ROS-2 wrapper for an interval analysis based method on a Robotnik Summit-XL robot

Adam Goux--Gateau
ENSTA Bretagne - Autonomous Robotics
adam.goux--gateau@ensta-bretagne.fr

Supervised by Dr. Emile Le Flécher
Researcher, project manager - Royal Military Academy of Belgium - Departement of Robotics and Autonomous Systems
emile.leflecher@mil.be

Contents

Abstract **2**

Résumé **2**

Acknowledgements **3**

1 Introduction **4**
1.1 General presentation 4
1.2 The Royal Military Academy 7
1.3 Internship purpose 8

2 Evolution of my internship work **9**
2.1 First steps of my project 9
2.2 Interval analysis : first tests 10
2.2.1 Ultrasounds emitters 10
2.2.2 Trajectory estimation 11

3 Fieldwork on the Robotnik **13**
3.1 IMU and interval analysis 13
3.2 Extended Kalman filter 15
3.3 Odometry 16
3.4 Odometry and interval analysis 18
3.4.1 Estimation of the error 18
3.4.2 Results of the experiment with odometers and intervals 19

4 Results and suggestion of further work **21**
4.1 Comparative analysis of the three approaches 21
4.2 Other ideas to explore 21
4.3 Takeaways from this internship 22

Bibliography **24**

Abstract

During my second year at ENSTA Bretagne in the Autonomous Robotics specialization, I undertook a 16-week engineering assistant internship at the Royal Military Academy (RMA) of Belgium in Brussels, in the Robotics and Autonomous Systems laboratory (RAS-lab). During this internship, under the guidance of Dr. Emile Le Flécher, my primary objective was to devise and implement a localization method based on interval analysis. The overarching goal was to compare the effectiveness of my algorithm against conventional localization methods such as IMU, Kalman filters, and odometry.

In the initial month, I familiarized myself with the robot's architecture that was integral to my work – a Robotnik Summit-XL. My tasks included identifying pre-existing nodes that were relevant to my project and enhancing my understanding of interval analysis to create meaningful test scenarios. Subsequently, the following three months were dedicated to the development of algorithms in C++ using ROS-2 to identify the best method according to the situation.

Résumé

Pendant ma deuxième année à l'ENSTA Bretagne en spécialité Robotique Autonome, j'ai effectué un stage d'assistant ingénieur de 16 semaines à l'École Royale Militaire de Belgique à Bruxelles, dans le laboratoire de Robotique et Systèmes autonomes. Au cours de ce stage, sous la supervision du Dr. Emile Le Flécher, mon objectif principal était de concevoir et de mettre en œuvre une méthode de localisation basée sur l'analyse par intervalles. L'utilité de ce travail était de comparer l'efficacité de mon algorithme par rapport aux méthodes de localisation conventionnelles telles que les centrales inertielles, les filtres de Kalman et l'odométrie.

Au cours du premier mois, je me suis familiarisé avec l'architecture du robot sur lequel j'allais travailler - un Robotnik Summit-XL. Mes tâches comprenaient l'identification des nodes préexistants pertinents pour mon projet et l'amélioration de ma compréhension de l'analyse par intervalles afin de créer des scénarios de test significatifs. Par la suite, les trois mois suivants ont été consacrés au développement d'algorithmes en C++ à l'aide de ROS-2 pour identifier la meilleure méthode selon les situations.

Acknowledgements

I would like to express my sincere gratitude to Pr. Luc Jaulin, my referent professor, for enabling me to work in Brussels for this internship. His guidance and support during my academic journey at ENSTA Bretagne, his expertise and dedication have been instrumental in shaping my understanding of robotics and its applications. I am truly thankful for the knowledge and skills I have gained under his teaching.

Additionally, I want to acknowledge the contributions of Pr. Simon Rohou at ENSTA Bretagne. He played a pivotal role in my development by imparting knowledge in C++ and interval analysis during my second year, which proved essential for my work in Belgium. Furthermore, his continuous availability to address my queries and provide guidance throughout my project was more than helpful.

I extend my heartfelt thanks to Dr. Emile Le Flécher for his supervision and mentorship during my internship at the Royal Military Academy of Belgium. His unwavering support, expert insights, and patience greatly contributed to the of my internship project. I am genuinely appreciative of the opportunity to work under his guidance and for the knowledge I have gained through his expertise.

Finally, I want to express my profound gratitude for the support I received from my colleagues at the RMA. While they certainly assisted me in my tasks, their warm embrace and inclusion were particularly meaningful. Their presence and camaraderie not only enhanced my work experience but also made my internship immensely enjoyable. I'm thinking of Ana, Manos, Loïc, Timothée, Nadège, Danial, Alexandre, Enzo, Guillaume. My apologies if I am forgetting someone.

To all of you : Thank you.

1 Introduction

1.1 General presentation

At ENSTA Bretagne, specifically within the Autonomous Robotics specialization, I have received instruction in several localization methods applicable to diverse scenarios. During my work at RMA, I needed to compare a few of them. I will provide a concise overview of each of these means of getting the position and orientation of a robot.

Inertial Measurement Units (IMU)

It is a mean to estimate the position using a gyroscope and an accelerometer [**1**]. It measures the acceleration following the three axis, and the yaw-pitch-roll. Hence, it can compute velocity, position and orientation of a robot.

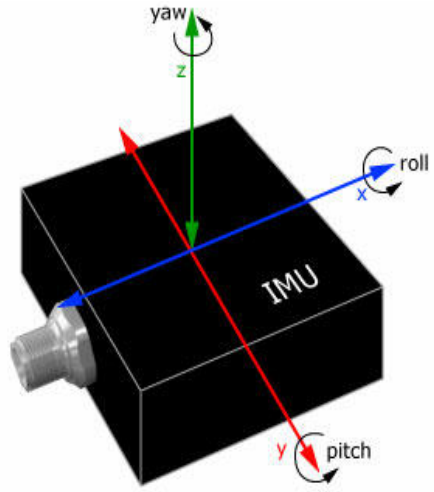

Figure 1: Basic representation of an IMU

Strength : This is a self-contained method, which means that it does not need any other external reference. It is also quite low cost if the precision needed is not too high. The main reason of its use is that it works everywhere, especially in obstructed areas. Underwater, underground, in heavily dusty areas, whatever the weather and the conditions are.

Weaknesses : Major disadvantage of using IMUs is that they suffer from the accumulation errors. The mathematical theory implies to integrate over time the acceleration. Hence, even a small error grows and leads to completely wrong estimations. A constant error in the acceleration, let us say 0.01 $m.s^{-2}$, leads to a velocity error of $0.01 \times t$, and finally a position error of $0.01 \times t^2$. This is called *drifting*. One way to counter this is to associate the IMU with an algorithm, such as Kalman filter (or interval analysis !), to correct the errors. However, the filter will need external measurements, like GPS, velocity sensor etc. to have a way to know the exact position and orientation of the robot.

Kalman Filter

The Kalman filter is a mathematical tool used to estimate the state of a system over time. It does this by combining information from two sources : a mathematical model that describes how the system behaves and measurements from the real world. It is optimal if the noise is Gaussian, and if the state equation is linear.
The filter starts with a model that describes how the system you're interested in behaves. This model includes information on how the system's state changes over time. It begins with an initial guess about the system's state and how certain or uncertain that guess is. Then, the algorithm is separated in two steps :

Prediction : The filter uses the system model to predict how the state of the system will change over time. It also predicts how uncertain this prediction is.

Correction : When real-world measurements become available, the filter compares them to its prediction. It calculates the difference between the prediction and the measurement. This difference, along with the uncertainties associated with the prediction and measurement, helps refine the estimate of the system's state.

This process repeats as measurements arrive, continuously improving the estimate of the system's state while considering uncertainties.

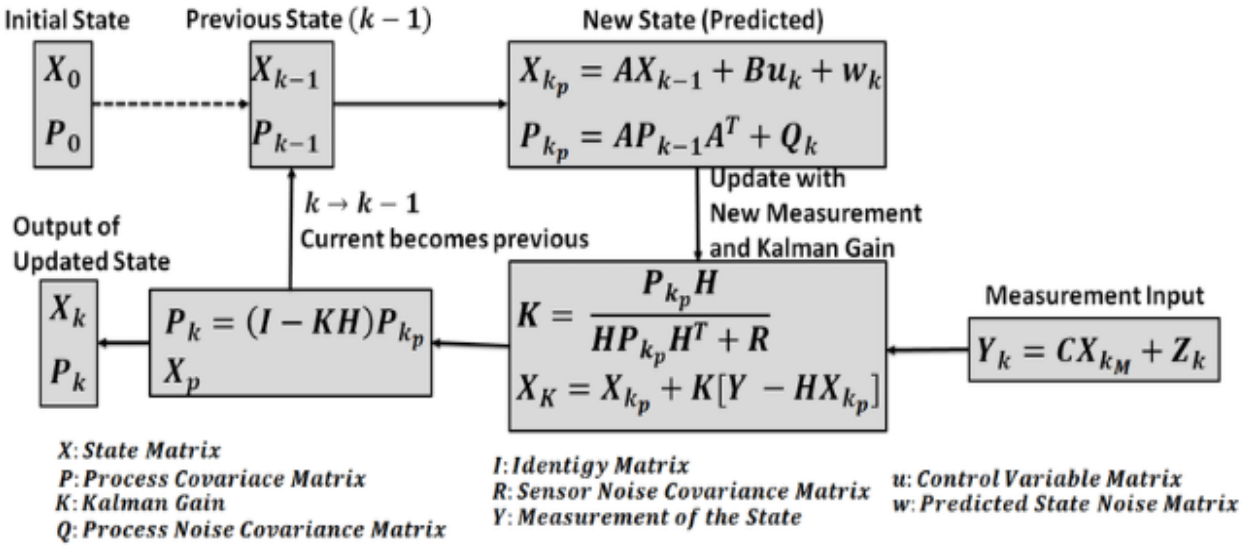

Figure 2: Equations of the Kalman filter

Its uses in robotics are mainly with an IMU. Indeed, to avoid the drifting effect, the Kalman filter can use other sensors like GPS to improve the accuracy. Even with noisy measurements and errors from the inertial units, the Kalman will make the data more precise and delete the accumulated errors [**2**].

Odometry

Odometry refers to the process of estimating the position and the orientation of a robot by tracking its movements. For ground robots, it relies on wheel encoders. An encoder is composed of a Hall effect sensors and a ring magnet mounted directly on the wheels of the robot. Together, they get incremental data on the rotation of the wheel, each "tick" corresponds to a fraction of the complete rotation of the wheel. Similarly, the rotation angle can also be estimated.

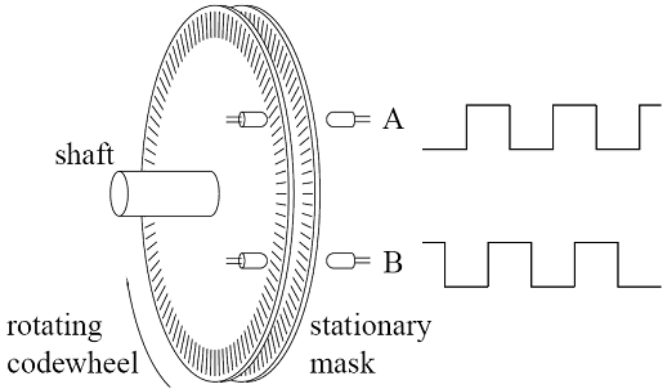

Figure 3: Simplified wheel encoder

Strengths : Gives directly the position and the orientation and provide clear sets of data.

Weaknesses : It assumes that the wheels are not slipping and that the ground is flat and level. In practice, wheel slippage, terrain irregularities, and other factors can introduce errors into the calculations over time, leading to drift in the estimated position. This is why odometry is often used in combination with other localization methods.

Interval analysis and constraint programming

Interval analysis is a field that involves replacing each value with an interval representing the associated uncertainties. Constraint programming [**3**], on the other hand, employs the systematic propagation of constraints as an approach in programming. I will delve into the mathematical aspects further. In summary, this method entails identifying and enumerating all the uncertainties related to sensors, measurements, etc. The solver then uses this list of constraints to provide an interval as a solution, often referred to as a 'box' (an interval in $\mathbb{R}^n$), or a 'tube'. While we can be certain that the solution lies within the solution set, it may not necessarily be optimal.

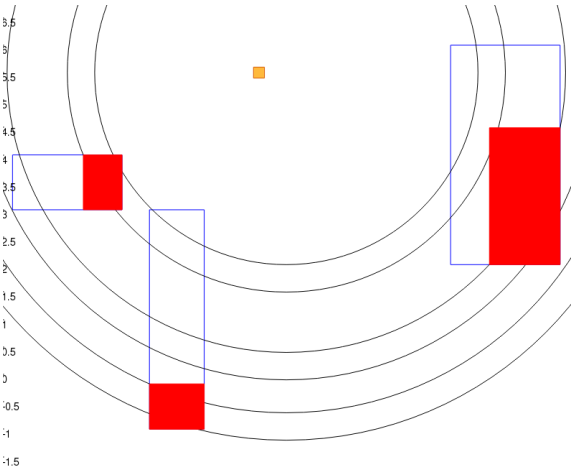

Figure 4: Contraction of boxes (from blue to red)

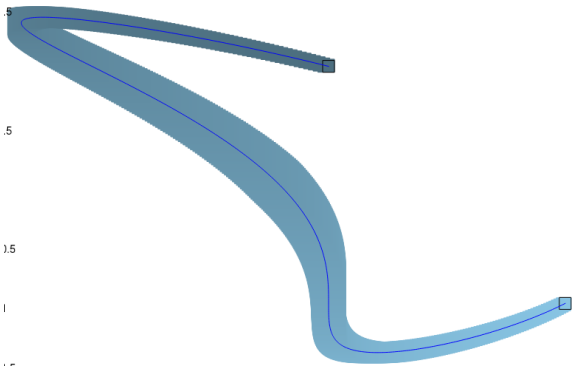

Figure 5: Estimation of a trajectory within a tube

Strengths : The reliability is perfect, we obtain a solution set that contains the real answer to our problem. It works really well when the problem is not linear, and if we do not know precisely the uncertainties of our problem. It is also quite easy to develop, we only need to provide a list of initial constraints.

Weaknesses : It is often not optimal, an can provide an imprecise solution.

1.2 The Royal Military Academy

The Royal Military Academy is the Belgian university that trains the officers of the five components of the Belgian defence, which are Army, Navy, Air Force, Cyber and Medical. The languages spoken during the courses and among the students are Dutch, French and English. The university is divided into two faculties : the Faculty of Applied Sciences, and the Faculty of Social and Military Sciences. The first one delivers Masters in engineering, and can be compared to French engineering schools under guardianship of the Ministère des Armées, such as Polytechnique or ENSTA Bretagne.

I worked in the Robotics and Autonomous Systems laboratory (RAS-lab), itself part of the Mechanical laboratory. This was at first strange for me, because I had always considered myself as a software student. The major part of robotics for me consisted in simulating and programming.
The team leader of the laboratory was Geert de Cubber, a senior researcher and mechanical engineer. While I had only limited interactions with him, my primary point of contact was my internship supervisor, Emile le Flecher. He holds a Doctoral degree in robotics and serves as a project manager at the RAS-lab. Alongside Mr. Le Flécher, all of his immediate colleagues mentioned in the acknowledgments were the individuals I interacted with daily and collaborated closely with. They were all here to guide me during my internship and to help if needed.

1.3 Internship purpose

As mentioned earlier, my research primarily centered on the comparative analysis of various localization methods, focusing on a Robotnik Summit-XL robot.

Figure 6: Robotnik Summit-XL

To initiate this comparison, I began by calibrating a GPS. Its purpose was to provide an accurate reference trace of the Robotnik's movement, which would serve as a benchmark for evaluating other localization techniques. This GPS calibration was not considered one of the primary localization methods for my internship. Indeed, inherent vulnerabilities are associated with GPS systems, especially in military contexts, like signal jamming or spoofing, and GPS signals may become unreliable. Instead, the calibrated GPS was used exclusively for comparative purposes.

Subsequently, after improving my understanding of interval analysis, I devised an algorithm that relied on the Robotnik's IMU. This algorithm involved double-integrating the IMU's acceleration data to estimate the robot's position. I did not focus on orientation for this task, the measurements were too noisy.

Then in the course of my research, I conducted separate evaluations of the robot's odometry, first in isolation and then in conjunction with interval analysis. This assessment aimed to determine if the application of constraint programming could enhance the robustness of the encoders' measurements.

Lastly, I explored the feasibility of utilizing the extended Kalman filter, which had already been implemented on the robot. While this filter demonstrated success when integrated with the IMU, due to time constraints, I was unable to arrive at conclusive findings regarding its effectiveness.

In the following sections of this report, I will first provide a summary of my work at RMA, followed by overview of the results and finally, the experience gained.

2 Evolution of my internship work

2.1 First steps of my project

At the very beginning, the tasks I had to accomplish were in collaboration with Etienne Roussel, the other ENSTA Bretagne student in the RAS-lab. We helped the PHD students by installing Ubuntu, ROS-2 and Unreal Engine on laptops. The team members also introduced themself, so that we can be aware of the field of research of each one. It was then convenient to ask for guidance to the right person.

Then I had to calibrate the Reach RS+ GPS-RTK (Global Positioning System - Real Time Kinematic) from Emlid. GPS RTK is an advanced satellite-based positioning system known for its exceptional accuracy. It relies on signals transmitted by multiple GPS satellites orbiting Earth. These satellites broadcast signals that include their positions and precise timestamps. A GPS RTK user receiver collects signals from at least four of these GPS satellites, all of which must be within its line of sight. Additionally, there is a stationary reference, the *base* station with precisely known coordinates that also receives signals from the same set of GPS satellites. Both the user receiver (the *rover*) and the base measure the phase of the incoming satellite signals, representing the precise timing of these signals.

The rover compares its phase measurements with those of the base to identify any discrepancies. The reference station calculates correction data based on the phase differences and transmits these corrections to the user receiver in real-time. The user receiver then applies these real-time corrections to its own phase measurements, resulting in a significantly improved accuracy in its position calculation.

The GPS RTK rover can determine its position with extraordinary precision, often reaching centimeter or millimeter-level accuracy or better.

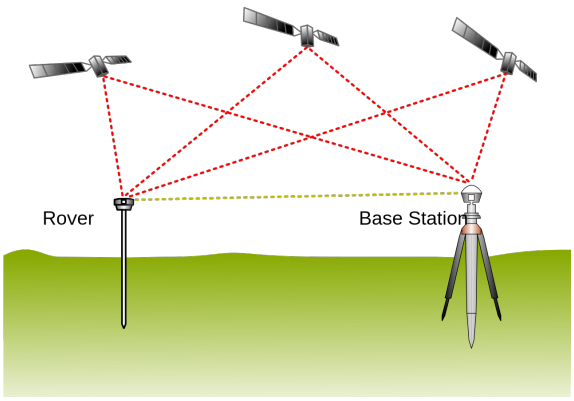

Figure 7: Basic diagram of a GPS RTK

Figure 8: Emlid Reach RS+

Emlid has an application to calibrate the base and the rover, but to check that the data sent were the right type (NMEA format), I had to use a ROS SDK provided by

the company in a Github repository. My first problem occurred here. The Operating System of my laptop is Ubuntu 22.04, however, ROS is not actively maintained on this version of Ubuntu.

That is why I needed to use a Docker image. Docker [**4**] is an open-source platform that enables developers to automate the deployment and scaling of applications inside lightweight, portable containers. These containers are self-sufficient units that encapsulate all the necessary code, runtime, libraries, and dependencies required to run an application. A Docker image is a static package built by a file called a *Dockerfile* and running in a container. I used the ROS-Noetic image available in the Docker Hub, a vast repository containing thousands of images.

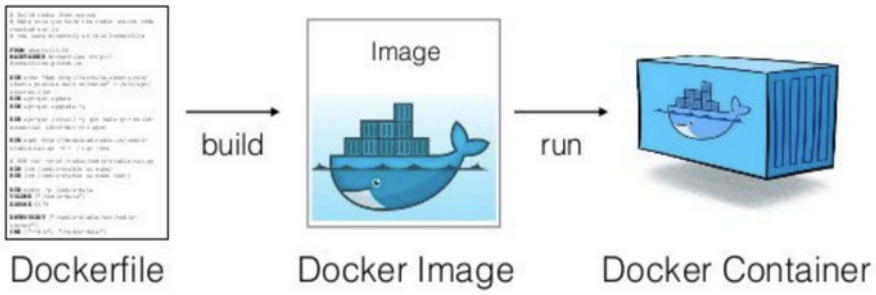

Figure 9: How a Docker image runs

I struggled a bit on this task, I had to get started with ROS, and to improve my knowledge of Docker. But starting almost from zero about ROS and Docker and debugging a code from a company was definitely a useful experience, I learned how to find resources and to understand them.

2.2 Interval analysis : first tests

2.2.1 Ultrasounds emitters

While my primary objective during my work was to conduct a comparative analysis of various methods, the focal point shifted significantly towards interval analysis and constraint programming. This methodology was indeed relatively unfamiliar within the lab, prompting me to delve deeply into the subject and gain a comprehensive mastery of it.

The mathematical details of the theory will not be reiterated in this report ; however, the reference is available in the bibliography for interested readers.

I started with a few simulations before working on the real robot. At first, I supposed that we had three of ultrasounds, and a receptor on the robot. It is a fairly cheap technology, and this situation seemed realistic to me. I added some noise to the measurements, and to the position of the emitters. Here are the results I obtained :

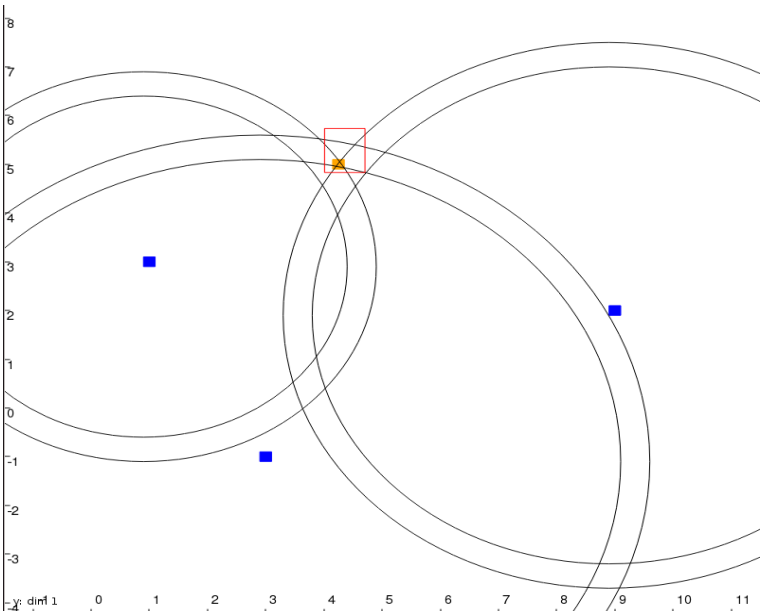

Figure 10: Localization of a robot thanks to three landmarks with interval analysis.

The emitters are represented by blue boxes, indeed, we know where they are with a good precision, the boxes are small. The rings are the distance measured by the receptor on the robot for each emitter.

Here, the uncertainties are bigger, because a lot of factors can influence the precision. For example the noise of the sensors, the precision about the sound speed etc.

I finally obtained the red box as a result, which represents a set within which we can ascertain the robot's position. The real position of the robot (unknown in reality) is indeed the orange square. This simulation was a success, we can obtain good results with only constraint programming and a few sensors.

2.2.2 Trajectory estimation

This simulation was my last step before the practical tests. Using the acceleration that can be given by an IMU, I integrated it twice to get the position, knowing the initial one.

To perform this model, I took a random curve equation for the acceleration :

$$\begin{cases} a_x = -2\sin t - 4\sin 2t \\ a_y = -\sin t + \cos t \end{cases}$$

To imitate the real conditions, I chose discrete values of the equation, as if I got data from an IMU. For each values of t, I plugged it into the parametric equation to get two vectors containing a_{x_i} and a_{y_i}. Then, I transformed each pair (a_x, a_y) into an IntervalVector. I inflated them to add artificial noise, and I integrated it twice using intervals arithmetic.

The analytical solution is :

$$\begin{cases} x = -2\sin t - \sin 2t \\ y = \sin t + \cos t \end{cases}$$

The constraint programming approach will provide a tube that encompasses the analytical solution. Setting the position x_0 to $(0, 0)$, we obtain this TubeVector :

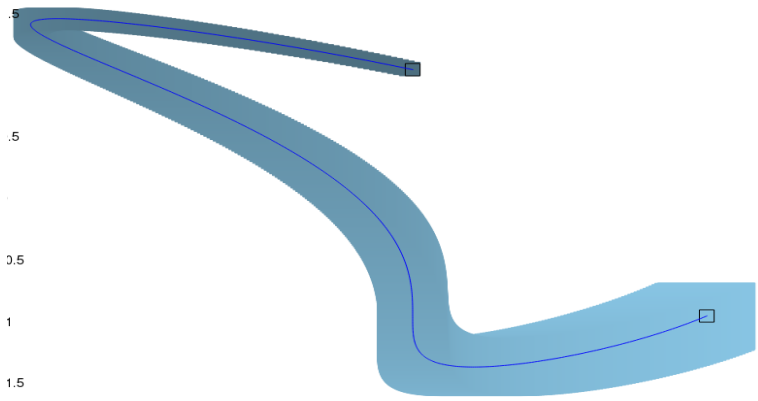

Figure 11: Tube solution 1

The blue line is the analytical solution, unknown in real life. The blue set is the tube containing the solution, which is already quite precise.

If we add the hypothesis that we know the final position of the robot, the TubeVector can be contracted even more :

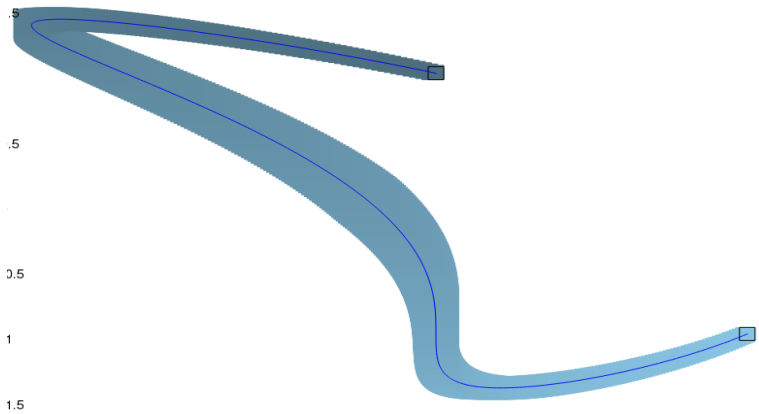

Figure 12: Tube solution 2

We obtain a solution set that is even more accurate, which is why I had a positive outlook on the upcoming experiments.

3 Fieldwork on the Robotnik

3.1 IMU and interval analysis

Regarding the results I obtained in the previous simulation, I tried the same experiment with the real IMU of the robot. To do so, I used Robot Operating System-2 (ROS-2) [**5**]. It is a set of softwares libraries and drivers for robotic application. It contains almost every open-source tools you need for your robot project. Once again, I will not go into details, all the infos are available on the ROS-2 website in the bibliography.

There were already a few *topics* and *nodes* created for all the Robotnik's sensors. On the topic `/imu` was published all the data sent by the IMU. I had the vector $\begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix}$, and the orientation vector given through quaternions. I did not focus on the orientation of the robot, only on its position. To connect with the Robotnik, I used a SSH protocol :

```
$ ssh -X robot@192.168.XX.XX
```

Where the `XX.XX` is the end of the local IP address of the robot. After performing this wireless connection, I listened to the `/imu` topic while moving the robot, everything seemed to work fine.

At this moment, my goal was to write a ROS-2 node using the same principle as the previous simulation. I collected the date from the accelerometer, and with a double integration in intervals arithmetic, I wanted to obtain a realistic tube. The method I used to know the uncertainties about the acceleration was to listen to the `/imu` topic while I gave him a constant value, then to record a ROS bag of the node running. I simply had to look at the minimum and the maximum value around the acceleration to inflate my intervals with the value : $\max(|a_{min} - a|, |a_{max} - a|)$.

I needed VIBes-Viewer to get plots, and I struggled to install it on the robot. This is why I chose to get the data from the sensors on my computer to produce the plots. This is how I organised my code :

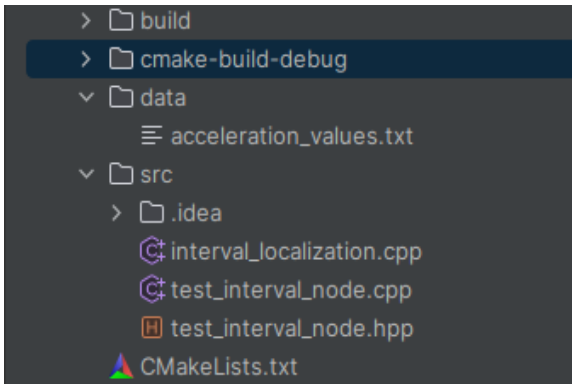

Figure 13: Architecture of the code

In the *data* folder, I had the list of acceleration values ; my node was separed into a header and a source file : `test_interval_node.hpp` and `test_interval_node.cpp` and all the algorithm was conducted in `interval_localisation.cpp`.

My first experiment was with a perfectly linear trajectory. I drove the robot in a perfect straight line, by giving exactly the same speed instructions to the motors. Here is what I obtained :

Figure 14: Linear trajectory

The distance was quite small, only a few meters. This is why knowing the initial an final position gave me enough contraction to obtain almost a rectangle as a tube. The method seemed to work quite well on such a case.

For a more complicated trajectory, I made the robot draw an ellipsoid on the ground, the big diameter was approximately 6 meters long. Here is the result :

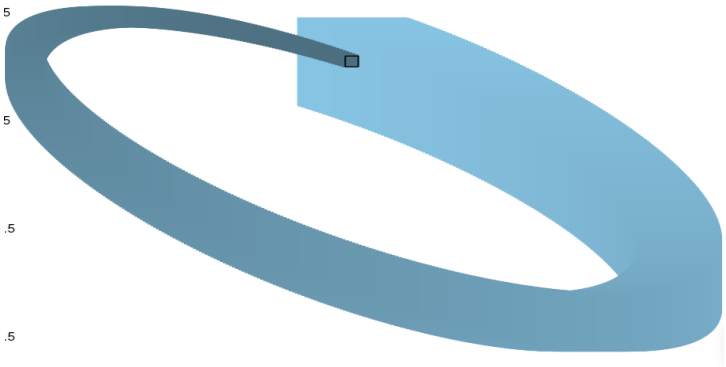

Figure 15: Elliptic trajectory without the final position known

As observed, the errors are growing rapidly, reaching approximately one meter by the end of the trajectory. Even if we suppose that the final position is known :

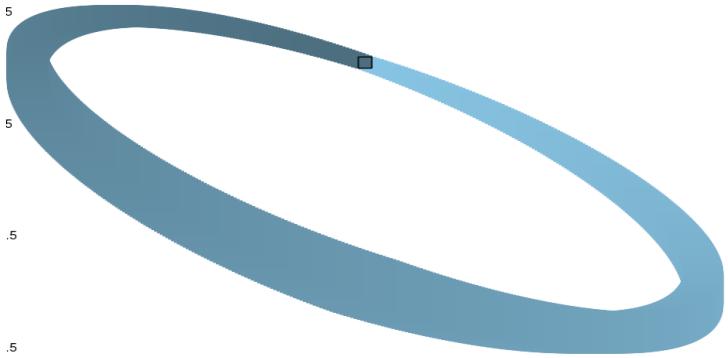

Figure 16: Elliptic trajectory knowing the final position

The precision has improved slightly, but it is still unsatisfying considering the small distance covered by the robot.

In summary, this method exhibits acceptable reliability for linear trajectories and short distances. However, achieving precise results becomes challenging when the robot makes turns. To enhance accuracy, the addition of more sensors and the implementation of periodic GPS calibration could yield more promising outcomes.

3.2 Extended Kalman filter

This part will not be as developed as the previous one, as said in the introduction, I did not manage to fully explore this method because of time constraint. First things first, an extended Kalman filter (EKF) is the non linear version of the Kalman filter. The noises still needs to be Gaussian, but the state function has no need to be linear anymore, only differentiable. The main default of the EKF is that it is not necessarily optimal, unlike the linear Kalman filter. The difference in the algorithm is that we differentiate the state function and the command function.

The EKF node was already integrated into the robot, so I didn't have to invest much effort into its implementation. It was combined with the IMU to mitigate system drift. Unfortunately, I conducted only a qualitative study on this method due to time constraints, and I didn't manage to perform a detailed quantitative comparison. However, the limited information I gathered did indicate that the EKF + IMU system performed better than the IMU alone, which isn't particularly surprising.

In terms of reliability, I had to venture outdoors to use the GPS to correct the IMU's data through the EKF. This approach generated a trajectory, but I couldn't definitively ascertain its accuracy. It did appear consistent with the path I instructed the Robotnik to follow, but I couldn't provide further insights into the method's performance.

3.3 Odometry

For this method, I basically just had to consider the wheel encoders that gave me directly the position, and the orientation with quaternions. They have many advantages : no gimbal lock, more compact than a 3×3 matrix (we need only 4 real numbers to represent them), but they are especially more stable numerically than matrices, and do not lead to divergences than can for example happen if $|tr(A)| \approx 0$ where A is the rotation matrix.

However, to use the information about orientation, I needed to get the orientation matrix. Considering the quaternion $a + bi + cj + dk$ were $(a, b, c, d) \in \mathbb{R}^4$, the associated orientation matrix is :

$$\begin{pmatrix} a^2 + b^2 - c^2 - d^2 & 2bc - 2ad & 2ac + 2bd \\ 2ad + 2bc & a^2 - b^2 + c^2 - d^2 & 2cd - 2ab \\ 2bd - 2ac & 2ab + 2cd & a^2 - b^2 - c^2 + d^2 \end{pmatrix}$$

It slowed down my node if the robot

This method provides only discrete values, resulting in a set of points instead of continuous curves due to the time sampling. Hence, assessing the accuracy was challenging in this context.
In my attempt to quantify the uncertainties of the encoders, I developed a basic ROS-2 node to command the robot to move in a straight line for a specified distance, provided through the argc and argv[] parameters of the main() function. After issuing the distance command, I measured the actual distance covered by the Robotnik. It became evident fairly quickly that the primary source of error stemmed from wheel slippage during the initial moments when the robot starts moving.

I encountered another issue where the encoders were programmed to reset to zero whenever the robot came to a halt. For example, if the robots had covered 4 meters, then stopped and then started again, I would have this kind of output :

0, 1.1, 2.4, 3.2, 4.0, 0, 0.9...

I had to consider it in my code, if the count got back to 0, I needed to add the following values to the previous ones.

For a linear trajectory, I of course got a perfect straight line, because the wheel did not turn, so the encoders could not send any data to my computer. However, it was difficult to evaluate the error about the distance covered :

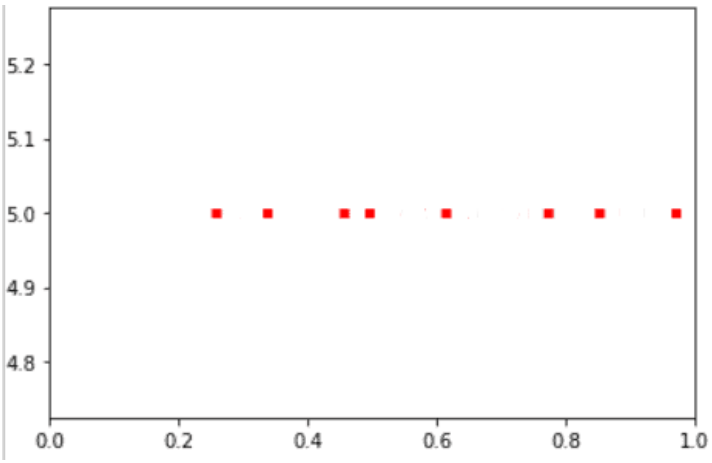

Figure 17: Straight line with odometry

Now it was time to test on a more complicated movement. Once again, the results were coherent but once hard to interpret. On this experiment, I moved the Robotnik very slowly, and got back to my starting point. The path was short, and the low speed enabled me to bind the points together to get a more readable curve :

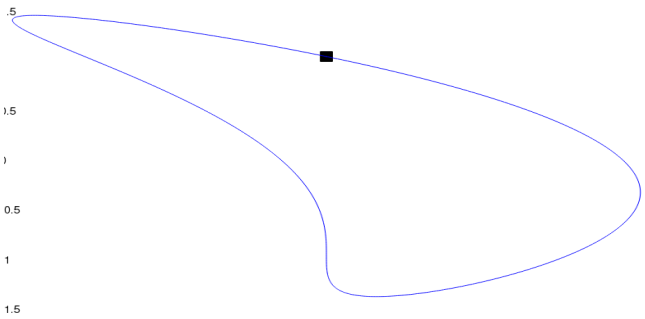

Figure 18: Error according to the distance

A positive aspect of this code is that it successfully returns the robot to its initial position, indicated by the presence of a black square. It proves that the accuracy of the encoders was really interesting, and reliable for our experiment. However, during the travelling, I did not find any reliable mean to detect an error of the data, because of the lack of information. With only a distance and an orientation given, I could not guess if it was far away from the reality or not. In the following days, I figured out a way of establishing the error of the odometers. This is what the following part is about.

3.4 Odometry and interval analysis

3.4.1 Estimation of the error

In my attempt to quantify the uncertainties of the encoders, I developed a basic ROS-2 node to command the robot to move in a straight line for a specified distance, provided through the `argc` and `argv[]` parameters of the main() function. After issuing the distance command, I measured the actual distance covered by the Robotnik. It became evident fairly quickly that the primary source of error stemmed from wheel slippage during the initial moments when the robot starts moving., I observed that as the commanded distance increased, the accuracy of the robot's movements improved as well, which is quite logical.

I even plotted a graph using the `matplotlib` library showing the percentage of error in a straight line regarding the distance covered. It is important to note that this "study" was not conducted rigorously ; I performed only a limited number of attempts for each distance and then calculated the mean to determine the percentage of error. It simply gives an idea on how the error decreases with the distance :

Figure 19: Percentage of error linked to the distance covered

As we could expect, the error decreases with the distance, because the sliding effect of the wheel becomes irrelevant after a few meters. Of course, if the Robotnik stops and starts again, this graph is not true anymore. I found interesting that the shape was almost linear, this is why I plotted a linear regression line. Here are its characteristics:

```
Slope: -1.5142857142857145
Intercept: 14.171428571428574
R-squared: 0.9584577326622881
```

Figure 20: Linear regression

The R^2 value is remarkably high, and I could not find any related studies [**6**], which suggests that this result might have been due to chance. Nevertheless, as the distance increases further, the error continues to decrease, albeit at a very gradual pace, and it never drops below 3%. The linear aspect remains as long as the distance is below approximately 9 meters.

WHat I did not test for is the effect of making turns. Indeed,The two left wheels of the robot rotate in the opposite direction to the two right wheels to perform a turn. This naturally leads to a drifting phenomenon. However, conducting tests for various angles, speeds, and other factors would have been exceedingly time-consuming. This is why I did not account for this in my testing and avoided making the Robotnik turn too sharply.

3.4.2 Results of the experiment with odometers and intervals

Using my earlier estimation as a foundation, I developed a method to quantify the error introduced by the encoders. To use this, I utilized a recorded ROS bag of the node responsible for generating Figure 18. The key modification involved implementing an interval-based approach in the algorithm to take errors into account, which were contingent on the distance traveled. I obtained this graph :

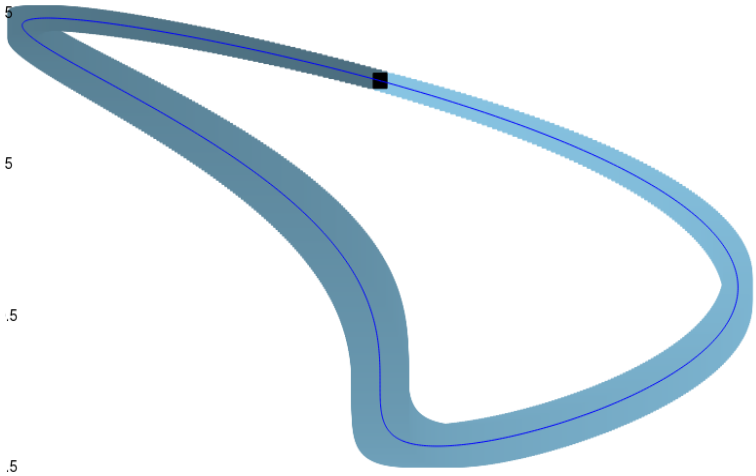

Figure 21: Trajectory combining odometers and interval analysis

The thin blue line represents the graph I obtained without performing interval

analysis. I overlaid it on the same figure to facilitate a comparison between the two experiments. Fortunately, this line is encompassed within the TubeVector. When it comes to the error, taking the scale into account, I approximated it to be around 40cm at the midpoint of the trajectory, while the robot covered ≈ 15 meters. While not ideal, this estimate is an improvement compared to using the IMU combined with the interval method.

4 Results and suggestion of further work

4.1 Comparative analysis of the three approaches

In controlled environments with smooth surfaces, the synergy between constraint programming and odometry has demonstrated a clear advantage over using the robot's IMU alone. The precision and quality of the encoders played a crucial role in achieving these favorable outcomes. However, it's crucial to acknowledge that when dealing with uneven terrains, such as outdoor scenarios, encoders can become less reliable due to factors like slipping and the presence of obstacles. These variables introduce significant uncertainties into the measurement process.

Furthermore, the absence of GPS signals indoors can limit the EKF's utility for localization and navigation tasks. Nevertheless, when operating in outdoor environments, the EKF exhibits its true potential, as it can leverage optimization techniques and seamlessly integrate with the robot localization package offered by ROS-2, thereby enhancing its performance and accuracy in outdoor settings.

4.2 Other ideas to explore

An excellent advantage of constraint programming is its ability to incorporate uncertainties into problem-solving, thereby refining the solution space. To illustrate this concept simply: Instead of evaluating IMU and odometry data separately, I could have employed interval analysis to integrate the results, effectively broadening the scope of potential solutions. Additionally, I could have leveraged the odometry data to mitigate IMU drift. In lieu of relying on GPS, I could have fed the odometry data into the Extended Kalman Filter and assessed whether its precision sufficed for the task at hand.

Given that I operated within a closed environment, it was entirely feasible to employ localization methods based on obstacle recognition, provided I possessed prior knowledge of the room's layout. Techniques such as LiDAR and ultrasound emitter-receiver systems could be utilized for this purpose. By obtaining distance measurements from all the room's walls and obstacles, it would be possible to employ methodologies like Monte Carlo simulations or least-square estimations to accurately determine the robot's position and orientation within the environment :

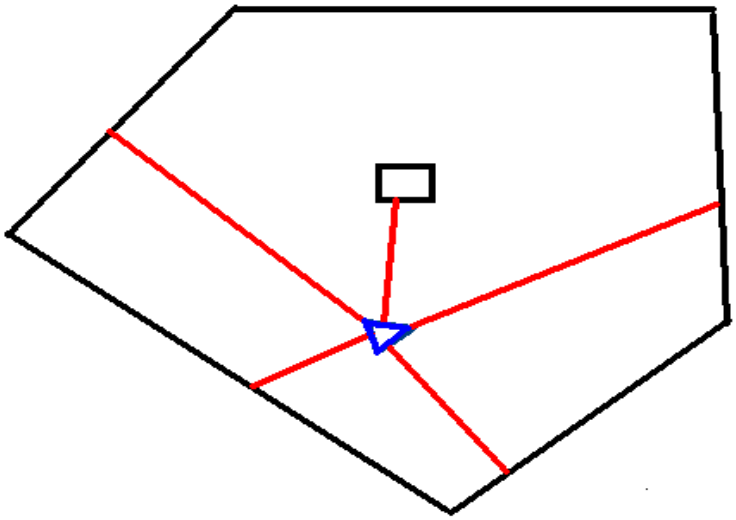

Figure 22: Robot in blue, distances in red, obstacles in black

A stochastic gradient descent method or Monte-Carlo would be particularly accurate here, we could iterate number of time without too much computing time. Of course, it will give us a local minimum, we would need to take several starting points.

4.3 Takeaways from this internship

During my internship I had the opportunity to work in a foreign laboratory. This was my first experience of working abroad. It also marked my introduction to the world of research. Indeed, I had no idea that working in a lab instead of in a company would be this enjoyable, it brought about significant personal and professional growth.

In terms of robotics, I improved my skills in multiple ways : programming, especially C++, interval analysis, Docker, ROS-2.... And I am probably forgetting aspects. Equally significant, beyond the technical aspects, the weekly reports I was tasked with delivering provided a profound lesson in the realities of working life. It required me to deliver tangible results to a team, a responsibility that was a novel experience for me. I cultivated my ability to present my findings and insights clearly and concisely. I firmly believe that this experience will be immensely valuable in preparing me for the demands of my future professional endeavors.

Furthermore, my internship was a valuable platform for improving my communication skills. I collaborated with researchers from diverse backgrounds, which necessitated effective communication and teamwork. This experience sharpened my ability to convey complex ideas and collaborate efficiently in a multicultural environment (at least 6 nationalities in the lab !).

In conclusion, my internship in robotics and localization was a transformative experience. It introduced me to the world of research, allowed me to refine my technical and communication skills, and instilled in me a profound appreciation for the endless possibilities within the field of robotics. This internship has undoubtedly equipped me with valuable

tools and insights that will shape my future in the world of engineering, and why not, research.

Bibliography

References

[1] *Mobile robotics : Inertial*, L. Jaulin, https://www.ensta-bretagne.fr/jaulin/inmooc.pdf

[2] *Robust Kalman Filtering for Signals and Systems with Large Uncertainties*. Ian R. Petersen , Andrey V. Savkin. (1999).

[3] *Codac : constraint programming for robotics*. S. Rohou, B. Desrochers, codac.io

[4] *Docker Docs*, Open-source, https://docs.docker.com/

[5] *ROS 2 Documentation*, Open-source, https://docs.ros.org/en/humble/index.html

[6] *Learning Wheel Odometry and IMU Errors for Localization*, M. Brossard, S. Bonnabel, **HAL** (2019), https://hal.science/hal-01874593/document