Assistant Engineer Internship Report: Development of an Unmaned Ground Vehicle with a tethered drone system

Samuel MAGNI
Engineering Student at ENSTA - Autonomous Robotics Major
samuel.magni@ensta.fr

May 2025 - September 2025

Supervisor:
Dr. James E. PICKERING
Lecturer of Control Engineering at Aston University
j.pickering1@aston.ac.uk

Address:
Aston University,
Birmingham, B4 7ET, UK

Résumé

Ce stage d'assistant ingénieur, réalisé au sein du laboratoire de robotique de l'Université d'Aston (Royaume-Uni), s'inscrit dans le cadre de ma spécialisation en robotique autonome à l'ENSTA. L'objectif principal du projet était le développement d'un véhicule terrestre autonome (UGV) capable de servir de station mobile d'alimentation pour un drone relié par un câble (UAV). Cette approche vise à pallier la faible autonomie énergétique des drones en leur fournissant une alimentation continue depuis le sol, tout en conservant une mobilité commune des deux systèmes.

Mon travail a porté sur la conception logicielle du UGV, incluant la localisation, le contrôle et la gestion des contraintes liées à l'environnement sans GNSS et symétrique. Deux méthodes de localisation sans GNSS ont été explorées : une détection de repères par LiDAR et un algorithme de SLAM.

Le contrôle du robot a été assurée par des régulateurs PID afin d'atteindre une cible dynamique tout en intégrant un système d'évitement d'obstacles et des sécurités liées au câble du drone.

En parallèle, j'ai participé à la création d'une simulation dans Gazebo d'un robot quadrupède à roues Go2-W, afin de développer un jumeau numérique pour des applications hospitalières. Ce stage m'a permis d'approfondir mes compétences en robotique, tout en découvrant l'environnement de recherche universitaire britannique.

Abstract

This assistant engineer internship, carried out at Aston University's Robotics Laboratory (United Kingdom), was part of my specialization in autonomous robotics at ENSTA. The main objective of the project was to develop an Unmanned Ground Vehicle (UGV) capable of serving as a mobile power station for a tethered Unmanned Aerial Vehicle (UAV). This approach aims to overcome the limited battery life of drones by supplying continuous power from the ground while maintaining coordinated mobility between the two systems.

My work focused on the software design of the UGV, including localization, control, and management of constraints related to symmetric and GNSS-denied environments. Two localization methods without GNSS were explored: LiDAR landmark detection and a SLAM-based approach.

The robot's control was implemented using PID regulators to follow a dynamic target, incorporating obstacle avoidance and safety mechanisms related to the UAV tether.

In parallel, I contributed to the development of a Gazebo simulation for the wheeled quadruped robot Go2-W, aiming to create a digital twin for hospital applications. This internship allowed me to strengthen my skills in robotics while discovering the research environment of a British university.

Contents

Introduction

Internship Context

In the context of my engineering studies specialized in autonomous robotics at ENSTA, I was required to complete an international assistant engineer internship. Driven by my curiosity about scientific research, I decided to contact the Robotics Laboratory at Aston University, located in Birmingham, United Kingdom. There, Dr. James E. Pickering offered me a position in a student group to work on a project involving a UAV tethered to a mobile ground station. Specifically, I was responsible for developing the mobile ground station, which was a UGV. Later in the summer, I was also assigned to a second project, which was about creating a digital twin of a quadruped robot.

Project Description

UAVs are a very efficient way to explore, measure and navigate through different and unknown environments thanks to their high maneuverability that allows them to reach specific areas. But they are quite limited by a big downside: their battery life. Indeed, even if they are able to achieve a lot of different missions, they can never be really long, because a big battery is heavy and restrains the UAV's motion, so they can only carry lightweight ones.

A solution to this problem is not to carry these heavy batteries on the UAV, but to use a ground station that is tethered to the UAV to send the needed power. This method fixes the battery life issue, but constrains the drone to an area next to the ground station, because it can't go further than the tether length. This is not an issue in some situations, where the drone doesn't need to go out of a specific range, but in the case of environment exploration, it needs to be free of this constraint. The solution in this situation is to use a mobile ground station that can carry the weight of the batteries and follow the UAV to the location of the mission, before letting it achieve the aerial mission. The most convenient mobile ground station in this situation is basically a UGV, which can be operated by a controller or set to be autonomous.

Researchers and students working at Aston University Robotics Laboratory are used to working with UAVs, and the issue of battery life is well known among them. Considering that, a project of a UAV tethered to a UGV had been proposed by Dr. Pickering to his students, for both academic and research purposes. The main goal set by the teacher was to perform autonomous navigation of both UAV and UGV in the same frame, and to perform a landing of the UAV on the UGV that carries a box to welcome it, and the whole mission should work in a GNSS-denied environment. To achieve this, the tether needs to be more than a power supply, but also a data transmitter, and the tether length needs to be adjusted dynamically. As the project was big and separated between several hardware parts (the UAV, the UGV, and the landing station

attached to the UGV), we were a team of four students and one intern to complete it, and we distributed the tasks among us to be efficient and to use the skills of everyone. The five main tasks that we used for the distribution are the following: conception and construction of the UAV, localization and control of UAV, communication between UAV and UGV, construction of the landing box, and localization and control of UGV. The latter was my task, and I will explain how I completed it during this internship in the following part of this report.

UGV development

The UGV development was made with an existing robot, the Jackal UGV, developed by Clearpath Robotics. So the main steps of my task were to understand the robot's features and sensors, and use them to locate and control it, while considering the UAV location for security reasons or for mission purposes.

Hardware and environment constraints

Figure 1: Picture of the Jackal UGV

The Jackal (see picture on Figure 1) is a differential drive robot, which allows to control it easily by both linear and angular speeds, and it is able to rotate around its center, which is really convenient. The only two actuators on the robot are motors, and each motor controls the two wheels on one side (left or right) of the robot. On another side, the Jackal is equipped with numerous sensors:

- A 270° LiDAR

- A camera
- A GNSS sensor
- 2 encoders (one on each motor)
- IMU 6-axis (accelerometer + gyroscope)

Technically, the IMU also contains a magnetometer, but as it is really close to the motors, it is not accurate and is disabled by default by the company, so we can't use it.

Additionally, the goal was to navigate into a GNSS denied environment, especially because the testing room of the laboratory is underground and doesn't get any GNSS data. It was considered to achieve some test outside if the GNSS was absolutely required, but that was not the preferred option for safety and flight authorization reasons. Another note about the environment is about the strong symmetry of the testing room, a rectangular shaped room, which is not convenient for accurate localization without GNSS sensor or magnetometer.

Software architecture

To complete the UGV mission, I ended up with this software architecture (Figure 2) which I will explain in the following sections of the report. All the software communications are performed by using a ROS communication.

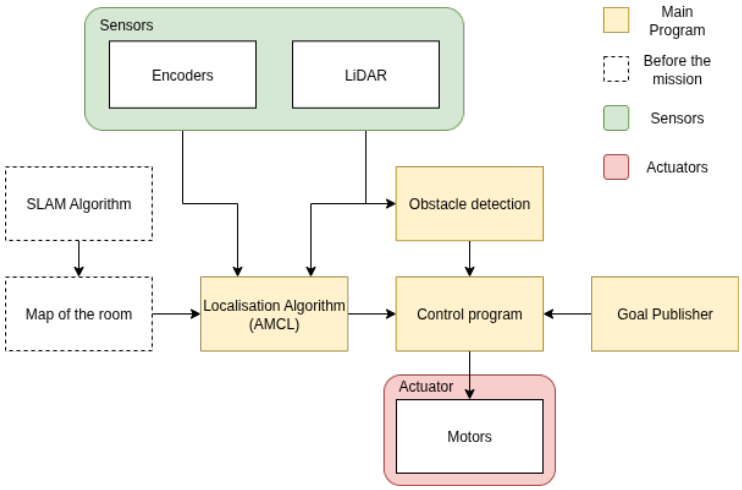

Figure 2: Software architecture

Localization

With the sensors equipped on the robot, the most logical option for localization seems to be using the LiDAR and the encoders. The encoders allow the UGV to locate itself in a local frame, by integrating the motion of each wheel over time. This method called odometry can perform at a high rate, but it is not accurate when used for a long time. This is why we need another way to find the pose of the robot, more accurate, to correct the error of the odometry. This new method doesn't need to perform at a high rate, because it will be used in addition to the odometry which is fast and accurate enough to locate between 2 measurements of the corrective method.

First method: Localization by LiDAR landmarks

This corrective measurement can be computed in several ways using the LiDAR. My first attempt was inspired by an exercise that I did in class at ENSTA which can be found in reference [4]. This exercise focuses on locating walls with LiDAR data, in a room with 90 degrees corners. This was really similar to my environment, so I decided to re-use this method to find the walls of the room.

To explain briefly, this method computes several linear regression on consecutive points obtained by the LiDAR, and computes the angle of the outputs lines. Those angles correspond approximately to the angles of the room's walls, and with these data it's possible to obtain one main angle θ that corresponds to one of the 4 walls, and since the corners' angles are known, we know the 4 angles that correspond to the walls. Combining that with the multiple linear regressions computed before, we can detect the distance between the sensor and the wall by taking the median of distances for all linear regression with approximatively the correct angle. With that method and because our LiDAR is only 270 degrees of field of view, we can detect between 2 and 4 walls (never only one wall because the field of view is more than 180 degrees). You can find some visual results of this localization method in Figure 3.

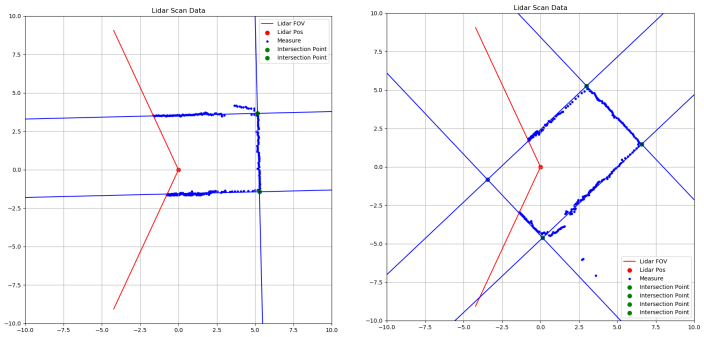

Figure 3: Results of the wall detection detecting 3 walls (left) and 4 walls (right)

We can see on the plot that even if the LiDAR doesn't have a 360 degrees field of view, it can detect the walls that are only partially seen, but when the wall is completely behind the sensor, it is not detected. We can also see that the program works even with some imperfections on the wall, for example, we can see that the door of the room is a bit open, but the program ignores it.

From this state, I know where are the walls around the robot, but not where is the robot in the room. To find this information, I need to use landmarks whose positions in the room frame are known exactly. The easiest landmarks to choose are the corners, because they are easy to get from the walls' position. Once I have the distance between the robot and the detected landmark, I can start from the landmark and make the reverse operation to find the robot position, and we can see on Figure 4 that it's working pretty well. When the landmarks don't give the same exact point from the others landmarks, we simply take the mean of the positions found.

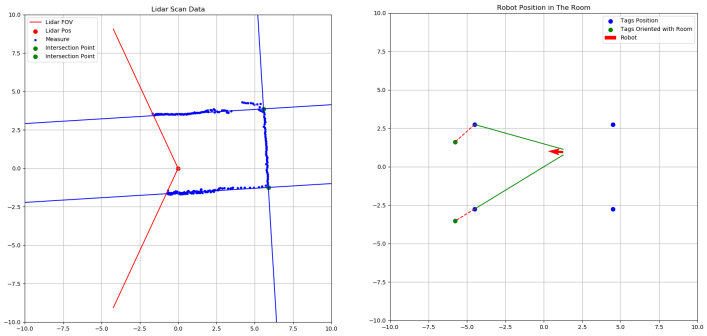

Figure 4: Results of the localization from landmarks (right) with corresponding LiDAR data (left)

We can see on the plot that shows the robot position on the room that the blue dots are the corners of the room in the room frame, and the green dots are the corners detected by the LiDAR if the robot was at coordinates $(0, 0)$, but oriented as the red arrow. That is a good way to understand which landmark is connected to which corner. The green lines are the distance between $(0, 0)$ and a green dot, but translated to start from the closest blue dot, so in the room frame. At the other end of the green lines we find the robot pose.

But the big issue with this method is that the four landmarks are indistinguishable from the others, so we don't really know which green dot should connect with a blue dot, and we can quickly end up in a situation like Figure 5. The four green dots form a vertical rectangle instead of an horizontal one, and the output position is broken.

This situation happens because we don't know the robot orientation. We

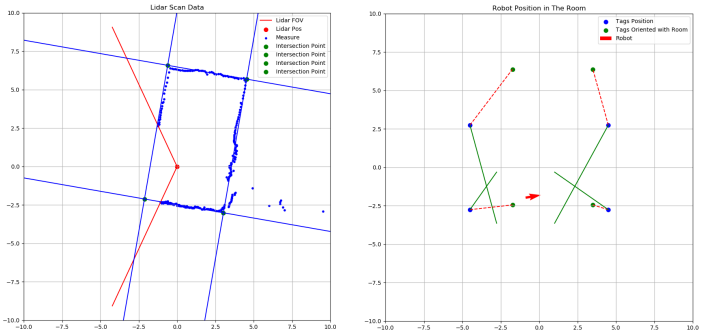

Figure 5: Results of a failed localization (right) with corresponding LiDAR data (left)

only know that the robot makes an offset α with the walls main angle θ. But there is not only one wall, and the four walls got those angles: $\left\{\theta, \theta + \frac{\pi}{2}, \theta + \pi, \theta + \frac{3\pi}{2}\right\}$. So there are 4 different positions for α too, and we can end in the situation above. We can partially solve this problem thanks to the different walls' size, by adding $\frac{\pi}{2}$ to α when we detect that the robot is facing the wrong wall's size. With this solution, the problem of the Figure 5 does not appear anymore, and the program seems to work well, but there is always 2 differents possible position for the robot, which is not convenient. Even if we start with the correct orientation, some errors may occur, and the robot position could pass from one side of the room to the other from the software point of view, which can cause a fail of the mission.

There are two main solutions to face this issue: the first one is to add a compass to the robot, because it does not depend on the symmetry of the room, and the second on is to break the room's symmetry by adding an object in one corner for example, but this program is paradoxically based on the symmetry to find the walls' position, so that's not a viable solution.

At this point, it appears that this program is too limited for the mission, firstly because of the symmetry issue, but also because the goal of the project is to work in several environments ; even if all the tests will be done in the testing room, that is not a good idea to limit the use of the project to square or rectangular rooms. That is the reason why I decided to start again from scratch with the localization algorithm, even if this solution may work with a compass in a rectangular room.

Second method: SLAM and AMCL algorithms

The new method which I came up with is a SLAM algorithm. Because this method is really popular for locating a robot with a LiDAR, there is plenty of documentation and libraries implementing this method, sometimes with the

ROS communication already included. That is the case for Gmapping [7], the software that I used to map the testing environment.

SLAM algorithm is made for simultaneously create a map of the environment and locate a robot inside this map, using LiDAR data and odometry. Firstly, the algorithm creates an occupancy map with the robot as the origin, and fills the map with the LiDAR data. Once this is done, it will wait for the robot to move, and thanks to the odometry, it will know where the new position approximately. At this point, it will fix the robot position by comparing the map and the new LiDAR data, and update the map. By repeating these actions the robot is always well located, and a map is created around it.

This method works well in complex environments, because it is easy for the program to fix the position when the point cloud is really different between one frame and the previous one, but in environments like long corridors, it is less accurate, due to the similarity between two frames of the corridor, and in such situations, the program relies on the odometry to estimate the motion of the robot, and that may cause bad distance estimations.

In the situation of my project, I need the robot to move in a fixed frame, common with the UAV, so they can perform a mission with the same origin. That's why I don't want a new map at each mission launch, but a fixed map for all the tests. So I decided to use the SLAM method to map my whole environment with the robot operated by a controller, then to use another algorithm to locate the robot in the map: AMCL. This time again, I used a library implementing the algorithm and the ROS communication [8].

This algorithm works by simulating several positions that the robot could take during the next frame, then it compares the LiDAR data to the occupancy map for all the simulated positions, and it selects the position with the minimum error between map and sensor data. To find the new list of hypothetical positions, the program uses the odometry, and randomly places some points around the position where the robot should be. There are a lot of different parameters to set up to find a correct position, such as the number of positions tested each frame or the noise on the odometry measurements (a lot of noise means that the positions tested can be further from the position predicted by the odometry).

Setting up all the parameters is a long but necessary step to optimize the computation, which needs a lot of processing and can be slow if the parameters demand too much calculation. It is a balance between performance and accuracy, especially in symmetrical environments, where each detail can help to break the symmetry.

The maps that I made for the mission can be found in Figure 6, and a visual plot of the AMCL method can be found in Figure 7.

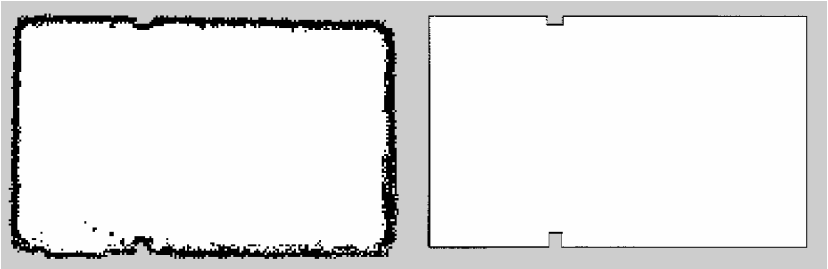

Figure 6: Maps obtained by SLAM method in real room (left) and in simulation (right)

We can notice that the real SLAM map is noisier than the simulation one, that's because a simulation has very few defaults, and the noise on the odometry is less important, so the result is cleaner. On both maps, we can notice the presence of beams that can help to break the symmetry, but they are only a small detail compared to the whole measurement so this is not optimal.

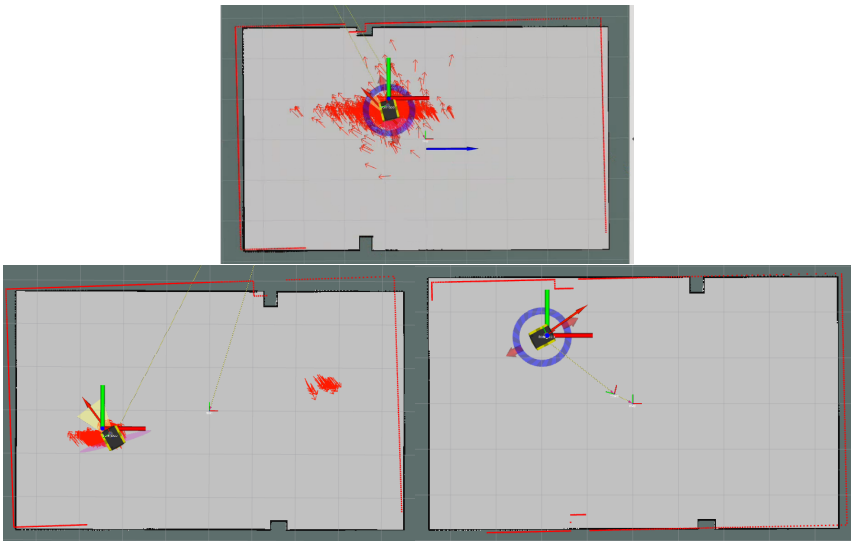

Figure 7: Example of AMCL algorithm and highlighting of symmetry issue

In Figure 7 we can see three examples of the AMCL method, with different parameters or levels of convergence. On each image, there is a 3D model of the robot that represents its position, a cloud of arrows that represents the next possible positions of the robot according to AMCL (except on the third image where the arrows are hidden), and a cloud of red points that represents the LiDAR data.

In those tests, I set up the initial position to $(0, 0)$ but with a lot of covariance on this value, so the program will begin with a really random cloud

of arrows that covers the whole room. On the first image we can see an early state of the localization, where the positions begin to converge toward the robot position, with the arrows approaching the robot position. This phase can last several seconds, and I have chosen to let the robot rotate but not move during the convergence for security reasons, because at this point the robot does not know precisely its position. The rotation helps the program to converge because it collects data with different angles. When the convergence is done, we can see two main output. Either the robot converges to only one position and it keeps it for the rest of the mission, or it converges to two separate clouds that have good results due to symmetry, and the output of the program switches from one cloud to the other. That is what happens in the second image, and that is really annoying because the robot can behave unexpectedly. On the third image, we can detect that the LiDAR data do not overlap well with the map shape, but only around the beams. The robot shows here again a symmetry issue, because it has converged toward the wrong position.

The solution for this issue is to set up an initial pose with little covariance. In that case, the algorithm will trust the initial pose and will begin already converged. But this means that we need to know where the robot will start, because if we begin the mission with a wrong initial position, the program will struggle to get back to its real position because with little covariance, it will only try new positions around its current position.

Overall, the AMCL localization works well. The major indicator of success is the overlapping of the LiDAR data with the map, and with a known initial position this was a success, with sufficient precision to control the robot.

Third Method: Localization by GNSS

Even if the project was focused on the GNSS-denied environment, Dr. Pickering asked me to try adding a localization by GNSS, because that is always useful if we want to try it outside one day. To achieve this, I have chosen to use a ROS library named robot_localization [6], and I especially tried to follow the documentation about navsat_transform_node [5].

Since I don't have any compass on the robot, I have to find another way to compute a heading. The best way that I found to achieve that is to use the course over ground. That means that I take two consecutive measurements and I use the difference between them to compute an angle. A schema of this method can be found on Figure 8.

This method is also useful to compute the speed of the robot, but at low speed, the proximity of two consecutive points reduces considerably the accuracy of the measurement, especially for the angle. But we need the heading of the robot at any moment, so we need to fuse multiple sensors in an EKF. To compute the angular velocity, we can use the IMU but also the odometry. By using the course over ground heading as a reference when the linear speed is sufficiently high and the integration of angular speed when the linear speed is too low, the robot should know its heading at any moment. The only problem with this

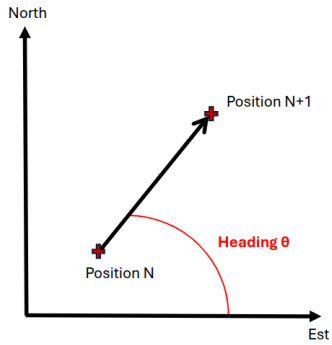

Figure 8: Schema of the course over ground

method is the initialization, because we need to have a sufficient linear speed to get the first reliable heading, but putting such speed without knowing the exact position of the robot is not really safe.

And for the position in a local frame, I simply use a transformation provided by the robot_localization library.

Finally, this method is easier to understand than the others, but I got several issues while implementing the EKF and the navsat_transform_node, and since that was not the priority, it was not working perfectly when I stopped working on it. Therefore, I don't have any kind of result to check the accuracy of this method, and even if I had some encouraging results, I can't prove it was really working.

If we use the GNSS localization instead of the SLAM one, the software architecture becomes as shown in the diagram on Figure 9.

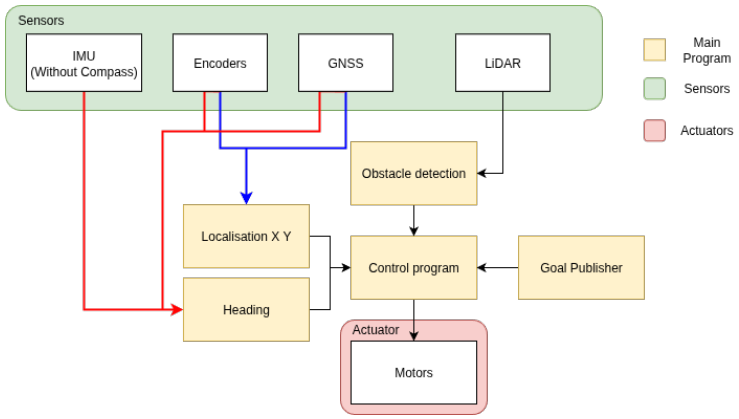

Figure 9: Software architecture with GNSS

Control and obstacle avoidance

Now that we can locate the robot in a map, we can begin to control it. The main goal is to let the robot follow a target. This target can be a registered path, so the drone can follow this path while carrying the UAV, but it can also be the position of the UAV, because the project for the UGV is to carry the battery for the UAV, and remaining under the aerial drone when there is no obstacle seems to be a good option.

To achieve that, I decided to use a ROS node that publishes continuously a position named Goal, computed from an equation. This equation can be anything, and can even be replaced by a list of points if preferred. The control program can read this goal position, and use it in its computing. The objective of this architecture is to be able to replace this publisher node by anything else that publish on the same topic, which can be the UAV. Thanks to this, it is easy to switch from the path following to UAV following, depending only on what publishes on this goal topic. For the following, I will mention this goal position as $\overrightarrow{X}_{goal} = \begin{pmatrix} x_{goal} \\ y_{goal} \end{pmatrix}$.

PID controllers and gains determination

Thanks to the programs already installed on the robot by the company that built it, I can directly control it using a linear speed v_{lin} and an angular speed v_{ang}.

To compute the correct speeds to reach the goal point, I decided to use a PID controller for each speed, based on the distance d between the robot position and the goal position for the linear speed, and on the angle error between the heading of the robot α and the direction between the robot position and the goal position. I will write those relations directly in equations, and I will mention the robot position as $\overrightarrow{X} = \begin{pmatrix} x \\ y \end{pmatrix}$.

$$\overrightarrow{d} = \overrightarrow{X}_{goal} - \overrightarrow{X} \tag{1}$$

$$d = \| \overrightarrow{d} \| \tag{2}$$

$$err_{ang} = \alpha - \arctan\left(\frac{d_y}{d_x}\right) \tag{3}$$

$$v_{lin} = K_{plin} \cdot d + K_{dlin} \cdot \dot{d} + K_{ilin} \cdot \int d \cdot dt \tag{4}$$

$$v_{ang} = K_{pang} \cdot err_{ang} + K_{dang} \cdot \dot{err}_{ang} + K_{iang} \cdot \int err_{ang} \cdot dt \tag{5}$$

Where K_{plin}, K_{dlin} and K_{ilin} are respectively the gains of proportional, derivative and integral terms for the linear speed, and K_{pang}, K_{dang} and K_{iang} the same gains for the angular speed. To determine those gains, I made a lot

of tests with different values but the same protocol. The test was following a trajectory in the shape of a Lissajous curve of size 2.5 meters by 1.5 meters with a revolution time of 30 seconds. With different gains, the results were not the same, and to compare them, I used two error mesurements: The distance to the path and the distance to the goal point.

The distance to the goal point is basically the distance between the robot and the goal point at each frame, and it is a representation of the proximity to the goal over the mission. To consider the whole mission, I used the absolute integral error computed with this value, which is the sum of all the errors over the duration of the mission. I was careful to take each time 30 seconds of mission, but only once the robot reached a stable motion behind the goal, and not taking into account the big errors corresponding to the beginning of the test where the robot was at one side of the room and took time to reach the point at the other side.

And the distance to the path is the distance between the robot position and the whole Lissajous curve, to check that even if the robot if far from the current goal point, it is able to follow the path with a little delay. This time again I used an absolute integral error to sum the error on 30 seconds of mission.

I began to make a lot of tests, to check by sight if they were efficient, and I registered 20 of them which were good enough to need the error measurement to compare them. On those 20 tests, no one got any derivative term, because the implementation of this term was not efficient. The complete list of tests is available in appendix on figure 14.

After the data analysis I found that the 2 tests that are the best are the 7th and the 10th tests. The 7th test got the best result for the distance to path error, and has only a proportional term, and the 10th test got the best result for the distance to goal point error, thanks to the addition of an integral term on its linear speed. Here are the plots of their trajectory overlapped with the goal path on Figure 10.

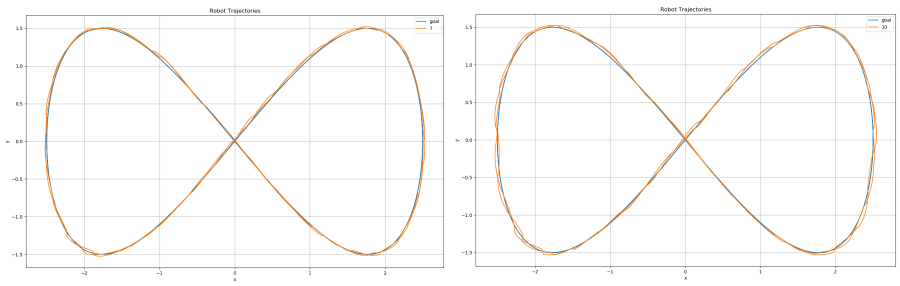

Figure 10: Trajectories of the robot on tests 7 (left) and 10 (right)

We can see that both of the trajectories are good and very close to the goal curve, but as the result showed before, the 7th test is better for the path following. The cause of this difference is that with an integral term on linear

speed, the 10th test is a bit faster than the 7th test when it approaches to the goal point, but the counterpart of this integral term is a higher speed on the turns when the robot needs to reduce its speed to not exit the trajectory. We can see on the plot of the 10th test that the big turns are messier than on the 7th test.

For the other indicator, the distance to the goal point, we can't really see on the plot what the results are, but the mean distance between the 10th test and the goal point is approximately 18 centimeters, while the mean distance on the 7th test is approximately 15 centimeters, which is pretty close with regard to the size of the robot. Since the two results on this indicator are similar, but the results on the path following are better for the 7th test, we decided to keep the gains of the 7th test.

Those gains are $K_{plin} = 3$ and $K_{pang} = 3.18$, with all other gains set to 0. That corresponds to a maximum linear speed of $v_{lin} = 1.5\,{}^{\mathrm{m}}/_{\mathrm{s}}$ reached when the robot is at a distance $d = 50cm$ or more from the goal point, and for the angular speed it corresponds at a maximum value of $v_{ang} = 2.5\,{}^{\mathrm{rad}}/_{\mathrm{s}}$ reached when the robot is at an angle error $err_{ang} = \frac{\pi}{4} rad$ or more.

Obstacle avoidance

The robot is now able to follow a target published on the goal topic, but there is no safety consideration at this point. If on the trajectory of the robot there is a wall or an obstacle such as a human, the robot will not stop, and there is a risk of breaking equipment or harming someone. The robot then needs an obstacle avoidance system to prevent any damage. I had several ideas for implementing this safety feature, and the first one is to simply check if there is any LiDAR data too close to the robot in its trajectory, and slow down or stop depending on the range of this data. This is working pretty well for the needs of the project, but if this obstacle is a wall, the robot will be stuck forever in front of it. So I upgraded it by stopping only the linear speed and not the angular speed. This causes the robot to rotate in the direction of the goal until the obstacle moves or until the line to reach the goal does not cross the obstacle anymore. This method is not perfect, because the robot could stop forever depending on the situation, but the safety requirements are checked.

After that, when I had time, I tried to upgrade this system by using an artificial potential field, where the goal point is represented as an attractive potential, and the obstacles are represented as repulsive potentials, like in this exercise that I made in class last year [3]. In my situation, I took all the LiDAR points as repulsive potentials, considering that each point is a new obstacle. But even after multiple tries, this method did not work, probably because of the big number of obstacles considered. I wanted to pass more time to fix this issue and come up with a solution, but I did not have time to work on this feature, as it was not the priority in the project since the safety requirements were checked by the first method.

UAV consideration

Another safety consideration is the presence of the UAV directly tethered to the UGV. Since the tether is not of an infinite range, we need to consider that in the control program. The feature's addition is easy, but depends on which robot follows the other. If the UAV follows the UGV, in a situation of landing procedure for example, we add a maximum distance. If the 3D distance between the 2 robots exceeds 90% of the tether length, the UGV stops, waiting for the UAV to reach the operating zone near the UGV. If the distance between the two robots is between 50% and 90% of the tether length, the UGV will slow down linearly to allow the UAV to reduce the distance.

At the opposite, if it is the UGV that is following the UAV, we can't do anything but send an emergency status on the communication to alert the UAV that we can't reach it. The UAV should have some emergency parameters to go back to the UGV when this situation happens. But this situation may signify the end of the mission if the UGV can't reach the UAV position.

And in a special situation where the two robots are not following each other but a target, which can be the same target or two different ones, the UGV will switch target to follow the UAV if the distance is higher than 90% of the tether length. So if the two targets are too far from each other for our system, the safety is automatically prioritized, and the UAV will continue its mission while the UGV will try to follow its target as best as possible, but remaining sufficiently close to the UAV for not stretching the tether.

The team and I were aware that there are a lot of other safety issues with our tethered system, such as an obstacle between the two robots which can trap the tether, but we admit that this situation will not appear in our testing room, because this was unfortunately too complex to solve.

Help on other parts of the project

Even if my task was focused on the software part of the UGV, this is only a part of a bigger project, and the implementation with the other parts needs my implication. Also, I was the only member working full-time, because I was an intern while the others were students, and so did not have the same time as me, so I could handle more parts than just mine. That's why I sometimes worked with the others to help them on their task.

The second part that engaged me was the communication between the UGV and the UAV, because this is really related to my part as well. I helped my colleague in charge of this part to find which method to use to complete the communication. The system is more complex than just 2 robots: There is the UGV, which carries the ground station that manages the tether, and at the other end of the tether there is the UAV. The communication system needs to connect the 3 parts of the project, because they all need to get information from the others. Also, the UGV was the only robot to work with ROS, while the other parts were using Matlab Simulink, which is the main tool used at

Aston University. So we needed to find a way to communicate between ROS and Simulink.

Our first thought was to use a serial communication, either through the tether, or through a radio communication, because it could be easier to only have one tether for the power supply and not add two others for the communication. The communication worked between two devices, but adding a third one into the chain complicates the whole thing and we gave up on this method. After that, my colleague searched for different methods and he decided to use the CAN bus to connect all the devices in one communication system. So I searched for a bridge between ROS and CAN, and I wrote a node to adapt it to my software architecture and the communication was done.

The next main part on which I worked is the localization system of the UAV. My colleague decided at the beginning of the project to navigate in dead reckoning only, because the only sensors that were already on the UAV were an IMU. But the big issue of this method is that it is not accurate after a few seconds if that is not corrected by other sensors that provide an accurate localization to correct the position. The other issue is that he noticed that it is not a viable option only one week before my leaving, and he had a lot of other things to carry out about the control part of the drone, so I got the task to find another way to locate the UAV.

On the suggestion of another colleague, I tried to adapt my SLAM algorithms to make them work with another LiDAR connected to a Raspberry Pi 3 which can be embedded on the UAV. I successfully got the LiDAR data on the Raspberry Pi, but I faced a big problem when I tried to use AMCL to locate the new system within the room map: I did not have any odometry from the UAV. The odometry is essential to predict where the next possible positions of the robot will be, and without odometry, the localization fails. So I tried to compute odometry with the IMU, but the double integration of the acceleration is too noisy and does not work at all. After spending several days on this issue, I remembered the first method that I used at the beginning of the project, which is the localization by landmarks. I re-used my code to implement this method on the Raspberry, and it works pretty well. Unfortunately, the IMU which my colleague provided me does not have a compass either, so the issues that I mentioned before are always present, but for a solution in a hurry that was sufficient. Unfortunately, I had to leave the United Kingdom before finishing the implementation with the UAV, but I left all my code for my colleague to finish it.

When I think again about the UAV localization in a GNSS denied environment, I don't think that the LiDAR is the right sensor to use. Maybe a solution with a camera and some ground visual landmark should have been better, but I did not have time to implement this solution.

Second project: Digital twin of a quadruped robot

At some point during my internship, when the main functions of the UGV project were completed, Dr. Pickering came up to me with a new project: Apian, a company working on the development of autonomous systems to simplify healthcare, had just acquired a new wheeled quadruped robot, the Go2-W developed by Unitree, and needed help to create a control model for it. The goal of the project is to enable the robot to navigate autonomously within a hospital. Especially, the robot must be able to interact with a lift by pressing the button with a robot arm.

This project immediately interested me, because it seems to necessitate a lot of different skills, such as control algorithms and computer vision. But there was one problem, the Aston Robotics Lab does not own a version of this robot, so we needed to work on a simulation, but as I searched online, no one was already existing. The first task is then to create the simulation.

Go2-W description

The Go2-W is a quadruped robot with wheels developed by Unitree. The use of wheels allows the robot to move faster than traditional quadruped robots, while its legs allow it to navigate in complex environments. The robot is equipped with several sensors, including camera, LiDAR, and IMU, which allow it to perceive its environment and navigate autonomously. In addition, the robot contains 12 motors, 4 for each leg, which allow it to move its legs and wheels independently.

Creation of a simulation

The goal of the simulation is to include a simulated version of the Go2-W in an environment with obstacles and lifts, to develop and test software for Apian. To achieve that, I need the model of the robot, and then to add the necessary joints and sensors on it. Once this is done, I will need to control those elements to make the robot move as the real one.

Go2W Robot Model

To create the robot model in xacro format, I used the URDF file provided by Unitree, and I also inspired myself from a simulation of the non-wheeled version of the robot, created by anujjain-dev on GitHub [1]. This model includes the robot's geometry, sensors, and motors, allowing it to be used in a simulation environment such as Gazebo. For now, the model works quite well in Gazebo, but some adjustments are still needed to improve the robot's stability, especially regarding the wheels contacting with the ground.

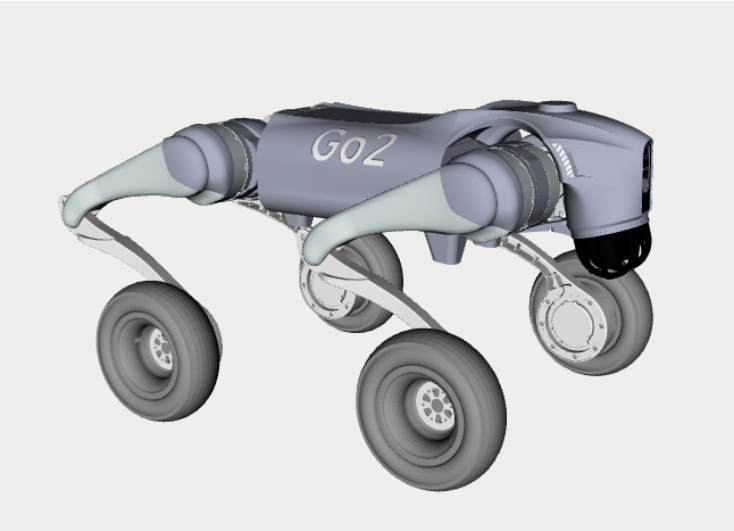

Figure 11: Go2-W model in Gazebo

Controlling the joints

The first step to control a quadruped robot is to control each motor independently. To achieve that, I used a framework named ros2_control to connect Gazebo with a ROS2 architecture.

I want to control the robot's legs joints (excluding the wheels) with a position controller, but with a constraint on the torque applied to the motors to make it safe and realistic. To achieve this, I used the joint_trajectory_controller set up with a command interface "effort". That means that we can send an angular position command to the controller, and it will compute the necessary effort to apply to the motors using a PID controller. The effort and position are limited by parameters set in the xacro file of the robot model.

To control the wheels, I use a joint_group_velocity_controller to send directly a velocity command to the wheels.

With those methods, I can control each joint of the robot directly by publishing a command in a ROS2 topic, which is really convenient for implementing a higher level control.

High level control

Before implementing the autonomous navigation, we need to implement a high-level control that will allow us to control the robot using simple commands. To achieve this, I implemented a software named Champ [2] that is made to control quadruped robots, but without wheels. The high level command is sent by a topic /cmd_vel. This message contains the X and Y linear velocity and the Z angular velocity of the robot, and is interpreted by Champ to control the joints

in position, transforming the velocities into angular positions that make the robot walk with small steps. In addition, we can control the robot's wheels independently using the joint_group_velocity_controller previously mentioned. Ultimately, the /cmd_vel topic will be used to control the wheels too, so there will be only one topic to control the robot.

Software architecture

The software architecture of the simulated Go2-W on Gazebo can be found in Figure 12.

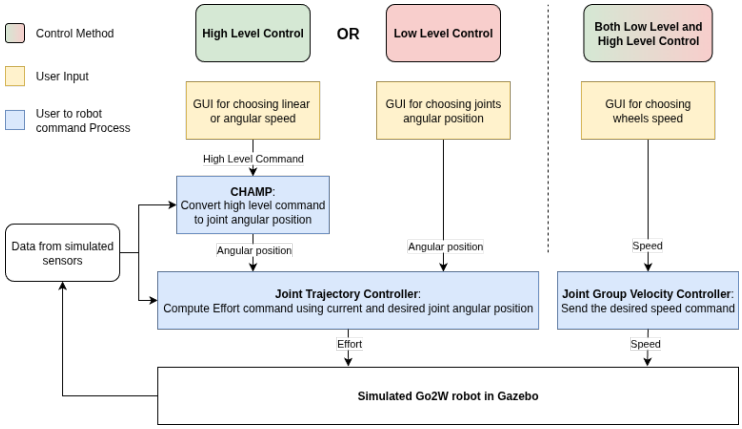

Figure 12: Go2-W model in Gazebo

Results of the Go2-W simulation and continuation of the project

After a lot of time implementing the robot in a Gazebo simulation, I ended with a mixed result. The robot spawns in the simulation with no problem, and I can control the joints as I want, but there are some issues about the high level control. I don't really understand why, but by using the Champ software on the Go2-W, I got very small steps, even when I modify the parameters, and the robot has no balance. That causes the robot to turn a bit when it is supposed to move straight, and the speed is also really reduced compared to a non-wheeled robot.

Despite this issue, the robot works in the simulation and we can try to go further in the project, even if we know that the dynamics is not the same as the real robot. But the next step was to implement a localization method, and I wanted to use a SLAM method because the Go2-W has a LiDAR, but I encountered some other issues with the library slam_toolbox and with the odometry of the robot.

I tried to debug each part of this localization program, but I don't really understand where the problem is. At this point, I had to come back to my first project because it was a priority, and we decided with Dr. Pickering to stop this project for now, as it takes too much time for too few results. This conclusion is sad because from my point of view the simulation part was only the preparation before beginning the project, and by not succeeding to create this simulation, I have the feeling of not having taken part in the project at all.

My simulation is available on GitHub even if the project failed [9].

Conclusion

Unfortunately, the tethered UAV project was not completed by the time I left the laboratory. Although my part of the work was mostly finished, we did not have the opportunity to test the full system to evaluate the communication and the interaction between the robots. It is always a bit frustrating to leave a project before seeing its final results, but since the rest of the team only stayed one week longer, I knew that the UGV–UAV tandem would not be operational that summer. Hopefully, the project will be continued next year, and thanks to the work we accomplished this summer, it should be much easier to complete.

Neither of the two projects I worked on at Aston University reached completion, but I still greatly enjoyed my time there. I learned a lot about ROS communication, SLAM algorithms, and simulation using Gazebo. All these skills that I developed at Aston University will be very useful for the remainder of my studies and for my future professional career.

When I decided to join the Aston Robotics Laboratory, my goal was to discover the world of scientific research. I was hoping to collaborate with researchers or PhD students, and to be honest, I was a bit disappointed to work mainly on student projects with limited ambitions. Even though the experience was valuable, it did not fully satisfy my curiosity about research work — as an intern, I felt more like a student than a researcher.

Finally, even though I went to the United Kingdom to discover its working culture, I mostly worked independently, which limited my immersion in this new environment.

To conclude, I had a great and enriching experience at Aston University, even if it did not fully meet my expectations as a student seeking a true research laboratory experience.

References

[1] GitHub anujjain dev. unitree-go2-ros2. Available at: https://github.com/anujjain-dev/unitree-go2-ros2/tree/humble, 2024. Accessed: 2025-09-01.

[2] GitHub chvmp. champ. Available at: https://github.com/chvmp/champ/tree/ros2, 2019. Accessed: 2025-09-01.

[3] Luc Jaulin. rob exo 4.7 : potentiels artificiels. YouTube video, 2015. Available at: https://www.youtube.com/watch?v=rwM4-hbvOio.

[4] Luc Jaulin. Kalmooc, exercice 25 : Localization of a robot in a rectangular room with a lidar. YouTube video, 2021. Available at: https://www.youtube.com/watch?v=V0hOejnwNEo.

[5] Wiki robot_localization. Integrating gps data. Available at: https://docs.ros.org/en/melodic/api/robot_localization/html/integrating_gps.html, 2016. Accessed: 2025-09-01.

[6] Wiki robot_localization. robot_localization 2.6.12 documentation. Available at: https://docs.ros.org/en/melodic/api/robot_localization/html/index.html, 2016. Accessed: 2025-09-01.

[7] Wiki ROS.org. gmapping. Available at: https://wiki.ros.org/gmapping, 2019. Accessed: 2025-09-01.

[8] Wiki ROS.org. amcl. Available at: https://wiki.ros.org/amcl, 2020. Accessed: 2025-09-01.

[9] GitHub Sam-Mag1. unitree-go2w-ros2. Available at: https://github.com/Sam-Mag1/unitree_go2w_ros2, 2025. Accessed: 2025-09-01.

Appendix

Assessment Report

ENSTA Bretagne

RAPPORT D'EVALUATION
ASSESSMENT REPORT

Merci de retourner ce rapport par courrier ou par voie électronique en fin du stage à :
At the end of the internship, please return this report via mail or email to:

ENSTA Bretagne – Bureau des stages - 2 rue François Verny - 29806 BREST cedex 9 – FRANCE
☎ 00.33 (0) 2.98.34.87.70 / stages@ensta-bretagne.fr

I - ORGANISME / *HOST ORGANISATION*

NOM / *Name* Aston University

Adresse / *Address* Aston University, The Aston Triangle, Birmingham, B4 7ET

Tél / *Phone (including country and area code)* 07359275763

Nom du superviseur / *Name of internship supervisor* Dr James E. Pickering

Fonction / *Function* Lecturer of Control Engineering

Adresse e-mail / *E-mail address* j.pickering1@aston.ac.uk

Nom du stagiaire accueilli / *Name of intern* Samuel Magni

II - EVALUATION / *ASSESSMENT*

Veuillez attribuer une note, en encerclant la lettre appropriée, pour chacune des caractéristiques suivantes. Cette note devra se situer entre **A (très bien)** et **F (très faible)**
*Please attribute a mark from **A (excellent)** to **F (very weak).***

MISSION / *TASK*

❖ La mission de départ a-t-elle été remplie ? A A B C D E F
Was the initial contract carried out to your satisfaction?

❖ Manquait-il au stagiaire des connaissances ? ☐ oui/*yes* ☒ non/*no*
Was the intern lacking skills?

Si oui, lesquelles ? / *If so, which skills?* _____

ESPRIT D'EQUIPE / *TEAM SPIRIT*

❖ Le stagiaire s'est-il bien intégré dans l'organisme d'accueil (disponible, sérieux, s'est adapté au travail en groupe) / *Did the intern easily integrate the host organisation? (flexible, conscientious, adapted to team work)*

A A B C D E F

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here* Sam was great - he worked with the team in a very professional manner.

COMPORTEMENT AU TRAVAIL / *BEHAVIOUR TOWARDS WORK*

Le comportement du stagiaire était-il conforme à vos attentes (Ponctuel, ordonné, respectueux, soucieux de participer et d'acquérir de nouvelles connaissances) ?

Did the intern live up to expectations? (Punctual, methodical, responsive to management instructions, attentive to quality, concerned with acquiring new skills)?

A A B C D E F

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here* Sam performed tasks to a high level, and also followed instructions well.

INITIATIVE – AUTONOMIE / *INITIATIVE – AUTONOMY*

Le stagiaire s'est –il rapidement adapté à de nouvelles situations ? A A B C D E F
(Proposition de solutions aux problèmes rencontrés, autonomie dans le travail, etc.)

Did the intern adapt well to new situations? A A B C D E F
(eg. suggested solutions to problems encountered, demonstrated autonomy in his/her job, etc.)

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here* _____

CULTUREL – COMMUNICATION / *CULTURAL – COMMUNICATION*

Le stagiaire était-il ouvert, d'une manière générale, à la communication ? A A B C D E F
Was the intern open to listening and expressing himself /herself?

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here* _____

OPINION GLOBALE / *OVERALL ASSESSMENT*

❖ La valeur technique du stagiaire était : A A B C D E F
Please evaluate the technical skills of the intern:

III - PARTENARIAT FUTUR / *FUTURE PARTNERSHIP*

❖ Etes-vous prêt à accueillir un autre stagiaire l'an prochain ?

Would you be willing to host another intern next year? ☒ oui/*yes* ☐ non/*no*

Fait à _____, le _____
In _____, on _____

I'd be very happy to host another student like Sam next year, if the opportunity arises.

Signature Entreprise James E. Pickering Signature stagiaire
Company stamp _____ *Intern's signature*

Merci pour votre coopération
We thank you very much for your cooperation

Figure 13: Assessment Report

Table of gains in Lissajous curve test

No	Control Method	Gains	Max Speed	Speed of ref	Absolute Integral error (Point Following)	Max Distance to ref (Point Following)	Absolute Integral error (Path Following)	Max Distance to ref (Path Following)
1&2	Proportional	Kp_lin = 0.75 Kp_ang = 1.27	Lin = 1.5 Ang = 2	Normal (30 sec per revolution)	Integral error: 43.9937 m Mean error: 0.7332 m	Max error: 1.0131 m Min error: 0.5114 m	Integral error: 3.7370 m Mean error: 0.0623 m	Max error: 0.1504 m Min error: 0.0004 m
3	Proportional	Kp_lin = 1.5 Kp_ang = 2.55	Lin = 1.5 Ang = 2	Normal (30 sec per revolution)	Integral error: 21.8034 m Mean error: 0.3634 m	Max error: 0.5412 m Min error: 0.2205 m	Integral error: 1.5614 m Mean error: 0.0260 m	Max error: 0.0637 m Min error: 0.0004 m
4	Proportional	Kp_lin = 1.5 Kp_ang = 2.55	Lin = 1.5 Ang = 2	Slow (60 sec per revolution)	Integral error: 11.6025 m Mean error: 0.1934 m	Max error: 0.2938 m Min error: 0.1163 m	Integral error: 0.5691 m Mean error: 0.0095 m	Max error: 0.0518 m Min error: 0.0003 m
5	Proportional	Kp_lin = 4 Kp_ang = 2.55	Lin = 2 Ang = 2	Normal (30 sec per revolution)	Integral error: 9.7683 m Mean error: 0.1628 m	Max error: 1.3840 m Min error: 0.0205 m	Integral error: 2.8934 m Mean error: 0.0482 m	Max error: 0.5439 m Min error: 0.0007 m
6	Proportional	Kp_lin = 3 Kp_ang = 2.55	Lin = 1.5 Ang = 2	Normal (30 sec per revolution)	Integral error: 9.6679 m Mean error: 0.1611 m	Max error: 0.2932 m Min error: 0.0832 m	Integral error: 1.3224 m Mean error: 0.0220 m	Max error: 0.0607 m Min error: 0.0008 m
7	Proportional	Kp_lin = 3 Kp_ang = 3.18	Lin = 1.5 Ang = 2.5	Normal (30 sec per revolution)	Integral error: 10.7012 m Mean error: 0.1784 m	Max error: 0.3329 m Min error: 0.0667 m	Integral error: 0.8849 m Mean error: 0.0147 m	Max error: 0.0468 m Min error: 0.0003 m
8	Proportional	Kp_lin = 3 Kp_ang = 2.38	Lin = 1.5 Ang = 2.5	Normal (30 sec per revolution)	Integral error: 9.6987 m Mean error: 0.1616 m	Max error: 0.2417 m Min error: 0.0951 m	Integral error: 1.2926 m Mean error: 0.0215 m	Max error: 0.0517 m Min error: 0.0000 m
9	Proportional	Kp_lin = 3 Kp_ang = 3.82	Lin = 1.5 Ang = 2	Normal (30 sec per revolution)	Integral error: 15.5061 m Mean error: 0.2584 m	Max error: 1.2860 m Min error: 0.1234 m	Integral error: 1.6167 m Mean error: 0.0269 m	Max error: 0.2312 m Min error: 0.0034 m
10	Proportional Integral	Kp_lin = 3 Kp_ang = 2.55 Ki_lin = 0.1 Ki_ang = 0	Lin = 1.5 Ang = 2	Normal (30 sec per revolution)	Integral error: 8.8519 m Mean error: 0.1475 m	Max error: 0.2328 m Min error: 0.0625 m	Integral error: 1.5937 m Mean error: 0.0266 m	Max error: 0.0565 m Min error: 0.0020 m
11	Proportional Integral	Kp_lin = 3 Kp_ang = 2.55 Ki_lin = 0 Ki_ang = 0.2	Lin = 1.5 Ang = 2	Normal (30 sec per revolution)	Integral error: 13.5256 m Mean error: 0.2254 m	Max error: 0.3364 m Min error: 0.1290 m	Integral error: 0.9697 m Mean error: 0.0162 m	Max error: 0.0457 m Min error: 0.0002 m
12	Proportional Integral	Kp_lin = 3 Kp_ang = 2.55 Ki_lin = 0.1 Ki_ang = 0.2	Lin = 1.5 Ang = 2	Normal (30 sec per revolution)	Integral error: 10.2372 m Mean error: 0.1706 m	Max error: 0.3061 m Min error: 0.0799 m	Integral error: 1.2275 m Mean error: 0.0205 m	Max error: 0.0917 m Min error: 0.0012 m
13	Proportional	Kp_lin = 3 Kp_ang = 2.55	Lin = 1.5 Ang = 2	Normal (30 sec per revolution)	Integral error: 13.5609 m Mean error: 0.2260 m	Max error: 0.3355 m Min error: 0.1309 m	Integral error: 1.0706 m Mean error: 0.0178 m	Max error: 0.0542 m Min error: 0.0008 m
14	Proportional	Kp_lin = 3 Kp_ang = 2.55	Lin = 1.5 Ang = 2	Slow (60 sec per revolution)	Integral error: 6.4876 m Mean error: 0.1081 m	Max error: 0.1677 m Min error: 0.0631 m	Integral error: 0.6324 m Mean error: 0.0105 m	Max error: 0.0333 m Min error: 0.0010 m
15	Proportional	Kp_lin = 3 Kp_ang = 2.55	Lin = 1.5 Ang = 2	Fast (20 sec per revolution)	Integral error: 17.5196 m Mean error: 0.2920 m	Max error: 0.4253 m Min error: 0.1471 m	Integral error: 3.0278 m Mean error: 0.0505 m	Max error: 0.0863 m Min error: 0.0208 m
16	Proportional	Kp_lin = 3 Kp_ang = 2.86	Lin = 1.5 Ang = 2.25	Normal (30 sec per revolution)	Integral error: 12.5517 m Mean error: 0.2092 m	Max error: 0.5839 m Min error: 0.1160 m	Integral error: 1.6670 m Mean error: 0.0278 m	Max error: 0.0877 m Min error: 0.0063 m
17	Proportional	Kp_lin = 3 Kp_ang = 2.86	Lin = 1.5 Ang = 2.25	Normal (30 sec per revolution)	Integral error: 11.5337 m Mean error: 0.1922 m	Max error: 0.2903 m Min error: 0.1124 m	Integral error: 1.6512 m Mean error: 0.0275 m	Max error: 0.0490 m Min error: 0.0085 m
18	Proportional	Kp_lin = 3 Kp_ang = 2.86	Lin = 1.5 Ang = 2.25	Slow (60 sec per revolution)	Integral error: 6.4874 m Mean error: 0.1081 m	Max error: 0.1634 m Min error: 0.0489 m	Integral error: 0.6808 m Mean error: 0.0113 m	Max error: 0.0398 m Min error: 0.0000 m
19	Proportional	Kp_lin = 2 Kp_ang = 2.86	Lin = 1.5 Ang = 2.25	Normal (30 sec per revolution)	Integral error: 16.5982 m Mean error: 0.2766 m	Max error: 0.4071 m Min error: 0.1690 m	Integral error: 0.9518 m Mean error: 0.0159 m	Max error: 0.0409 m Min error: 0.0006 m
20	Proportional	Kp_lin = 2 Kp_ang = 2.86	Lin = 1.5 Ang = 2.25	Slow (60 sec per revolution)	Integral error: 8.6909 m Mean error: 0.1448 m	Max error: 0.2233 m Min error: 0.0836 m	Integral error: 0.5727 m Mean error: 0.0095 m	Max error: 0.0457 m Min error: 0.0005 m

Figure 14: Table of errors depending on gains on Lissajous curve test
(Highlighted mesures are the minimum errors)

Acronyms

ENSTA	École Nationale Supérieure de Techniques Avancées
UAV	Unmaned Aerial Vehicle
UGV	Unmaned Ground Vehicle
GNSS	Global Navigation Satellite System
LiDAR	Light Detection And Ranging
IMU	Inertial Measurement Unit
ROS	Robot Operating System
SLAM	Simultaneous Localization And Mapping
AMCL	Adaptive Monte-Carlo Localization
EKF	Extended Kalman Filter
PID	Proportional Integral Derivative
CAN	Controller Area Network
URDF	Unified Robot Description Format