KUKA YouBot Revival and System Development Project

Documentation of progress through project, in requirements of Final Project in DSD

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

May 29, 2025

Contents

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Project Overview . . . 1

2 Initial state of YouBot **2**
2.1 About the Hardware . . . 2
2.2 Robot Base Overview . . . 2
2.3 Additional Hardware Components . . . 4
2.4 Onboard Computer Overview . . . 4
2.5 Battery . . . 4
2.6 Sensors . . . 5
2.6.1 Kinect . . . 5
2.6.2 Hokuyo URG-04LX Laser Rangefinder . . . 6
2.7 Documentation . . . 6
2.8 Initial Project Goals . . . 6

3 Project Overview **9**
3.1 Technical Goals . . . 9
3.2 Learning Objectives . . . 9
3.3 System Architecture . . . 10

4 System Development **11**
4.1 Booting up the YouBot . . . 11
4.2 Running original demos . . . 12
4.3 Enabling wireless connections . . . 12
4.4 Running ROS interface . . . 13
4.5 ROS1-ROS2 Bridge . . . 14
4.6 Writing Custom Programs . . . 15
4.7 Sensors . . . 15
4.7.1 Kinect Camera and libfreenect . . . 15
4.7.2 Hokuyo URG-04LX Laser Rangefinder . . . 15
4.8 Battery Replacement . . . 16
4.8.1 SLA Battery Overview . . . 16
4.8.2 Testing original batteries . . . 16
4.8.3 Replacing batteries . . . 17
4.8.4 Future Battery Options . . . 17
4.9 Experimental Setup for Movement and Odometry . . . 17

5 Future Applications **21**
5.1 Potential Use Cases for the YouBot Mobile Base . . . 21
5.2 Teach and Repeat (T&R) in Context . . . 22
5.3 Speculative and Emerging Applications . . . 22
5.4 Summary Table: Potential Use Cases for the KUKA YouBot Mobile Base . . . 23
5.5 Conclusion . . . 23

6 Reflection **24**
6.1 Project Evolution . . . 24

6.2 AGILE . . . 24
6.3 Final Thoughts . . . 24

A User Manual for the KUKA YouBot **26**
A.1 How to turn on the YouBot . . . 26
A.2 .bashrc recap . . . 26
A.3 How to run original demos . . . 26

B CSVs and code **26**
B.1 CSV files . . . 26
B.1.1 Linear movement, 0.2m/s, 1m . . . 26
B.1.2 Linear movement, 0.2m/s, 2m . . . 26
B.1.3 Linear movement, 0.4m/s, 1m . . . 27
B.1.4 Linear movement, 0.4m/s, 2m . . . 27
B.1.5 Linear movement, 0.6m/s, 2m . . . 28
B.1.6 Rotational movement, 45deg/s, 45deg . . . 28
B.1.7 Rotational movement, 45deg/s, 90deg . . . 29
B.1.8 Rotational movement, 45deg/s, 180deg . . . 29
B.1.9 Rotational movement, 45deg/s, 360deg . . . 30
B.1.10 Rotational movement, 75deg/s, 45deg . . . 30
B.1.11 Rotational movement, 75deg/s, 90deg . . . 31
B.1.12 Rotational movement, 75deg/s, 180deg . . . 31
B.1.13 Rotational movement, 75deg/s, 360deg . . . 32
B.2 Python Files . . . 33
B.2.1 rotate_graph.py . . . 33
B.2.2 linear_graph.py . . . 36

1 Introduction

1.1 Motivation

In the rapidly evolving field of robotics, expensive hardware platforms often become obsolete quickly. This is not because of physical wear and tear or a lack of functionality, but rather due to quick advancements in software and infrastructure. For example, artificial intelligence and machine learning have made significant strides in recent years, leading to the development of humanoid robots with advanced capabilities. As a result, older platforms become less relevant and lack the necessary support for extending their potential lifespan, despite inherent capabilities.

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. It can potentially serve a variety of purposes, including autonomous navigation, perception and human-robot interaction with modern tools and sensors. Instead of discarding such platforms, we should explore these avenues to extend their lifespan and utility.

Thus, this project aims to explore the potential of deploying and repurposting legacy robotic hardware like the KUKA YouBot within the current landscape of robotics. By doing so, we can demonstrate the potential of long-term viability of older platforms whilst allowing for smooth integration with other systems.

1.2 Project Overview

This project aims to document the reployment and quantitative measurement of the KUKA YouBot's capabilities, particularly in terms of navigation and odometry.

Section 2 provides an overview of the initial state of the YouBot, including its hardware and software components, as well as a brief description of the initial goals of the project. Section 3 outlines the technical and learning objectives of the project. Section 4 describes the system development process, including the booting up of the YouBot, running the original C++ demos and the ROS interface demos. This follows into custom program development and a experimental setup for testing movement and odometry. Additionally, there will be insights into battery replacement and sensor integration. Section 5 discusses future potential for the YouBot for long-term research and industrial use cases, and section 6 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 experiment results.

2 Initial state of YouBot

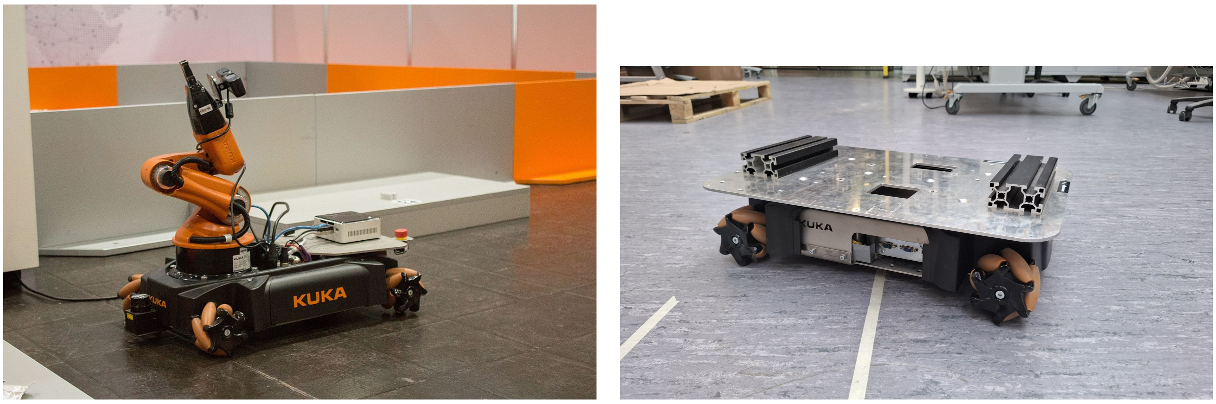
The KUKA YouBot was procured by the university in 2014, and has not been in use since 2019. At that time, the robot was considered to be quite antiquated, and the software and hardware were already outdated. Furthermore, the relevant websites hosting documentation and information are no longer available. This situation has only worsened over the years, and the starting point for this project through was an user manual uploaded on a third-party website marked as a draft version from 2013.

2.1 About the Hardware

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

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 [cite], 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 flexible 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 and perception, as well as human-robot interaction.



(a) The robot with the robotic arm attached.

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

Figure 1. The YouBot.

2.2 Robot Base Overview

The robot base features a 24V power input via a 3-pin XLR connector, accessed through the top of the robot. In addition to this, there are two 24V 3-pin XLR output ports, indeded to power external sensors or other components such as robotic arms. On this top panel, one can also find two EtherCAT ports for consistent real-time communication with motion-oriented robotics (i.e. the robotic arm), as well as a standard Ethernet port for wired interfacing with an external computer or network.

Mobility is provided by four omnidirectional mecanum wheels and relevant motors. This allows for relatively smooth and precise movement through a combination of linear and rotational motion. The motors are controlled via the onboard computer, which can send and recieve data from the motors and provide relevant feedback.

The onboard computer includes 6 USB2.0 ports, which can be used to connect various peripherals, such as a keyboard, mouse, USB, wireless adapter, etc. Additionally, there is a VGA port for connecting a monitor. [1]

The attached top panel is a sensor and mounting plate, designed to allow for the convenient attachment of various sensors and accessories. [cite https://web.archive.org/web/20160613151621/http://
store.com/accessories/mounting-and-sensor-plate]

The robot base is also equipped with a slot for a sealed lead-acid battery (SLA), which can be used to power the robot when not connected to a wall outlet. The battery can connect to the robot via a 4-pin XLR connector.

(a) The power input and output ports.

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

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

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

Figure 2. Connection points on the YouBot.

2.3 Additional Hardware Components

DESCRIBE THE INITIAL INSPECTION IN FULL HERE! TALK ABOUT THE PILLAR STUFF AND NUTS AND WHATNOT AS WELL!!!!

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 pillars can be combined in a variety of ways to create a stable and secure mounting platform for the sensors.

Figure 3. Centered image

2.4 Onboard Computer Overview

The computer runs [this software] and ROS version []. I wasn't able to immediately boot up any tests to ensure if the robot itself was working at any stage in time.

The onboard computer is an [] with an [].

It was running Ubuntu 12.04 LTS with ROS Hydro, which is a decade-old version of the operating system and the robot operating system. This version of Ubuntu is no longer supported, and the software is outdated and not compatible with most modern software and libraries. This will be detailed in a future section.

Alongside the OS and ROS, the computer also had various drivers and wrappers installed to enable communication with the robot's motors and sensors. This could be done directly through C++ programs, or through the use of pre-made ROS packages. We have elected to use the latter for the purposes of developing a system on the YouBot, to allow for a seamless integration with the ROS ecosystem in the future, alongside a level of standardization and ease of use.

2.5 Battery

The mobile robot base allows for a 24V power supply through a 4-pin XLR connector. It could also be operated without using a wired cable to the wall, by using a lead-acid battery originally which had a capacity of 5Ah and gave the YouBot an approximate runtime of 90 minutes. This lead-acid battery is of dimensions 23cmx9.4cmx7.3cm. This battery is also connected through a 4-pin XLR connector. While not a standard connector for power supply (4-pin XLR connectors are typically used for audio equipment), it is very robust and secure for the purposes of a mobile robot.

However, the SLA batteries that were initially included with the YouBot were completely unusable. The lab engineer (Vesa Korhonen) had also put together a makeshift SLA battery in 2019, which had also degraded severely.

As such, either the original batteries had to be replaced, or the robot had to be continuously used with a wired connection to the wall. This would have been a significant limitation to the project had we not been able to create a makeshift replacement battery which will be discussed in section 4.4.

(a) The original SLA battery.

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

Figure 4. Power supply components for the YouBot.

2.6 Sensors

Alongside the robot base and various hardware components, the YouBot was also equipped with a variety of sensors. These included a Kinect v1 camera and two Hokuyo URG-04LX laser rangefinders (, one of which had a broken mini-USB port, rendering it unusable). These sensors are commonly used in robotics research and are well-suited for navigation and perception tasks.

2.6.1 Kinect

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

This Kinect camera module can be used through the use of open-source software such as libfreenect [cite] to access the data from the various sensors on the module. Furthermore, this information could be processed using the OpenCV library and then used to enable the autonomous navigation as previously described.

Figure 5. Centered image

2.6.2 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.

Figure 6. Centered image

2.7 Documentation

The documentation for the KUKA YouBot is sparse and unavailable for the most part. KUKA has removed most references to the product from their website, and the original youbot-store.com website is no longer available. Furthermore, the only available user manual for the YouBot online is a decade-old PDF file that was uploaded by a third-party user, and is marked as a draft version from 2013.

Thankfully, the YouBot was marketed as an open-source platform, and the drivers as well as various parts of software are available on GitHub. Furthermore, we were able to find some more information about the YouBot through the Wayback Machine, which had archived parts of the original website. While this archive is not complete by any means and is very tedious to comb through, it provides us with a powerful tool to piece together data about the YouBot and its use cases during its hayday.

We've also taken the liberty of citing some YouBot related projects here.

add images of some of the more interesting websites here, and some of the images regarding the youbot's statistics.

2.8 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.

Teach & Repeat (T&R) was considered to be an ideal candidate to allow for the autonomous navigation. This[cite] 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 [cite]. Much of the work done on extending T&R aims to improve this robustness and scalability for a variety of sensors and use-cases.

I'm not sure how the ROS stack would look like, or if its necessary to explain. Discuss with advisor

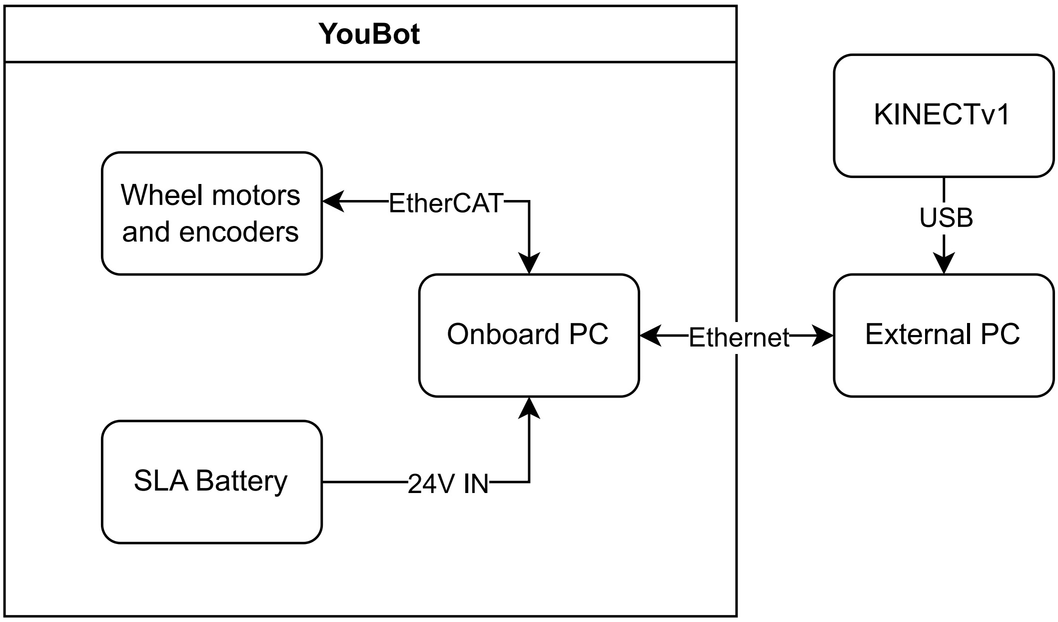


Figure 7. System Diagram

Here, figure [] shows an example of how the architecture for this system would have been implemented. This would have involved a wired connection to an external computer running ROS2 Jazzy and directly connected to the Kinect camera mounted on top of the YouBot. The onboard computer would have been used to control the motors and communicate with the sensors, while the external computer would have handled the processing and navigation tasks. The onboard computer and external computer would commuicate through a ROS1-ROS2 bridge, allowing for seamless integration of the two systems.

However, considering the age of the YouBot, lack of personal knowledge regarding robotics, as well as incompatible software, it was realised that this was not feasible. We will be discussing our new goals in the next 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

The learning objectives for this project involve the following:

- Develop hands-on experience in restoring and operating a legacy robot platform (KUKA YouBot)
- Gain proficiency in Linux systems and environment setup for robotics development
- Learn to write and use Bash scripts to automate setup, build, and deployment workflows
- Understand the architecture of the Robot Operating System (ROS1 & ROS2), and learn to configure and launch ROS-based systems
- Learn to establish and manage secure communication with robotic systems using SSH and Ethernet
- Write and deploy custom ROS nodes for basic motion control of the YouBot platform
- Understand the principles of robot navigation and odometry, including error measurement and correction techniques
- Learn methods for documenting system restoration, experimental results, and identifying directions for future development

3.3 System Architecture

Our intended setup was to use the YouBot as a mobile base with the Kinect camera mounted on top of it. The onboard computer of the YouBot would only be used to control the motors and communicate with the sensors. The actual bulk of the processing would be done on a seperate computer that runs ROS2 and enables the necessary navigation and perception stacks.

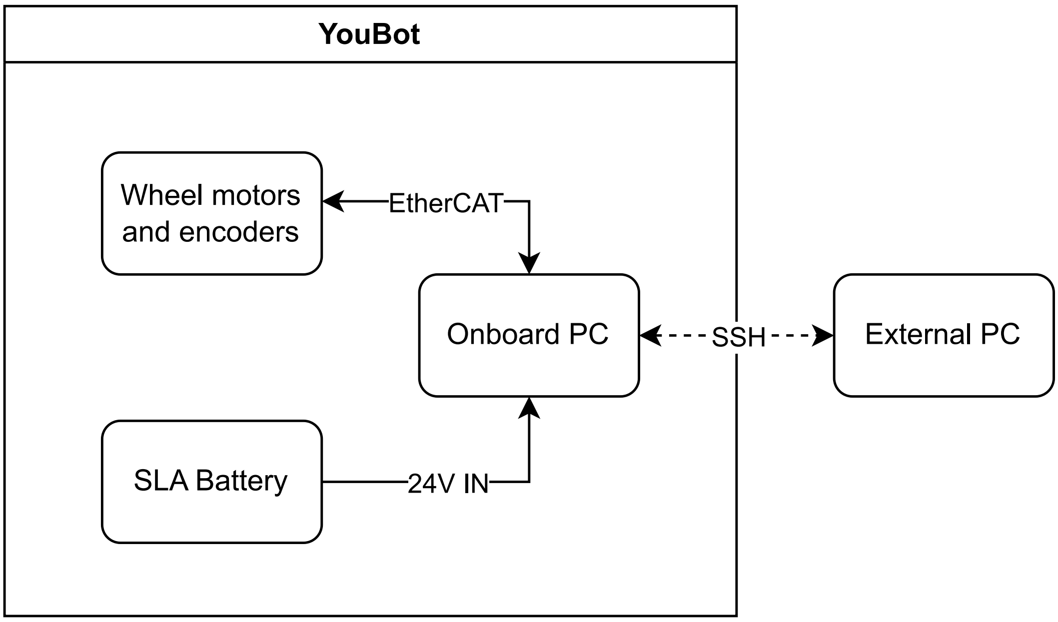


Figure 8. New System Diagram

4 System Development

4.1 Booting up the YouBot

describe the booting up process here, what ports to use, how to connect to the onboard computer, etc.

furthermore describe issues with software that needed to be fixed, such as the bashrc file, the networking, etc.

The initial inspection was carried out as described in section 2. Since the onboard battery was completely dead, I connected the YouBot to a 24V power supply and booted it up. 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.

After connecting a VGA monitor and keyboard and mouse, I was able to boot up the onboard PC and use it as any other computer. The wireless adapter connected to this onboard computer was also functional, and I was able to connect to the internet. This meant I could theoretically SSH into the computer from my personal laptop, but I was unable to do so at that time due to a lack of a static IP address.

I was able to see the following software installed on the onboard computer: insert ROS version here and discuss the github repos that were already installed here.

There's a lot of files from at least 6 years ago on this, and I believe a significant portion are from like 2013 or 14. Two folders that look especially important are ros_stacks and youbot_driver. One file of high importance is in the desktop: the Kuka YouBot User Manual. I took a copy of that to my personal computer using a thumbdrive.

discuss powering up, initial software inspection, static IP issue

4.2 Running original demos

describe fixing network issue, and hten running the original demos.

An attempt was made to run the original demos upon turning on the computer. These original demos were stored here: [insert path here]. However, the program was unable to run due to not detecting any devices on the eth1 port. This was assumedly due to the motor drivers not being initialised, or the wrong port being used. The user manual states eth0, but the onboard computer was using eth1. This was thus a simple fix of changing the port in the config file for the youbot_driver.

After this issue was fixed, you can turn the motors on using that button on the screen. running the demo using the following command then allows you to access this thing through the terminal. it works to do shit.

4.3 Enabling wireless connections

The wireless issues was caused by a combination of no IPv4 address assigned to the onboard computer, and the onboard computer not being able to connect to the internet. This was fixed by changing the network settings in the Network Manager, and setting a static IP address for the onboard computer. This was done by editing the /etc/network/interfaces file and adding the following lines:

```
auto eth1
iface eth1 inet static
    address
    netmask
    gateway
```

Initially, the following commands were used to set the static IP address:

```
sudo ifconfig eth1
```

however, this would automatically delete itself inconsistently due to the Network Manager. This was thus changed to the above method, which would set the static IP address permanently. This is much more convenient long term.

One can then connect to

4.4 Running ROS interface

so what packages were there?

describe the process of inspecting the installed software, the drivers and wrappers, and running the original demos. rqt, rviz, the various topics and nodes, etc. Finding the packages running these packages

I had to find these packages by running history and then going through the bash history. I was able to find some relevant packages and then use them to run my shit. I also found some youbot_driver to do this but i found it more convenient to do sudo bash this another youbot_Driver because it was visible much more easily on the home system.

The demos through ROS allowed me to inspect many things within ROS as well, like the nodes,

list nodes here and their purposes.

the topics,

list that here,

discuss rqt and rviz here

4.5 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.

4.6 Writing Custom Programs

Using the ROS demo packages as a reference, I was able to write my own custom programs to control the YouBot. This involved using the ROS packages and drivers on the onboard computer to send commands to the motors through the cmd_vel topic. While primitive, this allows for a simmple and intuitive way to control the robot's movement. This could be further extended to include more complex movement through python scripts and other ROS nodes and packages.

To do this, first create a new package using the catkin command. This will create a new folder in the src directory of your catkin workspace. You can then create a new python script within this folder and use the ROS packages and drivers to send commands to the motors.

remember to either source the catkin workspace everytime in a new terminal, or add the source command to your .bashrc file.

After doing this, you can create a new python script within this folder and use the ROS packages and drivers to send commands to the motors.

```
Text enclosed inside \texttt{verbatim} environment
is printed directly
and all \LaTeX{} commands are ignored.
```

4.7 Sensors

4.7.1 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.

4.7.2 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.

4.8 Battery Replacement

draft 1, 11.05 12pm
Here are some good web pages about the Slealed Lead Acid (SLA) batteries: https://batterymasters.co.uk/
https://www.power-sonic.com/blog/how-to-charge-a-lead-acid-battery/ and https://www.powerstream.co

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.

4.8.1 SLA Battery Overview

Sealed lead-acid (SLA) batteries operate based on a reversible chemical reaction between lead plates, lead dioxide and sulfuric acid electrolyte. 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. 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.

4.8.2 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.

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.

4.8.3 Replacing batteries

maybe here we can link to the specific batteries used? the dimensions would be neat as well...

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 maybe mention the pinout here?, 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.

4.8.4 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. 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.

4.9 Experimental Setup for Movement and Odometry

here, ill discuss the experimental setup for testing movement and odometry. this should include the two experiments of moving linearly and rotating, and discuss how error is calculated. probably discuss negative feedback and PID control as well, random, percentage, consistent error, etc.

Taking into account the various inconsistencies noticed whilst running custom programs, some experiments were set up to test the movement and odometry of the YouBot. Since this error seems to be a combination of the robot's own motor control as well as the odometry falling out of calibration, it is important to quantify this error and understand how it may be corrected.

different speeds? proportional or fixed error? does the odometry also drift?

20 times per experiment

Taking into account the various inconsistencies observed while running custom movement programs on the YouBot platform, a series of experiments were conducted to evaluate the accuracy of its movement and odometry. The goal was to quantify the types and sources of errors that arise during both linear and rotational motion, and to assess how these errors can be corrected or compensated for.

Experimental Procedure

Two main experiments were designed: one for testing linear forward motion and the other for in-place rotation. In the linear movement test, the robot was commanded to travel forward by a set distance, while in the rotation test, the robot was commanded to rotate by a fixed angle (e.g., 90°). Each command was executed at different speed settings to observe whether error behavior varied with velocity.

Each experiment was repeated 20 times to ensure statistical significance and to detect any patterns of consistency or randomness in the errors. Measurements of the actual displacement and orientation were obtained using external references (e.g., floor markings or video tracking), and compared against the internal odometry values reported by the robot.

Experiment Configurations

Test ID	Type	Speed (m/s or rad/s)	Target (m or deg)
L1	Linear	0.2 m/s	1.0 m
L2	Linear	0.4 m/s	1.0 m
L3	Linear	0.2 m/s	0.5 m
R1	Rotation	0.5 rad/s	90°
R2	Rotation	1.0 rad/s	90°

Table 1. Summary of movement and odometry test configurations

Each test case involved issuing a velocity command for a precise duration such that the robot would cover the target distance or angle, assuming ideal conditions. The difference between the expected and actual outcomes was used to calculate both positional and orientation errors.

Types of Errors Observed

During the experiments, several types of errors were encountered:

- **Systematic Errors:** These occurred consistently in the same direction or magnitude, such as under-rotation by a fixed angle or consistent overshoot in distance. Causes included calibration mismatches, wheel radius misestimation, or delay in motor response.
- **Random Errors:** Small variations due to factors like sensor noise, surface friction inconsistencies, or transient slippage. These were not consistent across trials and contributed to spread in the results.

- **Proportional Errors:** Errors that scaled with speed or distance, such as increasing overshoot at higher speeds. This suggested control loop latency or imperfect motor linearity.
- **Fixed Offsets:** Some errors were fixed and independent of input command, like a consistent 2 cm drift to one side. These often pointed to physical asymmetries or encoder misalignments.
- **Odometry Drift:** Over longer trials or repeated motion without reset, odometry drift became more apparent, particularly in rotational tracking.

Correction Strategies

To address the errors above, multiple correction strategies were explored:

- **PID Control:** Implementing a PID loop helped reduce overshoot and improve convergence during both linear and rotational motion. Negative feedback from odometry or external position estimates was used to dynamically adjust motor outputs.
- **Calibration:** Careful calibration of wheel radius, wheelbase, and encoder resolution helped reduce systematic errors.
- **Compensation Factors:** Where errors were consistent and repeatable, simple scalar correction factors were applied post hoc to odometry readings.
- **Sensor Fusion:** Although not implemented in this basic test, more advanced setups could integrate IMUs or vision-based SLAM for odometry correction and drift reduction.
- **Averaging Trials:** For random noise, statistical averaging over multiple trials helped establish a more reliable error profile.

These experiments form the foundation for understanding the motion characteristics of the YouBot and inform future work in navigation, localization, and control refinement.

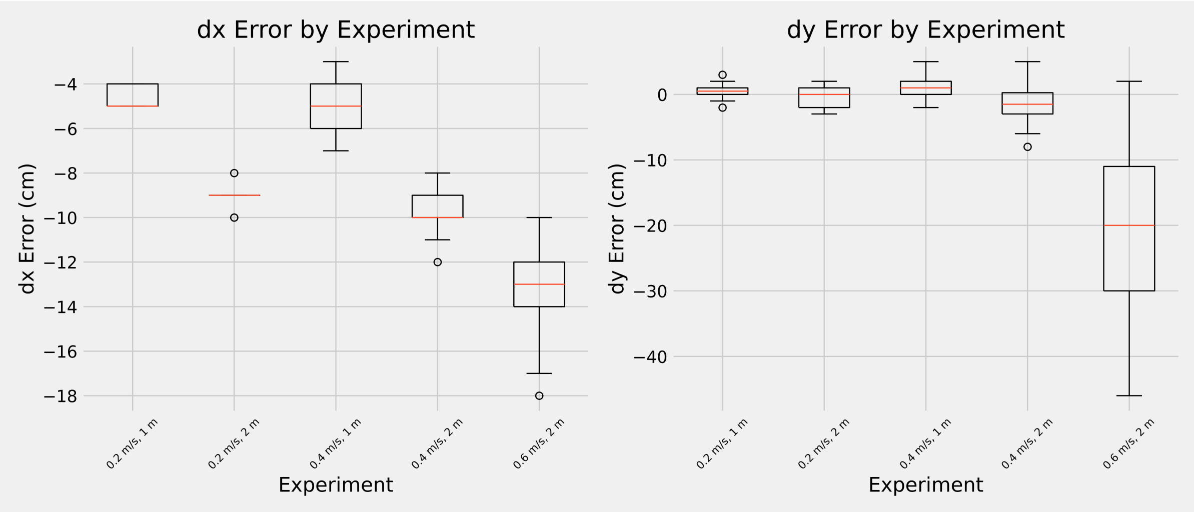


Figure 9. Centered image

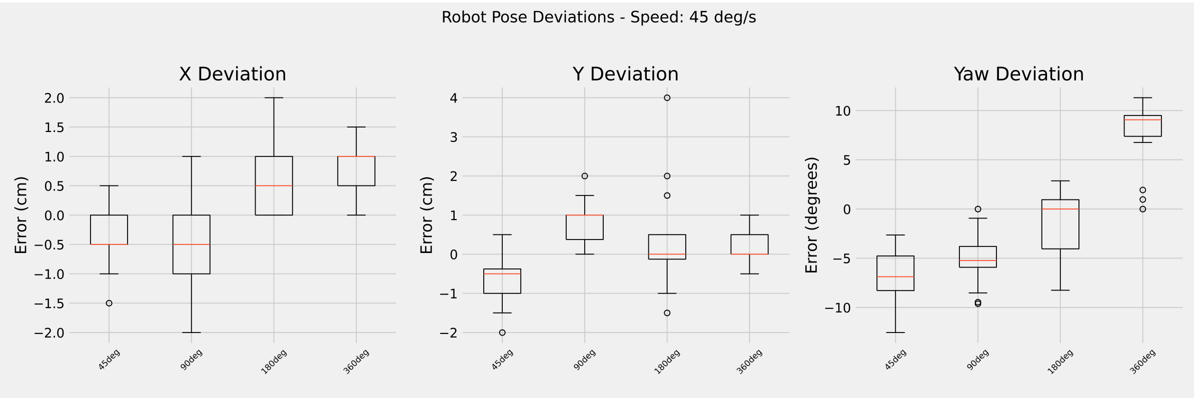


Figure 10. Centered image

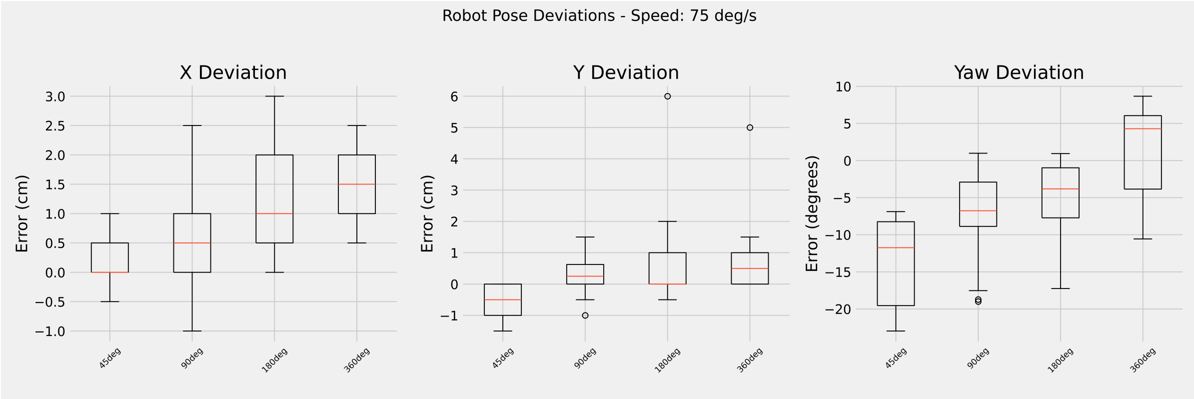


Figure 11. Centered image

5 Future Applications

In this section we will discuss a potential future application of the KUKA YouBot, in the context of the T&R method of navigation. The T&R method is a form of visual SLAM, where the robot can be "taught" a path through a static environment using a camera. The robot then uses this information to enable it to repeat the path movement autonomously through key frames it has saved from the environment. This method is simple yet effective in its use cases and can be used in a variety of applications, such as industrial automation, home robotics, service, etc.

t&r is not the best implementation of the youbot

platform and real world application

why im a bit skeptical about trying to implement t&r in the context of assistive platform trying to present as an application as a playform that can later on be used to execute repetitive task based on the experience of a platform with

speculative : presenting it in terms of real-world applications

flexible intra-logistics human awareness and flexibility seamless human-robot existence

While the YouBot holds the potential capabilities for undertaking Teach and Repeat, its abilities as a mobile industrial robot base would be better suited for real-world applications within closed environments as an assistive platform.

This robot base would be most useful in an assisstie sense, where it assists humans in completing rather basic and repetitive tasks, like fetching beer.

The KUKA YouBot, with its omnidirectional mobile base, offers a flexible platform for experimentation and prototyping in robotics. While the Teach and Repeat (T&R) method—where a robot is “taught” a path using visual cues and then autonomously repeats it—has proven effective in certain domains, its real-world utility as a navigation strategy for assistive platforms is nuanced and context-dependent.

5.1 Potential Use Cases for the YouBot Mobile Base

The YouBot’s mobile base, independent of its manipulator arm, is particularly well-suited for:

- **Flexible Intra-Logistics:** The YouBot can transport materials, components, or finished goods within factories or laboratories, adapting to dynamic layouts and avoiding obstacles thanks to its omnidirectional drive system. This makes it valuable for automating repetitive transport tasks in manufacturing, electronics, and pharmaceutical environments .
- **Assistive Service Robotics:** In assistive roles, the mobile base could help with tasks such as fetching objects (e.g., drinks, tools), delivering supplies, or guiding people in structured indoor environments. Such applications are particularly relevant in healthcare, offices, or smart homes, where repetitive, low-complexity tasks can be offloaded from humans to robots .
- **Research and Prototyping Platform:** The YouBot is widely used in academic and industrial research for developing and testing navigation, mapping, and multi-robot coordination algorithms. Its open interfaces and modular design make it ideal for experimentation in logistics, navigation, and human-robot interaction .
- **Quality Control and Inspection:** The platform can be equipped with sensors or cameras to autonomously inspect products, monitor environments, or perform routine checks in hazardous or hard-to-reach areas .

5.2 Teach and Repeat (T&R) in Context

The T&R method, while simple and robust for repeated path following in static environments, has limitations when applied to assistive platforms:

- **Limited Flexibility:** T&R excels in environments where the path and surroundings are relatively unchanging. In dynamic or human-populated spaces, its lack of real-time adaptability can be a drawback compared to more advanced SLAM or AI-based navigation systems.
- **Suitability for Repetitive Tasks:** For repetitive, well-defined routes—such as material delivery within a warehouse, or routine inspection rounds—T&R can be a practical, low-compute solution. However, its application as a general-purpose assistive platform is more speculative, especially where human interaction and environmental variability are significant.
- **Platform Capabilities:** The YouBot's mobile base is robust and maneuverable, making it a strong candidate for T&R-based applications in controlled settings. Its open architecture allows easy integration of additional sensors or navigation modules, enabling researchers to extend beyond T&R as needed.

5.3 Speculative and Emerging Applications

Looking forward, the YouBot mobile base could play a role in:

- **Seamless Human-Robot Coexistence:** As part of smart environments, YouBots could support flexible logistics, on-demand delivery, and collaborative tasks alongside humans, leveraging both T&R for routine paths and more advanced navigation for dynamic tasks.

- **Education and Training:** Its accessibility and open-source support make it an excellent tool for teaching robotics concepts, prototyping new algorithms, or participating in competitions such as RoboCup@Work.
- **Translational Research:** Innovations and algorithms developed on the YouBot can be scaled up to larger, industrial-grade mobile platforms, facilitating technology transfer from the lab to real-world production lines.

5.4 Summary Table: Potential Use Cases for the KUKA YouBot Mobile Base

Application Area	Description	Reference
Intra-logistics	Material transport, flexible routing in factories, labs, or warehouses	
Assistive service robotics	Fetching/delivery tasks, guidance, support in healthcare or home environments	
Research and prototyping	Testing navigation, SLAM, multi-robot systems, and human-robot interaction	
Inspection and quality control	Autonomous inspection, monitoring, and data collection in industrial or hazardous settings	
Education and training	Robotics teaching, student projects, and competitions	

Table 2. Potential Use Cases for the KUKA YouBot Mobile Base

5.5 Conclusion

While the KUKA YouBot's mobile base is not the optimal platform for all assistive applications—especially those requiring high flexibility and human awareness—it remains a valuable tool for developing and demonstrating repetitive, structured tasks. Its use in T&R navigation is best suited for closed, static environments, but its modularity and open architecture allow for broader experimentation and real-world prototyping in logistics, research, and assistive robotics.

6 Reflection

draft 1, 11.05 6pm

In this section, I will reflect on the outcomes of this project. This will include an overview of project management practices and planning, alongside my learnings and personal feelings about the project.

6.1 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. Furthermore, this project was a significant different experience from any other course I have taken in the past.

My initial approach to project management was very linear and straighforward, with the understanding that there would be no hiccups or issues along the way. I had the following time-table in mind: maybe insert a table showcasing the plan here? 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.

6.2 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.

6.3 Final Thoughts

Overall, despite the various challenges faced, I am very happy with my progress and learning outcomes regarding this 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, alongside the various hardware components of the robot. More importantly, I learned to manage projects with the understanding of personal and technical constraints. I'd like to thank my advisor for his provision of a different perspective on the project, explanations of everything and constant patience.

References

[1] "Youbot - Overview — github.com", https://github.com/youbot, [Accessed 23-04-2025] (page 3).

[2] I. Tsogias, "Kinematic Analysis of a KUKA YouBot", 2016. DOI: 10.26240/heal.ntua.12581. [Online]. Available: https://dspace.lib.ntua.gr/xmlui/handle/123456789/43358.

[3] "YouBot Store — web.archive.org", [Accessed 23-04-2025]. [Online]. Available: https://web.archive.org/web/20170318163736/http://www.youbot-store.com/developers/.

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

A User Manual for the KUKA YouBot

A.1 How to turn on the YouBot

A.2 .bashrc recap

A.3 How to run original demos

B CSVs and code

B.1 CSV files

B.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

B.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

B.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

B.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

B.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	00.003
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

B.1.6 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

B.1.7 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

B.1.8 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	3.3032
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

B.1.9 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

B.1.10 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

B.1.11 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

B.1.12 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

B.1.13 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

B.2 Python Files

B.2.1 rotate_graph.py

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

# === Config toggles ===
PRINT_COLUMN_DATA = True
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])
        yaw_diff = (yaw_rad - expected_yaw + np.pi) % (2 * np.pi) - np.pi

        x_err = center[0] - expected_center[0]
        y_err = center[1] - expected_center[1]

        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])

        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]
```

```python
        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.2.2 linear_graph.py

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

# === Config toggles ===
PRINT_COLUMN_DATA = True
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)
        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()

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