Rob 2019

UV 5.7 - Predictor

Predictor

March 16, 2019

Abstract

In a context of industrial AUV (Autonomous Underwater Vehicle) applications, engineers need to predict, before the mission, that their robot will not be in a forbidden area or lost.

This project aim to propose several methods able to guarantee margins of position error for submarine application (AUV).

Several solutions have been developed. Some aim to quantify the AUV position uncertainty in order to define a maximum error at the end of the mission. Others show that the robot is able to go through a cycle to infinity without ever getting lost. Finally a method is proposed to determine a navigation zone which we are certain that the robot never goes out.

Contents

1 Introduction

When dealing with the propagation of uncertainties in dynamical systems, we may consider three types of observers [11]:

- Filters. Their goal is to estimate the state of the system taking into account all data collected in the past.

- Smoothers. They provide an estimation of the state taking into account data that have been collected in the past but also in the future.

- Predictors. At the initial time, a predictor provides an estimate of the future states assuming that the initial state is known. No measurements can be used since they are not available yet.

In a context of industrial AUV [2] (Autonomous Underwater Vehicule) missions, engineers need to know, before the mission, that their robot will not be in a forbidden area or lost. This project aim to propose several methods able to guarantee, thanks to predictors, position error margins for submarine application (AUV). We developed a tool that certify that a mission made with these robots will satisfy some given specifications such as:

- The robot will never enter in a forbidden zone,
- The robot will surface in a given disk
- The robot will always know its current location with an accuracy smaller than 10 meters.
- The robot will see a given landmark so that it will be able to relocalize.
- These robots will be close enough to communicate at a given rate.

Several solutions have been developed. Some aim to quantify the AUV position uncertainty in order to define a maximum error at the end of the mission. Others show that the robot is able to go through a cycle to infinity without ever getting lost. Finally a method is proposed to determine a navigation zone which we are certain that the robot never goes out.

Each solution is developed in simulation, and tested thought tests in real condition in a second time. Both simulations and real applications will work on the same kind of mission, in order to be able to compare the results.

2 Prediction with a Kalman filter

2.1 Context

An AUV (Autonomous Underwater Vehicle) is an autonomous submarine is often used for exploratory missions without requiring any human intervention. To accomplish its mission, an AUV must determine its position for a long period of time using its exteroceptive and proprioceptive sensors [3]. During our example mission, the AUV has a GPS and an inertial measurement unit and must find its path along an archipelagos composed of several islands. The robot can only receive GPS data when it surfaces, when it is close to an island, and can only count on its inertial measurement unit when it moves underwater from one island to another. Thus, we expect the position accuracy of the submarine to worsen as it stays underwater and to greatly improve once it reaches the surface. The objective of our group of study is to quantify the worsening of the submarine position accuracy for a given mission before it actually takes place.

2.2 Objective

We aim at implementing a Kalman filter [5] to predict the shape of the AUV position error ellipsoid at the end of a given mission.

2.3 Method

The AUV can be modeled by the given set of equations :

$$\begin{cases} \mathbf{X}_{k+1} = A_k \mathbf{X}_k + B_k \mathbf{U}_k + \alpha_k \\ \mathbf{Y}_{k+1} = C_k \mathbf{X}_k + \beta_k \end{cases} \tag{1}$$

Where the first equation is the state evolution equation and predicts the position of the AUV at a future state given a current state. The second equation is the observation equation and is used to correct the AUV position using the data provided by on-board sensors.

Here, the state vector $\mathbf{X}$ stands for the following :

$$\mathbf{X}_k = \begin{pmatrix} x \\ y \\ v \end{pmatrix} \tag{2}$$

$$\mathbf{Y}_k = \begin{pmatrix} x \\ y \end{pmatrix} \tag{3}$$

The value of the given matrices are the following :

$$\mathbf{A}_k = \begin{pmatrix} 1 & 0 & dt * cos(\theta) \\ 0 & 1 & dt * sin(\theta) \\ 0 & 0 & 1 - dt \end{pmatrix} \tag{4}$$

$$\mathbf{C}_{k,GPS} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \tag{5}$$

One can argue that the heading θ is not explicitly stated in the state equations. It is no wonder as this project aims at realizing a prediction of the uncertainty evolution over time and not a simulation of the system. In other words, the robot's trajectory is supposed to be perfect (a.k.a the "field truth" or "vérité terrain") and only the standard deviation from this perfect trajectory is studied here. As the standard deviation is independent from the robot's heading, the heading θ's update implemented in the proposed algorithm only has a plotting purpose. At each iteration, the heading θ is updated with the angle between two consecutive islands and has no influence over the uncertainty accuracy.

2.4 Parameters

- Sensors' noise matrix (Γ_β)
  Once the AUV surfaces to receive GPS data, it becomes able to narrow down its position given a certain error defined by the covariance matrix Γ_β called the Observation Matrix. This matrix quantifies the variance of the (x,y) position accuracy provided by the GPS. The standard deviation of this error was provided by the real AUV study group and has a value of 0.48m as stated in the GraalTech submarine datasheet. When the robot is underwater however, the Kalman filter switches to predictive mode and the observation matrix becomes null.

- State vector's noise matrix (Γ_α)
  This covariance matrix characterizes the error made by the Kalman filter when predicting the future position

from the current one. It is system dependent and must be calculated from experiments on the actual AUV. The parameters of this matrix can be estimated by repeatedly performing the same mission and characterizing the variance of the dispersion of the AUV final positions with respect to the field truth.

- Islands' position : lm
  Although the position of the islands is not directly necessary for calculating error evolution over time, it is necessary to specify the mission of the AUV, i. e. the distances covered and the times the GPS packets are to be received. To simplify the input of missions to the user and to use the same structure of inputs as the other groups, it was decided to directly enter the positions of the islands to specify the position of the AUV.

- Initial uncertainty matrix: G_up
  This matrix quantifies the initial position error of AUV in position (x,y) and velocity v. In our case, we assume that we know perfectly the initial state vector of AUV ($G_up = 0$).

- Iteration step: dt
  Must be chosen relatively low.

2.5 Outputs

From the algorithm, we obtain 3 *matplotlib* displays:

- Display of the AUV trajectory with confidence ellipsoids:
  The robot's trajectory is represented by the black lines forming the mission triangle and is confused with the truth of the terrain. Tthe confidence ellipsoids are blue circles centered on the robot's position at each iteration.

- Display of positioning error as a function of time for dead reckoning mission, and for DVL mission:
  It can be seen that this error is increasing on the majority of the displacement except for the few moments when the AUV resurfaces to capture its GPS position where there is a sudden drop in uncertainty. It can also be seen that uncertainty initially grows as a square root characteristic of a dead reckoning location estimation and then increases linearly over time as the velocity uncertainty stabilizes. The uncertainty is represented by two elements : the standard deviation in orange, and the variance in blue.

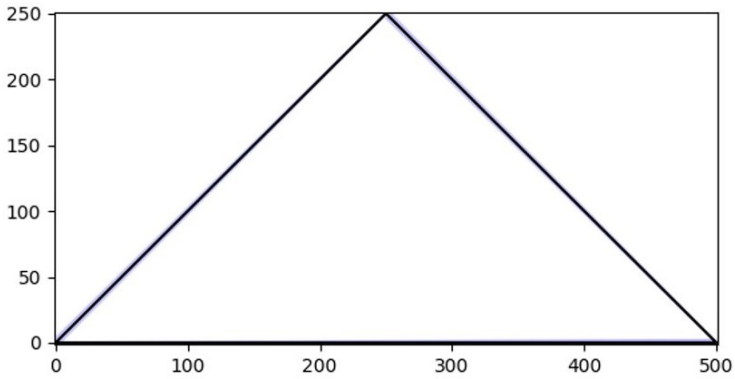


Figure 1: Trajectory of the mission

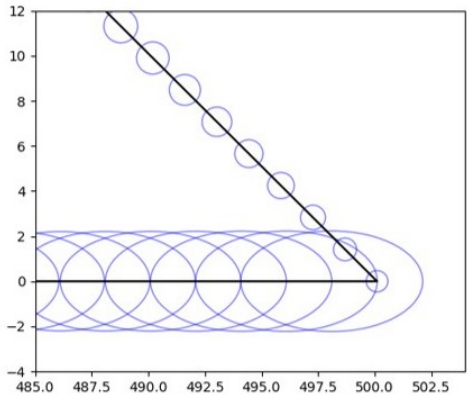


Figure 2: Zoom on a part of the trajectory

2.6 Results

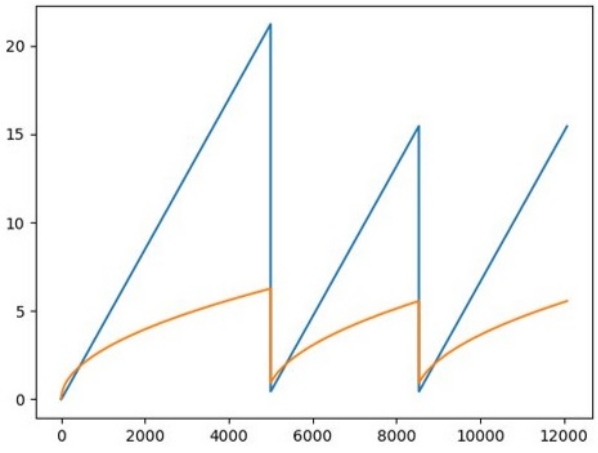


Figure 3: Uncertainty for a mission using a dead reckoning method

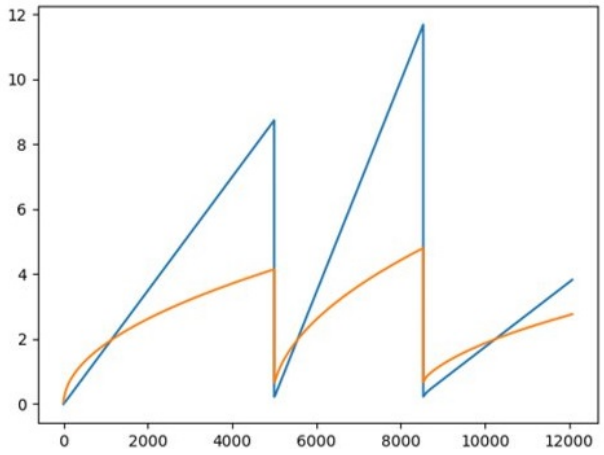


Figure 4: Uncertainty for a mission using a DVL

3 Particle method

3.1 Introduction

The particle method [11] consists in applying a same state equation and command to several systems characterized by different noise and sharing the same initial position so as to note different behaviours on a large scale. Each particle represent a robot.This method allows to predict a main behaviour adopted by most of the particles. It is also the opportunity to detect unexpected behaviours. The particle method is based on two main concepts: the law of large numbers and the Monte Carlo method.

3.1.1 Law of Large Numbers

The law of large numbers reflects that the behaviour of a large number of random individuals in a population is close to the general behaviour of that population.
Low Law of Large Numbers:

$$\lim_{n \to +\infty} P\left(\left|\frac{\mathbf{X}_1 + \mathbf{X}_2 + ... + \mathbf{X}_n}{n} - E(X)\right| \leq \epsilon\right) = 0 \tag{6}$$

Strong Law of Large Numbers:

$$P\left(\lim_{n \to +\infty} \left|\frac{\mathbf{X}_1 + \mathbf{X}_2 + ... + \mathbf{X}_n}{n}\right| = E(X)\right) = 1 \tag{7}$$

3.1.2 Monte Carlo Method

The Monte Carlo method is a method for calculating an approximate numerical value using random processes. It's a stochastic method.

For example, this method can be used to estimate the area of an area in a known area. To simplify, we will take a rectangular sector with a random zone whose area is to be determined (see figure 8). Thanks to measurements on the sides of the main area, the area of the rectangle is known. To find the blue area, X particles are randomly sent over this total area. Then count the number N of particles arriving on the green zone corresponding to the total zone minus the blue zone. So we know that X-N particles have arrived in the blue zone. The following reports are obtained:

$$\frac{\mathbf{Area}_{total}}{\mathbf{Area}_{blue}} = \frac{X}{X - N} \tag{8}$$

donc

$$\mathbf{Area}_{blue} = \frac{X}{X - N} \times \mathbf{Area}_{total} \tag{9}$$

3.2 Missions

The particular method is an estimation method based on simulations. This method is used and tested through two specific missions, presented in this section.

3.2.1 First mission

For each particle (i.e. each AUV) the first simulation will be the following:

- For the first 60 seconds, the AUV locates it-self using dead-reckoning, and is directed towards one point. The dead-reckoning is a Kalman filter in prediction mode only.

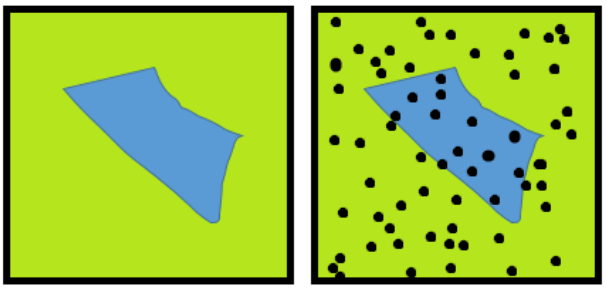


Figure 5: Estimation of an area by Monte Carlo Method

- After 60 seconds, the AUV emerges and gets a GPS fix.
- The position given by the GPS is then fused with the dead-reckoning estimated position via a Kalman filter.
- The GPS position being more precise than the dead-reckoning estimation, the new estimated position is more accurate and the associated covariance matrix is smaller than the previous one.
- Then, the AUV turns back and goes towards its initial position for 60 seconds. Here, the transition is instantaneous and will be more sophisticated in the second simulation, taking into account the necessary heading regulation.
- After the 60 seconds are passed, we check the AUV covariance matrix and estimated position.

3.2.2 Second Mission

3.3 Equations

Each particle i has the following discretized state equations :

$$\begin{cases} \mathbf{x}_{k+1,i} = A_{k,i}\mathbf{x}_{k,i} + B_{k,i}\mathbf{u}_{k,i} + \alpha_{k,i} \\ \mathbf{y}_{k+1,i} = C_{k,i}\mathbf{x}_{k,i} + \beta_{k,i} \end{cases} \tag{10}$$

where $\alpha_{k,i}$ and $\beta_{k,i}$ are random white noises and we use their variances to fix the noise covariance matrix of the Kalman filter Γ_α.

More precisely, we have:

$$\begin{cases} \dot{x} = vcos(\theta) + \alpha_1 \\ \dot{y} = vsin(\theta) + \alpha_2 \\ \dot{v} = u - v + \alpha_3 \end{cases} \tag{11}$$

And then we update the state vector $\mathbf{X} = (x, y, v)^T$ with:

$$\mathbf{X}_{k+1} = \mathbf{X}_k + dt\dot{\mathbf{X}} \tag{12}$$

During the dead-reckoning period, the observation equation is null:

$$while \;\; k < 60s : y_{k,i} = 0 \tag{13}$$

At this stage, the estimated state vector and covariance matrix are given by:

$$\begin{cases} \hat{\mathbf{x}}_{k+1,i} = A_{k,i}\hat{\mathbf{x}}_{k,i} + B_{k,i}\mathbf{u}_{k,i} \\ \Gamma_{k+1,i} = A_{k,i}\Gamma_{k,i}A_{k,i}^T + \Gamma_\alpha \end{cases} \tag{14}$$

At 60 seconds, we get:

$$when \;\; k = 60s : y_{k+1,i} = Cx_{k,i} + \beta_{k,i} \tag{15}$$

3.4 Simulation

The simulation part will allow to assess the behaviour of the equation to simulate through particle method.

3.4.1 Algorithms

We implemented and run the entire simulation in Python. But as python is too slow we can not simulate a lot of robots. In order to simulate way more robots we implement the particle method in C++.

With this method we are able to simulate 500 robots during 1200 seconds, in a reasonable execution time, few minutes. The program writes a text file containing the positions of each robot at each iteration.

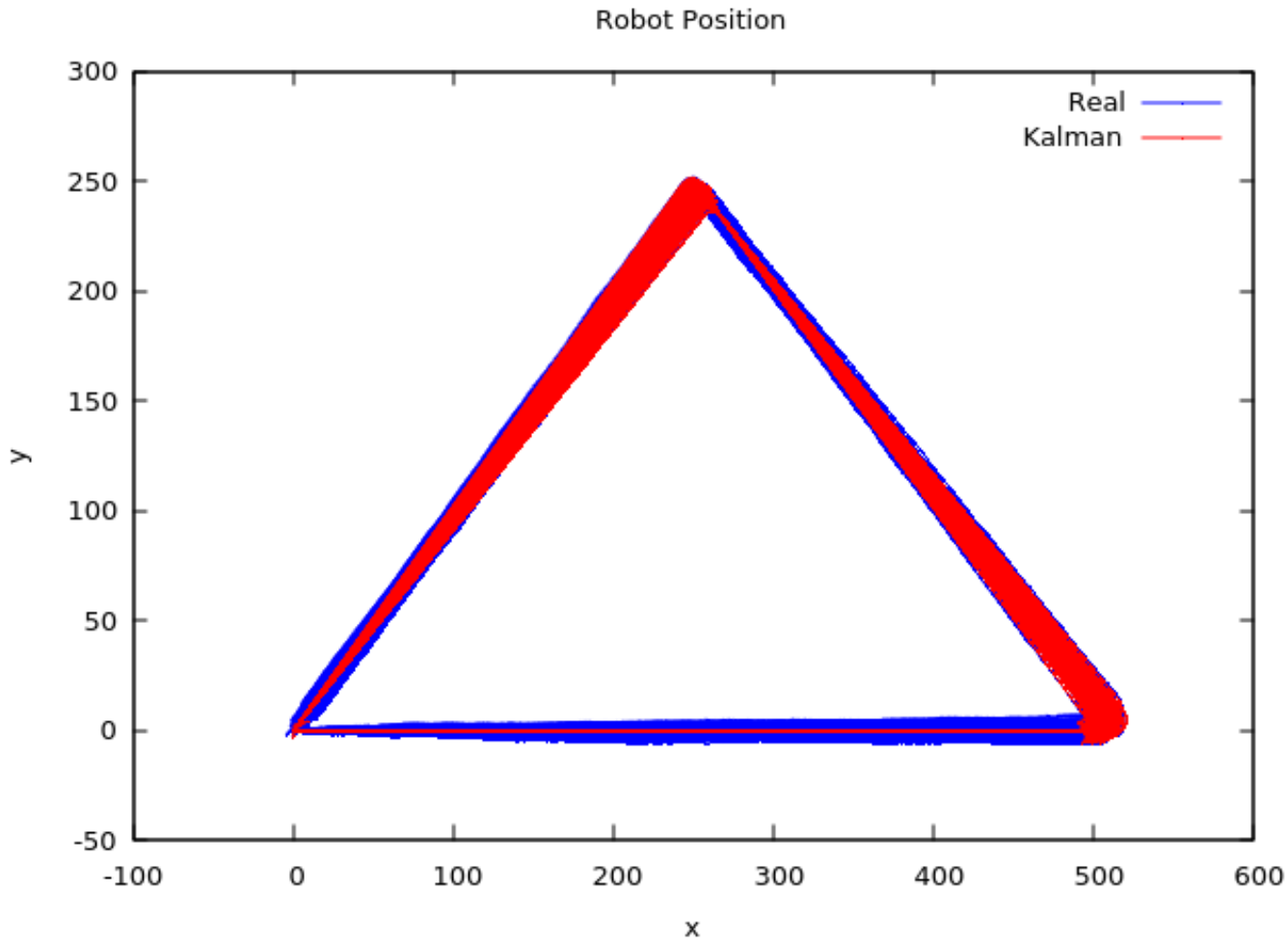


Figure 6: Results of the simulation

Then the text file will be processed (parse) by a python program in order to be launched in Unity-Vibes. That way we will see the different particles moving in a 3D environment. We will then see a cloud of particles.

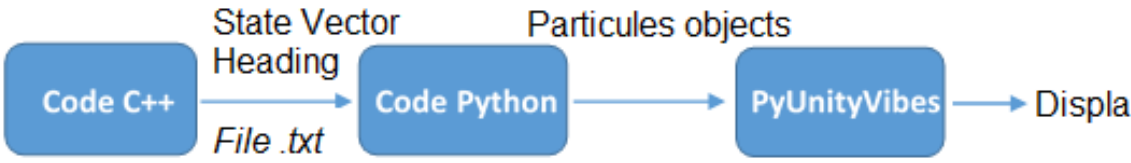


Figure 7: Organization of particular group

3.4.2 Architecture

The particles were chosen to be represented as Particle objects through object-oriented programming in Python.

The particle object is characterised by six instance variables :
-X : the state vector of the particle comprising the coordinates of the particle and its speed
-U : the input vector of the particle comprising the speed input and the heading input of the particle
-cov : the covariance matrix of the particle
-theta : the heading of the particle
-auv : the graphic object of the particle that will be displayed

3.4.3 Display

We chose to display the particles through the 3D-view simulator Unity-Vibes created by Rémi RIGAL (https://github.com/RemiR
Vibes).

To use it throught a Python script, we need to use PyUnityVibes, a Python version of UnityVibes created by Noélie RAMUZAT (https://github.com/NoelieRamuzat/PyUnityVibes).

Installation:

- In a terminal : pip3 install PyUnityVibes
- Install the binary application suitable for your OS at : https://github.com/RemiRigal/Unity-Vibes/releases

To display the AUV through PyUnityVibes, it is just required to choose a pattern like the one that can be seen below. Then, PyUnityVibes just needs the 3 position coordinates and the 3 angles of the AUV to display it properly.

Figure 8: Submarine pattern

3.4.4 Results

The results of the simulation will allow to check the theory of particle approach.

First of all, after only a small time of simulation, the simulations begins to show different trajectories for each particle. Indeed, it can be noted that the submarines displayed by UnityVibes are not superimposed anymore as it can be seen on the picture below. This is due to the random noise assigned to each particle.

Figure 9: submarine cloud

Now, considering the confidence ellipses of the different particles below, it can be noticed that the ellipse are getting bigger as the mission progresses. This is due to the Kalman filter that increases the uncertainties at each new iteration.

Afterwards, a new feature has been implemented so as to consider a new behaviour during the simulation: each submarine was allowed to perform one measure at some point during the trip, which is supposed to be equivalent to surfacing for submarines as their GPS are only available outside the water. After this measure, it can be noticed that the following ellipses became smaller. This can be explain by the fact that with a new measure, the particle got true information about its position and its speed and consequently, the uncertainty about the position and the speed has been reset.

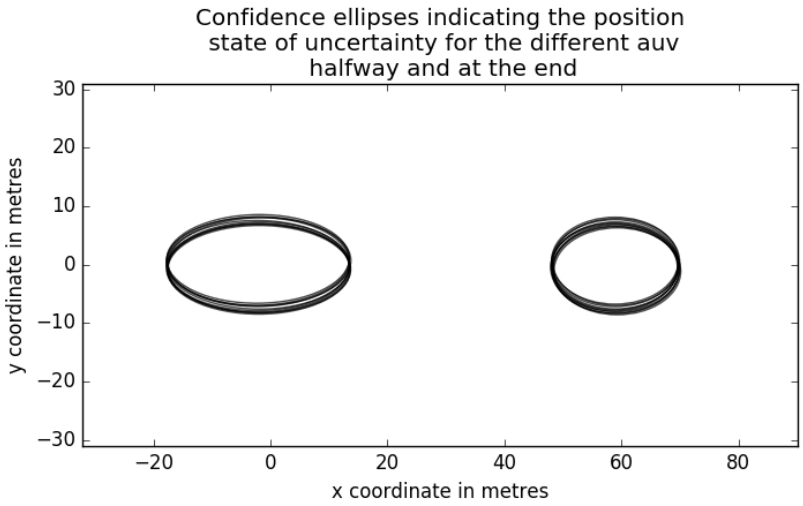


Figure 10: the 99 % confidence ellipses indicating the position state of uncertainty: halfway on the right and at the end of the mission on the left

4 Simulation of uncertain movements with Tubex

4.1 Tubex

Tubex is a C++ library, created by Simon Rohou and the installing and use instructions can be found easily [**?**]. It was made to work on dynamic systems (which evolve with time) defined by fixed constrains.

It is named after its primary tool: the tube [9], [10].

A tube is a set of intervals [7] on a temporal domain, with a know timestep, which represents a temporal evolution [1]. It is also possible to initialize the tube with a dynamic function.

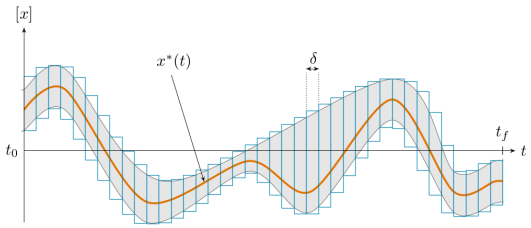


Figure 11: Example of tube [x] by time, in blue the intervals, in red the function

4.2 Objective

We chose to use Tubex to add another comparison to our simulations of the position accuracy. Thus, we created similar missions as those of the Kalman and particular filters.

Contrary to the Kalman filters, and because they take only the initial conditions and the temporal function as parameters, tubes are independent from the ground truth. Our goal is thus to simulate the AUV's movements, and check if they are invariant after some time. If so, it would mean that, in the end, we could predict a trajectory of boxes the AUV would never leave.

In our work, we use three different tubes for our missions:

- the theta (wanted yaw) command tube
- the x position tube
- the y position tube

Displaying the tubes x and y gives us the AUV's possible 2D trajectory, with the uncertainty counted in.

4.3 Equations

We reuse the following equations from the Kalman and the particular filters:

$$\begin{cases} \dot{x} = vcos(\theta) + \alpha_1 \\ \dot{y} = vsin(\theta) + \alpha_2 \end{cases} \tag{16}$$

Our speed v is considered invariant by simplicity and our alphas are nulls because Tubex computes uncertainties by itself.

4.4 Fist mission

4.4.1 Steps

The first mission is just a round trip to a known waypoint to the immediate rigth. Its steps are :

- AUV computes the yaw to the waypoint.
- The AUV goes toward this waypoint for 60s and its positional errors grow.
- The AUV surfaces and gets its GPS position. It allows the simulation to restrict the possible interval box in which the AUV could be.
- The AUV turns back to its origin position for 60s.

4.4.2 Results

Figure 12: Mission 1 : Round trip

In the preceding figure, The initial position box of our AUV is delimited in blue. The light green box is the estimated position of the AUV at 60s and the filled-in dark green one is the restriction found with the GPS measure. The red box is the final estimated position box.

Interval x	**Interval y**
[-1.012, 1.0922]	[-2.47552, 2.57336]
[-1.012, 1.0278]	[-2.52666, 2.52666]
[-1.012, 1.0922]	[-2.57336, 2.47552]

Table 1: Table of final boxes

To check if there was an invariance created with this mission, we repeated this round trip with differing and random GPS restrictions. The union of all our final boxes is: [-1.012, 1.0922] [-2.57336, 2.57336] . We can say that, with this mission, no matter where the AUV ends up after 60s,; it will be in this box at the end of the mission.

4.5 Second mission

4.5.1 Steps

The goal of the second mission is to know if it is possible to end with an invariant movement, no matter the AUV's trajectory. To explore this, we repeatedly loop a movement in our three-waypoints archipelago.

The algorithm for this is:

- For the number of archipelago loops wanted:
  - The AUV computes the yaw to the first waypoint to reach from its current position box.
  - The AUV moves toward this waypoint for a time t.
  - When the AUV supposedly reaches the waypoint, we divide its errored position box in sub-boxes.
  - For each sub-boxes the AUV computes the yaw to the next waypoint.

  - We keep the union of all the movements toward the second waypoint.
  - When the AUV reaches the second waypoint, we redivide the unioned positional box.
  - We compute the next yaws and once again keep the union of all movements toward the last waypoint.
- We keep the union of the tubes x and y done for one cycle.
- We begin a new cycle of the archipelago with the last unioned positional box as the new initial one.

The number of sub-boxes is computed by the diameter of the box and a fixed value representing the size of the AUV. This allows us to have a greater number of sub-boxes for bigger boxes, and thus a better precision in our model than propagating this full errored box. This is illustrated in figure 14: propagating sub-boxes, with each their own yaw, reduce the scope of our final tube.

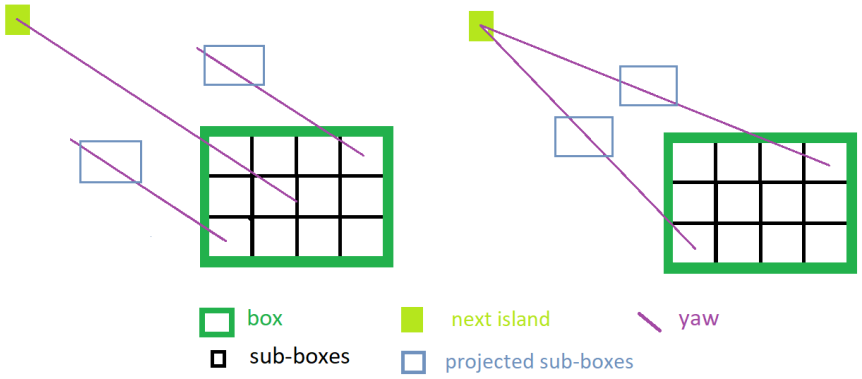


Figure 13: Sub-boxes interest

4.5.2 Results

In the following figures, full green boxes are our waypoints. The red box is the AUV's final positional box and its trajectory is traced in grey. The blue box in the second loop figure is its initial positional box (the previous final box).

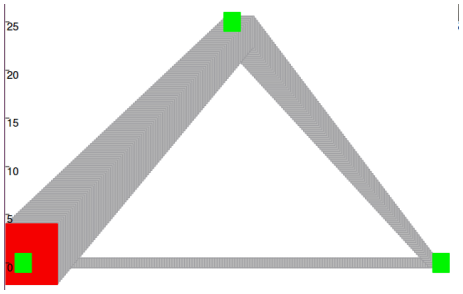


Figure 14: Mission 2 : First archipelago loop

As can be seen in these figures, the positional boxes keep growing during the cycle. While our sub-division of the boxes gives us a better precision, it is not enough to get an invariant movement.

This could be caused by too high uncertainties, or simply too few loops. Unfortunately, as the mission takes a lot of computing power, it is difficult to simulate a greater number of loops.

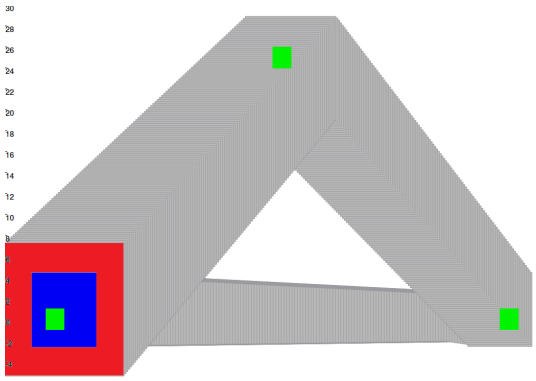


Figure 15: Mission 2 : Second archipelago loop

While we can't predict where the AUV will be after an unknown number of loops with this method, we can do it for short missions.

5 Focus on the navigation area

In the scope of this project, we want to be able to prove that we will not lose our AUV during the mission. Instead of running a simulation in which we estimate the error in order to validate our experiment [4], this section explains how we can obtain a mathematically proven solution over an area given an assumption.

5.1 Largest invariant set

If we make the following assumption:

Our AUV's state vector x follows the equation $\dot{x} = f(x)$

Then given a certain volume $\boldsymbol{X}$ in our state vector's space, we can calculate the largest positive invariant set of that volume. The largest positive invariant set $A \subset \boldsymbol{X}$ is described in mathematical terms as such:

$$if\, \exists\, t0\, such\, that:\, x(t0) \in A,\, then\, \forall t \geq 0,\, x(t0+t) \in A \tag{17}$$

This set therefore has the following properties: for any trajectory, if at any given time it meets the largest positive invariant set, all subsequent points of this trajectory will also be included in this set. Therefore we have a mathematical proof that our AUV's state vector is constrained in a bounded volume in the state vector space. Therefore the position of the AUV - which is a projection of the state vector on a 2D plane - is also constrained in space.

5.2 PyInvariant

5.2.1 Description

The tool We will be using - PyInvariant - allows us to calculate the largest positive invariant set described in the above section [6]. In order to use this tool, the state vector of the AUV - x - must be fully described by a function f such that $\dot{x} = f(x)$. We then give the function f and the area we wish to evaluate as arguments and the program returns a visualization with unstable areas shown in blue and stable areas shown in yellow. (see the example below)

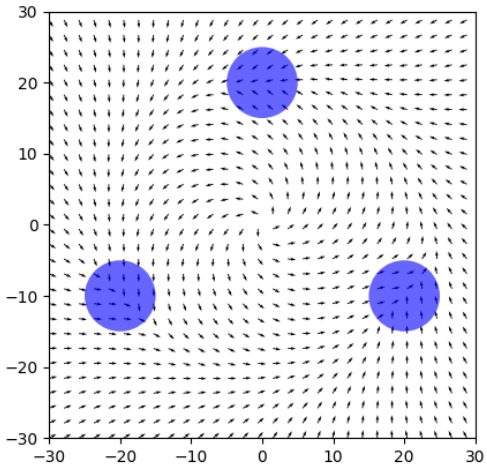
5.2.2 Limits

Unfortunately this software has only been coded to handle 2 dimensional state vectors for now. Our state vector x is therefore limited to representing the x and y coordinates of our robot in a plane.
Another limitation of this approach is that we are limited to commands that can be put in the form of a vector field, which substantially limits our possibilities.
Indeed, having commands that can only be expressed as a function of the robot's x and y coordinates is equivalent to saying that our command function is equivalent to a marine current. We are unable to take into account any form of complex control mechanism.

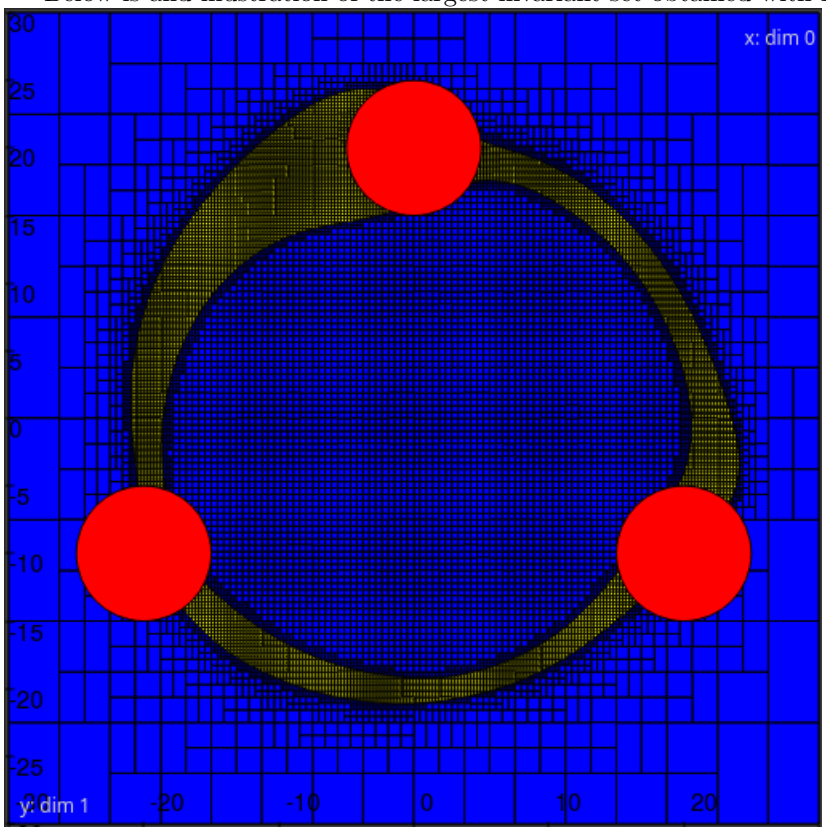
5.2.3 Example

In this example, we consider that our robot is subject to a rotating vector field - to simulate our robot going around an island - and when seeing points of interest in the space, it will aim for the coordinates of the next point of interest. The rotating vector field also has a repulsive component, which pushes the AUV outside of the area. If this was the only field in play, the robot would leave the area and never be seen again. The effect of the points of interests can be considered as that of a lens on the trajectory of the AUV, focusing its trajectory towards the next point of interest. We can model this behavior with a vector field as well.

Here is a visualization of the sum of the vector fields, with the points of interest located in the blue circles:



Below is and illustration of the largest invariant set obtained with PyInvariant :



This image shows in blue all the areas in which our AUV is sure to leave the selected window and in yellow all the areas within which the robot will stay constrained if its trajectory ever encounters it.

6 Simulation results

Testing the AUV directly in the sea results in a lot of difficulties. First, we need a place to test, then we need to calibrate the drone, finally the wind and the current does not help checking the results.

The simulation is then important to prevent useless real tests by allowing us to recognize possible development errors in the code in advance. The AUV uses a middleware, called MOOS, to integrate its different modules like sensors acquirers, data treatment and propellers commander. MOOS works by a method of publishers and listeners that provide the connections of this different parts used to control the AUV. Within this framework, it is in our best interest to do a simulation using the same middleware, in order to create a test environment that better represents the real one, also aiming to test the connection of different MOOS applications before using the model in the real context.

MOOS have in its standard library some applications that replace the real sensor acquirers and propellers commanders modules. This allow us to easily build a simulation context. Those applications are :

- pMarineViewer : Handle the visualization GUI to see how the drone is moving, allows to set the map to the geographical position of the mission
- uSimMarine : Handle the movement of the drone through its original position and the command sent to it
- pNodeReporter : Define the dynamic of the chosen drone, in our simulation the dynamic of the drone is different than the real one as we do not know the real dynamic of our drone.
- pMarinePID : Handle the command of the robot in depth, speed and yaw.

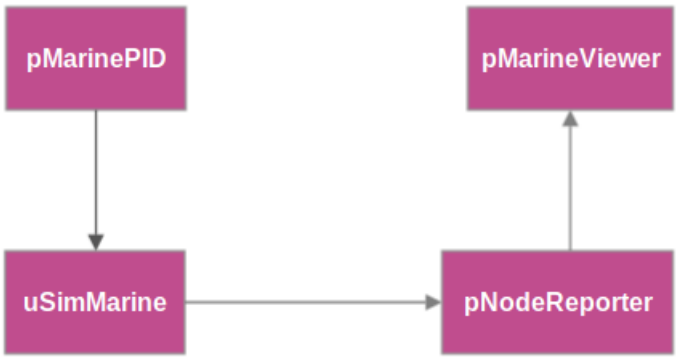


Figure 16: Connection between Moos applications

For our missions we also used the moos-application pHelmIVP. This application allows us to implement a state machine during the mission through defined behaviors. MOOS presents already a big variety of behaviors to be used, such as, waypoints tracking, area inspection and heading tracking. The main idea is that we define the states to be used and their entering conditions and based on that the action selection is autonomously done. The behaviors can actuate in two different domains, the speed and the heading of the vehicle.

Another important feature of MOOS is the current simulator, by using it, we can take current into account and have a better understanding of how our vehicle reacts in the real world and how different external aspects interfere in its performance. However, unfortunately, the wind can not be modeled and it is known that it can affect the drones navigation.

6.1 First Mission - Round Trip

The first mission we implemented is a back and forth following a heading. In this case we used the pHelmIVP application to create a state machine with three states, among them, one of the state is the inactive state. We start by following a given heading during a certain amount of time. The simulated heading tracking is implemented in pHelmIVP as a behaviour. Then, we take the GPS position of the drone. In a second time, the AUV goes back to its initial position through a second heading tracking.

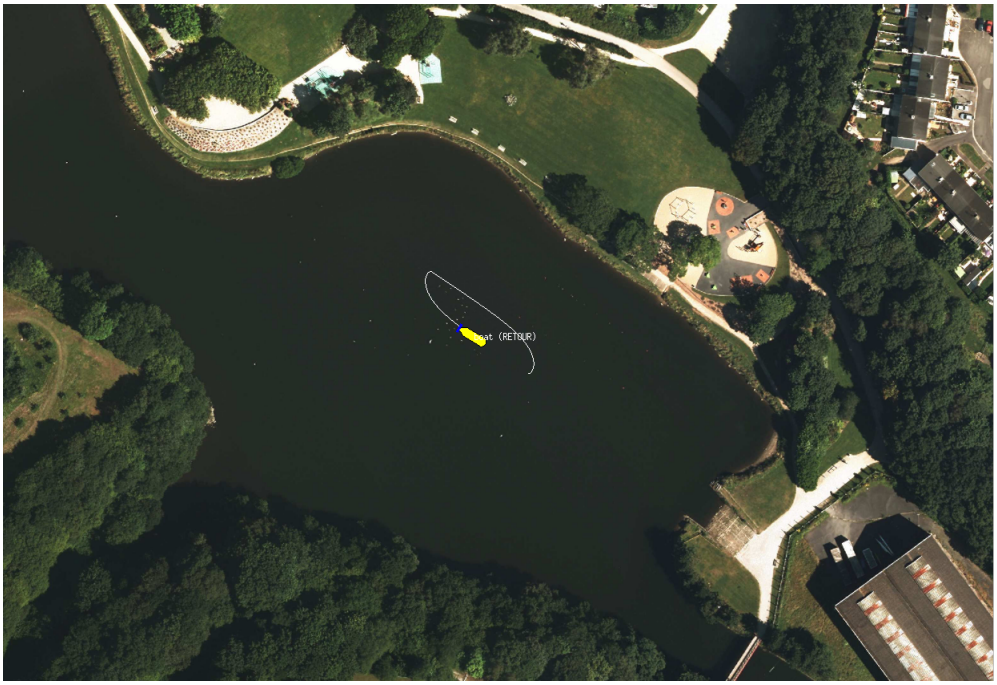


Figure 17: Auv during first mission

During the simulation, the heading tracking is done by the behavior already implemented in the moos library. For the real tests with the AUV, this behavior will be replaced by a moos application that acquires the data from the sensors and calculates the power to be given to the propellers that controls the heading.

During this first test, the vehicle's speed is considered constant and the used behavior commands it only in heading. In order to impose a constant speed during the two active states, we had to use an other behavior in parallel to the heading tracking. The behavior is called constantspeed and it maintains the speed constant. The following graphic allows us to have a better understanding of the behaviors implemented.

In the scheme, presented in figure 18, the red boxes represent the behaviors and the blue boxes represent the states.

6.2 The Second Mission - Triangle following

For the second mission, we tried to make the robot follow a triangle during an unlimited amount of time.

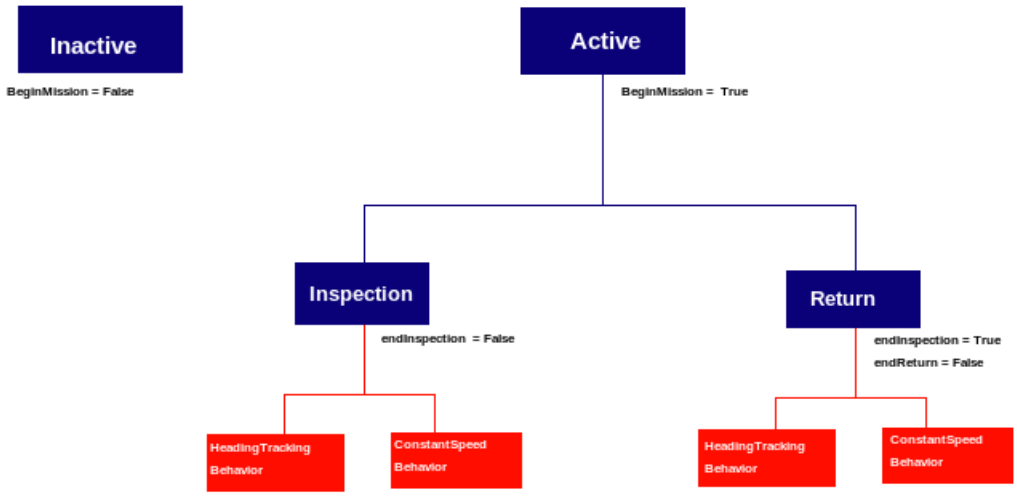


Figure 18: First Mission Diagram

Figure 19: AUV during second mission

We consider that its initial position is the first point of the triangle. Then, the robot is underwater and it navigates

at a given angle to go to the second point of the route. It follows the given heading and when it considers being at the good position, it surfaces and it takes its GPS position. Then, it calculates the angle to follow in order to arrive at the last point and repeat the loop. The goal is that after each iterations the drone is closer to the aimed point.

In a more accurate approach, it would know that it had arrived at the desired position by a fusion of sensors that would replace the GPS underwater. However, the development of this algorithm does not concern this part of the project and it is already cover in other sections of this report. In this case we set an amount of time for the underwater heading tracking before changing the state to mounting and acquiring the position.

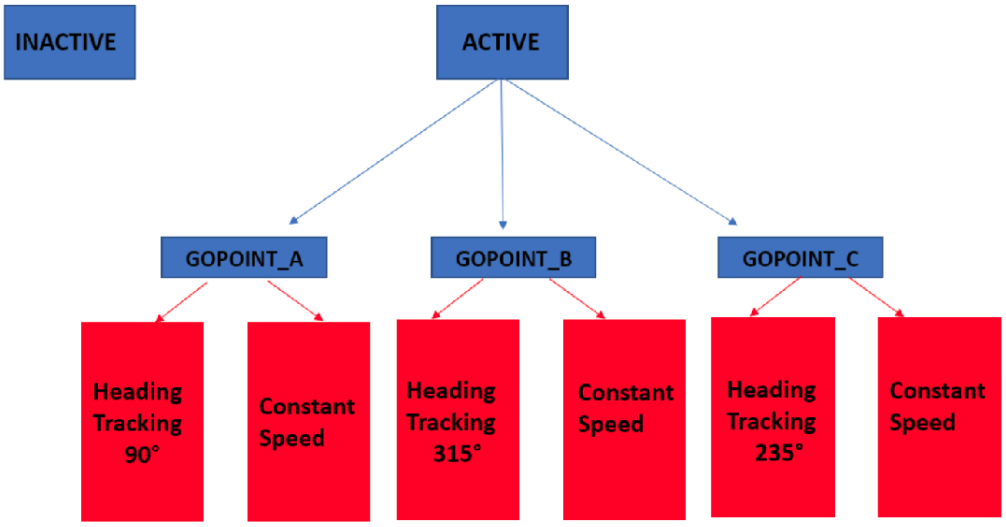


Figure 20: Second Mission States Diagram

This model developed has some conception problems. In the simulation, it works well, however it does not simulate the actual performance that the AUV would present in a real mission. This happens, first because we use the time passed in the state as a condition for changing, so it is not guaranteed that we will be in the desired position for that. The second point that it is not interesting is that by using behaviors in the application HelmIVP we can not command the robot in vertical speed, only in heading and longitudinal speed as it was already mentioned, this is not a huge problem but all the simulation was done on surface, and this will not happen in practice. The last, and most problematic point in this implementation is that the behaviors do not present a flexibility concerning the heading command, meaning that the heading to be maintained by each state (GOPOINT_A, GOPOINT_B,GOPOINT_C) will always be the same, as it is defined in the Figure 6.2.

We know that, in practice, after each iteration the robot will always arrive in a different position, see figure 6.2, comparing to the one achieved in the last turn, around the defined points A,B and C. Therefore, the direction to be followed should also be calculated at the beginning of each state considering the next point to be achieved and the actual point of the robot, this last value is not constant so the heading should not be either. So, because of its limitations, we decided to stop using the HelmIVP application to simulate the state machine, we contested that it works really well for simple simulations, however it is not capable of creating an environment that cover all the uncertainties of the real model.

6.3 The Third Mission - Triangle following Corrected

Our first approach, to simulate a more fair representation of our context, was to try to create new behaviors to be used with the MOOS application HelmIVP, since the behavior destined to heading follow, defined in the MOOS standard library, does not offer the flexibility we need. This task showed itself really complicated and not practicable given the time we disposed before the simulation on the sea.

In this case, we opted for creating a new MOOS application to replace HelmIVP as a whole. In this new application, we developed a new state machine in c++. The first state in this machine is responsible for mounting the AUV on the surface to acquire its GPS positions for a certain amount of time. The second state submerges the robot again and it fix its heading in order to reach the next point on the triangle. Finally, the last state realizes an heading follow during the time calculated previously to reach it. The time is calculated considering that we know the actual position of the robot, its speed and the position to be reached.

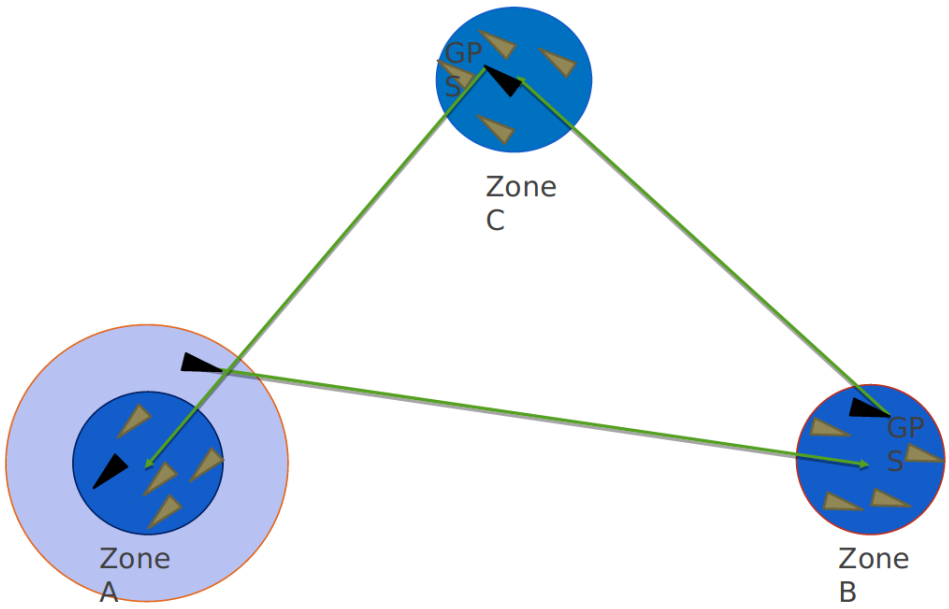


Figure 21: Uncertainty over arriving points

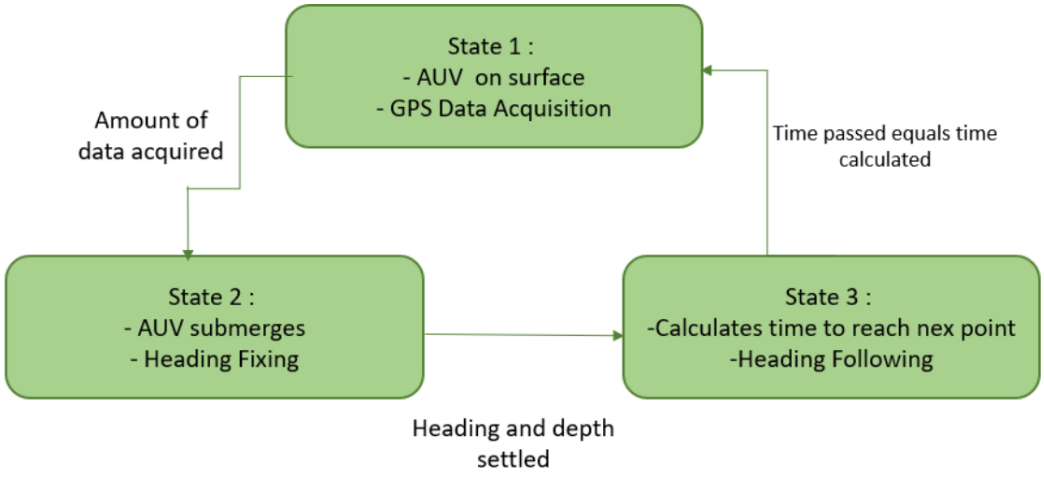


Figure 22: State machine of the last simulation

7 Results at sea

8 Conclusion

Predictor was the opportunity to develop several theoretical points in simulation in order to predict the evolution of our robot. All simulations focused on a triangle-shaped mission, which allowed comparisons to be made. Thus the Kalman method and the particle method seem to lead to the same estimation of uncertainty.Using these two methods, we seems able to quantify how much we will be lost at the end of the mission

The tubes made it possible to give a visual answer to the problem although no numerical data are obtained, and the invariances approach tackles the problem but only superficially. Each method deserves to be deepened and adapted to an industrial use, but the elements presented in this report show that an answer to the problem can be brought

List of Figures

References

[1] F. Le Bars, J. Sliwka, O. Reynet, and L. Jaulin. State estimation with fleeting data. *Automatica*, 48(2):381–387, 2012.

[2] T. Fossen. *Marine Control Systems: Guidance, Navigation and Control of Ships, Rigs and Underwater Vehicles.* Marine Cybernetics, 2002.

[3] L. Jaulin. *Mobile Robotics.* ISTE editions, 2015.

[4] T. Le Mézo L. Jaulin and B. Zerr. Bracketing the solutions of an ordinary differential equation with uncertain initial conditions. *Applied Mathematics and Computation*, 318:70–79, 2018.

[5] R. E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the AMSE, Part D, Journal of Basic Engineering*, 82:35–45, 1960.

[6] T. Le Mézo, L. Jaulin, and B. Zerr. An interval approach to compute invariant sets. *IEEE Transaction on Automatic Control*, 62:4236–4243, 2017.

[7] R. E. Moore. *Interval Analysis.* Prentice-Hall, Englewood Cliffs, NJ, 1966.

[8] S. Rohou. *Reliable robot localization: a constraint programming approach over dynamical systems.* PhD dissertation, Université de Bretagne Occidentale, ENSTA-Bretagne, France, december 2017.

[9] S. Rohou, L. Jaulin, M. Mihaylova, F. Le Bars, and S. Veres. Reliable non-linear state estimation involving time uncertainties. *Automatica*, pages 379–388, 2018.

[10] C. Aubry S. Rohou, P. Franek and L. Jaulin. Proving the existence of loops in robot trajectories. *International Journal of Robotics Research (accepted)*, 2018.

[11] S. Thrun, W. Bugard, and D. Fox. *Probabilistic Robotics.* MIT Press, Cambridge, M.A., 2005.
[8]