ROBOTICS PROJECT – ROBO4

School year 2024-2025

„GRABBY - the Warehouse Robot "

Final Project Report

Students:

Hendrikse Jérémy

Ortstadt Julius

Ecole Polytechnique Universitaire de Nice Sophia-Antipolis, Département robotique et systèmes autonomes
1645 route des Lucioles, Parc de Sophia Antipolis, 06410 BIOT

Table of content

Introduction

Overview

Grabby is designed to be an autonomous warehouse robot whose goal is to minimize cost and maximise efficiency in a real-world warehouse setting. To do this, it employs an array of sensors and algorithms to easily navigate the environment and perform the required tasks. This approach will long term reduce the number of errors in logistics and reduce the overall cost for the company.

The robot consists of two main parts which we will explain in further detail in this report.

The first part is the mobile base which houses the main computer, the motors, the power delivery system and some control boards as well as different sensors.

The second part is the lifting mechanism which contains a lifting and grabbing system that allows to pick up packages at different heights. There is also a stereo camera mounted on the top to assist on navigation and help localize packages.

The hardware for this project was made during the first development year in Robo3. This year, we focused on the autonomy of the robot which meant we developed the software that runs the robot.

GitHub

By following the link below, you can go to the GitHub repository for the Grabby project:

https://github.com/JeremyHendr/GrabbyBotics

The GitHub contains all the code and CAD files necessary to achieve the functionalities of the robot shown during the project presentation. Additionally, each code has its own explanation file to help with the setup and the understanding of how it actually works. Plus, we added some useful documents we found during our research and added some other files that can be helpful.

The repository is an addition to the reports and presentation to offer further insight into our project.

Objectives

Software development played a major role in this year's project. The main objectives for this were:

- Setup the different components of the robot to work individually and collaboratively.
- Achieve good mapping results.
- Achieve autonomous navigation.
- Create a ROS package for the platform and stereo camera
- Create a depth map from the stereo camera for package identification and additional navigation purposes

Some of those objectives were met and some not completely due to technical difficulties which we'll explain in this report.

The mobile base

Description

The mobile base of the robot is a critical part of the robot as previously explained. The different sensors and systems present need to work together to allow the robot to perform the desired tasks. Basically, the base must do the following:

- Drive the motors to move the robot to the desired position.
- Offer enough structural strength and stability to ensure that the robot can hold the weight of the upper platform and of the package.
- Map the environment.
- Navigate in the environment and update the saved map if unexpected changes occur.
- Position itself in the environment using the different sensors.
- Manage power delivery to all the necessary systems.

In order to do this, the structural part as well as the code behind it have to work. As previously said, the structure was already complete for this prototype, which meant that we had to develop the code behind everything.

Development

The development of the software for Grabbys' mobile base was split into different main parts:

- ROS
- Motors and encoders
- Physical parameter estimation
- IMU
- LiDAR
- Teleoperation
- Mapping
- Navigation

In this section, we'll go over each development step.

ROS

We chose to develop the software using **ROS1 Melodic**, a well-documented and stable distribution that is compatible with Ubuntu 18.04—the latest version for the Jetson Nano used on Grabby. Although ROS2 has more advanced capabilities and support for newer hardware, we needed a stable running system and support for the packages we either used or wrote ourselves. ROS provided the communication infrastructure to interconnect all components: motor controllers, sensors, SLAM systems, mapping algorithms and the navigation stack.

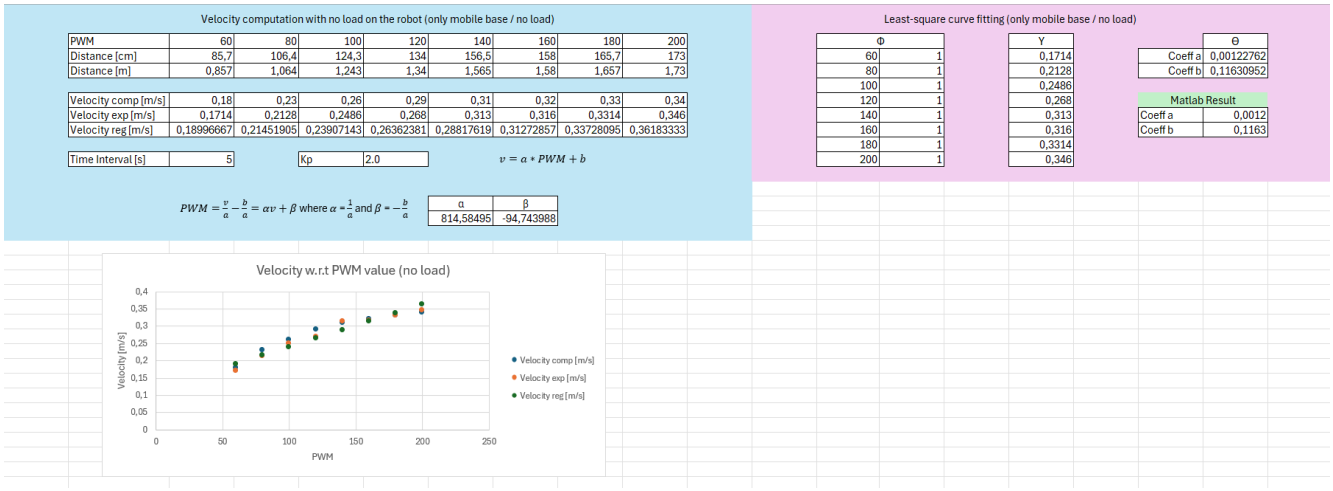
Motors and encoders

One of the major tasks this year was integrating and exploiting data from the motor encoders. These encoders output tick values that we used to derive several useful parameters, such as velocity, distance travelled, and wheel revolutions. We also calculated the drift between the two motors as they are not perfectly identical, and the base of the robot is not perfectly symmetrical. This determination allowed us to experiment and implement a drift correction mechanism.

This work was essential for implementing wheel odometry, which feeds directly into both the SLAM and navigation systems. To support debugging and development, we implemented Bluetooth-based data transmission. This allowed real-time testing (as we didn't have access to a very long USB cable) and adjustment of parameters, particularly useful when observing how drift correction improved the robot's trajectory.

Physical parameter estimation

To ensure accurate control, we needed to estimate the relationship between PWM signals and the resulting robot velocity [m/s]. For this, we implemented a batch least-squares curve fitting approach which we saw in our *Applied Estimation* classes. We collected experimental data and processed it using a custom Excel sheet that applied the least-squares approach. The resulting parameters allowed us to get the equation between PWM and velocity. This allowed us to fine-tune the drift correction and also accurately convert the velocity commands from the base controller of the navigation stack to PWM signals that could be applied to the motors directly.



Velocity computation with no load on the robot (only mobile base / no load)

PWM	60	80	100	120	140	160	180	200
Distance [cm]	85,7	106,4	124,3	134	156,5	158	165,7	173
Distance [m]	0,857	1,064	1,243	1,34	1,565	1,58	1,657	1,73
Velocity comp [m/s]	0,18	0,23	0,26	0,29	0,31	0,32	0,33	0,34
Velocity exp [m/s]	0,1714	0,2128	0,2486	0,268	0,313	0,316	0,3314	0,346
Velocity reg [m/s]	0,18996667	0,21451905	0,23907143	0,26362381	0,28817619	0,31272857	0,33728095	0,36183333

Time Interval [s]	5	Kp	2.0

$v = a * PWM + b$

$PWM = \frac{v}{a} - \frac{b}{a} = \alpha v + \beta$ where $\alpha = \frac{1}{a}$ and $\beta = -\frac{b}{a}$

α	β
814,58495	-94,743988

Least-square curve fitting (only mobile base / no load)

Φ	
60	1
80	1
100	1
120	1
140	1
160	1
180	1
200	1

Y
0,1714
0,2128
0,2486
0,268
0,313
0,316
0,3314
0,346

	Θ
Coeff a	0,00122762
Coeff b	0,11630952

Matlab Result	
Coeff a	0,0012
Coeff b	0,1163

Batch least-squares approach

IMU

The robot's inertial measurements were gathered using the **MPU-6050**, a 6-DoF IMU combining a gyroscope and accelerometer. While several existing ROS packages supported this IMU, we developed a custom ROS package to better control data retrieval, scaling and publishing.

This decision allowed us to implement complementary filtering—a method discussed seen in our Sensor Fusion lectures—to fuse accelerometer and gyroscope data for more accurate orientation estimates. This filtered orientation data was essential in stabilizing mapping and improving navigation performance by reducing sensor noise. This combined with the motor encoder data, proved essential in creating accurate odometry data for mapping and navigation.

LiDAR

For environmental awareness and mapping, we mounted an **RPLiDAR A1** unit on the mobile base, positioned between the robot's two main structural sections. This 2D LiDAR continuously scans the surroundings and produces a point cloud representing nearby obstacles and structures.

This sensor was essential for enabling LiDAR-based SLAM and navigation, providing real-time feedback for map generation and obstacle detection.

Teleoperation

The robot isn't able to map an environment autonomously, at least right now. So, for us the actually map the environment, we had multiple choices:

- Use a teleoperation package
- Use blueprints of the place in question and convert them into a ROS map
- Move the LiDAR e.g. the robot manually trough the place to map.

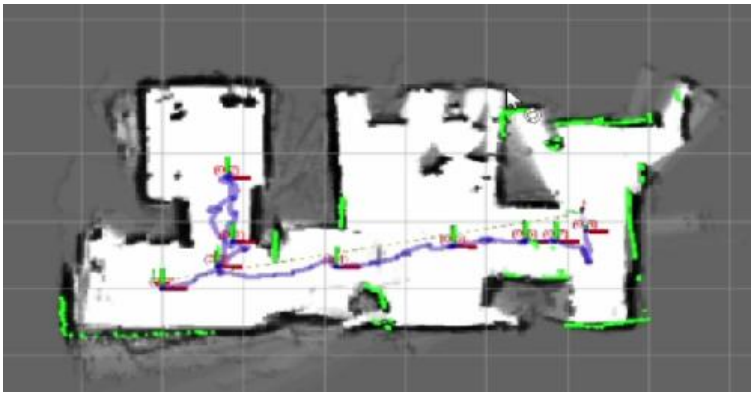
We choose to use a teleoperation package in ROS allowing us to simply move the robot which a button press on the keyboard of our PC. However, after looking at multiple packages we decided to write our own as it allowed more customization possibilities for our specific case of use.

The package can be found in the GitHub repository under the "Code -> Mapping" folder.

Mapping

To construct maps of the environment and localize within them, we used Google Cartographer, a real-time SLAM system. Cartographer supports multiple sensor configurations, and in our case, it combined wheel odometry, IMU, and LiDAR data to generate accurate 2D maps.

Compared to alternatives like Hector-SLAM (Heterogeneous Cooperating Team of Robots), Cartographer offered greater accuracy and robustness but mainly the support for multiple sensor configurations. It works by applying scan matching and pose graph optimization to incrementally build and refine a map of the environment.



Mapping example

Navigation

Once mapping and localization were working reliably, we moved on to the implementation of autonomous navigation using the ROS Navigation Stack (Navstack).

- Costmaps were generated from sensor data to identify obstacles, free space, and unknown areas. These 2D grid maps were essential for safe path planning and collision avoidance.
- We used AMCL (Adaptive Monte Carlo Localization) to determine the robot's pose on the known map. AMCL maintains a set of particles representing possible poses, updating them based on sensor data and motion estimates
- Sensor fusion was handled by an Extended Kalman Filter (EKF), implemented using a pre-made ROS package. The EKF combines noisy inputs from the encoders, IMU, and LiDAR to produce smooth, accurate pose estimates, which significantly improved navigation stability.
- For path planning and movement execution, we configured the Base Local Planner, which computed the velocity commands necessary to move the robot safely and efficiently towards its goals while reacting dynamically to obstacles. The base controller then converts the commands to PWM and applies it to the motors while also publishing the encoder values.
- The move base node incorporates a configuration for the navstack to work and to actually move the robot.
- The pose and goal converters were essential in converting the pose specified in the RVIZ visualization to an understandable message for the navstack.
- Different types of transforms were adding. The EKF and AMCL provided two transforms which we dynamic while the others were static as there was no physical change on the robot during movement.

To help visualize the planning and mapping processes, we developed several Rviz configurations showing the LiDAR scan cloud, costmaps, path planning outputs, and robot trajectory in real-time.

Problems and solutions

The development of the mobile base had several problems this year. Some were software based while some were due to the hardware.

On the hardware side, the slipping of the wheels and the bending of the frame, which is made out of a combination of PETG and wood, have been a challenge as the movement of the base was not uniform. Possible solutions for this include reinforcing the frame out of stronger materials and changing the wheels and the drive-unit.

On the software side, we encountered several challenges, particularly during the mapping phase. Setting up Google Cartographer on the Jetson Nano proved difficult due to the board's limited computational resources, which impacted real-time performance. In configuring the navigation stack, we also had to fine-tune the robot's physical and dynamic transforms to ensure consistent frame alignment across all components.

Another issue arose with the LiDAR scan data, which appeared both mirrored and inverted. We successfully resolved the mirroring problem, but the scan data still appears inverted when the

robot rotates. This is primarily caused by inaccurate yaw angle estimation from the current 6-DoF IMU. To address this, we plan to upgrade to a 9-DoF IMU, which would improve orientation estimation and correct the remaining inconsistencies.

Also, the visualization of the resulting map in RVIZ was problematic as we had to create a specific configuration for RVIZ to correctly display and relay all the data to the corresponding nodes.

Some other minor issues were also solved but consisted in mostly debugging the code and adapting it to fit our specific sensor configuration etc.

The lifting mechanism

Description

The lifting mechanism is a platform that can reach different heights, retrieve and store packages using a grabbing mechanism. The elevation of the platform is achieved using a scissor lift type system that can reach great heights while being compact at its lowest position. To grab the package, we use a suction cup connected to a vacuum system to pick up any size and form of boxes.

Specific electrical parts are all stored on this part of the robot, but it uses the Arduino board and Jetson Nano from the mobile base. Both parts are connected using Molex connectors to easily split both parts.

In future iterations, this electrical distribution may be switched or upgraded for a more modular approach.

Development

The development of the hardware part has been done and described in the previous year report. This year a focus on software was meant to be done.

The development objective where:

- Allow parallel working on the project even from home.
- Create a ROS package to operate the platform, this package can be added or removed from the mobile base software with no other impact on the robot capacities.
- Create a ROS package to handle the stereo camera, this package can be added to improve navigation and detect packages.

Achieving these objectives meant setting up a new Jetson Nano with the same software as the one in the mobile base and write custom drivers and packages to support our hardware and integrate as we want to meet our requirements.

Problems and solutions

Problems have been numerous and won't be explicitly described because they don't affect directly the robot's development.

The first issue was that the robot and work is split in half, except we had access to only one Jetson Nano, the core logical unit of the robot.

To allow parallel development we thought of creating a virtual machine based on the jetson nano image or equivalent. After multiple tries it came out that using when using the virtual machine, a specific **ros_lib** library for Arduino wasn't built correctly which made development impossible, at least using a virtual machine.

The second idea was to try using another jetson that we borrowed from another team. The setup wasn't easy, and we encountered lots of problems including a network interface issue that made the usage of Wi-Fi, thus the card impossible. This issue was later solved using a Wi-Fi USB dongle.

Finally having a working second jetson at 2 weeks from the deadline meant that there was just enough time to create a basic communication protocol between the Jetson Nano and the Arduino using ROS to operate the platform.

Future improvements and changes

As the project continues to evolve, there are several key areas we have identified for future improvements and system upgrades. These changes aim to improve the reliability, modularity, and overall performance of the robot in real-world warehouse applications.

Planned Changes:

- **Upgrade to Raspberry Pi 5**
  The current Jetson Nano setup poses compatibility issues, especially with newer ROS distributions. Migrating to a Raspberry Pi 5 will enable the use of ROS2, offering better real-time performance, enhanced communication. This will also improve the overall development of the project as the Raspberry Pi community offers more support on various topics.
- **Custom PCB Design**
  To improve wiring and hardware integration, we plan to design custom PCBs. This will not only reduce complexity and the chances of hardware-related bugs but also improve the robustness and maintainability of the robot.
- **Modular System Architecture**
  Adding modularity to both the hardware and software systems would allow for easier upgrades, repairs, and custom configurations. This could involve designing plug-and-play sensor modules for example.
- **Increased System Robustness**
  We want to increase the robustness of the robot by changing the used materials for the panels and the frame to prevent bending, deformation etc.

Targeted Improvements:

- **Installation of a 9-DoF IMU**
  The current 6-DoF IMU has shown limitations in accurately tracking the robot's yaw rotation, affecting localization and mapping. Upgrading to a 9-DoF IMU, which includes a magnetometer, would enhance orientation estimation and help resolve the remaining issues with LiDAR scan inversion during turns.
- **Power System Redesign**
  Implementing a dedicated battery system and redesigning the overall power architecture will allow for more stable voltage delivery and greater autonomy.
- **Drive System Optimization**
  We aim to improve the drive system by experimenting with better motors and controllers or adding feedback mechanisms such as current sensing or traction monitoring to create better correction algorithms etc.

Conclusion

Costs

We resumed the costs of the entire project up until now.

Last year we made a quick list of the component costs and the work costs. This totalled at 990,99 € for the components and 13.934,74 € with the work time.

This year we haven't ordered a lot of components mainly just two new IMU (MPU-6050 and MPU-9250) as well as the LiDAR which totals at 130€.

Plus, this year we worked 204,25h on the mobile base and 92h on the lifting mechanism. By considering (like last year) an average salary of 38k€ for 1600h of work, we get a total of 7036€.

This means that this year, further development of the robot would have cost 7166€

Finally, the entire cost of the project over the two-year period is of 21100,74€

Project

After already working during Robo3 on the hardware of the robot, this year we focused our efforts on the software and code of the robot. Despite hardware and software challenges, we successfully implemented key features such as mapping, teleoperation, and autonomous navigation.

The project helped us apply and deepen our knowledge in robotics, sensor fusion, and system integration. While some limitations remain—like the IMU accuracy and hardware robustness—we've identified clear paths for future improvements, including better computing hardware, a 9-DoF IMU. We were also able to directly use what we saw in our lectures, on our robot.

Bibliography

Below are some of the websites that helped us during the development of the robot this year:

- Various forums for the setup and debugging of the second Jetson Nano
- ROS Wiki
- MPU-6050 (User manual & Register Map)
- Guides on sensor data filtering (the links can be found in the description of the respective code on our GitHub repository)