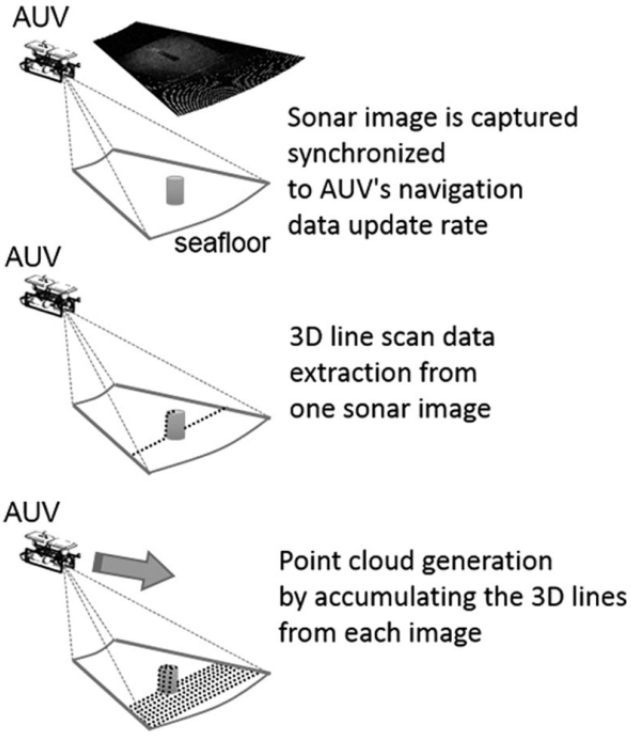


Figure 12: Three-dimensional point cloud of seafloor generation using AUV

4. EXPERIMENTS AND RESULTS

In the context of Guerlédan's project, measurement campaigns must be carried to collect data for an offline SLAM application. Among other data, sonar images of the seafloor are captured by an immersed test bench at a given depth.

4.1. Material

The forward-looking sonar used during this experiment is the Oculus M1200d sonar. Its characteristics are the following:

- Operating frequencies: 1.2MHz / 2MHz
- Maximal range: 40m / 10m
- Minimal range: 0.1m
- Range resolution: 2.5mm
- Horizontal aperture: 130°/80°
- Vertical aperture (spreading angle s): 20° / 12°
- Maximal number of beams: 512

The sonar was attached to the test bench with a tilt angle of 15°, which in turn was attached to the *Panopée* boat by means of a section as shown in Fig. 13.

4.2. Description of the experiment

For this experiment, the boat explores the commercial port where an old windmill tripod is immersed. Only two large metal pylons exceed water. The boat went back and forth in a straight line over the tripod while capturing sonar images as shown in Fig. 14.

4.3. Results

4.3.1. 2D POLAR COORDINATES SONAR IMAGES

The original sonar image in the Cartesian form can be seen in Fig. 16(a). It has been converted into its polar form as shown in 15.

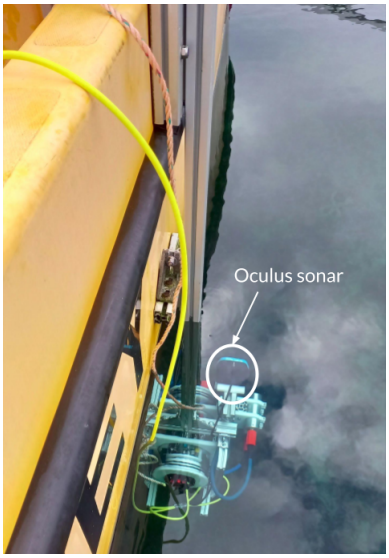


Figure 13: Oculus sonar fixed to the boat

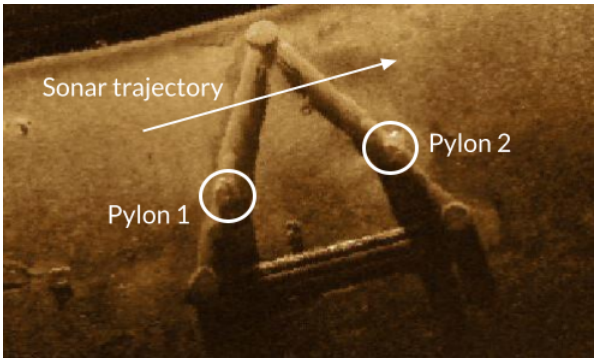


Figure 14: tripod with its pylones observed with a Klein lateral sonar

4.3.2. 3D POINT-CLOUDS OF UNDERWATER STRUCTURES

The object detection process is shown in Fig.16 step by step.

After computation of the object's 3D coordinates, the 3D point cloud obtained is shown in Fig. 17.

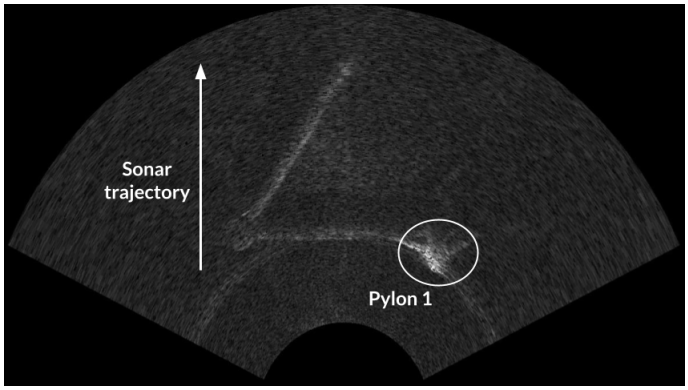


Figure 15: 2D polar sonar image of the tripod

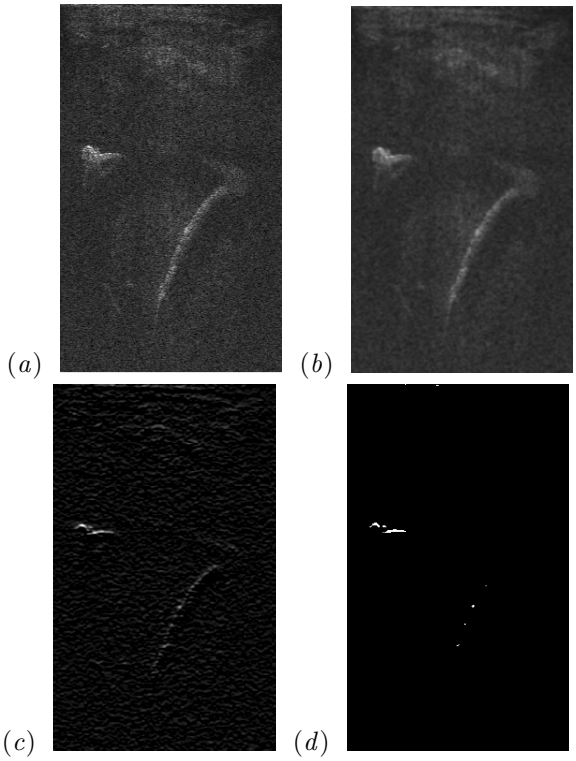


Figure 16: Image processing for object detection.
(a) Original image.
(b) Gaussian-blurred image.
(c) Vertical-derived image.
(d) Thresholded image.

5. DISCUSSION

The results are mitigated due to several aspects:

- The low quality of the FLS, a more low-frequency one would be better to have a sharper contrast between the seafloor and the objects observed.
- The turbid water and the mainly unstructured observed objects prevent from a clear detection of the front-end highlight extension of the object.

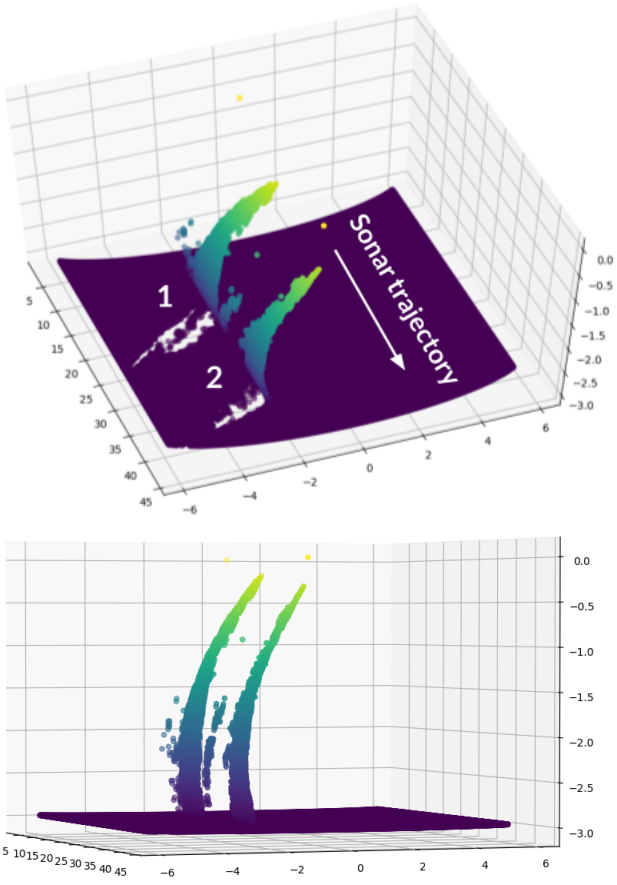


Figure 17: 3D reconstruction of the pylones of the tripod

- The poor image processing applied to the images. Maybe using the polar coordinates representation would lead to less curved objects. An optical flux to track the objects would be more robust to noise.
- The 3D projection method used distorts the objects. They appear curved because of the shape ambiguity with a 2D FLS described in Fig. 5.

6. CONCLUSION

This article gives a rather simple method to retrieve the lost elevation of 2D-sonar-observed underwater objects. It may not appear enough for very specific applications but it gives a rough idea of the underwater relief. Integrated into a SLAM application, the approximate estimated position by a Kalman filter associated with this sonar observation can be enough to recognized a particular 3D structure and match it with its located 3D model or the DTM of the local area, and thus, reposition the AUV.

References

[1] Hyeonwoo Cho, Byeongjin Kim, and Son-Cheol Yu. AUV-Based Underwater 3-D Point Cloud Generation Using Acoustic Lens-Based Multibeam Sonar. *IEEE Journal of Oceanic Engineering*, 43(4):856–872, October 2018.

[2] Jian Zhang, Ferdous Sohel, Hongyu Bian, Mohammed Bennamoun, and Senjian An. Forward-looking sonar image registration using polar transform. In *OCEANS 2016 MTS/IEEE Monterey*, pages 1–6, Monterey, CA, USA, September 2016. IEEE.

[3] Byeongjin Kim, Jason Kim, Meungsuk Lee, Minsung Sung, and Son-Cheol Yu. Active Planning of AUVs for 3D Reconstruction of Underwater Object using Imaging Sonar. In *2018 IEEE/OES Autonomous Underwater Vehicle Workshop (AUV)*, pages 1–6, November 2018. ISSN: 2377-6536.

[4] Filip Mandić, Ivor Rendulić, Nikola Mišković, and ula Na. Underwater Object Tracking Using Sonar and USBL Measurements. *Journal of Sensors*, 2016:1–10, 2016.

[5] Minsung Sung, Jason Kim, Hyeonwoo Cho, Meungsuk Lee, and Son-Cheol Yu. Underwater-Sonar-Image-Based 3D Point Cloud Reconstruction for High Data Utilization and Object Classification Using a Neural Network. *Electronics*, 9(11):1763, October 2020.

[6] Albert Palomer Vila. 3D Underwater SLAM Using Sonar and Laser Sensors. page 120.

Determination of a robust socially adaptative path planning frameworks using IRL

Antonin BETAILLE ANTONIN.BETAILLE@ENSTA-BRETAGNE.ORG
student at ENSTA Bretagne

Abstract

To collaborate with people, robots must be able to evolve in the same environment as them. Navigation must not only match the deployment space but also the comings and goings of passers-by and the behavior of crowds. In order for the robot's movement to be able to withstand these constraints, it is therefore not enough to choose the shortest path to reach a set position. In fact, the optimization of the path rather corresponds to a logic of adaptation of the behavior, having for goal to respect the comfort of displacement of the passers-by. For this, I rely on a framework for socially adaptive path planning in dynamic environments, by generating human-like path trajectory. This framework rely on three modules : a feature extraction module, inverse reinforcement learning (IRL) module, and a path planning module. Thanks to a RGB-depth sensor, the first module extract the data linked to the velocity of surrounding obstacles. The second module compute a cost function that is based on social variables. This is achieved through the inverse reinforcement learning that learns an expert's behavior which had been converted in a set of demonstration trajectories accordingly to the received feature. The third one integrates a three-layer architecture, where a shortest-path objective and a global path are combined to plan an hybrid path in a global map supposedly known. In principle, this path is planned on a over on the short term accordingly to the features extracted from the RGB-D sensor, the cost function inferred by the IRL module, and eventually a low-level system that manages the avoidance of immediate obstacles. This approach was primary evaluated by deploying a real robotic wheelchair platform in various scenarios, and comparing the robot trajectories to human trajectories. Nevertheless the actual work is forecasting its application on a telepresence robot.

Keywords Obstacle avoidance · RGB-D ·optical flow · Learning from demonstration · Inverse reinforcement learning · multi-hypothesis tracking

1. INTRODUCTION

Remote work and isolation issues have shed the light on the the growing interest in telepresence robots. In this context, the capacity to navigate in a crowded and dynamic environment have been required for autonomous robots too. However the point of our approach is not only to reach a goal while avoiding obstacle but doing it in a socially adaptive manner. To impersonate the presence of somebody, the robot must adopt a socially adaptive navigation behaviour. Consequently, pedestrian should be taken into account as social variables for navigation behaviours rather than as common obstacles. (7)

To adress this issue, the framework does not apply standard search techniques using conventional cost functions (e.g. shortest path) to find a good path. The goal is to generate a socially adaptive cost function from human navigation demonstrations that will allow the robot to have a behavior adhering to social rules for human comforts. (8)

I recall that the proposed framework consists of three modules: the feature extraction module, the IRL module, and the planning module. The feature extraction module extracts the local state features such as densities and velocities of pedestrians observed via RGB-D sensor during a navigation. The IRL module uses demonstrations from a human operator to learn the socially adaptive navigation behaviors as a function of these observed state features. The behavior is represented as weights (i.e. parameters) for the cost function, which are combined with the features extracted from the feature extraction module to compute a socially adaptive cost function that defines costs at local grid cells. In the planning module, a global path is planned using a global

map given a priori and a local path is planned using the local grid cells associated with the costs. It then chooses which of these two plans to use, depending on the situation.

The kind of method is particularly appealing for domains where it is difficult to specify a cost function analytically. And that is the case with the scenarios selected to evaluate the conduct of the telepresence robot.

Then in a second phase I will present the implementation of gym-gazebo2 on a specific environment, the control of a simple car. Multiples scenarios were chosen as the reaching point scenario or the path of a maze. The models and algorithms combined in this application, will be exposed and the results analysed. The different RL method used will be compared for this specific case.

2. PROBLEM STATEMENT

The problem that has to be tackled through the creation of the framework is defined as follows : Given a global path plan from the current position of the robot to the global goal, determine the socially adaptive local path plan from the current position of the robot to the sub-goal.

Moreover, the navigation is evaluated via a similarity study between the path that is planned by a human and the one planned by the robot supposed to be socially adaptive.

And as a forecast, the cost function should generalize well to different environments with similar pedestrian patterns thanks to the IRL framework. This means the robustness of the algorithm should allow a telepresence robot to navigate in crowed open-spaces, conventions and even in schools.

2.1. LIMITATIONS OF OTHER FRAMEWORKS

The problem of navigation in human environment is not something new. That is why there was plenty of works to evaluate in order to constrast the qualities of my framework and limitations from others

The limited frameworks have been divided into four sub-groups:

Those that involve training a motion predictor (2). Their weakness is the lack of generalization of their motion predictor. Because people observed in the training data and those observed in the test data do not behave exactly the same, the datasets are non-identically and independently distributed (non-iid), whereas the iid is standard assumption in supervised learning approaches.

Those that focus on path-planning while assuming motions of dynamic obstacles are given (10). To achieve such assumption, there is no other way than relying on simulation only. In this case, a noise-free pedestrian velocity is given. In real-life scenario, it might not work as well as in the conclusions of the authors.

Those that focus on human-aware navigation using social variables (8) with hand-designed cost function. To generate an adequate path planner (7), estimating pedestrian models (11) might be to long and too complex compared to learning social models in the form of a cost function

And finally the closest work are (5; 6). While in (5) their work is limited to a simulated settings, in (6) they focus on a local path planner integrated with a feature extraction module that extracts pedestrian movement, as well as the full planning architecture that provides interactions with a global planner.

2.2. CHALLENGES

To begin with, there is no planning module if the local feature extraction is not performed. There is a particular attention on the extraction densities and velocities of crowds around the robot. In (6) , rather than tracking individuals in the crowd, they extract summary features using a RGB-D sensor. This has the advantage of working online and without any prior training.

To continues, the implementation of the Markov decision process in order to learn the adaptive cost function for local path planner with IRL technique is necessary. There is a specific design in the representation of the features, and the structure of the functions in particular local areas. This is detailed in Sect.4

Then, in view of the choice between local and global path planner, the re-plan operations and the emergency stops, developing the planning architecture is a major challenge. This is from this issue that stems the three-layer planning architecture created in (5) in which all layers run in parallel to resolve the issues mentioned.

Finally, the essence of any new approach is to be evaluated. To asses the benefits of the framework, there is an unavoidable comparison to submit be-

tween the trajectories of an human driver and those of a robot under the same or similar conditions

2.3. ROBOT PLATFORM

The robot model selected for this paper is Beam robot from Awabot on wich has been embedded a Kinect. This sensors combines RGB and depth information, and represents them as a point cloud in 3D coordinates (x, y, z), in which horizontal, vertical, and depth locations are respectively represented. It allows a resolution of 640 × 480 and has a horizontal field of view of 57○ and approximately 5m in depth.

Moreover, to ease the calculation of the velocity to be executed by the robot to reach each waypoint of a local path, the framework is implemented as packages in the middleware ROS.

This beam robot can easily be controlled remotely with a ROS driver for a generic Linux joystick. In this way it is possible to gather the data required to make the demonstration dataset. However, as it is explained in (6) the use of a robotic wheelchair may result in more abudant data because it could be used on a daily basis by mobility-impaired individuals.

Figure 1: Beam robot from awabot

2.4. OUTLINE OF THE APPROACH

Our approach is a combination of two off-line stages and one on-line execution stage. These three stages are summarized in Fig. 2.

2.4.1. DATA GATHERING STAGE

The way the data is acquired and used is the key of the IRL modulde. First, a human expert has to navigate in the environment in different situations. Through the demonstration, the state features of the trajectories and the decisions are collected to form a demonstration dataset.

2.4.2. LEARNING STAGE

The dataset is then transferred to the IRL module to learn the behavior of the expert when navigating. This generates the computation of weights for the cost function.

2.4.3. EXECUTION STAGE (BY CYCLES)

Finally, the weights are passed to the planning module while the feature extraction module extracts features at the local area using the RGB-D sensor. This result in the cost computation. Eventually, the planning module outputs a path plan such that the cost is minimized and the robot can execute this plan in the environment as long as it is not manually stopped or it has not reached its goal.

This approach associates a numerical cost to each grid cell from the local area. However, unlike the planners that use occupancy grids to calculate costs, the planner of (6) uses environmental features that all contribute to the cost of a cell.

3. FEATURE EXTRACTION

to characterize the dynamic aspects of a scene, the use of a RGB-D sensor is useful to represent it as a 3D point cloud. Four distinct features arise from it : crowd density, speed, velocity (i.e. speed and direction of a crowd), and distance to the goal. The density feature estimates where the crowds currently are by counting the number of points in each cell. The speed and direction features estimate where the objects will be in the future, by estimating the velocity of each point in a point cloud. The velocity of each cell is calculated by averaging the velocities of points in the cell. The distance to the goal is calculated based on the distance between each cell and the global goal.

3.1. RGB-D OPTICAL FLOW

it determines how quickly a robot can respond to changes in the environment. In order to calculate velocities using 3D point clouds, (6) needed an algorithm that extends an RGB optical flow algorithm to the RGB-D setting.

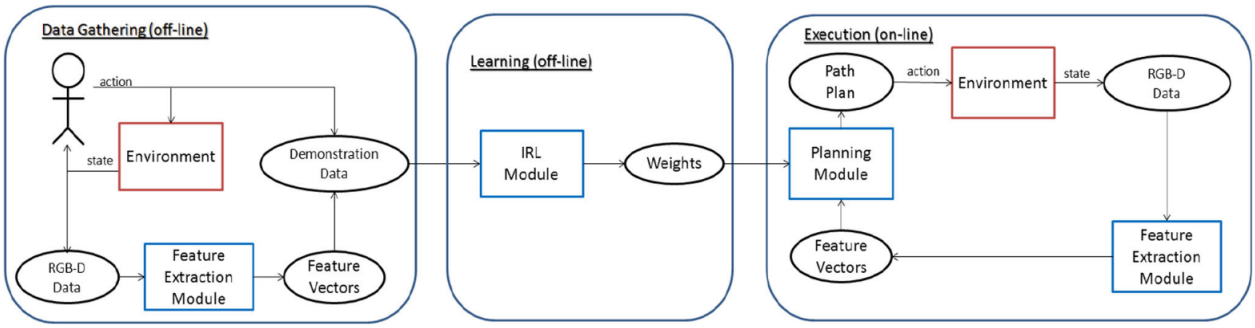


Figure 2: Different stages of our approach and module interaction during the execution stage. Square blue boxes indicate the three modules. The human demonstrator is denoted with the stick figure. Arrows indicate the information flow. (Color figure online)

Hence they employ an RGB optical flow algorithm known as Farneback optical flow, which entails a fast flow estimation algorithm.

The principle is schematized as follows :

The sensors capture a series a image frames that are stacked together. As a result we obtain a 3D image volume in which there are two spatial dimensions, and a third describing the temporal dimension. The translation of a pixel through the frames can be considered as a movement. Consequently the sequences of images would give a vector with a particular orientation in this 3D image volume, describing the velocity direction. All the details of the equations involving the velocity and using the orientation tensor and the cost function are detailed in (6)

Besides, since the velocity information is extracted on a per-point basis, it is necessary to average the velocities of points in each cell from the navigation grid to calculate the velocity of that cell.

In addition the movement off the robot has to be taken into account. It is imperative to subtract the sensor's (estimated) velocity vector from the extracted velocities to keep a non biased information.

3.2. FEATURE VECTOR

To decrease the difficulty of the representation and to offer robust results it is common to represent the features in binary features vectors for IRL (1; 5)].

In each grid cell, there is an associated 12 dimensional binary feature vector, that describes pedestrian velocities, which consists of pedestrian speed and direction, density, and distance to the goal in that cell. Speed, density, and distance features are represented using high, medium and low bins depending on some thresholds (4*3). Pedestrian direction feature describes whether there are points (i.e. pedestrians) moving into or out of a cell.

To help the comprehension of the cell's risk management, the Figure 3 can be used as a reference.

Risk level of the cell	Cell with points flowing in from direct neighboring cells	Points flowing in from second-degree neighboring cells	Points in the cell moving out of it
HIGH			
MEDIUM			
LOW			

Figure 3: Table defining the risk of the cells

4. INVERSE REINFORCEMENT LEARNING

To learn a socially adaptive cost function, we consider Inverse Reinforcement Learning (IRL), a sub-field of Reinforcement learning (RL). The IRL problem, is to infer the cost function from demonstrations of an expert (4), assuming that the expert is unconsciously minimizing this (unknown) cost function for planning its action sequence.

The inferred cost function is then used by an AI agent, such as a robot, to plan in new situations in such a way as to achieve performance similar to that of the expert.

4.1. MARKOV DECISION PROCESS

A Markov Decision Process (MDP) is defined as a tuple $<$ S, A, T, C, γ $>$, where S represents the finite set of states, A represents the finite set of actions, T (s, a, s) represents the state-to-state transition function, C(s, a) represents the cost received when action a is taken in state s, and $\gamma \in [0, 1)$ represents the discount factor

The objective is usually to find an optimal plan π : S $\rightarrow$ A that minimizes the expected sum of future costs, or the value function.

$$Q^{\pi}\ (\mathrm{s}_0, a_0) = E\left[C(s_0, a_0) + \sum_{t=1}^{\infty} \gamma^t C(s_t, \pi(\mathrm{s}_t))\pi\right] \quad (1)$$

4.2. MDP FORMULATION FOR NAVIGATION

4.2.1. STATE AND ACTION SETS

The state set, S, is defined as the local grid cells in front of the robot. Each cell defines a state, and a cell is reachable if the cell is one cell away from the current cell. And, the number of local grid cells gives the cardinality of S.

The action set, A, for our navigation system includes a discrete action for moving into each cell that is adjacent to the current cell (directly in front, diagonally to the left, diagonally to the right), according to the state space defined above.

4.2.2. COST FUNCTION

The expected cumulative cost of executing action a in state s and following a policy afterwards is given by the function Q^{π}(s, a), as in a standard MDP. The goal in the MAP-BIRL (maximum-a-posteriori Bayesian inverse reinforcement learning) framework is to determine the cost function by computing the weight vector w that maximizes the probability of the given demonstration data X (ensemble of action sequences). (**?**)

There have been defined a feature function ϕ(s, a) that tells what is the feature of the state that the robot will transition into by using action a in state s. The cost function for an action a in state s cell is calculated using a linear combination of associated features in the next state:

$$\mathrm{C(s,a)} = \sum_{i=1}^{d} \omega_i \phi_i(s, a) \quad (2)$$

$$= \omega.\phi(\mathrm{a,s}) \quad (3)$$

4.2.3. REGULARIZATION

In many case in RL, the features influence on the actions is heterogeneous. For instance, in (6) experiments, they observed that the speed feature surprisingly had almost no effect in navigation behaviour, and only density, flux direction, and distance features mattered. This means that when trying to imitate the behaviour of the expert, it may not be necessary to utilize all the features of the environment. Keeping them could lead to over-fitting the cost function, especially when we do not have a lot of training data. Besides, avoiding some features reduces the number of estimations to be performed. All in all, it is recommended to employ a regularization technique within our IRL method, and this as been applied in the IRL module.

	Den	Speed	Dir	DistGoal
High	1.0	0.0	1.7	0.3
Medium	-0.5	0.0	1.2	-0.1
Low	-1.2	0.0	-2.3	-0.4

5. PLANNING

5.1. PLANNING ARCHITECTURE

The architecture consists in fact of three layers: the global and local path planners, and a low-level collision detector. This is similar to the idea of a three-layer architecture suggested in (**?**), where you have several layers running in parallel, each responsible for different tasks simultaneously.

As for many basic planning methods, the first step is to give the robot a global goal which is specified by the robot user. Then, a path from the current position to the sub-goal is obtained with the local path planner. The local path planned is represented as a set of points, which we call waypoints, that needs to be followed in order for the robot to get to the sub-goal. In this configuration the robot tries to reach the closest waypoint from its current position (which can be on the global path if there is no obstacle). Meanwhile, the low level collision detector runs in parallel and can stop the robot if an obstacle is too close to the robot (e.g. $<$ 1m).

5.2. PLANNING ARCHITECTURE

The local path planner, Algorithm 1, is an event-driven algorithm that returns the next destination

to be reached. It proceeds as follows.

Algorithm 2 : Local Path Planning Algorithm

```
Input : FeatureVectors, WeightVector, GlobalPath, CurrentWayPt
while RobotRunning do
  //Waiting for feature vectors
  WaitOn(FeatureVectors)

 //Calculate cost at each cell using feature vector, weight vector, and Eqn.(3)
 GetCostAtEachCell(FeatureVectors, WeightVector)

 if SafeZone Reached then
   if Obstacle Not Detected then
    //Reach for the global goal
    return GlobalPath
   else
     //Plan a local path to the sub-goal
     LocalPath=DjkistraGraphSearch()
     NextDestination=ExtractWaypoint(LocalPath)
     return NextDestination
    end if
 else
  return CurrentWayPt
 end if
end while
```

From this algorithm 3 things need to be specified :
- The safe zone is defined as a small (e.g. 0.25 m) radius around the current waypoint.

- The second test (for obstacle detection) is true if one of the cells within the local action radius (e.g. 4 m) has high density.

- The local planner optimizes a local path using Djkistra's algorithm over the learned feature-based IRL cost function.

6. EXPERIMENTS

To assess the socially adaptive navigation framework, 2 types of trajectories are compared with the framework's one. The reference is the one performed by human driver. The competitive one is obtained by he Dynamic Window Approach (DWA) (3), a shortest-path type planning method that uses laser data, which has mostly be proven successful in static environments.

To have relevant results the scenario-based experiments are repeated many times by the human driver and the robot in the same initial conditions. Three metrics are considered to measure the social adaptivity : closest distance to the pedestrian, avoidance distance to the pedestrian, and average time to reach the goal.

In the experiments, the avoidance distance is measured by calculating the distance from the center of the robot to the center of the pedestrian when the angular velocity of robot is increased to a particular threshold, in an attempt to avoid the pedestrian.

To give more insight on the choice of the metrics, it is important to highlight the fact that humans have a comfort space which they try to keep not overrun by unknown individuals during their navigation. That is why the closest distance to the pedestrian is a significant parameter to start avoiding pedestrian from a certain distance.

6.1. EXPERIMENTS IN CONTROLLED ENVIRONMENT

With the robot platform (6) performed empirical evaluations in scenario-based experiments. They collected log files in conditions where there were multiple pedestrians approaching or moving away from the wheelchair, from various directions. The behavior that stems from the observed trajectories shows that the framework is able to produce paths that are really close to those adopted by human operators.

Note that the conditions in which the data were collected are not identical to the conditions for the experiments below. This is intentional, to show robustness of the approach to a variety of conditions. The same training set is used for the three test scenarios listed below.

The first scenario considered a pedestrian walking towards the robot. In this case the trajectories generated by the DWA are not socially adaptative. The robot does not avoid the pedestrian early enough to clearly define its intention and then drives to closely to make the pedestrian feel more comfortable and safe. Whereas the trajectories from the framework are closer to what a human could do.

The second scenario considered a pedestrian walking horizontally to the robot. The behavior of the DWA shows other limitations such as stop and restart repetitions while trying to avoid the current position of the pedestrian. This issue has not been repeated with the navigation algorithm because the learned demonstrations have shown that a human being has

the reflex to avoid the future position of the pedestrian and not their current position.

The third one is considered as one of the essential abilities of socially adaptive path planners (5). In this scenario, we have multiple pedestrians approaching the robot from the front at a distance of 3–4 m, and a pedestrian that is moving parallel to robot to its left (and in the direction of the goal). This can be summarized as joining a flow direction while avoiding an opposing flow direction. The common behavior of the three methods is to start by avoiding the group. However, after that, the DWA approach tries to avoid the pedestrian as he approaches the goal, making an unnecessary turn right before reaching the goal. This is not the case with the IRL planner but it differs to human behavior because it reaches the goal faster. This is because once the crowd is avoided, the distance feature outcomes the density features in the cells closer to the goal.

6.2. EXPERIMENTS IN NON-CONTROLLED ENVIRONMENT

For this experiment the goal is to place the robot in an unknown environment, a crowded hallway, that could be observed in a convention or a school building with additional static obstacles. This time, the goal is not a position but a pathway from one end of the hallways to the other. The purpose of this experiment is to show that the method is sufficiently robust for non-controlled environments, and in particular that the cost function learned can be used in previously unseen settings.

To assess the behavior of the robot, the comparison is a bit different because the conditions of the experiment could not be repeated. This time, the operator is also supposed to intervene in the robot navigation when deemed necessary. In (6), they measured the human intervention percentage during the autonomous navigation. The same kind of comments on social adaptivity as before could be done on the tested approaches. However we can notice that the DWA tend to get stuck in the crowd and consequently the planner is mainly controlled by the operator. In the other hand, IRL only needs assistance when there is occlusion or the avoidance comes too early.

7. ISSUES OF THE FRAMEWORK

7.1. LIMITATIONS

First, the estimated RGB-D optical flow is inherently noisy and does not account for occlusions. This complicates the well-known correspondence problem (i.e. identifying which points from two different RGB scans correspond to the same physical item).

Another limitation is the fact that the IRL framework performs best when given precise feature measurements. As the cost function is linear in the features, if these features are not measured precisely, the cost estimates can be noisy, which can lead to poor planning.

Another limitation is linked to the lack of precision in measurements. If the features are badly estimated, the performances of the IRL frameworks suffers from a noisy cost estimates and by definition a lack of reactive behavior.

This can be entirely caused by the field of view of the Kinect, which is very limited compared to that of human vision, which makes horizontally approaching obstacles harder to detect. For information, humans have an almost 180-degree forward-facing horizontal field of view.

7.2. ADDITION OF OTHER SENSORS

During the study of (6), I could not ignore the fact that the presence of only one Kinect on the robot platform led to limited possibilities. In fact, the presence of pedestrian is never limited to the front of the path. When navigating in a crowed environment, pedestrian navigate to different speeds and might take the initiative to go ahead of you. In this case, even humans can be limited because they often need to slow down and turn their head before choosing their future action. With a back camera it could be possible to create thorough demonstration datasets with an extended field of view. Of course, human operators may not have been confronted to such point of view, and that may be a step ahead the common social behavior of the pedestrians. However, the framework would not be different and could show interesting results.

7.3. MULTI-HYPOTHESIS SOCIAL GROUPING FOR PEDESTRIAN MOVEMENT TRACKING

The problem of detecting, analyzing and tracking groups of people, particularly from mobile platforms,

is relevant for a number of scenarios including multi-party human-robot interaction and collaboration, efficient and socially compliant robot navigation among people, analysis of social group activities, and understanding of social situations.

In (9) they state the problem as an estimation problem of social relations between individuals from perceived track motion features using SVM classifiers and Bayesian smoothing. Since the spatial organization of groups is typically not random and remains largely stable over time, they also learn group-specific geometric relations between individuals.

Their extended multi-hypothesis tracking approach works in real time using 2D range data. With the collected data they recursively learn from the behavior of a group and from the behavior inside the group while handling more occlusion cases. They also shed the light on the better trade-off between false negative and false positive tracks that it deals with.

Thus it is interesting to compare the sensor of (9) a SICK LMS 291 laser range finder at around 0.80 m height and 0.5 deg angular resolution and the low cost Kinect of (9). The features extraction is for sure more accurate with such sensor. With roughly 40% fewer track identity switches and 28% fewer false negative tracks, the results suggest that tracking people in 2D range data can strongly benefit from estimates on social and geometrical relations, mostly due to their ability to explain lengthy occlusion events.

8. DISCUSSION

The framework implemented for socially adaptive path planning in dynamic environments is satisfying enough to mitigates its limitations. It has proven to be quite robust to complex and unknown scenarios. The main drawback is the quality of the sensor which is not accurate enough to deal with occlusion cases. This issue lay the emphasis on the lack of apprehension toward group behaviors and its individual components.

9. CONCLUSION

To conclude it is interesting to recall that the main framework that have been discussed is more of a prototype than a real robot which is supposed to be commercialized. Nevertheless, the experiments can be used as a proof of concept to show that IRL and RGB-D cameras can be used to infer a socially adaptive behavior to a robot. However, to be more efficient, this cannot be the only way of approaching it.

References

[1] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the Twenty-First International Conference on Machine Learning*, ICML '04, page 1, New York, NY, USA, 2004. Association for Computing Machinery.

[2] Amalia F. Foka and Panos E. Trahanias. Probabilistic autonomous robot navigation in dynamic environments with human motion prediction - international journal of social robotics, Jan 2010.

[3] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics Automation Magazine*, 4(1):23–33, 1997.

[4] Russell S g AY. Agorithms for inverse reinforcement learning. 2000.

[5] Peter Henry, Christian Vollmer, Brian Ferris, and Dieter Fox. Learning to navigate through crowded environments. In *2010 IEEE International Conference on Robotics and Automation*, pages 981–986, 2010.

[6] Pineau J. Kim, B. Socially adaptive path planning in human environments using inverse reinforcement learning. *Int J of Soc Robotics*, 8, 2016.

[7] Thibault Kruse, Alexandra Kirsch, E. Akin Sisbot, and Rachid Alami. Exploiting human cooperation in human-centered robot navigation. In *19th International Symposium in Robot and Human Interactive Communication*, pages 192–197, 2010.

[8] Thibault Kruse, Amit Kumar Pandey, Rachid Alami, and Alexandra Kirsch. Human-aware robot navigation: A survey. *Robotics and Autonomous Systems*, 61(12):1726–1743, 2013.

[9] Kai O. Arras Matthias Luber. Multi-hypothesis social grouping and tracking for mobile robots. *Robotics: Science and Systems*, 2013.

[10] Marija Seder and Ivan Petrovic. Dynamic window based approach to mobile robot motion control in the presence of moving obstacles. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 1986–1991, 2007.

[11] Masahiro Shiomi, Francesco Zanlungo, Kotaro Hayashi, and Takayuki Kanda. Towards a socially acceptable collision avoidance for a mobile robot navigating among pedestrians using a pedestrian model - international journal of social robotics, Apr 2014.

Using the Julia language to solve set problems in robotics

Enzo-Loïd ESSONO

Engineering student
ENSTA Bretagne
Brest, FRANCE
enzo-loid.essono@ensta-bretagne.org

February 16, 2022

Abstract

This article aims to present the Julia language and to show how we can use it to solve set problems in robotics. To do so, we will apply it to the case of SLAM where we will have to localize ourselves with only the estimated position of some markers.

1 Introduction

Interval arithmetic provides an inexpensive way to compute an overestimate of the range of a function over an input set. These estimates are guaranteed to be correct (mathematically rigorous), even if the calculations are performed with floating point arithmetic, using directed rounding. In order to solve some localization problems, interval methods are becoming more and more recognized. However, the programming of interval arithmetic is not always easy to code as well as to understand in languages such as Python, R, C...

In this article we will show how with the help of Julia and its bridge to already existing libraries, can become a very powerful tool in the robotics world.

2 Jula's presentation and background

2.1 What is Julia?

Julia is a programming language created specifically for data science, complex linear algebra, data mining and machine learning. The creators of Julia wanted to address the drawbacks of Python and other programming languages, by offering a more practical tool. Julia was intended for users of scientific languages and environments such as R, Octave, Matlab and Mathematica. The syntax of this language resembles the formulas used by non-programmers, making it easier for mathematicians to learn.

2.2 Objective

The goal is to learn how to use the Julia language in order to use it in a set context. This would allow us to have an additional tool in robotics programming.

3 Interval in Julia

3.1 IntervalArithmetic

In this section, we examine a recently developed Julia package for performing validated numerical computations, i.e., rigorous computations with finite precision floating point arithmetic, IntervalArithmetic.jl [1]. Its basic object is the parameterized type Interval. The objects are initialized in float64. In order to create an interval, the macro @interval must be used. **Examples of use**

```
using IntervalArithmetic
X1 = @interval(0.1, 0.3)
# note: lower-case 'i'
X2 = 0.1..0.3     # .. operator
X1 + X2
f(x) = x^2 + 2x
X = -1..1
f(X)

Output:
X1 = [0.0999999, 0.300001]
X2 = [0.0999999, 0.300001]
X1 + X2 =[0.199999, 0.600001]
f(x) = x^2 + 2x :f (generic function with 1
X = [-1, 1]
f(X)=[-2, 3]
```

3.2 IntervalConstraintProgramming

In 2014, Jaulin et al. established properties with a novel interval approach to validate a quasi-capture tube, i.e., a candidate tube (of simple shape) from which the mobile can escape[2]. Thus, we will see

from here an interval constraint programming solver dedicated to the validation of near-capture tubes.

The Julia package IntervalConstraintProgramming.jl allows you to specify a set of constraints on real-valued variables, given by (in)equalities, and rigorously compute inner and outer approximations of the feasible set, i.e. the set that satisfies the constraints. This uses the interval arithmetic provided by the IntervalArithmetic.jl package, in particular the multidimensional IntervalBoxes, i.e. the Cartesian products of one-dimensional intervals.

Example of programming with constraints

```
using IntervalConstraintProgramming
using IntervalArithmetic

vars = @variables x, y
C = Separator(vars, (y - 5) *
cos(4 * sqrt((x - 4)^2 + y^2)) - x *
sin(2 * sqrt(x^2 + y^2)) > 0)
X = IntervalBox(-10..10, 2)
p = pave(C, X, 0.05)
```

Plotting

In this section, we will illustrate how to visualize the interval extension of a given function over an interval. The process of splitting the interval into many smaller adjacent pieces for range evaluations of the given function is called mincing.

The figure below is an illustration of the sample code for the Interval constraint programming section[3].

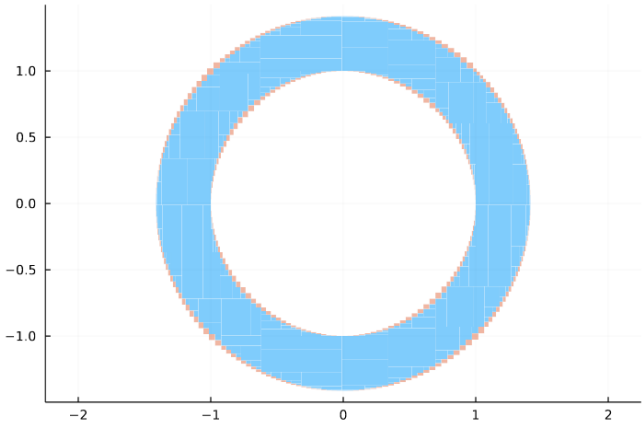


Figure 1: Ring in Julia

3.3 Other methods

The biggest strength of Julia is that we can call libraries from other environments such as python, C++... so if the user is more comfortable with interval definition in other modules like ibex and tube illustration with tubex, he can call his modules in Julia and the conversion will be done automatically as long as there are no syntax conflicts. To illustrate, we will use PyCall to call pyibex and vibes for the display.

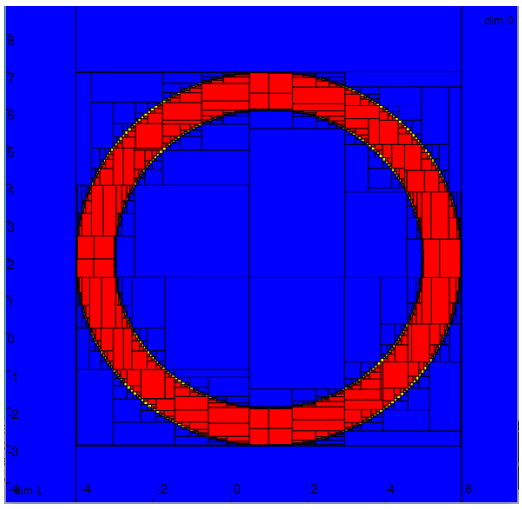


Figure 2: PyIbex ring en Julia

4 SLAM

Simultaneous Localization and Mapping (SLAM) is a classical localization problem in robotics.
Consider a robot at position (x, y) moving inside an unknown environment. We assume that its motion is described by the discrete time state equations[4]:

$$\begin{cases} x(k+1) = x(k) + 10 \cdot \delta \cdot \cos\theta(k) \\ y(k+1) = y(k) + 10 \cdot \delta \cdot \sin\theta(k) \\ \theta(k+1) = \theta(k) + \delta \cdot (u(k) + nu(k)) \end{cases}$$

Where :

k $\in \{0, ..., 100\}$
$\delta = 0.1$
θ : to the heading
u : to the desired rotational speed
nu :to a noise

The desired input u (k) is chosen as:

$$u(k) = 3 \cdot \sin^2(k\delta)$$

We assume that the heading is measured (using a compass for instance) with a small error:

$$\theta_m(k) = \theta(k) + n\theta(k)$$

For all k we assume that n_u (k) and n_θ (k) belong to [0.03, 0.03].
In the environment, we assume that we have 4 landmarks, the coordinate of which are given by the following table.

i	0	1	2	3
m(i)	(6,12)	(-2,-5)	(-3,20)	(3,4)

We can see on figure 3, that we can correctly represent the trajectory and the position uncertainties with the Julia language. At the beginning of the trajectory, we can see the contraction of a box in magenta it is the one of the first landmark. For

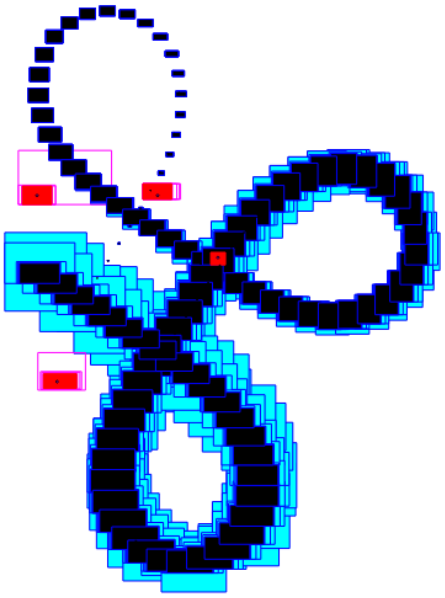


Figure 3: SLAM in Julia

the trajectory, at the beginning the accuracy is good but the more the position of the robot will become uncertain, the larger the contraction will become. The robot is in the black trajectory. The accuracy is managed by the possibility that the robot sees a landmark.

5 Conclusion

Finally, just as it was possible to solve problems with interval theory in robotics with languages like C++, Python and Matlab, it is now possible to do it with Julia. Julia's synthaxis is quite intuitive for scientists and programmers, it's a bit of a mix between Python and Matlab. With the right packages installed, it is possible to have performances equivalent to the C language.

I can still note that there are some problems in Julia, like the fact that it is not possible to make classes natively. I have to call Python if I want to make object oriented. There is also the problem of indexing which, like Matlab, starts at 1 instead of 0, which can cause problems for some programmers.

References

[1] Evgeniya Vorontsova. Interval computations in julia programming language. *HAL*, 2019.

[2] Luc Jaulin Stéphane Le Menec Abderahmane Bedouhene Bertrand Neveu, Gilles Trombettoni. An interval constraint programming approach for quasi capture tube validation.

[3] David P. SANDERS. Interval constraint satisfaction. 2019.

[4] Luc JAULIN. Iamooc: exercises.

Utilisation de réseaux adversariaux génératifs (GAN) pour casser les captchas de texte

Martin GOUNABOU
Elève ingénieur
ENSTA Bretagne
martin.gounabou@ensta-bretagne.org

***Abstract*—Using a captcha synthesizer and a refiner trained with GAN (Generative Adversarial Network), it is possible to generate synthesized training pairs to classify captchas. The objective is thus through unsupervised simulated learning, to succeed in breaking and then protecting text captchas.**

INTRODUCTION

En utilisant un synthétiseur de captchas et un raffineur entraîné avec GAN (Generative Adversarial Network), il est possible de générer des paires d'entraînement synthétisées pour classer les captchas. L'objectif est donc à travers l'apprentissage simulé non supervisé, de réussir à casser les captchas de texte.

I. PRÉSENTATION DU PROBLÈME

Avec le développement rapide de l'Internet, de plus en plus de services en ligne et de ressources de sites web sont menacés par des bots malveillants. Ces bots effectuent une série d'actions malveillantes sur Internet en simulant l'identité de véritables utilisateurs réels, notamment l'envoi de spam, le vol de ressources de sites web par le biais de robots d'indexation, etc (cf. ref. [1]). Le captcha est un moyen utilisé par plusieurs sites web, pour différencier de manière automatisée un utilisateur humain d'un ordinateur. Un captcha requiert donc à l'utilisateur de saisir des lettres et/ou des chiffres à partir d'une image fortement bruitée. Pour améliorer la sécurité, les chercheurs ont essayé d'ajouter des mécanismes de sécurité anti-reconnaissance pour les systèmes CAPTCHA textuels existants. Ces mécanismes de sécurité populaires comprennent l'utilisation de plusieurs styles de police, des interférences complexes en arrière-plan, la rotation des caractères, le chevauchement, la distorsion, les structures à deux couches, les arcs bruyants etc (cf. ref. [1]). Ces mécanismes de sécurité rendent la segmentation des caractères difficile et font que les méthodes basées sur l'apprentissage profond ont besoin d'un grand volume e données d'entraînement [4]. Et en général, des dizaines de milliers d'échantillons doivent être étiquetés pour atteindre des taux de réussite élevés en matière d'attaque. En outre, de plus en plus de sites Web ont ajouté de nombreuses mésures de protection pour empêcher les CAPTCHAs d'être téléchargés de manière malveillante par des attaquants, ce qui rend plus difficile de rassembler des échantillons de CAPTCHAs réels qu'auparavant. Les réseaux adversaires génératifs (GAN) sont une solution pour palier à ce problème de base de données d'apprentissage, car avec les GAN, nous seront capables de générer des images captchas qui viendront à leurs tours enrichir notre base de données.

II. FONCTIONNEMENT DES RÉSEAUX ADVERSAIRES GÉNÉRATIFS

Un GAN ou Generative Adversarial Network (réseau antagoniste génératif en français) repose sur la mise en compétition de deux réseaux au sein d'un framework. Ces deux réseaux sont appelés "générateur" et "discriminateur". Le générateur est un type de réseau neuronal convolutif dont le rôle est de créer de nouvelles instances d'un objet. De son côté, le discriminateur est un réseau neuronal "déconvolutif" qui détermine l'authenticité de l'objet ou s'il fait ou non partie d'un ensemble de données. Pendant le processus d'entraînement, ces deux entités sont en compétition et c'est ce qui leur permet d'améliorer leurs comportements respectifs. C'est ce que l'on appelle la rétropropagation. L'objectif du générateur est de produire des outputs sans que l'on puisse déterminer s'ils sont faux, tandis que l'objectif du discriminateur est d'identifier les faux. Ainsi, au fil du processus, le générateur produit des outputs de meilleure qualité tandis que le discriminateur détecte de mieux ne mieux les faux. De fait, l'illusion est de plus en plus convaincante au fil du temps.

Le schéma ci-dessous résulte le fonctionnement des GANs.

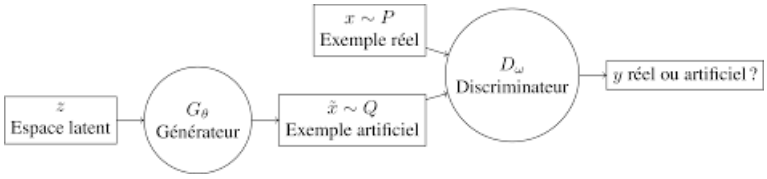


Fig. 1. Schéma de fonctionnment d'un GAN

III. IMPLEMENTATION

A. Prétraitement

Grâce au site web, "https:captcha.delorean.codesurickyhanchallenge", nous avons recolté 20000 captchas qui serviront à l'entrainement de notre GAN. Ces captchas sont binarisés car la binarisation permet d'économiser un calcul important.

Fig. 2. Captcha en couleur

Fig. 3. captcha binarisé

Puisque, ces données étaient destinées à un concours, les lignes dépendaient du nom du participant. De ce fait, nos données ont les mêmes lignes horizontales. Dans la réalité, ces bruits, peuvent être filtrés en utilisant une transformation morphologique avec Opencv((cf. ref. [3])

En utilisant un masque bits(&=) pour filtrer itérativement les pixels noirs environnants, c'est-à-dire :

```
mask = np.ones((height, width))
for im in ims:
    mask = im
```

, on réussit à extraire les lignes horizontales (bruits) dans les images captchas. on obtient pour l'image ci-dessus :

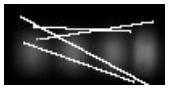


Fig. 4. Extraction des lignes horizontales

Ensuite, on utilise la fonction flow-from-directory de keras qui nous permet de prétraiter automatiquement les captchas à notre disposition.

B. Définition du modèle

Il est question dans cette section de définir un générateur qui va servir à générer des paires (captcha, étiquette).

Notre modèle sera défini à partir des trois composants suivantes :

- Raffineur: Le réseau de raffineurs, R, est un réseau résiduel (ResNet). Il modifie l'image synthétique au niveau du pixel, plutôt que de modifier de manière holistique le contenu de l'image, en préservant la structure globale et les annotations.
- Discriminateur: Le réseau discriminateur D, est un simple ConvNet qui contient 5 couches conv et 2 couches max-pooling. C'est un classificateur binaire qui indique si un captcha est synthétisé ou réel.
- Combiné: Canalisé l'image raffinée dans le discriminateur.

Le générateur fournit une valeur de sortie au discriminateur lorsqu'il reçoit une valeur d'entrée. Ensuite, le discriminateur calcule la différence entre la valeur reçue du générateur et les données source originales en utilisant une fonction de perte. Pour minimiser cette fonction de perte, le générateur ajuste ses paramètres et fournit une valeur de sortie différente au discriminateur. Au fur et à mesure que ce processus se répète, les données de sortie du générateur se rapprochent des données de la source d'origine. Dans ce processus, l'objectif principal du discriminateur est de faire la distinction entre la sortie du générateur et les données source originales. En revanche, l'objectif du générateur est de créer des données de sortie aussi proches que possible des données source originale que possible. Désignons par G et D les modèles génératifs et discriminatifs, respectivement. Les modèles jouent le jeu minimax à deux joueurs suivant avec une fonction de valeur V(G, D) comme décrit dans (cf. ref. [2])

$min_G max_D V(D, G = E_{x\ p_{data}(x)} + E_{z\ p_z(z)}[log(1 - D(G(z))]$

où D(x) et G(z) représentent respectivement les fonctions discriminantes et génératives du modèle.

C. Entraînement et résultats

Entraînement

Il s'agit de l'étape de formation la plus importante dans laquelle nous affinons un captcha synthétisé, puis le passons à travers les gradients de discriminateur et de backprop.

Le générateur fournit une fonction G(z) où z est un bruit. La distribution de G(z) se rapproche de celle de x en minimisant la fonction de perte du discriminateur. La fonction de perte peut exprimée comme :

$lossfunction = -y \log \hat{y} - (1 - y) log(1 - \hat{y})$

où y et ŷ représentent respectivement une distribution de probabilité valide et une probabilité logarithmique non échelonnée.

Résultats

Dans cette section, nous présentons des résultats expérimentaux et analysons la performance du modèle proposé. Le taux de reconnaissance du solveur CAPTCHA pour l'image source et l'image générée est 80 %. Notre résulat est plutôt bon, même si nous ne sommes pas dans les conditions réels. Les différents mécanismes tels que la distribution du texte, le niveau de distorsion et la rotation des caractères et l'image de fond n'ont pas été pris en compte. Si l'on avait consisidéré ces mécanismes, les taux de reconnaissance pour les images originales seraient très variables, car les images auraient des caractéristiques différentes(cf. ref. [2]).

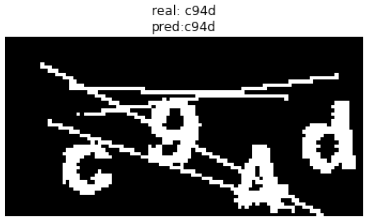


Fig. 5. Prédiction captcha synthétisé

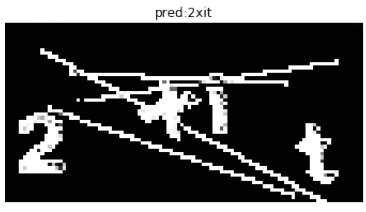


Fig. 6. Prédiction captcha réel

IV. Conclusion

Dans cet article, nous avons proposé une méthode de rupture de CAPTCHA de bout en bout basée sur des réseaux adversariens génératifs. Nous avons entraîné des synthétiseurs basés sur le cycle GAN pour générer un grand nombre d'échantillons CAPTCHA synthétiques, ce qui résout le problème du manque de données d'entraînement. Les résultats expérimentaux démontrent que plus de mécanismes de sécurité peuvent améliorer la sécurité des CAPTCHAs. Néanmoins, une fois que les attaquants obtiennent un nombre massif de données d'entraînement à travers diverses méthodes, ces mécanismes de sécurité peuvent ne plus être valables. Cela montre que les systèmes CAPTCHA textuels actuels ne sont pas sûrs et ne peuvent pas résister aux attaques basées sur l'apprentissage automatique (cf. ref. [2]).

Bibliographie

[1] Chunhui Li, Xingshu Chen, Haizhou Wang, Yu Zhang, Peiming Wang, "An End-to-End Attack on Text-based CAPTCHAs Based on Cycle-Consistent Generative Adversarial Network", 2017

[2] Hyun KWON, Member, Yongchul KIM, Hyunsoo YOON, Daeseon CHOI, "CAPTCHA Image Generation Systems Using GenerativeAdversarial Networks"

[3] https://github.com/0b01/SimGAN-Captcha/

[4] D. George, W. Lehrach, K. Kansky, M. Lazaro-Gredilla, C. Laan, ´ B. Marthi, X. Lou, Z. Meng, Y. Liu, H. Wang et al., "A generative vision model that trains with high data efficiency and breaks text-based captchas," Science, vol. 358, no. 6368, p. eaag2612, 2017.

Reconstitution 3D de câble par stéréo-vision.

GOURRET Yohann, FISE 2022

***Abstract* – L'objective de ce document est de présenter une méthode pour pouvoir reconstituer un câble en 3D à partir de deux caméras de position relative et de propriété connue. Cet article a vocation à se pencher sur les câbles sous-marins des ROVs (Remotely Operated underwater Vehicle) dont la connaissance de leur position permettrait d'éviter de faire des noeuds.**

I. INTRODUCTION

Les robots sous-marins opérés à la surface nécessitent un lien filaire par l'intermédiaire d'un câble. En effet sous l'eau les ondes radio sont à prescrire pour la communication. La lumière peut être utilisée pour de faibles distances et de faible turbidité. On pourrait utiliser les ondes sonores mais elles ne permettent pas de débit important.

Dans ce cadre, la communication filaire est tout adaptée, mais ajoute de nouvelles contraintes. Bien sûr, il y a d'abord la porté qui est limité par la longueur du câble. De plus, il y a un risque que le câble s'emmêle, fassent des noeuds, et ce risque est d'autant plus élevé que le câble est long. Sur ce deuxième point, pour pouvoir éviter de s'emmêler, il faut d'abord connaître la position en trois dimensions du câble dans le relativement au robot.

II. IDENTIFICATION DU CABLE

La première étape pour déterminer la position du câble en 3D est de l'identifier dans les images 2D. Comme base travaille nous utiliserons une image Fig. 1 issus d'une vidéo capturée par ESSONO AUBAME Enzo-Loïd, PROUTEN Samuel, SABATIER Hugo et GOUNABOU Martin dans le cadre de leur projet de Guerlédan. On y voit le ROV avec son câble jaune sur la droite.

La première étape, standard en traitement d'images, est de filtrer les hautes fréquences. Pour cela on commencera par appliquer un filtre gaussien. Ensuite nous tâcherons d'identifier le câble par sa couleur.

La stratégie va être de créer une image en niveau de gris qui représentera la distance entre chaque pixel et la couleur de référence, et cela en étant invariant en intensité. La première étape va être de transformer notre image RGB (Red, Green, Blue) en image HSV (Hue, Saturation, Value). On a donc une image matricielle où chaque pixel est définit par un vecteur 3 représentant sa couleur, sa saturation et son intensité. Puisque l'on veut être invariant en intensité, on peut la mettre à 0 pour l'ensemble de l'image. On a donc une image formée de pixel :

$$\forall x,y : \vec{p_{x,y}^{init}} = \begin{pmatrix} H_{x,y} \\ S_{x,y} \\ 0 \end{pmatrix} \tag{1}$$

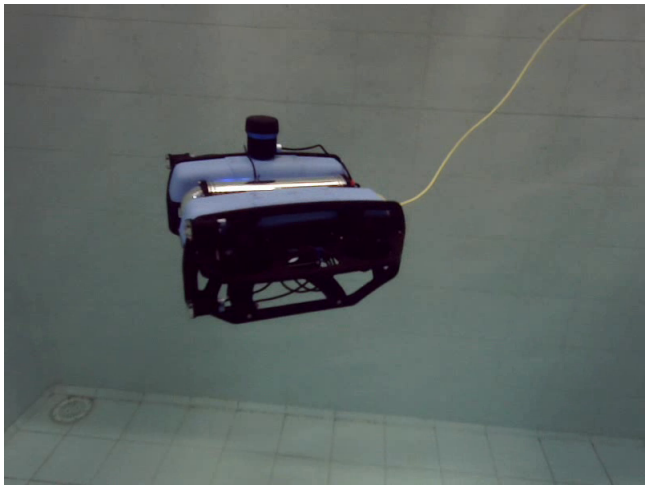


Fig. 1. Image initiale.

Soit $\vec{C}$ la couleur référence :

$$\vec{C} = \begin{pmatrix} H_c \\ S_c \\ 0 \end{pmatrix} \tag{2}$$

On définit l'image de la distance à la couleur de référence comme étant :

$$\forall x,y : \vec{p_{x,y}^{dist}} = ||\vec{p_{x,y}^{init}} - \vec{C}||_2 \tag{3}$$

La Fig. 2 représente l'image de distance à la couleur une fois normalisée. Comme on peut le voir le câble est plus sombre. Pour l'identifier il faut suivre une vallée de potentiel.

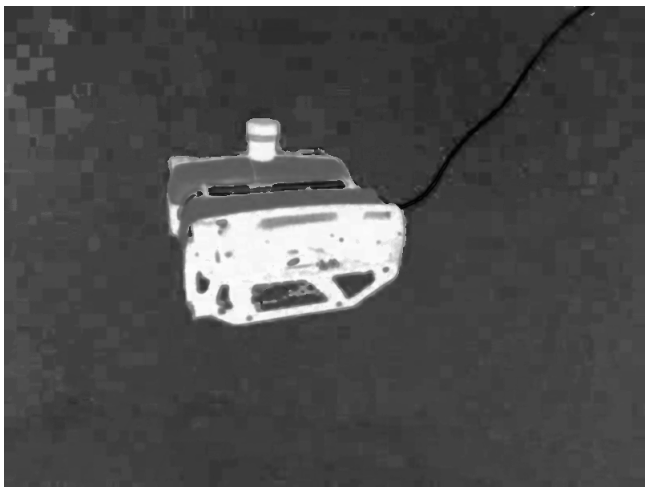


Fig. 2. Vallée de potentiel.

Pour cela, ont crée une courbe en trouvant les points qui au fur et à mesure minimisent le potentiel et la variation de trajectoire. La minimisation de la variation de trajectoire permet de prendre en compte le cas où le câble boucle. En pratique, le premier point est trouvé en longeant les bords de l'image.

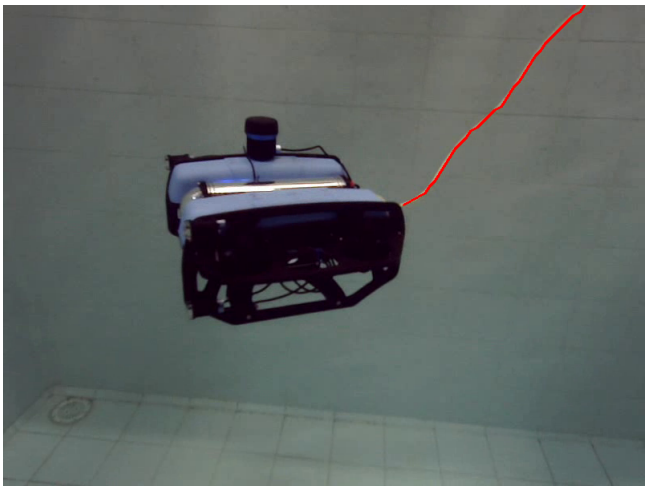


Fig. 3. Identification du câble.

Sur la Fig. 3 on peut voir en rouge la trajectoire du câble trouvé par l'algorithme. Il s'agit d'une série de points reliés entre eux pour l'affichage.

L'intérêt d'utiliser une vallée de potentielle plutôt que de segmenter l'image directement à partir par un seuil de couleur, est qu'une segmentation par seuil, en tout ou rien, sera souvent soit trop restrictive, soit pas assez. Elle sera peu résiliente face au variation de couleur. Un critère continu permet de diminuer ce risque.

III. STEREO-VISION

Le principe de la stéréo-vision est d'utiliser deux caméras, ayant deux points de vue différents, pour pouvoir retrouver l'information de profondeur, l'information 3D. Dans notre cas, puisque l'on peut placer 2 caméras sur notre ROV on peut estimer que l'on connaît déjà le positionnement relatif et l'orientation relative de ces caméras.

La configuration la plus simple est celle présenté dans le livre Machine Vision (Jain et al. 1995) dont on peut voir un schéma en Fig. 4. Dans cette configuration, le passage de la caméra gauche à la caméra droite n'est qu'une translation sur l'axe ici définit comme x. De plus les caméras sont identiques : même angle de vue, même résolution. On choisira de prendre le foyer de la caméra gauche C_1 comme origine.

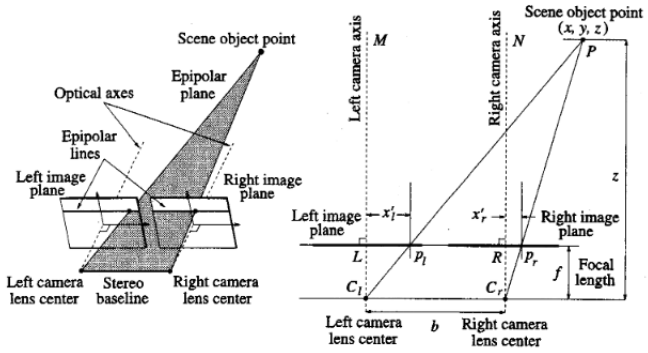


Fig. 4. Stereo-vision simple. Schéma tiré de Machine Vision, Jain et al. 1995.

Cette configuration nous permet d'obtenir la relation suivante :

$$z = \frac{bf}{x_l' - x_r'} \tag{4}$$

Avec b la distance entre les deux caméras, f la focale des caméras, x_l' la projection du point sur la caméra gauche et x_r' la projection du point sur la caméra droite.

Ensuite on obtient facilement les coordonnées x, y grâce au théorème de Thalès :

$$x = x_l' \frac{z}{f} \tag{5}$$

$$y = y_l' \frac{z}{f} \tag{6}$$

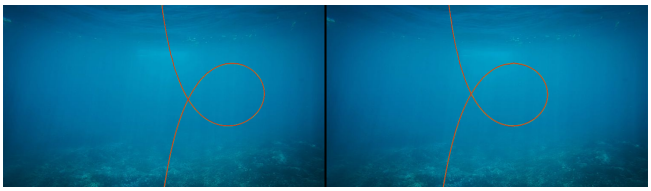


Fig. 5. Image Gauche et droite

La Fig. 5 (obtenus par ordinateur, ce n'est pas une photo) représente un câble vue de deux caméras séparé de 0.5 m et avec un angle de vue de 90° sur l'axe horizontal.

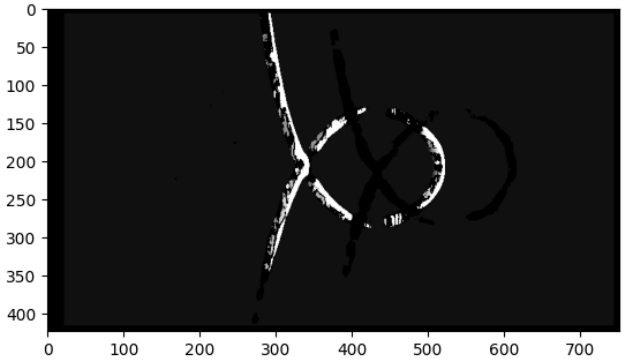


Fig. 6. Stéréo-vison de OpenCV

La Fig. 6 montre le résultat d'un algorithme de stéréo-vision classique (ici celui par défaut de OpenCV). Il n'y a pas de variation de profondeur sur le câble. En effet les algorithmes de stéréo-vision vont généralement segmenter l'image et ensuite faire un matching pour retrouver l'information de profondeur. Cela marche bien des objets mais pour un câble monochrome (donc un objet) qui varie beaucoup en profondeur, cela ne fonctionne pas.

Pour y parvenir, on utilise dans un premier temps l'algorithme précédent pour obtenir les trajectoires 2D projetées sur les caméras. Dans notre cas on choisit la caméra gauche comme caméra principale. Pour chaque point de la trajectoire gauche, on cherche les points possibles sur la

trajectoire droite. Dans la configuration simple des caméras de notre cas, il s'agit de tous les points de la trajectoire qui coupent l'horizontal du niveau du point de la trajectoire gauche. Ensuite, on calcule la position 3D de tous ces points et on détermine quel est le plus adapté. Soit simplement le point le plus proche du précédent, soit avec un filtre de Kalman : un tracking de trajectoire avec un filtre de Kalman. La seconde approche étant plus adaptée en cas de croisement. Ici on a pu se contenter du point le plus proche. En effet, il y certes un croisement en 2D, mais ce n'est pas le cas en 3D.

On a donc pu obtenir la Fig. 7. Les points rouges sont les points de la trajectoire réelle et les points bleu ceux de la trajectoire obtenue par l'algorithme de stéréo-vision. A noté que la Fig. 7 est d'un autre angle de vue que les images de la Fig. 5. On remarque que les points issus de la zone où la trajectoire du câble est horizontale sont moins précis, car l'algorithme ne peut pas discriminer des cas comme cela.

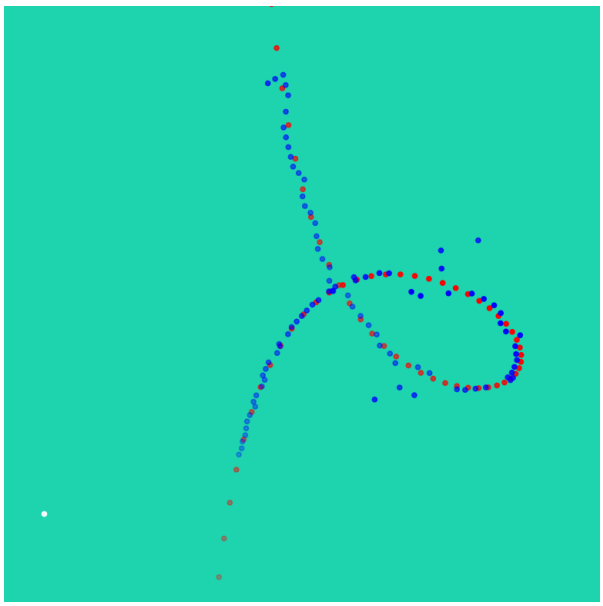


Fig. 7. Reconstitution 3D du câble

BIOGRAPHIES

Nalpantidis Lazaros, Georgios Christou Sirakoulis & Antonios Gasteratos (2008) Review of Stereo Vision Algorithms: From Software to Hardware, International Journal of Optomechatronics, 2:4, 435-462, DOI: 10.1080/15599610802438680

Jain, R., R. Kasturi, and B. G. Schunck. 1995. Machine vision. New York, USA: McGraw-Hill.

Bleyer, Michael and Margrit Gelautz. 2004. A layered stereo algorithm using image segmentation and global visibility constraints. Proceedings of International Conference on Image Processing 5:2997–3000.

Bleyer, Michael and Margrit Gelautz. 2005. A layered stereo matching algorithm using image segmentation and global visibility constraints. ISPRS Journal of Photogrammetry and Remote Sensing 59(3):128–150.

Introduction to the Formalization of Robotic Tasks

Bernardo Hummes Flores*
* ENSTA Bretagne, Lab-STICC, Brest, France
bernardo.hummes@ensta-bretagne.org

***Abstract*—The general context of this work is the logical and categorical formalization of mobile robots tasks. The aim of this formalization is to be able to prove correct algorithms that perform some tasks between robots (such as robot gathering) under various hypotheses (e.g. perfect or imperfect localization, bounded or unbounded visibility, synchronous or asynchronous decisions etc.), or to prove the impossibility that such tasks can be performed under these types of hypotheses. An initial approach is made by introducing the elements from category theory and modal logic needed for constructing the algebra that will represent such formalization.**

I. Introduction

The starting point for this project is to create formal links between the known computing models for robotics, such as the look-compute-move model [1], and distributed computing models [2]. For such purposes, modal logic [3], [4] is a suitable formal system, as it allows for the representation of multiple aspects related to the states and knowledge of robots over time, mainly with the extensions of temporal and epistemic logic [5].

Those interpretations are possible from the construction of an algebra capable of expressing those relations, and category theory offers an expressive and versatile framework over which the modeling can happen. As an approach to an even more general view of transitions over collections than universal algebra [6], category theory allows us to recast the models as algebras over a functor acting on a chromatic simplicial set [7], which are higher-order representations of directed graphs, partially ordered sets and categories. Such developments are the object of future work and only the starting basis will be enunciated here, as a resource for future introducing to the topic and research.

This work will continue with a brief introduction to modal logic in section II, going over the basic formalism and the key extensions considered for this work. A series of important definitions in category theory will be given in section III, while interesting intersections of categorical and modal logic concepts are presented in section IV. The expected continuations over this introduction will be shown in section V and some concluding words in VI.

II. Modal logic

Modal logic constitutes a super-set of the standard formal logic, with added modal operators that allow for the expression of notions similar to "possibility" and "necessity", which use as notation the symbols $\Box$, read as "box" and $\Diamond$, read as "diamond". Those additions allow us to create formulas composed of propositions, statements that can be either true or false, with added meaning, such as "*p* will *eventually* become true" and "*p always* is true throughout some relevant range of situations".

Multiple interpretations exist of modal logic, interesting examples are Godel's view of $\Box p$ as the "mathematical provability of p" and Tarski's view of modal formulas as describing subsets in topological spaces. Another of its crossings to different fields is Montague semantics, where intentional expressions are studied with a mix of modal logic and type theory. Notable intersections are also with works in Philosophy and Computer Science. The idea itself of modal logic being an enrichment of classical logical is not agreed by all, and a common perspective is of it as "local" quantifiers that refer to objects "accessible" from the current one, which will be explored a bit with the possible worlds model in II-A.

The first step for a formalization is the definition of the atomic elements

$$AT = p, q, r, ..., \top, \bot$$

with generic atoms, an atom representing "always true" and a last one representing "always false", respectively.

This is then followed by a way to construct formulas from those atoms

$$\varphi ::= AT \mid \neg\varphi \mid (\varphi \wedge \psi) \mid (\varphi \vee \psi) \mid (\varphi \rightarrow \psi) \mid \Diamond\varphi \mid \Box\varphi$$

From this, the first concept to be understood is that all atoms are formulas, and if an atom is a formula, then so is its constructions as $\neg\varphi$, $(\varphi \wedge \psi)$, and so it goes recursively. With this, all formulas can be defined in a finite number of steps. Examples of equivalent expressions are:

1) $\Diamond\varphi \leftrightarrow \neg\Box\neg\varphi$
2) $\Box\varphi \leftrightarrow \neg\Diamond\neg\varphi$
3) $\neg\Box(\Diamond\varphi \wedge \Diamond\neg\varphi) \leftrightarrow \Box\Diamond\varphi \rightarrow \Diamond\Box\varphi$

Where the first one can be read as "it is possible that φ is true if and only if it is not necessary that φ is not true", and the second as "φ is necessarily true *iff* it is not possible that φ is not true". The third one is a formalization of the idea "nothing is absolutely relative", with the possible interpretation of the first half as "it is not necessary that the possibility of φ coexists with the possibility of $\neg\varphi$", and the second second half is its rewriting into McKinsey axiom, an important conclusion in modal logic that will not be explored here.

In our case, modal logic will be the tool, together with the extensions bellow, used to represent the robotic scenarios with actionable tasks.

A. Possible worlds model

An important usage of the modal logic system is the multiple worlds model, in which we give a semantic interpretation to the formulas as a graph-like structure that represent the possible truth values for the propositions involved in a given formula.

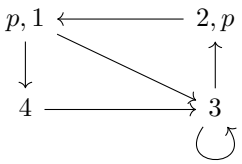
The possible worlds model is defined as a triple $M = (W, R, V)$, composed of a collection of worlds W, an accessibility relation between worlds R, and a valuation map with truth values V for a proposition *p* at the world s. The relation R expresses the constraints in transitioning between worlds, while the complete graph may display the possible evolutions from an initial proposition. The truth values obtainable from $V(p, s)$ can also be expressed as $\mathbf{M}, s \models \varphi$, whenever φ is true in a given state s of the model $\mathbf{M}$.

The truth definition in the possible worlds models happens as follows

1) $\mathbf{M}, s \models p$ iff $V(p, s) = 1$
2) $\mathbf{M}, s \models \neg\varphi$ iff not $\mathbf{M}, s \models \varphi$
3) $\mathbf{M}, s \models \varphi \wedge \psi$ iff $\mathbf{M}, s \models \varphi$ and $\mathbf{M}, s \models \psi$
4) $\mathbf{M}, s \models \Box\varphi$ iff for all inciding t as: $\mathbf{M}, t \models \varphi$
5) $\mathbf{M}, s \models \diamond\varphi$ iff for some inciding t as: $\mathbf{M}, t \models \varphi$

Here, 1 is the basic definition of the valuation map, 2 and 3 expand this definition to the negation and conjunction. 4 says that for the valuation of φ in the world s in the model $\mathbf{M}$ to be necessarily true, it has to hold in *every* world t of the same model that leads to s. 5 says that for possibility of φ being true when valuated in a similar situation, *some* world t of the same model that leads to s has to be true. Finally, the concept of modal validity can be affirmed to a given formula φ if $\mathbf{M}, s \models \varphi\ \forall(\mathbf{M}, s)$, also written as $\models \varphi$.

Having those definitions, we can profit of the possible worlds diagrams to both have a visual representation of our formulas and to infer the possible values of each "node" from the relationship graph of an existing system. A way of using the latter is finding where a given modal formula holds in the analyzed graph, such as



(1)

In the case that wish to know in which worlds the formula $\Diamond\Box\Diamond p$ holds, we can use the truth value definitions in the graph II-A to expand the known information, giving us the following (non exhaustive) valid formulas after some iterations:

- 1: $\Diamond\Box\Diamond p$
- 2: $p,\ \Diamond p$
- 3: $\Diamond p,\ \Box\Diamond p,\ \Diamond\Box\Diamond p$
- 4: $\Box\Diamond p,\ \Diamond\Box\Diamond p$

Note that the values *p* are true in 1 and 2 since the beginning of the evaluation. As it is impossible to conclude $\Diamond\Box\Diamond p$ for the world 2, no matter how many more iterations, we cannot say that it is a valid formula, i.e. it is not true in all worlds of the model. However, we have arrived to the fact that the modal formula holds for the worlds 1, 3 and 4.

Those examples only make use of the base formal system of modal logic, however it is important to note that they are compatible with the temporal-epistemic extensions and that integration is relevant for the envisioned work, even though they are not discussed here.

B. Temporal extension

The first extension to modal logic that suits our needs increasing our capacity of formally representing robotics tasks is the temporal logic. It is capable of placing propositions in the past or future, with certainty or not, either as a finite event or a continuous one. The formalization starts with the following addition to the syntax rules and interpretations

$$\varphi ::= ... \mid F\varphi \mid P\varphi \mid G\varphi \mid H\varphi$$

where $\mathbf{F}$ and $\mathbf{P}$ are initially defined as the idea of "at least once in the future, φ will be true" and "at least once in the past, φ has been true", respectively. Those two operators are derived into the universal modalities $\mathbf{G}$ and $\mathbf{H}$, that inform that "always in the future, from now, φ is true" and "always in the past, until now, φ was true".

From this initial specification some interesting expressions can already be formed, such as:

1) $\mathbf{GF}\varphi$
2) $\mathbf{PH}\varphi$
3) $\mathbf{G}(\varphi \rightarrow \mathbf{F}\varphi)$

Expressions 1 and 2 join join the base operators with universal modalities, generating finite events that are sure to happen at some point, being them read as "φ is always going to be true at some later stage" and "once upon a time, φ had always been true", respectively. Example 3 can be read as "φ will always imply that φ will be true in the future", that can also be understood as φ being true will always "enable" itself afterwards.

Those constructions also enable the interpretation of validity throughout time across worlds (possibilities of a system, shown in section II-A), such as:

1) $\mathbf{M}, t \models \mathbf{F}\varphi$ *iff* for some $t' > t : \mathbf{M}, t' \models \varphi$
2) $\mathbf{M}, t \models \mathbf{P}\varphi$ *iff* for some $t' < t : \mathbf{M}, t' \models \varphi$

which shows how the ordering of moments in time "before" or "after" others may be used for creating expressions in this extension.

Similarly to what will be presented on the following section on epistemic logic, there are further tools available that allow for the deduction and extraction of truths about the scenario

that are fundamental in guaranteeing the behavior of some studied system.

C. Epistemic extension

The epistemic extension to modal logic deals with knowledge, what information is shared and what can be assumed from what each one has access to. With it, the ideas of an agent having a certain information or something being common knowledge among a group of agents become formalized through the following increased syntax rules

$$\varphi ::= ... \mid K_i\,\varphi \mid C_G\,\varphi$$

where $K_i\,\varphi$ says that the agent i knows φ, and $C_G\,\varphi$ that the group of agents G know φ.

The idea of an agent i considering the possibility of some information φ has the special notation of $\langle i\rangle\varphi$, being it is equivalent to $\neg K_i\,\neg\varphi$. The latter can also be read as "the agent i does not know if φ is false".

Here are some examples of how this new syntax can be used:

1) $\neg K_Q\,\varphi \wedge \neg K_Q\,\neg\varphi$
2) $\langle Q\rangle(K_A\,\varphi \vee K_A\,\neg\varphi)$

The first example models the idea that Q does not know weather φ is true, a translation from the more immediate reading of "Q doesn't know if φ and it also doesn't know if $\neg\varphi$". The second example makes use of the "knowledge possibility" notation, being possible to read it as "Q thinks that A might know φ".

This extension is mainly relevant for the study of distributed robotics scenarios, where the possible knowledge of the individual robots is fundamental for evaluating the feasibility of those scenarios. Other operators and interactions have not been explored so far but are equally interesting, such as the "universal knowledge" operator and the question/answer scenarios between two or more agents, that can generate the best estimation of the knowledge available with a series of rules on what are truth conditions for the system, such as the valuation map presented in the model above.

III. Category theory

Category theory came to existence as a set of formalizations allowing for the study of generic abstract structures in mathematics, being first proposed by S. Mac Lane in 1971 [8]. The key take from the categorical view is that *relations* are more important than *elements* to describe structure. The three main concepts are categories, functors, which allow for the transformation from one category to another, and natural transformations, that allow for the transformation from one functor to another. The first two will be discussed bellow.

Much of the interest in category theory comes from the fact that it reflects on itself, meaning that all constructions when viewed from the proper perspective give rise to their own category, such as functors between categories creating the functor category. This multitude of perspectives can be evaluated with varying levels of "sameness" (with a corresponding rigor), such as equality, isomorphism, equivalence, natural isomorphism and adjunction. In the same line of thought, it is also useful how that reflexivity allows for the perception of universal constructions, phenomena that are present throughout the different categories, each with their own interpretation but sharing the same properties, such as limits, colimits, adjunctions and ends. Those constructions will not be covered here but it must be noted the richness in theorical representations.

A. Categories

The first definition for such study is that of categories, which may be understood as families of structures with structure-preserving maps between them. A category $\mathscr{C}$ is composed of two kinds of things:

(1) $\mathscr{C}$ -objects, usually expressed with upper case letters;

(2) $\mathscr{C}$ -arrows, also called $\mathscr{C}$ -morphisms, usually expressed with lower case letters.

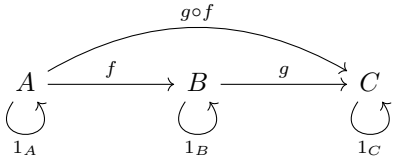
Where the objects must respect three axioms:

1) *Sources and targets*. For each arrow f, a specific collection of objects is associated with its domain and codomain, those not being necessarily distinct. We may write $f : A \to B$ for the arrow f with $src(f) = A$ and $tar(f) = B$, or $dom(f) = A$ and $cod(f) = B$.
2) *Composition*. Any pair of arrows $f : A \to B$ and $g : B \to C$, where $dom(g) = cod(f)$, may be composed into a new arrow $g \circ f : A \to C$. This is understood as the application of f followed by the application of g in the previous result, and is called the *composite* of f and g.
3) *Identity arrows*. Any given object A is equipped with an identity arrow $1_A : A \to A$, that has $dom(1_A) = cod(1_A)$.

And the arrows must respect other two axioms:

1) *Associativity of composition*. The composition operation is associative, which means that for any $f : A \to B$, $g : B \to C$, $h : C \to D$, we will always have that $h \circ (g \circ f) = (h \circ g) \circ f$.
2) *Identity arrows behave as identities*. The identity arrows can always be added to operations without change in result, such that for any arrow $f : A \to B$, we have that $f \circ 1_A = f = 1_B \circ f$.

Those properties are exemplified in the following commutative diagram, where paths that start and finish on the same objects are always equivalent.



(2)

B. Functors

Functors appear as the first level of higher-order abstraction, as they represent transitions between two categories. It is attributed to John Baez, a researcher in higher-order category theory, the quote "every sufficiently good analogy is yearning to become a functor.". This expressed the fundamental functioning of functors as ways to transfer acquired notions from one perspective into another.

Functors are one of the structure-preserving maps mentioned at the beginning of the section, and given two categories $\mathscr{C}$ and $\mathscr{D}$, the functor $F : \mathscr{C} \to \mathscr{D}$ satisfies the following attributes:

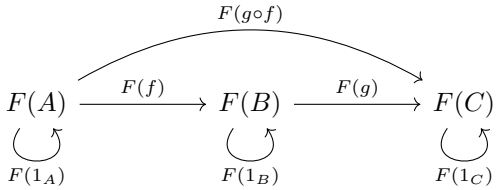
(1) $\mathscr{D}$ -objects are obtainable from $\mathscr{C}$ -objects through F, expressed as $F(A) = B$, belonging to the respective categories.

(2) $\mathscr{D}$ -arrows between $\mathscr{D}$ -objects are preserved through F, such that the arrow $F(f) : F(A) \to F(B)$ in $\mathscr{D}$ is equivalent to the arrow $f : A \to B$ in $\mathscr{C}$.

These operations via the functor F must respect the following conditions:

1) *Preserve identities*. Any identity arrow for a $\mathscr{C}$ object A will be mapped to the corresponding identity in $\mathscr{D}$, such that $F(1_A) = 1_{F(A)}$.
2) *Respect composition*. Any two arrows f and g in $\mathscr{C}$, with the corresponding composite $g \circ f$, with always have $F(g \circ f) = F(g) \circ F(f)$.

Those properties are expressed in the following diagrams, which is the same as III-A after having a functor F applied. It always has similar paths commuting.



(3)

C. Examples of categories

As mentioned before, categories may be used to represent an infinity of mathematical structures, table I shows some categories with their respective interpretations of the main elements. It is interesting to note that in this list there are algebraic categories (Set, Grp, Rng, Bool), order categories (Pos, Tot), geometric categories (Top, Met, Vect_k) and a logic category (Proof_T).

D. Duality

A useful property of categorical constructions is the immediately available dual interpretation. It corresponds to the $\mathscr{C}^{op}$ category associated with $\mathscr{C}$, where the following happens:

1) $\mathscr{C}^{op}$ -objects are the same as $\mathscr{C}$ -objects.
2) Given the arrows in $\mathscr{C}$, such as $f : A \to B$, the $\mathscr{C}^{op}$ will have the source and target swapped, such as $f^{op} : B \to A$.
3) The identity arrows are preserved the same, $1_A^{op} = 1_A$.
4) Composition also has the source and target swapped, such that $f \circ^{op} g = g \circ f$;

This way of constructing categories allows us to have "free proofs", from whatever we produce categorically. All concepts have dual interpretations and, even though we'll not explore it now, they are helpful tools in producing the required algebra.

IV. Formalization of robotic tasks

Category theory is capable of representing algebraic structures in multiple ways, one of them being through the usage of endofunctors, functors that map structures inside the same category. This can be represented with the following commutative diagram, where a given functor F may have an algebra as an object X of in $\mathscr{C}$, and the morphism $\alpha : F(X) \to X$, where X is the carrier of the algebra.

$$\begin{array}{ccc} F(X) & \xrightarrow{F(m)} & F(Y) \\ {\scriptstyle\alpha}\downarrow & & \downarrow{\scriptstyle\beta} \\ X & \xrightarrow{m} & Y \end{array} \tag{4}$$

given two algebras (X, α) and (Y, β), and having the composition $\circ$ to be the morphisms inside the category $\mathscr{C}$. This is an interesting example where the same element (morphisms) may have different interpretations according to the context it is used, in this case is is both a functor and the composition operator. This pattern of constructions gives the category of F-algebras, from a functor F.

This part of the work is still very uncertain, and some of the interpretations may still change considerably, consequently the explanations are not as deep as it would be ideal.

The core idea to be taken from this is the creation of an algebraic structure via functors operating over simplicial sets - which are a specific version of functors that map a given category into the Set category, but swap target and source during the mapping - may make it possible to represent the aforementioned modal logic. Once that modal logic is successfully described with the categorical algebra, it will be possible to reason within it using the categorical tools that have been presented, and many others that are available for reaching meaningful interpretations over the structures.

The robotic tasks may be represented as modal formulas, which could then be operated inside this algebra and, in an ideal scenario, reduced to a formula that represents a successful mission. An example would be starting from a long description of the initial configuration, with conjunction of facts as formulas such as "robot A does not know the map", "the knowledge of robot B comes after the Look step of robot A, but before the Look step of robot C", among much more information on the task, and have provable that there is a path from it to a formula representing "robot A meets robot B".

V. Future work

Assuming the successful formulation of the algebra mentioned in the previous section IV, some initial developments

TABLE I
SOME EXAMPLES OF NOTORIOUS CATEGORIES OF WELL KNOWN MATHEMATICAL STRUCTURES.

Category	Structure	Objects	Morphisms
Set	Sets	sets	total functions
Grp	Groups	groups	group homomorphisms
Ab	Abelian groups	abelian groups	group homomorphisms
Rng	Rings	rings	ring homomorphisms
Bool	Boolean algebras	Boolean algebras	structure preserving maps
Pos	Partially-ordered sets	sets	order preserving maps
Tot	Totally-ordered sets	sets	order preserving maps
Top	Topopologies	topological spaces	continuous maps between spaces
Met	Metric spaces	set of points S equipped with a real metric d	non-expansive maps
Vect_k	Vector spaces over a field k	set of vectors equipped with addition and multiplication by scalars in the field k	linear maps between spaces
Proof_t	Formal theory T	sentences φ, ψ, ... of the formal language T	the target of the arrow is sufficient to prove the source

will be possible on the study of distributed systems working on the gathering problem, where multiple robots must be guaranteed to meet in a finite amount of time. It should be interesting to study the classical gathering algorithms in the look-compute-move model [1], both synchronous and asynchronous, and to prove their convergence casting into the temporal-epistemic logic.

This work, initially considering only discrete systems, could then be extended for dynamical [9] and hybrid [10] ones, where the particularities of having a discrete processing unit for a continuous real world robot are better represented. This could also be followed by an extension of the algebra to support collections of states, in non-deterministic systems, as it was suggested with either the transition to co-algebras [11] or with multidimensional temporal logic [12].

Finally, multiple models of robots would be available for study, with varying combinations of sensory and movement capacities, also enunciated in [1].

VI. CONCLUSION

We have seen an introduction to the two fundamental theories necessary for the desired formalization of robotics tasks: modal logic, with the temporal-epistemic extensions, and category theory. The main goals are to be able to verify the feasibility and proper working of algorithms in mobile robotics scenarios, principally considering distributed systems, which present a higher amount of interdependent knowledge and implied information that are hard to evaluate empirically.

First we see the modal logics formal system as the common language to be used for such formalization, where tasks may be expressed as formulas containing information about possibility, knowledge and its temporal aspects, being possible to create dependencies between those affirmations, such as "robot A is possibly aware of an obstacle *o*, while it is sure to share the map *m* with robot B, being necessary that this one has knowledge of *o* after its perception by A."

Secondly, we explore category theory, as the mathematical tool for expressing all of those relations in a single algebra using its constructions. Although not all of the proceedings to arrive at the desired results are clear as of the writing of this article, the process of studying this domain already gives interesting hints on its expressive power in relating seemingly disconnected parts of mathematics. Notions of how categories allow for the study and representation of relations between objects such different perspectives with the same constructions, the understanding of functors as analogies between different interpretations of similar patterns that rise in distinct domains and the immediate view (and proof!) of dual concepts as a natural consequence of the categorical representation are enticing by themselves.

Finally, we have a glimpse on how the usage of that formalization may allow for the reasoning of interesting and reasonably complicated systems in a guaranteed way. All of it allowing us to consider the working of robots with tasked functionalities much before their implementation.

REFERENCES

[1] P. Flocchini, G. Prencipe, and N. Santoro, Eds., *Distributed Computing by Mobile Entities: Current Research in Moving and Computing*, ser. Lecture Notes in Computer Science. Springer International Publishing, vol. 11340. [Online]. Available: http://link.springer.com/10.1007/978-3-030-11072-7

[2] M. Alcántara, A. Castañeda, D. Flores-Peñaloza, and S. Rajsbaum, "The topology of look-compute-move robot wait-free algorithms with hard termination," vol. 32, no. 3, pp. 235–255. [Online]. Available: https://doi.org/10.1007/s00446-018-0345-3

[3] J. van Benthem, *Modal Logic for Open Minds*, ser. CSLI Lecture Notes. Center for the Study of Language and Information, no. no. 199.

[4] W. Carnielli and C. Pizzi, *Modalities and Multimodalities*.

[5] S. Knight, "The Epistemic View of Concurrency Theory," p. 217.

[6] M. Hyland and J. Power, "The Category Theoretic Understanding of Universal Algebra: Lawvere Theories and Monads," vol. 172, pp. 437–458. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1571066107000874

[7] H. van Ditmarsch, E. Goubault, J. Ledent, and S. Rajsbaum. Knowledge and simplicial complexes. [Online]. Available: http://arxiv.org/abs/2002.08863

[8] S. Mac Lane, *Categories for the Working Mathematician*, ser. Graduate Texts in Mathematics. Springer-Verlag, no. 5.

[9] M. Behrisch, S. Kerkhoff, R. Pöschel, F. M. Schneider, and S. Siegmund, "Dynamical Systems in Categories," vol. 25, no. 1, pp. 29–57. [Online]. Available: https://doi.org/10.1007/s10485-015-9409-8

[10] E. Goubault and S. Putot, "Inner and outer reachability for the verification of control systems," in *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*. ACM, pp. 11–22. [Online]. Available: https://dl.acm.org/doi/10.1145/3302504.3311794

[11] B. Jacobs, *Introduction to Coalgebra: Towards Mathematics of States and Observation*, ser. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, no. 59.

[12] M. Marx and Y. Venema, "Multi-Dimensional Modal Logic," in *Multi-Dimensional Modal Logic*, ser. Applied Logic Series, M. Marx and Y. Venema, Eds. Springer Netherlands, pp. 1–9. [Online]. Available: https://doi.org/10.1007/978-94-011-5694-3_1

Supervised pre-learning for deep networks

Thibault Jadoul
ENSTA Bretagne
Brest, France
thibault.jadoul@ensta-bretagne.org

***Abstract*—Gradient back propagation algorithms for neural network learning perform well if the initialization of the network weights is performed close to a good local minimum. One clever way to perform this initialization is the gluttonous, hierarchical layer pre-learning strategy. In the literature, most of the pre-learning approaches are akin to unsupervised feature learning and use for a good part of them autoencoder. In this paper, we will compare a "normally" trained network with a network pre-trained with an autoencoder. Thus, we determine a more suitable initialization than the simple random initialization of the input layers as well as the output layers.**

***Index Terms*—Deep learning, supervised pre-learning, autoencoder**

I. Introduction

In recent years, collections of papers on neural architecture learning have appeared in the literature, reflecting the resurgence of research activity around neural networks. This resurgence is largely due to the use of a strategy called deep learning. Indeed, neural networks are known to be sensitive to the choice of learning parameters (for example, the learning rate) and also to initialization, which are generally random. In this case, the back propagation algorithm used to estimate the weights of the neural network may converge to a local minimum located in a basin of attraction, which may not be very interesting from a model generalization perspective. This problem is mainly related to the phenomenon of gradient weakening across layers, which makes it difficult to adapt the weights of the input layer. To circumvent this difficulty, a clever approach is pre-learning, which consists in learning the intermediate layers of the network in an unsupervised and hierarchical way. The resulting weights can be used as an initialization for supervised refinement of the weights of the output layer (and possibly the first layers). An older paper by Erhan et al. provides some analysis and understanding of the success of this strategy. A typical way to implement pre-learning is an autoencoder (or auto-encoder). The basic idea is to project the x-inputs in space through the encoder. To measure the quality of the projection, a decoder is then used to reconstruct the original data. The quality of the encoder is measured via a reconstruction criterion. Once an autoencoder is learned, its decoder module is removed from the architecture and the encoder output is used as input data for the next autoencoder stage. This learning of the autoencoder can be likened to hierarchical feature learning that is used later to facilitate supervised learning tasks, and without the use of outputs that one wishes to approximate.

Identify applicable funding agency here. If none, delete this.

II. Principe

A. Autoencoder

Let $D = \{(xi, yi) \in X * Y\}_{i=1,---,n}$ be an input-output data set that we wish to approximate with a neural network. Typically, neural architectures include an input layer, one or more hidden layers and an output layer. In the deep learning strategies of the architectures proposed in recent years, the initialization of the weights is done by an autoencoder. An autoencoder consists of two modules: an encoder and a decoder, as shown in figure (1). The encoder is used to non-linearly project the data into a space of the same dimension as the number of hidden layer units. This projection generates a new representation h of the data that is likely to preserve useful information for the supervised learning task. For example, for data $x \in \mathbb{R}^m$ (the input layer will have m units) and the first hidden layer with p units, the autoencoder produces a code $h \in \mathbb{R}^p$ whose k-th coordinate has the expression:

$$h^{(k)} = f(a_k), k = 1, ..., p \; with; a_k = b_k + w_k^T x; (1)$$

In this equation, $w_k \in \mathbb{R}^m$ represents the weight vector, b_k the bias term. The function *f(a)* introduces the nonlinearity in the projection.
The decoding module aims at reconstructing the input x from the vector h. This decoding step is intended to check if the encoder has captured the useful information contained in the data. The reconstruction is based on the following equation:

$$\hat{x}^{(j)} = g(c_j), \; j = 1, .., m \; where; c_j = \overline{b}_j + \overline{w}_j^T h; (2)$$

As before, $\overline{b}_j$ represents the bias and $\overline{w}_j j$ the decoder weight vector. The function *g(c)* introduces here also a nonlinearity. The activation functions *f* and *g* are chosen according to the problem treated, but they are often *tanh* or *sigmoid* functions. The learning of the auto-encoder proceeds by minimization of a criterion depending on the reconstruction error $x - \hat{x}$. In general, cost functions of the least squares or cross-entropy type are used. In some situations, to avoid pathological cases where the auto-encoder performs a simple recopy of the input data (i.e., $W^T \overline{W} = I$ where I is the identity matrix of appropriate size), we impose the constraint $W = \overline{W}^T$.

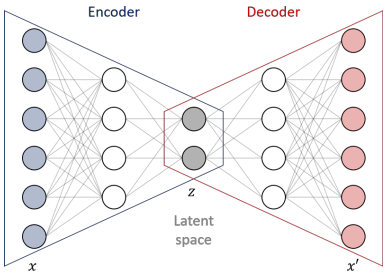


Fig. 1. Example of autoencoder

B. Architecture

The autoencoder is constituted in this way (see figure 2):

- We have in first the encoder which is constituted of two layers of convolution 2D which allows passing from the dimension (28, 28, 1) to (7, 7, 8), this allows us to make a good extraction of featuring.
- We have the part of decoding which is constituted of two layers of convolution 2D Transpose, which makes the reverse of the convolution 2d then a last layer of convolution 2D to put back the image dimension (28, 28, 1).

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 14, 14, 16)        160

 conv2d_4 (Conv2D)           (None, 7, 7, 8)           1160

=================================================================
Total params: 1,320
Trainable params: 1,320
Non-trainable params: 0
_________________________________________________________________
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_transpose_2 (Conv2DT  (None, 14, 14, 8)        584
 ranspose)

 conv2d_transpose_3 (Conv2DT  (None, 28, 28, 16)       1168
 ranspose)

 conv2d_5 (Conv2D)           (None, 28, 28, 1)         145

=================================================================
Total params: 1,897
Trainable params: 1,897
Non-trainable params: 0
_________________________________________________________________
```

Fig. 2. Encoder and decoder architecture

Our networks have the same architecture as our encoder, we then added two more layers to finish them (see figure 3):

- Flatten(), this layer allows flattening our vector.
- Dense(10) to be able to make our classification on our 10 classes.

```
Model: "model_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_14 (InputLayer)       [(None, 28, 28, 1)]       0

 conv2d_36 (Conv2D)          (None, 14, 14, 16)        160

 conv2d_37 (Conv2D)          (None, 7, 7, 8)           1160

 flatten_2 (Flatten)         (None, 392)               0

 dense_5 (Dense)             (None, 10)                3930

=================================================================
Total params: 5,250
Trainable params: 5,250
Non-trainable params: 0
_________________________________________________________________
```

Fig. 3. RN architecture

III. Experimentation

In our study, we want to compare a network with randomly initialized weights against a network with weights already pre-initialized with an autoencoder. To do this, we used the MNIST handwritten digit recognition database. The input images are 28×28 pixels in size. The images have been adjusted in such a way that the pixel values are between 0 and 1. For each image, there are 10 possible labels, corresponding to the digits from 0 to 9, as can be seen in figure (2).

Fig. 4. Example of MNIST

First, we train the autoencoder so that it can learn to deconstruct the image in the dimension we want, the reconstruction of the image allows us to know if it has studied well and modify these parameters if necessary to have a good extraction of data..

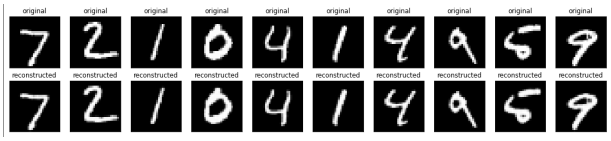


Fig. 5. Reconstruction of the image

Then we load the weights of the encoder in our first network, and we train our two networks. We chose the values 0.05 and 0.01 for the learning rate in an empirical way, it is what gave better results. Here is what it looks like with the value of 0.05:

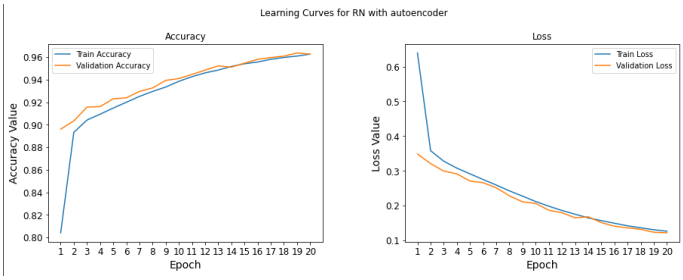


Fig. 6. Learning Curves for RN with autoencoder

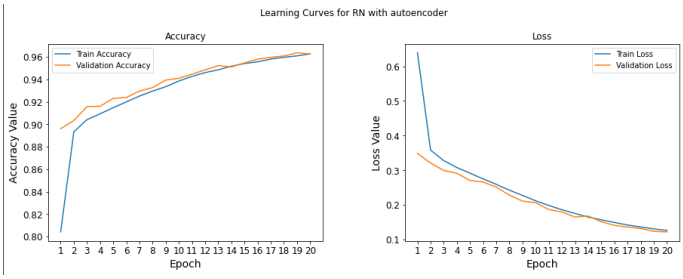


Fig. 7. Learning Curves for RN without autoencoder

Here is what it looks like with the value of 0.01:

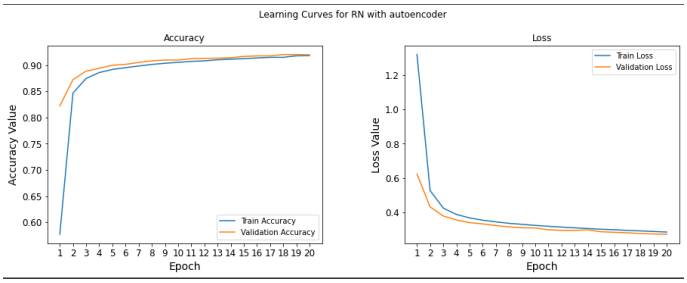


Fig. 8. Learning Curves for RN with autoencoder

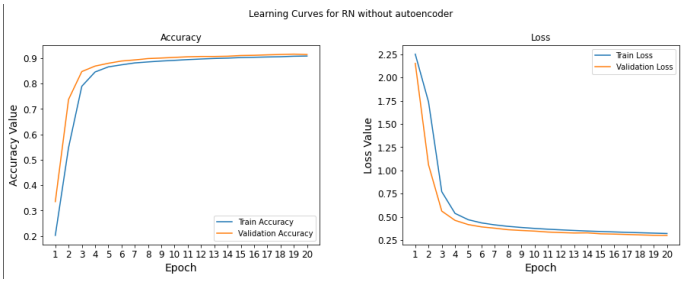


Fig. 9. Learning Curves for RN without autoencoder

IV. Conclusion

In our study, we want to compare a network with randomly initialized weights against a network with weights already pre-initialized with an autoencoder, The network with the weights already pre-trained is slightly more efficient than the other without the training. The difference is minimal, because our network is not very complex, but on much more complex networks, a much bigger difference could appear. We can also see that with a low learning rate, the networks learn faster and make fewer errors earlier.

References

[1] X. Tian1 R. Hérault1 G. Gasso1 S. Canu. Pré-apprentissage supervisé pour les réseaux profonds, 2010.

[2] Dumitru Erhan, Pierre-Antoine Manzagol, Yoshua Bengio, Samy Bengio, and Pascal Vincent. The difficulty of training deep architectures and the effect of unsupervised pre-training. In Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics, volume 5, pages 153–160, 2009.

[3] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, Pierre-Antoine Manzagol; Stacked Denoising Autoencoders: Learning Useful Representationsina Deep Network with a Local Denoising Criterion, 2010

Mosaïque d'images sonars à l'aide d'une méthode de Fourier et d'un graphe d'optimisation

Katell Lagattu
katell.lagattu@ensta-bretagne.org

***Résumé*—L'objectif de ce travail est de constituer une mosaïque d'images prises par un sonar frontal à bord d'un véhicule sous-marin. Chaque prise de vue du robot a une orientation et une position différente, et il s'agit de les assembler. On ne peut pas appliquer les méthodes classiques de traitement d'images, par exemple la méthode des points d'intérêt, à cause du bruit dans les données. On utilise donc des méthodes alternatives comme la transformée de Fourier. Une méthode basée sur les fréquences est pertinente pour surmonter la difficulté liée au bruit. Il s'agit aussi d'utiliser un graphe d'optimisation afin de recaler les images grâce aux rebouclages de trajectoire. Cette approche permet d'obtenir une mosaïque d'images sonars précise, avec une résolution accrue par rapport aux images seules. La position et le cap du drone sous-marin sont aussi estimés. Une approche par intervalles est également appliquée pour le recalage d'images.**

***Index Terms*—Mosaïque d'images sonars, Fourier, graphe d'optimisation, intervalles, recalage**

I. Introduction

La reconstitution d'images sonars est un enjeu majeur pour déterminer avec précision les caractéristiques du fond marin, ce qui est notamment utile dans le contexte de la recherche d'épaves. Réaliser une mosaïque d'images revient à déterminer les transformations successives d'une série d'images se chevauchant, afin de constituer une image globale. Afin de calculer ces transformations, il est nécessaire de trouver des points communs entre ces images. Une méthode classique de recalage d'images consiste à détecter des points d'intérêt et de les faire correspondre entre deux images successives. Cependant les difficultés que l'on rencontre dans le monde de l'imagerie sonar sous-marine sont des bruits, une basse résolution et des occlusions, ce qui entraîne une absence de zones caractéristiques sur les images sonars. L'article sur lequel se base ce projet s'intéresse à une méthode de recalage d'images sonars à l'aide d'une transformée de Fourier [1]. Cette méthode permet de surmonter la difficulté liée à l'absence de points d'intérêt sur les images sonars. Les auteurs de l'article appliquent ensuite une méthode basée sur un graphe d'optimisation pour recaler les images lors des rebouclages de trajectoire. A partir de cette méthode détaillée en troisième partie, les résultats de son implémentation sur une image sonar seront présentés, puis une autre approche qui utilise l'analyse par intervalles sera détaillée.

II. Etat de l'art

La reconstitution d'une mosaïque d'images est nécessaire dans de nombreuses applications, comme former un panorama à partir de plusieurs images se superposant ou former une carte géographique à partir d'images satellites. Cela peut également être utilisé pour localiser un robot doté d'une caméra [2].

Une méthode classique de formation de mosaïques d'images est une méthode basée sur les correspondances entre les points d'intérêt. Il s'agit de faire correspondre des zones caractéristiques entre deux images se chevauchant. De nombreux algorithmes sont proposés dans la littérature [3].

Une approche classique est d'utiliser l'algorithme SIFT (Scale Invariant Feature Transform) pour détecter et décrire ces points d'intérêt [4]. Une mise en correspondance est réalisée, puis l'algorithme de Ransac est utilisé pour déterminer la transfomation entre deux images.

Cependant cette technique n'est pas pertinente pour les images sonars car elles ne contiennent que peu ou pas de points d'intérêt.

Une autre approche consiste à utiliser la globalité des images pour établir des zones caractéristiques à grande échelle [5]. Cela permet d'améliorer le calcul de la transformation entre deux images bruitées, mais nécessite tout de même un minimum de zones caractéristiques distinguables, ce qui n'est pas toujours le cas pour les images sonars.

Hurtos et al. sont les premiers à proposer une méthode de Fourier pour faire correspondre des images 2D d'un sonar frontal [1]. Ils s'intéressent aussi au recalage d'images détaillé en quatrième partie.

III. Méthode de recalage d'images sonars par transformée de Fourier

A. Géométrie des sonars frontaux

Afin de recaler des images sonars, il est nécessaire de déterminer un modèle adapté à la géométrie du sonar [6]. On considère ici un sonar frontal caractérisé par son azimut (θ), son élévation (ϕ) et sa portée (r) (figure 1).

Un point P de coordonnées sphériques (r, θ, ϕ) peut être défini dans le repère du sonar, en coordonnées cartésiennes, par l'équation (1).

$$P = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} rcos(\theta)cos(\phi) \\ rsin(\theta)cos(\phi) \\ rsin(\phi) \end{pmatrix} \quad (1)$$

La projection p de P dans le plan image du sonar est décrite par l'équation (2).

$$p = \begin{pmatrix} x \\ y \end{pmatrix} = \frac{1}{cos(\phi)} \begin{pmatrix} X \\ Y \end{pmatrix} \quad (2)$$

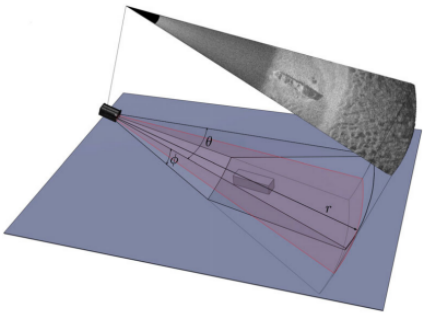


FIGURE 1. Géométrie de l'image sonar [6]

Pour calculer la transformation d'une image par rapport à une autre, il faut déterminer la matrice d'homographie H décrivant la translation et la rotation entre deux points p_1 et p_2. Dans notre cas, seule la rotation θ du plan est prise en compte, ainsi que les translations selon x et y. La matrice d'homographie est donnée par l'équation (3).

$$p_1 = Hp_2 = \begin{pmatrix} cos(\theta) & -sin(\theta) & t_x \\ sin(\theta) & cos(\theta) & t_y \\ 0 & 0 & 1 \end{pmatrix} p_2 \qquad (3)$$

Cette matrice d'homographie permet de recaler les images les unes par rapport aux autres pour une transformation donnée. Pour déterminer cette transformation, une méthode de Fourier est utilisée.

B. Méthode de Fourier

La transformation entre deux images est donc caractérisée par une translation et une rotation. La méthode utilisée ici pour déterminer la translation entre deux images est un algorithme de corrélation de phases [7]. En effet, cette méthode est robuste aux bruits et occlusions présents dans le milieu sous-marin. Pour implémenter cette technique, il est préférable d'appliquer aux images une fenêtre de Hamming pour réduire les effets de bord. Ensuite le passage dans le domaine fréquentiel se fait avec un algorithme de FFT (Fast Fourier Transform) afin de limiter le temps de calcul [1].

Le cross-power spectrum est ensuite calculé [8] avant d'appliquer une transformée de fourier inverse afin d'obtenir la matrice de corrélation de phases. Les pics de plus grandes amplitudes de cette matrice correspondent à la translation entre les deux images. On peut adapter cette méthode pour déterminer la rotation entre deux images en appliquant la même procédure à l'amplitude log-polaire des transformées de Fourier des deux images [9]. Selon les propriétés de la transformée de Fourier, le spectre d'amplitude est invariant à la translation, ce qui permet de déterminer la rotation. Une synthèse de l'ensemble de l'algorithme est présentée figure 2.

A présent il est donc possible de déterminer la transformation entre deux images, puis de l'appliquer concrètement grâce à la matrice d'homographie. La question se pose alors de l'efficacité de cette méthode et de ses conditions d'applications.

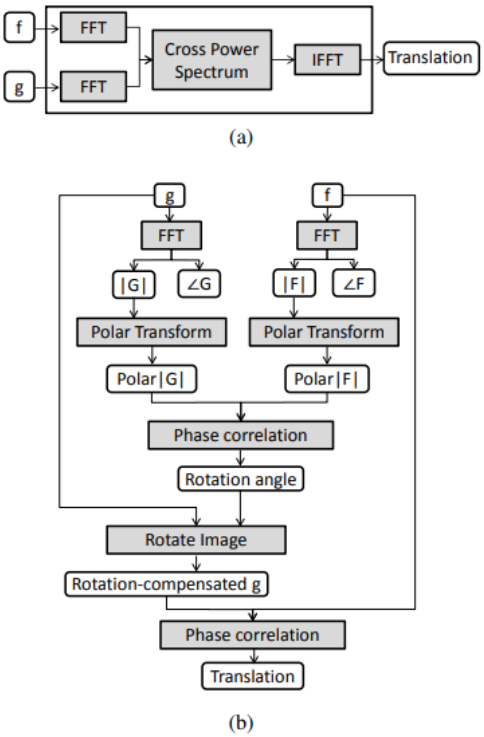


FIGURE 2. (a) Procédure basique pour retrouver la translation entre deux images. (b) Procédure plus élaborée permettant de retrouver la translation et la rotation entre deux images [1]

C. Efficacité de la méthode de corrélation de phases sur des images de sonar frontal

Des effets non désirés comme les occlusions peuvent introduire des pics arbitraires dans la matrice de corrélation de phases, ce qui peut perturber la détermination de la transformation entre deux images [1]. Le phénomène de repliement de spectre peut également altérer la matrice de corrélation de phases.

Une mesure d'efficacité possible de la méthode de corrélation de phases est d'utiliser une trajectoire telle que la première image prise coïncide avec la dernière. Il suffit alors de comparer la transformation de ces deux images avec la tranformation obtenue en appliquant l'algorithme décrit précédemment sur les paires d'images successives.

Une autre manière de mesurer la précision de la méthode est de comparer la trajectoire du drone estimée par les transformations successives d'images avec les données d'une centrale inertielle.

Maintenant que nous savons déterminer les transformations entre deux images, il faut pouvoir utiliser une méthode de recalage efficace pour minimiser les erreurs.

IV. Optimisation d'un graphe de contraintes

A. Utilisation d'un graphe de contraintes

Pour créer une mosaïque d'images sonars, une première approche est d'appliquer l'algorithme de corrélation de phases à chaque paire de deux images successives qui se superposent.

Cependant cette méthode accumule les erreurs, et plus le drone de surface avance et moins la reconstitution des images est précise. L'idée de l'article [1] est d'appliquer l'algorithme de corrélation de phases non seulement entre les paires d'images consécutives mais aussi entre les images non-consécutives. Les rebouclages de trajectoire (c'est-à-dire quand une zone est vue plusieurs fois à des instants différents) vont ainsi être utilisés pour ajouter des contraintes sur les transformations des images.

Pour estimer la meilleure configuration possible des états des images, une optimisation d'un graphe de contraintes est utilisée. L'algorithme Levenberg-Marquardt [10] va être appliqué à la méthode des moindres carrés lors de la phase d'optimisation.

Il s'agit ici d'un post-traitement de données, et non pas d'un traitement des données en temps réel, à cause de la lourdeur des calculs.

B. Définition du graphe

Les sommets du graphe sont définis par les positions et orientations (x_i, y_i, θ_i) des images sonars [1]. Les arêtes sont définies par les contraintes de correspondance entre les deux sommets correspondants. Ainsi, une image vue deux fois aura une contrainte supplémentaire par rapport à une image vue une unique fois. Ces contraintes sont formées de la matrice de transformation z_ij entre les deux images et de la matrice de covariance $\Omega_{z_{ij}}$ associée. La fonction d'erreur correspondante est donnée par l'équation (4) où X_i et X_j sont deux états d'images sonars (translation et rotation).

$$e(X_i, X_j, z_{ij}) = z_{ij} \ominus (X_j \ominus X_i) \tag{4}$$

Où $\ominus$ correspond à l'équation (5).

$$X_j \ominus X_i = \begin{pmatrix} (x_j - x_i)cos(\theta_i) + (y_j - y_i)sin(\theta_i) \\ -(x_j - x_i)sin(\theta_i) + +(y_j - y_i)cos(\theta_i) \\ norm(\theta_i - \theta_j) \end{pmatrix} \tag{5}$$

C. Initialisation du graphe

On peut initialiser le graphe de contrainte de plusieurs manières. Par exemple, on peut appliquer l'algorithme de corrélation de phases à chaque paire d'images successives et initialiser le graphe avec le résultat obtenu. On peut également initialiser le graphe de contraintes à partir des données de navigation du drone sous-marin.

D. Minimisation des contraintes

Il faut maintenant déterminer la configuration v des états de chaque image qui minimise la fonction F de l'équation (6). Cette fonction prend en compte l'ensemble des contraintes C du graphe.

$$F(v) = \sum_{(i,j)\in C} e(X_i, X_j, z_{ij})^T \Omega_{z_{ij}} e(X_i, X_j, z_{ij}) \tag{6}$$

La solution (équation (7)) est donnée par l'algorithme de Levenberg-Marquardt.

$$v = argmin(F(v)) \tag{7}$$

E. Détermination des correspondances pertinentes

Afin de déterminer si une correspondance est pertinente ou non et ainsi éviter les fausses contraintes, on introduit le concept de peak-to-noise ratio. L'amplitude du pic de la matrice de corrélation de phases doit être supérieure à une certaine valeur pour que l'on considère que la correspondance est pertinente.

V. Implémentation de l'algorithme de corrélation de phases avec la transformée de Fourier

A. Choix de l'image

Pour tester l'algorithme décrit dans l'article [1] et détaillé dans la première partie, on peut utiliser une image prise par un sonar latéral (figure 3).

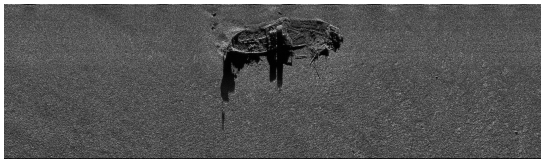


FIGURE 3. Image utilisée pour tester l'algorithme

On peut ensuite découper cette image en huit sous-images se superposant, pour mimer plusieurs prises de vues successives d'un drone sous-marin (figure 4).

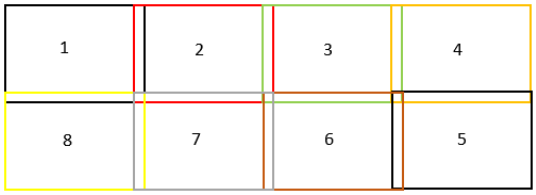


FIGURE 4. Découpage de l'image de l'épave

B. Algorithme de corrélation de phases

Il a ensuite fallu calculer la transformée de Fourier de ces sous-images, puis appliquer l'algorithme de corrélation de phases à chaque paire d'images successives [11], c'est-à-dire sans prise en compte de rebouclages. La mosaïque d'images obtenue est visible figure 5.

On peut noter que le problème est ici simplifié car les images n'ont pas d'orientations différentes, il s'agit uniquement d'un problème de translations.

On peut voir qu'on obtient une image à première vue très fidèle, ce qui s'explique par la bonne résolution des images utilisées. On peut toutefois remarquer de légères déformations dans la mosaïque obtenue. Les transformations relatives données par l'algorithme ont environ 0.12% d'écart avec les transformations réelles. De plus, des rotations relatives ont été détectées, alors qu'il ne s'agit bien que de translations.

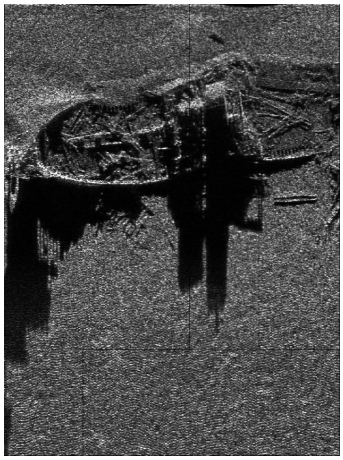


FIGURE 5. Mosaïque d'image obtenue

On peut penser qu'une reconstitution de ces images à l'aide d'un graphe de contraintes, en prenant en compte les rebouclages, peut améliorer le résultat.

C. Comparaison avec l'algorithme SIFT

On peut vouloir comparer les résultats obtenus avec un algorithme classique de détection de points d'intérêt. Les transformations obtenues avec l'algorithme SIFT sont très similaires aux transformations obtenues avec la transformée de Fourier. Cela s'explique par la présence de nombreuses zones caractéristiques dans l'image, et par sa bonne résolution.

VI. OPTIMISATION D'UN GRAPHE DE CONTRAINTES

A. Implémentation d'un exemple d'optimisation de graphe

Avant d'implémenter le graphe de contraintes des sous-images de l'épaves, on peut d'abord tester un premier exemple plus simple.

Dans cet exemple, le graphe est composé de trois sommets modélisés par trois points. Des contraintes de distances entre ces trois points ont ensuite été appliquées [12]. Cela revient à ajouter les arêtes entre chaque sommet (figure 6). Il y a deux contraintes de distances contradictoires : le triangle doit être équilatéral de longueur de côté fixée et deux points du triangle doivent être confondus.

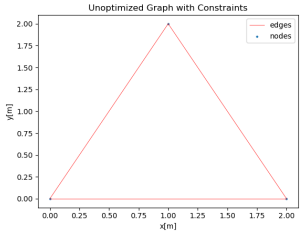


FIGURE 6. Graphe non optimisé avec contraintes

L'algorithme effectue plusieurs itérations avant de converger vers la solution idéale (figure 7).

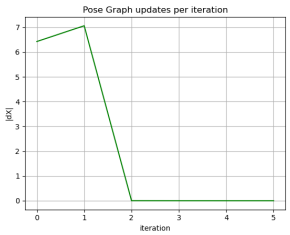


FIGURE 7. Mises à jour du graphe au fur et à mesure des itérations

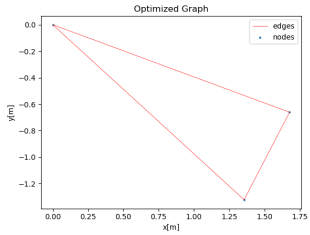


FIGURE 8. Graphe optimisé

Finalement, la configuration optimale des coordonnées des trois points est celle donnée par la figure 8.

On voit dans la figure 8 que la distance entre deux des points a diminué. Il s'agit bien du compromis entre la contrainte du triangle équilatéral et la contrainte des deux points confondus.

B. Implémentation d'un graphe d'optimisation des sous-images de l'épave

A présent, on n'utilise plus seulement un recalage entre deux images successives, mais aussi avec toutes les autres images à condition que la corrélation soit pertinente. Par exemple, la sous-image numéro 2 de la figure 4 ne va pas seulement être recalée par rapport à l'image de référence numéro 1, mais aussi par rapport à l'image 7 qui contient une partie de l'image numéro 2 vue à un moment différent.

On peut voir le graphe optimisé des images de l'épave en 2D (seules les contraintes de translations sont visibles) sur la figure 9.

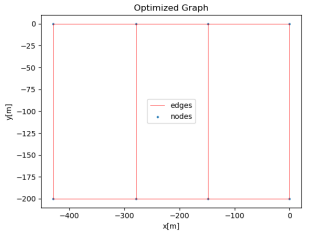


FIGURE 9. Graphe optimisé des images de l'épave

L'image obtenue en reconstituant la mosaïque est visible sur la figure 10.

L'écart avec les translations et les rotations véritables est quasi nul. Le recalage d'images à l'aide d'un graphe d'optimisation a donc amélioré la mosaïque.

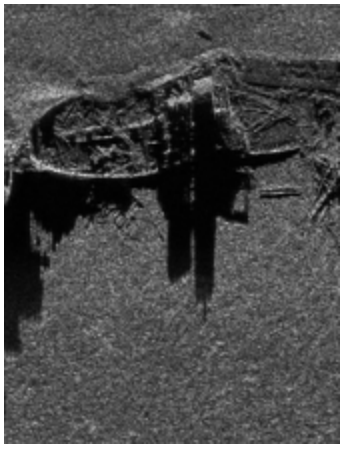


FIGURE 10. Mosaïque avec recalage et optimisation de graphe

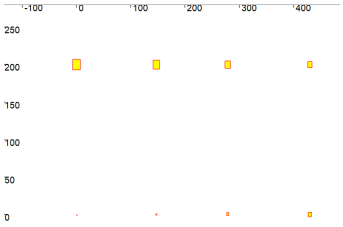


FIGURE 11. Erreurs cumulatives du recalage d'images successives

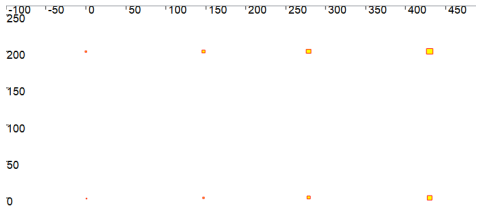


FIGURE 12. Recalage des images après ajouts de contraintes liées au rebouclage

Il est également intéressant de retrouver ces résultats par une approche par intervalles.

VII. APPROCHE PAR INTERVALLES

A. *Définition des boîtes*

Une autre méthode de recalage possible consiste à partir d'un ensemble infini de configurations possibles pour les états des sous-images, puis de supprimer les configurations qui ne satisfont pas les contraintes que l'on ajoute. On peut utiliser pour cela une méthode ensembliste.

Dans cette partie, on peut utiliser l'algorithme de corrélation de phases pour établir les contraintes entre chaque sous-image, puis utiliser une approche par analyse par intervalles au lieu d'une approche par graphe d'optimisation.

Pour cette approche, seuls des vecteurs d'états (x, y) et uniquement des contraintes de translations sont pris en compte. Chaque position des centres des sous-images de l'épave est représentée par des interval-vectors ou boîtes, pour représenter l'incertitude associée [13]. Initialement, toutes les positions relatives des images sont des boîtes infiniment grandes.

B. *Contracteur*

Un contracteur est un opérateur mathématique utilisé pour réduire un ensemble, sans perdre d'informations. Dans le cas de notre projet l'ensemble que l'on veut contracter est l'ensemble des boites définies représentant les positions des sous-images.

Les contraintes ajoutées au contracteur sont des contraintes de distances (dx, dy) entre chaque boîte ou image successive. A chaque contrainte est associé un biais, qui induit une propagation des erreurs d'estimation (figure 11).

Pour éviter de cumuler les erreurs, il faut ajouter les contraintes dûes au rebouclage de la trajectoire du drone sous-marin. On peut voir que le résultat de la contraction donne une estimation plus fine des positions des centres des sous-images (figure 12).

L'approche par intervalles permet donc de déterminer le plus petit ensemble de configurations possibles qui satisfont toutes les contraintes. Cette méthode est pertinente dans le cas de création de mosaïques d'images sonars.

VIII. CONCLUSION

L'implémentation réalisée à partir de la méthode décrite par l'article [1] a confirmé l'efficacité de passer dans le domaine fréquentiel afin de recaler des images se chevauchant. Le graphe d'optimisation prenant en compte les contraintes liées aux rebouclages de trajectoire a permis d'améliorer l'estimation des transformations des images jusqu'à obtenir une erreur quasi nulle. Une approche par intervalles est également pertinente dans la recherche des transformations d'images satisfaisant un ensemble de contraintes. Des méthodes plus fines de réalisation de mosaïques d'images sonars, prenant par exemple en compte les données de navigation dans le graphe de contraintes, peuvent être envisagées.

RÉFÉRENCES

[1] Natalia Hurtos, Xavier Cuf, Yvan Petillot, Joaquim Salvi, "Fourier-based Registrations for Two-Dimensional Forward-Looking Sonar Image Mosaicing " IEEE/RSJ International Conference on Intelligent Robots and Systems October 7-12, 2012.

[2] A.E.S. Lucas, C. Christo, M.P. Silva, C. Cardeira, Mosaic based flexible navigation for AGVs, in : IEEE International Symposium on Industrial Electronics (ISIE), 2009, pp. 3545–3550.

[3] Debabrata Ghosh and Naima Kaabouch, A survey on image mosaicing techniques, Journal of Visual Communication and Image Representation.

[4] Lowe, "Distinctive image features from scale-invariant keypoints," Int. Journal of Computer Vision, vol. 60, no. 2, pp. 91–110, 2004.

[5] H. Johannsson, M. Kaess, B. Englot, F. Hover, and J. Leonard, "Imaging sonar-aided navigation for autonomous underwater harbor surveillance," in Proc. IEEE/RSJ Int. Intelligent Robots and Systems (IROS) Conf, 2010, pp. 4396–4403.

[6] Natalia Hurtos, Xavier Cuf, Yvan Petillot, Joaquim Salvi, David Ribas, 'Fourier-based Registration for Robust Forward-looking Sonar Mosaicing in Low-visibility Underwater Environments," in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York : Academic, 1963, pp. 271–350.

[7] E. De Castro and C. Morandi, “Registration of translated and rotated images using finite fourier transforms,” IEEE Transactions on Pattern Analysis and Machine Intelligence, no. 5, pp. 700–703, 1987.

[8] H. Shekarforoush, M. Berthod and J. Zerubia, "Subpixel image registration by estimating the polyphase decomposition of cross power spectrum," Proceedings CVPR IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1996, pp. 532-537, doi : 10.1109/CVPR.1996.517123.

[9] Hanzhou Liu, Baolong Guo and Zongzhe Feng, "Pseudo-log-polar Fourier transform for image registration," in IEEE Signal Processing Letters, vol. 13, no. 1, pp. 17-20, Jan. 2006, doi : 10.1109/LSP.2005.860549.

[10] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery, Numerical Recipes. Cambridge University Press, 1992

[11] Y. Ri and H. Fujimoto, “Drift-free Motion Estimation from Video Images using Phase Correlation and Linear Optimization,” in The 15th International Workshop on Advanced Motion Control, 2018, p. N/A.

[12] Giorgio Grisetti Rainer Kummerle Cyrill Stachniss Wolfram Burgard, A Tutorial on Graph-Based SLAM, Department of Computer Science, University of Freiburg, 79110 Freiburg, Germany

[13] Jaulin L., Kieffer M., Didrit O., Walter É. (2001) Interval Analysis. In : Applied Interval Analysis. Springer, London.

Python implementation of task partitioning in a robotic swarm to avoid spatial interference

Maxime LEGEAY MAXIME.LEGEAY@ENSTA-BRETAGNE.ORG
student at ENSTA Bretagne

Abstract

This paper introduces a simple implementation of task separation in a swarm of ground robots, in order to verify whether it can reduce spatial interference between the robots. It is a simulation in which the robots have to move between a source and a nest to transport food, in a restricted space. This research implements different ways of distributing tasks in order to optimise the use of space.

1. INTRODUCTION

In swarm robotics, multiple robots collectively solve problems by forming beneficial structures and behaviours similar to those observed in natural systems, such as swarms of bees, birds or fish (4). However, the step towards industrial applications has not yet been successfully taken. The literature is scarce on real swarm applications that apply real swarm algorithms.

Interference is a critical problem limiting the growth of a group: the time each robot spends in task-irrelevant behaviours, such as obstacle avoidance increases as the density of individuals increases. Performance in tasks that that suffer from physical interference can generally be improved by spatial partitioning.

This work implements with Python an example of spatial partitioning with threshold-based approaches (3). The paper has several sections: Section 2 gives the starting assumptions, Section 3 explains the choices made for the model, and Section 4 lists the simulation results and attempts to explain them. Section 5 gives future perspectives.

2. ABOUT TASK PARTITIONING

2.1. The source-nest model

Our robots have to harvest food in a source area and carry it on to the nest area. The more robots that perform this task in parallel, the more food is transported to the nest. However, space is limited and the robots must not collide, so they are equipped with obstacle avoidance sensors so as not to jostle the other robots in the swarm. (3)

2.2. Task division

Given the chosen problem, there are already two distinct tasks assigned to the robots: collecting food and transporting it to the nest. Each robot must perform these two tasks, one after the other. Thus we can create two sub-tasks and divide the space by adding an exchange area between the source and the nest and thus create a relay for our robots. The robots that collect food at the source go to the exchange zone, then give their food to another robot that will go from the exchange zone to the nest. The first robot then turns around and goes back to the source, it has only exploited a part of the map. The threshold comes into play if a robot has to wait too long in the exchange area: if it does not find a partner to give its food to for too long, then it continues and is assigned to the second area. This is a model where robots are single-task robots - each robot can be assigned to one task at a time - multi-robot tasks - each task can be achieved by one or several robots - and finally instantaneous assignment - the tasks are allocated instantaneously and cannot be predicted due to the information each robot has (1).

3. MODEL CHOICES AND IMPLEMENTATION

3.1. Environment

Our environment choice is simple and will be the same for all experiments: we consider a rectangular area with shapes (4m, 1m). The source is placed on the left of the map, with a width of 0.5m and the nest is on the right, with the same width. Robots are represented by a circle of radius 0.07m.

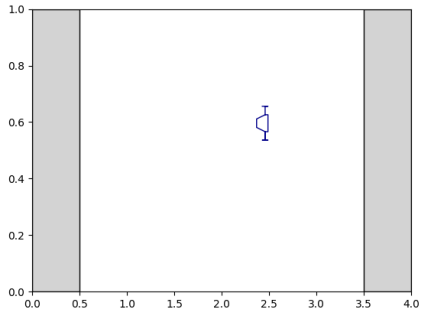


Figure 1: Environment with the source, nest and a robot

At t=0, robots are randomly placed all over the map with random starting orientation and have a 50/50 chance of already carrying food.

3.2. Controller

Each robot has the same intern controller which is simple: if the robot is carrying food, it will always try to go to the right. Otherwise, it will go to the left. A proportional derivative controller in angle is then operated on the robot's heading. The state equation is written simply:

$$\begin{pmatrix} x \\ y \\ \theta \\ v \end{pmatrix} = \begin{pmatrix} v * \cos\theta \\ v * \sin\theta \\ u1 \\ u2 \end{pmatrix}$$

with

$$u1 = Kp * (\theta_0 - \theta) + Kd * \frac{d\theta}{dt}$$

$$u2 = Kp * (v_0 - v)$$

$\theta \in [0; 2\pi]$ $v \in [0; 0.1]$ θ_0 is whether 0 or π depending on which direction they are heading. Moreover, they all obey to a finite state machine representation, to compute where each robot should go and stop. It is described in this figure:

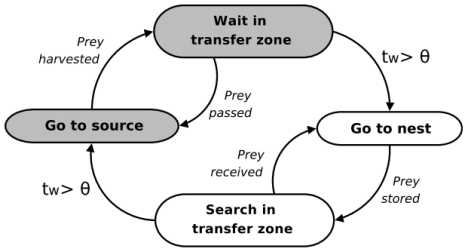


Figure 2: Finite state machine representation simplified, gray represent the harvest tasks and white the store tasks.

If a robot in the harvest state waits too long ($t_w > \Theta$) then it switches to the store state and vice versa.

Finally, robots cannot go through the walls and they cannot get through each other. They have no obstacle detection, but if they meet, they will bounce off each other and each go in an opposite direction.

3.3. Simulation

Food are not represented physically but as an attribute of each robot: if they possess food, they will be displayed in green. Otherwise, they are displayed in blue. Each robot knows the positions of the zones as an input when they are created and know their position at every moment. It allows them to know if they are in a zone or not. If we use an exchange zone, it will be displayed in blue. It is deliberately not perfectly in the middle to force a different distribution of robots than a simple 50/50. There can be more than one exchange zone, in this paper we will study the following cases: no exchange zone, one zone, two zones, peer-to-peer exchange. The size of the map, source, nest and robots never change.

4. APPLICATION AND RESULTS

4.1. Direct transfer

I first decided to check the impact of spatial interference on the amount of food collected in a given time. All the experiments are done with a time $t_max = 200$ and a step $dt = 0.1$. First, I have plotted the amount

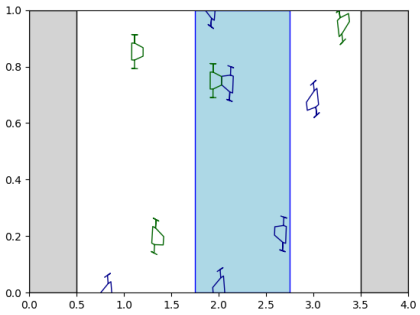


Figure 3: Representation of a simulated environment with 10 robots and one exchange zone.

of food delivered as a function of the number of drones used, all the other parameters were static. It gave this plot:

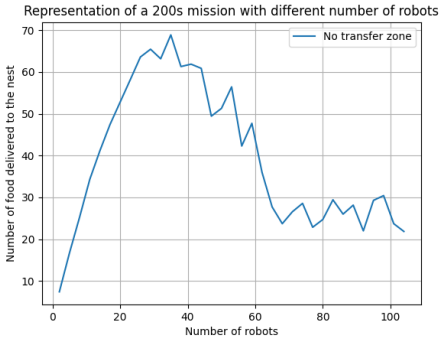


Figure 4: A 200s mission with no transfer zone

But for our case, it is more interesting to plot and compare the efficiency of every drone to evaluate the efficiency of using more robots to do the mission. Thus we define $I_eff = \frac{F}{N}$ with F the food brought by the robots and N the number of robots used. We obtain this graph:

The self efficiency of the robots is decreasing when their number increases. This phenomenon highlights the spatial interference and shows that our robots are "bumping" into each other, reducing their time spent transferring food. (2)

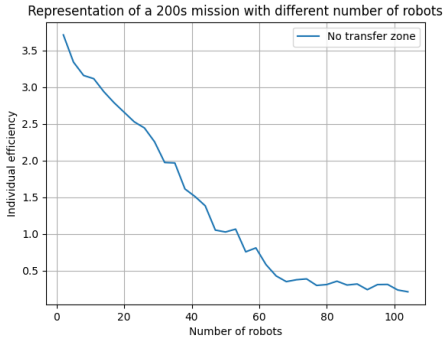


Figure 5: Individual efficiency with no exchange zone

4.2. With one transfer zone

We will now add a transfer zone and compare the two experiences:

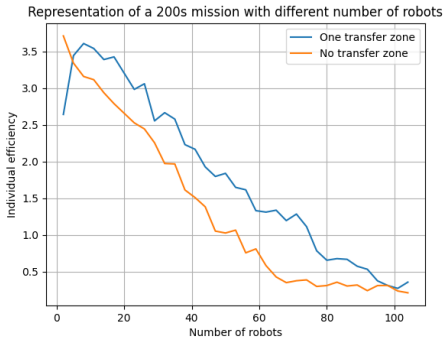


Figure 6: Individual efficiency comparison

The exchange zone seems to have a positive impact on the individual efficiency. Several conclusions can be drawn from this graph. Firstly, the impact is not as consequent as I expected, the exchange zone slightly improved the model but not that much. Indeed, the majority of the interactions between the robots happen inside the zones (nest, source or exchange zone) because they have to maneuver to turn around. Outside the zones, the robots are just moving in a line and are not disturbed that much by spatial interference. Secondly, with no exchange zone, the individual efficiency only decreases, whereas with an exchange zone it increases at the beginning, then decreases. This is probably due to the waiting time

in the exchange zone: we could compute the wasted time as follow $T_{wasted} = T_{interference} + T_{waiting}$. With no zone, $T_{waiting} = 0$. But with a zone, if there are not enough robots, they will waste time waiting for an exchange to occur. T_{wasted} seems to decrease when the number of robot increases.

4.3. With two transfer zones

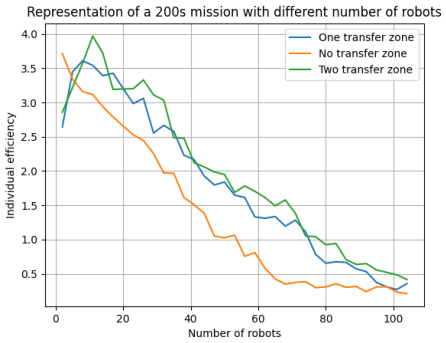


Figure 7: Individual efficiency comparison

With a second exchange zone we can highlight this phenomenon of waiting time even better.

4.4. Peer-to-peer transfer

Now we don't consider any zone but any robot meeting another will give it its food, and both will return in the opposite direction. This could allow to reduce sensible areas such as the transfer zone.

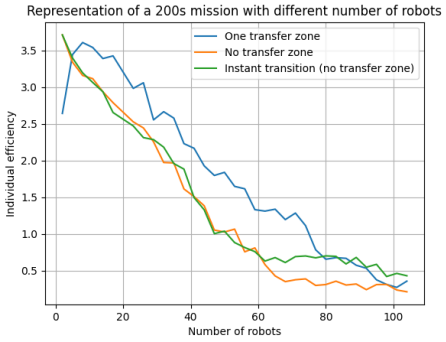


Figure 8: Individual efficiency comparison

This solution seems very close to the direct source to nest solution, except with a higher number of drones. It has the advantage of making the waiting time disappear, but it does not slit the available space for the robots. As a result, spatial interference stay very high and disturbs our robots. With a higher number of robots, peer-to-peer transfer makes the robots cover a minimum distance, as they turn around very frequently.

5. CONCLUSION AND PERSPECTIVES

Interference can be a problem when working with swarms of robots. Partitioning space can be a solution to reduce the time wasted in collisions but it is not as efficient as expected and there are many situations where partitioning would not be the best solution. We used a simple task distribution state machine to allow tasks but it needs a model much more elaborate to show convincing results. Nonetheless, we achieved to show slightly better results with an easy implementation so it gives good prospects for the future. Perspectives now could be an adaptive controller that could decide whether it is useless to partition the tasks or not during a mission. It could be based on anthills functioning for instance, with paths with high average speed that would be good to follow for the robots, and paths with reduced speed where is would be better to pass on the food and go back to the source.

References

[1] Brian P. Gerkey and Maja J. Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research*, 23(9):939–954, 2004.

[2] Mehrdad Jangjou and Mohammad Mansour Riahi Kashani Koosha Sadeghi Oskooyee Alireza Bagheri. An extended simulation of complex task partitioning method in aself-organized robotics swarm. *International journal of new computer architectures and their applications*, 1:714–720, 2011.

[3] G Pini, A Brutschy, Mauro Birattari, and Marco Dorigo. *Interference Reduction Through Task Partitioning in a Robotic Swarm*, pages 52–59. 01 2009.

[4] Melanie Schranz, Martina Umlauft, Micha Sende, and Wilfried Elmenreich. Swarm robotic behaviors and current applications. *Frontiers in Robotics and AI*, 7, 2020.

Decentralized Multirobot SLAM

Badr MOUTALIB

***Abstract*—This article presents a comparison of different decentralized multi-robot simultaneous localization and mapping (SLAM) approaches. The issue of decentralized SLAM revolves around bandwidth limitations and thus necessitates the development of novel approaches to the classical SLAM problem. We showcase the advantages surrounding the decentralized approach and present different solutions to the inherent challenges and constraints faced by decentralized multi-robot SLAM.**

I. Introduction

Certain scenarios like search and rescue or emergency situations that require rapid deployment and execution benefit greatly from multi-robot SLAM systems. Different approaches have been undertaken to create a multi-robot SLAM by relying on a ***centralized*** system to compute a map of the environment. However, such system is vulnerable to damage to the central unit and does not provide the redundancy and robustness necessary for the kind of scenarios where it is deployed. Moreover, these centralized systems require that robots operate around the central unit which vastly limits the speed and the range of possible actions.

The use of a ***decentralized*** multi-robot SLAM system can allow us to overcome these issues. But this approach poses a new set of challenges that stem from bandwidth limitations and the need for autonomous continuous operation requirements of each robot outside of the swarm.

II. Related work

Decentralized SLAM systems is a new and active domain of research. Most of the work in the field deals with decentralized optimization [1, 2] and decentralized data association [3].

A. DDF_SLAM

The work done by Cunnigham et al. [1] relies on using a graphical model approach to augment *Smoothing and Mapping* (SAM) [4] with a *Constrained Factor Graph*. This takes advantage of the sparse matrices involved in solving *Maximum A Posteriori* (MAP) inference in nonlinear SLAM [5]. The work presented only deals with inference system of SLAM and assumes a reliable data association front-end. Such an approach aims at satisfying the following DDF requirements:

1) Scalable computation cost.
2) Scalability with bandwidth limitations
3) Robustness to node failure.
4) Robustness to changes in network topology.

Each robot observes common landmarks and generates a local factor graph that represents the local map of that robot as well as serve to generate a neighborhood graph from the shared graphs of other robots. To achieve this, each robot in the system contains :

- A local optimization module that generates a local map and a condensed form of the local graph with *Constrained Factor Graphs* (CFG) to be shared "Fig 1a". The dark blue circles represent the consecutive robot position while the light blue ones represent the landmarks observed. The dots between the vertices are the factors that represent the factored functions linking between the two ends.
- A communication module to cache and share the condensed graph "Fig 1c". It keeps track of other robots latest shared graph even when they are outside of communication range.
- A neighborhood optimizer module to merge condensed graphs into a neighborhood graph "Fig 1b".

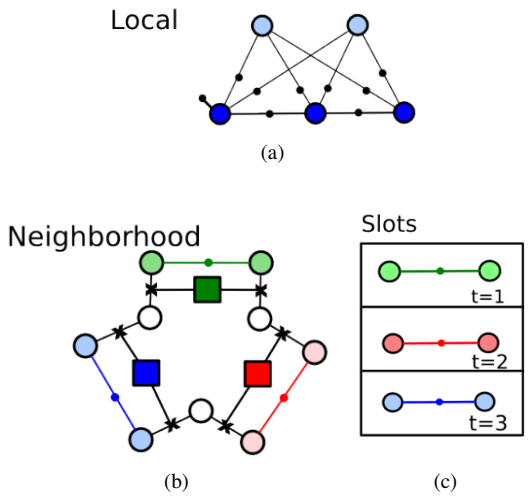


Fig. 1. (a) local optimization module (b) neighborhood optimizer module (c) communication module

This system shows good performance compared to a naive approach where every robot sends every sensor measurement to every other robot.

B. Data-Efficient Decentralized Visual SLAM

In the paper published by Titus Cieslewski et al. [3] the authors, through the use of visual SLAM on a multi-robot system, try to solve both the optimization back-end part of SLAM as well as the place recognition and Pose estimation front-end in a decentralized manner.

For *decentralized optimization* back-end part of SLAM, the Gauss-Seidel approach [6] is favored over the Gaussian elimination seen in DDF-SLAM [1] since it avoids the complex linearization and book keeping involved in the Gaussian elimination approach.

On the other hand, for the *decentralized Place Recognition and Pose Estimation* front-end part of SLAM, the indirect

method for measurement is more suited since it does not necessitate the other robot to be in direct line of vision and requires less hardware either for inter-robot measurement or intra-robot measurement. The downside of this approach is the bandwidth limitation required for the communication between the robots unlike direct measurement where each robot infers the other robots position through direct observation. This will require an optimization of the data transfer.

The method used in this paper consists on generating through an image of the scene a compact image representation with *NetVLAD* [7], and then feeding it into a *decentralized visual place recognition* (DVPR) module which produces candidate place recognition matches. These candidate matches are then fed into *RelPose* module that will generate the inter-robot pose measurement P_S. On the other hand we feed the same image into a visual odometry that will return the intra-robot pose measurement P_I. These measurements are used to update the map directly and indirectly through the decentralized optimization module. The requirements on the visual odometry are:

1) *Produce a pose graph*: given that the robot A captures the frames $F_A = \{(A,1),(A,2),(A,3),..\}$ the VO needs to provide the measurements :

$$P_I = \{Z_{A_i,A_{i+1}} | \forall (A,i),(A,i+1) \in F_A\} \quad (1)$$

$$Z_{A_i,A_{i+1}} = \{R_{A_i,A_{i+1}}, t_{A_i,A_{i+1}}\}s \quad (2)$$

2) *Pose association for every image used in place recognition*
3) *Absolute scale*

For this the author used the ORB-SLAM.

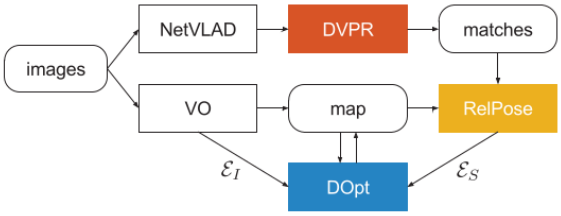


Fig. 2. Experimental Waveform Example

To accommodate for the bandwidth limitations on communication, the author proposes the following solutions:

1) The full-image descriptor is only sent to one robot and the selection for that robot is done using the NetVLAD
2) Decentralized optimisation similar to what was proposed in the DDF-SLAM but using a different method.

which returns a vector v_{A_i} from the image I_{A_i} such that $\|v_{A_i} - v_{X_k}\|_{l_2}$ is lower if I_{A_i} and I_{X_k} represent the same scene. Thus we pick the I_{B_j} that had the shortest l_2 to our image and is under a certain threshold

$$\|v_{A_i} - v_{B_j}\|_{l_2} < \tau_{NetVLAD} \quad (3)$$

Once (B,j) is established A sends a pose estimation request to B

To test the system, the author used the KITTI dataset. The results show that a good global map can be genrated by only exchanging 2mb.

III. Conclusions

We have reviewed some of the latest developments in decentralized multi-robot SLAM which is still a growing topic. Decentralized multi-robot SLAM presents a lot of advantages but comes with challenging difficulties mainly surrounding communication constraints. The solutions that are proposed to overcome these limitations tackle decentralized optimisation like what is proposed in DDF-SLAM [1] or the Gauss-Seidel approach [6], or reducing the scene descriptors in the front-end part of SLAM. This part benefits greatly from the improvements realized in deep learning and Convolutional Neural Networks.

References

[1] A. Cunningham, M. Paluri, and F. Dellaert, "Ddf-sam: Fully distributed slam using constrained factor graphs," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010, pp. 3025–3030.

[2] A. Cunningham, V. Indelman, and F. Dellaert, "Ddf-sam 2.0: Consistent distributed smoothing and mapping," in *2013 IEEE International Conference on Robotics and Automation*, 2013, pp. 5220–5227.

[3] T. Cieslewski, S. Choudhary, and D. Scaramuzza, "Data-efficient decentralized visual slam," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 2466–2473.

[4] F. Dellaert and M. Kaess, "Square root sam: Simultaneous localization and mapping via square root information smoothing," *The International Journal of Robotics Research*, vol. 25, no. 12, pp. 1181–1203, 2006. [Online]. Available: https://doi.org/10.1177/0278364906072768

[5] N. Carlevaris-Bianco and R. M. Eustice, "Generic factor-based node marginalization and edge sparsification for pose-graph slam," in *2013 IEEE International Conference on Robotics and Automation*, 2013, pp. 5748–5755.

[6] S. Choudhary, L. Carlone, C. Nieto, J. Rogers, H. I. Christensen, and F. Dellaert, "Distributed trajectory estimation with privacy and communication constraints: A two-stage distributed gauss-seidel approach," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 5261–5268.

[7] R. Arandjelović, P. Gronat, A. Torii, T. Pajdla, and J. Sivic, "Netvlad: Cnn architecture for weakly supervised place recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 6, pp. 1437–1451, 2018.

ENSTA Bretagne

Troisième année Filière Ingénieur sous Statut Élève (FISE 2022)
Spécialité Robotique Mobile

Rapport d'initiation à la recherche

Vers une localisation autonome des drones par le biais de l'enregistrement d'images aériennes

---

Travail réalisé par

NGNEPIEPAYE WEMBE Stephane

Sous la direction de

Luc JAULIN - Enseignant chercheur

A Brest (29)

15 février 2022

Table des matières

Table des figures

1 Introduction

Les systèmes de navigation basés uniquement sur des capteurs proprioceptifs développent des erreurs qui s'amplifient avec le temps. C'est la raison pour laquelle pour des longues missions de navigation nous avons tendance à utiliser des capteurs extéroceptifs tels que les solutions de GNSS pour borner l'erreur [1].

Les solutions de GNSS marchent bien et permettent de borner les erreurs inertiels. Mais nous devons noter que les solutions de GNSS ont une précision de l'ordre de 1 à 3 m, ce qui peut être très grand pour certains contexte de localisation (comme la localisation indoor, la localisation en milieu urbain).

Dès méthodes de localisation plus précises sont développées telles que le SLAM (Simultaneous Localisation And Mapping), l'odométrie visuelle ou encore le tracking vidéo [2]. Ces méthodes combinées à d'autre capteurs, permettent de donner la position du robot relativement à une scène. Quelque soit la méthode utilisée, le fait de fusionner l'information de position avec les données d'un capteur externe permet d'améliorer la précision.

Dans l'article principal de ce travail, il s'agit d'implémenter un système de localisation instantané basé sur la comparaison des images aériennes d'un UAV [3]. pour une validation du système, le scénario est le suivant : mettre en œuvre un système de traitement d'images en ligne qui peut fournir automatiquement des commandes de pilotage d'un drone en le conduisant dans une scène dont l'emplacement est connu, en comparant les images vidéo aériennes embarquées avec une image géoréférencée.

Le challenge principal est donc d'implémenter un algorithme de traitement d'image qui pourra comparer les deux images et déterminer la matrice de transformation entre les deux images. ce qui permettra de déduire la commande à donner à l'UAV.

Nous allons tout d'abord vous présenter l'implémentation qui a été faite dans l'article, ensuite nous commenterons les résultats obtenus et nous finirons par des propositions d'améliorations.

2 Présentation des travaux de l'article

Dans l'article, il s'agit de diriger un UAV vers une position géoréférencé, en utilisant uniquement des images aériennes prise par celui ci. Le principe est le suivant : L'UAV prend une image aérienne, la compare à l'image de référence, une matrice de transformation entre les deux images est générée et l'erreur de similitude est calculée ; si aucune similitude n'est observée (erreur très élevée) , l'UAV utilisera son système de navigation inertielle pour calculer la commande à appliquer pour le rapprocher de la zone souhaitée c'est le mode search (recherche de similitude), par contre s'il existe un matching entre les deux images (erreur acceptable), la commande de contrôle est calculée à partie de la matrice de transformation entre les deux images c'est le mode localisation (figure 2.1).

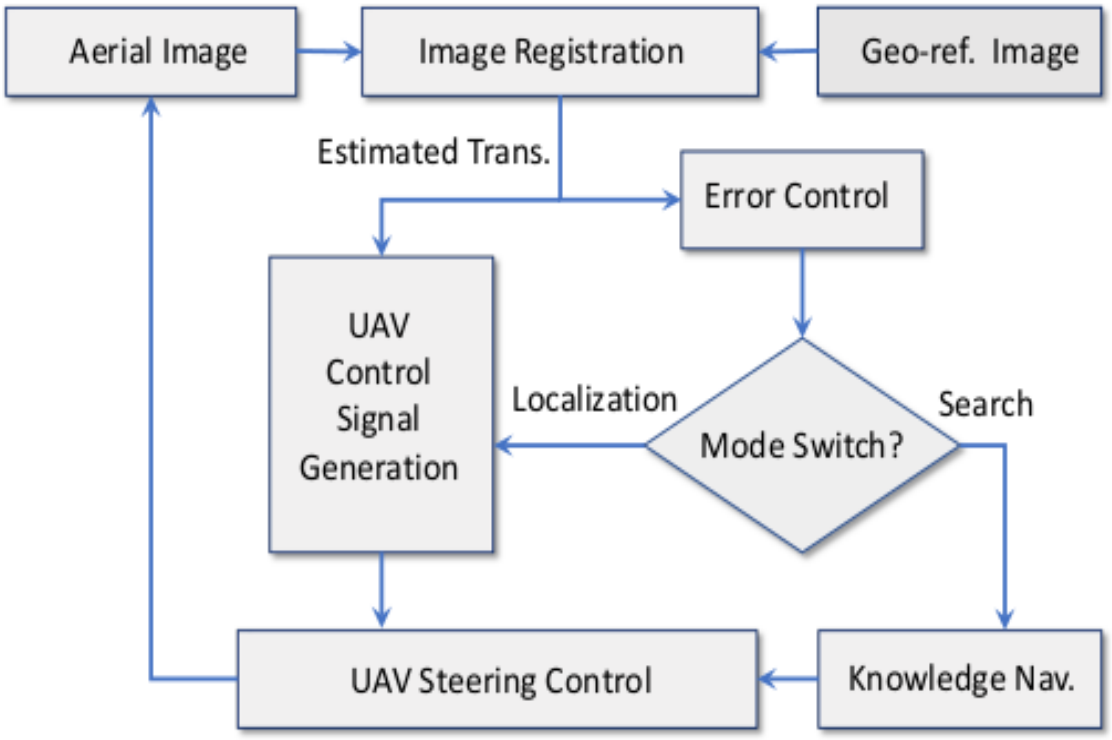


FIGURE 2.1 – Architecture de fonctionnement

2.1 Enregistrement automatique des images et comparaison avec l'image de géoréférencée

Ici l'hypothèse selon laquelle la caméra de l'UAV est stable lors de la prise des photos est faite. On suppose ainsi qu'un parallélisme est préservée dans les différentes photos prises. Ceci permet de dire qu'il existe une transformation linéaire entre l'image prise et l'image géoréférencée.

soit I_r l'image géoréférencée et I_a l'image aérienne. Prenons un point $[x_r, y_r]'$ de l'image géoréférencée et $[x_a, y_a]'$ son correspondant dans l'image aérienne, nous avons la relation suivante :

$$\begin{bmatrix} x_r \\ y_r \end{bmatrix} = \alpha \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_a \\ y_a \end{bmatrix} + \begin{bmatrix} b_x \\ b_y \end{bmatrix} \tag{2.1}$$

Où α est le coefficient d'échelle entre les deux images
θ l'angle de rotation et
$\begin{bmatrix} b_x \\ b_y \end{bmatrix}$ le vecteur de déplacement.

Pour une image donnée, la transformation est déterminée par 4 paramètres : α, θ, b_x, b_y . Donc l'algorithme doit pouvoir :

- Déterminer s'il existe une zone commune entre les deux images.
- Si elle existe, on détermine la transformation géométrique pour faire la correspondance

Pour cette mission, l'algorithme SURF [4] est utilisé. Car cette étape de détermination de la transformation est la plus importante et a donc besoin d'algorithme robuste.

2.1.1 Correspondance de points caractéristiques basée sur la méthode SURF

Le détecteur de points caractéristiques SURF adopte une mesure de matrice hessienne où le Laplacien de la gaussienne est approximée par la différence de la gaussienne. l'algorithme SURF a l'avantage qu'il est invariant par changement d'échelle et d'orientation des images traitées. La figure 2.2 vous montre comment il est implémenté dans ce contexte.

2.1.2 Estimation de la matrice de transformation géométrique

L'équation (2.1) peut s'écrire sous forme généralisée :

$$\begin{bmatrix} x_r \\ y_r \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha\cos\theta & -\alpha\sin\theta & b_x \\ \alpha\sin\theta & \alpha\cos\theta & b_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_a \\ y_a \\ 1 \end{bmatrix} \tag{2.2}$$

La matrice de transformation est $T = \begin{bmatrix} \alpha\cos\theta & -\alpha\sin\theta & b_x \\ \alpha\sin\theta & \alpha\cos\theta & b_y \\ 0 & 0 & 1 \end{bmatrix}$.

L'algorithme SURF permet de déterminer une approximation $\hat{T}$ de T. Ceci permet la détermination des paramètres α, θ, b_x, b_y. L'objectif du contrôleur est donc de faire tendre cette matrice vers la matrice identité.

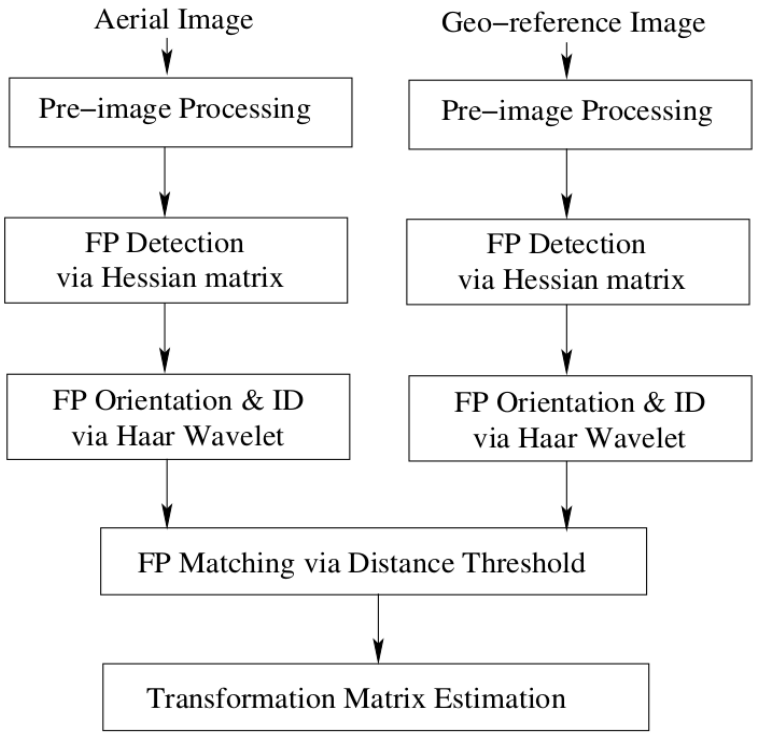


FIGURE 2.2 – Fonctionnement de l'algorithme de correspondance par SURF

2.2 Contrôle de l'UAV

2.2.1 Principe du contrôle en mode localisation

Pour contrôler l'UAV, nous avons accès à 4 mouvements figure 2.3 :

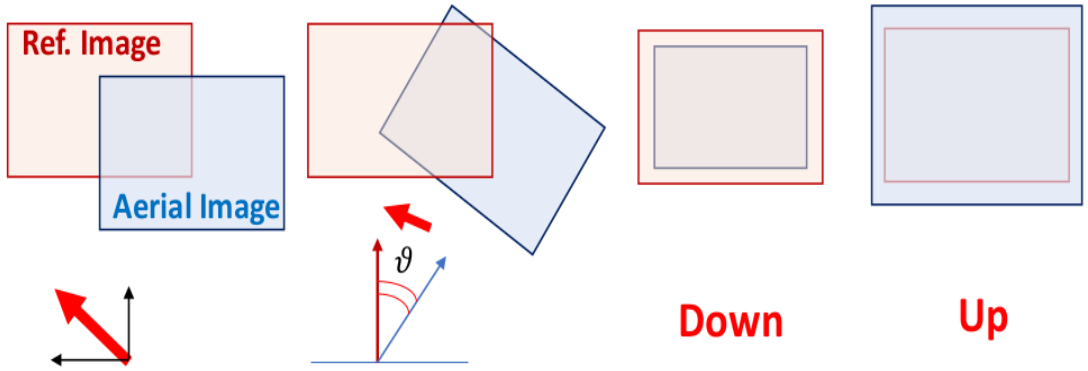
- La translation suivant un vecteur donné dans le plan horizontal
- La rotation d'un angle donné
- La descente et
- La montée.

L'angle de rotation est donné par :

$$\theta = \arctan\left(\frac{\hat{T}(1,2)}{\frac{1}{2}(\hat{T}(1,1)+\hat{T}(2,2))}\right) \tag{2.3}$$

Et le coefficient d'échelle est :

$$\alpha = \frac{\hat{T}(1,1)+\hat{T}(2,2)}{2\cos\theta} \tag{2.4}$$



Parameter	Value	UAV Control
θ	> 0	Turn counterclockwise
	< 0	Turn clockwise
a	> 1	Up
	< 1	Down
b_x	> 0	Left
	< 0	Right
b_y	> 0	Forward
	< 0	Backward

FIGURE 2.3 – Mouvements et commandes de contrôle possible

2.2.2 Évaluation du contrôleur

Pour évaluer le contrôleur, et vérifier la convergence de l'UAV vers la position souhaitée, deux paramètres ont été définis :

- Le déplacement normalisé Cb : c'est le coût du contrôleur. Si le contrôleur est bien implémenté, cette valeur doit décroître avec le temps : preuve que l' UAV se rapproche de sa cible.

$$Cb = \sqrt{(\frac{b_x}{x_l})^2 + (\frac{b_y}{y_l})^2} \tag{2.5}$$

  Où $x_l \times y_l$ est la taille de l'image géoréférencée.

- Le déterminant de la matrice de transformation Cd :

$$Cd = |\hat{T}| \tag{2.6}$$

2.3 Présentation des résultats

Pour l'expérimentation, l'auteur de l'article utilise un simulateur. qui génère des images aériennes. Ces images sont ensuite comparées à une image de référence choisie au départ. Ne pouvant pas reproduire cette simulation faute de logiciel, nous allons tout simplement analyser les résultats. Sur la figure 2.4 nous avons le résultat de l'algorithme surf La figure 2.6 montre l'évolution du contrôle de l'UAV. Et la figure 2.5 montre comment les coûts de contrôle converge vers 0, ce qui confirme que l'UAV converge vers sa zone cible.

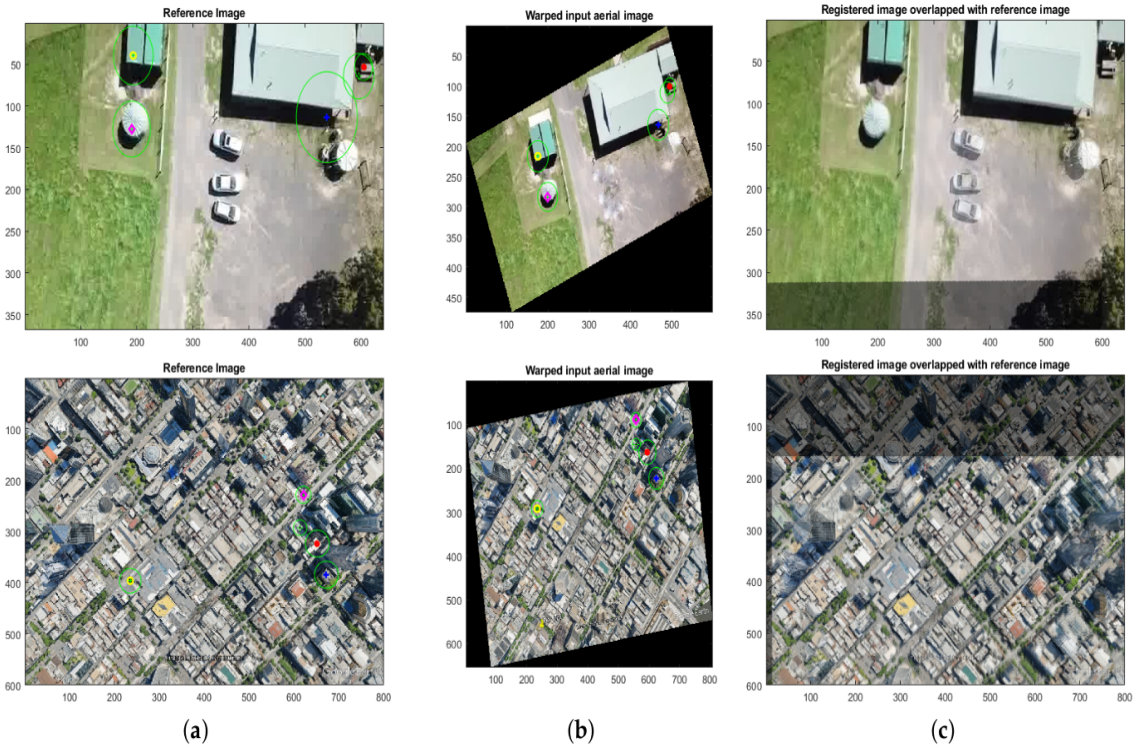


FIGURE 2.4 – Correspondance par SURF : dans la colonne (a) nous avons les images de références, sur la colonne (b) l'image aérienne qui contient des zones communes avec l'image de référence, la colonne (c) est superposition de l'image aérienne sur l'image de référence.

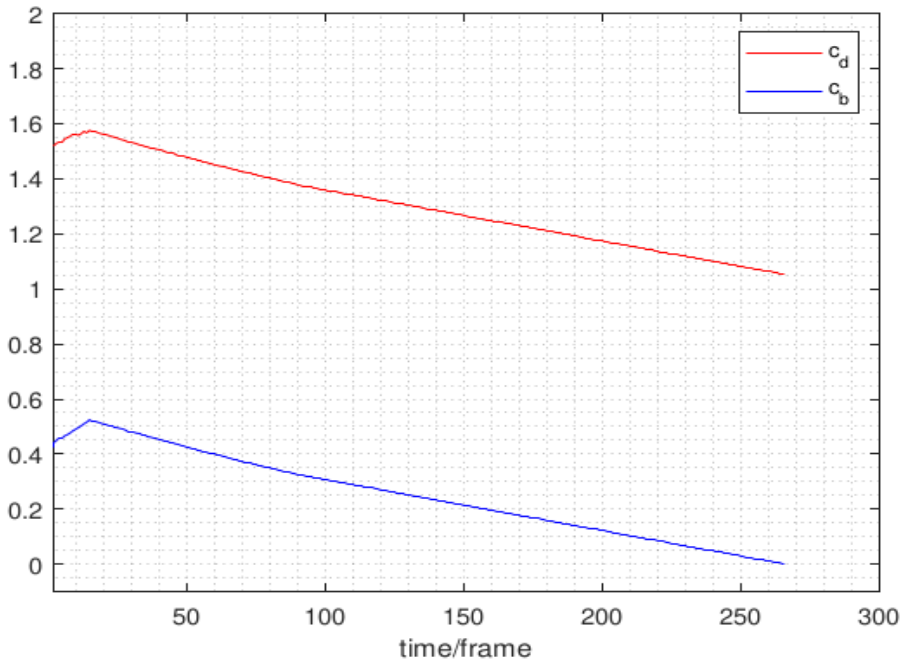


FIGURE 2.5 – Évolution du coût de contrôle

FIGURE 2.6 – Correspondance par SURF : dans la colonne (a) nous avons les images aériennes dans la colonne (b) nous avons superpositions de l'image aérienne sur l'image de référence : la flèche rouge est la décision de contrôle choisie.

3 Limites et améliorations

L'une des limites de la solution du mode localisation est la fiabilité des images passées à l'algorithme SURF. En effet l'image de référence est prise à un moment donné de la journée. Cependant les images aériennes ne sont pas forcément prises dans les mêmes conditions météorologiques. La présence du soleil peut générer des ombres des immeubles qui vont alors polluer l'image. Ceci pose un problème car l'algorithme peut signaler une différence d'image rien que du fait de la présence des ombres. figure 3.1

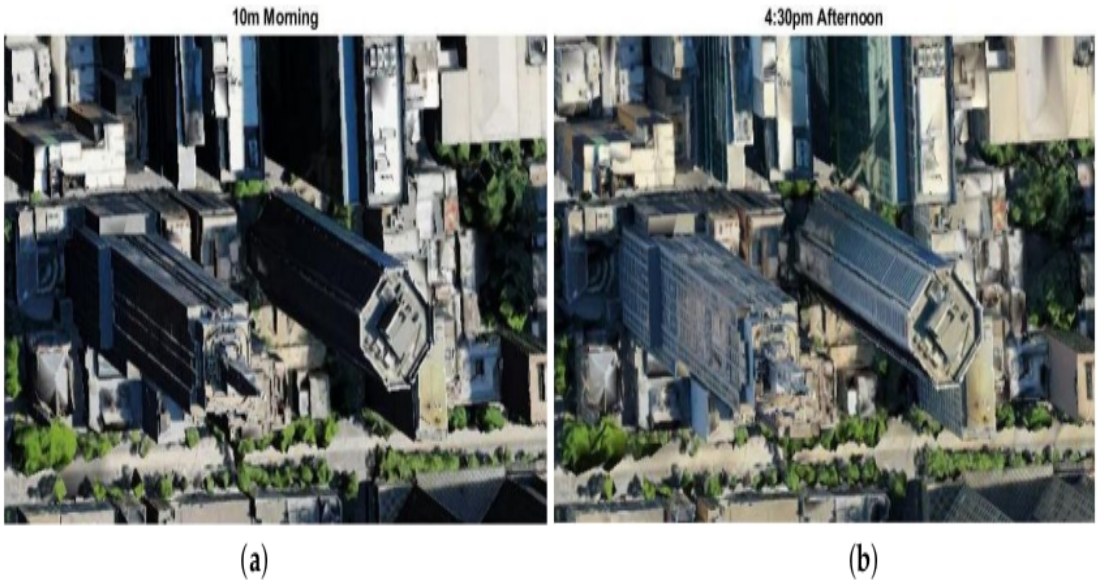


FIGURE 3.1 – (a) Image aérienne prise à 10 heure du matin polluée par la présence d'ombre , (b) Image aérienne 16h30 sans ombre

Une solution serait de réaliser les expériences dans des conditions météorologique similaire. Et pour rendre la méthode plus robuste, nous pourrions penser à un algorithme de filtrage d'ombre [5].

4 Conclusion

Ce document nous a permis de présenter une méthode de localisation basée sur le traitement comparatif d'images aériennes. Cette méthode nous a permis de contrôler un UAV pour qu'il rejoigne une zone prédéfinies. Grâce à la méthode SURF, nous comparons l'image instantanée aérienne avec l'image aérienne géoréférencée de la zone à atteindre. Cet algorithme nous permet d'obtenir une matrice de transformation géométrique sur laquelle nous nous appuyons pour contrôler l'UAV. Toute fois des limites ont été soulevées, dont la plus importante est l'influence de la météo sur les images aériennes. Il s'agit de la pollution des images aériennes par l'ombre des immeubles en milieu urbain. Pour y remédier nous proposons de joindre à ce travail un algorithme de filtrage d'ombres.

Bibliographie

[1] A. Budiyono, "Principles of GNSS, Inertial, and Multi-sensor Integrated Navigation Systems," *Industrial Robot : An International Journal*, vol. 39, p. ir.2012.04939caa.011, Apr. 2012.

[2] R. Munguía, I. Urzua, Y. Bolea, and A. Grau, "Vision-based slam system for unmanned aerial vehicles," *Sensors*, vol. 16, p. 372, 03 2016.

[3] X. Wang, A. Kealy, W. Li, B. Jelfs, C. Gilliam, S. May, and B. Moran, "Toward autonomous uav localization via aerial image registration," *Electronics*, vol. 10, p. 435, 02 2021.

[4] H. Bay, T. Tuytelaars, and L. Van Gool, "SURF : Speeded Up Robust Features," in *Computer Vision – ECCV 2006* (A. Leonardis, H. Bischof, and A. Pinz, eds.), vol. 3951, pp. 404–417, Berlin, Heidelberg : Springer Berlin Heidelberg, 2006. Series Title : Lecture Notes in Computer Science.

[5] D. Boldo, "B.2 Fabrication de vraies ortho-images et correction des ombres," p. 12.

Auto calibrage d'une camera

***Abstract*—L'auto calibration des caméras est l'une des principales problématiques rencontrée en computer vision. Cette technique de calibrage permet de réaliser une calibration, juste en ce servant des images provenant de différentes prises de vu d'une scène. Depuis, près de 10 ans de nombreuses études ont été faite sur cette problématique. Cet article présente les différentes approches de résolution ainsi que l'implémentation de détection de points d'intéret en python ainsi que les différents problèmes rencontrés.**

***Index Terms*—auto calibration, Kruppa équations, paramètres intrinsèques, SIFT, ORB**

I. Introduction

En vision par ordinateur, avant tout utilisation de la caméra une phase de calibration est souvent souhaitée. Cependant, une méthode traditionnelle consiste à faire recours à une mire 2D dont on connaît les dimensions a priori, puis de prendre des images de celle-ci et d'en tirer les paramètres de la caméra. Cette calibration permet donc d'avoir une bonne estimation des distances. En effet, les avantages de cette méthode sont donc évidentes, faciles à utiliser, et ne nécessitent pas l'utilisation d'une mire. Néanmoins de nombreux problèmes se posent, notamment le choix des bonnes images pour l'auto-calibrage. Afin d'avoir une vu d'ensemble sur les travaux qui ont été faits il y a plus une décennie, l'on s'appuiera sur les travaux d'Elsayed E. Hemayed en [2].

II. Auto calibrage des caméras

Avant de bien comprendre comment fonctionne l'auto-calibrage d'une caméra il est important de comprendre comment est-ce qu'une caméra fonctionne. À cet effet, la modèle mathématique qui s'en rapproche le plus est le modèle de Pin Hole. Il permet donc de trouver la projection d'un objet 3D vu dans le monde à son image 2D par la caméra.

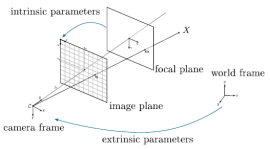


Fig. 1. Pinhole modèle

Ce modèle prend en compte les distorsions géométriques ou le flou des objets non focalisés causés par les lentilles et les ouvertures de taille finie. Très utilisé dans la littérature, ce modèle permet d'avoir des résultats qui se rapproche de la réalité. La formule résumant ce modèle est donnée par :

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} f & 0 & Tu \\ 0 & f & Tv \\ 0 & 0 & 1 \end{pmatrix} + \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} (1)$$

En prenant en compte la dimension d'un pixel on arrive à la représentation suivante :

$u = m_u \frac{fX}{Z} + m_u t_u$ et $v = m_v \frac{fY}{Z} + m_v t_v$

Ainsi, on arrive à isoler la matrice des paramètres K qui nous intéresse pour la calibration.

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} \alpha_x & s & u_o \\ 0 & \alpha_y & v_o \\ 0 & 0 & 1 \end{pmatrix} \text{ P = KP (2)}$$

où (u_o, v_o) sont les coordonnées du point principal, α_x et α_y sont les facteurs d'echelle , et s est l'angle entre l'image et le point principal. Auto calibrer une caméra consite donc à determiner la matrice K à partir d'au moins 3 vues. On expliquera pourquoi juste 3. De nombreuses méthodes ont été développé à savoir : la résolution des équations de Kruppa(Faugeras et. al., 1992), l'application de contraintes linéaires sur la matrice de calibration(Hartley, 1994), une méthode qui détermine la quadrique absolue, qui est l'image du cône sur un plan à l'infini.

III. Les équations de Kruppa

Après avoir pris connaissance des différentes méthodes développées en [4], notre intérêt s'est posé dessus.

A. Estimation de la matrice H_∞

Pour déterminer les paramètres intrinsèques de la caméra, il est primordial de déterminer la matrice H_∞. En effet, elle représente l'homographie du plan à l'infini. Ainsi, permet de décrire la caméra en mouvement. A un instant τ, la projection des points de la scène à l'infin dans le repère de référence. Si K est la matrice de référence alors,

$$\lambda m_\infty = \begin{bmatrix} K & 0 \end{bmatrix} \begin{bmatrix} M_\infty \\ 0 \end{bmatrix} = KM_\infty, (3)$$

où λ est un facteur d'echelle

Lorsque l'on déplace la caméra et à l'instant $\tau + \Delta\tau$ elle prend une image. Considérant $\begin{bmatrix} R & t \end{bmatrix}$, on a alors :

$$\lambda m\prime_\infty = \begin{bmatrix} K & 0 \end{bmatrix} \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} H_\infty \\ 0 \end{bmatrix} = KM_\infty (4)$$

$$\implies \lambda m\prime_\infty = KRm\prime_\infty = KRK^1 m\prime_\infty = H_\infty m\prime_\infty (5)$$

Dans cette équation H_∞, permet de lier tous les points de la scènes au plan à l'infini. Ainsi, pour la déterminer il faut prendre en considaration 4 points de l'horizon mis en correspondance dans une scène image. Il faut donc au moins 2 vues.

B. Recherche des paramètres intrinsèques

A nous allons nous intéresser à l'estimation des paramètres intrinsèques. Pour ce faire, nous allons allons nous interesser au travaux en [3] qui repose sur *la théorie de la conique absolue et des équation de Kruppa* En partant de l'équation en (5). On remarque, que l'homographie H_∞ ne dépend que des composantes de rotation relatif au mouvement de la caméra, elle est donc indépendante de la translation t.
Ainsi, on a : $R = K^{-1}H_\infty R$ (6)
Par ailleurs, on sait : $RR^T = I$.

$(K^{-1}H_\infty K)(K^{-1}H_\infty K) = I$

$\Longrightarrow H_\infty KK^T H_\infty^T = KK^T$

$\Longrightarrow H_\infty \ \Delta \ {H_\infty}^T = \Delta$ (7)

Avec Δ la matrice de Kruppa, qui définit la conique duale ω, l'image de la conique Ω de l'espace scène se trouvant au plan de l'infini comme l'illustre la figure 2.
Or $\Delta = B^* = det(B)(B^{-1})^T$,
avec
$B = (K^{-1})^T K^{-1}$

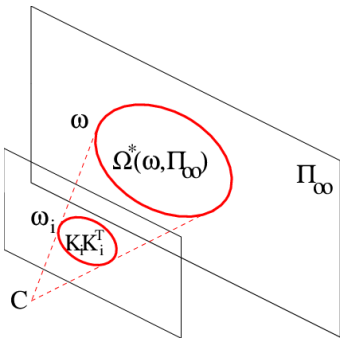


Fig. 2. Projection de la conique absolue

En résolvant l'équation de Kruppa, on arrive à
$\lambda\Delta = \lambda KK^T$

$$\lambda\Delta = \lambda \begin{bmatrix} u_0 + \alpha^2 \frac{}{sin^2}\theta & u_0 v_0 - \alpha_u \alpha_v \cot\theta \frac{}{sin}\theta & u_0 \\ u_0 v_0 - \alpha_u \frac{}{sin}\theta & u_0 + \alpha^2 \frac{}{sin^2}\theta & v_0 \\ u_0 & v_0 & 1 \end{bmatrix}$$

Il en résulte que la matrice H_∞ peut être déterminé par la relation (7). Ce qui permet de retrouver la matrice des paramètres intrinsèques K. Il suffit donc de deux vues pour déterminer entièrement H_∞. Cependant, pour des raisons de stabilité numérique il est préférable d'utiliser deux déplacements, l'un pour corriger lea disparité verticale et une autre pour la disparité horizontale.

C. Recherche des paramètres extrinsèques

Une fois la matrice K des paramètres intrinsèques déterminées, on peut simplement retrouver la matrice de rotation R grâce à l'équation (6). Par ailleurs, pour ce qui est du calcul de la matrice de translation, il suffit de prendre un point M appartenant à la scène mais qui n'appartient pas au plan infini.
On a donc :
m′≠Hm
ce que l'on peut vérifié par les équations suivantes :

$$m = \begin{bmatrix} R & 0 \end{bmatrix} \begin{bmatrix} R \\ 1 \end{bmatrix} \Longrightarrow m = KM$$

$$m' = \begin{bmatrix} RR & Kt \end{bmatrix} \begin{bmatrix} M \\ 1 \end{bmatrix} = KRM + Kt$$

$$\Longrightarrow m' = KRK^{-1}m + Kt = H_\infty m + Kt$$

En considérant les correspondances entre deux images successives de la séquence et ayant calculé l'homographie de l'infinie et les paramètres intrinsèques de la caméra, le vecteur de translation t peut être récupéré(à un facteur d'échelle près) par résolution aux moindres carrés de l'équation :

$\sum_{i=0}^{n} \| \| H_\infty m + Kt - m_i' \| \|$

IV. Mise en commun de deux images

À présent que nous avons bien compris comment fonctionne la détermination des paramètres intrinsèques et extrinsèques, il vient que l'un des points importants qui nous permettent d'y parvenir à déterminer ces paramètres est la capacité à identifier des points identiques entre 2 images d'une même scène. En effet, de nombreux travaux ont été faits sur cette question.

A. Détection des points d'intéret par SIFT

Dans cette section, je me suis intéressé à l'implémentation d'un algorithme SIFT en me référant à l'article [1] de M. Chapron, dans le but de qualifier son impact sur la qualité de l'auto-calibration. L'algorithme SIFT permet de prendre en compte les rotations entre différentes images d'une scène, il est donc une solution au problème posé par le détecteur d'Harris. Le détecteur d'Harris-Stephens présente un défaut majeur : les coins où il localise son invariant par rotation, mais pas par changement d'échelle. Ce qui signifie qu'on parvient à détecter les mêmes coins (avec des orientations différentes) lorsqu'on applique une rotation à l'image, mais pas lorsqu'on change le zoom. Ce qui pose problème lorsque l'on n'a que deux 3 vues pour déterminer les paramètres de la caméra. On pourrait illustrer son mode opératoire par la figure ci-dessous.

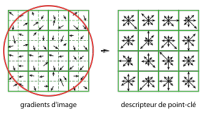


Fig. 3. sift invariant par les rotations

B. *Descripteur ORB*

ORB fonctionne aussi bien que SIFT sur la tâche de détection de caractéristiques tout en étant presque deux ordres de grandeur plus rapide. ORB s'appuie sur le célèbre détecteur de point-clé FAST et le descripteur BRIEF. Les principales contributions d'ORB sont les suivantes : l'ajout d'un composant d'orientation rapide et précis à FAST; le calcul efficace des caractéristiques BRIEF orientées; l'analyse de la variance et de la corrélation des caractéristiques BRIEF orientées.

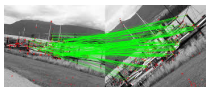


Fig. 4. Application de l'algorithme ORB

Cependant, les fonctionnalités FAST ne disposent pas se composant d'orientation ni de fonctionnalités multiéchelles. L'algorithme orb utilise donc une pyramide d'images à plusieurs échelles. Une pyramide d'images est une représentation multiéchelle d'une seule image, constituée de séquences d'images qui sont toutes des versions de l'image à différentes résolutions. Chaque niveau de la pyramide contient la version sous échantillonnée de l'image par rapport au niveau précédent. Une fois que orb a créé une pyramide, il utilise l'algorithme rapide pour détecter les points-clés dans l'image. En détectant les points-clés à chaque niveau, l'orbe localise efficacement les points-clés à une échelle différente. De cette façon, ORB est invariant d'échelle partielle.

V. Conclusion

En conclusion de nombreux articles ont été réalisés sur l'auto-calibration des caméras. De nombreuses solutions sont proposées. Elle repose pour la plupart au recours aux équations de Kruppa et à la conique absolue. Par ailleurs, afin d'apporter un peu d'améliorations face à ce que l'on retrouve dans la littérature. La deuxième partie de cet article visait à implémenter les différents algorithmes de détection de points d'intérêt en vue d'avoir une calibration plus rapide. Les expérimentations n'ont pas été mené à terme. Mais on peut dire que l'algorithme ORB semble être le meilleur mais il faudrait poursuivre les tests pour en être certain.

References

[1] Michel Chapron. Sur la détection de points d'intérêt dans les couples d'images stéréo et végétales. In *22ème Conférence du Columa, Journées internationales sur la lutte contre les mauvaises herbes*, pages 1–8, 2013.

[2] Elsayed E Hemayed. A survey of camera self-calibration. In *Proceedings of the IEEE Conference on Advanced Video and Signal Based Surveillance, 2003.*, pages 351–357. IEEE, 2003.

[3] Mohamed Moncef Ben Khelifa and Moez Bouchouicha. Du traitement de l'information visuelle à la planification de trajectoires: Application à la robotique mobile d'assistance. *rapport de stage, laboratoire SIS/AI université de Toulon et du Var*, 2003.

[4] Guanghui Wang, QM Jonathan Wu, and Wei Zhang. Kruppa equation based camera calibration from homography induced by remote plane. *Pattern Recognition Letters*, 29(16):2137–2144, 2008.

Intelligent tracking of hand position and movement for the sake of reconstitution and use as a controller in Virtual Reality

ORLHAC Baptiste
Robotics
ENSTA Bretagne
Brest, France
baptiste.orlhac@ensta-bretagne.org

***Abstract*—This document is a paper summarizing a practical study about ways to use learning algorithms and computer vision to implement hand tracking, hand gesture recognition and other hand movement features for use in virtual reality environments.**

***Index Terms*—vision, hands, VR, tracking**

I. Introduction

The goal of this paper is to extend the use of hand tracking and hand gesture recognition algorithms to get rid of the remotes still currently used in most virtual reality devices, and provide better user experience. This paper will also try and highlight the current limitations of such learning algorithms and how it could be possible to add other technologies to such a system in order to compensate.

II. Hand tracking

The base reference material for this whole section is Shangchen et al. 2020 [1].

A. Basics of hand tracking through leaning algorithms

The use of computer vision and more typically the use of RGB cameras for hand tracking has been at the core of many reflections for the past years. The field is dominated either by algorithms using depth cameras either limiting the field of view and limiting their use in lightweight portable systems, or by algorithms using deep-learning and RGB cameras, which can be really efficient at tracking hand poses, but less efficient at tracking hand movement.

In Reference [1], approach consists of four fish-eye monochrome cameras (which can be mounted on a VR headset for example) and a depth camera was used for the sake of enhancing the model produced by the learning algorithm by labeling the keypoints.

This algorithm uses two different networks to first identify the position of the hand and then produce a heat map of the position of twenty-one (21) tracking points to locate joints and position fingers and palm. This model allows for twenty-six (26) degrees of freedom and thus for a good tracking of both movement and position of the hand.

B. Applications of hand tracking in VR

In virtual reality or augmented reality (VR/AR) environments, hand tracking has two major applications. The first use to be made of a tracking algorithm is the recreation of the hand in the simulated environment. For instance, in [1], a mesh is added and superposed in real-time to the tracked hand pattern, which can then be implemented in the examples in AR.

This typically enhances the user experiences, as he is able to locate his own hands in either a fully-simulated environment, or a partially-simulated one, and thus feel immersed in the this world.

The second major application of hand tracking in VR is movement-tracking for the sake of interaction with a virtual world. In this case, the algorithm has to be coupled with a physics engine and/or a gesture recognition algorithm and/or haptic sensors to detect and simulate collision with simulated objects.

III. Hand gesture recognition

The base reference material for this whole section is Siddarth S. et al. 2012 [2]

A. Basics of hand gesture recognition

As presented in [2], hand gesture recognition has first been implemented as a means to control one's computer's interface. At the time the paper was written, 10 years ago, they envisaged using it for interacting with various menus, but also, more importantly, for uses in gaming.

Reference [2] describes how to use the basic computer vision library OpenCV to track and map the pose of the hand, as long as it is fairly static. The process is in some ways pretty similar as the one developed in the former section, in that it first tracks a primal mass in which the hand is contained, to then track a more precise shape of the hand.

From this article dating back to 2012, and the previous one, we could imagine building a learning algorithm based around the same method and keypoints as in [1] and proceeding to associate a pose recognition to the tracking and reconstitution with a mesh.

B. Uses for hand gesture recognition in VR

Nowadays, the market of VR systems is mostly dominated by VR headsets, equipped to immerse the user, through his eyes, into this other reality. Most of the work revolving around VR is centered around ways to interact with the virtual environment, for example through infinite treadmills to virtually move the user in it [3]. This focus has been going on for the last 20 years, while another type of system has been involved in tests for use in both VR and AR : haptic interfaces [4] [5].

The use of hand gesture recognition in VR is another way of providing interaction with the virtual environment, for instance through a virtual interface. With a system combining both solutions presented above, it is easy to imagine tracking a combination of both pose and movement of the hand thus making it possible to perform actions such as scrolling through a virtual menu or pushing virtual buttons for instance. However, the user experience would not be complete without also integrating haptics to such a system

IV. The combination of algorithms

A. Work basis

The idea of this paper is to illustrate my work on computer vision involved in VR, and possibly couple it to an haptic interface, which now exist in light and portable forms, to generate an environment in which the player most likely would not have to interact with any remote in order to interact with a game, interface or environment.

My first approach revolved around trying to recreate the learning models produced in [1] to try and get an approximate result at tracking hand joints and pose. Unfortunately, I was not able to do so, after investing most of my time in this task. I then chose to dedicate my hours to finding potential clever ways to combine algorithms or external systems to create an efficient and easy-to-use VR interface.

B. Combining hand tracking and hand gesture recognition

As mentioned before, the models and algorithms presented in [1] and [2] both use the same basic principle : first tracking a heat map of the position of the hand to then switch to a more defined and detailed tracking, either by using keypoints or by using the contour features of OpenCV.

From this, I understood that there could be two possible ways to implement the solutions together :

1) Improving the keypoint tracking system and associate different relative positions of keypoints to specific hand poses and link those poses to different actions
2) Applying the hand gesture recognition system proposed in [2] to the hand mesh used in [1] to generate trigger poses to produce various actions. This method, eventhough potentially increasing the risk of non-detection, or bad detection, can be optimized by applying the gesture recognition algorithm to the mesh on a virtual camera, thus being able to enhance contrast and adjust illumination

As stated right above, these solutions are still limited, and even more bound by the limits of the models presented in the respective papers. The major issues in such a system would be interaction with real-world objects, as they appear to confuse the learning algorithms, as shown in [1]

C. Introducing haptics

Haptic interfaces are hardware solutions that are capable of tracking force applied and/or potentially generating feedback forces from virtually-generated objects. It means that, by moving a finger in real life, it is possible to track the force applied by the user, and thus calculating the adequate acceleration for the object they touched in the virtual world. Moreover, when approaching their hand from an object in the virtual worlf, the haptic device is able to generate feedback forces adapted to resistance, softness, and global texture the virtually generated object mimics.

D. Haptics as a means to improve the system

Haptics can be a real asset in the system we envisaged above. Indeed, given their ease to recreate collision with an object in the virtual world, an haptic interface would help tracking hand position more precisely around an object by detecting which points are in contact with the given object. It would allow for a better detection of the nature of hand object interaction and help clarify the detection given by a system such as in [1].

We also mentioned above creating a virtual interface the user would use to navigate menus in a game or any virtual environment, and haptics could help recreate the feeling of actually pressing a button, by sending a feedback force to the device, in order to stimulate and block the finger as if it had touched a real one.

V. Conclusion

Learning algorithms and portability of vision devices have greatly improved over the course of the last 10 years, allowing for a better adaptability of VR devices. With such possibilities, envisaging VR hardware not necessitating any remote becomes a necessity.

The solutions proposed in [1] and [2] are both efficient at tracking either movement or pose of hands, and they could be adapted to work in pair, allowing for better tracking and the implementation of user actions in the virtual world.

Recent progress made on portability of haptic interfaces is to take into account in the performance in such a system, adding interesting functionnalities that both enhance the user's experience, and could help reducing error rate of learning algorithms by crossing data during model creation.

References

[1] S. Han et al., "MEgATrack: monochrome egocentric articulated hand-tracking for virtual reality," ACM Trans. Graph., vol. 39, no. 4, Aug. 2020, doi: 10.1145/3386569.3392452.

[2] S. S. Rautaray, "Real Time Hand Gesture Recognition System for Dynamic Applications," IJU, vol. 3, no. 1, pp. 21–31, Jan. 2012, doi: 10.5121/iju.2012.3103.

[3] H. Iwata, "Walking about virtual environments on an infinite floor," in Proceedings IEEE Virtual Reality (Cat. No. 99CB36316), Mar. 1999, pp. 286–293. doi: 10.1109/VR.1999.756964.
[4] A. M. Kunz and A. Burri, "Design and Construction of a New Haptic Interface," Dec. 2020, pp. 497–502. doi: 10.1115/DETC2000/CIE-14582.
[5] V. Q, P. Lammertse, E. Frederiksen, and R. B, "The HapticMaster, a new high-performance haptic interface," Jul. 2002.

Implementation of Gym-gazebo2, a Reinforcement learning package for ROS2

Hugo PIQUARD HUGO.PIQUARD@ENSTA-BRETAGNE.ORG
student of ENSTA Bretagne

Abstract

This paper presents an application of the Gym-gazebo2 (5) simulator, composed of the Robot Operating System (ROS), Gazebo and Reinforcement Learning (RL) toolkit from OpenAI. The purpose of this article is to making its use clearer through the example of an implementation. But also to discuss of the software limits and application. For this purpose, I will describe the overall functioning of the simulator and its architecture, including the algorithms used in the implementation.

1. INTRODUCTION

For several years and now with the second version, the middleware ROS is omnipresent in industry and research design robot. Combine with a powerful simulator, Gazebo, it allows to perform tests easier than before. The idea to merge ROS and current research in machine learning and more specifically in reinforcement learning has already demonstrated the great results with the previous version, gym-gazebo. That is why gym-gazebo2 is relevant to follow the rapid new advances in the area.Thus in a first step I will describe the use of the different modules of gym-gazebo2[1] software.

Then in a second phase I will present the implementation of gym-gazebo2 on a specific environment, the control of a simple car. Multiples scenarios were chosen as the reaching point scenario or the path of a maze. The models and algorithms combined in this application, will be exposed and the results analysed. The different RL method used will be compared for this specific case.

1. See https://github.com/AcutronicRobotics/ros2learn

2. STATE OF THE ART

2.1. Reinforcement learning

Reinforcement learning in robotics is one of the most research subject in the machine learning area. Unexpected progress have been made over the last year. Its robotics field applications are varied, AUV, drones, autonomous driving, industrial robots and arms... Numerous well-konw research institutes and Compagnies have made some fundamental achievement but still face some issues (6) (1) (9).

The landscape of algorithms in modern Reinforcement learning is diverse. The major branching points in an RL algorithm is the question of whether the agent has access to a model of the environment. A model is function that predict environment state transition and reward. Reward allows the agent (ie the robot) to improve its behaviour (called policy) by giving a way to provide a value of each action taken. To compute this value, the model use environment state (in based model RL algorithm).The advantage to having a model is that it allows the robot to plan and compute the action possibility range. The main downside is that a ground-truth model of the environment is usually not available to the agent. If an agent wants to use a model in this case, it has to learn the model from experience (7).

In this paper, we will use the algorithm Proximal Policy Optimization 2 (PPO2) (8) to compute the best policy for the robot which is a free model RL algoritm based on Policy Optimization, compound of Deep neuronal networks.

2.2. Gym-gazebo2

One of the challenging issues in RL is to acquirer enough data to train specific models.The use of simulators as close to reality as possible is therefore essential to continue research in this field. Thus AI institute managed to develop toolkit in order to easier

implemented their research in simulated environment as Gym API from OpenAI company (4).

Gym-gazebo 2 (5) rely on ROS 2 which provide an a compact and easy-to-tune robot architecture.The toolkit use Gym API and Baseline (3) package, a set of RL algorithm developed by OpenAI. Also the node implementation is in Python ROS Client Library. They have add Modular Articulated Robotic Arm (MARA) designed by Acutronics Robotics, company closed in 2019. This arm allows the user to evaluate algorithms and test different models.

3. ARCHITECTURE AND USAGE

3.1. Architecture

The toolkit Gym-gazebo 2 is composed of three independent and different parts Figure 1. The first one is the ROS 2 environment which allow the user to design and build the robot through packages. In this context I used one ROS 2 name descriptions with the complete robot description such as parts of the robot (Links) and connections between them (Joints). The idea behind use independent library is that, the robot description package could be re-use in another way such as another simulation. The second part is Gazebo, a dynamic and graphical simulator. In this architecture, Gazebo is used by another package. The last part is composed of two libraries, Gym and Baseline. The both were design and share by OpenAI. Gym (2) is an API to build environment for our control in Reinforcement learning. It defined the possible space of the robot action and the whole process. This library called the second one, during the learning process. Baseline contains a major part of modern reinforcement learning and tools to use them.

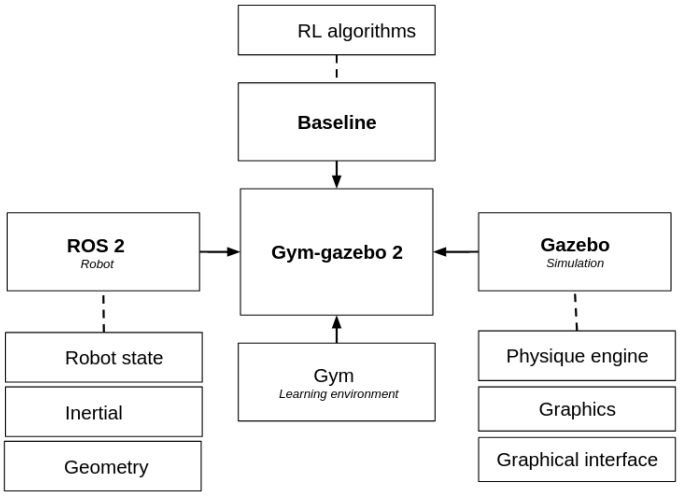


Figure 1: Gym-gazebo 2 architecture

3.2. Usage

The utilisation of the MARA example has been already detail on the Github repesitory. I will detail here how to design and use the gym-gazebo 2 toolkit in order to use our proper robot and model. As I explained in Section 3.1, the robot have to be build thought package in a ROS 2 workspace. In this paper I chose to separate the description package and Gazebo simulator package, *robotDescription* and *robotGazebo*. At least on of this package contains a launcher to start the simulation and instantiate the robot. An another folder must contains the Gym model *gymModel* where step learning will be process. This model has an initialisation function to declare and subscribe gazebo topic such as get the coordinates of the robot from the simulator and setting the Gym environment . An observation function to get the diverse information to compute environment state, a processing function to compute this state and reset function. In the same folder, function to compute the reward of actions and execute the robot launch file must be implemented. And finally, a last file has to instantiate the Gym model, initialise the model environment, chose and execute the RL algorithm from Baseline library save training data Table 1.

4. APPLICATION AND RESULTS

4.1. A simple car

I have chosen a simple car to test the Gym-gazebo 2 software. The car is composed of three rolls, two lateral to propels the car and one in front to control the direction. The last one is similar to a sphere in order to change the vehicle's yaw. Geometry and dynamics component could be find here[2].The car controller is a ROS 2 plugin[3] which control only the velocity and the yaw of the car. I suppose in this context that the car

2. See https://github.com/hugoPiq/Initiation_recherche_GG2
3. See https://github.com/ros-simulation/gazebo_ros_pkgs/blob/foxy/gazebo_plugins/worlds/gazebo_ros_diff_drive_demo.world

Table 1: Folders descriptions to create a new model and environment in Gym-gazebo2

robotDescription	Package ROS 2 of the robot description (URDF, Xacro file)
robotGazebo	Package ROS 2 with Gazebo simulator and robot launch (ROS 2 Launch files)
gymModel	Gym model and environment (Python files)
main (file)	Instantiate model, environment and RL algorithm selection (Python file)

can holy move on a horizontal plane, so the model is a three dimensional model, x and y coordinates and γ, the car's yaw.

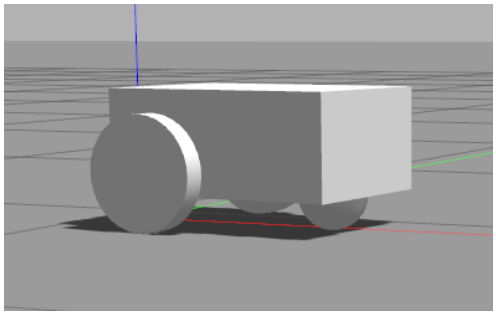


Figure 2: Simple car model, Gazebo

4.1.1. Gym model

The first step to define the gym model is to design the action space, which is a continual two dimensional box here for velocity and yaw $[-\pi;\pi]\times[-5;5]$. Then we have to define the observation space. In this context, from the simulator I can get the coordinates and rotation of the cars $[-\pi;\pi]\times[-\infty;\infty]\times[-\infty;\infty]$. This observation state in this case corresponds of the environment state. This environment state allows the model to compute the reward of each action. For that, I use different reward function depending of the scenario Section 4.1.2.

4.1.2. Scenarios

4.1.3. Reaching point

In this article, I have designed two scenarios with the simple car model. The first one corresponds to a point reaching. The car has to forward toward a target point as near as possible. For this purpose, I have chosen a specific reward function. To compute these reward function, I use car's coordinates from environment state to obtain distance between car and target position. The reward function (5) in this scenario is

$$rew_{dist} = \underbrace{\frac{e^{-\alpha*x} - e^{-\alpha} + 10 * (e^{-\alpha*\frac{x}{done}})}{1 - e^{-\alpha}}}_{rew_{dist}} - 1$$

x corresponds to the target-car distance, α and *done* are the hyper-parameters.

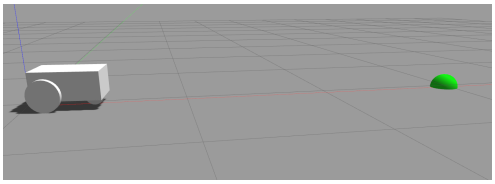


Figure 3: Reaching point scenario, Gazebo

4.1.4. Maze

The second is a find-maze-exit scenario. The simple car has to find straight forward in the maze until finding the end. This scenario is more complex than the previous one, even if at first sight the walls can guide the car to the exit and thus restrict the possible movements. Indeed it is necessary to add new collision constraint on the reward.

$$rew = \begin{cases} rew_{dist} - 1 - \delta * (2 * min(rew_{dist}, 0.5))^{\tau} \\ \text{if colliding} \\ rew_{dist} - 1 \text{ not colliding} \end{cases}$$

τ and are the hyper-parameters. Theses reward function are normalised between -1 and 10.

4.1.5. Reinforcement learning algorithm

For both scenario I have chosen the Proximal Policy Optimization 2 as MARA (5). Similar to the Policy gradient method, it based on deep neuronal network optimisation but PPO2 (8) is faster and lower data consumption. This algorithm allows the robot to compute the better policy ie set of action by tacking into account the reward.

4.2. Results

4.2.1. Reaching point

For the first scenario, the results are quite conclusive. I got a relatively good accuracy. The possible errors are due to the hyperparameters that I refined during the tests but that would need more experimentation. The following data are obtained with 10 tests using the trained model.

Table 2: Mean error distribution of the training with respect to the target for the simple car (distance) environment

Accuracy	**Axe x**	**Axe y**
Distance(mm)	8.12 ±1.23	7.93 ±2.3

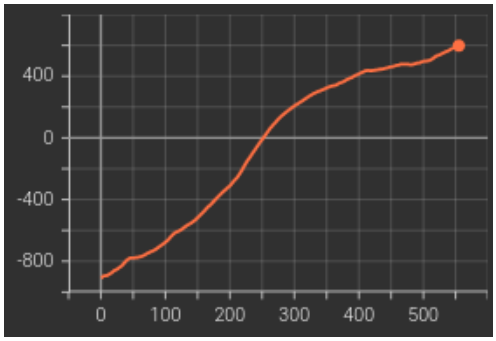


Figure 4: Evolution of reward during the training, result smoothed

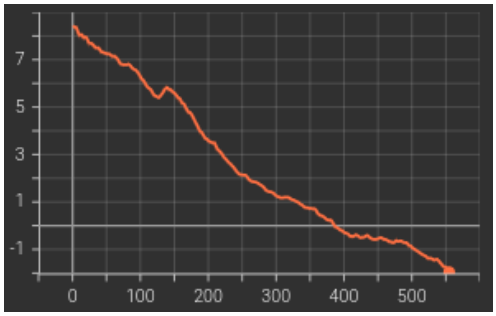


Figure 5: Evolution of entropy during the training

4.2.2. Maze

For the second scenario, the results are less promising. I got a bad accuracy. The errors probably come from the colission model which does not seem to be relevant. Unfortunately, I did not find the explanation.The following data are obtained with 10 tests using the trained model

Table 3: Mean error distribution of the training with respect to the target for the simple cars (distance + collision) environment

Accuracy	**Axe x**	**Axe y**
Distance(mm)	15.42 ±6.3	12.41 ±7.8

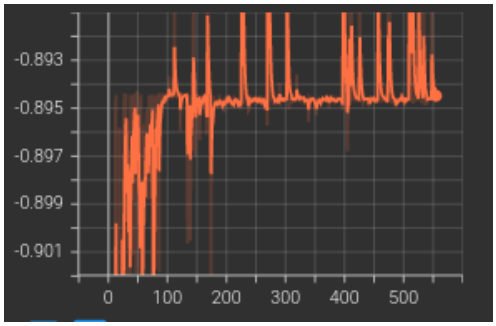


Figure 6: Evolution of reward during the training, not smoothed

5. DISCUSSION

Some aspects of Gym-gazebo 2 software should be improved or fixed. Firstly, this architecture only works on an older version of ROS 2, Dashing which is less stable than the current versions. In addition, Gym-gazebo 2 and the Baseline package is no longer updated since 2019. The closure of Acutronics Robotics led to the closure of the Github repository and therefore there is no more support. Finally, the documentation of the baseline and in general of Gym-gazebo2 is not up to its performances.

6. CONCLUSION

To conclude, the Gym-gazebo 2 toolkit is a powerful tool to design and test Reinforcement learning commands. The simple modelling I have chosen here can be extended to more complex models such as the control of an AUV. Nevertheless, modern reinforcement learning algorithms are not trivial, so it is necessary to understand them well and to carry out a large number of experiments to determine which ones are interesting as well as their hyperparameters. As a result, it is sometimes complicated to train even a simple model successfully. Finally, the Gym-gazebo 2 software would need to be updated, so that it could be more widely used and more easily accessible.

References

[1] Bansal T Burda Y et al Al-Shedivat, M. Continuous adaptation via meta-learning in nonstationary and competitive environments. corr 2017. 2017.

[2] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[3] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. https://github.com/openai/baselines, 2017.

[4] L. Pettersson J. Schneider J. Schulman J. Tang G. Brockman, V. Cheung and W. Zaremba. Openai gym. 2016.

[5] Elias Barba Moral Lander Usategui San Juan Alejandro Solano Rueda Víctor Mayoral Vilches GNestor Gonzalez Lopez, Yue Leire Erro Nuin and Risto Kojcev. gym-gazebo2, a toolkit for reinforcement learning using ros 2 and gazebo. 2019.

[6] Dhruva TB Sriram S et al Heess, H. Emergence of locomotion behaviours in rich environments. corr 2017. 2016.

[7] A. Ray J. Schneider W. Zaremba J. Tobin, R. Fong and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. 2017.

[8] Prafulla Dhariwal Alec Radford Oleg Klimov John Schulman, Filip Wolski. Proximal policy optimization algorithms. 2017.

[9] Finn C Darrell T et al. Levine, S. End-to-end training of deep visuomotor policies. j mach learn res 2016. 2017.

Trajectory following with a double Ackermann steering robot.

Samuel Prouten

***Abstract*—This document is an explanation of the double Ackermann steering geometry and how it can be used to follow complex trajectories. Additionally, a controller for a robot with four independently steerable wheels is discussed.**

***Index Terms*—Ackermann steering, Control Theory, State Space Model, Simulation**

I. Introduction

Highly maneuvering vehicles are frequently needed in particular environments such as the ones in which agricultural robots evolve [1]. Solutions involving totally holonomic vehicles such as omniwheel-based platforms are often not well suited for outdoor rough terrain, due to poor traction on uneven surfaces. However, research has been conducted on the matter [2]. More frequently, the solution to obtain high maneuverability in delicate terrain, such as field crops or grass, is to use robots with fully steerable wheels. This ensures that the wheels can always turn in the direction of motion and thus, limits the potential side-slip that can occur, reducing damage to the terrain and improving performance.

Many robots are equipped with such steering. However, the literature on how to control the wheel angles and speeds in order to follow complex paths is surprisingly modest. Thus, developing a model of a fully steerable wheeled robot and a controller able to drive such a model makes sense developing a controller for these robots makes sense considering the expansion of the field of agricultural robotics [3].

II. Double Ackermann steering geometry

The double Ackermann steering geometry is an extension of the Ackermann steering geometry [4]. Like the latter, the double Ackermann geometry follows the same principle: the rotation center is the same for all wheels of the vehicle (Fig.1).

The interest of such a type of directional geometry is that it allows to strongly reduce the minimum rotation radius of the vehicle. If the wheels have no angle limits, the vehicle can even turn on the spot by orienting all its wheels towards its center for example. The vehicle can also move sideways, in a "crab" motion, which allows it to move freely while maintaining the desired heading.

III. Model

Inspired by [5], we can model this type of vehicle with a double bicycle model (Fig.2). The center of the vehicle is named O, it is also the origin of the robot's frame of reference. The front and rear wheels are positioned in F and B, and are oriented according to δ_f and δ_r respectively, positively defined with respect to the $\overrightarrow{x}$ axis of the robot. The length of the robot

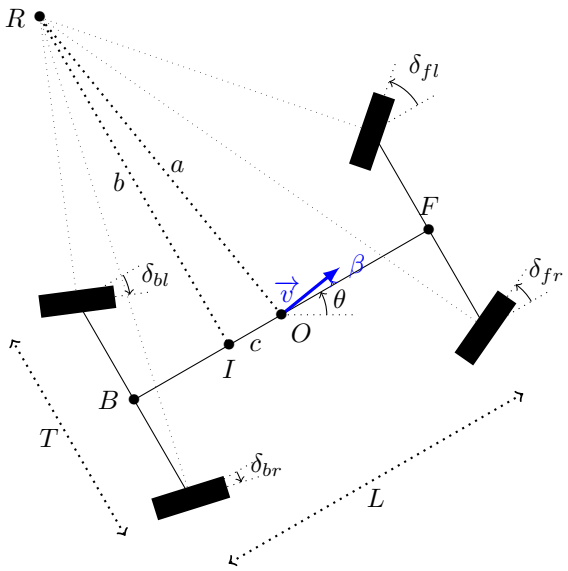


Fig. 1. Double Ackermann steering geometry

is noted L and in the robot's frame of reference, F and B are placed in $+\frac{L}{2}$ and $-\frac{L}{2}$ along the $\overrightarrow{x}$ axis. θ is the orientation of the robot in the world frame of reference.

As all the wheels rotate around the same point noted R, we can define a distance $a = OR$, as well as a point I being the projection of R on axis $\overrightarrow{x}$ of the robot. We can also compute the distances $b = IR$ and $c = OI$. We obviously have $a^2 = b^2 + c^2$.

IV. Control

To be able to follow complex trajectories, it is necessary to be able to follow a given target position, while being able to control the heading of the robot. It is therefore necessary to develop a controller capable of calculating the angles and speeds of the wheels necessary to follow the trajectory. Control theory is a well documented field, however the way to tackle a problem can sometimes be large [6]. In the next sections, I will explain the different controllers I have developed in order to follow a specified path with a double Ackermann steering robot.

A. Feedback Linearisation

We can calculate the dynamics of the robot according to the model in Fig.2. The state vector of the robot is $X = (x, y, \theta, \delta_f, \delta_b, v_f)^T$. Where :

- x: the position of the robot in x
- y: the position of the robot in y

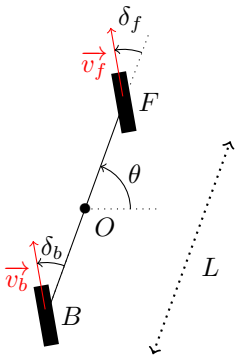


Fig. 2. Model used for feedack linearization

- θ: the heading of the robot
- δ_f: the angle of the front wheel
- δ_b: the angle of the rear wheel
- v_f: the speed of point F along the robot's $\overrightarrow{x}$ axis

If we define $s = sin(\delta_f) + cos(\delta_f) \cdot tan(\delta_b)$, the evolution equations of the model are :

$$\begin{cases} \dot{x} &= v_f \cdot (cos(\theta) \cdot cos(\delta_f) - \dfrac{sin(\theta)}{2} \cdot s) \\ \dot{y} &= v_f \cdot (sin(\theta) \cdot cos(\delta_f) + \dfrac{cos(\theta)}{2} \cdot s) \\ \dot{\theta} &= \dfrac{2}{L} \cdot v_f \cdot (sin(\delta_f) - cos(\delta_f) \cdot tan(\delta_b)) \\ \dot{\delta_f} &= u_1 \\ \dot{\delta_b} &= u_3 \\ \dot{v_f} &= u_2 \end{cases}$$

With $y = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix}$, such that $\ddot{y} = A(X) \cdot u + b(X)$, we use pole placement and find $v = A(X)^{-1} \cdot (u - b(X))$. So $\ddot{y} = v$. Please see appendix A for the equation details.

The problem is that the second column of $A(X)$ is proportional to $\frac{1}{cos^2(\delta_b)}$. So when the singularity of $\delta_b = \pi/2$ occurs, neither the evolution equations nor the controller are valid. This problem stems from the geometrical constraint that point F and point B must remain at equal distance so $v_f \cdot cos(\delta_f) = v_b \cdot cos(\delta_b)$

B. Model using virtual variables

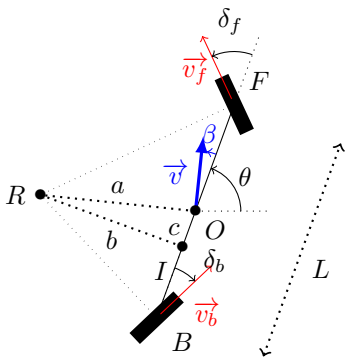


Fig. 3. Model introducing virtual variables

Faced with the difficulty of applying a feedback linearization control of the state, another model of the robot state is necessary. Therefore, we need to find a state vector that perfectly describes the robot state, while ensuring that the variables are not related. The model presented in Fig.3 respects these requirements.

The state vector is now chosen as $X = (x, y, \theta, v, b, c)^T$. The evolution functions are:

$$\begin{cases} \dot{x} = v \cdot cos(\theta + arctan(\frac{c}{b}) \\ \dot{y} = v \cdot sin(\theta + arctan(\frac{c}{b}) \\ \dot{\theta} = \dfrac{v}{\sqrt{b^2 + c^2}} \end{cases}$$

And v, b and c are chosen as inputs such as $u = (v, b, c)^T$.

With these equations, we can hope to directly control $y = (\dot{x}, \dot{y}, \dot{\theta})^T$, which would allow to control the speed and rotation of the robot. Thus the objective is to solve a non-linear system of three equations.

An analytical resolution produces singularities as it introduces the $arccos$ or $arcsin$ function, which is only defined for values in $[-1, 1]$ (appendix B).

I have experimented with numerical optimizers in an attempt to solve the system, or at least have an approximation of the solution. Although the method did deliver results that could be used in the simulation, the command was still very unstable when the robot switches from turning left to turning right as it forces b to jump from $+\infty$ to $-\infty$.

C. Final model

Thank you to Mr. Jaulin, teacher at ENSTA Bretagne, for this model which allowed me to establish an adequate controller for the problem.

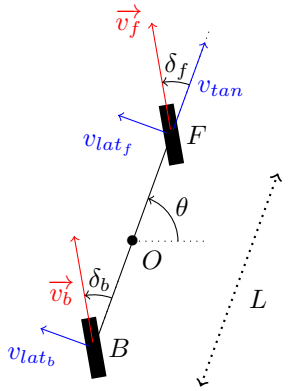


Fig. 4. Model used for the final controller

The robot's state vector is $X = (x, y, \theta, v_{tan}, v_{lat_f}, v_{lat_b})^T$. Where :

- x: the position of the robot in x
- y: the position of the robot in y
- θ: the heading of the robot
- v_{tan}: the tangent speed of the robot, i.e. its speed along the its $\overrightarrow{x}$ axis
- v_{lat_f}: the speed along the robot's $\overrightarrow{y}$ axis of point F

- v_{lat_b}: the speed along the robot's $\overrightarrow{y}$ axis of point B

Once again, We can see that the variables are all independent and the state vector can be chosen in a totally random way without creating singularities.

We can then calculate the dynamics of the model. The inputs of the system are $u = (u_1, u_2, u_3, u_4)^T = (\dot{v}_f, \dot{\delta}_f, \dot{v}_b, \dot{\delta}_b)^T$

We first calculate the evolution of $s = (v_{tan}, v_{lat_f}, v_{lat_b})^T$ with regards to u:

$$\begin{cases} \dot{v}_{tan} &= \dot{v}_f \cdot cos(\delta_f) - v_f \cdot \dot{\delta}_f \cdot sin(\delta_f) \\ \dot{v}_{lat_f} &= \dot{v}_f \cdot sin(\delta_f) + v_f \cdot \dot{\delta}_f \cdot cos(\delta_f) \\ \dot{v}_{lat_b} &= \dot{v}_b \cdot sin(\delta_b) + v_b \cdot \dot{\delta}_b \cdot cos(\delta_b) \end{cases}$$

However, the geometrical constraints mentioned above impose $v_b \cdot cos(\delta_b) = v_f \cdot cos(\delta_f) = v_{tan}$.

So:

$$\begin{cases} \dot{v}_{tan} &= \dot{v}_f \cdot \dfrac{v_{tan}}{v_f} - v_{lat_f} \cdot \dot{\delta}_f \\ \dot{v}_{lat_f} &= \dot{v}_f \cdot \dfrac{v_{lat_f}}{v_f} + v_{tan} \cdot \dot{\delta}_f \\ \dot{v}_{lat_b} &= \dot{v}_b \cdot \dfrac{v_{lat_b}}{v_f} + v_{tan} \cdot \dot{\delta}_b \end{cases}$$

And with regards to u :

$$\dot{\mathbf{s}} = \underbrace{\begin{pmatrix} \dfrac{v_{tan}}{v_f} & -v_{lat_f} & 0 & 0 \\ \dfrac{v_{lat_f}}{v_f} & v_{tan} & 0 & 0 \\ 0 & 0 & \dfrac{v_{lat_b}}{v_f} & v_{tan} \end{pmatrix}}_{\mathbf{S(X)}} \cdot \mathbf{u}$$

We can then calculate the variation of $p = (x, y, \theta)^T$ as a function of s :

$$\begin{cases} \dot{x} = v_{tan} \cdot cos(\theta) - \dfrac{1}{2} \cdot sin(\theta) \cdot v_{lat_f} - \dfrac{1}{2} \cdot sin(\theta) \cdot v_{lat_b} \\ \dot{y} = v_{tan} \cdot sin(\theta) + \dfrac{1}{2} \cdot cos(\theta) \cdot v_{lat_f} + \dfrac{1}{2} \cdot cos(\theta) \cdot v_{lat_b} \\ \dot{\theta} = \dfrac{2}{L} \cdot v_{lat_f} - \dfrac{2}{L} \cdot v_{lat_b} \end{cases}$$

Which gives us in matrix form:

$$\dot{\mathbf{p}} = \underbrace{\begin{pmatrix} cos(\theta) & -\dfrac{1}{2} \cdot sin(\theta) & -\dfrac{1}{2} \cdot sin(\theta) \\ sin(\theta) & \dfrac{1}{2} \cdot cos(\theta) & \dfrac{1}{2} \cdot cos(\theta) \\ 0 & \dfrac{2}{L} & -\dfrac{2}{L} \end{pmatrix}}_{\mathbf{A(X)}} \cdot \mathbf{s}$$

We thus have the sequence of 2 stages:

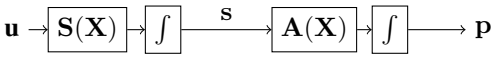


Fig. 5. Evolution stages of the system

To control the system, we must first be able to control the vector $\mathbf{s}$. So we want to invert the first block of the system.

We can simply write $u = S(X)^{-1} \cdot v_1$, then do a pole placement at -1 which gives $v_1 = (s_d - s) + \dot{s}_d$

So we need to get s_d and $\dot{s}_d$ from the second block $\mathbf{A(X)}$.

So we write $s_d = A(X)^{-1} \cdot v_2$ so that we have $\dot{p} = v_2$. With a pole placement we get $v_2 = (p_d - p) + \dot{p}_d$.

We differentiate this expression to get $\dot{s}_d = \frac{dA^{-1}}{dX}(X) \cdot A(X) \cdot s_d + A(X)^{-1} \cdot (\dot{p}_d - \dot{p} + \ddot{p}_d)$. We know that $\dot{p}$ is equal to $A(X) \cdot s$.

Finally, the controller is :

$$\begin{cases} s_d &= A(X)^{-1} \cdot (p_d - p + \dot{p}_d) \\ \dot{s}_d &= \dfrac{dA^{-1}}{dX}(X) \cdot A(X) \cdot s_d + A(X)^{-1} \cdot (\dot{p}_d - A(X) \cdot s + \ddot{p}_d) \\ u &= S(X)^{-1} \cdot (s_d - s) + \dot{s}_d \end{cases}$$

Once the controller has computed the command $u = (\dot{v}_f, \dot{\delta}_f, \dot{v}_b, \dot{\delta}_b)^T$, we can apply it to the robots actuators such as wheel and steering motors.

V. Simulations

The first and second controllers presented singularities when $|\delta_b| = \pi/2$. We can see this behavior in Fig.6. As soon as the rear wheel has an angle of $-pi/2$, the command becomes infinite and the state diverges severely.

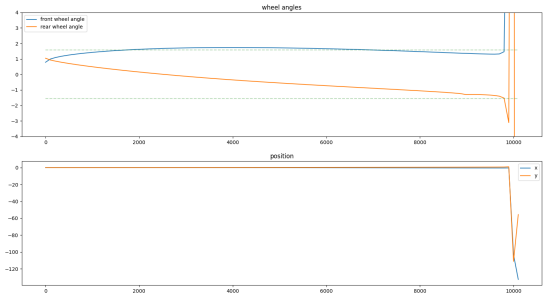


Fig. 6. Evolution of the robot until the singularity

The controller based on the numerical optimizer was able to follow a trajectory but computed noisy commands (Fig.7).

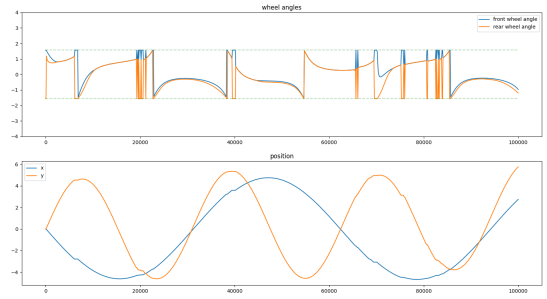


Fig. 7. Wheel angles and robot position for the optimizer based controller

Finally, using the last controller, the robot was able to follow a trajectory while aiming at a focus point (Fig.8).

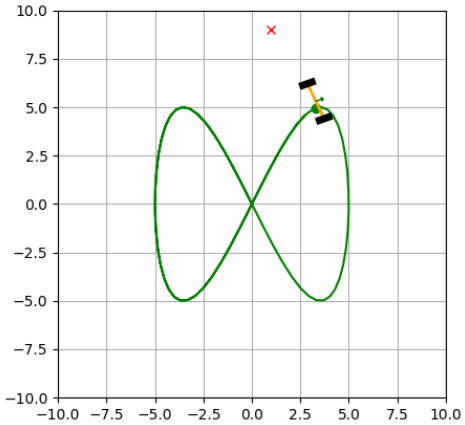


Fig. 8. Simulation of the robot

VI. Conclusion

In this article, the double Ackermann steering method has been discussed. Such a steering geometry enables vehicles to perform complex motion with high precision. Several controllers have been created to try to implement trajectory following with a double Ackermann steering robot. The difficulty of the task lies in the choice of the appropriate state space representation to compute the robots motion and develop the controller. A working controller has been created and gave satisfactory results in simulations. Future work would be to integrate the controller in a robot to test it in realistic environments.

Acknowledgements

Thank you to Mr Jaulin, professor at ENSTA Bretagne, for his help with the vehicle modeling and its advise on developing the controller for the robot.

References

[1] D. LECA, “Navigation autonome d’un robot agricole,” Theses, Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier), Apr. 2021. [Online]. Available: https://hal.laas.fr/tel-03278956

[2] C. McMurrough, H. Enotiades, S. Phan, S. Savoie, and F. Makedon, “Development of an omniwheel-based holonomic robot platform for rough terrain,” in *Proceedings of the 6th International Conference on PErvasive Technologies Related to Assistive Environments*, ser. PETRA ’13. New York, NY, USA: Association for Computing Machinery, 2013. [Online]. Available: https://doi.org/10.1145/2504335.2504398

[3] T. Duckett, *Agricultural Robotics: The Future of Robotic Agriculture*, ser. UK-RAS White Papers. UK United Kingdom: UK-RAS Network, Jun. 2018.

[4] W. Norris, *Modern steam road wagons, 1906 / by William Norris*. David and Charles Newton Abbot, 1972.

[5] K. Spentzas, I. Alkhazali, and M. Demic, “Kinematics of four-wheel-steering vehicles,” *Forschung im Ingenieurwesen*, vol. 66, pp. 211–216, 05 2001.

[6] A. De Luca and G. Oriolo, “Modelling and control of nonholonomic mechanical systems,” *Kinematics and Dynamics of Multi-Body Systems*, 01 1995.

APPENDIX A
EQUATIONS FOR FEEDBACK LINEARIZATION

$$\dot{x} = v_f \cdot (cos(\theta) \cdot cos(\delta_f) - \frac{sin(\theta)}{2} \cdot (sin(\delta_f) + cos(\delta_f) \cdot tan(\delta_b)))$$

$$\mathrm{A}_x = \begin{pmatrix} v_f \left(\frac{sin(\theta)}{2} \big(sin(\delta_f)tan(\delta_b) - cos(\delta_f)\big) - cos(\theta)sin(\delta_f) \right) & -v_f \frac{sin(\theta)}{2} \frac{cos(\delta_f)}{cos^2(\delta_b)} & cos(\theta)cos(\delta_f) - \frac{sin(\theta)}{2}(sin(\delta_f) + cos(\delta_f)tan(\delta_b)) \end{pmatrix}$$

$$b_x = -v_f \cdot \left(\dot{\theta} sin(\theta) cos(\delta_f) + \frac{\dot{\theta} cos(\theta)}{2} \big(sin(\delta_f) + cos(\delta_f) tan(\delta_b)\big) \right)$$

$$\ddot{x} = A_x \cdot \begin{pmatrix} u_0 \\ u_1 \\ u_2 \end{pmatrix} + b_x$$

$$\dot{y} = v_f \cdot (sin(\theta) \cdot cos(\delta_f) + \frac{cos(\theta)}{2} \cdot (sin(\delta_f) + cos(\delta_f) \cdot tan(\delta_b)))$$

$$\mathrm{A}_y = \begin{pmatrix} v_f \left(\frac{cos(\theta)}{2} \big(cos(\delta_f) - sin(\delta_f)tan(\delta_b)\big) - sin(\theta)sin(\delta_f) \right) & v_f \frac{cos(\theta)}{2} \frac{cos(\delta_f)}{cos^2(\delta_b)} & sin(\theta)cos(\delta_f) + \frac{cos(\theta)}{2}(sin(\delta_f) + cos(\delta_f)tan(\delta_b)) \end{pmatrix}$$

$$b_y = v_f \cdot \left(\dot{\theta} cos(\theta) cos(\delta_f) - \frac{\dot{\theta} sin(\theta)}{2} \big(sin(\delta_f) + cos(\delta_f) tan(\delta_b)\big) \right)$$

$$\ddot{y} = A_y \cdot \begin{pmatrix} u_0 \\ u_1 \\ u_2 \end{pmatrix} + b_y$$

$$\dot{\theta} = \frac{2}{L} \cdot v_f \cdot (sin(\delta_f) - cos(\delta_f) \cdot tan(\delta_b))$$

$$A_\theta = \frac{2}{L} \begin{pmatrix} v_f \big(cos(\delta_f) + sin(\delta_f)tan(\delta_b)\big) & v_f \frac{sin(\delta_f)}{cos^2(\delta_b)} & sin(\delta_f) - cos(\delta_f)tan(\delta_b) \end{pmatrix}$$

$$b_\theta = 0$$

$$\ddot{\theta} = A_\theta \cdot \begin{pmatrix} u_0 \\ u_1 \\ u_2 \end{pmatrix} + b_\theta$$

$$A = \begin{pmatrix} A_x \\ A_y \\ A_\theta \end{pmatrix}$$

$$b = \begin{pmatrix} b_x \\ b_y \\ b_\theta \end{pmatrix}$$

$$\begin{pmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{\theta} \end{pmatrix} = A \cdot u + b$$

APPENDIX B
EQUATIONS TO SOLVE THE NON-LINEAR SYSTEM

$$u_0 = \frac{\dot{x}_d}{cos\left[arctan\left(\frac{\dot{y}_d}{\dot{x}_d}\right)\right]}$$

$$u_1 = \sqrt{\frac{u_0^2}{\dot{\theta}_d \cdot \left[tan^2\left(arccos\left(\frac{\dot{x}_d}{u_o}\right) - \theta\right) + 1\right]}}$$

$$u_2 = u_1 \cdot tan\left[arccos\left(\frac{\dot{x}_d}{u_0}\right) - \theta\right]$$

Self-Play Reinforcement Learning for Two Agents

Hugo Sabatier, ENSTA BRETAGNE

***Abstract*—In this paper we will compare two self-play methods in Reinforcement Learning (RL) : Genetic Algorithm (GA) and Proximal Policy Optimization (PPO). We show that self-play can produce agents that can defeat the baseline policy without the need to train against it. Even if the results are not as good as for agents who train directly against the expert, they are still very encouraging and offer an interesting solution when there is no expert agent to train against.**

***Index Terms*—Reinforcement Learning, Self-Play, Genetic Algorithm, Proximal Policy Optimization.**

I. Introduction

THIS papers is about one of the most difficult tasks in artificial intelligence, the sequential decision making problem, whose applications include robotics and games. As for games, the successes are numerous. Machine surpasses man for several games, such as backgammon, checkers, chess, and go. A major class of games is the set of two-player games in which players play in turn, without any chance or hidden information. This class is sometimes called two-player perfect information games. There are still many challenges for these games. For example, for the game of Hex, computers have only been able to beat strong humans since 2020 [1]. For general game playing (even restricted to games with perfect information): man is always superior to machine on an unknown complex game (when man and machine have a relatively short learning time to master the rules of the game). In this article, we focus on two-player zero-sum games with perfect information, although most of the contributions in this article should be applicable or easily adaptable to a more general framework. [2]

Recent improvements in Monte Carlo Tree Search (MCTS) have focused on the automatic learning of MCTS knowledge and their uses. This knowledge was first generated by supervised learning then by supervised learning followed by reinforcement learning, and finally by only reinforcement learning. This allowed programs to reach and surpass the level of world champion at the game of Go with the latest versions of the program AlphaGo. In particular, AlphaGo zero [3], which only uses reinforcement learning, did not need any knowledge to reach its level of play. This last success, however, required 29 million games of self-play (with 1,600 state evaluations per move). This approach has also been applied to chess. The resulting program broke the best chess program (which is based on minimax).

II. State of the art

In reinforcement learning, the term self-play describes a kind of Multi-Agent Learning (MAL) that deploys an algorithm against copies of itself to test compatibility in various stochastic environments. [4]

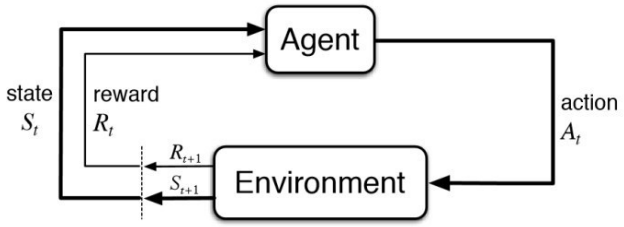


Fig. 1: Reinforcement learning principle.

Policy gradient methods are fundamental to recent breakthroughs in using deep neural networks for control, from video games, to 3D locomotion, to Go. But getting good results via policy gradient methods is challenging because they are sensitive to the choice of stepsize — too small, and progress is hopelessly slow; too large and the signal is overwhelmed by the noise, or one might see catastrophic drops in performance. They also often have very poor sample efficiency, taking millions (or billions) of timesteps to learn simple tasks.

Researchers have sought to eliminate these flaws with approaches like TRPO and ACER, by constraining or otherwise optimizing the size of a policy update. These methods have their own trade-offs — ACER is far more complicated than PPO, requiring the addition of code for off-policy corrections and a replay buffer, while only doing marginally better than PPO on the Atari benchmark; TRPO — though useful for continuous control tasks — isn't easily compatible with algorithms that share parameters between a policy and value function or auxiliary losses, like those used to solve problems in Atari and other domains where the visual input is significant. [4]

Algorithm 1 Proximal Policy Optimization

Initialize $Champion$ as the expert agent.
for $k \leftarrow 1$ to $NumberOfEpisodes$ **do**
  while $Performances \leq Threshold$ **do**
    $TrainingAgent$ trains against the Champion
  end while
end for

With supervised learning, we can easily implement the cost function, run gradient descent on it, and be very confident that we'll get excellent results with relatively little hyperparameter tuning. The route to success in reinforcement learning isn't as obvious — the algorithms have many moving parts that are hard to debug, and they require substantial effort in tuning in order to get good results. PPO strikes a balance between ease of implementation, sample complexity, and ease of tuning,

trying to compute an update at each step that minimizes the cost function while ensuring the deviation from the previous policy is relatively small (Algorithm 1). [5]

III. Environment

SlimeVolleyGym is a simple gym environment for testing single and multi-agent reinforcement learning algorithms [6]. The game is very simple: the agent's goal is to get the ball to land on the ground of its opponent's side, causing its opponent to lose a life (Figure 2). Each agent starts off with five lives. The episode ends when either agent loses all five lives, or after 3000 timesteps has passed. An agent receives a reward of +1 when its opponent loses or -1 when it loses a life. This environment is based on Neural Slime Volleyball, a JavaScript game created in 2015 that used self-play and evolution to train a simple neural network agent to play the game better than most human players.

Fig. 2: SlimeVolley environment.

IV. Self-Play methods

We have shown that standard RL algorithms can defeat the baseline policy by simply training agents to play from scratch directly against the built-in opponent. But what if we didn't have an expert opponent to begin with to learn from? With self-play, we train agents to play against a version of itself (either a past version for the case of PPO, or a sibling in the case of a genetic algorithm (GA)), so they can become incrementally better players over time. We also want to measure the performance of agents trained using self-play against agents trained against the expert.

A. Self-Play Proximal Policy Optimization

Reinforcement learning can also incorporate self-play, by incorporating in the environment an earlier version of the agent, allowing the agent to continually learn to improve against itself. This approach also leads to a natural curriculum that adapts to the agent's current abilities, because unlike starting out against an expert opponent, here, the level of difficulty will be on par with the agent. An outline of a self-play algorithm for RL (Algorithm 2).

B. Self-Play Genetic Algorithms

While self-play has gained popularity in Deep RL, it has actually been around for decades in genetic algorithms [7] in the evolutionary computing literature. It is also really easy to implement–our example consists of a dozen or so lines of code that implements it (Algorithm 3). For demonstration purposes, we are going to use the simplest version of tournament selection GA, without any bells and whistles.

Algorithm 2 Self-Play Proximal Policy Optimization

Initialize $ChampionList$ as a list containing a random policy agent.
for $k \leftarrow 1$ to $NumberOfEpisodes$ **do**
  $Opponent \leftarrow$ load the most recent agent archived in the $ChampionList$ inside the Environment.
  while $Performances \leq Threshold$ **do**
    $TrainingAgent$ trains against the Opponent
  end while
  $ChampionList$.insert($TrainingAgent$)
end for

Algorithm 3 Tournament Selection by Genetic Algorithm

Initialize $Population$ as a set of agents with random initial parameters.
for $k \leftarrow 1$ to $NumberOfGames$ **do**
  $players \leftarrow$ two random players from $Population$
  $match \leftarrow$ players play against each other
  if $match$ is tied **then** add a bit of noise to the second agent's parameters.
  else
    Replace the loser with a clone of the winner, and add a bit of noise to the clone.
  end if
end for

V. Results

In our example, the current PPO agent must achieve an average score above 0.5 against the previous champion in order to become the next champion. After running the self-play code, the training converged at around 140 generations (it takes a long time after that to produce the next Champion). The most recent champion is then evaluated for the first time against the baseline policy achieving an average score of -0.371 ± 1.085 over 1000 episodes. We also logged all previous Champion policies and evaluated those as well against the baseline policy to retroactively measure its training progress (Figure 3):

[8] discuss alternate ways to sample from the history of archived agents, since setting it to the most recent opponent may lead to the agent specializing at playing against its own policy. This may explain the fluctuations observed in the performance chart. In our implementation, we modified the call back evaluation method in stable-baselines to assign a number label to each champion, so if we wanted to, we can implement such sampling methods, and we leave it as a challenge to the reader to experiment with variations.

In the actual code, we track how many generations an agent has survived for (e.g. its evolutionary lineage), as a proxy for how good it is within the population without actually computing who is best to save time. We ran this code for

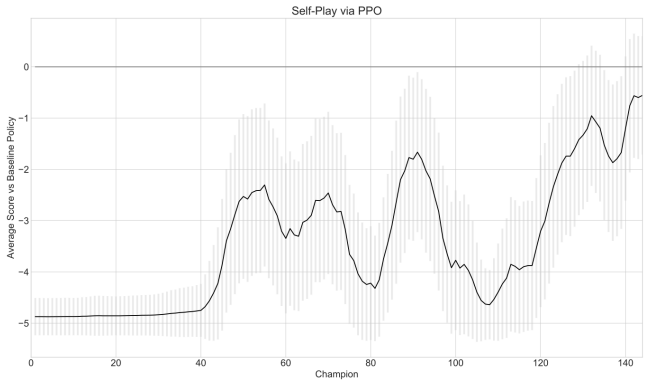


Fig. 3: Self-Play via PPO

500,000 tournaments on a single CPU which took only a few hours. Due to the simplicity of this method, the wall clock time per step is surprisingly fast compared to RL or even other ES algorithms. As a challenge, you may want to try implementing an asynchronous parallel processing version of this algorithm for performance. After 500K games, the agent in the population that had the longest evolutionary lineage is used as a proxy for the best agent in the population. It played against the original baseline policy for the first time, and achieved an average score of 0.353 ± 0.728 over 1000 episodes. We also logged the historical agent parameters during the tournament selection process, and evaluated each of them against the baseline policy afterwards to get a sense of the improvement over time (Figure 4):

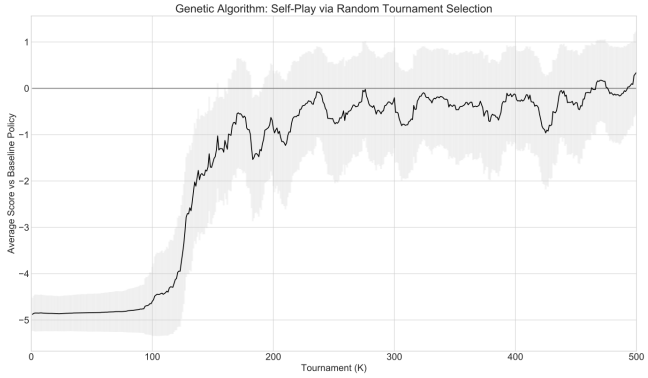


Fig. 4: Self-Play via Genetic Algorithm

VI. Conclusion

Thus, Self-Play Genetic Algorithm performed slightly better than Self-Play Proximal Policy Optimization but both proved that Self-Play learning has potential. Its most interesting use case is when we do not have access to an expert to train the agent. Moreover, even when there is an expert, the agent risks over-fitting against this expert and it is therefore better to vary the encounters. Applications of these techniques have been made by OpenAI in simulation for humanoid robots and the results are very encouraging.

References

[1] T. Cazenave, Y.-C. Chen, G.-W. Chen, S.-Y. Chen, X.-D. Chiu, J. Dehos, M. Elsa, Q. Gong, H. Hu, V. Khalidov *et al.*, "Polygames: Improved zero learning," *ICGA Journal*, vol. 42, no. 4, pp. 244–256, 2020.

[2] Q. Cohen-Solal, "Learning to play two-player perfect-information games without knowledge," *arXiv preprint arXiv:2008.01188*, 2020.

[3] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[4] A. DiGiovanni and E. C. Zell, "Survey of self-play in reinforcement learning," *arXiv preprint arXiv:2107.02850*, 2021.

[5] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[6] D. Ha, "Evolving stable strategies," *blog.otoro.net*, 2017. [Online]. Available: http://blog.otoro.net/2017/11/12/evolving-stable-strategies/

[7] B. L. Miller and D. E. Goldberg, "Genetic algorithms, tournament selection, and the effects of noise," *Complex Syst.*, vol. 9, 1995.

[8] T. Bansal, J. Pachocki, S. Sidor, I. Sutskever, and I. Mordatch, "Emergent complexity via multi-agent competition," *arXiv preprint arXiv:1710.03748*, 2017.

Using winding numbers in 3D reproduction

I.A. Witt[1],

1. *isaac-andrei.witt@ensta-bretagne.org, ENSTA Bretagne, Brest, France*

Abstract

Following the discovery of the article by Jacobson A., Kavan L. and Sorkine-Hornung O. on the segmentation of a 3D representation using winding numbers, we attempt in this article to implement a similar method in Python based on the contours of an image.

Keywords: Winding numbers; 3D reproduction; inside-outside segmentation

Introduction

In the field of 3D reproduction, there are many ways to try to design a 3D object. However, practice shows that in order to have a reconstruction that remains faithful to reality in various applications (e.g. 3D animation), it is better to construct a 3D object by visualising it as a set of volumes. We therefore based our work on a study by Jacobson A., Kavan L. and Sorkine-Hornung O. presenting a 3D design method using winding numbers, i.e. the number of times a contour encircles a point [1] [2]. This makes it possible to consider cavities or solid areas on a 3D body (e.g. an eye or an organ). Also in this study we tried to implement a new 3D reconstruction method using windings numbers.

Winding Numbers

The winding number of a point in a picture corresponds to the number of times a point is surrounded by a contour. This index can be negative because we take into account the direction of progression of the contour (by convention, a contour is established according to the trigonometric direction).

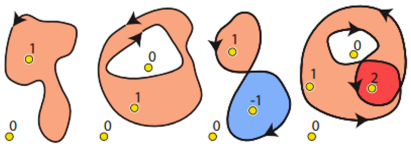


Figure 1: Calculate windings numbers [1]

By definition, for a contour C (in two dimensions), this index is calculated as follows with dtheta an infinitesimal cutting of C :

$$w(\mathbf{p}) = \frac{1}{2\pi} \oint_C d\theta$$

Figure 2: Formula for calculating the winding number for a two-dimensional contour [1]

Also we can generalize this formula for a 3D envelope S to :

$$w(\mathbf{p}) := \Omega(\mathbf{p})/4\pi \quad \text{,avec } \Omega(\mathbf{p}) = \iint_S \sin(\phi)\, d\theta d\phi$$

Figure 3: Formula to calculate the number of windings for an envelope (in three dimensions) [1]

2D Algorithm

In this study, we will suggest an implementation of an algorithm capable of determining a winding map from an image. To do this, I propose to work with the contour detection function of the open cv library, which returns a list of points corresponding to a contour of the image. However, this is not enough, indeed it is also necessary to take into account the direction of implementation and the contours of fine thickness which the detection function tends to separate in various segments. So I created a new Contours object, with various attributes and special basic functions. As far as the direction is concerned, I decide to implement a function calculating the arctangent2 between 2 different vectors created according to 3 different points of the contour in order to know the evolution of the contour when I go through the list following its stacking

direction and to be able to put it back in the right orientation. I also implement a function of addition and calculation of distance between two contours, merging two contours considered too close, which could be useful later. In order to establish a winding map, we will discretise our contours by subsampling them, but keeping the start and end points, so that we can process the open contours and apply the formula below for each point of the image:

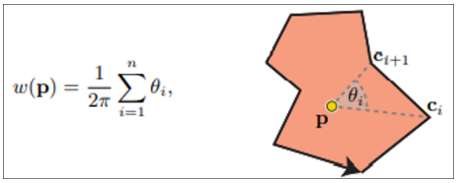


Figure 4: Discretization of the winding number calculation to implement it in 2D [1]

Results

After having extracted the contours of the image thanks to the opencv library and implementing a contour class, we sample these contours and for each point of the image we are the angle of the discretized shapes surrounding the point, as proposed before. The more the image tends towards orange, the higher the wind number of the points in the area. The white points correspond to the points whose angles could not be calculated by the algorithm.

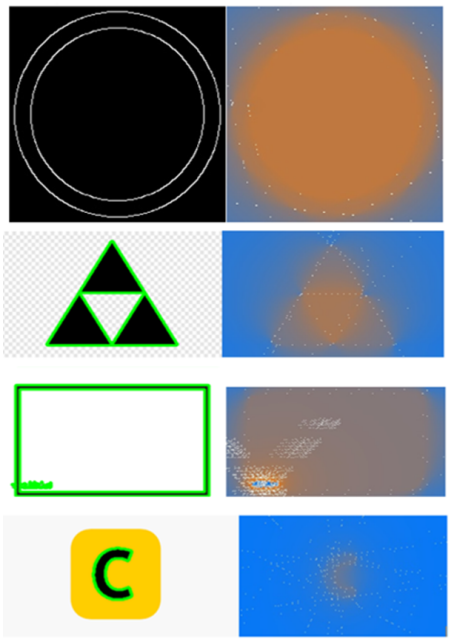


Figure 5: Application results with python in 2D : on the left edge detection and on the right we can see orange shadows left by the algorithm corresponding to a high winding number

Comments

The presence of numerous white spots can be noticed, mainly in the extension of the contours. To avoid these phenomena, we could apply a medium filter, even if it means making the algorithm a little more approximate. Moreover, we have a very high complexity which depends on the number of contours, the proposed discretization and the size of the picture.Paradoxically, with python (and the capacities of a classic computer) regular pauses are necessary for its good functioning and to avoid Runtime errors (Complexity $O(shape_{picture} * number_{contours} * size_{simple})$).

Nota Bene

Another way of calculating winding numbers is to use the PyMesh library, but the disadvantage is that this forces us to decompose the picture with Meshlib ways and therefore limits our possibilities.[3]

3D Algorithm

Where Jacobson A., Kavan L. and Sorkine-Hornung O. presented two implementation methods, a so-called naive one using the discretization formula below, and a so-called hierarchical method, more approximate but having a better complexity, aiming at conceptually expressing our mesh as a union of manifold patches (a topological space that near each point resembles Euclidean space), I suggest to visualise a 3D object as a succession of 2D images and the aim of this study would be to study the application of different filters according to the direction orthogonal to the plane of the images (in other words, we would visualise an object as a succession of radio).

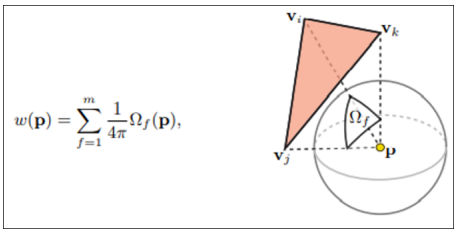


Figure 6: Discretization of the winding number calculation to implement it in 3D [1]

Conclusion

Unfortunately, in the time available, the transition from 2D to 3D could not be envisaged. The 2D implementation, even if it offers interesting results, is optimisable both in terms of results and complexity. We could prefer approximation and focus only on the points surrounding the contours and study the barycentres of the points.

References

1. Jacobson A. KLSHO. Robust Inside-Outside Segmentation Using Generalized Winding Numbers. ACM Transactions on Graphics, 32 2013; 33. DOI: 10.1145/2461912.2461916
2. Yger A. Analyse complexe et distributions. Paris: Ellipses, 2001
3. team P development. PyMesh. Available from: https://github.com/PyMesh/PyMesh/blob/main/python/pymesh/winding_number.py