# TABLE OF CONETENTS

# 1. Team introduction

- Our team consists of two Form Four students,Thomas and Kate. This is our first time participating in this competition. Through this experience, we hope to apply theoretical knowledge in a practical setting, fostering creativity and innovation. Our ultimate goal is to design and build a robot that meets the competition requirements while showcasing our skills and dedication.

# 2. Mobility

- Motor Selection & Justification
  Since we are new to this competition and we lack preparation time, our design principle is to be convenient and powerful. Therefore, we mainly use the LEGO® MINDSTORMS® Education EV3 Core Set to assemble our robot. We prefer EV3 over Arduino due to its convenience and ease of use. Our vehicle uses two Lego EV3 medium motors, selected for their balance of speed (240-250 RPM) and torque (8 N·cm running torque). These motors are ideal for:

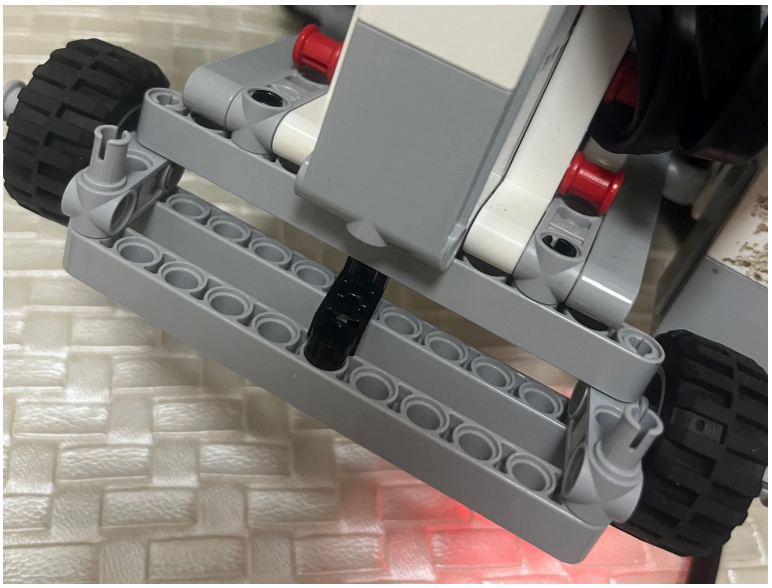
- Agile Movement: Medium motors provide sufficient acceleration for indoor navigation while maintaining precision.
- Compact Integration:Fits the lightweight Lego Technic chassis without overloading the structure.

- Torque Validation:
  Calculations confirm the motors handle the robot's weight (~ 750g):
  Torque Required= Mass x Gravity x Wheel Radius x Friction Coefficient
  = 0.75 kg x 9.8 m/s² × 0.028 m x 0.3 ≈ 0.062 Nm (6.2 N.cm)
  The 8 N·cm running torque provides a 22.5% safety margin.

- Chassis Design

  -Differential System
  Our chassis incorporates a differential Gear part for advanced mobility. It enables Independent Wheel Control

  -Front Steering System Advantages
  1. Hight speed:
     - Achieves full 45° turns in 0.3 seconds
     - Direct 3-bar linkage eliminates gear train delays (figure 2.1)

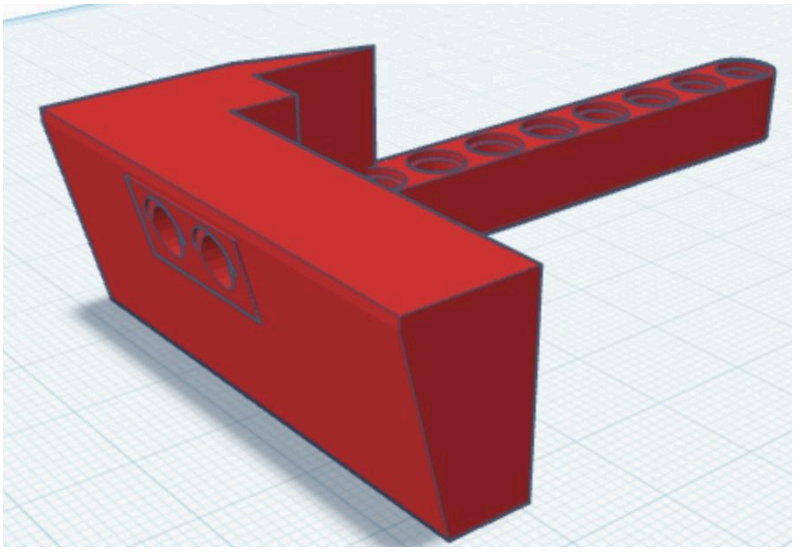


(figure 2.1)

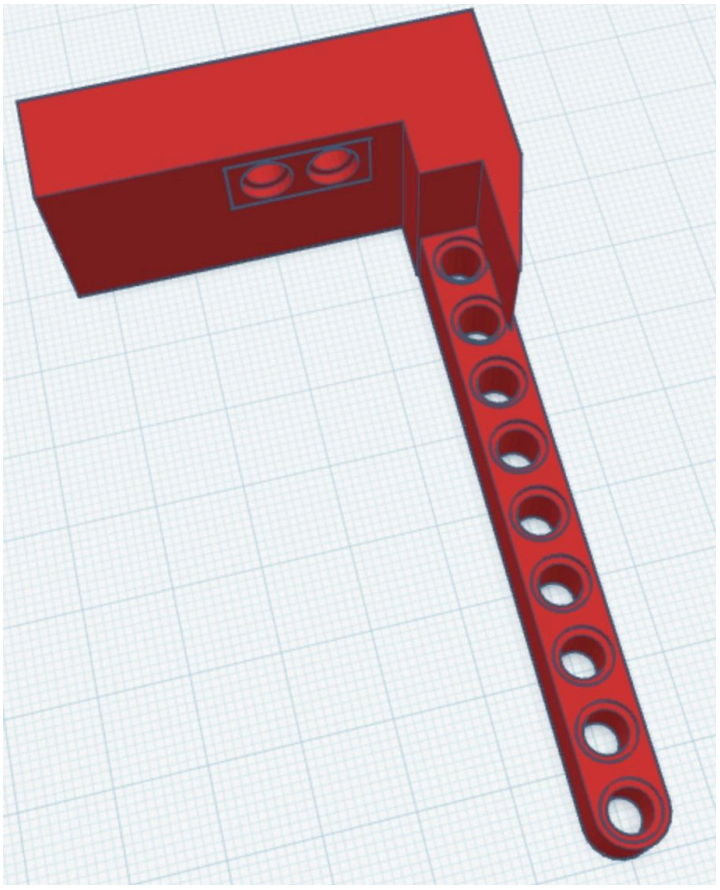
  2. Accurate and no wheel slippage thanks to Instant torque application no center-point alignment design

- Component Mounting
  - Sensor Placement:
  - Color Sensor:Mounted 4mm from the ground for accurate line detection.
  - Ultrasonic sensor: left hand side of the robot to detect the wall

- Camera: Elevated on a 3d-printed arm for adjustable viewing angles.  (The angle of elevation: 75°)(figure 2.2 and 2.3)



(figure 2.2)



(figure 2.3)

# 3. Power and sense management

- Power Source & Distribution
  All electron components are connected by Lego $I^2C$ connection.
  The vehicle is powered by the Lego EV3 rechargeable battery (2050 mAh), selected for:
  - Stable Voltage Output: Ensures consistent sensor/motor performance.
  - Modularity:Quick-swappable design for extended operation.
  - Efficiency: Integrated voltage regulation minimizes power spikes.

- Power Budget Analysis:

| component | quantity | estimated avg. current (mA) | estimated total (mA) |
|---|---|---|---|
| EV3 Medium Motor | 2 | 300 | 600 |
| Color sensor | 1 | 50 | 50 |
| ultrasonic sensor | 1 | 60 | 60 |
| gyro sensor | 1 | 40 | 40 |
| camera | 1 | 70 | 70 |
| ev3 brick | 1 | 150 | 150 |
| Total | | | 970 |

Theoretical runtime: ~2.11 hours (2050 mAh ÷ 970 mA).

We use 4 sensors in total, including 1 ultrasonic sensor, 1 gyro sensor, 1 colour sensor and one pixy2 camera. We choose to use the 3 sensors from the lego ev3 set as it is easy to manage and use. We use the pixy2 camera as it has a powerful built-in colour object detection function. We use the ultrasonic sensor and gyro sensor to get the actual distance from the wall. Moreover, a gyro sensor is used to get the heading of the vehicle and enhance the turning performance. Also, we use the colour sensor for detecting the blue and orange line markings, we mounted it low to avoid ambient light interference  Lastly, pixy2 camera is used to detect the traffic signs and get its centre coordinates.

# 4. Obstacle  Management

We use 4 sensors in total, one ultrasonic sensor,one gyro sensor,one colour sensor and one Pixy2 camera.

For Open Challenge, our principle is to make everything simple to minimize the effort we need to put into and easier for our maintenance. Therefore, we didn't use the Pixy2 camera. We use 2 P-control algorithms,ultrasonic sensor and gyro sensor to ensure the vehicle has a consistent distance. Moreover, we use a colour sensor to detect blue and orange lines for turning.

For the P-control, we use an ultrasonic sensor to detect the distance between the vehicle and the right wall of the vehicle. We get the current heading by a gyro sensor. As the heading is not consistent during the round, we calculate the actual distance by cos(heading)*detected distance(figure 4.1). After having actual distance and the heading, we use 2 P-control algorithms to ensure the vehicle has a consistent distance. We calculate the target heading by inputting actual distance (figure 4.2). Moreover, to prevent losing the moving direction, we set the heading limit between -15 to +15 degree(figure 4.3). Then, we calculate target steering motor position by inputting target heading(figure 4.4). Furthermore, to prevent the front wheel in contact with other parts of the vehicle which interrupt the forward movement of the vehicle, we set the steering motor position to -45 to +45 unit(figure 4.5).

```
def getWallDistance(detectDistance,currentHeading):
    currentHeading = float(currentHeading)
    detectDistance = float(detectDistance)
    return float(cos(abs(currentHeading/180*pi)) * detectDistance)
```

(figure 4.1)

```
102     def distanceCorrection(targetDistance):
    targetDistance = float(targetDistance)
    currentDistance = getWallDistance(horDis.distance_centimeters,getHeading(gyro.angle))
    distanceError = float(targetDistance - currentDistance)
    if (abs(distanceError) > 50):
        distanceError = 0
    elif(abs(distanceError)<2):
        distanceError = 0
    elif (distanceError > 0):
        distanceError = distanceError * 4
    distance_output = float(distanceError * distance_kp)
    # distance_output = targetHeading
    headingCorrect(distance_output)
```

(figure 4.2)

```
if(getHeading(gyro.angle)>15 and targetHeadingCorrection <0):
    return int(15)
elif(getHeading(gyro.angle)<-15 and targetHeadingCorrection >0):
    return int(-15)
```

(figure 4.3)

```
def headingCorrect(targetHeading):
    targetHeading = float(targetHeading)
    headingError = float(targetHeading - getHeading(gyro.angle))
    heading_output = headingLimit(headingError)
    heading_output = float(heading_output * heading_kp)

    steeringMotor.on_to_position(SpeedPercent(100),int(heading_output + steeringInit ))
    print("heading error,output",headingError,heading_output)
```

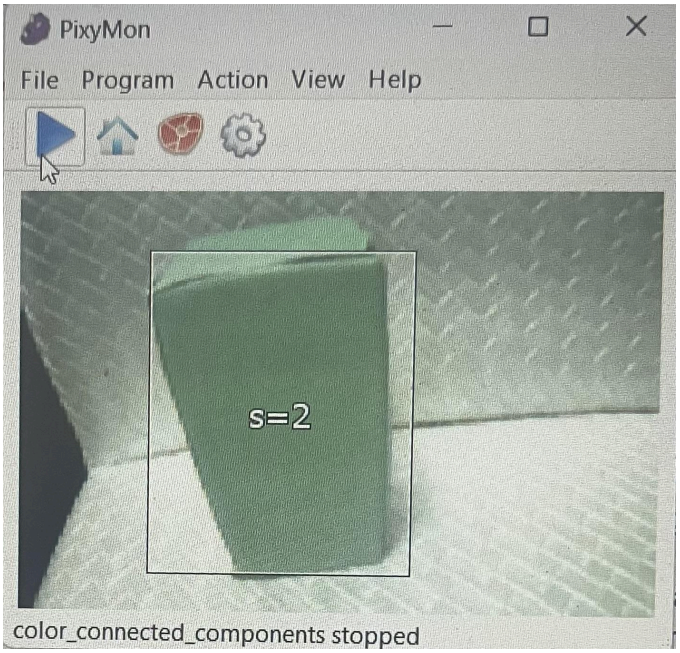
(figure 4.4)

```
    elif(abs(targetHeadingCorrection) >45):
        return int(targetHeadingCorrection / abs(targetHeadingCorrection) * 45)
```

(figure 4.5)

When the colour sensor detects the blue line first, the vehicle will turn left and run a forward direction part of the program after selection, vice versa. After turning once, we will add our moved sessions variable by 1. After the moved sessions reach 12, we will stop at the next session.

For the Obstacle Challenge, we use all four sensors. We use a Pixy2 camera built-in function which is colour detection(figure 4.6) to detect the traffic signs as it is easy for both training and access. We calculate the centre of the traffic signs and compare it to the middle of the camera. After having the distance difference, we calculate the target heading and targeting to put green traffic lights to the left of the camera, vice versa. After moving across the traffic lights, we set our target heading to 0. After that, we detect the blue and orange lines, and we set our vehicle to turning mode. We use the same method in Open Challenge to determine when our vehicle should be stopped.



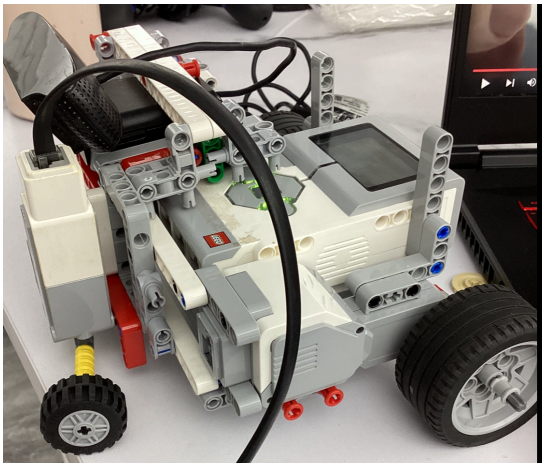

(figure 4.6)

# 5. Design Journey and Difficulties

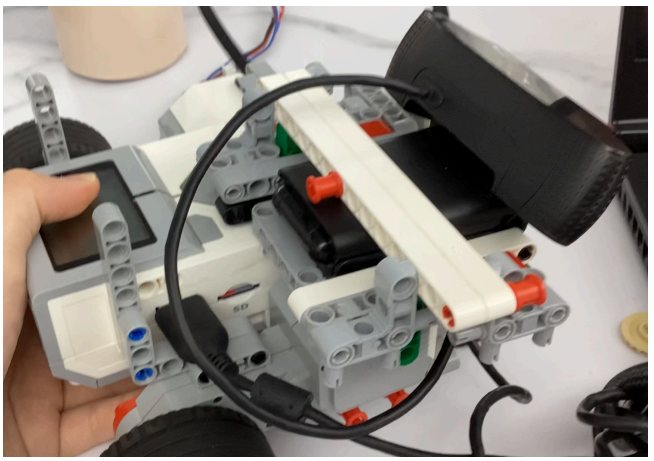
The first obstacle we overcome is choosing a camera for detecting the traffic signs. As we have experience with writing vision with OpenCV and C720 logitech cameras(figure 5.1). We tried to use C720 on EV3 at first. Although there are some methods to connect C720 to EV3 online, we still cannot connect the camera after using the online methods. Therefore, we move on to choose another camera that is compatible with EV3. We chose to buy a pixy2(figure 5.2) camera at last as it is very user-friendly and has a strong built-in function, colour object detection which can perfectly detect the traffic signs. After using the pixy2 camera, it worked very well for us. Moreover, we didn't notice that the drive wheels must be physically connected and are not allowed to use one motor per side, so we changed our design to be a 4 wheeled vehicle with one driving axle and one steering actuator. Figure 5.3 and 5.4 are our original design.


(figure 5.1)

(figure 5.2)



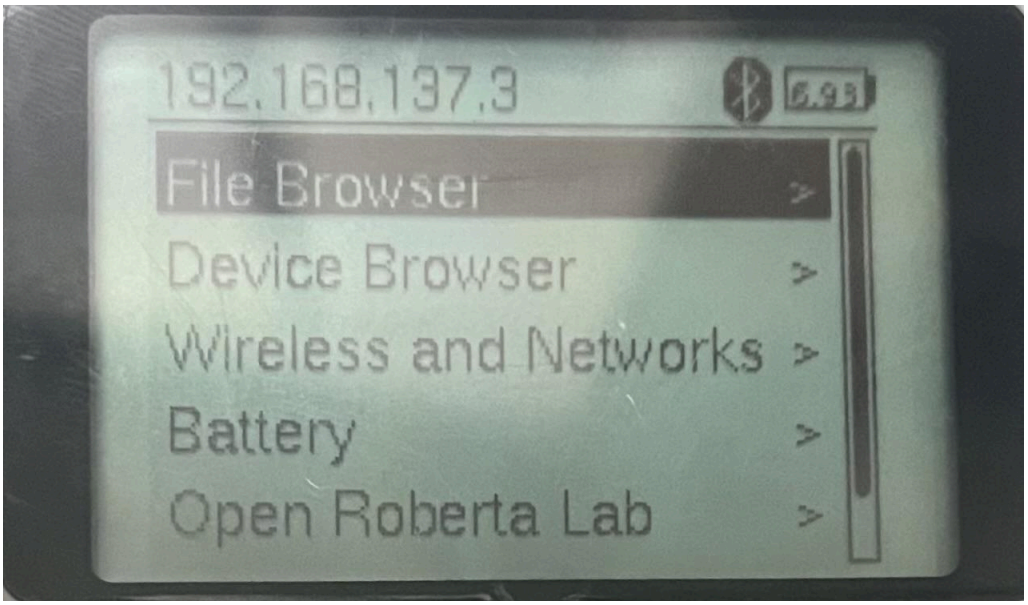
(figure 5.3)



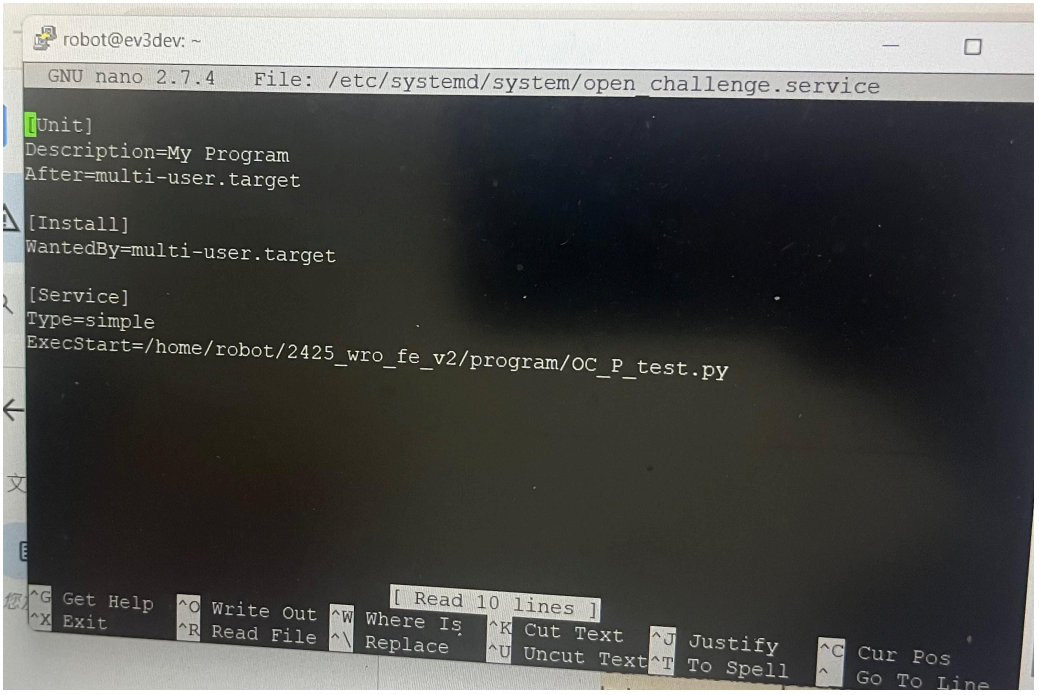
(figure 5.4)

The second obstacle we faced is to meet the starting game rule(9.11). At the very beginning, we chose to use a Debian Linux-based operating system called ev3dev rather than the original system in EV3. ev3dev provides us more power hardware and software environment as it can increase the cpu Hz up to 456 Hz. Moreover, we can write code in python which lets us have better maintenance as python is much more concise than block language. However, ev3dev required more than one button to start the last program run on the EV3 as we needed to click into the file browser to run a program in its home page(figure 5.5). In view of this, our final solution is to add a document that will be run during the start of EV3 by ev3dev(figure 5.6). The document will automatically run our saved program with a saved path after starting the EV3 and wait for pressing the right arrow button

.

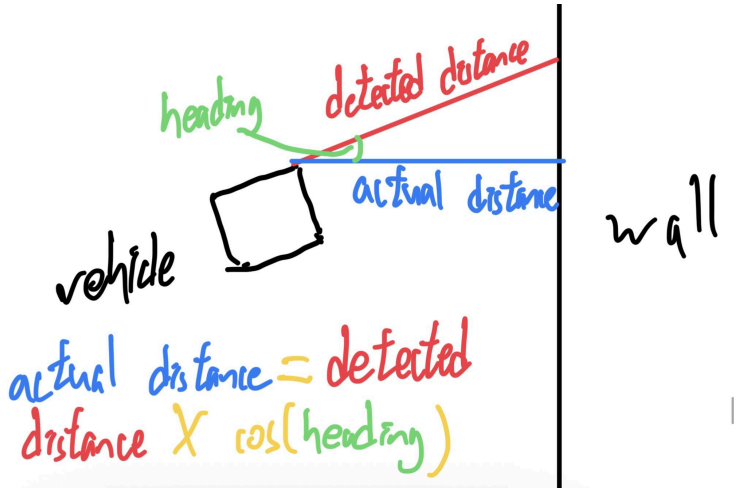



(figure 5.5)




(figure 5.6)

The third difficulty we faced is getting the actual distance with the wall. At first,we just used the reading of the ultrasonic sensor. However, we found that the reading is quite far from its actual distance and its heading keeping changing, Therefore, we draw a diagram between the wall and vehicle(figure 5.7), found that the actual distance is equal to the detected distance * cos(heading)




.

(figure 5.7)