R&D Project
Grabby

Jeremy Hendrikse & Julius Ortstadt

GitHub:
https://github.com/JeremyHendr/GrabbyBotics

Content

01

Robot Overview

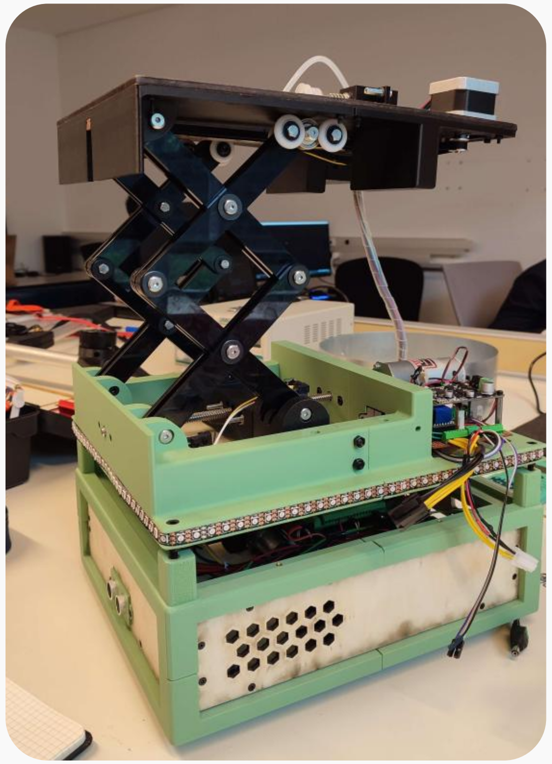
What is Grabby ?

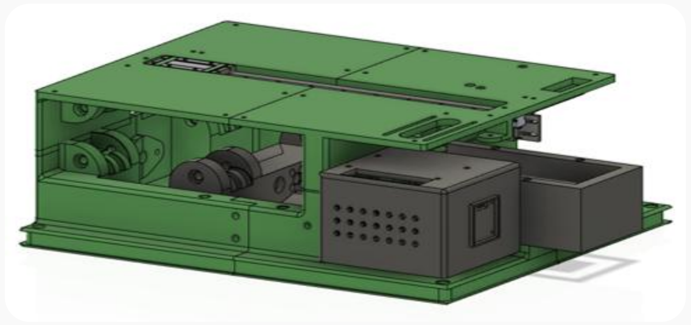
Grabby is designed to be an autonomous warehouse robot aiming to enhance productivity while reducing costs and errors in warehouse environments.

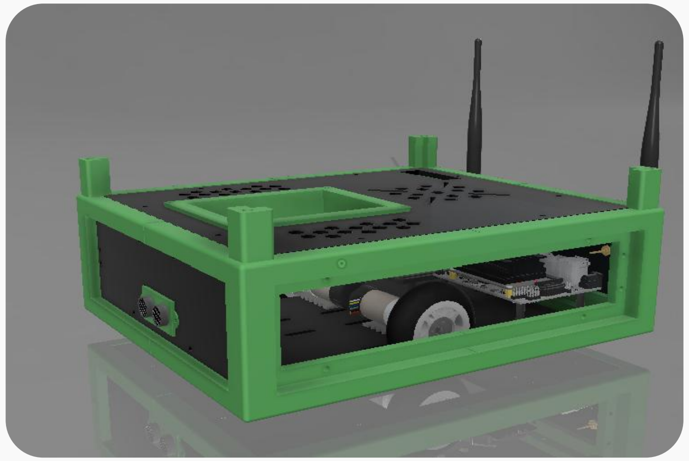
It is comprised of two main parts:

- The mobile base which houses the main processing, the battery and the movement capabilities.
- The lifting mechanism which enables the robot to identify the boxes, reach them and retrieve them.

The physical prototype was created during Robo3. This year, we focused our efforts on the software of the robot to achieve mapping, navigation and basic autonomy.







02

Mobile Base

Software development for the mobile base

Multiple steps were necessary for the development of the mobile base:

- Motor encoder implementation
- IMU (MPU-6050) implementation
- LiDAR (RPLiDAR A1) implementation
- Google Cartographer implementation
- ROS1 Melodic
- ROS1 Navstack setup
- Physical parameter estimation

Motor encoder

Motor encoder tick data is very important for:

- Odometry information.
- Correction factor determination.
- Physical parameter estimation.

We implemented a Bluetooth data transmission option utilizing either the Serial Monitor in the Arduino IDE or via a connection with PuTTY.

Challenges:

- Interference in the measurements.
- Bluetooth setup.
- Difference in the physical properties between the two motors.

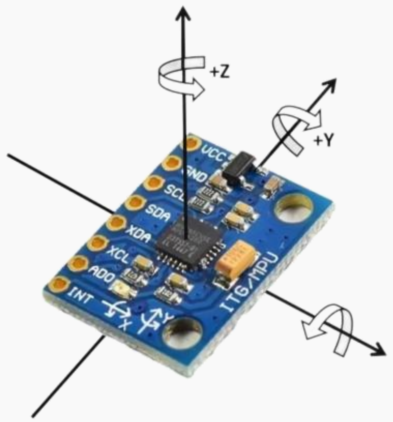
IMU (MPU-6050)

Implementing an IMU was a necessary step to get reliable odometry information that could be combined with the encoder data thanks to different filtering approaches:

- Complementary filter (testing and mapping).
- Extended Kalman Filter (EKF for navigation).

Challenges:

- Only 6-DoF meant no precise estimate of the yaw angle.
- Custom driver and ROS package for calibration and data acquisition.



MPU-6050 (6-DoF IMU)

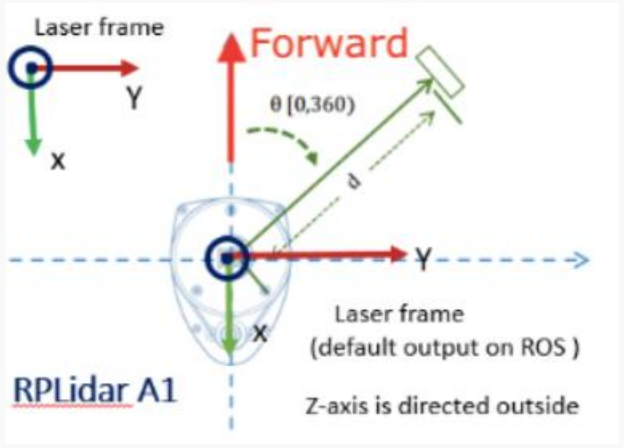
LiDAR (RPLiDAR A1)

The LiDAR sensor is the main part necessary for the mapping and navigation.

- Scans the environment in a 2D plane.
- Gives distance measurements and a perception of the environment.

Challenges:

- Correct LiDAR tf frame orientation and setup for mapping and navigation.
- Precise robot footprint determination to configure the LiDAR correctly.



Google Cartographer

Google Cartographer was chosen over HECTOR-SLAM (Heterogeneous Cooperating Team of Robots) to perform the mapping process as it allows for odometry information to be used to increase precision, and the overall algorithm is more effective to create good maps for navigation.

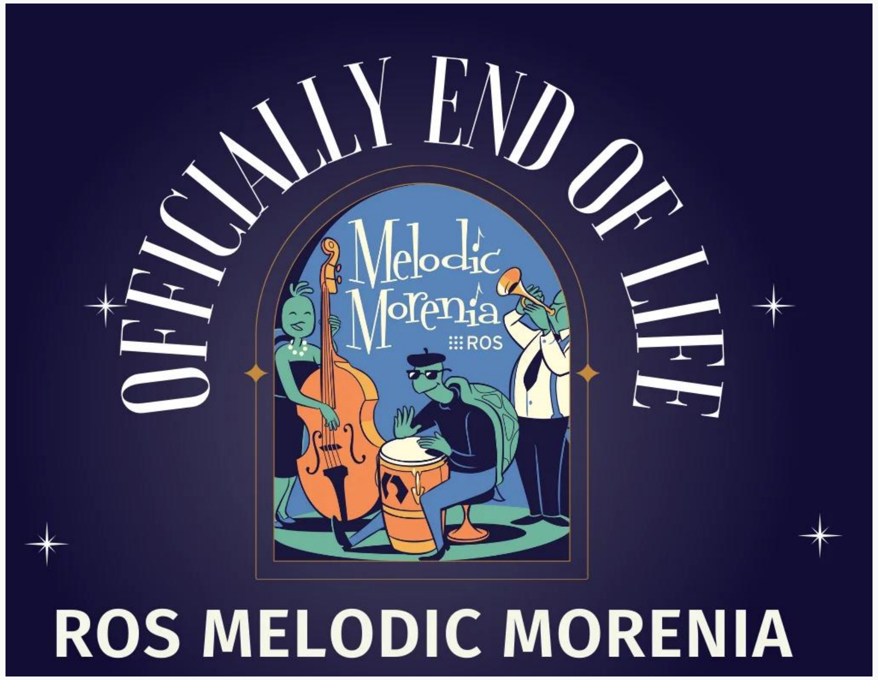
Challenges:

- Google Cartographer installation for ROS1 Melodic as it is mostly intended for ROS2.
- Configuration of the mapping with different sensors.
- Configuration of the RVIZ environment.

ROS1 Melodic

We are running Ubuntu 18.04 LTS on the NVIDIA Jetson Nano which only allows us to run efficiently a ROS1 (Melodic) implementation.

This distribution offers lots of documentation making it easier to work with, although it is already EOL (End-Of-Life).



ROS1 Navstack setup (1)

The ROS1 Navstack offers the capability of autonomous navigation using SLAM and different sensors.

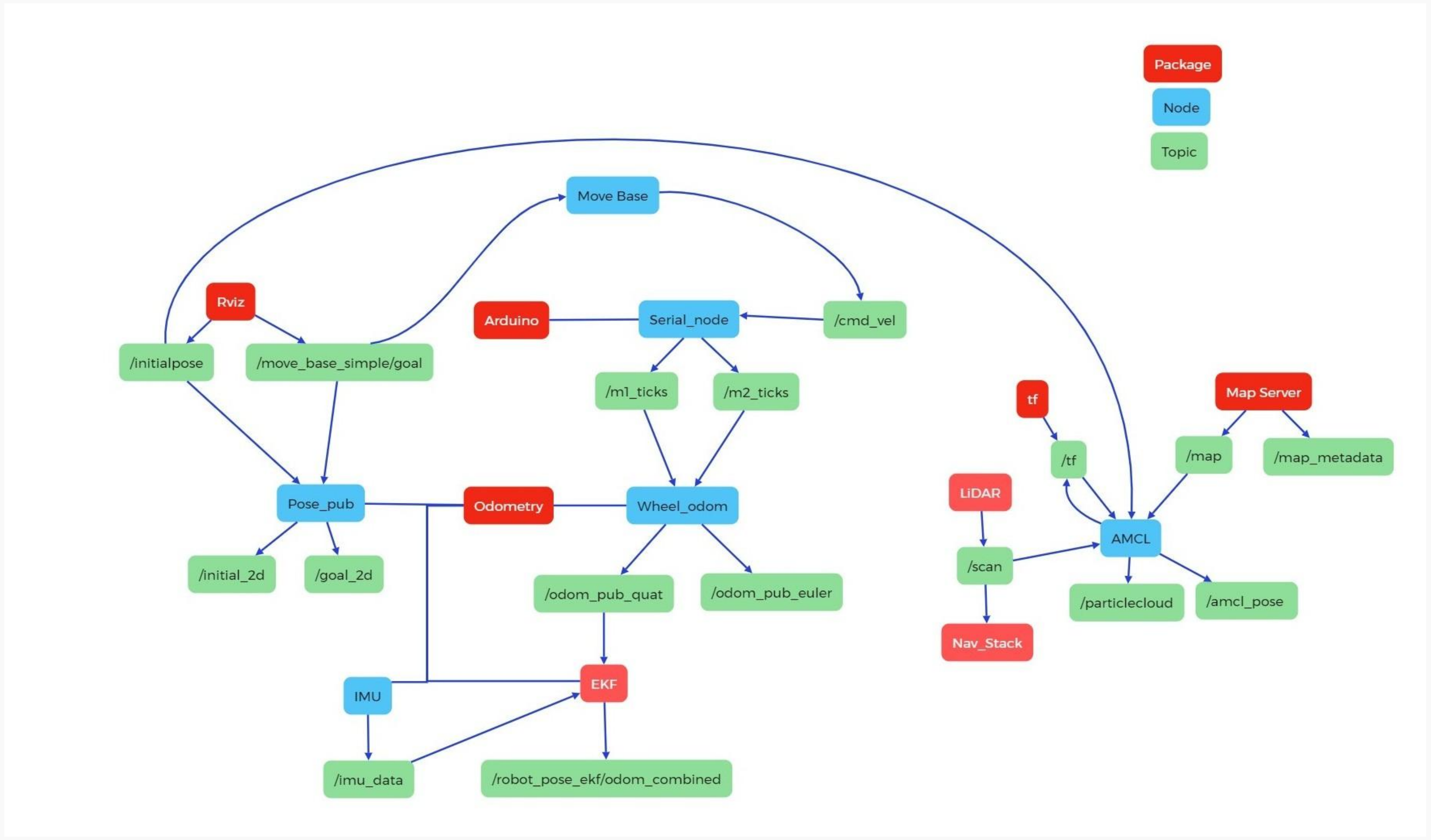
For it to be setup it needs different things:

- Costmaps (for global and local map updates)
- RVIZ visualization configuration
- AMCL (Adaptive Monte Carlo Localization) configuration
- LiDAR scan data
- Odometry information (IMU + Encoders)
- Data filtering with an EKF (Extended Kalman Filter)
- Base controller
- Map
- Pose and goal converter
- Transforms
- Move base node
- Base local planner configuration

ROS1 Navstack setup (2)

Challenges:

- Static transforms
- LiDAR data orientation
- Costmap configuration
- Map precision
- AMCL configuration
- 6-DoF IMU
- Odometry information precision



Custom ROS package for autonomous navigation

Physical parameter determination (1)

We needed to determine some physical parameters of the mobile base in order to apply certain drift correction etc.

The most important parameter however was the relationship between the velocity of the robot [m/s] and the applied PWM signal.

The navstack provides velocity commands in [m/s] which need to be converted to PWM values.

Challenges:

- Variations depending on the load.
- Variations depending on the ground surface.
- Differences between the two motors.

Physical parameter determination (2)

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

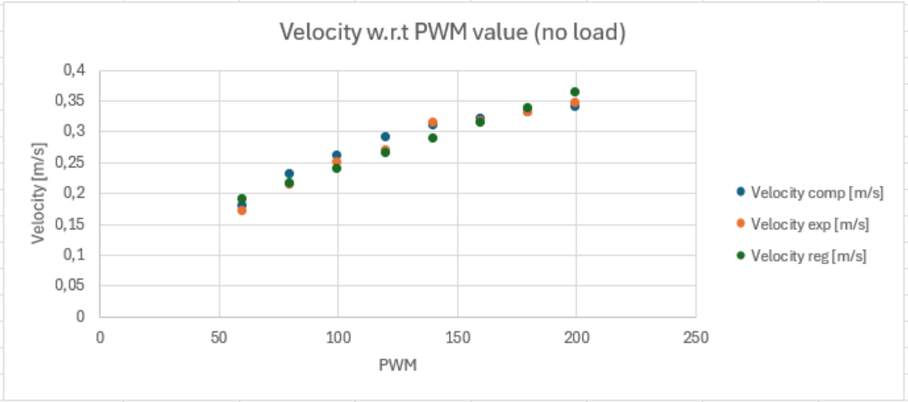
Least-square curve fitting (only mobile base / no load)

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 curve fitting approach

03

Lifting Mechanism

Software development for the lifting mechanism

Objectives of this year:

- Create a ROS package to easily operate the platform
- Calibrate and use a stereo camera for depth analysis
- Improve the mechanical grip system

Objectives realized this year:

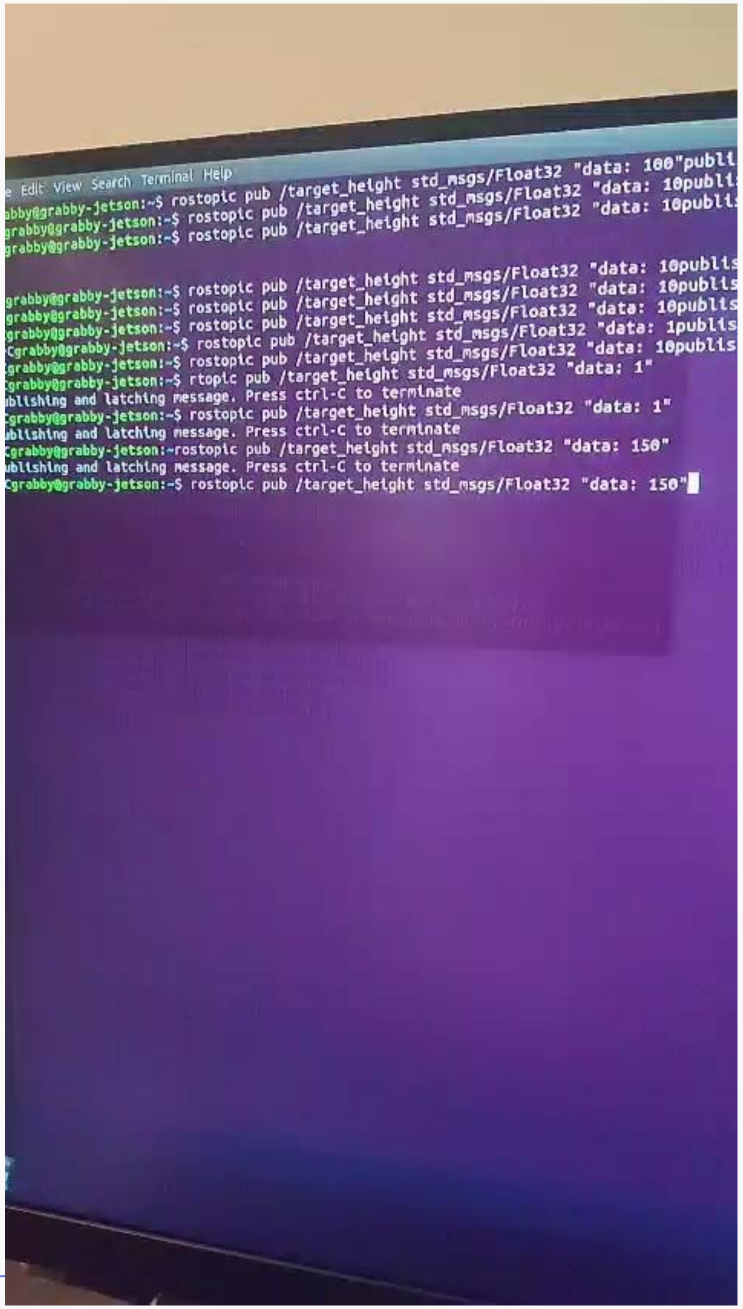
- ROS package creation
- Problem solving
- More problem solving
- Even more problem solving

04

Demonstration

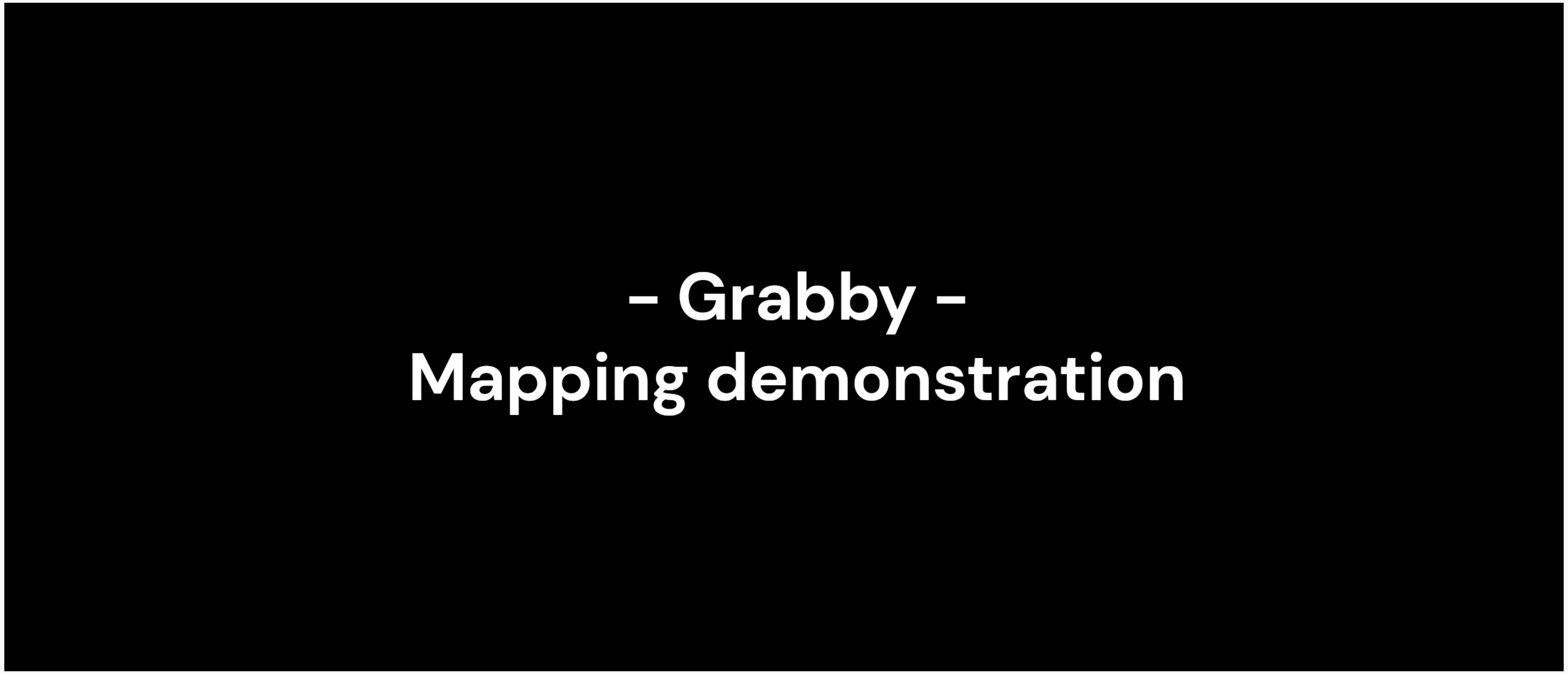
Lifting mechanism

Lifting Mechanism demonstration



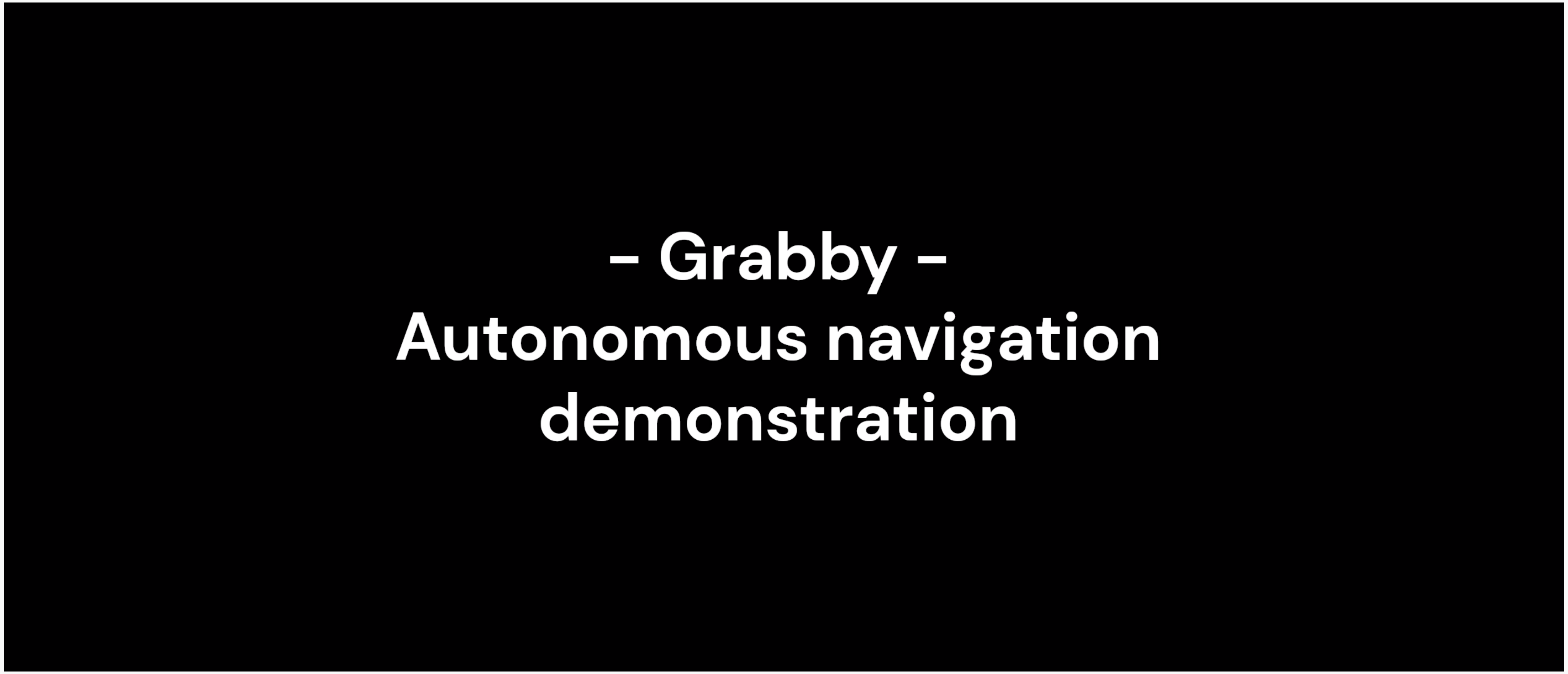
```
                                 rostopic pub /target_height std_msgs/Float32 "data: 100"publi
abby@grabby-jetson:~$ rostopic pub /target_height std_msgs/Float32 "data: 10publi
grabby@grabby-jetson:~$ rostopic pub /target_height std_msgs/Float32 "data: 10publi

                                  rostopic pub /target_height std_msgs/Float32 "data: 10publis
grabby@grabby-jetson:~$ rostopic pub /target_height std_msgs/Float32 "data: 10publis
grabby@grabby-jetson:~$ rostopic pub /target_height std_msgs/Float32 "data: 10publis
grabby@grabby-jetson:~$ rostopic pub /target_height std_msgs/Float32 "data: 1publis
Cgrabby@grabby-jetson:~$ rostopic pub /target_height std_msgs/Float32 "data: 10publis
grabby@grabby-jetson:~$ rostopic pub /target_height std_msgs/Float32 "data: 1"
grabby@grabby-jetson:~$ rtopic pub /target_height std_msgs/Float32 "data: 1"
blishing and latching message. Press ctrl-C to terminate
Cgrabby@grabby-jetson:~$ rostopic pub /target_height std_msgs/Float32 "data: 1"
blishing and latching message. Press ctrl-C to terminate
Cgrabby@grabby-jetson:~rostopic pub /target_height std_msgs/Float32 "data: 150"
blishing and latching message. Press ctrl-C to terminate
Cgrabby@grabby-jetson:~$ rostopic pub /target_height std_msgs/Float32 "data: 150"
```

Mobile Base (Mapping)


– Grabby –
Mapping demonstration

Video demonstration of the mapping process using Google Cartographer and a custom-made teleoperation ROS package

Mobile Base (Navigation)



– Grabby –
Autonomous navigation demonstration

Video demonstration of the navigation

05

Future

Future improvements and changes

Changes:
- Architecture switch to RaspberryPi 5 to enable better software support (ROS2 etc.)
- Creation of custom PCBs for easier management and implementation of components.
- Adding modularity to the system.
- Making it more robust.

Improvements:
- Installation of a 9-DoF IMU for better navigation.
- Implementation of a battery and redesign of the power system.
- Strengthening of the frame.
- Improvements to the positioning system.
- Improvements to the drive system.

Thank you !