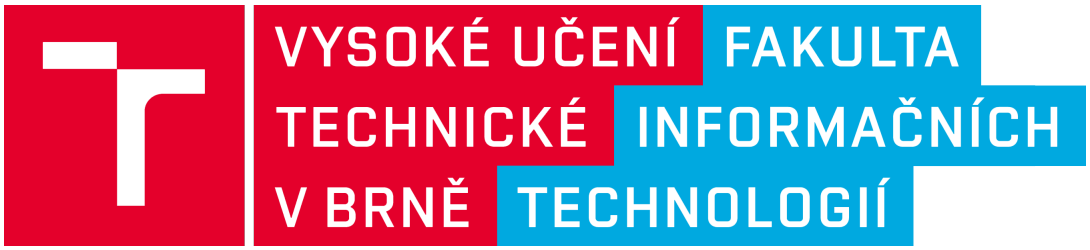
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



Path following robot with model in ROS and PID controller

January 30, 2024

Stanislav Jochman

Contents

1 Assignment

Design, create, and implement a PID controller for a line-following robot using the Robot Operating System (ROS). The goal is to create a robot that can autonomously follow a predefined path on the ground by tracking a black line using sensors.

1.1 Acceptance criteria

- **ROS environment setup** - section 3
- **Arduino enviroment** - section 4
- **Robot selection** - section 5
- **PID** - section 6
- **Implementation** - section 7

2 Introduction

ROS is an open-source framework that provides a comprehensive suite of tools, libraries, and software for designing, building, and controlling robots. It is fundamentally based on a concept called `nodes`. Nodes are individual software modules or processes that perform specific tasks within a robotic system. They represent a key architectural element in ROS, and the entire ROS ecosystem revolves around the concept of nodes. Individual software modules (nodes) perform specific tasks, communicate through publish-subscribe mechanisms, and can be distributed across multiple machines. This modular and distributed approach is one of the key strengths of ROS, making it a versatile framework for developing, testing, and operating robotic systems. This project uses Arduino Nano for controlling motors and reading data from sensors and Jetson Nano which runs the control algorithm. The topology of the robot and the connection between Arduino Nano and Jetson Nano is described in section 5. Topics used in this project are described in section 7.1.

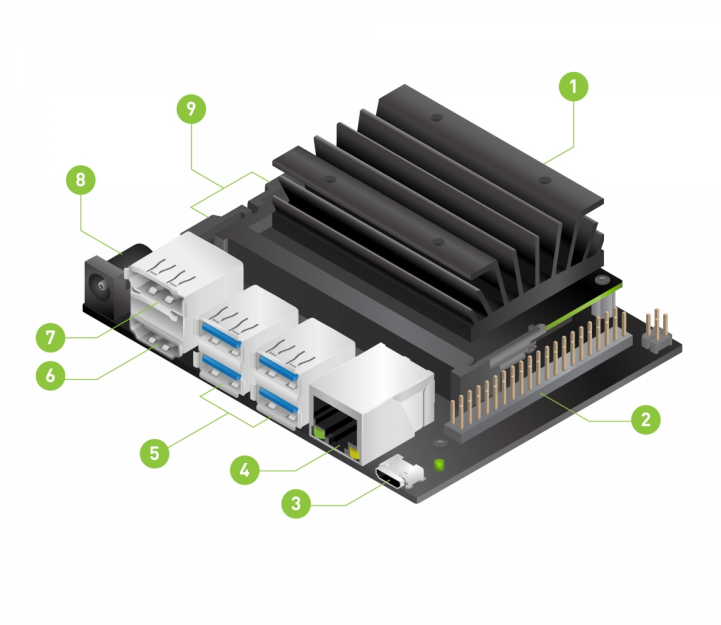


Figure 1: Jetson Nano - Small single board computer running Ubuntu with ROS

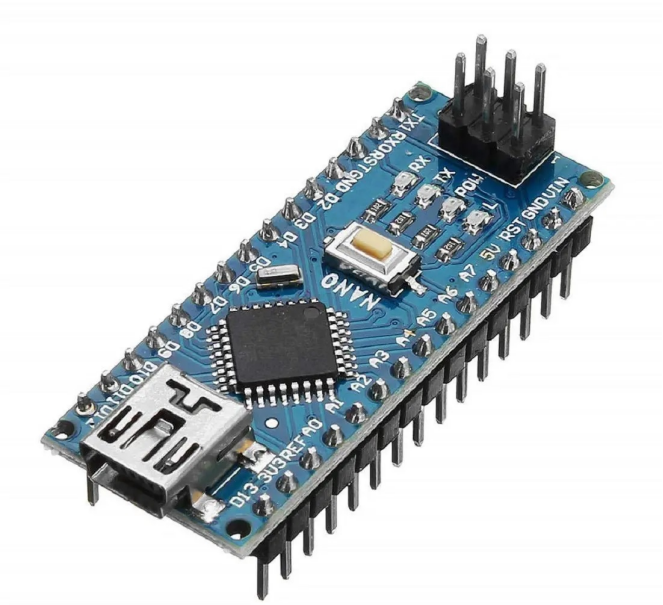


Figure 2: Arduino Nano - MCU controlling robot running ROSserial

3 ROS setup

The selected robot will use Arduino Nano, so I decided to install ROS1 for performance reasons. As a first step, it is necessary to install Ubuntu on Jetson Nano. Nvidia provides a detailed guide on how to install Ubuntu 18.04. It is important to install the correct version of Ubuntu because ROS versions are dependent on specific versions of the OS. After booting into Ubuntu we will proceed to ROS installation. First, we need to create `ros-melodic.list` and add the repo URL and OS meta info to that file[3].

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu bionic main" > /etc/apt/sources.list.d/ros-melodic.list'
```

After that, we need to add an apt-key.

```
sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
```

Finally, we can update the system and install ROS.

```
sudo apt update
sudo apt install ros-melodic-desktop-full
```

4 Arduino setup

For uploading code to Arduino we need to install Arduino IDE 2 and necessary libraries. As the next step, we need to install libraries. That can be done in `Sketch > Include Library > Manage Libraries`. After opening Library Manager we will search for `Rosserial Arduino Library` version `0.7.9`.

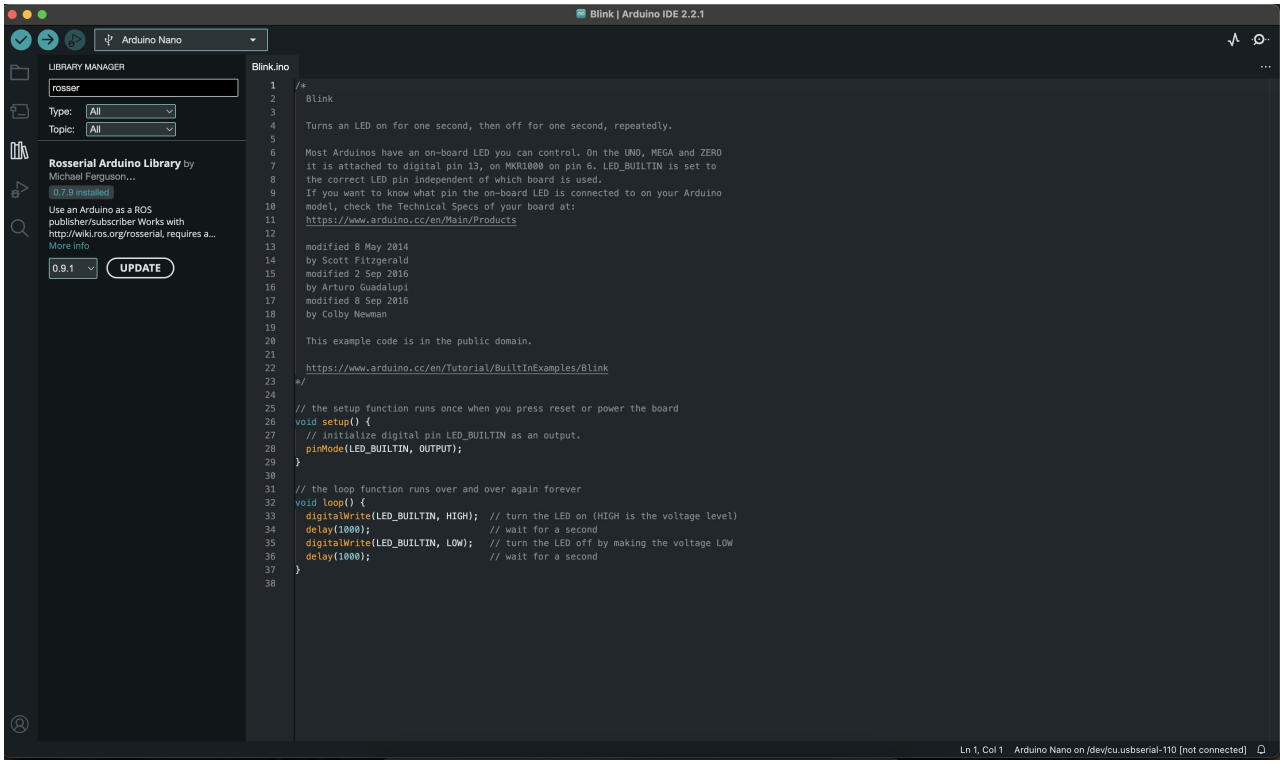


Figure 3: Arduino IDE 2 with correct library installed

5 Robot

The robot used for this project was developed by me and uses multiple sensors to gather data from the environment. The robot is powered by Lipo batteries. Two small DC motors in combination with one friction point in front of the robot. This robot configuration is called a differential drive. Robot steering can be achieved by changing the speed of individual wheels.

5.1 Sensors available to robot

- **MPU6050** - Gyroscope
- **12x TCRC5000** - LineSensor
- **Battery voltage sensor**
- **Button**

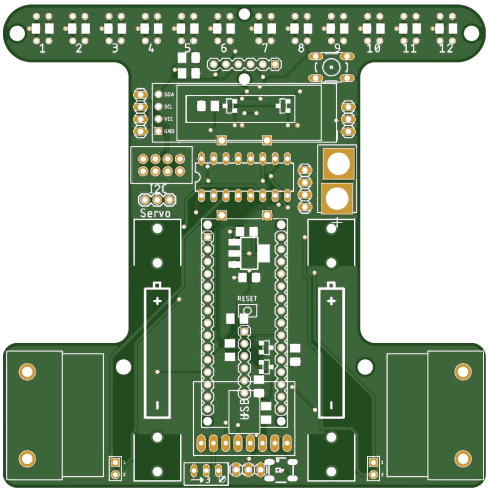


Figure 4: Robot PCB

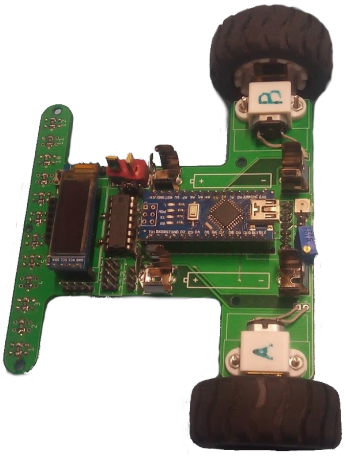


Figure 5: Assembled robot

5.2 Schematics

Robots consist of multiple processors that handle sensor processing and expand the IO of the main Arduino Nano. Two Attiny 84 collect data from 12 light sensors and provide these values through I2C. Additionally, up to 4 buttons can be connected to these Attiny84 processors. The main Arduino Nano is directly connected to the L293D motor driver and the Jetson Nano using USB.

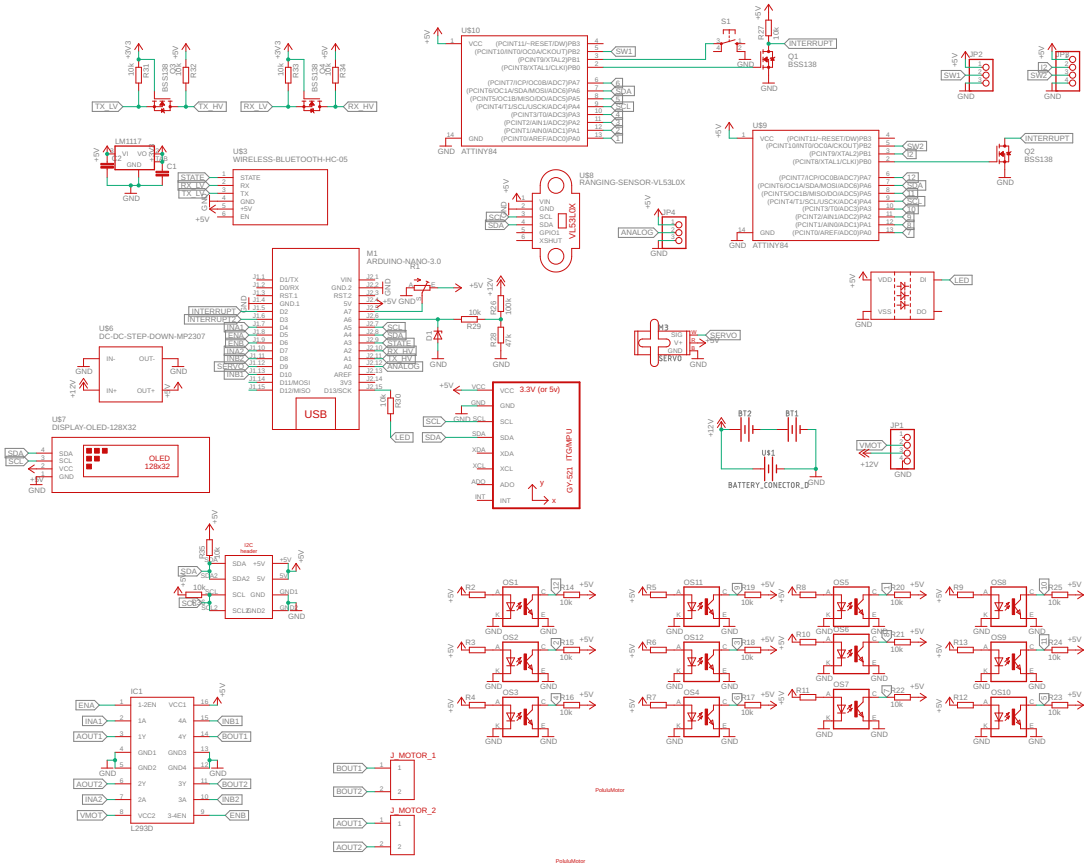


Figure 6: High-level topology of robot.

5.3 Topology

Robot consist of 2 Nodes. Node named `pid_controller` is running on Jetson Nano and computes PID values for motors using data from robot sensors. Sensor data are gathered by a node running on Arduino Nano. This node controls motors and gathers light sensor data from Attiny84 and preprocesses them before sending line position to `pid_controller` node.

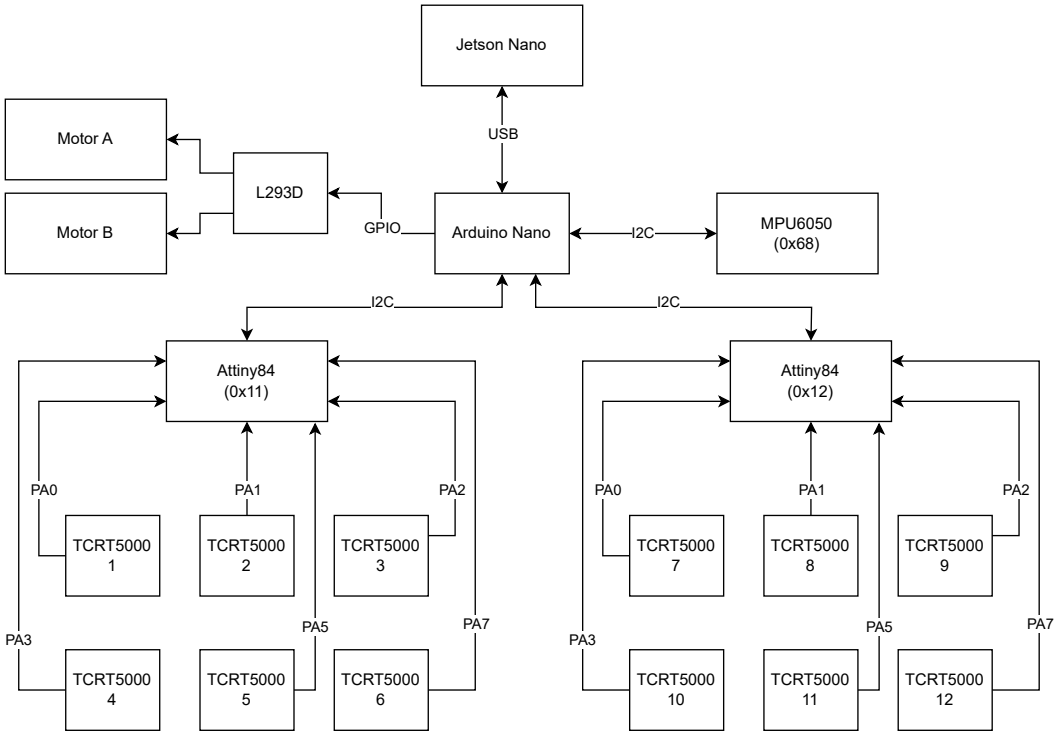


Figure 7: High-level topology of robot.

6 PID

There are multiple ways how to control robots such as line followers. The easiest is to use bang-bang[4] control which works by running the left motor if the robot is not on line and running the right motor when it is. This simple way of controlling the robot works but it is slow and oscillates quite a lot. To get better accuracy we can use more than 1 sensor and assign different speeds of motor for every sensor. This system can oscillate less but it is quite tricky to calibrate these values and it oscillates still. If we assign values equally distributed we have created a P controller. P controller can be extended by integral (I) and derivative (D) which eliminates oscillations and makes the system more dynamic[2]. When implementing `PID` regulator it is important to implement an anti-windup filter as well[1].

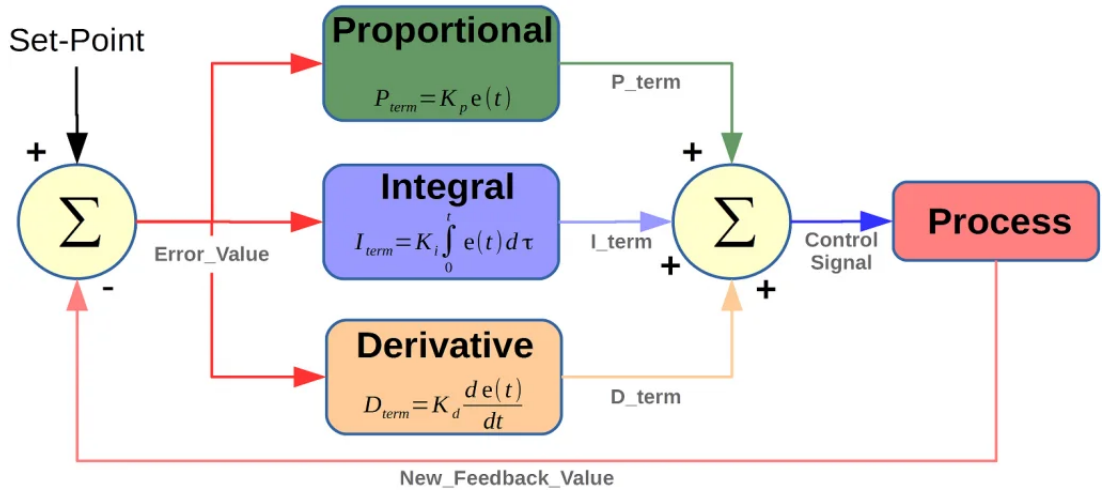


Figure 8: PID controller schema[2].

6.1 PID tuning

PID tuning is a method used to optimize the performance of a control system. Overview of each component:

- **Proportional Gain (P)** - Component produces an output value that is proportional to the current error value. A higher proportional gain increases the system's response to errors, but too high a value can cause the system to oscillate.
- **Integral Gain (I)** - Component sums up the error over time and integrates it into the control output. It helps to eliminate steady-state error (the remaining error after the system has settled). However, too much integral action can lead to instability and oscillations.
- **Derivative Gain (D)** - Component reacts to the rate of change of the error, providing a damping effect. It helps to reduce overshoot and improve the system's stability. However, too much derivative action can make the system overly sensitive to noise.

For Setting P, I, and D parameters and tuning them to get a fast-reacting system that does not overshoot and is stable we can use multiple methods.

6.1.1 Manual tuning

For the line following we do not necessarily need an integral part so we can set it as `0`. Then we need to set the `P` parameter in such a way that we get the maximum speed of one of the motors with the biggest error possible. So if the output from the PID controller should be in the range `0-120` we need to set `P` for a maximum error of `6` to `20`. This will probably lead to a lot of oscillations so we will dampen them using the `D` part which we will increase until the oscillations stop.

6.1.2 Ziegler-Nichols method

This method is similar to 6.1.1 because we set the `P` value in such a way that robots oscillate but it is still stable. Then we measure the period of oscillation and set the I and D parameters according to the table below. Similarly, as in 6.1.1 we do not need `PID` controller for line following `PD` controller is sufficient.

Controller	K_r	T_i	T_d
P	$K_r = 0.5K_{rk}$	-	-
PI	$K_r = 0.45K_{rk}$	$T_i = 0.85T_k$	-
PD	tune	-	$T_d = 0.12T_k$
PID	$K_r = 0.6K_{rk}$	$T_i = 0.5T_k$	$T_d = 0.12T_k$

Table 1: Table of values:

There also exist more advanced methods like Relay Auto-tuning Method[5].

7 Implementation

Github Repository

7.1 ROS topics

Implementation uses 2 nodes called `pid_controller` and `arduino_nano`. These nodes subscribe to each other's topics called `motors` where motor values are sent from the `pid_controller` and `line`, where the position of the line is sent from `arduino_nano` node.

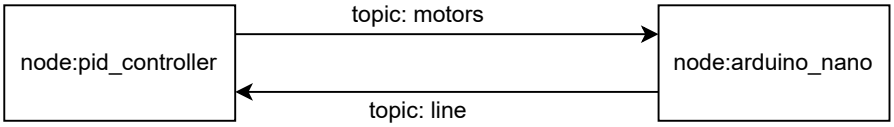


Figure 9: ROS nodes and corresponding topics.

7.2 Arduino

Arduino code gets values of all 12 light sensors using I2C communication with Attiny84. It digitalizes values from sensors and calculates the position of the line. This line is sent to Jetson Nano, which uses this value to calculate the speed of motors which is then sent back to Arduino. Arduino motors run on constant speed `127`, which is adjusted by adding and subtracting value from Jetson Nano.

7.3 ROS

ROS code recieves value from Arduino using `ROS serial`. After receiving the position of the line it calculates the speed of the motor using line position as an error value using the PID controller. Motor speed is than send to topic `motors`.

8 Running code

To run the PID line follower it is necessary to connect Arduino Nano to the USB of Jetson Nano. After that, you can SSH to Jetson Nano and run the following commands.

```
cd ROS_follower
./setup_ros_environment.sh
catkin_make
source devel/setup.bash
sudo nvpmodel -m 1
roscore &
rosrun rosserial_python serial_node.py /dev/ttyUSB0 &
rosrun line_follower PID.py
```

To get an Arduino Nano USB port you can run:

```
./listserials.sh
```

Alternatively, you can run it using the launch file, however, if the connection to Arduino Nano is not established it needs to be edited before running:

```
roslaunch line_follower PID_robot.launch
```

9 Demonstation

Demonstration of line following.

10 Potential improvements

10.1 Line position history

Currently, the robot processes only the current position of the line. This can be improved by keeping a history of the last 10 samples to avoid noise caused by non-perfect lines. In some places, lines can be missing or pale, which using the current algorithm will lead to a big error value for a moment. Also when a robot loses line because it is sharper than its minimum turning radius, the robot does not know on which side line is located so it assumes the left side by default, which is not ideal.

10.2 Using sensors analog value to calculate line position

Extrapolation of line position can be achieved using analog values from the light sensor to get more than 8 positions of the line which is used as an error for the PID controller. This will provide more granular control of motors, however, this approach is more prone to errors, because the track with the line is never homogeneously lit.

10.3 Camera line tracking

The robot is built with modularity in mind thanks to ROS, so it is possible to replace line sensors with a camera that will be running another node. Line detection can provide better resolution than traditional sensors and also can much further which can be helpful. Also, it would be possible to combine data about line from the camera and line sensors as well, which will result in a more durable system.

11 Bibliography

References

[1] *PID Anti-Windup Schemes* [[online]]. Available at: https://www.scilab.org/pid-anti-windup-schemes.

[2] *PID controller implementation using Arduino* [[online]]. Available at: https://microcontrollerslab.com/pid-controller-implementation-using-arduino/.

[3] EDITOR, V. *How to Install ROS Melodic on Ubuntu 18.04* [[online]]. 2020. Available at: https://varhowto.com/install-ros-melodic-ubuntu-18-04/.

[4] EVANS, L. C. *An Introduction to Mathematical Optimal Control Theory* [[online]]. Available at: https://math.berkeley.edu/~evans/control.course.pdf.

[5] HORNSEY, S. *Relay Auto-tuning Methods* [[online]]. 2012. Available at: https://warwick.ac.uk/fac/cross_fac/iatl/reinvention/archive/volume5issue2/hornsey/.