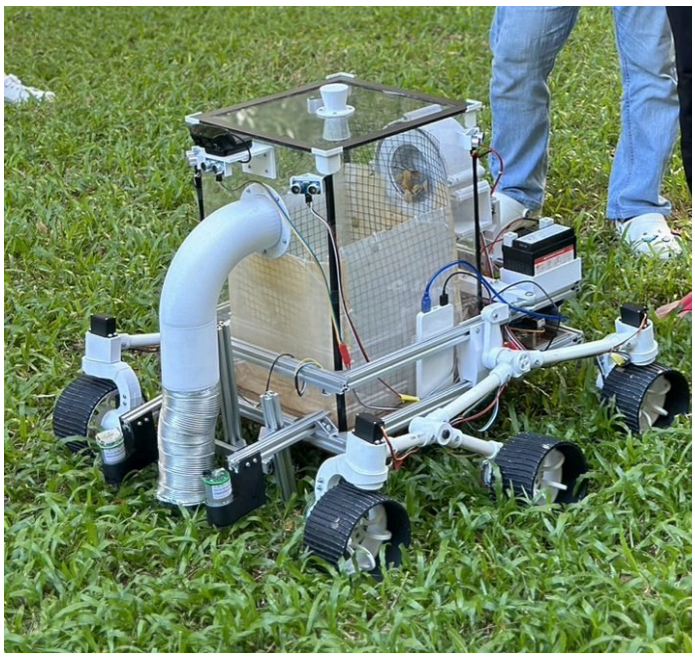
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LATKRABANG
SCHOOL OF ENGINEERING
Group 7 of **Robotics and AI: Sec 2**



LEAVE COLLECTOR ROBOT

01416304 - FEEDBACK CONTROL

INSTRUCTED BY: PROF. DR. PITIKHATE SOORAKSA

NAME: PAING, PYAE, TANAKORN, SURIYARAT, THITIPHAN, RACHATA, SWAN, THURA, SITTIPON, SORNPAWEE

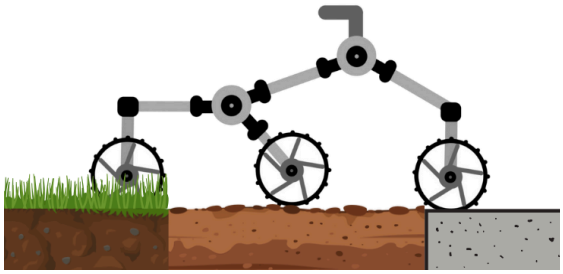
ID NUMBER: 65011416, 65011497, 65011563, 65011557, 65011598, 65011500, 65011683, 65011685, 65011539, 65011546

Tables of Contents

1. Concepts and Drawing

1.1. Suspension System

Purpose: To be able to move on different terrain



The robot design is based on the Mars Perseverance Rover. It was equipped with the **Rocker-Bogie Suspension** system , which allows the robot body **to fully stabilize relative to the ground** even on uneven terrain.

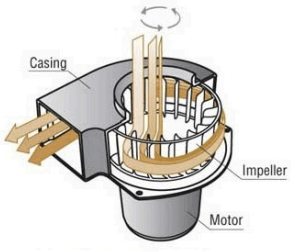
1.2. Leave-collecting Mechanism

Purpose: To be able to collect leaves as much as possible

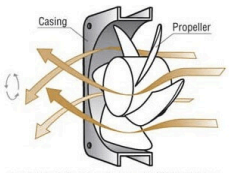
There are several options available for the leave-collecting mechanisms. The following shows some examples. We needed an ensured mechanism that can work on all terrains, (side road, soil, grass). Thus, we decided to use the **“Vacuuming”** mechanism to suck leaves from the ground no matter what type of terrain.

1.2.1. Vacuuming Mechanism

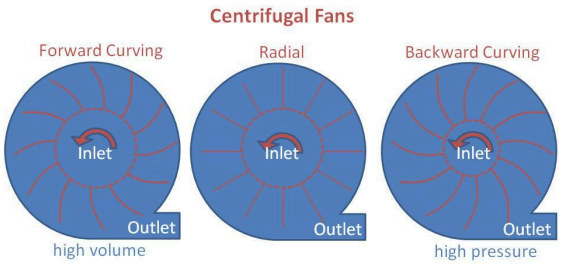
Purpose: To have a powerful vacuuming system

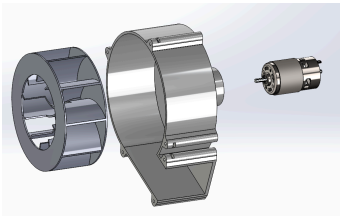


Centrifugal Blower



Axial Blower

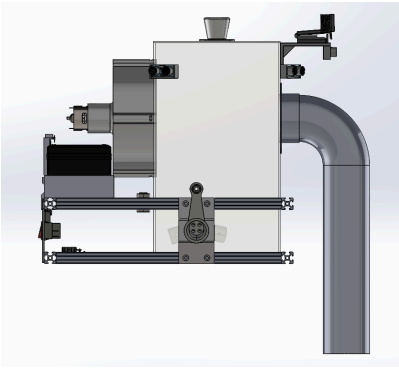


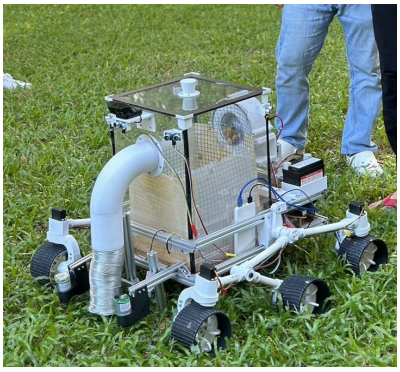


There are several vacuuming mechanisms available, Among them, **the centrifugal pump** is the most suitable choice for our project for its **high pressure pumping power**.

1.2.2. Leave Take-Out System

Purpose: To take out the leaves easily

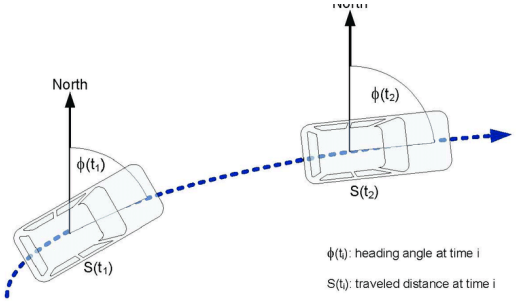


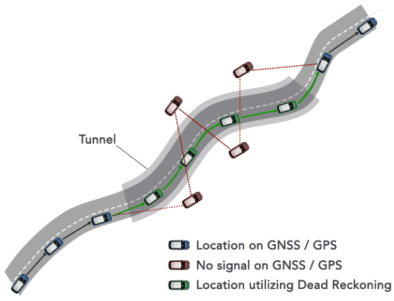


We create a **vacuum box** to take in the leaves and inside there is a **basket made with net** which does not affect the air flow but prevents the leaves from entering the impeller. Moreover, we make sure the box is sealed to prevent leaking air which might reduce the vacuuming pressure.

1.3. Navigation (Dead Reckoning)

Purpose: To be able to know the robot's position relative to the original position

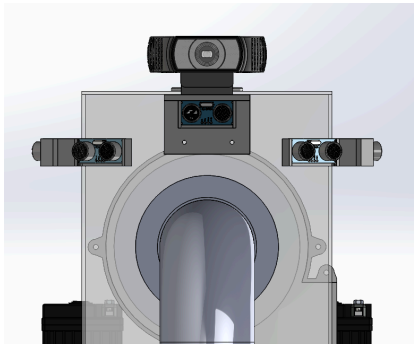


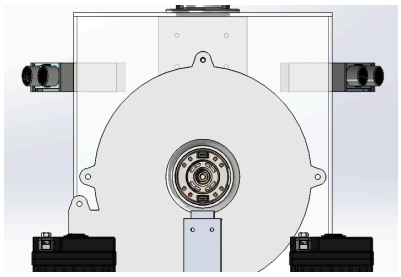


Initially, we tried to detect the robot position by sensor fusion of GPS sensor and accelerometer sensor but the data of GPS sensors that we bought were not precise enough. As we are having a budget limit and the robot will only be run for a short distance, we decided to switch to the **Dead Reckoning Method of Navigation** which calculates the position of the robot from the data of **the motor encoder(Linear Displacement) and IMU sensor(Orientation).**

1.3.1. Obstacle Avoiding

Purpose: To be able to avoid the obstacles along the path while keep tracking on the path

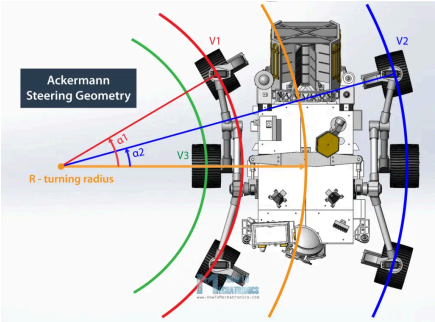




We use ultrasonic sensors to detect the obstacles and camera to know the relative degree of obstacle from the robot so that the robot can smoothly turn towards the movable area. We use two more ultrasonic sensors at the back so that it can track the original path once it passes the obstacle.

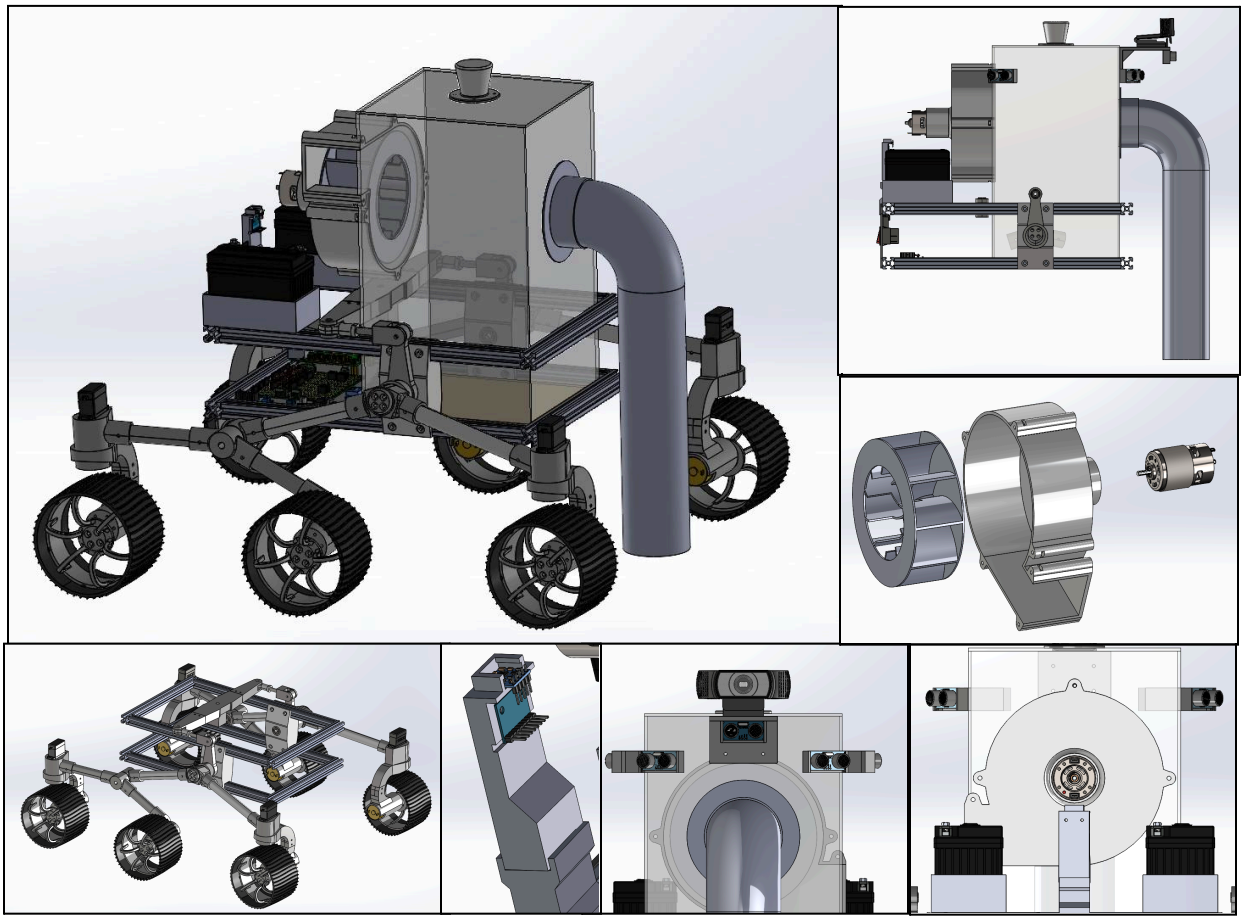
1.3.2. Steering System

Purpose: To be able to turn the robot smoothly



In order to avoid the tire slipping and have efficient turning, we integrated **Ackermann Steering geometry**. It is one of the basic steering mechanisms which does not require much calculations. Since it is hard to implement the four-bar linkage steering to this type of medium-sized project, we decided to use **servo motors** as main actuators for **turning wheels**

2. Design



2.1. Components and Parts

Materials

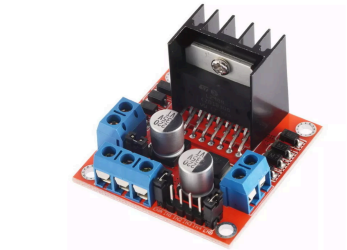
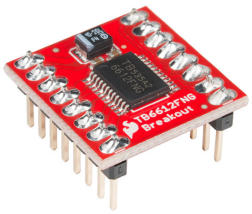
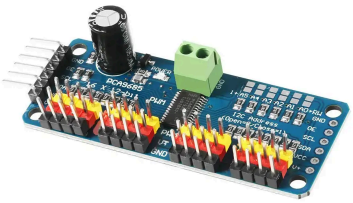
- 20x20mm t-slot aluminum profile
- Acrylic plates
- 20 mm PVC pipes
- Duct Tube
- 3d-printed PLA parts
- T-slot nuts
- Regular nuts & washers
- Screws

Electronic Components

- 6x GB37 12V 250 rpm Geared DC Encoder Motor
- 1x RS775 12V Geared DC Motor
- 4x MG996R Servo
- 3x TB6612 Motor Driver
- 1x PCA9685 Servo Motor Driver
- 5x HC-SR04 Ultrasonic Sensor
- 2x XL4015 Step-Down Module
- 2x 12V Motorcycle Battery
- Logitech C922 USB Webcam
- Raspberry Pi 4B
- Arduino Mega

2.1.1. Electronic component design concepts

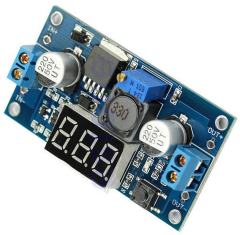
Motor Drivers

In order to control high torque DC motors efficiently, we need to be careful with the choice of motor drivers. We used L298N H-bridge drivers in earlier projects. That module dissipates a lot of heat and the voltage drop is higher than 1.4V. This time we decided to switch to TB6612FNG, a new MOSFET based H-Bridge driver with a small form factor.

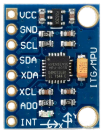
For **servo motors**, we need some module with robust and precise control. Thus, we decided to use a PCA9685 I2C 16-channels servo driver which supports 12 bit PWM.

Step Down Module



In every robotic project, the most important factor is power distribution and power consumption. The robot needs to get required power (more or less) from the battery without any restrictions. The solution for that problem would be a step down module. This step down module can drive 3A when in full load. We used this module in two places. First, servo drivers. Servo needs a 5V power supply and the main power line is 12V. So, we needed to step down to 5V. Second, the impeller fan needs its maximum RPM (~3A). So, we connected the RS 775 motor with a step down module.

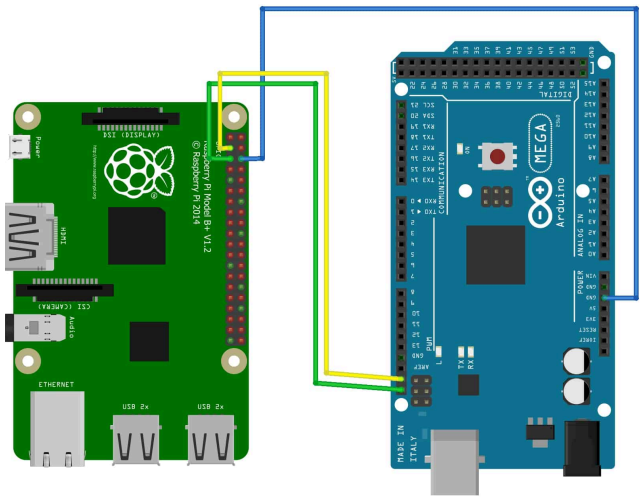
Sensory Unit

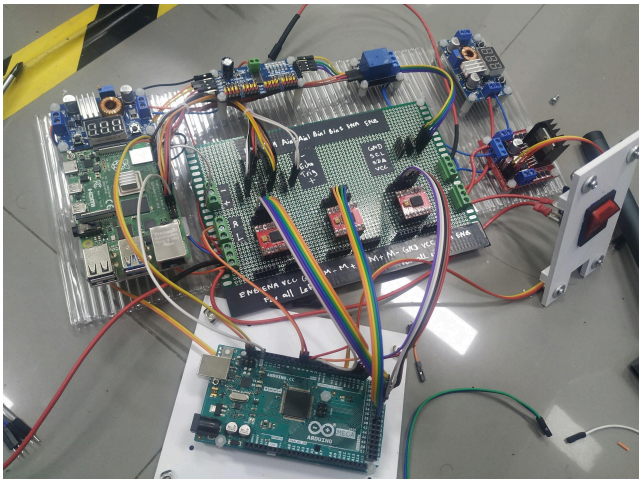



1. **Ultrasonic Sensors** - It was used for obstacle detection and to give feedback to our control algorithm and used for obstacle avoidance.
2. **Logitech Webcam** - It was intended to use as a visual input for the computer vision algorithm we designed.
3. **6-DOF Gyroscope and Magnetometer** - It was used as a navigation assistance that enabled the robot to know its current orientation in terms of row, pitch, and yaw values.
4. **Relay Module** - It was utilized as a controllable switch for the **Impeller-Battery** power line.

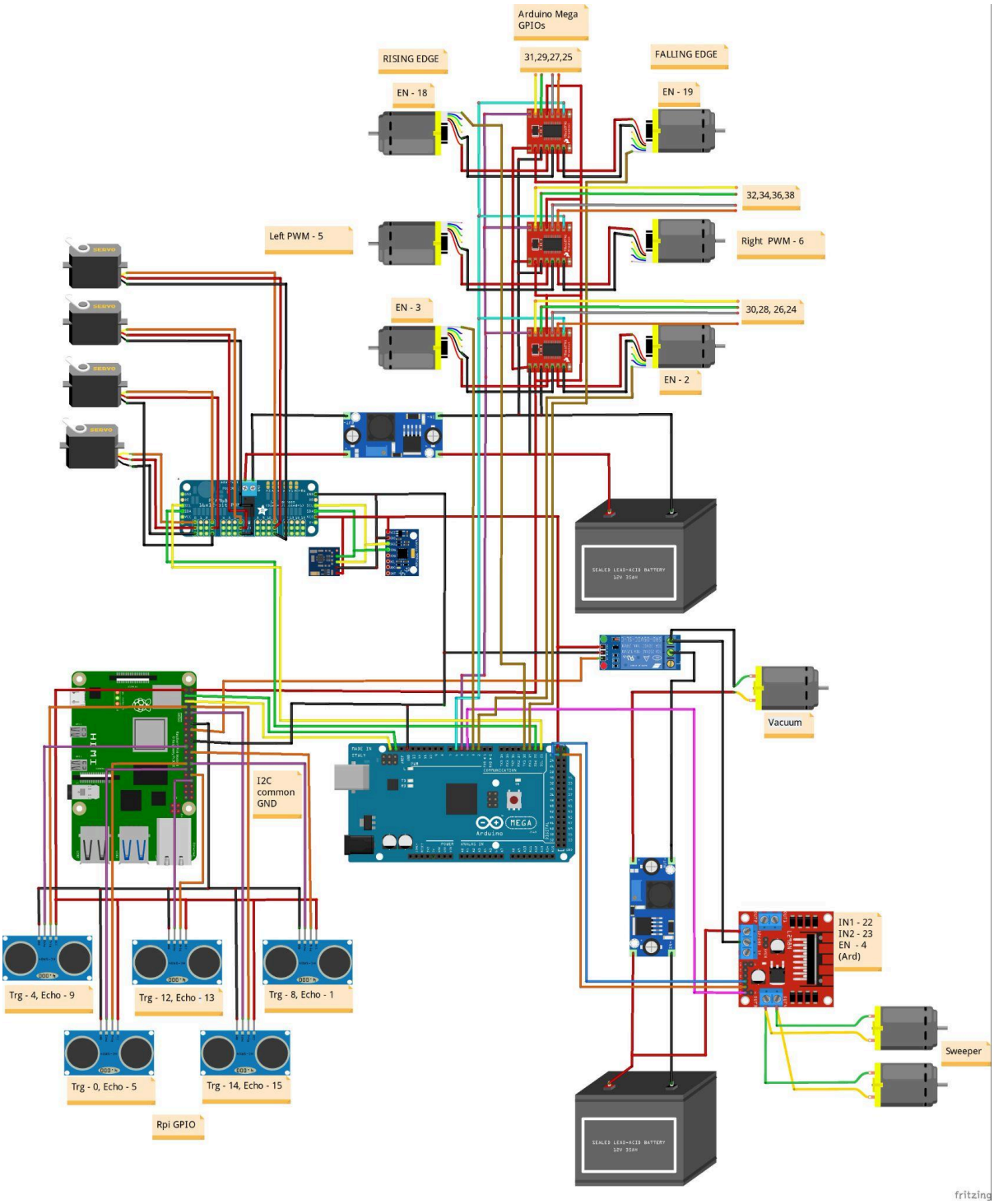
Controller Unit

The brain of the operation is Raspberry Pi 4 Model B. It takes feedback from various sensors, activates its control algorithms and sends commands to the motor drives(actuators). However, with Raspberry Pi's limited pin count, we used an additional Arduino Mega to split the workload–**Raspberry Pi for sole calculation**(brain) and **Arduino Mega for actuation**(Limbs). These two controllers are connected by using I2C protocol to ensure seamless communication.





Actual Connection

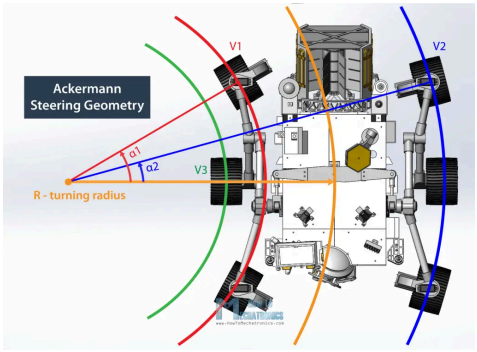


Complete Schematics along with pinout diagram

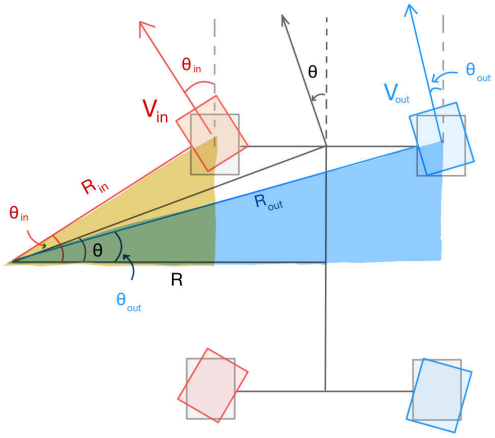
2.2. Kinematics and Dynamics

Ackermann steering

We did some modification on the Actual Ackermann Steering System and we calculated relative speed ratios of left and right wheels for turning speed.
Here is the calculation:



l = length of our robot
w = width of our robot
θ = angle of robot required to turn
θ_{in} = angle of inner path wheel
θ_{out} = angle of outer path wheel
R = turning radius of the robot
R_{in} = turning radius for the inner path wheel
R_{out} = turning radius for the outer path wheel



First we need to find the **Robot's Turning Radius**
By using simple trigonometry,

$$R = \frac{l/2}{\tan\theta}$$

From R, we get the turning degree of each motor.

$$\theta_{in} = \arctan\left(\frac{l/2}{R - w/2}\right)$$
$$\theta_{out} = \arctan\left(\frac{l/2}{R + w/2}\right)$$

For controlling the velocity of the inner path and outer path wheels, we calculate the gain for PWM. This way we can control the velocity of each wheel making it smooth in turning.

$$R_{in} = \frac{l/2}{\sin\theta_{in}} \qquad R_{out} = \frac{l/2}{\sin\theta_{out}}$$

$$\text{gain } PWM_{in} = \frac{R_{in}}{R_{in}+R_{out}} \qquad \text{gain } PWM_{out} = \frac{R_{out}}{R_{in}+R_{out}}$$

2.3. Control System

Purpose: To be able to know the robot's position relative to the original position

We use encoder pulse values to know the distance traveled and we know the orientation of the robot from Magnetometer and Gyro.

Calculating number of pulse per revolution:

N = number of pulse per encoder revolution * gear ratio

N = 448 * 43.8 = 19622.4 pulse per revolution

Calculating distance traveled per revolution

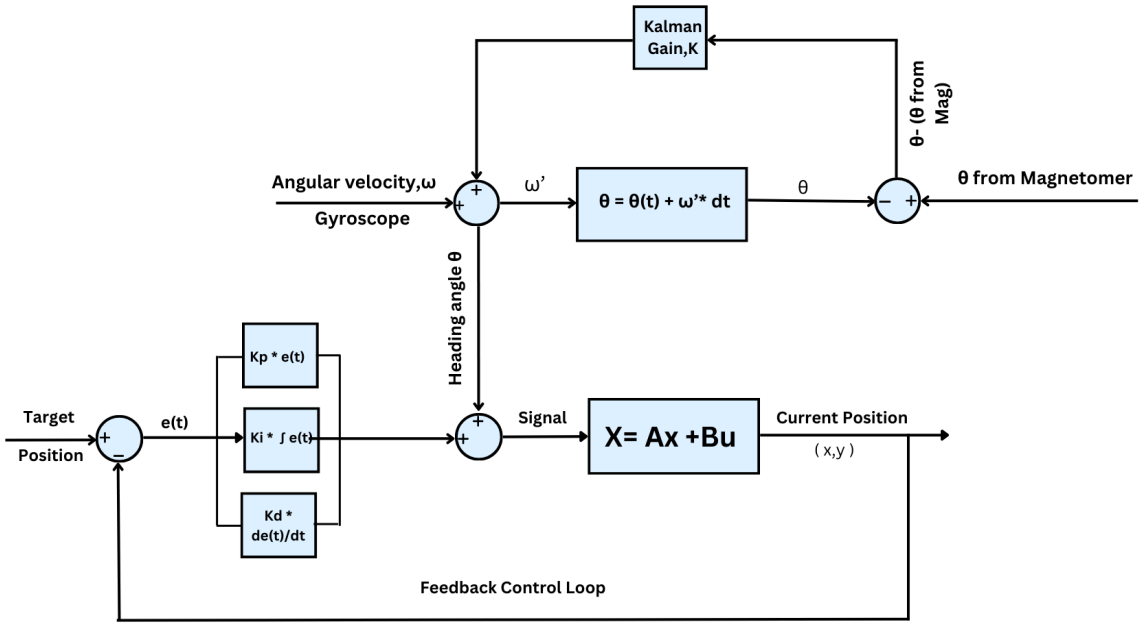
d = π * radius of the wheel

d = π * 0.12 m = 0.377 m per revolution

State Equation (X = Ax + Bu)

State Matrix, **X =** $\begin{bmatrix} x_t \\ y_t \end{bmatrix}$

A = $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ **,x =** $\begin{bmatrix} x_{t-1} \\ y_{t-1} \end{bmatrix}$ **,B =** $\begin{bmatrix} d_t & 0 \\ 0 & d_t \end{bmatrix}$ **,u =** $\begin{bmatrix} \sin\theta \\ \cos\theta \end{bmatrix}$



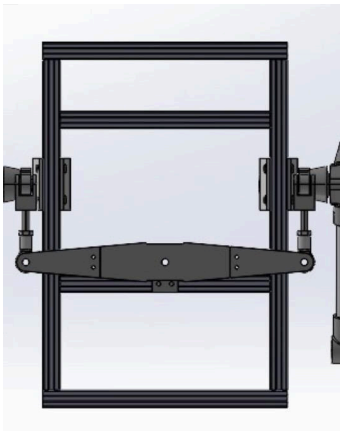
3. Implementation

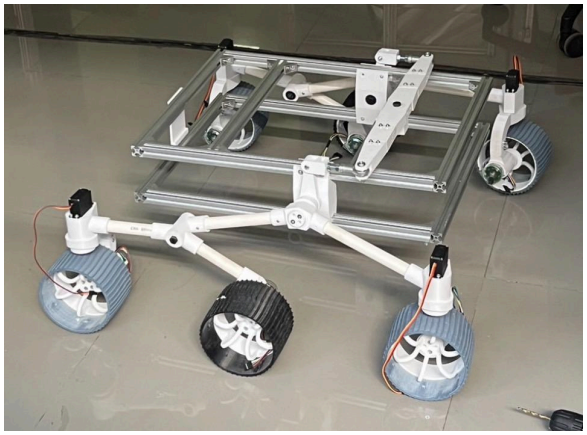
3.1 Hardware

Chassis:

We used a t-slot aluminum profile to build the frame of the chassis. Since aluminum is both strong and lightweight, it can be used to build a foundation for acrylic plates to be attached to. Furthermore, the t-slot on the aluminum profile helps for easy and sturdy assembly using t-slot nuts and bolts.

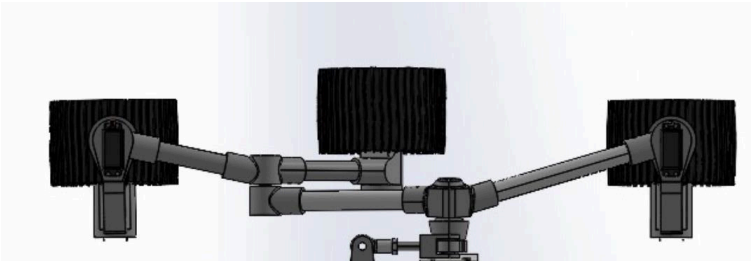
For stability of the robot's body, we added the differential bar.





Suspension:

We built the Rocker-Bogie suspension system using PVC pipes and 3d-printed joints aiming to reduce the overall weight of the robot. The wheels are also 3d printed.



Other implementations:

- 3d-printed fan for vacuuming leaves
- Storage box made of glued acrylic plates
- 3d-printed ultrasonic sensor mounts and camera mount
- Using duct tube for the vacuum system

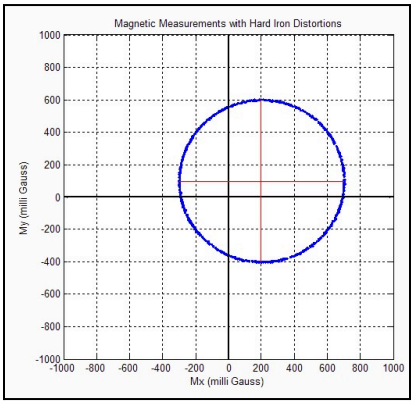
3.2 Software

Calibration of sensors:

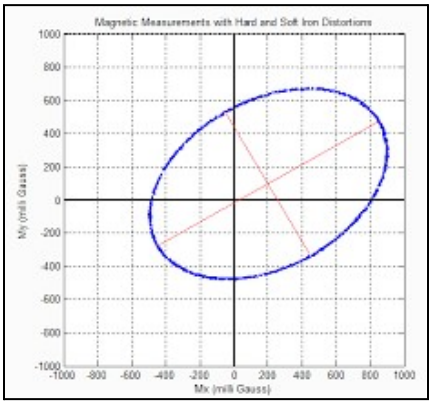
1. **Magnetometer**

Magnetic measurements from the magnetometer are subjected to distortion.

Case 1: Hard Iron Distortions



Case 2: Soft and Hard Iron Distortions

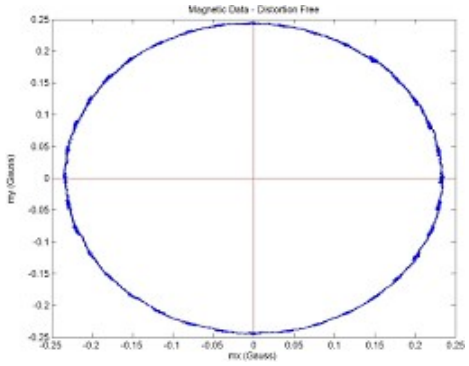


Hard iron such as magnet and current flowing coil could affect the magnetometer data to bias shifting from real measurements. Soft iron such as iron and other metals affect the shape of data to transform from circle to ellipse.

Calibration Method:

Step 1: Eliminating hard iron offsets from the raw data

```
# Calculate mean of raw data samples
x_raw_bias = sum(x_raw_samples) / len(x_raw_samples)
y_raw_bias = sum(y_raw_samples) / len(y_raw_samples)
z_raw_bias = sum(z_raw_samples) / len(z_raw_samples)


# Append Calibrated data to the calibrated lists
x_calibrated_samples.append(x_raw - x_raw_bias )
y_calibrated_samples.append(y_raw - y_raw_bias )
z_calibrated_samples.append(z_raw - z_raw_bias )
```



Step 2: Applying linear transformation for soft iron distortions (ellipse shape) to form a circle.

```
# Transfoming ellipse into circle

# Create Rotational Matrix
c, s = np.cos(0), np.sin(0)
rotationalMatrix = np.array([[c, s], [-s, c]])

# Create Reverse Rotational Matrix
inverse_rotation_matrix = rotationalMatrix.T

soft_factor = rotationalMatrix @ scaleMatrix @ inverse_rotation_matrix  @ sens_vec
```

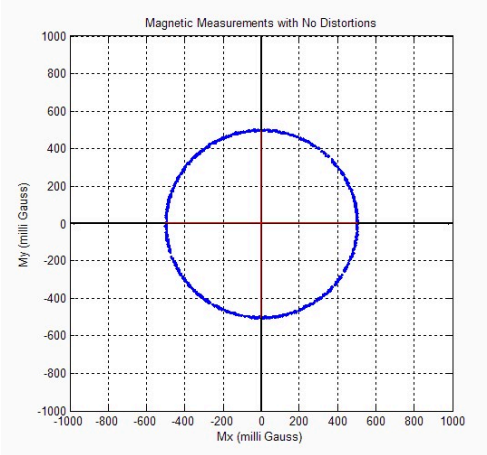


Fig: No Distortion

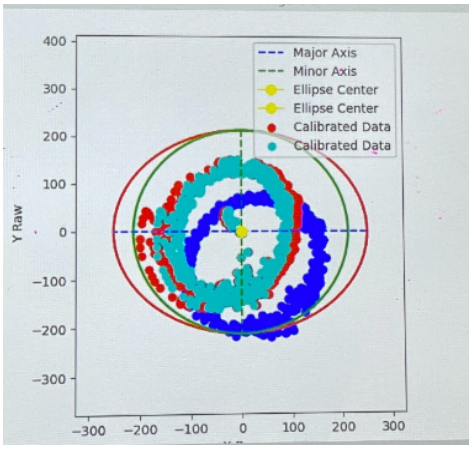


Fig: Calibration Result

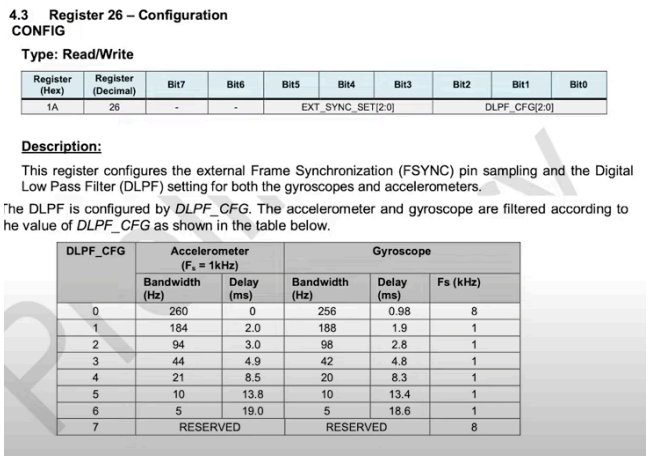
2. Gyroscope

Gyroscopes are susceptible to environmental vibrations, necessitating regular calibration before use to remove bias.

Calibration Method:

Step 1: Enabling low pass filter

We enable the low pass filter to reduce the vibrations caused by motors.



4.3 Register 26 – Configuration
CONFIG

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1A	26	-	-	EXT_SYNC_SET[2:0]			DLPF_CFG[2:0]		

Description:

This register configures the external Frame Synchronization (FSYNC) pin sampling and the Digital Low Pass Filter (DLPF) setting for both the gyroscopes and accelerometers.

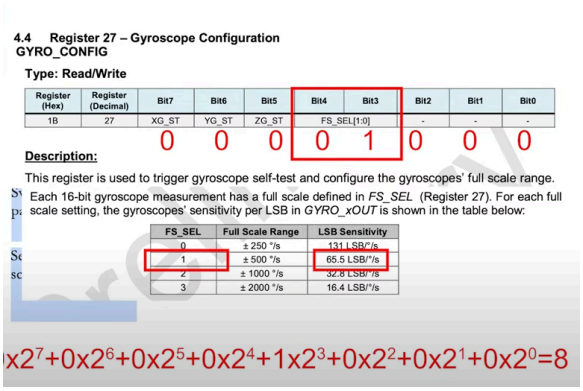
The DLPF is configured by *DLPF_CFG*. The accelerometer and gyroscope are filtered according to he value of *DLPF_CFG* as shown in the table below.

DLPF_CFG	Accelerometer (F_s = 1kHz)		Gyroscope		
	Bandwidth (Hz)	Delay (ms)	Bandwidth (Hz)	Delay (ms)	Fs (kHz)
0	260	0	256	0.98	8
1	184	2.0	188	1.9	1
2	94	3.0	98	2.8	1
3	44	4.9	42	4.8	1
4	21	8.5	20	8.3	1
5	10	13.8	10	13.4	1
6	5	19.0	5	18.6	1
7	RESERVED		RESERVED		8

```
CONFIG = 0x1A

bus.write_byte_data(Device_Address, CONFIG, 0)
```

Step 2: Choosing the sensitivity of gyroscope

We chose the sensitivity of 65.6 LSB / degree/s



4.4 Register 27 – Gyroscope Configuration
GYRO_CONFIG

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1B	27	XG_ST	YG_ST	ZG_ST	FS_SEL[1:0]		-	-	-
		0	0	0	0	1	0	0	0

Description:

This register is used to trigger gyroscope self-test and configure the gyroscopes' full scale range.

Each 16-bit gyroscope measurement has a full scale defined in *FS_SEL* (Register 27). For each full scale setting, the gyroscopes' sensitivity per LSB in *GYRO_xOUT* is shown in the table below:

FS_SEL	Full Scale Range	LSB Sensitivity
0	± 250 °/s	131 LSB/°/s
1	± 500 °/s	65.5 LSB/°/s
2	± 1000 °/s	32.8 LSB/°/s
3	± 2000 °/s	16.4 LSB/°/s

$0x2^7+0x2^6+0x2^5+0x2^4+1x2^3+0x2^2+0x2^1+0x2^0=8$

```
GYRO_CONFIG = 0x1B
bus.write_byte_data(Device_Address, GYRO_CONFIG, 8)
```

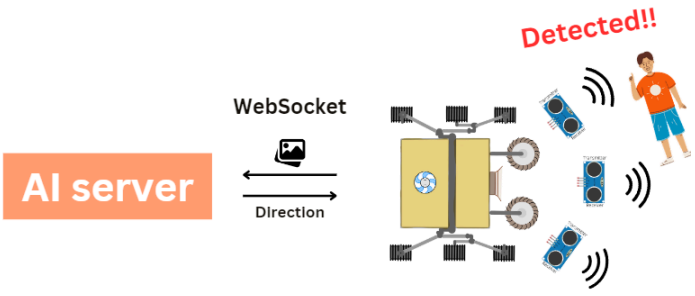
Step 3: Scaling the raw data and calculating the bias

```
def get_cali():

    gyro_cali_x = 0.0
    gyro_cali_y = 0.0
    gyro_cali_z = 0.0

    MPU_Init()
    gyro_scale = 65.5  # Sensitivity scale factor for ±500 dps range

    print("Reading MPU6050...")

    for _ in range(2000):
        gyro_cali_x += read_raw_data(GYRO_XOUT_H) / gyro_scale
        gyro_cali_y += read_raw_data(GYRO_YOUT_H) / gyro_scale
        gyro_cali_z += read_raw_data(GYRO_ZOUT_H) / gyro_scale

        sleep(0.01)  # Sampling rate of 1 Hz

    gyro_cali_x /= 2000
    gyro_cali_y /= 2000
    gyro_cali_z /= 2000

    print("Finished Calibration")

    return gyro_cali_x, gyro_cali_y, gyro_cali_z
```

Obstacles Avoidance

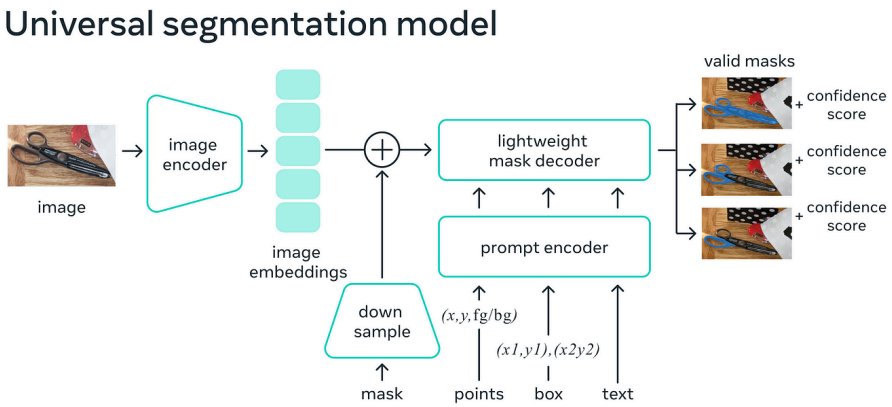
Ultrasonic sensor:

Ultrasonic sensors require a little time interval to read the distance value, and this will cause delay to the whole system. We used the advantage of Python for parallelly reading ultrasonic sensors over time together with robot controlling by running the ultrasonic sensors coroutine in the background using Thread programming. The distance signal of these sensors were used as a trigger condition to send a request for an AI server.

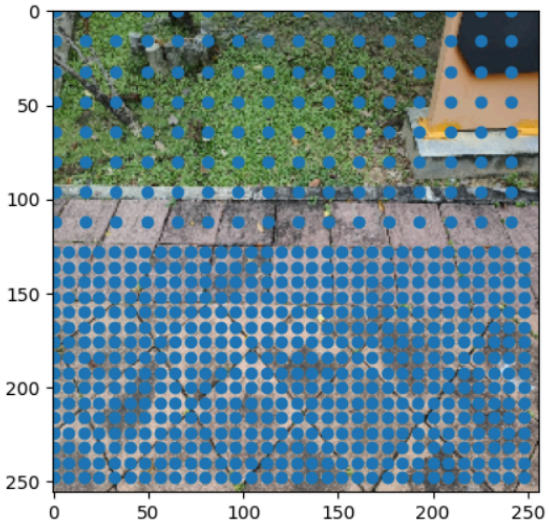


Camera and AI:

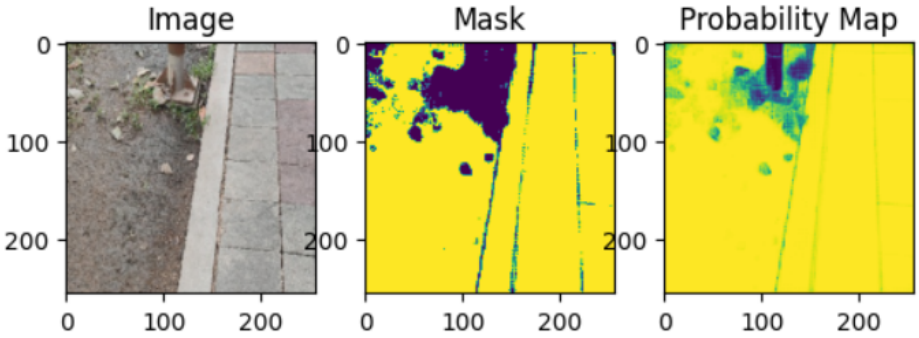
The segmentation AI was applied in this project to fusion with the ultrasonic sensors for obstacle avoidance tasks. With only the responses from ultrasonic sensors, we can observe only the existence of obstacles but not the size and exact position which is not enough for determining the direction of the robot to precisely avoid the obstacles. We used the fine tuning of SAM (segment anything model), the segmentation weight developed by Meta, to detect movable places. The advantage of SAM is that it can be fine tuned with prompts, such as bounding boxes and set of points, to increase the accuracy of the specific task.



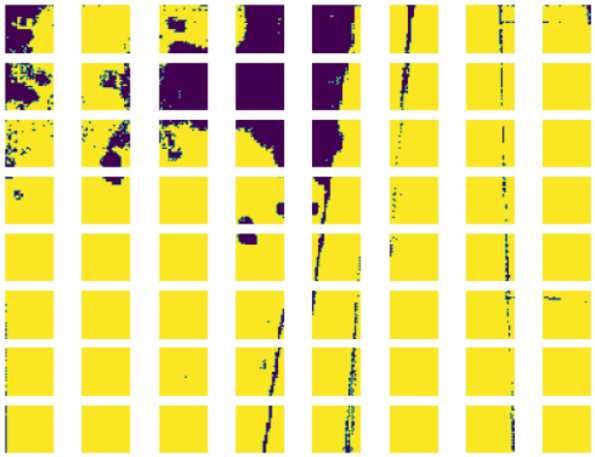
The figure above is the architecture of the SAM model showing that it has a prompt encoder available for extra parameters.



The figure above shows how we prompt the model in the fine tuning pretrained weight. As you can see, we feed the high density points into the bottom half of the image and few density points into the top half. With this prompting, it points the model to emphasizing on the closer area more than the farther area with respect to the position of the robot.



The figure above are the results of the fine tuning SAM model to detect the movable area. Yellow means no obstacle.



To calculate the direction of the robot, we split the detected image into small patches, then distinguish between movable and non-movable patches. The path finding algorithm will return the weight of each row from the bottom to top.

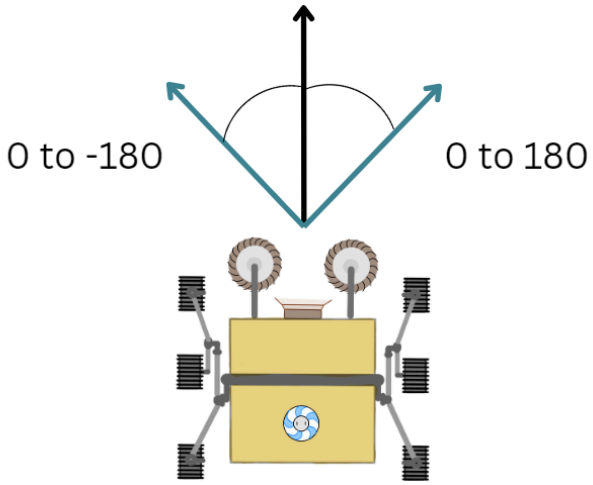
```
[0, 0, 0, 0, 0, 777, 50, 777]
```

0 means the robot can move normally.
777 means either left or right, -777 means neither left nor right.
Positive means right, Negative means left.
The magnitude means how much should the robot turn.

Controller

PID controller

A PID controller technique was applied to make the robot move more stable. First, we set the target yaw position. Next, calculate for the error by subtracting it with the current position.



The figure above shows the position system that we used. Positive position means the robot needs to turn right (more left speed) and Negative position mens turn left (more right speed).
Here is how we calculated the PID using the stated principle.

```
# The robot use positive and negative of 180 degree system
# RHS is the positive angle and LHS is the negative angle
if tar > 180:
    tar = tar - 360

if pos > 180:
    pos = pos - 360

# Get time for each iteration
delta_time = 1
if self.prev_time != -1:
    delta_time = time.time() - self.prev_time
self.prev_time = time.time()

# PID calculation
P = tar - pos # (+) need more left speed, (-) need to more right speed
I = (self.sum_error + P) * delta_time if self.prev_time != -1 else 0
D = (P - self.prev_error) / delta_time if self.prev_time != -1 else 0

# Sum error and mem prev error
self.sum_error += P
self.prev_error = P
```

```python
def pid_forward(l: int, r: int, Max_speed: int, p: float, i: float, d: float) -> [int, int]:

    pid = p + i + d

    L_speed = (l) + pid
    R_speed = (r) - pid

    L_speed = abs(L_speed) if abs(L_speed) <= Max_speed else Max_speed
    L_speed = L_speed if L_speed > 0 else 0

    R_speed = abs(R_speed) if abs(R_speed) <= Max_speed else Max_speed
    R_speed = R_speed if R_speed > 0 else 0

    return L_speed, R_speed
```

API and Communication protocol:

With only a single Raspberry PI 4B, it cannot handle all of the operations required to drive the robot due to the limitation of available pins and computational power. To solve this problem, we decided to separate each service into multiple microcontrollers and microprocessors in order to reduce the workload of Raspberry PI. There are 2 communication protocols used in this project.

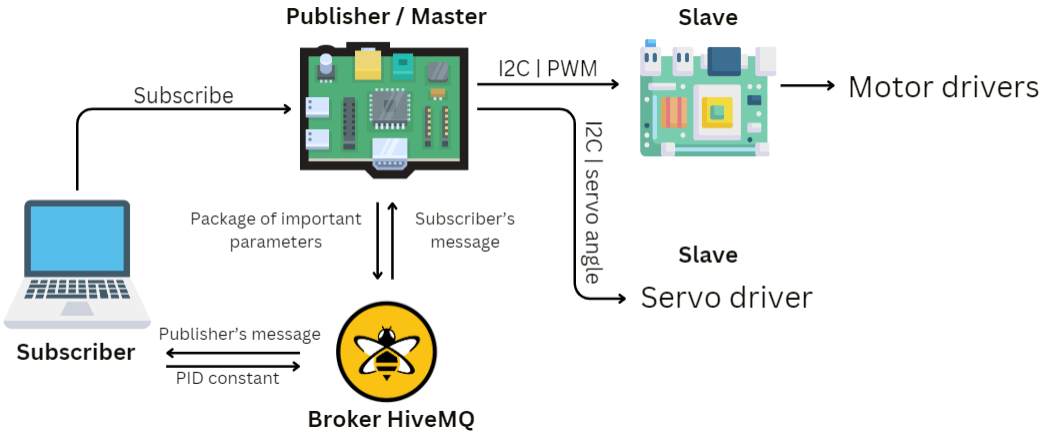
1. **I2C:**

   This protocol allows us to have a fast data transmission channel between Raspberry PI and Arduino using only 2 wires connection. We setted Raspberry PI as a master. Arduino and Servo driver as slaves. The data from the sensors was used to calculate the speed of the motors and direction of the servo motors in PI, then were sent to Arduino to control the H-bridge motor driver and Servo driver to determine the direction of each servo motor.

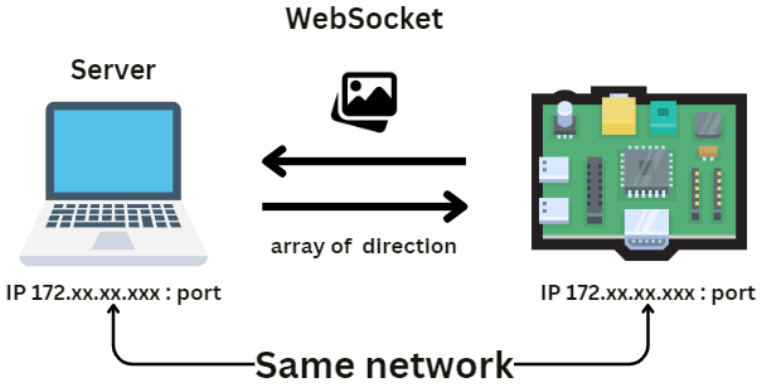
2. **MQTT and Websocket:**

   These 2 APIs were used for different purposes. The MQTT was used for real time monitoring and adjusting the parameters of the robot. In our case, Raspberry PI is the publisher who publishes important parameters, such as PWM, servo angle and yaw from Kalman filter to any subscriber (the dev's laptop in this case). The subscriber can observe those parameters and wirelessly adjust the PID constants in real time thanks to the MQTT. We decided to use the MQTT because

of its abilities to transmit data in the unreliable network (Failure is acceptable), consume less memory and computational power, and quickly transfer a small package of data.



The Websocket was used for transferring the delicate data, image in our case. Websocket is more stable (Failure is not acceptable) and faster when we have to transfer a big package of data. The image will be feeded through the segmentation model in the server side, then send the data to PI or client side.



I. Implement a Kalman filter for sensor fusion(data fusion) of gyro sensor and magnetometer for yaw position calibration and giving the position for Ackerman turning .

II. The Yaw position also feeds into the PID controller for robot movement speed controlling.

III. By creating a robot.py file for initial basic movement use for path planning in main file(main.py) which would handle all the function utilities from other sub-files that would be called on to and feed the data in.

IV. On the sensors directory it provides all the necessity on sensor reading and calibration as well as motor control, for example we choose to manually access all the sensors using Smbus2 python library for more concrete and predictable output as we want.

V. We also chose I2C to communicate between raspberry-pi and arduino mega, which used to control the motor drivers separately from servo. Because of that we need a method to communicate between them.

VI. I2C has advantages on speed of communication and it was relatively simple to implement and we can ensure that it would work in an environment that we want which requires the most lees wire for communication possible.

Kalman Filter

```
def kalmanfilter(predictUncertainty , kalmanPredict, kalmanMeasurement):
    # Predict the current State
    #kalmanState = (0.5 * kalmanPredict ) % 360
    #print(kalmanInput)

    # Calculate the uncertainty
    # predictUncertainty = predictUncertainty + 0.5 * 0.5 * 2 * 2  # kalmanUncertainty = uncertaintyPrevious + deltaTime ** 2 *

    # Calculate Kalman Gain
    kalmanGain = predictUncertainty / (predictUncertainty + 6 * 6)  # is the variance angle of magnetometer

    # Angle wrapping
    error = angleDifference(kalmanMeasurement, kalmanPredict)

    # print("Error", error)
    # Update the predicted state and wrap it to [0, 360) range
    kalmanPredict = (kalmanPredict + (kalmanGain * error)) % 360

    # Update uncertainty
    predictUncertainty = ((1 - kalmanGain) * predictUncertainty) +(0.0001)

    return kalmanPredict, predictUncertainty
```

```
class Kalman_Yaw:
    def __init__(self):
        self.gyro = GyroscopeSensor(x, y, z)
        self.offset = runMag()
        self.reset = False

    def get_yaw(self):
        x_anglularRate, y_angle, z_angle = self.gyro.runGyro()
        initialKalmanPredict = (0.2 * x_anglularRate) % 360
        magAngle = runMag() - self.offset
        if magAngle < 0:
            yawAngleMag = 360 + magAngle
        else:
            yawAngleMag = magAngle

        optimalYawAngle, uncertainty = kalmanfilter(kalmanUncertaintyAngle, initialKalmanPredict, yawAngleMag)

        print("YAW: ", optimalYawAngle)
        return optimalYawAngle

    def re_initialize(self):
        del self.gyro
        del self.offset

        self.gyro = GyroscopeSensor(x, y, z)
        self.offset = runMag()
```

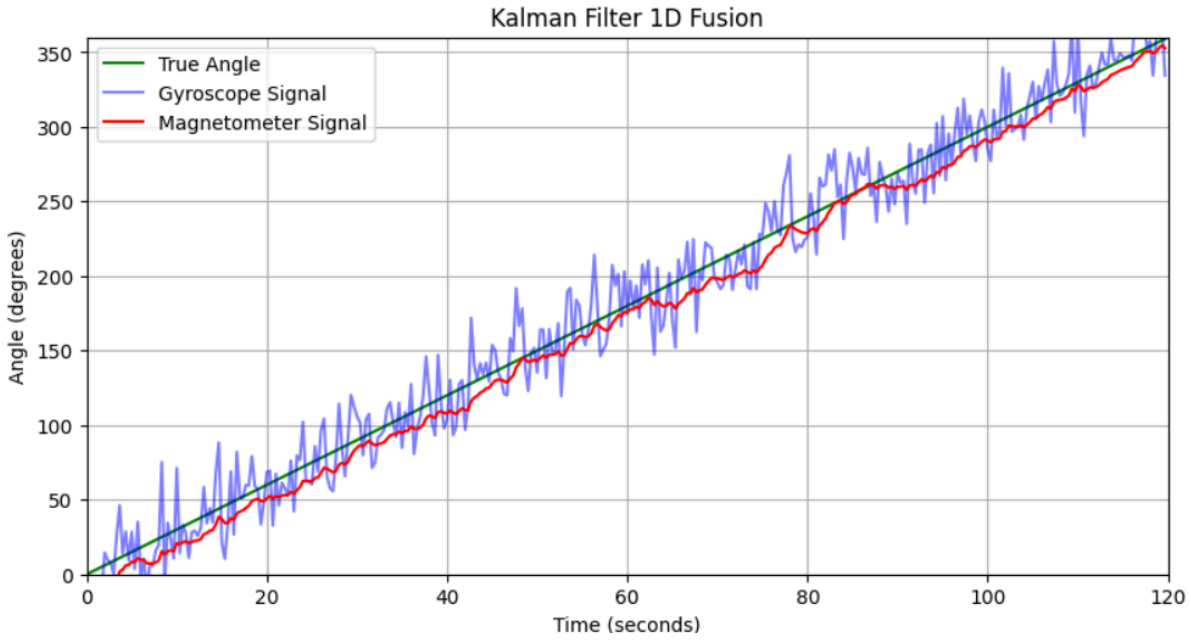
4. Test Results

1. Overall result of our robot

- In the given video below, you can see the robot is moving straight because of the pid controller.
- The turning mechanism calculated from Kalman filtered optimal degree and ackerman steering also works as expected.

Youtube Link – https://youtu.be/87p56Rra9FQ?si=ri9ly3AmlpDKeXMY

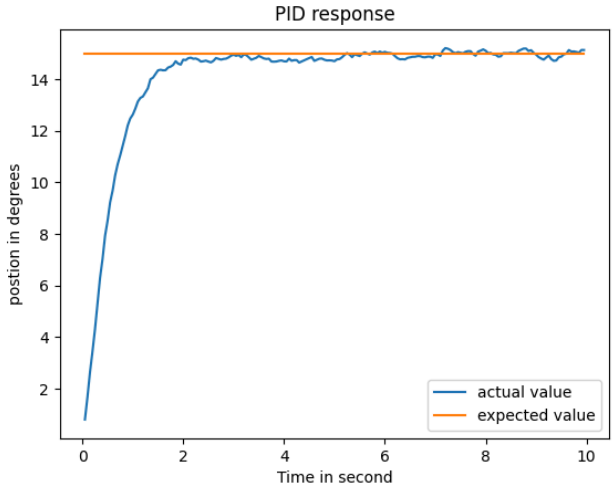
2. Kalman filter



With Kalman filtering, the result of orientation(Degree) is very precise with only $\pm$ 2 degrees of variation in each test.

3. PID controller

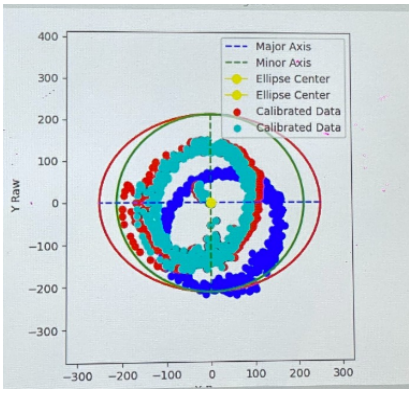
The PID controlling worked as intended although the problem is more likely to be with the power of the motor that it cannot pass through the tall grass section as the encoder-motors have too low of a power to push through it.



4. Sensor Calibration

Magnetometer

```
soft_factor = np.array([[1, 0], [0, 0.84465789]])
x_raw_bias = 53.148
y_raw_bias = -48.19
```



- This is the transformation matrix and bias value to get the best result angle measurement of magnetometer

5. Conclusion

Concluding Remarks

In conclusion this project expanded our expertise in many ways for example, in using the Kalman Filter for filtering noise and errors of the data from sensors, and also fusion the data together to get the true position of the robots. And we have gained the experience of working as a large team from the start. Finally we can learn to adapt and adjust the spec of our robots to suit our budgets.

Suggestions

1. Changing the navigation method for the robot for example instead of using time intervals for a predicted routine we can change to using a SLAM map and PURE PURSUIT method for completely automating navigation. In that way we can just use the waypoint from the map and let it handle the path plan by itself.
2. Changing the structure for the rover skeletons that used PVC pipes as a structure support, Due to the overall weight of the robot is higher than expected can it's the way to cut the cost of the robot. If possible we could use metal pipes instead of PVC pipes.

Project Code Github :
https://github.com/ptkoo/leave_collector_robot.git

Member Participation

Name	Student ID	Purchase Item	Design Thinking	Design	3D Print	Assembly	Circuit Design	Circuit Implementation	Wiring	Sensor Calibration	Camera and AI	Programming	Testing	Report Making	Percentage	Percentage Bar
Paing Thet Ko	65011416	☐	☑	☑	☐	☑	☐	☑	☑	☑	☐	☑	☑	☑	70%	
Pyae Htoo Khant	65011497	☑	☑	☑	☐	☑	☐	☑	☑	☑	☐	☑	☑	☑	77%	
Rachata Raktham	65011500	☑	☑	☑	☐	☑	☐	☑	☑	☐	☐	☐	☑	☐	54%	
Sittipon Kumda	65011539	☐	☑	☐	☐	☐	☐	☐	☐	☑	☑	☑	☑	☑	47%	
Sornpawee Jukmongkol	65011546	☑	☑	☑	☑	☑	☐	☐	☐	☐	☐	☐	☑	☐	47%	
Suriyarat Woraumponkul	65011557	☑	☑	☐	☐	☑	☐	☐	☑	☐	☐	☐	☐	☐	31%	
Tanakorn Youngmeesuk	65011563	☐	☑	☐	☐	☐	☐	☐	☐	☐	☑	☑	☑	☑	39%	
Thitiphan Chenrukmatupoom	65011598	☑	☑	☑	☐	☑	☐	☐	☑	☐	☐	☐	☑	☑	54%	
Swan Pyae Sone	65011683	☑	☑	☑	☑	☑	☐	☑	☑	☐	☐	☐	☑	☑	70%	
Thura Zaw	65011685	☑	☑	☑	☑	☐	☑	☑	☑	☐	☐	☐	☑	☑	70%	