Robot swarm project report

A group of people standing in a room

Description automatically generated with medium confidence***Embedded systems semester 3***

***Last revised date: 2023-12-16***

**Authors:**

* Iskren Zhechev ()
* Farros Ramzy (3767353)

**Version: 2.0X (Draft)**

# Table of contents

[Table of contents 1](#_Toc153918904)

[List of Figures 2](#_Toc153918905)

[Introduction 3](#_Toc153918906)

[Equipment 3](#_Toc153918907)

[Procedure 3](#_Toc153918908)

[Phase 1: Robot distancing 3](#_Toc153918909)

[Analysis and pre-design 3](#_Toc153918910)

[Design 3](#_Toc153918911)

[Code 5](#_Toc153918912)

[Phase 2: PID and feedback control 8](#_Toc153918913)

[Analysis and pre-design 8](#_Toc153918914)

[PID design 10](#_Toc153918915)

[Calculations 12](#_Toc153918916)

[Tuning 14](#_Toc153918917)

[And from that tuning, it is decided that the PID data which will be used for the program are: 15](#_Toc153918918)

[Code 15](#_Toc153918919)

[Phase 3: Communication to Dashboard 16](#_Toc153918920)

[Analysis 16](#_Toc153918921)

[Design 17](#_Toc153918922)

[Retrospective & Recommendation 20](#_Toc153918923)

[References 21](#_Toc153918924)

# List of Figures

[Figure 1 robot flowchart. 4](#_Toc153919113)

[Figure 2 robot state machine. 4](#_Toc153919114)

[Figure 3 input and output timers initializations. 5](#_Toc153919115)

[Figure 4 distance measurement method. 5](#_Toc153919116)

[Figure 5 output control function. 6](#_Toc153919117)

[Figure 6 extended output control function. 6](#_Toc153919118)

[Figure 7 robot state machine implementation. 7](#_Toc153919119)

[Figure 8 proportional concept. 9](#_Toc153919120)

[Figure 9 the concept of integral. 9](#_Toc153919121)

[Figure 10 the concept of derivative. 10](#_Toc153919122)

[Figure 11 position control-concept formula. 10](#_Toc153919123)

[Figure 12 PID simulation design. 11](#_Toc153919124)

[Figure 13 robot flowchart with PID to accelerates and decelerates. 11](#_Toc153919125)

[Figure 14 maximum oscillation of PID. 13](#_Toc153919126)

[Figure 15 Auto tuning the PID calculation. 14](#_Toc153919127)

[Figure 16 PID results after auto tuning method. 15](#_Toc153919128)

[Figure 17 PID implementation. 16](#_Toc153919129)

[Figure 18 Node-Red nodes for robot. 17](#_Toc153919130)

[Figure 19 Node-Red UI dashboard. 18](#_Toc153919131)

[Figure 20 Node-Red debug terminal. 18](#_Toc153919132)

# Introduction

This technical report goes into the process of designing and implementing a feedback control system for a robot swarm project. The project uses MATLAB's Simulink for system design and the practical application of embedded C programming for PID (Proportional – Integral – Derivative) controller. We apply theoretical design using Simulink and the application of control algorithms in embedded systems, providing an overview of the design and implementation phases.

## Equipment

The project makes the use of a laptop equipped with MATLAB IDE and its Simulink extension. Additionally, embedded development software – STM32CubeIDe is used. The required documentation for the project includes the “ParallaxStepResponse” Excel document for data.

# Procedure

## Phase 1: Robot distancing

This phase focuses on enabling the robot to navigate and avoid obstacles. The programming depends on real-time sensor data to guide the robot's movements and make it to operate safe in the environment.

### Analysis and pre-design

For the successful of this project an analysis was conducted and understanding of the functional requirements. The requirements are:

**FR1 - Obstacle detection and movement:** The robot is programmed to move straight forward until it encounters an obstacle within a 10 cm range, measured by an ultrasonic sensor.

**FR2 - Directional change upon obstacle detection:** Upon detecting an obstacle, the robot is designed to change its movement direction randomly within a range of -90 to 90 degrees.

**FR3 - Speed Control and distance measurement via serial Command:** The robot's speed and the obstacle detection distance can be adjusted through a serial command interface.

Following the requirements, the logic of speed control in relation to the obstacle distance was investigated. This involved how the robot's velocity adjustments would be made in real-time based on the distance readings from the ultrasonic sensor. From the datasheet for - “Parallax Feedback 360° High-Speed Servo” the value of the control signal (how to control the speed) for clockwise (faster to slower), stop and counterclockwise (slower to faster) were used. A simple state machine and a diagram for the algorithm for obstacle detection and directional change were made and then changed based on modifications of the system.

### Design

#### Flowchart

The robot's initialization is at the start. The main loop reflects the continuous operation of the robot. Key actions like measuring distance, sending data via UART, and updating measurements based on new data are included. A decision-making process checks the current state and determines the next action. Execution of movement commands and monitoring for obstacles are presented as ongoing actions within the loop. The flowchart concludes and the robot's loop would run continuously.

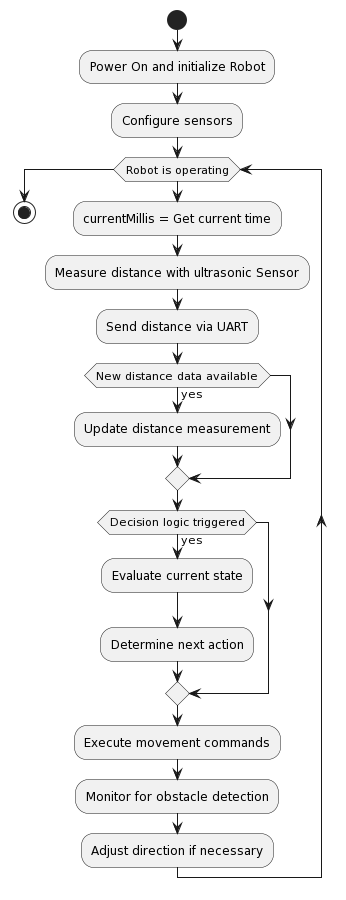


Figure robot flowchart.

#### State machine

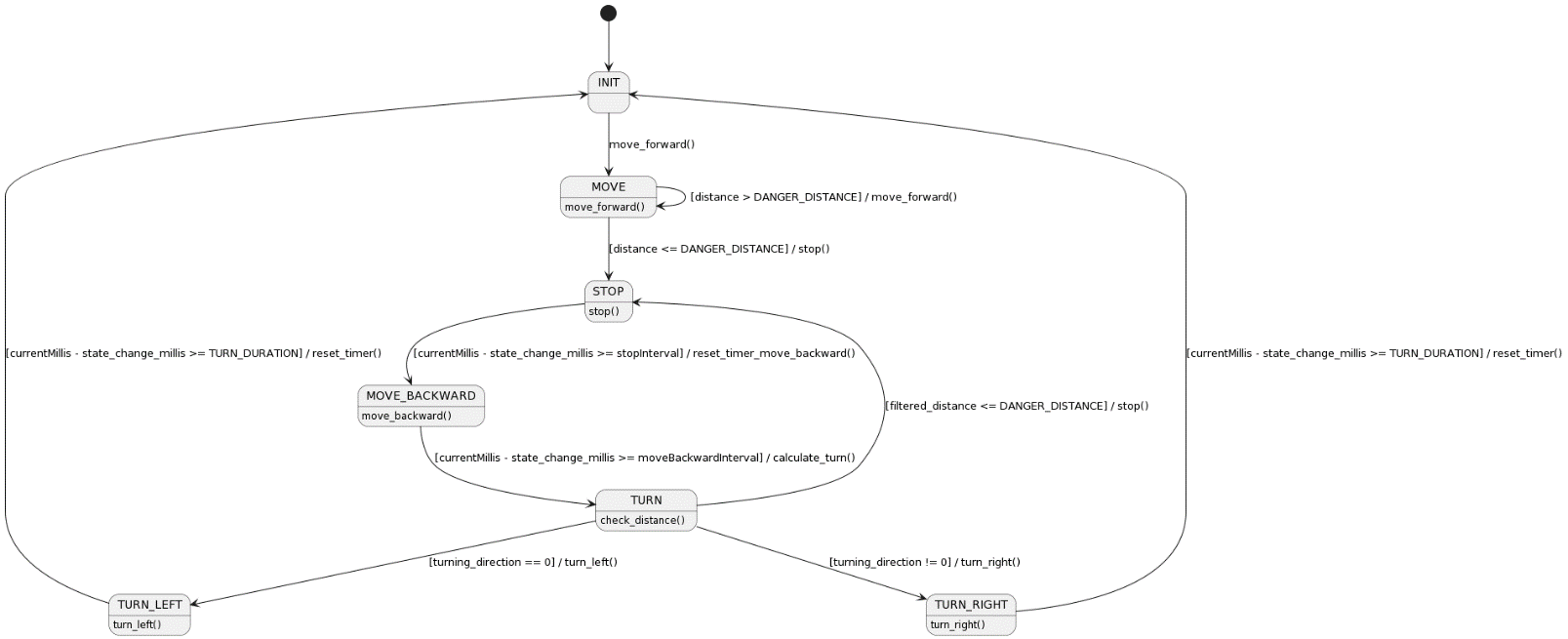


Figure robot state machine.

The state machine for phase 1 is the blueprint of the robot's programming, enabling it to navigate and avoid obstacles. It is based on the functional requirements. The robot begins in the INIT state to start its operation. Once the initialization is complete, the robot transitions to the MOVE state to begin its primary navigation. In this state, the robot continuously moves forward. It constantly measures the distance to potential obstacles using its ultrasonic sensor. If the measured distance is greater than the danger distance (safe threshold etc. 10 cm), the robot continues moving forward within the MOVE state. If an obstacle is detected within the danger distance, the robot transitions to the STOP state. Here, the robot stops its forward movement due to an obstacle being too close. After a predefined interval (checked by comparing currentMillis with state\_change\_millis), if the robot has waited long enough, it transitions to the MOVE\_BACKWARD state to create distance from the obstacle. The robot moves backward for a set duration. After moving backward for the required interval, it prepares to change direction by transitioning to the TURN state. This state is where the robot decides its next direction. It either turns left or right based on a random decision (expressed by the turning\_direction variable), or it may stop if the filtered distance is less than or equal to the danger distance. The TURN state is linked to two sub-states: TURN\_LEFT and TURN\_RIGHT, which dictate the specific turning action. These states represent the robot's action of turning in a specific direction. After completing the turn (which is timed by comparing currentMillis with state\_change\_millis), the robot transitions back to the INIT state to start the process again.

### Code

In our STM32-based robot project, the focus was initially on the timers, recognizing the need for concurrent operation. Two timers were dedicated for the servos controlling the robot's wheels, configured via Timer3\_Init(), and a separate timer for the ultrasonic sensor's trigger and echo, set up through Timer2\_Init() and Timer4\_Init().

A computer code with text

Description automatically generated with medium confidence

Figure input and output timers initializations.

Our first coding steps involved basic functionality for ultrasonic distance reading and servo speed control. The measure\_distance() function was developed for ultrasonic readings:

A screenshot of a computer code

Description automatically generated

Figure distance measurement method.

We then established UART communication for the ultrasonic distance data, using the UART\_send(distance) function. For movement control, we wrote functions like move\_forward() to set the robot in motion:

A close-up of a number

Description automatically generated

Figure output control function.

Similar functions such as move\_backward(), turn\_right(), and turn\_left() were created, each controlling the robot's direction and duration of movement:

A screenshot of a computer code

Description automatically generated

Figure extended output control function.

Our initial logic for moving the robot was implemented in a state machine using HAL\_Delay(). However, we noticed this approach blocked the system, prompting us to refactor the logic using millis(), leading to a more responsive control flow.

A screenshot of a computer program

Description automatically generated

Figure robot state machine implementation.

#### Difficulties

##### *Conflict of timer Interrupts*

Initially, we faced a challenge with the timer interrupts. The interrupts for the servo timers were interfering with each other, and with the timer used for the ultrasonic trigger. This conflict led to malfunctioning timers.

To resolve this, we decided to remove the interrupts from the servo timers, limiting the use of timer interrupts exclusively to the ultrasonic trigger. This adjustment ensured that the timers functioned without interrupting each other.

##### *Removing blocking delays*

Another issue we encountered was the use of HAL\_delay in our system. This function was causing blocking, particularly noticeable during certain actions like turning. For instance, while the robot turned for 4 seconds, the HAL\_delay blocked all other processes, including distance measurement, rendering the robot unresponsive to its environment during this period.

To overcome this, we implemented a non-blocking approach using millis(). Transitioning to this method was challenging but successful. It allowed our system to execute time-based actions without halting the execution of other critical tasks, such as distance monitoring.

##### *Challenges with random direction*

Implementing the random direction feature for the robot also created some difficulties. Our initial approach to incorporate this within a specific state led to complications in determining a new direction.

To address this, we moved the random direction calculation in a different state. So we calculate the turn direction into the move backwards state and then apply it when we are already in the turning state. This approach enabled us to compute the direction independently and then apply it within the state machine.

## Phase 2: PID and feedback control

This phase focuses on enabling the robot to use a feedback control system with PID and control the speed of the robot based on the distance.

### Analysis and pre-design

To unseen the volatile behavior of the robot while it is moving and detecting object in front of it at the same time, a feedback control system is needed.

Feedback control is a method to controls the output of a system, based on an input to maintain performance of the system. And in this case, the robot is using the ultrasonic sensor as an input which controls the output speed and directions of the servos.

As the obstacle distance is getting closer to the robot, the robot must slow down the speed of its servos until the robot stopped at the limit distance of the danger/obstacle. It is possible for the robot to just stop immediately if this feedback controlling method is being ignored. However, the safety of the simulation cannot be guaranteed by this option.

In an assumption that the robot is transporting an object, direct brake and acceleration would make the robot to drop the object by accident. And other than that, a frequent high change of power from the PWM will make the brain and actuators of the robot to be less durable.

To do this feedback control of the robot, a PID method is being implemented.

PID is an abbreviation of the Proportional Integral and Derivative. Each part of this method is described and visualized in these images below.

A graph showing two people

Description automatically generated

Figure proportional concept.

This image in the left is a representation of the proportional. Proportional is the distance between the start point to the target. As the object move from position A to B, the distance between the object to its position will become shortened. This distance change is a value needed for the integral of the PID to control the change of velocity of the object as it is moving to the target.

A diagram of a curve

Description automatically generated

Figure the concept of integral.

This picture above is a representation of an integral. Integral is an accumulation to speed up and slow down the velocity of an object due to its proportional while moving from position A to position B. As the object start moving, the integral will speed up the object until it become close enough to the end position. Suppose if the object is the robot and the end position is the setpoint of the robot, the robot will start to move as fast as it can, gradually, until it recognize the setpoint which is being detected by the sensor, then the integral will slow down the robot until it stopped, exactly on the set point.

However, the servo of the robot has its limit to speed up and to slow down. And the PWM values of the servo also can be bounced uncontrollably which make the robot speed reaction oscillates a lot.

This is where the derivative of the PID needed.

A diagram of a person and a graph

Description automatically generated

Figure the concept of derivative.

This picture on the left represents the derivative behavior of the PID. Derivative is an accumulation to control the integral while changing the speed of the object to prevent an oscillation, considering the quick-change behavior of the integral itself. The derivative task is to make sure that the change of the object’s velocity is always aligned with the ideal change, which is drawn as the dotted curve line in this picture on the left.

### PID design

#### Simulation Concept

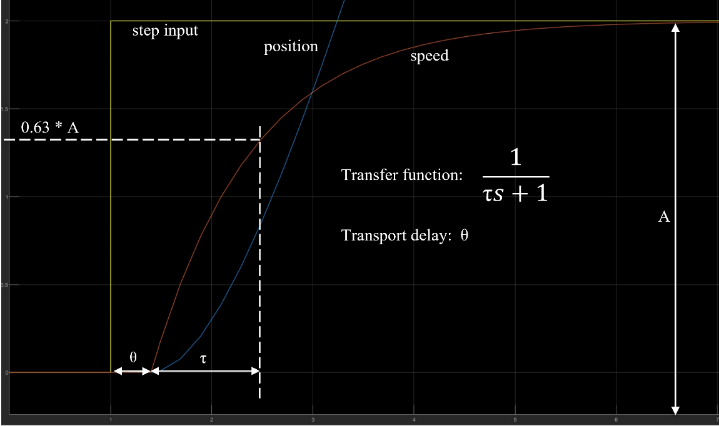


Figure position control-concept formula.

This image on the left is a concept how to get the PID value for the robot system. This method contains the default multiplication of a servo speed area, the formula of the transport function, and a hint of where the transport delay and transport change are placed.

This information will be useful to do the manual calculation of the PID by using the Ziegler-Nichols method.

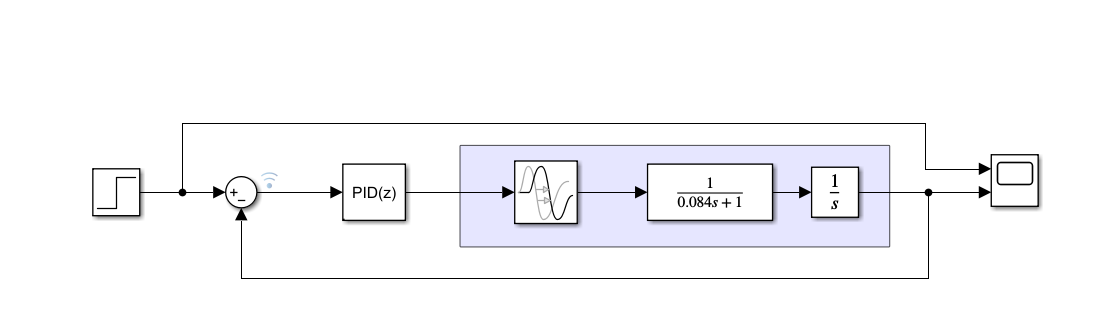


Figure PID simulation design.

This image above is the simulation design of the servo using PID method in Simulink MATLAB.

By using this simulation design, a method to find the PID values for the robot can be easier than by only using the manual calculations.

#### Robot Flowchart with PID

A diagram of a robot

Description automatically generated

Figure robot flowchart with PID to accelerates and decelerates.

This picture above shows how the PID affect the system of the robot while it is moving forward until it detect an obstacle.

If the robot sees nothing, the robot will accelerate itself to the max speed. And when the robot detects an obstacle, it will slow down itself until it stopped at the setpoint, then it moves backward a little bit before turning to the other direction.

The robot will loop itself between the stops, move backward, and turning until it sees nothing in front of it, then the robot will move forward again with its speed which is slowly accelerating or decelerating.

### Calculations

As what described on the analysis and design, the feedback control simulation concept for this project needed the PID calculation. The PID values can be received by only using the manual calculation using the Ziegler-Nichols method. However, adding a simulation in Simulink from the MATLAB (as an example) would improve the feedback control’s precision.

In this project, both methods are combined to make sure that the feedback control of the robot is perfect.

To start with the calculations, a reference to the parallax rotation speed data must be read first. This data provides the average speed and the starting point of the servo in two conditions, which are the loaded and unloaded servo.

And from there, it is decided to use the loaded version since the servos which are being used for the robot are holding the body of the robot itself, including the microcontrollers.

And by this decision, a data taken from the parallax speed datasheet can be listed in this table below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Timestamp (ms) | Change of Position (∆ pos) | Servo position | Speed load (A) |
| Start point | 41 | 0 | 0 | 0.05 |
| Right wheel | 819 | 2 | 1436 | 1.95 |
| Left wheel | 884 | 2 | 1571 | 2 |

From this data and the graph of the formula in the simulation concept above, we can do some manual calculations to get the kP, kI, and kD constants for the PID controller of the robot.

As shown in the graph of the formula in the design, speed of servo can be searched by:

And the period (T) from the to τ is formulated in:

From these, the kP, kI, and kD calculations for the left and right wheels’ PID can be listed as below:

|  |  |
| --- | --- |
| Left Wheel:   * Speed       (This value is precisely equal to the speed with ∆ 2 pos in the parallax table)   * Period   to the with 1.26 value in the parallax data document is 127  127 =  =  =  **=** | Right Wheel:   * Speed       (This value close by the 1.21 speed value with ∆ 2 pos in the parallax table)   * Period   to the with 1.21 value in the parallax data document is 125  125 =  =  =  **=** |

After calculating both necessary speed and period, now is the time to use the Simulink App from the MATLAB to find the maximum oscillation. This oscillation can be searched by adding the transport delay and the period to the transfer function to the PID design in Simulink.

Then, simulate the scope of that design multiple times, and changing only the proportional value to get the balanced volatile graphic which is presented in this image below.

A screenshot of a computer

Description automatically generated

Figure maximum oscillation of PID.

This volatile graphic is the maximum oscillation of the servo. By this, it is known that the max oscillation of the right wheel is 21.45. and the left wheel is 21.415.

The time change between one wave in this oscillation () of the right wheel is 0.45 ms. And the time change between one wave in this oscillation () of the left wheel is 0.458 ms.

From this value and the calculation using the Ziegler-Nichols method, the PID that were received are:

kP = 5.2697

kI = 2.413

kD = 0.3339

However, these results still oscillate the servo behavior a little bit on its starting point. And this is where tuning for the PID is needed.

### Tuning

PID tuning means auto-tune the manual calculation to precise the servo behavior while it changes its speed.

This tuning in the MATLAB needed the kP, kI, and kD values, followed by the sample time of the PID which then can be manipulated by its response time and transient behavior on a tuning graph such as shown in this image below.

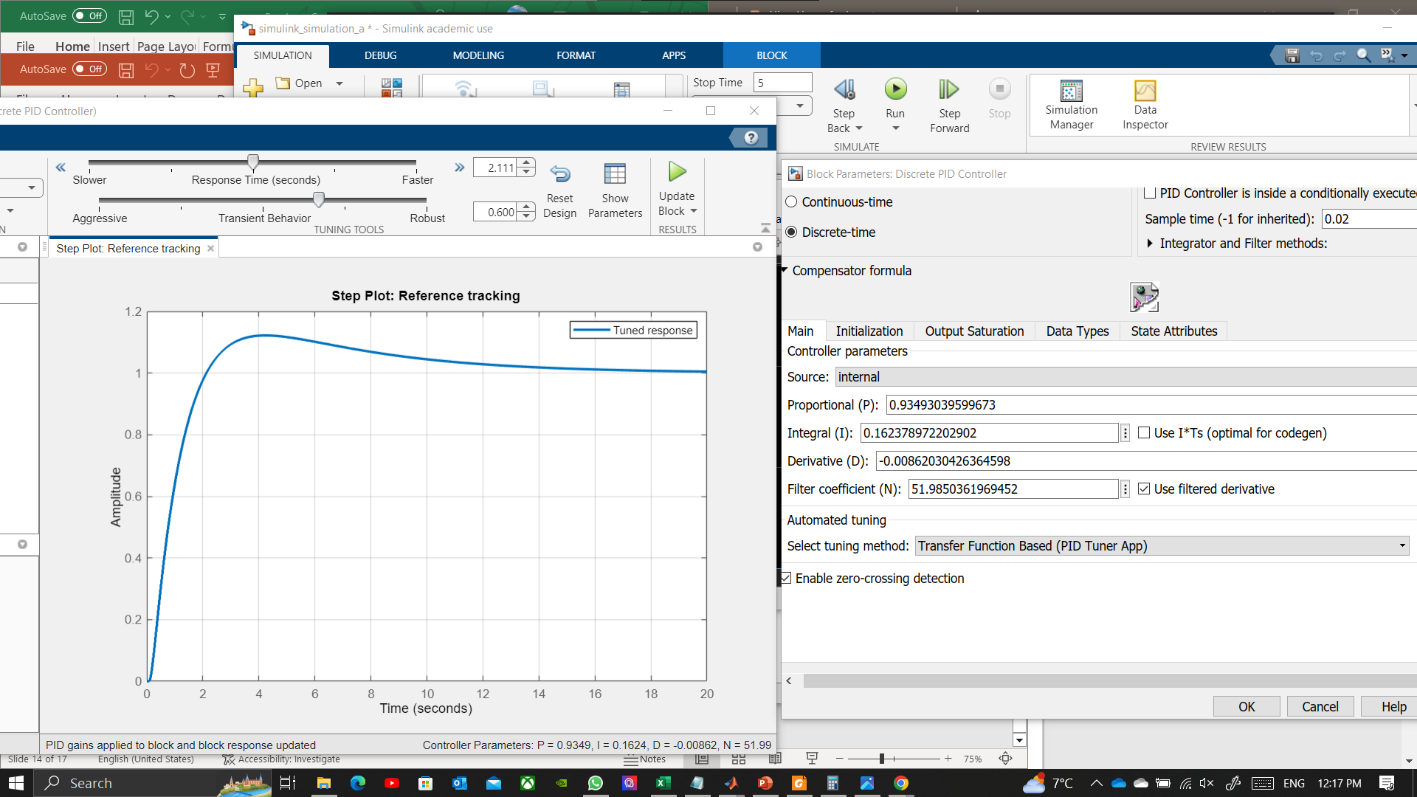


Figure Auto tuning the PID calculation.

However, adjusting this tuning in Simulink needs time to make it correct. As can be seen in this image above for example, there is still a negative value resulted from the tuning while the graph itself still over bounce a little bit from 2 until 12 seconds which is still needed to be smoothen.

And by adjusting the response time and transient behavior properly and repeatedly, this smooth PID behavior such as shown in this picture below can be received.

A screenshot of a computer

Description automatically generated

Figure PID results after auto tuning method.

### And from that tuning, it is decided that the PID data which will be used for the program are:

kP = 0.943420206938106

kI = 0.0916093322883466

kD = 0.247757540890173

### Code

Implementing the PID to the code is much easier than implementing the timer’s setup and synchronizing the servo with the sensor.

As the trial and error only happened in the calculation, the coding part of the PID implementation are only adding the kP, kI, and kD values to a function which will be used for the wheels whenever the robot is moving.

This picture below is how the PID data from the calculations are implemented.

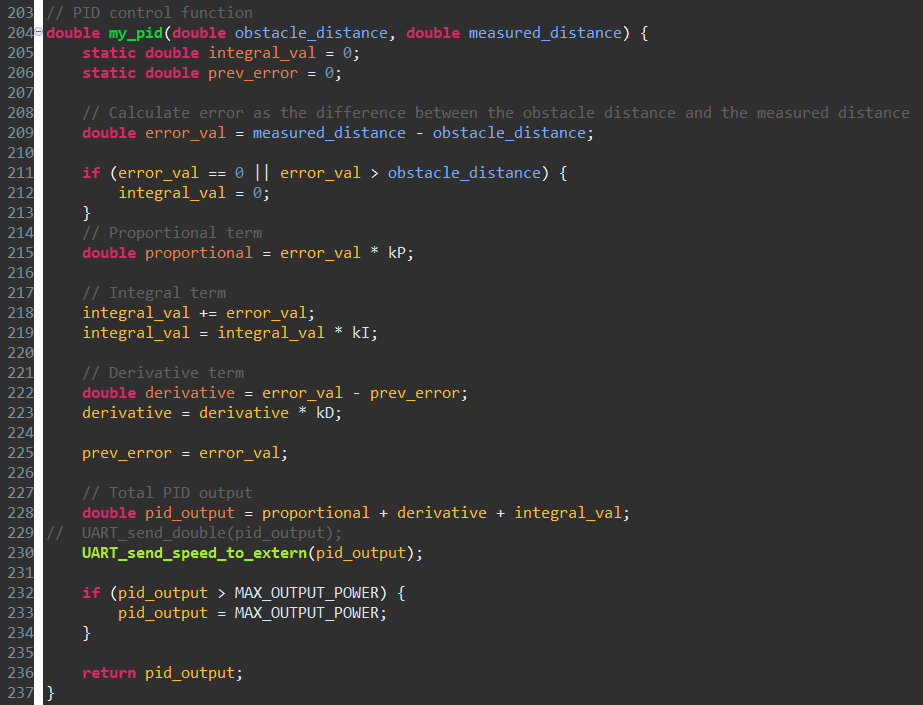


Figure PID implementation.

Since the speed of the robot depended on the integral, the integral should not do anything to the robot if the distance of the obstacle is still too far from the danger distance. This will prevent the robot to behave anomaly when it reads something too far from its sensor’s vision area. And this is the reason why the integral value is manually tweaked to 0 in the beginning of this function.

Other than that, it is known that the max speed of the servo is about 1200 clockwise and 1700 anti-clockwise. If the PID output plus the normal speed of each servo are exceeding these values, the robot will speed up and slow down uncontrollably which will result the oscillations which are supposed to be prevented by the PID itself.

Therefore, if the PID value is higher than the maximum output of the servo, the PID value should be tweaked to be equal with the maximum output power.

## Phase 3: Communication to Dashboard

This phase is an extra phase to communicate the robot to an online UI so the user could observe the speed and direction’s behavior of the robot.

### Analysis

To make sure that the speed and state direction of the robot is visible to be observed by the user, a user-friendly interface should be applied.

There are several options to do this method, like from using the wired communication to the desktop terminal, until using a wireless communication to a certain website.

And this robot is using the website method, using the MQTT and Node-Red protocols.

To do this, the robot must use an UART communication from the STM32 to the ESP32 since the STM does not have access to the internet, unlike the ESP device.

After the ESP receive the message, the ESP is tasked to filter the message and upload the data to the MQTT topic.

And from that topic, the Node-Red which is connected on the other side will sort the message again and visualize them on its UI.

### Design

#### Node Red

Node red is website protocol which allows maintainer to organize the flow of communication between clients and servers.

And this organizing method can be done by placing nodes and connect them to which do the receive message, send message, and filter or change message in some operational nodes.

For this robot, the Node-Red design which is being implemented is a design that use the MQTT communication between the server and client.

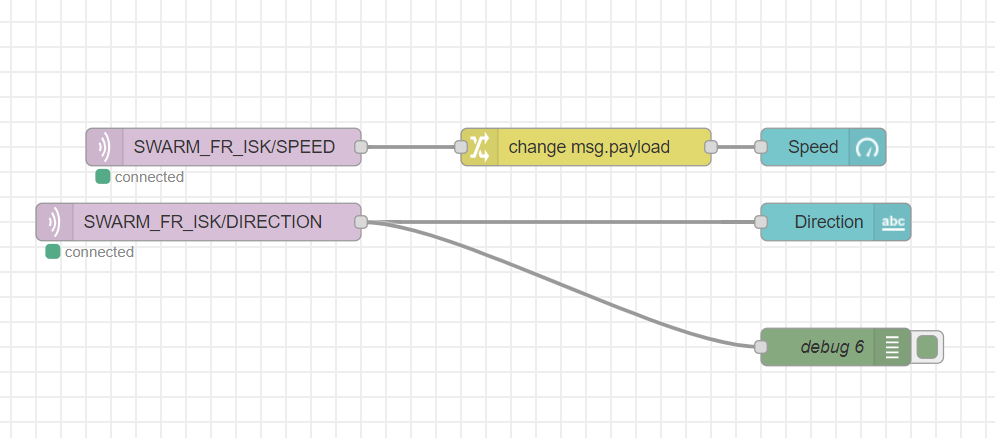


Figure Node-Red nodes for robot.

What can be seen in this picture above, the Node-Red only acts as the receiver of the message from MQTT before displaying the incoming message as a text or gauge in the online UI.

The yellow node from the Node-Red design is placed to change the message payload from text to number which will be displayed on the gauge dashboard. And the green node is only a debug printer which is placed to allow the maintainer to check the incoming message on the Node-Red’s debug terminal.

The dashboard design is represented in this picture below.

A screenshot of a dashboard

Description automatically generated

Figure Node-Red UI dashboard.

And the debug terminal is presented in this image below.

A screenshot of a computer

Description automatically generated

Figure Node-Red debug terminal.

#### Communication Protocol

Since the robot is only using a one-way communication to the dashboard as a user monitor, the message protocol is rather simple on the UART part.

In the UART communication, the STM32 need to print the necessary data to the ESP32, encapsulated by symbols as shown as in this table below.

|  |  |  |
| --- | --- | --- |
| Symbol | Name | Description |
| # | START\_MSG | This is a start symbol, indicates where the connected device should read the message. |
| : | SPLIT\_MSG | This is a separator symbol, which tell the connected device that there is multiple type of values which are incoming within one message. |
| ; | END\_MSG | This is an end symbol, which tell the wire-connected device that a message is completely sent. |

Implementation:

1. Speed message

#Speed:20.00;

1. Direction message

#Forward;

And since the message that uses the SPLIT\_MSG is only the speed message, there is no message type needed to be written in this protocol.

The only detection needed for the program is if the message contains speed, the ESP must send the message to the SPEED topic. Otherwise, the incoming message should be printed in the DIRECTION topic.

# Retrospective & Recommendation

By doing this experiment, multiple tuning results can be produced depending on the time range choices of the desired speed based on the provided data. And the tuning itself is not truly ideal to use for the system at raw. Even the Ziegler-Nichols method itself still have flaws in the end with the integration start of the system, even after the max oscillations for the system have been tuned perfectly at start. However, the auto tune system of the Simulink can be used to fix these flaws.

In the end, the tuning system for speed based on the time is really needed to make sure that the system is not overused the power and keep its movement in secure condition.

# References

RM0316 Reference manual (ST, January 2017)

STM32F303xD STM32F303xE User manual  
(ST, April 2019)

900-00360-Feedback-360-HS-Servo-v1.2

ParallaxStepResponse