YouBot Project

Redeployment and quantitative assessment of the KUKA YouBot

Author: Aditya Agrawal

Advisor: assistant professor Tomasz Kucner

Supervisor: senior university lecturer Salu Ylirisku

ELEC-C0302 Final Project in Digital Systems and Design
Aalto University

June 18, 2025

Contents

1 Introduction

1.1 Motivation

In the rapidly evolving field of robotics, expensive hardware platforms often become obsolete quickly. This is not purely because of physical wear and tear or a lack of functionality, but also due to quick advancements in hardware, software and infrastructure. For example, artificial intelligence (AI) and related fields like Machine Learning (ML) and Deep Learning (DL) has accelerated progress within robotics, driving demand for specialised, high-performance hardware and tightly integrated software ecosystems. Consequently,older robotic platforms, despite their inherent capabilities, become less relevant and lose extended support. This furthermore could lead to increased e-waste and reduced sustainability in the robotics industry [1].

The KUKA YouBot is a prime example of this phenomenon. It was once a popular platform for research and education in robotics in the mid-2010s, but has been since discontinued and is no longer supported by the manufacturer. This mobile robot base is quite robust, compact and utilizes open-source drivers. While this platform has reduced performance and integration capabilities compared to its modern peers, it can still serve a variety of purposes for the purposes of industrial and academic research. This includes autonomous navigation, perception and human-robot interaction. Instead of discarding such platforms, we should explore avenues to extend their lifespan and utility. This would reduce e-waste and costs associated with acquiring new hardware, and thus lower the barrier to entry for research and development in robotics.

This project aims to explore and demonstrate the potential of legacy robotic hardware like the KUKA YouBot within the modern robotics landscape. By repurposing such platforms in the context of its technical capabilities and applications, we can showcase the long-term viability of older platforms whilst allowing for smooth integration with other systems.

1.2 Project Overview

This report documents the progress of redeploying the KUKA YouBot mobile base, and quantitatively assessing its capabilities in terms of navigation and odometry.

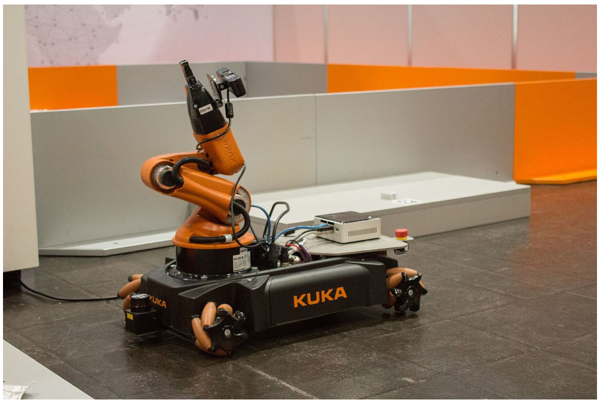
Section 2 details the initial state of the YouBot, including its hardware and software components, and describes the initial objectives of this project. Section 3 presents the revised technical and learning objectives of the project. Section 4 describes the system development process, covering initial configuration, battery and sensor integration, and custom script development. Section 5 explains the experimental setup for evaluating movement and odometry performance. Section 6 discusses the future potential of the YouBot for long-term research and industrial use cases, while section 7 concludes the report with a personal reflection on the project and its outcomes. The appendix includes a user manual for the YouBot, as well as relevant code files and experimental data.

2 Initial state of YouBot

The YouBot is a mobile robotic platform developed by German automation company KUKA [2]. First introduced in the early 2010s, it was primarily designed for research and educational purposes in the field of mobile robotics [3]. To further this purpose, a significant portion of software used on the YouBot is open-source and available on GitHub [3], [4].

The YouBot typically consists of two main parts: a mobile base and a robotic arm. The mobile base is equipped with four omnidirectional mecanum wheels and motors for movement, alongside an onboard computer for processing and control. This onboard computer runs Ubuntu and ROS1, with conveniently provided drivers and wrappers, allowing for smooth software integration. The robotic arm has 5 degrees of freedom (DOF) and a two-finger gripper [3], enabling it to perform a variety of tasks through the onboard computer. Since this project involves only the mobile base, we will not be discussing the arm in detail.

The YouBot's open-source software stack and ROS compatibility provide a versatile foundation for both low-level hardware interfacing and high-level algorithm development. This robot and the attached sensor modules are thus particularly well-suited for research within mobile robotics, particularly those pertaining to navigation, perception and human-robot interaction.



(a) The robot at RoboCup 2016 Leipzig [5].

(b) The robot as used in the Aalto University Robotics Lab.

Figure 1. The YouBot.

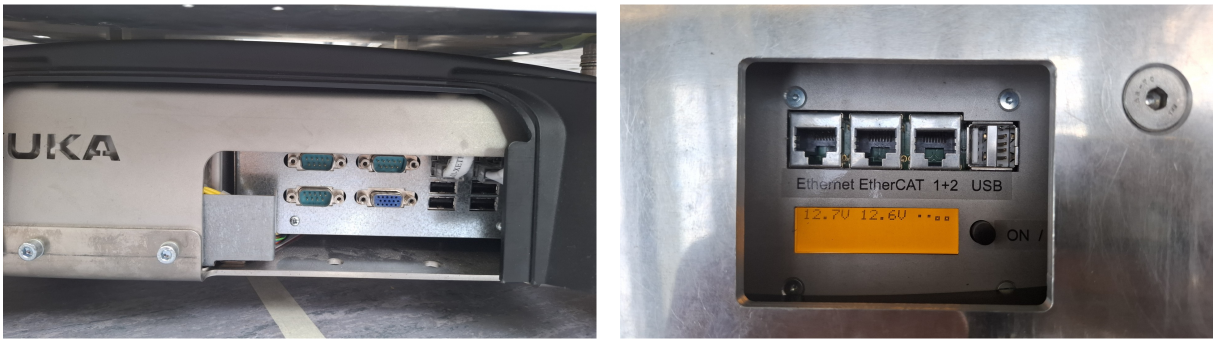
2.1 Onboard Computer

The onboard computer features a Intel Atom D510 @ 1.66GHz processor, with a 2GB DDR2 RAM and 32GB SSD hard drive.

The computer currently runs Ubuntu 12.04.5 LTS with ROS1 Hydro, which is a decade-old version of the operating system and the robot operating system. Given the age of the OS and ROS, compatibility with modern libraries and software is limited. Essential drivers and wrappers were already installed to enable communication with the robot's motors and sensors. These could be accessed directly through C++ programs or through ROS packages. For the purposes of this project, we have elected to focus on the latter to allow for a seamless integration with the ROS ecosystem in the future, alongside a level of standardization and ease of use.

2.2 Connection Interfaces

To interface with the onboard PC, the YouBot features panels with several connection ports. The top panel of the robot contains two EtherCAT ports for consistent real-time communication with motion-oriented systems (i.e. robotic manipulators), and a standard Ethernet port for wired interfacing with an external computer or network. Adjacent to these communication ports are a power button and a small screen that displays input voltage and ON/OFF state of the onboard PC and motors (Figure 2b). Furthermore, the right side of the robot features a panel with a VGA port for video output, as well as six USB 2.0 ports for connecting peripherals such as a keyboard, mouse, or wireless adapter (Figure 2a).



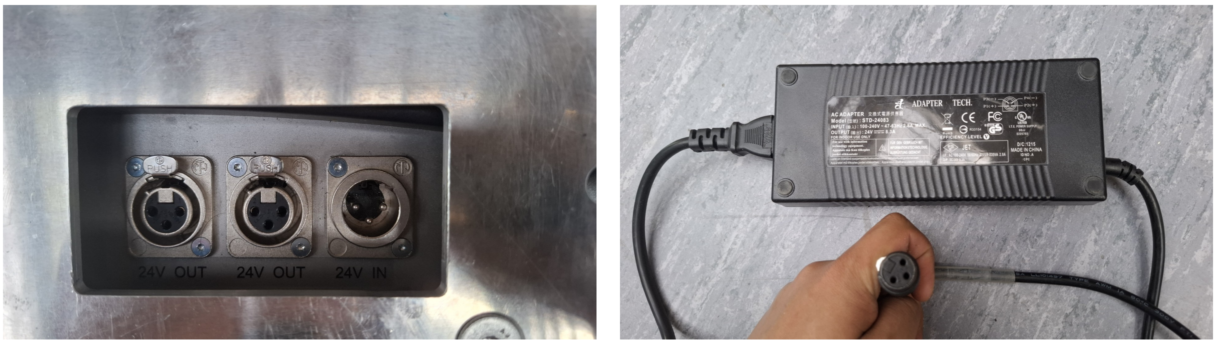
(a) The right side of the robot, showing the VGA and USB ports.

(b) The onboard computer screen, alongside the power button and EtherCAT/Ethernet ports.

Figure 2. Connection points on the YouBot.

2.3 Power

The YouBot is powered by a 24V power supply, which can be connected through a 3-pin XLR connector located on the top panel of the robot. Additionally, the base includes two 24V 3-pin XLR output ports, which are intended for powering external components such as robotic manipulators or sensors (Figure 3a).



(a) The power input and output ports.

(b) The 24V power adapter used to power the YouBot.

Figure 3. Direct power supply components for the YouBot.

However, continuously powering the YouBot through a wall connection is not ideal for the goal of mobile robotics. Thus, the YouBot was also originally equipped with a maintainence-free sealed lead-acid (SLA) battery (Figure 4a) . This battery had a capacity of 5Ah and provided an approximate runtime of 90 minutes [6]. This battery furthermore connects to the robot via a 4-pin XLR connector, and is located in a dedicated slot on the left side of the robot (Figure 4b).

(a) The original SLA battery.

(b) The left side of the robot, showing the battery in its holder.

Figure 4. The YouBot's SLA Battery.

Upon receiving the robot, four SLA batteries were available: three of them were the original units supplied with the YouBot, and the fourth was a makeshift replacement assembled by the lab engineer, Vesa Korhonen, in 2019. Unfortunately, none of these batteries were functional. The original units had degraded beyond usability, and the makeshift battery had similarly deteriorated over time.

As a result, the robot could not operate on battery power in its current state and required a constant wired connection to a wall outlet—an obvious limitation for a mobile robotics platform. A battery replacement would be ideal to restore the robot's potential mobility.

2.4 Sensors

Alongside the robot base and various hardware components, the YouBot was also equipped with a variety of sensors to enhance perception and navigation. These included a Kinect v1 camera and two Hokuyo URG-04LX laser rangefinders.

2.4.1 Kinect

The Kinect v1 camera is a depth and RGB camera that was originally designed and sold in tandem with the XBOX 360 to support motion tracking and gaming. Due to its low cost, high availability and ease of use, it had indirectly become a popular choice in the robotics community for perception tasks in the 2010s.

Access to the Kinect's sensor data can be achieved through the use of open-source software such as libfreenect [7]. The retrieved data can then be processed using computer vision libraries such as OpenCV to enable the autonomous navigation as previously described.

Figure 5. The Kinect v1 camera module.

2.4.2 Hokuyo URG-04LX Laser Rangefinder

The Hokuyo URG-04LX is a lightweight 2D laser rangefinder that is commonly used in robotics research. It provides distance measurements in a 240-degree field of view and can detect objects upto 5.6m meters away with an accuracy of ±30 mm and angular resolution of 0.352deg [8]. It communicates with the onboard computer using a serial interface, enabling straightforward integration into existing systems for mapping, localization, and obstacle avoidance tasks.

Figure 6. The Hokuyo URG-04LX laser rangefinder.

2.5 Additional Hardware Components

The YouBot also came with a variety of additional components and accessories to assist with sensor mounting and operation. These included a variety of nuts, bolts, and screws, as well as a hex key and several horizonal and vertical pillars. These components can be combined in a variety of ways to create a stable and secure mounting platform for the sensors.

A key component provided was a pre-attached sensor and mounting plate on the top panel (see Figure 1b), designed to also assist in the convenient attachment of various sensors and accessories [9]. A wireless adapter was also included, which can be used to connect to the internet without the need of Ethernet. Additionally, a separate box contained various miscellaneous items, including previously used batteries, panels for the Youbot and controllers; which could prove potentially useful for customization or repair.

(a) Various pillars, nuts and bolts amongst other components.

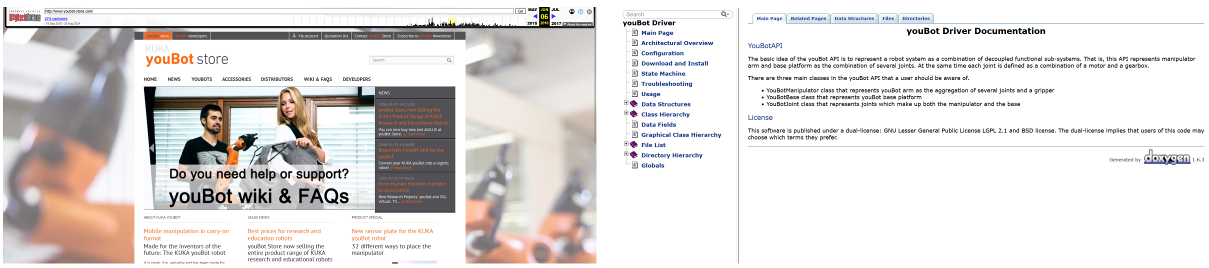
(b) The box containing previously used batteries and miscellaneous components.

Figure 7. Additional hardware components supplemented with the YouBot.

2.6 Documentation

The documentation for the KUKA YouBot is sparse. Since the product's discontinuation in the mid-2010s, KUKA has removed most references to the product from their official website, and the official website for the YouBot (`youbot-store.com`) is no longer active. The only starting point for official documentation was through a user manual dated 2013 found on the onboard PC [3].

Fortunately, as the YouBot was marketed as an open-source platform, many of its drivers and software components remain available on the YouBot GitHub repositories [4]. Furthermore, we leveraged the Internet Archive's Wayback Machine [10] to retrieve snapshots of the original website. While this archive is incomplete and tedious to comb through, they provide valuable insights into the hardware specifications, software stack, and applications of the YouBot during its active develpoment period. Furthermore, the youbot official GitHub repository [4] contains all original packages and software components. Driver documentation is additionally available on a dedicated GitHub page [11].



(a) The YouBot Store page on the Internet Archive's Wayback Machine.

(b) The YouBot driver documentation on GitHub Pages.

Figure 8. Websites for YouBot documentation.

2.7 Initial Project Goals

Considering the hardware and software components of the YouBot, the initial goals of this project involved a comprehensive revival of the robot and the enabling of autonomous navigation using the Kinect camera. This would showcase the potential of the YouBot in cutting-edge research despite its age.

The Teach & Repeat (T&R) method [12] was considered to be an ideal candidate to allow for autonomous navigation. This is a two-phase robotic navigation method where a robot is first "taught" a path via human guidance or pre-recoded data. The sensor data captured during this phase can then be used to allow the robot to autonomously "repeat" the path later on, even in different environments.

T&R only requires a single camera for a basic implementation. It is a relatively simple method to implement through the use of open-source software and libraries, such as OpenCV and ROS. Furthermore, it is robust to changes in the environment and can dynamically correct errors through the use of visual odometry. As such, it has been a very popular research topic within the field of robotics, particularly in the context of mobile robots [13] [14]. Much of the work done on extending T&R aims to improve this robustness and scalability for a variety of sensors and use-cases.

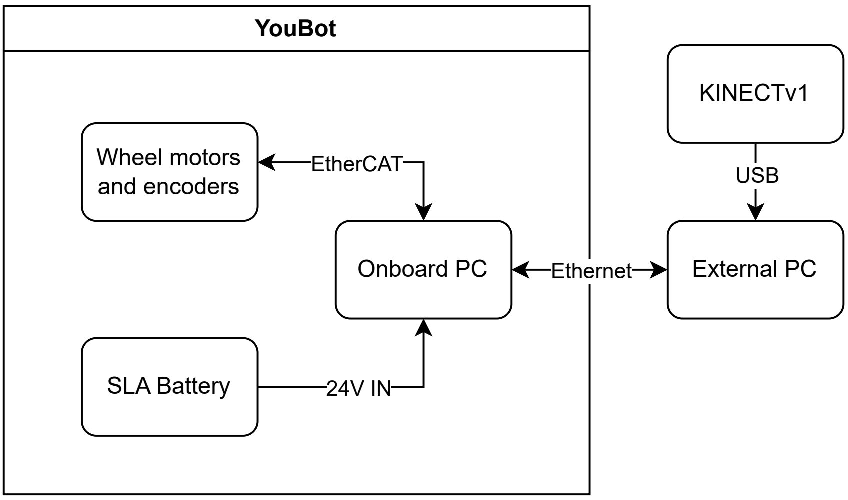


Figure 9. System Diagram

Figure 9 shows the proposed system architecture for this implementation. This would involve the onboard PC running ROS 1 Hydro, which would handle the low-level control tasks regarding motor actuation. The navigation and perception tasks would be handled by a more capable external computer running ROS 2 Jazzy. The two systems would then communicate through a ROS1-ROS2 bridge, enabling compatibility between legacy software on the YouBot and the more modern ROS2 stack. The Kinect camera would be then mounted on top of the YouBot, and would be connected to the external computer via USB. This setup allows for the YouBot to be used without much modification or strain to its legacy hardware, while still enabling the use of modern software and sensors.

However, due to several challenges—namely the age of the YouBot, limited prior experience with robotics, and software incompatibilities—this plan proved infeasible. A revised set of project goals was subsequently developed, as discussed in the following section.

3 Project Overview

3.1 Technical Goals

The technical goals for this project involve the deployment and quantification of the robot's capabilities, particularly in terms of navigation and odometry. This would involve the following goals:

- Inspecting the initial physical state of the YouBot
- Enabling the robot start up
- Enabling the running of the original demos
- Enabling the running of the ROS interface
- Enabling the ROS1-ROS2 bridge
- Enabling wireless connections to external computers using SSH and Ethernet
- Writing custom programs to control the YouBot
- Testing the movement and odometry of the YouBot
- Measuring the quality of movement and odometry by measuring their error
- Replacing the current deprecated batteries with a usable battery
- Documenting the revival process and potential future applications of the YouBot

The set of goals that have been formulated have deviated from the original goals, which were formulated based on experimenting and identifying the limitations of the robot.

3.2 Learning Objectives

In parallel with technical outcomes, this project required gaining a range of practical and theoretical competencies in robotics. These learning objectives included:

- Gaining hands-on experience restoring and operating a legacy robotic platform (KUKA YouBot)
- Building proficiency in Linux environments relevant to robotics development
- Writing Bash scripts to automate system setup, compilation, and deployment workflows
- Understanding the architecture and functionality of both ROS1 and ROS2, including their integration
- Establishing and managing secure connections between host and robot using SSH and Ethernet
- Creating and deploying custom ROS nodes for low-level motion control
- Studying robot navigation and odometry principles, including error sources and correction strategies
- Developing skills for documenting restoration efforts and formulating directions for future work

3.3 System Architecture

Our current system architecture is starkly different from the initial goal. While the YouBot itself remains unchanged, the external computer only serves the purpose of SSH access and remote control into the onboard computer. SSH could be conveniently done through a wireless connection, or through a more reliable Ethernet connection. The onboard computer runs ROS1 Hydro and communicates with the wheel motors and encoders through EtherCAT. No external sensors will be used, highlighting our focus on enabling the YouBot's existing capabilities rather than extending them.

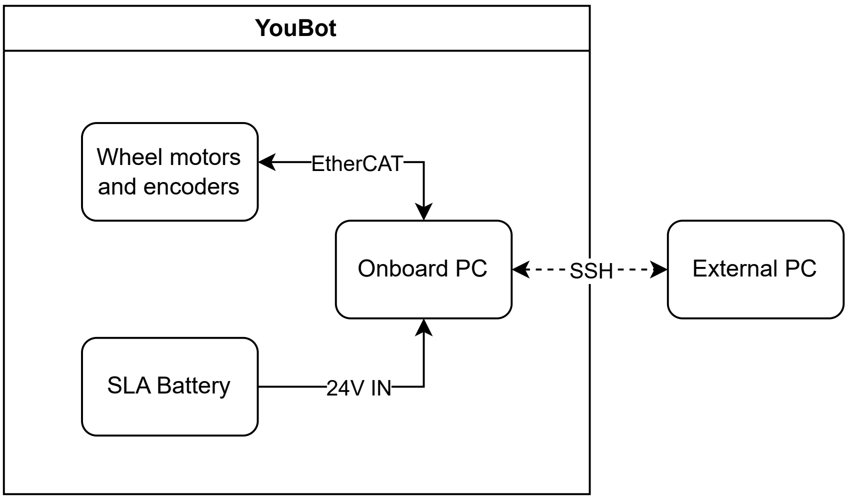


Figure 10. New System Diagram

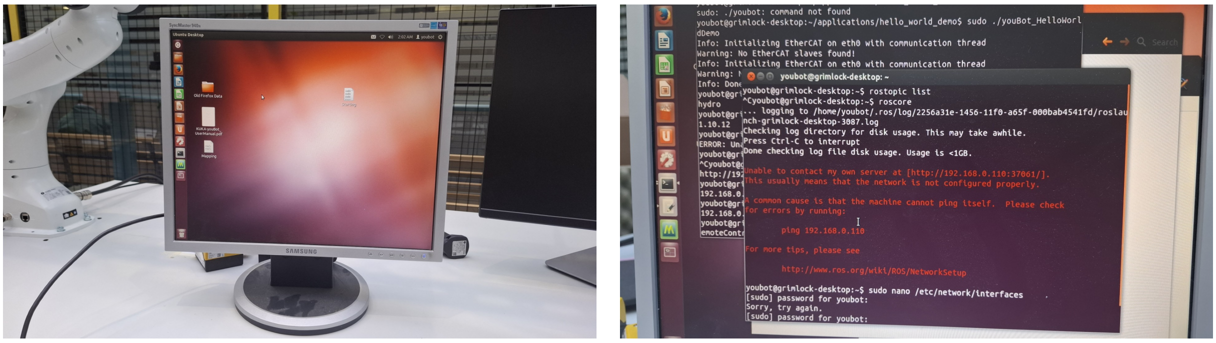
4 System Development

4.1 Booting up the YouBot

After conducting the initial inspection as described in section 2, the next step was to boot up the YouBot. Since the onboard battery was completely dead, the YouBot was connected to an external 24V supply. On long pressing the power button, the screen flashed on and I could see the voltage input for the robot, alongside options to turn the PC and motors on/off seperately.

Using a VGA monitor and USB peripherals, the onboard PC was accessed. It booted successfully and functioned like a standard Linux desktop. The wireless adapter was also operational and could connect to the internet. However, remote access via Ethernet was initially unavailable due to the absence of a static IP configuration.

Upon further inspection, several YouBot-specific software packages were also found installed on the onboard computer. These packages include the `youbot_driver`, which provides the necessary drivers to communicate with the roboto's motors through EtherCAT, and the `youbot_ros_pkg`, which provides a complete ROS stack for the YouBot. These packages are essential for enabling the YouBot to communicate with ROS and to control its motors and sensors. Alternatively, a simpler ROS interface can be used through the `youbot_driver_ros_interface` package, which provides a minimal set of ROS nodes to control the YouBot. This package has been used for the purposes of this project to simplify the interaction with the robot. Additionally, the `youbot_description` package provides the necessary URDF files to describe the robot's kinematics and dynamics, allowing for simulation and visualization in RViz and Gazebo. Finally, the `youbot_applications` package provides a set of example applications and demos to get started with the YouBot.



(a) The onboard computer booting up.

(b) The initial attempt at booting up the ROS interface.

Figure 11. Initial attempts at booting up the YouBot.

The ROS interface as well as the relevant demo programs were attempted to be run upon startup without prior configuration. The C++ demo programs were unable to detect any devices on the eth0 port, which was not the port the motors were using. The ROS interface was also unable to ping itself and start `roscore`. This was due to the onboard computer not being binded to the correct static IP address, which is necessary for the ROS interface to function properly.

Furthermore, the onboard computer did not have a static IP address, which meant that it could not be accessed through Ethernet. This would prevent Ethernet-based communication with the YouBot, which would be more reliable than wireless communication.

4.2 Configuration & C++ demos

To boot up the C++ programs, the onboard computer required some configuration to suit my current needs. The YouBot config files had to be changed to provide the correct interface (`eth1` instead of `eth0` for the motors). Refer to A.2 for relevant details.

The Ethernet connection was also configured to use a static IP address, which would allow for reliable communication with the onboard computer. This was done by editing the `interfaces` file in the `/etc/network/` directory and adding a static IP address for the `eth0` interface. Refer to A.2.2 for the relevant details.

After making these changes, the onboard computer was able to communicate with my external computer through Ethernet. Furthermore, I was able to run the C++ demo programs. Refer to A.3 for a guide on doing the same. Running the keyboard and hello world demos was particularly successful, and showcased the YouBot's robust movement capabilities despite half a decade of inactivity. While this movement has degraded over time, the YouBot was still able to move in all directions and turn in place. The YouBot was also able to detect the motors and encoders, and could report basic issues through the C++ error code.

4.3 ROS Interface

While the C++ demos worked quite nicely, they would be unsuitable for potential extension into actual use-cases in mobile robotics. There is no in-built system for networking across multiple machines, utilising multiple sensors, debugging, status echoing, etc. Thus, ROS was chosen to be the basis for my experimental setups. This included changing the `.bashrc` file to provide ROS with the correct environment variables, which would in turn allow it to communicate with itself. Refer to **??** for the relevant details.

I was able to start up `roscore` after making this change. However, I was not able to run the YouBot-ROS interface due to a lack of permissions to access the `eth1` interface. This was due to the fact that the YouBot-ROS interface requires root privileges to access the EtherCAT devices. To resolve this, I had to run the ROS interface in with `sudo bash -c`, which allowed it to access the EtherCAT devices and communicate with the motors.

After fixing this issue, I elected to try and run a keyboard teleoperation demo program to test the YouBot's movement capabilities through ROS. This file was stored locally at `~/ros_stacks/youbot-ros-pkg/youbot_drivers/youbot_oodl` and could be run by typing:

```
$ rosrun youbot_oodl youbot_keyboard_teleop.py
```

I was able to fulfill one of the key project goals, of enabling the YouBot to move through the ROS interface. While there were some issues regarding overcurrent that showed up on some wheels with high speeds, the YouBot was able to move in all directions and turn in place.

4.4 ROS1-ROS2 Bridge

One of the first tasks I decided to embark on after getting the robot up and running was to set up the ROS1-ROS2 bridge. This was the most important link within the system, as it would allow the robot to seamless access the ROS2 navigation stack and enable it to access the computational resources of a more recent computer. However, the version of ROS1 on the robot is Hydro, which is more than a decade old and predates ROS2 development, i.e. it is not compatible with a ROS1-ROS2 bridge.

There are thus a handful of options to consider: One would be to upgrade the ROS1 version of the robot to a version that supports the bridge, such as Melodic or Noetic. However, this would require a significant amount of work and may cause compatibility issues with the outdated software on the robot. Another option would be to use a different computer with a more recent version of ROS1 and use it as a middleware between the robot and the ROS2 navigation stack. This would allow for a more seamless integration of the two systems, but would require additional hardware and software setup and create an unneccesarily complex system.

4.5 Custom Programs

I elected to write my own custom programs to control the YouBot, as this would allow me to gain a better understanding of the ROS interface and how to interact with the robot. I started by writing a simple program that would move the YouBot in a square pattern, using the `geometry_msgs/Twist` message type to control the robot's movement. This message type is used to specify the linear and angular velocities of the robot, which can be used to control its movement. This program is very basic in implementation since it calculates the time to move by dividing the speed by distance. However, this already showcased many interesting issues that occured with the YouBot. Firstly, some wheels would often show an overcurrent error when moving at high speeds, which indicates that the motors were drawing too much current and could potentially damage the motors or the robot's electronics. This was particularly noticeable when moving in a straight line at high speeds, where the robot would often drift slightly off course, potentially due to the motors not being perfectly calibrated with each other.

4.6 Sensors

4.6.1 Kinect Camera and libfreenect

The Kinectv1 camera is a depth and RGB camera that was originall designed and sold in tandem with the XBOX 360 to enable motion tracking and gaming. Due to its low cost, high availability and ease of use, it has indirectly become a popular choice for robotics research as well.

We can use open-source software such as libfreenect to access the data from the various sensors on the module.

discuss the libfreenect library, how to install it, and how it could be used within ROS.

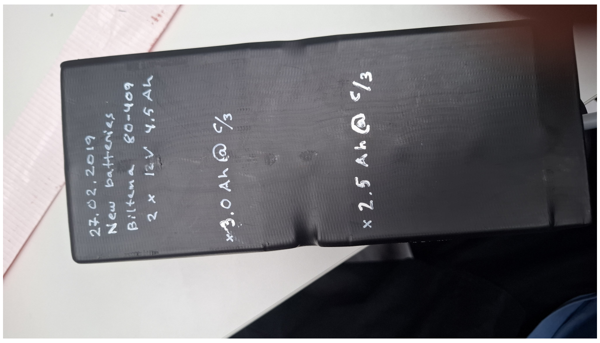
4.6.2 Hokuyo URG-04LX Laser Rangefinder

The Hokuyo URG-04LX is a 2D laser rangefinder that is commonly used in robotics research. It is a compact and lightweight sensor that provides high-resolution distance measurements in a 240-degree field of view. The URG-04LX is capable of measuring distances up to 4 meters with an accuracy of +/- 10 mm. It communicates with the onboard computer using a serial interface, making it easy to integrate into existing systems.

discuss installing the drivers and whatnot, and how to use it within ROS and rqt.

4.7 Battery Replacement

As previously mentioned, the YouBot came with three 24V SLA batteries. Two of these were the original batteries included with the YouBot, and one was a makeshift battery put together by Vesa in 2019. All three batteries were unusable. This section thus documents the technology behind the SLA batteries, the testing of the original batteries, replacing and testing a new battery, and future battery options.



(a) The original YouBot battery.

(b) The handmade battery, case removed.

4.7.1 SLA Battery Overview

Sealed lead-acid (SLA) batteries operate based on a reversible chemical reaction between lead plates, lead dioxide and sulfuric acid electrolyte [15]. When the battery discharges, the substances react to form lead sultfate and water, releasing electrical energy. During charging, this reaction is reversed.

These batteries are quite simple, robust, inexpensive and safe to use, indicating the reason for use in this scenario. However, they require some maintainence, and should undergo full discharge and charge cycles regularly to keep them in a good state. If they are not used regularly, they can suffer from sulphation, where lead sulfate crystals gradually form on the plates of the battery. This process is irreversible and permanently impairs the battery's capabilities. Cheaper SLA batteries are more prone to this issue due to lower quality materials, and one may only expect a maximum lifespan of 3-5 years from them.

SLA batteries are furthermore sealed and contain one-way valves to prevent internal pressure buildup due to production of hydrogen and oxygen gas. Normally these gases recombine back into water, but overcharging can force gas release, leading in gradual water loss.

The recommended voltage for charging SLA batteries is 2.3 volts per cell (2V), or 13.8V for a 6-cell (12V) battery, or 27.6V for a 12-cell (24V) battery [16]. Charging at a lower voltage (i.e. 2V per cell) will not fill up the battery completely, and increase the risk of sulphation, since the lead sulfate crystals will not be fully converted back into active materials [17].

4.7.2 Testing original batteries

To confirm the degradation of the original batteries, we decided to test them using a multimeter. The two original batteries were completely dead, and did not show any voltage when connected to the multimeter. The makeshift battery showed some voltage, but it was far too low to be usable out the gate. This battery was put together using two 12V SLA batteries, where each battery was individually connected to the 4-pin XLR connector. The YouBot internally connects them in series to create a 24V battery.

Vesa attempted to revive this battery through desulphation, where a higher-than-recommended voltage (in this case, 30V) is applied to the battery to force current through the hardened sulphate layers and dissolve them.

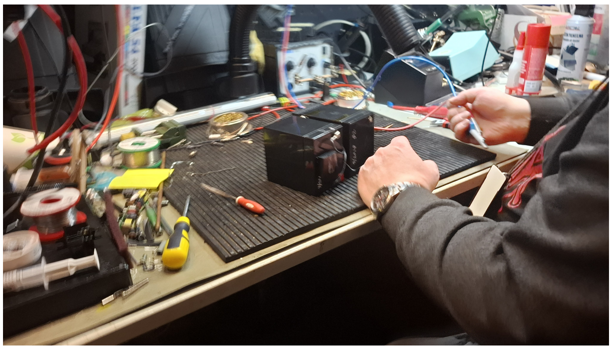


Figure 13. Testing the old batteries with Vesa.

This seemed initially promising: the battery was accepting a charge and its voltage was increasing. However, this was only temporary, as the battery quickly lost voltage again after charging, suggesting that the sulphation was very severe. While some surface conductivity was perhaps restored, the effective area of the electrode plates was still very small, resulting in a very low capacity. As such, the battery was not usable for our purposes.

4.7.3 Replacing batteries

We decided to remake the makeshift battery using two new 12V SLA batteries of the same dimensions. While the previous batteries were Bitelma batteries, we bought some from Leader this time. These batteries were 12V 5.4Ah batteries. We tested them using a multimeter and used a car charger to charge them overnight.

The old batteries were removed from their casing and all relevant connections and pieces to structure the battery were removed. The new batteries were then appropriately connected to the 4-pin XLR connector, and the structural pieces were reattached with tape. After putting the casing back on, we were able to connect the new battery to the YouBot and power it on. The YouBot detects the two batteries and shows their individual voltages, confirming that the battery works. While it's not hermetically sealed, it works well for the purposes of this project.

Figure 14. The new batteries taped together in the needed form factor.

4.7.4 Future Battery Options

While we have replaced the SLA batteries with a makeshift one at a rather inexpensive cost, these batteries will not last long and will be prone to the same issues as the original batteries.

An ideal candidate for a replacement battery techonology would be lithium-iron-phosphate (LiFePO4) batteries. These batteries are more expensive, but have multiple advantages over SLA batteries [18]. They have longer lifespans, higher energy density, lighter weight, and are less prone to degradation. However, they would also need an integrated battery management system (BMS) to ensure safe operation, and potentially some custom hardware to fit the dimensions of the YouBot's battery compartment.

5 Experimental Setup for Movement and Odometry

There were various inconsistencies that occured during the running of custom programs. For example, the square movement program showed that the odometry data, measured data and expected data disagreed with each other. To formally quantify these irregularities, a controlled set of experiments was conducted to assess error rates between odometry measurement and actual movement.

5.1 Motivation and Approach

The experimental effort was designed to:

- Quantify the error between odometry measurement and actual movement.
- Identify whether errors were consistent (systematic), random, and/or scale-dependent.
- Assess the reliability of onboard odometry over short and long trajectories.
- Test how speed and distance affected performance.
- Provide insights into feedback control and calibration strategies for error reduction.

5.2 Experiment Design

A total of 13 experiments were conducted, split into:

- 5 linear movement tests at different speeds and distances.
- 8 rotational movement tests at different speeds and angles.

Each experiment was carried out 20 times to ensure statistical significance and to allow observation of both deterministic and stochastic error components. The below tables summarize the configurations for linear and rotational motion tests, respectively.

Test ID	**Speed (m/s)**	**Distance (m)**
L1	0.2	1.0
L2	0.4	1.0
L3	0.2	2.0
L4	0.4	2.0
L5	0.6	2.0

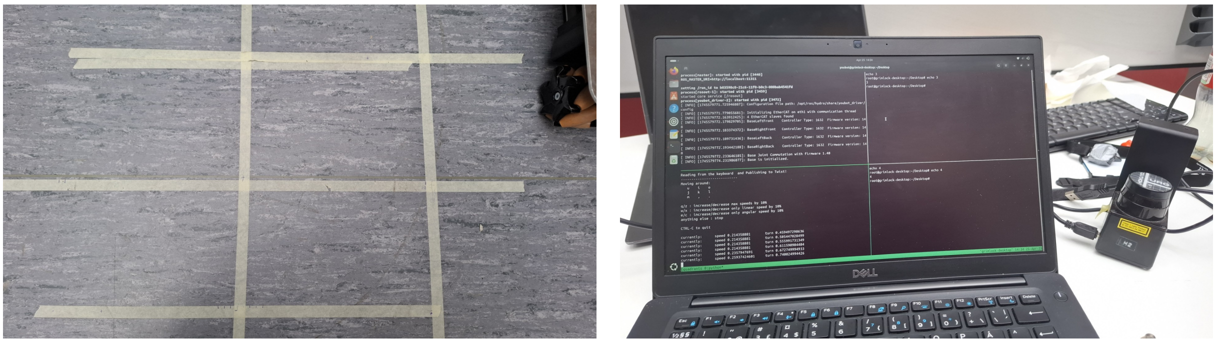
Table 1. Linear motion test configurations

Test ID	**Speed (deg/s)**	**Target Angle**
R1	45	45°
R2	75	45°
R3	45	90°
R4	75	90°
R5	45	180°
R6	75	180°
R7	45	360°
R8	75	360°

Table 2. Rotational motion test configurations

5.2.1 Experimental Procedure

The experiments were conducted in a controlled environment on a flat surface. Masking tape was used to make the start and end points of each trajectory, ensuring consistent measurements. The robot then was commanded to move at a specified speed for a calculated duration of time to cover the target distance or angle, using custom programs (see Appendix B.2).



(a) The masking tape with marked start and end points.

(b) The tmux setup used. The two right panes were used to run the custom programs, while the left pane was used to run the ROS interface and keyboard program for ease of adjustment.

Figure 15. The experimental setup.

The deviation of the final position from the expected position was measured using measuring tape to the nearest centimeter and recorded in a CSV file, alongside the odometry deltas the custom program outputted. Note that for the linear movement tests, the robot center's x and y drift were directly recorded without any additional recordings for the angle deviation. Whereas for the rotational movement tests, the x and y drift of two points of the robot (specifically the edge of the middle of the front and back panels) were recorded, and using this data the center of the robot's deviation as well as angle deviation were calculated (see Appendix C for more details). The measured data was then subtracted from the odometry data to calculate the error in movement. While the expected values play a role in undertaking the experiment itself, we did not use them in the error calculations. The goal was to measure the deviation of the robot's movement from its own odometry, rather than from a fixed reference point.

The data was then processed and plotted using Matplotlib to visualize the errors and variances in the robot's movement (see Appendix B.2).

5.3 Results and Error Analysis

5.3.1 Linear Motion Error Characteristics

The linear motion experiments revealed a consistent underestimation in forward movement (dx), with mean errors increasing approximately linearly with the commanded distance. For instance, at 0.2 m/s over 1 meter, the mean dx error was -4.55 cm, while at 2 meters it was -9.05 cm, indicating a scale-dependent and systematic error. This bias suggests a miscalibration in either velocity estimation or time-based control. Similar proportional errors occur for 0.4 m/s. As speed increases, the robot tends to be more inconsistent with its goal and overshoots more on average, showing some speed-dependent behaviour as well.

By contrast, the y-axis errors were mostly small and random, with no clear systematic bias. An exception was at 0.6 m/s over 2 meters, where the mean dy error was -20.83 cm. The CSV files reveal that this was due to the robot odometry dy measurements increasing in value as experiments were carried out, while the measured distance remained relatively constant. This suggests that the robot's odometry is unable to accurately track its position at high speeds, potentially due to wheel slip or encoder inaccuracies.

Speed (m/s)	**Distance (m)**	**dx (cm)**		**dy (cm)**	
		Error	**Var**	**Error**	**Var**
0.2	1	-4.5500	0.2475	0.5000	1.4500
0.2	2	-9.0500	0.1475	-0.4500	2.9475
0.4	1	-4.9524	0.9025	0.8095	2.5351
0.4	2	-9.7000	0.8100	-1.4000	10.0400
0.6	2	-13.3043	3.0813	-20.8261	203.5350

Table 3. Linear motion statistics: Mean error and variance in dx/dy (cm)

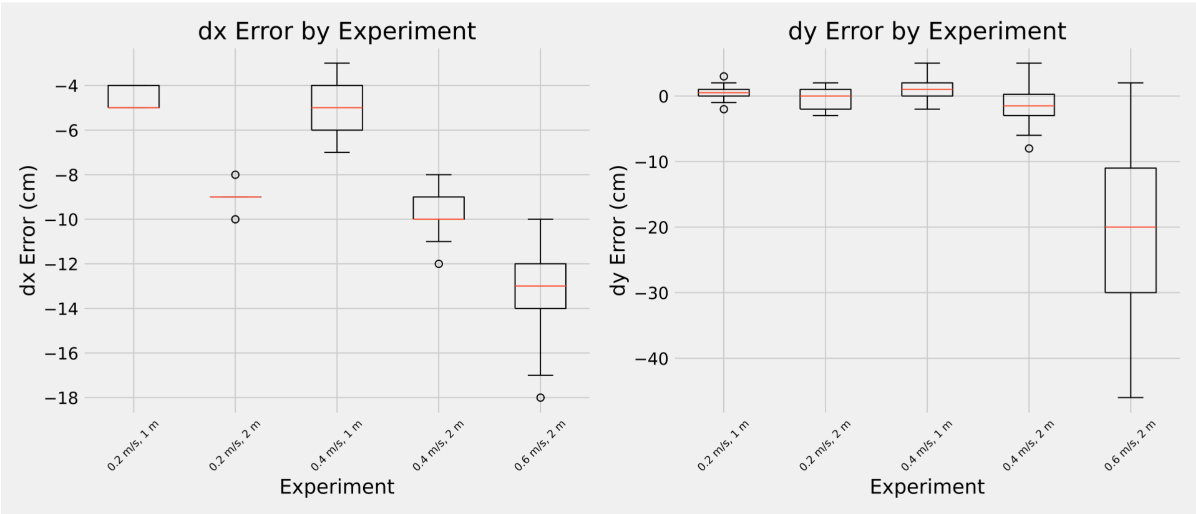


Figure 16. Box plot graph showing the errors in x and y for the linear motion tests. The boxplots show the distribution of errors for each speed and distance combination, with the mean error shown as an orange line.

5.3.2 Rotational Motion Error Characteristics

Rotational experiments exhibited both systematic and random error components. Despite significantly different speeds (45°/s vs 75°/s), their results were similar, suggesting that the robot's control system is more consistent in angular motion than linear. Y drift was generally small and random, with no clear systematic bias. However, x drift was more pronounced, with errors generally becoming larger with an increase in angular distance. The variance for x and y drift are consistently high, showing a lack of capability for the YouBot to do pure rotational motion without some translational drift, most likely due to wheel slip.

The yaw error (difference between odometry and actual rotation) was particularly interesting. At the lower target angle of 45°, the mean error was neglible. However, as the target angle increased, the yaw error became more negative, indicating over-rotation. This error is also roughly proportional to the commanded angle, suggesting a systematic bias in the robot's angular velocity estimation or control timing.

Speed (°/s)	Angle (°)	X		Y		Yaw (°)	
		Error	Var	Error	Var	Error	Var
45	45	0.3500	0.2525	0.6000	0.3150	-0.6841	1.3144
45	90	0.5750	0.7819	-0.6750	0.4569	-2.6462	1.5169
45	180	-0.5000	0.8000	0.0000	1.8000	-5.6806	11.0494
45	360	-1.2500	0.5875	-0.6000	0.3650	-13.9718	3.1095
75	45	-0.2500	0.2125	0.6000	0.2900	-0.3685	1.7324
75	90	-0.4250	0.9569	-1.1000	0.4150	-3.5968	8.4881
75	180	-1.3571	1.5748	-0.5714	2.2449	-6.9914	5.8088
75	360	-1.6000	1.6150	-0.8250	2.1069	-13.6940	11.8580

Table 4. Rotational motion statistics: Mean error and variance in position (m) and yaw (°)

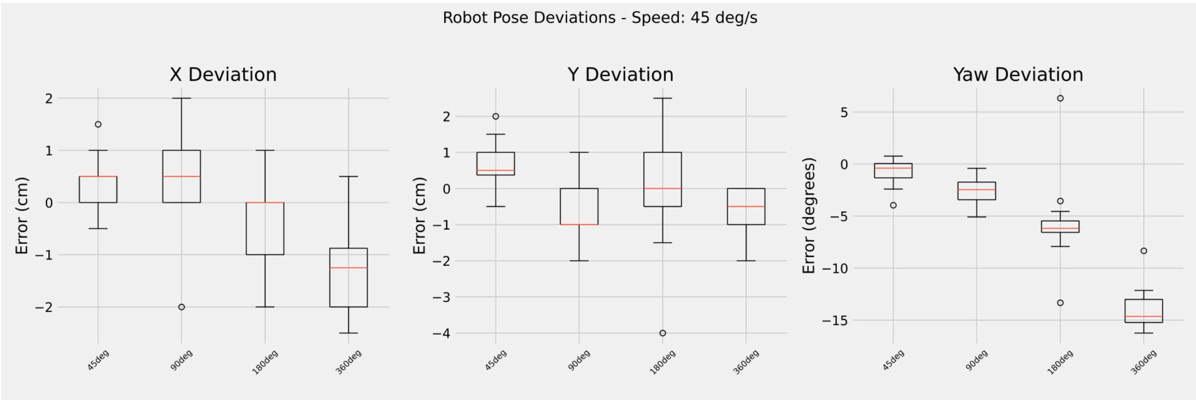


Figure 17. Box plot graph showing the errors in x, y, and yaw for the rotational motion tests at 45°/s.

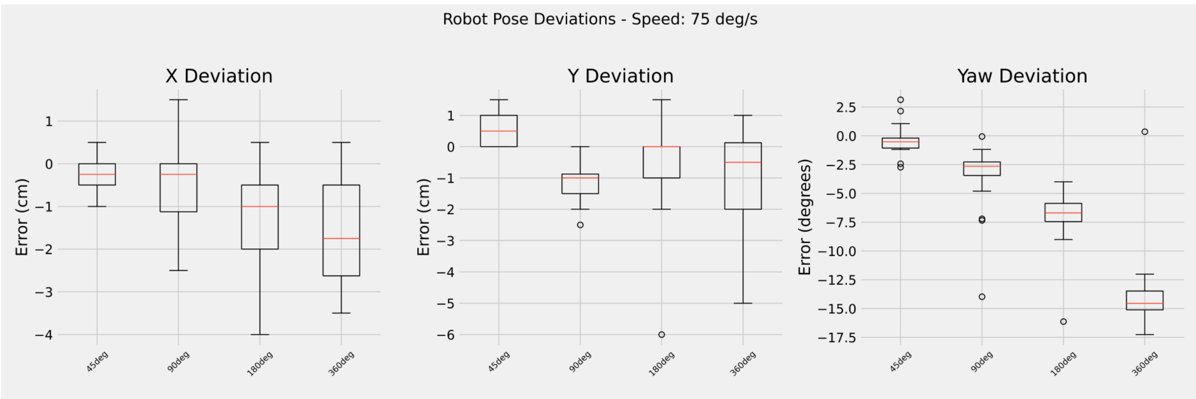


Figure 18. Box plot graph showing the errors in x, y, and yaw for the rotational motion tests at 75°/s.

5.4 Conclusion

Based on the experimental results, we can draw several conclusions about the YouBot's movement and odometry capabilities. Firstly, the YouBot exhibits systematic errors in both linear and rotational motion, with a tendency to underestimate distances and angles. The robot's odometry is more reliable for short distances and angles, but becomes less accurate as the commanded distance or angle increases. This error is roughly proportional to the commanded distance or angle, indicating a scale-dependent error. This error is also proportional to the speed of the robot, with higher speeds leading to larger errors and more variability in the results.

This suggests several potential issues. First, the YouBot's motors may be performing with reduced efficiency, leading to a loss of power delivered to the wheels. This furthermore results in a lower speed than expected, which in turn leads to a lower distance travelled than commanded. Second, the YouBot suffers from wheel slip, which is particularly pronounced at higher speeds. This theory would also explain the large error bars seen for in-place rotational movement in the x and y directions, as the robot is unable to maintain a consistent position while rotating.

These issues can be addressed through several strategies. For example, the YouBot's motors and motor encoders could be re-calibrated, taking into account the current state of the motors and their efficiency. This would allow the measured distance to more closely resemble the commanded distance. A closed feedback loop using PID could also be implemented to adjust the speed of the motors based on the measured distance travelled, allowing for more accurate movement. To address wheel slip, the YouBot's wheels could be completely cleaned or replaced with ones that have better traction to better resist slipping in real-world conditions.

6 Future Applications

This project has demonstrated that legacy hardware such as the KUKA YouBot can still offer meaningful value when approached with purpose. While the mobile base is no longer a competitive platform for cutting-edge industrial and research applications, it can act as a platform for conducting research, education and prototyping in the field of mobile robotics. This would allow us to make full use of the YouBot's capabilities, while also extending its operational life and reducing e-waste.

6.1 Automated Guided Vehicles (AGVs)

In the context of warehouse logistics and factory automation, the YouBot could emulate AGV systems for transporting light items within controlled environments. Traditional AGV systems, like those developed by Kollmorgen, utilize infrastructure-dependent navigation methods such as reflector stripes and laser-based positioning systems to follow predefined paths [19][20]. These systems require significant setup with deployment engineers configuring the infrastructure and programming routes [20].

The YouBot's omnidirectional drive system, with its four Mecanum wheels allowing movement in any direction, provides an excellent platform for prototyping and testing AGV applications [21]. This mobility system enables the YouBot to navigate tight spaces with precision, making it ideal for simulating warehouse logistics operations in controlled environments. While it lacks the industrial robustness of commercial AGVs, its compact size and reasonable payload capacity (20kg) make it suitable for educational demonstrations and small-scale prototyping of material handling solutions.

Companies like Kollmorgen are advancing AGV technologies with smart algorithms that optimize routing and facilitate flexible material handling, significantly reducing installation and commissioning time [20]. The YouBot can serve as an affordable platform to test similar algorithms before implementing them on more expensive industrial systems [21].

Figure 19. Kollmorgen NDC8 Hardware.

6.2 Human-Robot Interaction (HRI)

The YouBot's small size and mobility make it suitable for operation in tandem with humans in controlled environments, such as labs or workshops. This would allow for the testing of HRI algorithms and assistive applications, with a focus on navigation and cognition [22].

Modern HRI research focuses on developing robots that can interact naturally with humans, understand social cues, and operate safely in shared spaces. The YouBot provides an accessible platform for implementing and testing these interactions due to its non-threatening size and open software architecture. Researchers can develop applications that explore how robots and humans can collaborate effectively, testing various interaction paradigms before implementing them on more specialized platforms.

Recent advancements in HRI include frameworks like RISE (Robotics Integration and Scenario-Management Extensible-Architecture), which supports interdisciplinary research and reproducible human-robot dialog studies [22].

6.3 Autonomous Mobile Robots (AMRs)

As urban and industrial environments become more complex, the need for autonomous mobile robots (AMRs) that can navigate and operate in these environments is increasing [23]. Unlike traditional AGVs that follow fixed paths, AMRs are smart machines capable of replanning routes according to the current state of their environment [24].

The YouBot's ROS compatibility and open interfaces make it a suitable platform for testing and prototyping AMR algorithms, such as SLAM (Simultaneous Localization and Mapping) and real-time localization. SLAM algorithms enable robots to build maps of unknown environments while simultaneously tracking their position within those maps. This capability is fundamental for AMRs that need to navigate dynamically changing environments without predefined infrastructure.

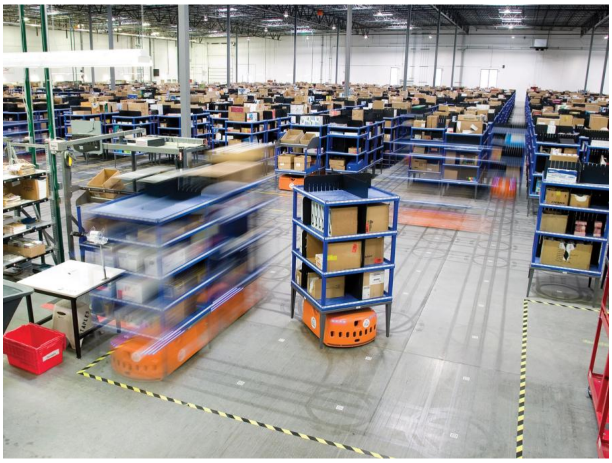


Figure 20. Amazon Robotics in action at a warehouse.

Companies like Amazon have revolutionized warehouse operations with their Kiva robots (now Amazon Robotics), which autonomously transport shelving units to human operators [25]. While these systems operate at industrial scale, the fundamental navigation and coordination principles can be studied and prototyped using platforms like the YouBot [26].

6.4 Swarm Robotics

Swarm robotics represents an emerging field where multiple simple robots collaborate to accomplish tasks that would be difficult or impossible for individual robots [27] [28]. This approach draws inspiration from natural swarm behaviors observed in social organisms like ants and bees, where collective intelligence emerges from simple individual behaviors and local interactions [27] [29].

The YouBot's compact size, omnidirectional mobility, and open software architecture make it suitable for small-scale swarm robotics experiments [27]. Multiple YouBots can be programmed to coordinate their movements and actions, demonstrating principles of distributed control and collective decision-making [27][29]. While industrial-scale swarm applications might require dozens or hundreds of robots, fundamental research can be conducted with smaller numbers of units.

6.5 Robotics Education and Prototyping

Modern robotics education emphasizes hands-on learning approaches that enhance problem-solving skills and boost creativity through practical application. The YouBot facilitates this approach by providing a complete system that integrates mechanical design, electronics, and software programming. Students learn to work with real-world challenges such as sensor noise, mechanical tolerances, and system integration issues that cannot be fully replicated in simulation environments. Furthermore, the age of the YouBot provides an interesting opportunity for users to create creative solutions to integrate the mobile base into modern toolsets. Students can use the platform to prepare for events like RoboCup, where KUKA has historically provided support and sponsorship.

7 Reflection

This project was a novel and challenging experience for me, as it was my first time dabbling in the field of robotics. It served as my first exposure to ROS, Linux, and the various hardware components of a robot. It was furthermore a very open-ended and self-directed project, where I had to manage my own time and resources, and make decisions about the direction of the project.

7.1 Project Evolution

As disccused previously, the initial goal of this project was much more concrete and ambitious, with an ultimate aim of enabling Teach and Repeat on a decade-old robot whilst using something like ROS2 Jazzy. This was simply not feasible with the outdated software and hardware, lack of documentation, my own lack of experience, and the overall time and resource constraints for completing this project.

My initial approach to project management was very linear and straightforward, with the understanding that there would be no hiccups or issues along the way. While this was initially rewarding as I made progress in individual tasks (i.e. Kinect camera, booting up computer, static IP, etc.), momentum was quickly lost when it came to integrating these tasks into a cohesive system, through the use of the ROS1-ROS2 bridge. This led me to feel overwhelmed and frustrated, as I was unable to make progress towards the initial goal. Thus, a pivot was necessary to reframe the project and its goals, and a change in perspective allowed me to make progress in a more manageable and healthy way.

7.2 AGILE

The new mindset I undertook was inspired by the AGILE project management methodology, which takes a much more flexible and iterative approach to project management. While the long-term goals are still important, the focus is on short-term goals and iterations. This allows for a more flexible approach, where the ending goal can be adjusted based on constraints, feedback, and progress.

While this also indirectly means that the project may not be concretely completed, it led to a more flexible exploration of the robot's capabilities and limitations. It furthermore relieved me of the pressure of having to complete a specific (and steep) goal, and gave me breathing space to play around with the available components more freely. This was much more rewarding and enjoyable.

Undertaking this mindset required time and patience, where I had to learn to accept that not everything would go according to plan. It furthermore required me to be more open to feedback, and actively focusing on short-term goals.

7.3 Conclusions

The revised technical goals and learning outcomes were mostly completed by the end of the project. I was able to successfully revive the KUKA YouBot to a usable state, gain a deeper understanding of the ROS ecosystem, Linux, bash scripting, and the various hardware components of the robot. I was also able to write custom programs to control the YouBot, and test its movement capabilities through controlled experiments. Soft skills such as project management, time management, and self-directed learning were also improved greatly.

There are several areas of the YouBot that could be further explored, such as an actual implementation for the ROS1-ROS2 bridge, further integration of sensors, and more advanced control algorithms. However, I am satisfied with the overall progress made despite my pitfalls.

I'd like to thank my advisor for his provision of a different perspective on the project, explanations of everything and consistent patience. I also want to thank Vesa Korhonen for his help with replacing the YouBot battery and his insights into the YouBot's hardware and documentation.

References

[1] H. McGloin et al., "Introducing the concept of repurposing robots; to increase their useful life, reduce waste, and improve sustainability in the robotics industry", in *International Conference on Robot Ethics and Standards*, 2023.

[2] KUKA, "Youbot product flyer", https://web.archive.org/web/20170829075021/http://www.youbot-store.com/media/wysiwyg/pdf/youbot_product_flyer.pdf, Accessed: 2025-06-05, 2017.

[3] Locomotec, "Kuka youbot user manual", version 1.02, Unpublished manual, available as PDF., Feb. 2013.

[4] "Youbot - Overview — github.com", https://github.com/youbot, [Accessed 23-04-2025].

[5] ubahnverleih, "Robocup 2016 leipzig - kuka youbot", https://commons.wikimedia.org/wiki/File:RoboCup_2016_Leipzig_-_KUKA_youBot_(2).jpg, Own work. Image retrieved from Wikimedia Commons., Jul. 2016.

[6] "Youbot detailed specifications", https://web.archive.org/web/20160121124150/http://www.youbot-store.com/wiki/index.php/YouBot_Detailed_Specifications, Accessed: 2025-06-05, 2016.

[7] OpenKinect, "Libfreenect: Drivers and libraries for the xbox kinect device on windows, linux, and os x", https://github.com/OpenKinect/libfreenect, Accessed: 2025-06-05, 2024.

[8] L. Hokuyo Automatic Co., "Urg-04lx-ug01 scanning rangefinder", https://www.hokuyo-aut.jp/search/single.php?serial=166, Accessed: 2025-06-05, 2016.

[9] "Mounting and sensor plate, kuka youbot store", https://web.archive.org/web/20160613151621/http://www.youbot-store.com/accessories/mounting-and-sensor-plate, Accessed: 2025-06-05, 2016.

[10] "Wayback machine", https://web.archive.org/, Accessed: 2025-06-05.

[11] "YouBot Driver — janpaulus.github.io", https://janpaulus.github.io/, [Accessed 23-04-2025].

[12] E. Shahrivar. "Visual teach and repeat: A closer look (part 1)". Accessed: 2025-06-05. [Online]. Available: https://clearpathrobotics.com/blog/2018/07/visual-teach-and-repeat-closer-look-part-1/.

[13] P. Nourizadeh, M. Milford, and T. Fischer, "Teach and repeat navigation: A robust control approach", in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 2024, pp. 2909–2916. DOI: 10.1109/ICRA57147.2024.10611662.

[14] M. Nitsche, F. Pessacg, and J. Civera, "Visual-inertial teach and repeat", *Robotics and Autonomous Systems*, vol. 131, p. 103 577, 2020, ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2020.103577. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889020304176.

[15] B. Masters, "How do sealed lead acid batteries work?" Accessed: 2025-06-16. [Online]. Available: https://batterymasters.co.uk/blog/post/How%20Do%20Sealed%20Lead%20Acid%20Batteries%20Work.

[16] PowerStream, "Lead acid battery charging basics and chargers", Accessed: 2025-06-16. [Online]. Available: https://www.powerstream.com/SLA.htm.

[17] P. Sonic, "How to charge a lead acid battery", Accessed: 2025-06-16, 2021. [Online]. Available: https://www.power-sonic.com/blog/how-to-charge-a-lead-acid-battery/.

[18] P. Sonic, "The complete guide to lithium vs lead acid batteries", Accessed: 2025-06-16. [Online]. Available: https://www.power-sonic.com/blog/lithium-vs-lead-acid-batteries/.

[19] Kollmorgen, "Kollmorgen", Accessed: 2025-06-18, 2024. [Online]. Available: https://automaticguidedvehicles.com/kollmorgen/.

[20] Kollmorgen, "Agvs using smart algorithms for smart factories", Accessed: 2025-06-18, 2017. [Online]. Available: https://www.youtube.com/watch?v=T3Z2CKDhJEo.

[21] M. Group, "Application story robotics: Kuka youbot", Tech. Rep., 2015, Accessed: 2025-06-18. [Online]. Available: https://www.maxongroup.in/medias/sys_master/8815857795102.pdf?attachment=true.

[22] A. Groß et al., "Rise: An open-source architecture for interdisciplinary and reproducible human-robot interaction research", *Frontiers in Robotics and AI*, vol. 10, p. 1 245 501, 2023, Published: 2023-12-06. DOI: 10.3389/frobt.2023.1245501.

[23] I. Mecalux, "Slam navigation in robotics", Accessed: 2025-06-18, 2023. [Online]. Available: https://www.interlakemecalux.com/blog/slam-navigation-robotics.

[24] MathWorks, "What is slam?" Accessed: 2025-06-18, 2024. [Online]. Available: https://www.mathworks.com/discovery/slam.html.

[25] Amazon, "10 years of amazon robotics: How robots help sort packages, move product, and improve safety", Accessed: 2025-06-18, 2022. [Online]. Available: https://www.aboutamazon.com/news/operations/10-years-of-amazon-robotics-how-robots-help-sort-packages-move-product-and-improve-safety.

[26] Exotec, "How amazon robotics has changed the landscape of fulfillment", Accessed: 2025-06-18, 2023. [Online]. Available: https://www.exotec.com/insights/how-amazon-robotics-has-changed-the-landscape-of-fulfillment/.

[27] F. Mondada et al., "Swarm-bot: A new distributed robotic concept", IDSIA, Tech. Rep., 2004, Accessed: 2025-06-18. [Online]. Available: https://people.idsia.ch/~luca/swarmbot-hardware.pdf.

[28] U. P. de Madrid, "Swarm robotics research", Accessed: 2025-06-18, 2020. [Online]. Available: https://oa.upm.es/id/eprint/1947/contents.

[29] AZoRobotics, "Swarm robotics applications", Accessed: 2025-06-18, 2021. [Online]. Available: https://www.azorobotics.com/Article.aspx?ArticleID=657.

A User Manual for the KUKA YouBot

A.1 Booting Up the YouBot

To begin operating the YouBot, follow these steps to power up the system and access the onboard computer interface using a monitor and peripherals.

A.1.1 Powering On

1. Plug the YouBot into a power source using the provided power cable and adapter through the top panel. Ensure that the supply voltage is compatible (24V DC).
2. Press the main power button located on the top panel of the YouBot. This will power the system on. Note that the onboard computer and motors are not powered on by default.
3. Connect a monitor using the provided VGA port. Note that this is required to be plugged in before powering on the YouBot, as the onboard computer will not boot up with a GUI otherwise.
4. Turn the onboard computer and motors on. Long press the power button to cycle through options for powering the subsystems. Toggle by releasing the button when the desired option is highlighted.
5. Wait for the onboard computer to boot up. This may take a few minutes.
6. Plug in USB peripherals such as a keyboard and mouse to interact with the onboard computer.
7. Once the computer has booted, you should see a desktop environment similar to a standard Linux distribution. You can now interact with the onboard computer using the keyboard and mouse. Note that the default credentials are:
   - Username: `youbot`
   - Password: `youbot`

add image of the booting up process.

A.2 Initial Configuration

After booting up the YouBot, some initial configuration is required to enable the ROS interface and the YouBot drivers. This includes setting up the ROS networking configuration, configuring the Ethernet interface, and editing the YouBot driver configuration file.

A.2.1 ROS Networking Configuration (`.bashrc`)

Update the onboard computer's `.bashrc` file with the following:

```
export MY_IP=localhost

export ROS_IPV6=off
export ROS_HOSTNAME=$MY_IP
export ROS_MASTER_URI=http://$MY_IP:11311
export ROS_IP=$MY_IP

export ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH://home/youbot/youbot_driver:/home/
    youbot/ros_stacks:/home/youbot/applications
```

```
export LIBGL_ALWAYS_SOFTWARE=1

source /opt/ros/hydro/setup.bash
```

To verify configuration:

```
$ roscore
```

A.2.2 Ethernet Configuration (/etc/network/interfaces)

To enable stable communication between the YouBot and external systems, a static IP address is preferably assigned to the onboard Ethernet interface. This enables reliable wired access for remote control.

ote: Not all Linux distributions name Ethernet interfaces as `eth0`, and instead may use interfaces named `eth1`, `enp2s0`, etc. To identify the correct interface, use the following command:

```
$ ip link show
```

Look for an interface name associated with a physical Ethernet port. In the following instructions, replace `eth0` with the appropriate interface name.

Assign a permanent static ethernet IP by editing the file `/etc/network/interfaces` and adding:

```
auto eth0
iface eth0 inet static
    address 192.168.10.1 ## The static IP address to be assigned
    netmask 255.255.255.0 ## Standard subnet mask
```

Replace `eth0` with your actual interface name as determined above. Save the file and apply changes:

```
$ sudo /etc/init.d/networking restart
```

Confirm the static IP assignment by checking the interface configuration:

```
$ ip addr show eth0
```

Note: An alternative to an Ethernet configuration is to assign a static IP via the network router on a wireless interface. This eliminates a wired connection dependency and allows the YouBot to obtain a static IP automatically when connected to the same network. Refer to your router's documentation for specific instructions on conducting DHCP reservations.

Testing the Connection After configuration, connect another device (e.g., a laptop) to the YouBot via Ethernet. Assign the other device a static IP in the same subnet (e.g., `192.168.10.2`). Then test connectivity:

```
$ ping 192.168.10.1 ## From the laptop to the YouBot
```

For remote terminal access:

```
$ ssh youbot@192.168.10.1
```

Enter the password when prompted (default is `youbot`). If the connection is successful, the YouBot should be accessible through an external PC.

A.2.3 Youbot Driver Configuration (`~/youbot_driver/config/youbot-ethercat.cfg`)

The YouBot driver requires the correct Ethernet interface to be specified for EtherCAT communication with the base and arm motors. This is configured in the file `~/youbot_driver/config/youbot-ethercat.cfg`. Note that the interface name used here may be different from the Ethernet interface name used in the previous step. Type the following to see your Ethernet interfaces:

```
$ ifconfig
```

Change the Ethernet interface name in the youbot configuration file (`~/youbot_driver/config/youbot-ethercat.cfg`) to match your actual interface name:

```
#EtherCAT port
[EtherCAT]
EthernetDevice = eth1 # Change this to your actual EtherCAT interface name
```

This will enable the YouBot driver to communicate with the motors and sensors through the correct port for the C++ demos. Run and test either the C++ demos or the ROS interface to verify that the configuration is correct.

A.3 Running C++ demos

Note: Remember to turn the motors on before running the demos. This can be done by long-pressing the power button on the top panel of the YouBot and toggling the "Motors" option to ON.

The YouBot provides precompiled C++ demo applications located in `~/applications`. These are also available from the official `youbot_applications` GitHub repository. The demos include basic tests for motor communication and manual control interfaces.

A.3.1 Hello World Demo

This demo verifies that the EtherCAT communication is working correctly by issuing movement commands to the robot.

To run it:

```
$ cd ~/applications/hello_world_demo/bin
$ sudo ./youBot_HelloWorldDemo
```

If successful, the output will indicate that the application is connected to the motors, and the robot should briefly move in the four cardinal directions (forward, backward, left, and right), confirming functional motor control.

A.3.2 Keyboard Control Demo

This demo enables basic manual control of the YouBot using the keyboard. It allows testing of real-time responsiveness and manual teleoperation.

Run it with:

```
$ cd ~/applications/keyboard_control_demo/bin
$ sudo ./youBot_KeyboardControlDemo
```

Use the on-screen instructions to control the robot's movement via the keyboard.

One may run the other demos in a similar manner. Refer to the GitHub repository's README for more details.

A.4 Running the ROS Interface

Once the C++ demos have been confirmed functional, the ROS (Robot Operating System) interface can be used to enable modular software integration for motion planning, visualization, diagnostics, and teleoperation.

A.4.1 Verifying Installed ROS Packages

To verify the required packages are installed on the YouBot, run:

```
$ rospack list
```

Ensure the following packages are present:

- `youbot_driver_ros_interface` – Core ROS hardware interface for the YouBot.
- `youbot_teleop` – ROS node for keyboard-based teleoperation.
- `youbot_common` – Common message definitions and configuration files.
- `youbot_description` – URDF robot model and related assets.

A.4.2 Launching the ROS Driver

Before launching the ROS interface, ensure the correct environment variables are set (see section: *ROS Networking Configuration*). Then launch the ROS master and the YouBot driver:

```
$ roscore
$ roslaunch youbot_driver_ros_interface youbot_driver.launch
```

The driver initializes the EtherCAT interface and exposes the robot's hardware components to the ROS environment.

A.4.3 Inspecting Nodes and Topics

Use the following commands to inspect active ROS nodes and topics:

```
$ rosnode list
$ rostopic list
```

Example nodes:

- `/youbot_driver` – Node that manages all hardware-level communication.
- `/rosout` – Default ROS logging node.

Example topics:

- `/joint_states` – Publishes joint positions and velocities.
- `/cmd_vel` – Accepts velocity commands for the base.
- `/tf` – Publishes coordinate frame transforms.

A.4.4 Using Visualization Tools: `rqt` and `rviz`

ROS visualization tools are used to monitor system behavior and robot state:

- `rqt_graph` – Displays real-time node and topic communication graph.
- `rviz` – Visualizes robot model, coordinate frames, sensor data, and state feedback.

Run the following commands in separate terminals:

```
$ rosrun rqt_graph rqt_graph
$ rosrun rviz rviz
```

A.5 Writing Custom Programs

This section describes how to create and run custom ROS programs to control the YouBot using Python. The provided ROS demo packages serve as a reference.

A.5.1 Creating a Catkin Workspace

Before writing custom ROS programs, create a Catkin workspace to manage packages and dependencies. Note that we have named our folder `DSD` to reflect the course name.

```
$ mkdir -p ~/DSD/catkin_ws/src
$ cd ~/DSD/catkin_ws/
$ catkin_make
```

Source the workspace to update your ROS environment:

```
$ source devel/setup.bash
```

To source automatically in every new terminal, add this line to your `.bashrc`:

```
echo "source ~/DSD/catkin_ws/devel/setup.bash" >> ~/.bashrc
```

A.5.2 Creating a Custom Package

Create a new ROS package named `youbot_control` with dependencies:

```
$ cd ~/DSD/catkin_ws/src
$ catkin_create_pkg youbot_control rospy std_msgs geometry_msgs #
    youbot_control is the package name, the following are dependencies
$ cd ~/DSD/catkin_ws
$ catkin_make
```

A.5.3 Writing a Python Control Script

Inside your package directory, create a `scripts` folder and a Python control script:

```
$ cd ~/DSD/catkin_ws/src/youbot_control
$ mkdir scripts
$ touch scripts/simple_move.py
$ chmod +x scripts/simple_move.py # Make the script executable
```

Sample code files can be found in appendix for code

To run a script:

```
$ rosrun youbot_control simple_move.py
```

Note: Note that certain scripts require additional arguments to be passed in. Refer to the comments for details.

A.5.4 Automation with Bash Scripts and Tmux

Manually opening multiple terminals and running ROS nodes is tedious. This process can be automated using a combination of bash scripts and `tmux`.

First, install `tmux` if not already available:

```
$ sudo apt-get install tmux
```

Then, create a bash script to launch the environment in multiple panes:

```
$ touch launch_youbot.sh
$ chmod +x launch_youbot.sh
```

Sample content for `launch_youbot.sh` can be found in appendix for code. This script will create a new `tmux` session with three panes: one for `roscore`, one for the YouBot driver, and one for a custom movement script. Run the script by navigating to its directory and running the command:

```
$ ./launch_youbot.sh
```

Note that a password will be required to initialize the YouBot-ROS interface. Navigate between panes using `Ctrl+b` followed by the arrow keys. Detach from the session with `Ctrl+b d` and reattach later with:

```
$ tmux attach-session -t youbot
```

Refer to the `tmux` documentation for more advanced usage and commands.

A.5.5 Using TMuLE for Configuration Management

One may use TMuLE to simplify the scripting process further using toml files. The youbot did not allow for the installation of TMuLE due to no support for Python 2.7, so this was not used in this project. However, it is a useful tool to consider for future projects. Refer to the documentation here.

B Code files

B.1 Custom Scripts for YouBot ROS Interface testing and development

B.1.1 simple_square_movement.py

This script was used to test the YouBot's ability to move in a square pattern. It uses the ROS interface to send velocity commands to the YouBot.

```python
#!/usr/bin/env python

import rospy
from geometry_msgs.msg import Twist

import time, math
rospy.init_node('simple_square_movement', anonymous=True)
def move_fwd():


  #Publisher to /cmd_vel
  pub = rospy.Publisher('/cmd_vel', Twist, queue_size=10)

  #Create a Twist message for movement
  move_cmd = Twist()
  move_cmd.linear.x = 0.2 # Move at 0.1 m/s

  #Set distance and calculate time needed
  distance = 1
  speed = move_cmd.linear.x
  move_time = distance / speed

  #Set a rate to publish messages

  start_time = time.time()

  while time.time() - start_time < move_time and not rospy.is_shutdown():
    pub.publish(move_cmd)


  #After 10 cm, stop robot
  move_cmd.linear.x = 0
  pub.publish(move_cmd)

  rospy.loginfo("Moved forward 10cm.")
  rospy.sleep(2)

def turn_right():


  #Publisher to /cmd_vel
  pub = rospy.Publisher('/cmd_vel', Twist, queue_size=10)
```

```
    #Create a Twist message for movement
    move_cmd = Twist()
    move_cmd.angular.z = -math.pi/5 # Move at 0.1 m/s
    rospy.loginfo(math.pi)


    #Set distance and calculate time needed
    distance = math.pi*89 / 180
    speed = move_cmd.angular.z
    move_time = -distance / speed

    #Set a rate to publish messages

    start_time = time.time()

    while time.time() - start_time < move_time and not rospy.is_shutdown():
      pub.publish(move_cmd)


    #After 10 cm, stop robot
    move_cmd.angular.z = 0
    pub.publish(move_cmd)

    rospy.loginfo("Turned 90 degrees clockwise.")
    rospy.sleep(2)


  if __name__ == '__main__':
    try:
      move_fwd()
      turn_right()
      move_fwd()
      turn_right()
      move_fwd()
      turn_right()
      move_fwd()
      turn_right()
    except rospy.ROSInterruptException:
      pass
```

B.1.2 Tmux automation script

This script is used to automate the process of launching the ROS interface and the custom control scripts in a tmux session. It creates a new tmux session with four panes: one running the YouBot-ROS interface, one running the keyboard controller node, and two free panes for miscellaneous purposes.

```
#!/bin/bash

# Create a new tmux session named 'quadrants', in detached mode
tmux new-session -d -s quadrants

# Split the window into two vertical panes (left and right)
tmux split-window -h

# Split the left pane into two horizontal panes (top-left and bottom-left)
tmux split-window -v -t quadrants:0.0

# Split the right pane into two horizontal panes (top-right and bottom-right)
tmux split-window -v -t quadrants:0.1

# Start youbot-ROS interface in the top-left pane (quadrants:0.0)
tmux send-keys -t quadrants:0.0 "sudo bash -c 'source /opt/ros/hydro/setup.bash &&
    roslaunch youbot_driver_ros_interface youbot_driver.launch'
" C-m

# Start the keyboard control interface in the bottom-left pane (quadrants:0.2)
tmux send-keys -t quadrants:0.2 "rosrun youbot_control keyboard.py" C-m

# Send 'echo 3' to the top-right pane (quadrants:0.1)
tmux send-keys -t quadrants:0.1 "echo 3" C-m

# Send 'echo 4' to the bottom-right pane (quadrants:0.3)
tmux send-keys -t quadrants:0.3 "echo 4" C-m

# Attach to the tmux session so you can see the result
tmux attach-session -t
```

B.2 Scripts for Experimental Setup and Data Collection

B.2.1 linear_test.py

This script is used to undertake the linear movement experiments. It uses the ROS interface to send velocity commands to the YouBot, and then calculates the delta in odometry from the robot. The script is designed to be run multiple times with different parameters through the terminal to collect data for analysis. Refer to insert something here idr for further details.

```
#!/usr/bin/env python

import rospy
from geometry_msgs.msg import Twist
from nav_msgs.msg import Odometry
import sys

import time, math, tf
rospy.init_node('rotate_test', anonymous=True)
def move_fwd(speed, distance):


  #Publisher to /cmd_vel
  pub = rospy.Publisher('/cmd_vel', Twist, queue_size=10)

  #Create a Twist message for movement
  move_cmd = Twist()
  move_cmd.linear.x = float(speed) # m/s

  #Calculate time needed
  move_time = float(distance) / float(speed)

  #Set a rate to publish messages

  start_time = time.time()

  while time.time() - start_time < move_time and not rospy.is_shutdown():
    pub.publish(move_cmd)


  #After specified distance, stop robot
  move_cmd.linear.x = 0
  pub.publish(move_cmd)

  rospy.loginfo("Moved forward " + str(int(100 * distance)) + " cm.")
  rospy.loginfo("%.2f", move_time)

def get_yaw_from_quaternion(ori_q):
  qu = (
    ori_q.x,
    ori_q.y,
    ori_q.z,
```

```
    ori_q.w
  )
  _, _, yaw = tf.transformations.euler_from_quaternion(qu)
  return yaw

def odom_get():
  try:
    rospy.sleep(0.2)
    odom_msg = rospy.wait_for_message('/odom', Odometry, timeout=5)
    position = odom_msg.pose.pose.position
    orientation = odom_msg.pose.pose.orientation
    yaw = get_yaw_from_quaternion(orientation)

    return (position.x, position.y, position.z, yaw)
  except rospy.ROSException:
    rospy.logwarn("Timeout while waiting for odom msg.")

def odom_delta_print(start, end):
  dx = end[0] - start[0]
  dy = end[1] - start[1]
  dz = end[2] - start[2]
  dyaw = end[3] - start[3]

  dyaw = (dyaw + math.pi) % (2 * math.pi) - math.pi
  rospy.loginfo('Odometry Change:')
  rospy.loginfo(' dx = %.3f m:', dx)
  rospy.loginfo(' dy = %.3f m:', dy)
  rospy.loginfo(' dz = %.3f m:', dz)
  rospy.loginfo(' dyaw = %.3f rad (%.1f deg):', dyaw, math.degrees(dyaw))


if __name__ == '__main__':
  try:
    start = odom_get()
    if start is None:
      rospy.loginfo("Failed to get initial position.")
    move_fwd(float(sys.argv[1]), float(sys.argv[2]))
    end = odom_get()
    if end is None:
      rospy.loginfo("Failed to get final position.")

    odom_delta_print(start,end)
  except rospy.ROSInterruptException:
    pass
```

B.2.2 rotate_test.py

This script is used to undertake the linear movement experiments. It uses the ROS interface to send velocity commands to the YouBot, and then calculates the delta in odometry from the robot. The script is designed to be run multiple times with different parameters through the terminal to collect data for analysis. Refer to insert something here idr for further details.

```
#!/usr/bin/env python

import rospy
from geometry_msgs.msg import Twist
from nav_msgs.msg import Odometry
import sys

import time, math, tf
rospy.init_node('rotate_test', anonymous=True)

def turn_left(speed, distance):
  # speed and distance are in degrees here.

  #Publisher to /cmd_vel
  pub = rospy.Publisher('/cmd_vel', Twist, queue_size=10)

  #Create a Twist message for movement
  move_cmd = Twist()
  move_cmd.angular.z = float(speed) * math.pi / 180# rad/s

  #Set distance and calculate time needed
  move_time = float(distance) / float(speed)

  #Set a rate to publish messages

  start_time = time.time()

  while time.time() - start_time < move_time and not rospy.is_shutdown():
    pub.publish(move_cmd)


  #After specific distance, stop robot
  move_cmd.angular.z = 0
  pub.publish(move_cmd)

  rospy.loginfo("Turned %.2f degrees.", distance)

def get_yaw_from_quaternion(ori_q):
  qu = (
    ori_q.x,
    ori_q.y,
    ori_q.z,
    ori_q.w
  )
```

```
  _, _, yaw = tf.transformations.euler_from_quaternion(qu)
  return yaw

def odom_get():
  try:
    rospy.sleep(2.0)
    odom_msg = rospy.wait_for_message('/odom', Odometry, timeout=5)
    position = odom_msg.pose.pose.position
    orientation = odom_msg.pose.pose.orientation
    yaw = get_yaw_from_quaternion(orientation)

    return (position.x, position.y, position.z, yaw)
  except rospy.ROSException:
    rospy.logwarn("Timeout while waiting for odom msg.")

def odom_delta_print(start, end):
  dx = end[0] - start[0]
  dy = end[1] - start[1]
  dz = end[2] - start[2]
  dyaw = end[3] - start[3]

  dyaw = (dyaw + math.pi) % (2 * math.pi) - math.pi
  rospy.loginfo('Odometry Change:')
  rospy.loginfo(' dx = %.3f m:', dx)
  rospy.loginfo(' dy = %.3f m:', dy)
  rospy.loginfo(' dz = %.3f m:', dz)
  rospy.loginfo(' dyaw = %.3f rad (%.1f deg):', dyaw, math.degrees(dyaw))


if __name__ == '__main__':
  try:
    start = odom_get()
    if start is None:
      rospy.loginfo("Failed to get initial position.")
    turn_left(float(sys.argv[1]), float(sys.argv[2]))
    end = odom_get()
    if end is None:
      rospy.loginfo("Failed to get final position.")

    odom_delta_print(start,end)
  except rospy.ROSInterruptException:
    pass
```

B.3 Graph Plotting Scripts

B.3.1 rotate_graph.py

This script read the data collected from the rotational movement experiments to parse the data and plot the drift of the centre of the robot in x and y axes as well as difference in yaw in a graph format. It also prints out the mean and standard deviation of the relevant errors. Refer to insert something here idr for further details.

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os
from glob import glob

# === Config toggles ===
PRINT_COLUMN_DATA = False
PRINT_STATS = True
SAVE_FIG = True
SHOW_FIG = False

CSV_DIR = 'code/csv/' # Adjust as needed
DATA_DIR = 'code/images/'

plt.style.use('fivethirtyeight')
#plt.rcParams["font.family"] = "Times New Roman"

def compute_pose_errors_and_print(df, rotation_deg, speed):
    theta = np.radians(rotation_deg)
    expected_left = np.array([30 * np.cos(theta), 30 * np.sin(theta)])
    expected_right = np.array([-30 * np.cos(theta), -30 * np.sin(theta)])
    expected_center = (expected_left + expected_right) / 2
    expected_yaw = theta % (2 * np.pi)

    x_errors, y_errors, yaw_errors = [], [], []

    if PRINT_COLUMN_DATA:
        print(f"\nResults for {speed} deg/s, {rotation_deg} deg:")
        print(f"{'Row':<5} {'Center_X':>10} {'Center_Y':>10} {'Yaw_deg':>10} "
              f"{'dx_odom':>8} {'dy_odom':>8} {'dyaw_odom_deg':>14}")

    for idx, row in df.iterrows():
        actual_left = expected_left + np.array([row['dx1_measured'], row['
            dy1_measured']])
        actual_right = expected_right + np.array([row['dx2_measured'], row['
            dy2_measured']])
        center = (actual_left + actual_right) / 2

        vector_left_to_right = actual_left - actual_right
        yaw_rad = np.arctan2(vector_left_to_right[1], vector_left_to_right[0])

        # Compare odometry-based pose vs measured pose
        x_err = row['dx_odom'] - center[0]
```

```
        y_err = row['dy_odom'] - center[1]
        odom_yaw = row['dyaw_odom']
        yaw_diff = (odom_yaw - yaw_rad + np.pi) % (2 * np.pi) - np.pi

        x_errors.append(x_err)
        y_errors.append(y_err)
        yaw_errors.append(yaw_diff)

        if PRINT_COLUMN_DATA:
            print(f"{idx:<5} {center[0]:10.3f} {center[1]:10.3f} {np.degrees(yaw_rad
                ):10.2f} "
                  f"{row['dx_odom']:8.3f} {row['dy_odom']:8.3f} {np.degrees(row['
                      dyaw_odom']):14.2f}")

    if PRINT_STATS:
        mean_yaw_deg = np.degrees(np.mean(yaw_errors))
        var_yaw_deg = np.var(np.degrees(yaw_errors))
        print(f"\nStatistics for {speed} deg/s, {rotation_deg} deg:")
        print(f" Mean X Error: {np.mean(x_errors):.4f}, Variance: {np.var(x_errors)
            :.4f}")
        print(f" Mean Y Error: {np.mean(y_errors):.4f}, Variance: {np.var(y_errors)
            :.4f}")
        print(f" Mean Yaw Error (deg): {mean_yaw_deg:.4f}, Variance (deg sq): {
            var_yaw_deg:.4f}")

    return np.array(x_errors), np.array(y_errors), np.array(yaw_errors)

def plot_rotational_errors_by_speed():
    file_list = sorted(glob(os.path.join(CSV_DIR, 'rotate_*.csv')))

    grouped_data = {}

    for file_path in file_list:
        filename = os.path.splitext(os.path.basename(file_path))[0]
        try:
            _, speed_str, deg_str = filename.split('_')
            speed = int(speed_str)
            rotation_deg = int(deg_str)
        except ValueError:
            print(f"Skipping invalid filename format: {filename}")
            continue

        df = pd.read_csv(file_path)
        x_err, y_err, yaw_err = compute_pose_errors_and_print(df, rotation_deg,
            speed)

        grouped_data.setdefault(speed, []).append((rotation_deg, x_err, y_err, np.
            degrees(yaw_err)))

    for speed in sorted(grouped_data.keys()):
        group = sorted(grouped_data[speed], key=lambda x: x[0])
```

```python
        x_err_all = [item[1] for item in group]
        y_err_all = [item[2] for item in group]
        yaw_err_all = [item[3] for item in group]
        labels = [f"{item[0]}deg" for item in group]

        fig, axes = plt.subplots(1, 3, figsize=(18, 6))

        axes[0].boxplot(x_err_all, labels=labels)
        axes[0].set_title('X Deviation')
        axes[0].set_ylabel('Error (cm)')
        axes[0].tick_params(axis='x', rotation=45, labelsize=9)

        axes[1].boxplot(y_err_all, labels=labels)
        axes[1].set_title('Y Deviation')
        axes[1].set_ylabel('Error (cm)')
        axes[1].tick_params(axis='x', rotation=45, labelsize=9)

        axes[2].boxplot(yaw_err_all, labels=labels)
        axes[2].set_title('Yaw Deviation')
        axes[2].set_ylabel('Error (degrees)')
        axes[2].tick_params(axis='x', rotation=45, labelsize=9)

        plt.suptitle(f'Robot Pose Deviations - Speed: {speed:.0f} deg/s')
        plt.tight_layout(rect=[0, 0.03, 1, 0.95])

        if SAVE_FIG:
            if not os.path.exists(DATA_DIR):
                os.makedirs(DATA_DIR)
            filename = f'rotational_errors_speed_{int(speed):02d}.png'
            plt.savefig(os.path.join(DATA_DIR, filename), dpi=600)

        if SHOW_FIG:
            plt.show()
        else:
            plt.close()

if __name__ == "__main__":
    plot_rotational_errors_by_speed()
```

B.3.2 linear_graph.py

This script read the data collected from the linear movement experiments to parse the data and plot the drift of the centre of the robot in x and y axes in a graph format. It also prints out the mean and standard deviation of the relevant errors. Refer to insert something here idr for further details.

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os
from glob import glob

# === Config toggles ===
PRINT_COLUMN_DATA = False
PRINT_STATS = True
SAVE_FIG = True
SHOW_FIG = False

CSV_DIR = 'code/csv/' # Adjust as needed
DATA_DIR = 'code/images/'

plt.style.use('fivethirtyeight')
#plt.rcParams["font.family"] = "Times New Roman"

def print_column_data(label, dx_meas, dy_meas, dx_odom, dy_odom):
    print(f"\nResults for {label}:")
    print(f"{'Row':<5} {'dx_meas':>10} {'dy_meas':>10} {'dx_odom':>10} {'dy_odom
        ':>10}")
    for i in range(len(dx_meas)):
        print(f"{i:<5} {dx_meas[i]:10.3f} {dy_meas[i]:10.3f} {dx_odom[i]:10.3f} {
            dy_odom[i]:10.3f}")


def print_stats(label, errors_dict):
    print(f"\nStatistics for {label}:")
    for key, errors in errors_dict.items():
        mean = np.mean(errors)
        median = np.median(errors)
        var = np.var(errors)
        print(f" {key} -> Mean: {mean:.4f}, Median: {median:.4f}, Variance: {var:.4
            f}")

def plot_linear_errors():
    file_list = sorted(glob(os.path.join(CSV_DIR, 'linear_*.csv')))

    dx_errors_by_exp = []
    dy_errors_by_exp = []
    labels = []

    for file_path in file_list:
        df = pd.read_csv(file_path)
```

```python
    data = list(df.itertuples(index=False, name=None))

    dx_meas = np.array([t[0] for t in data])
    dy_meas = np.array([t[1] for t in data])
    dx_odom = np.array([t[2] for t in data])
    dy_odom = np.array([t[3] for t in data])

    error_dx = dx_odom - dx_meas
    error_dy = dy_odom - dy_meas

    dx_errors_by_exp.append(error_dx)
    dy_errors_by_exp.append(error_dy)

    filename = os.path.splitext(os.path.basename(file_path))[0]
    parts = filename.split('_')
    speed = float(parts[1]) / 10
    distance = int(parts[2])
    label = f"{speed:.1f} m/s, {distance} m"
    labels.append(label)

    if PRINT_COLUMN_DATA:
        print_column_data(label, dx_meas, dy_meas, dx_odom, dy_odom)


    if PRINT_STATS:
        print_stats(label, {'dx_error': error_dx, 'dy_error': error_dy})

fig, axes = plt.subplots(1, 2, figsize=(14, 6))

axes[0].boxplot(dx_errors_by_exp, labels=labels)
axes[0].set_title('dx Error by Experiment')
axes[0].set_xlabel('Experiment')
axes[0].set_ylabel('dx Error (cm)')
axes[0].grid(True)
axes[0].tick_params(axis='x', rotation=45, labelsize=9)

axes[1].boxplot(dy_errors_by_exp, labels=labels)
axes[1].set_title('dy Error by Experiment')
axes[1].set_xlabel('Experiment')
axes[1].set_ylabel('dy Error (cm)')
axes[1].grid(True)
axes[1].tick_params(axis='x', rotation=45, labelsize=9)


plt.tight_layout()

if SAVE_FIG:
    save_path = os.path.join(DATA_DIR, 'linear_error_boxplots.png')
    plt.savefig(save_path, dpi=600)
if SHOW_FIG:
    plt.show()
```

```python
if __name__ == "__main__":
    plot_linear_errors()
```

C Experimental Data

C.1 Linear movement experiment

C.1.1 Linear movement, 0.2m/s, 1m

dx_meas	dy_meas	dx_odom	dy_odom	dz_odom	dyaw_odom
100	-4	96	-1	0	-0.012
103	-2	98	-2	0	-0.012
102	-1	98	-3	0	0.003
99	0	95	-1	0	-0.006
102	-2	98	0	0	-0.013
100	-1	95	0	0	-0.001
102	-1	98	0	0	0.005
103	-2	98	0	0	0.005
104	-1	99	-1	0	0.006
102	-2	97	0	0	-0.004
103	0	98	0	0	-0.010
104	-1	99	-1	0	0.003
102	-1	98	-1	0	0.004
102	0	97	0	0	0.002
100	0	96	-1	0	0.012
103	-1	98	0	0	0.007
103	-1	98	0	0	0.003
100	-1	96	0	0	0.004
100	-1	96	0	0	-0.003
103	0	98	-1	0	0.008

C.1.2 Linear movement, 0.2m/s, 2m

dx_meas	dy_meas	dx_odom	dy_odom	dz_odom	dyaw_odom
207	2	198	-1	0	0.012
204	0	195	0	0	0.002
207	0	199	0	0	-0.014
205	-1	196	0	0	0.001
206	0	197	0	0	0.003
207	0	198	0	0	-0.006
208	-2	199	0	0	0.000
208	-2	198	0	0	0.008
208	2	199	0	0	-0.001
208	1	199	0	0	0.009
205	2	196	0	0	0.004
209	2	200	0	0	0.007
205	1	196	0	0	0.000
206	2	197	0	0	0.014
208	-1	199	0	0	0.000
209	-2	199	0	0	-0.013
207	2	198	-1	0	0.012
207	-2	198	0	0	-0.008
207	0	198	0	0	0.004
205	3	196	0	0	0.002

C.1.3 Linear movement, 0.4m/s, 1m

dx_meas	dy_meas	dx_odom	dy_odom	dz_odom	dyaw_odom
94	3	88	2	0	0.021
99	-2	94	3	0	-0.005
101	0	95	2	0	0.002
93	1	88	2	0	0.015
100	0	94	2	0	0.019
100	1	94	3	0	0.005
100	0	97	3	0	-0.007
99	0	94	2	0	0.012
102	2	98	2	0	0.018
101	1	96	2	0	0.020
102	3	97	2	0	0.011
97	2	90	3	0	0.013
95	2	90	0	0	0.008
94	2	90	0	0	0.007
103	0	97	0	0	0.004
97	-1	92	0	0	0.002
99	-1	95	0	0	-0.003
96	0	91	0	0	-0.006
100	-1	96	0	0	0.012
101	-1	97	0	0	-0.005
94	0	90	0	0	-0.006

C.1.4 Linear movement, 0.4m/s, 2m

dx_meas	dy_meas	dx_odom	dy_odom	dz_odom	dyaw_odom
204	3	194	-1	0	0.022
205	3	194	-2	0	-0.010
206	8	196	0	0	0.006
202	5	192	-1	0	0.012
205	-2	195	-1	0	0.012
206	3	197	2	0	0.004
199	2	190	2	0	0.012
198	3	190	2	0	0.000
200	0	190	2	0	-0.002
205	5	196	2	0	0.009
200	6	190	3	0	0.005
206	4	196	2	0	0.020
205	3	196	4	0	0.015
198	6	190	3	0	0.008
207	3	197	2	0	0.024
207	3	197	2	0	0.021
205	5	195	3	0	0.004
207	-2	198	3	0	-0.009
207	4	195	2	0	0.013
206	-1	196	4	0	0.004

C.1.5 Linear movement, 0.6m/s, 2m

dx_meas	dy_meas	dx_odom	dy_odom	dz_odom	dyaw_odom
197	0	187	-4	0	0.003
193	1	179	-7	0	0.006
203	5	190	-10	0	0.002
200	5	188	-15	0	0.005
202	3	189	-20	0	0.003
201	0	188	-23	0	0.003
194	0	180	-25	0	0.004
192	4	178	-30	0	0.003
198	2	186	3	0	-0.001
193	-2	181	0	0	0.007
201	0	189	-3	0	-0.003
193	-2	181	-8	0	-0.002
204	2	191	-13	0	-0.007
201	0	188	-14	0	-0.002
202	1	189	-17	0	0.006
203	0	190	-18	0	-0.001
192	-1	180	-21	0	0.003
192	2	180	-24	0	-0.002
193	5	180	-29	0	0.003
195	7	180	-34	0	0.004
200	5	184	-38	0	-0.001
204	3	186	-43	0	0.006
201	1	184	-45	0	-0.002

C.2 Rotational movement experiment

C.2.1 Rotational movement, 45deg/s, 45deg

dx1_meas	dy1_meas	dx2_meas	dy2_meas	dx_odom	dy_odom	dz_odom	dyaw_odom
1	-3	-2	1	0	0	0	0.683
1	-3	-2	1	0	0	0	0.667
2	-4	-2	4	0	0	0	0.596
2	-6	-5	5	0	0	0	0.554
2	-4	-3	3	0	0	0	0.648
0	-3	-1	2	0	0	0	0.711
0	-2	-2	2	0	0	0	0.726
0	-3	-2	1	0	0	0	0.708
2	-4	-3	3	0	0	0	0.633
2	-3	-2	2	0	0	0	0.656
1	-5	-3	1	0	0	0	0.665
1	-4	-2	1	0	-1	0	0.690
3	-7	-4	4	0	0	0	0.553
2	-3	-2	3	0	0	0	0.663
2	0	-1	1	0	0	0	0.714
2	-4	-2	2	0	0	0	0.674
3	-6	-2	5	0	0	0	0.559
3	-4	-3	4	0	0	0	0.549
2	-4	-2	3	0	0	0	0.665
2	-4	-2	2	0	0	0	0.666

C.2.2 Rotational movement, 45deg/s, 90deg

dx1_meas	dy1_meas	dx2_meas	dy2_meas	dx_odom	dy_odom	dz_odom	dyaw_odom
3	0	-3	0	0	0	0	1.407
2	0	-2	0	1	0	0	1.439
0	1	-1	0	0	0	0	1.476
3	0	-4	1	0	0	0	1.411
1	0	-1	0	0	1	0	1.479
-1	0	-1	0	0	0	0	1.482
0	1	-4	1	0	0	0	1.460
0	1	0	1	0	0	0	1.487
1	1	-5	1	0	0	0	1.421
1	1	-5	1	0	0	0	1.423
3	0	-3	0	0	0	0	1.464
1	2	-3	1	0	0	0	1.475
4	1	-5	1	0	0	0	1.403
1	1	-3	1	0	0	0	1.474
3	2	-3	2	0	0	0	1.418
2	1	-3	1	0	0	0	1.458
4	1	-6	1	0	0	0	1.368
3	2	-4	1	0	0	0	1.438
6	0	-4	1	-1	0	0	1.363
2	1	-2	1	0	1	0	1.469

C.2.3 Rotational movement, 45deg/s, 180deg

dx1_meas	dy1_meas	dx2_meas	dy2_meas	dx_odom	dy_odom	dz_odom	dyaw_odom
1	1	1	2	-1	0	0	3.042
1	1	1	-1	-1	1	0	2.994
0	1	0	2	-1	0	0	3.020
1	2	1	2	-1	1	0	3.041
0	-1	0	0	0	0	0	3.060
0	1	0	-3	-1	0	0	2.967
0	2	0	-3	0	1	0	2.949
0	0	1	1	0	0	0	3.047
1	4	3	-5	0	1	0	2.918
0	2	1	-2	0	1	0	2.996
1	-3	1	0	1	1	0	3.302
0	0	0	0	0	0	0	3.022
0	0	0	0	0	0	0	3.040
0	0	1	1	1	0	0	3.047
0	3	0	-3	1	0	0	2.980
1	3	1	-3	1	0	0	2.955
1	0	0	0	1	0	0	3.041
1	4	1	4	0	0	0	2.909
0	0	0	0	0	0	0	3.026
0	3	0	-3	0	0	0	2.934

C.2.4 Rotational movement, 45deg/s, 360deg

dx1_meas	dy1_meas	dx2_meas	dy2_meas	dx_odom	dy_odom	dz_odom	dyaw_odom
2	5	1	-4	0	0	0	-0.120
1	4	1	-4	0	-1	0	-0.133
1	5	1	-5	0	-1	0	-0.092
1	6	1	-5	0	-1	0	-0.093
0	5	1	-5	0	0	0	-0.055
1	6	1	-6	0	0	0	-0.086
0	0	0	0	0	-1	0	-0.213
0	5	2	-5	-1	0	0	-0.092
1	5	1	-5	-1	0	0	-0.103
0	2	1	0	1	-1	0	-0.196
1	1	2	0	-1	0	0	-0.195
0	4	1	-4	-1	0	0	-0.101
0	4	1	-3	-1	0	0	-0.133
1	5	1	-5	-1	0	0	-0.093
1	5	1	-5	-1	0	0	-0.089
1	1	1	1	-1	0	0	-0.220
1	5	1	-5	-1	0	0	-0.101
0	5	1	-6	0	-1	0	-0.044
1	4	1	-4	0	-1	0	-0.013
0	5	1	-4	0	-1	0	-0.111

C.2.5 Rotational movement, 75deg/s, 45deg

dx1_meas	dy1_meas	dx2_meas	dy2_meas	dx_odom	dy_odom	dz_odom	dyaw_odom
3	-5	-3	3	0	0	0	0.630
6	-10	-6	10	0	0	0	0.370
4	-6	-4	5	0	0	0	0.551
7	-10	-5	8	0	0	0	0.422
5	-10	-6	10	0	0	0	0.416
5	-6	-4	6	0	0	0	0.589
7	-11	-6	9	0	0	0	0.422
4	-7	-3	4	0	0	0	0.546
3	-3	-2	3	0	0	0	0.642
5	-9	-5	8	0	0	0	0.443
3	-5	-4	4	0	0	0	0.590
6	-10	-6	9	0	0	0	0.393
3	-7	-3	5	0	0	0	0.556
3	-5	-2	3	0	0	0	0.609
3	-5	-2	2	0	0	0	0.623
2	-4	-3	3	0	0	0	0.594
3	-6	-4	3	-1	0	0	0.585
2	-3	-2	3	0	0	0	0.650
2	-3	-2	3	-1	0	0	0.623
2	-3	-2	3	-1	0	0	0.658

C.2.6 Rotational movement, 75deg/s, 90deg

dx1_meas	dy1_meas	dx2_meas	dy2_meas	dx_odom	dy_odom	dz_odom	dyaw_odom
4	0	-6	0	0	0	0	1.346
11	2	-10	1	0	-1	0	1.205
1	0	-2	0	0	-1	0	1.437
-1	-2	0	0	0	-1	0	1.460
0	-1	0	1	0	-1	0	1.445
0	6	1	-5	0	-1	0	1.341
4	-1	-3	0	-1	-1	0	1.415
2	0	-2	0	0	-1	0	1.464
3	1	-3	1	0	0	0	1.426
4	1	-4	1	0	-1	0	1.394
10	1	-10	2	0	0	0	1.243
4	0	-2	1	-1	-1	0	1.422
3	0	0	0	0	-1	0	1.477
5	-1	-3	1	1	0	0	1.386
5	0	-4	0	0	-1	0	1.377
7	1	-8	2	1	0	0	1.287
10	-1	-8	2	0	-1	0	1.211
3	0	0	1	1	-1	0	1.463
5	0	-2	1	0	-1	0	1.432
7	-1	-2	0	0	-1	0	1.358

C.2.7 Rotational movement, 75deg/s, 180deg

dx1_meas	dy1_meas	dx2_meas	dy2_meas	dx_odom	dy_odom	dz_odom	dyaw_odom
1	8	3	-8	1	0	0	2.769
0	1	0	-1	0	0	0	2.991
0	1	0	-1	0	0	0	2.999
1	0	1	0	1	-1	0	3.005
0	1	0	1	0	-1	0	3.001
0	1	1	1	0	-1	0	2.988
0	0	1	1	0	-1	0	3.028
1	2	1	-2	-1	0	0	2.972
1	3	1	-3	-1	0	0	2.950
3	7	3	-7	-1	0	0	2.810
2	10	3	-7	-1	0	0	2.753
1	1	1	-1	0	1	0	2.951
1	4	1	-2	0	1	0	2.959
1	1	0	0	1	1	0	2.997
3	7	0	5	1	0	0	2.825
1	7	3	-7	-1	0	0	2.794
1	2	1	-2	-1	0	0	2.967
2	2	2	2	-1	1	0	3.030
2	5	1	-3	-1	1	0	2.926
2	2	1	-3	0	1	0	2.961
3	10	1	-8	1	1	0	2.771

C.2.8 Rotational movement, 75deg/s, 360deg

dx1__meas	dy1__meas	dx2__meas	dy2__meas	dx__odom	dy__odom	dz__odom	dyaw__odom
1	-5	2	6	1	0	0	-0.394
1	3	1	-2	1	0	0	-0.167
0	3	1	0	1	1	0	-0.203
1	3	3	7	1	0	0	-0.370
1	5	2	-2	1	-1	0	-0.136
0	5	1	-3	1	-1	0	-0.122
1	-1	1	3	1	-1	0	-0.305
1	4	2	-4	0	-1	0	0.141
1	-2	1	4	0	-1	0	-0.321
2	2	2	-2	-1	1	0	-0.191
1	2	2	0	0	-1	0	-0.213
1	2	1	-2	-1	-1	0	-0.184
2	3	3	-2	-1	-1	0	-0.184
1	-4	1	4	-1	0	0	-0.351
2	3	2	-3	-1	0	0	-0.164
2	3	3	-3	-1	0	0	-0.177
1	-4	1	4	-1	1	0	-0.358
2	4	2	-4	-1	1	0	-0.132
1	5	2	-4	-1	1	0	-0.121
1	3	2	-2	-1	1	0	-0.170