Table of Contents

Table of Figures :

1 Abstract :

Our project is simply a robot that detects human face by camera module, consequently it stops, actuating a motor that ejects a flyer that is meant to provide info or advertise whatever organization. It has a proximity sensor that detects any obstacle and stops avoiding the collision and changes its direction within the pre trajected area.

The body of the robot and its appearance is inspired from the r2d2 – robot from the famous ‘Star Wars’ movie.

We implemented what we studied in the course and used Fusion 360 to design the outer body of the robot while maintaining shape details.

Figure 1 : Star Wars Robot

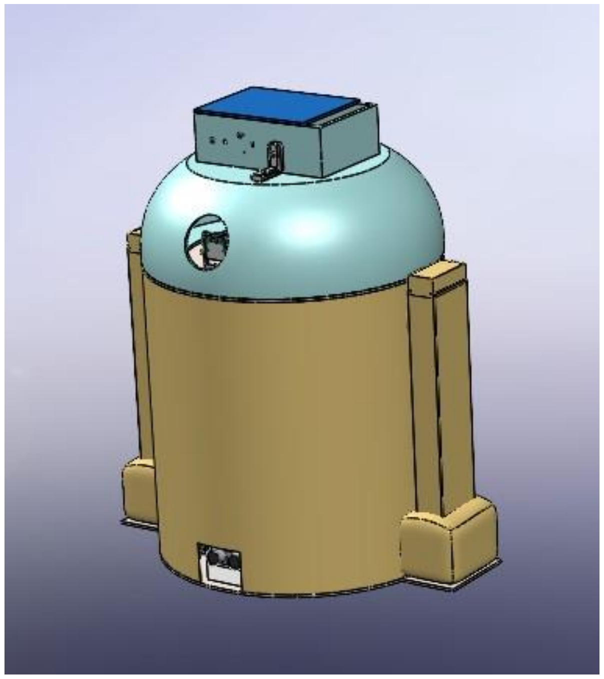


Figure 2: Robot Cad Model

2 Mechanical Design :

2.1 CAD Model :

The camera is located in the top part

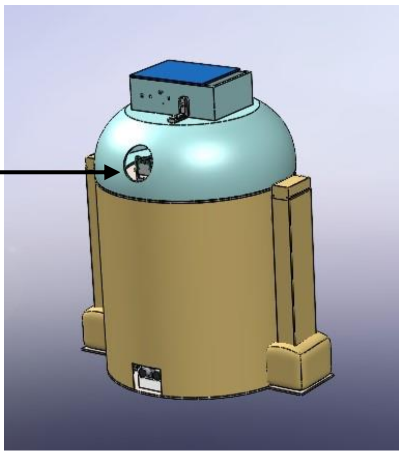


Figure 3: Camera

Top part is fixed on 4 shafts connected to the moving base part

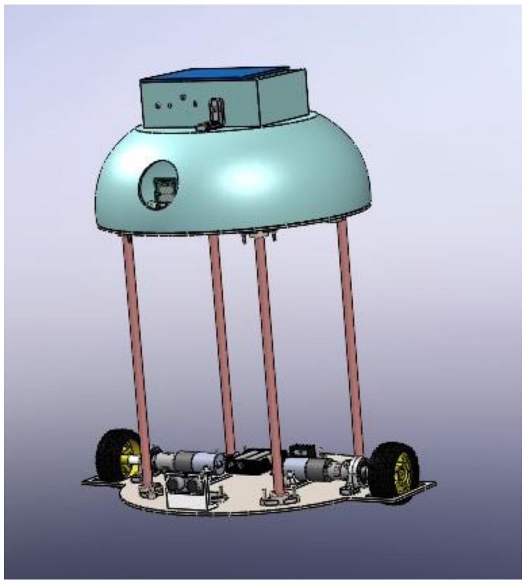


Figure 4: Robot Without Body

Top part with fixations

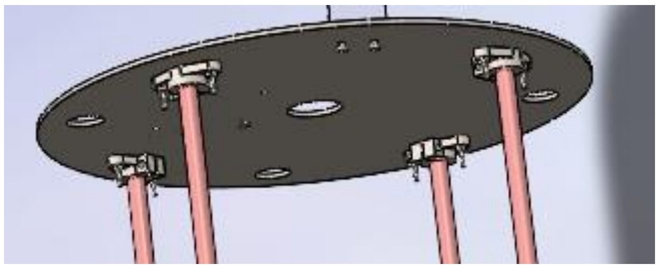


Figure 5: Upper Part

Raspberry controller fixation on top part

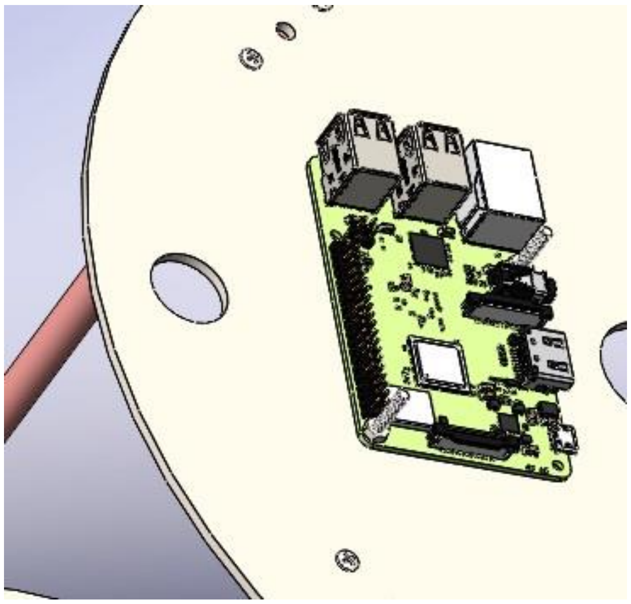


Figure 6: Raspberry pi Fixation

Storage area and pop up mechanism for flyers located over the Top part (3rd floor level)

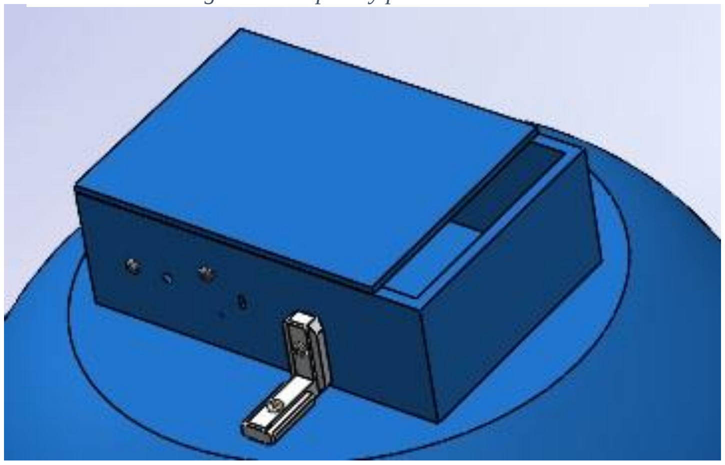


Figure 7:Upper Part

Fixations of wheels with motor (base level)

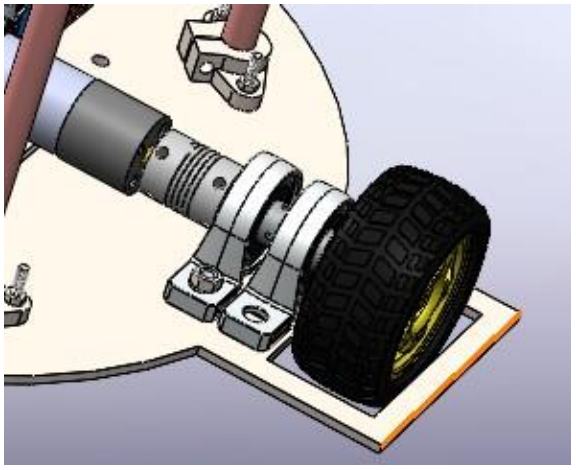


Figure 8: Bearing Fixation

Rechargeable batteries (base level)

With charging module

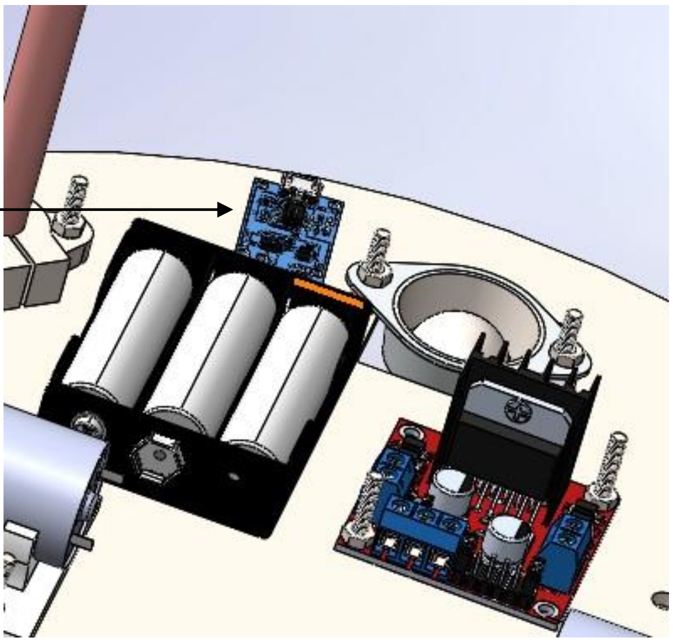
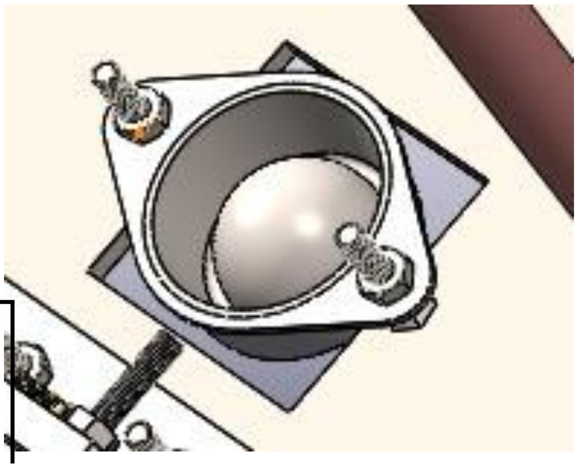


Figure 9:H-Bridge Fixation



Caster balls with spacer

to adjust level of contact points

Figure 10: Caster Wheel Fixation

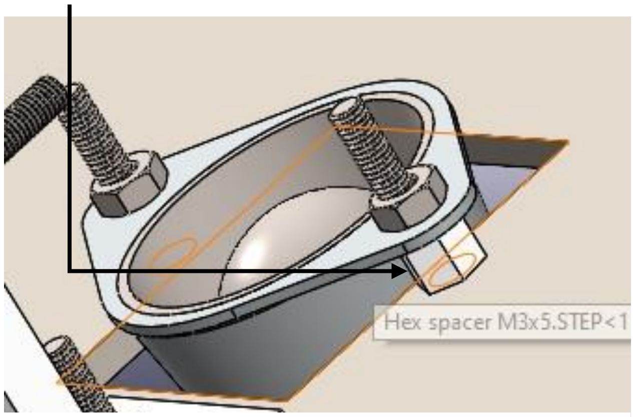


Figure 11: Spacer

Back of the robot showing input socket for charging

On/off button for the robot

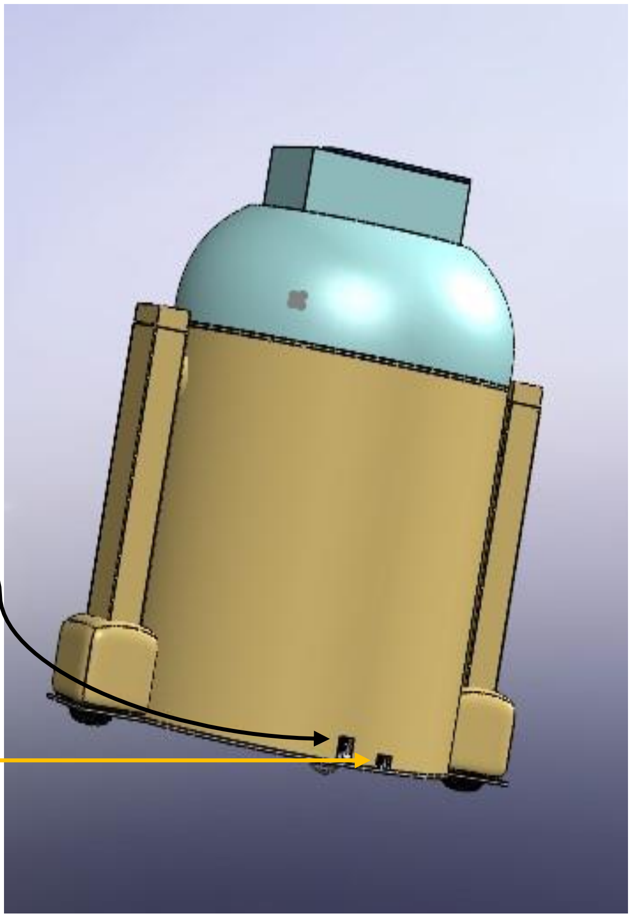


Figure 12: Charging and Switch

Motor for the flyer mechanism over top part (3rd floor)

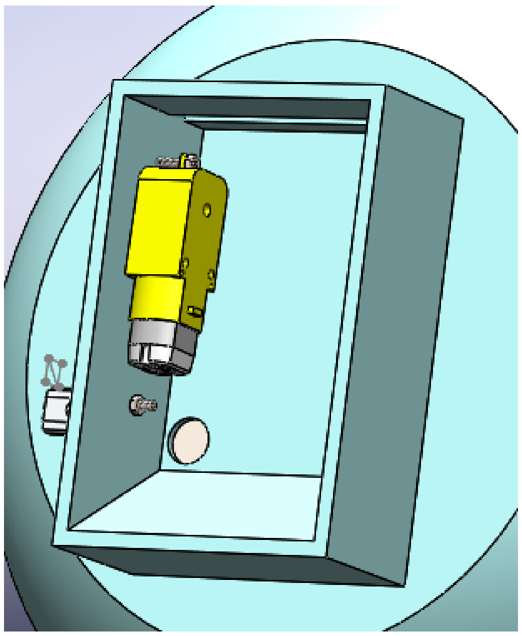


Figure 13: Upper Mechanism

2.2 Working Drawing :

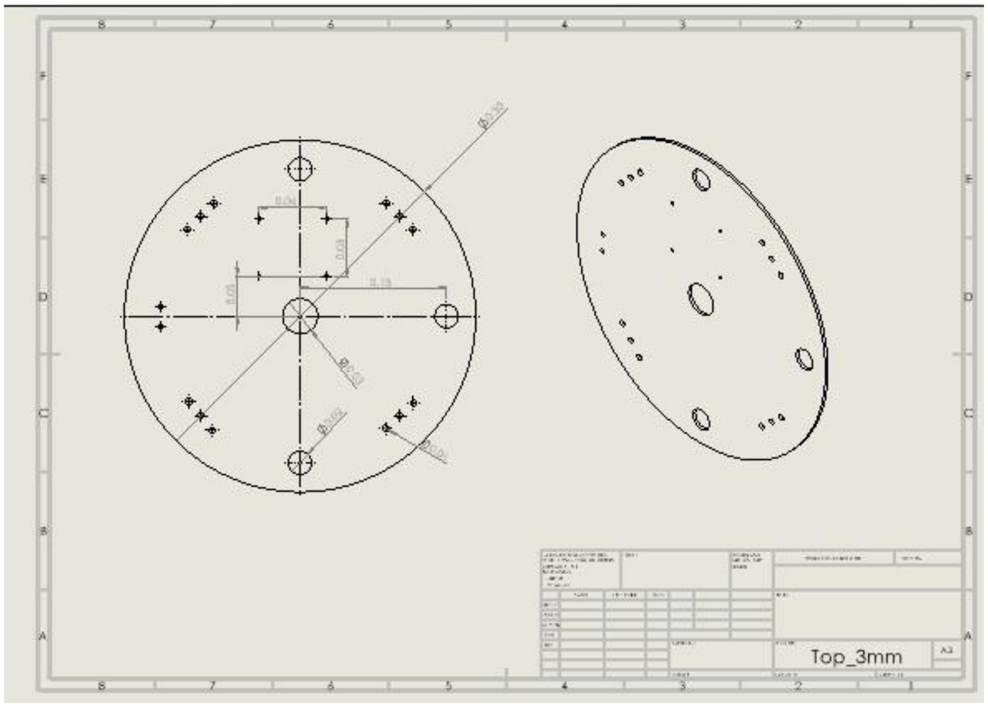


Figure 14: Top Part Drawing

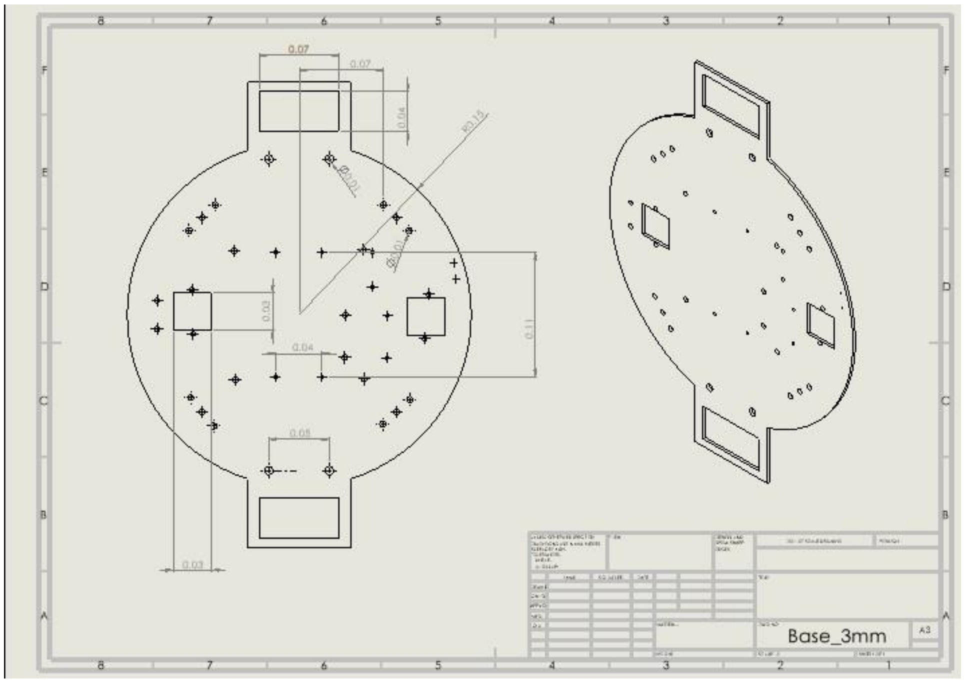


Figure 15: Bottom Part Drawing

2.3 Stress Analysis :

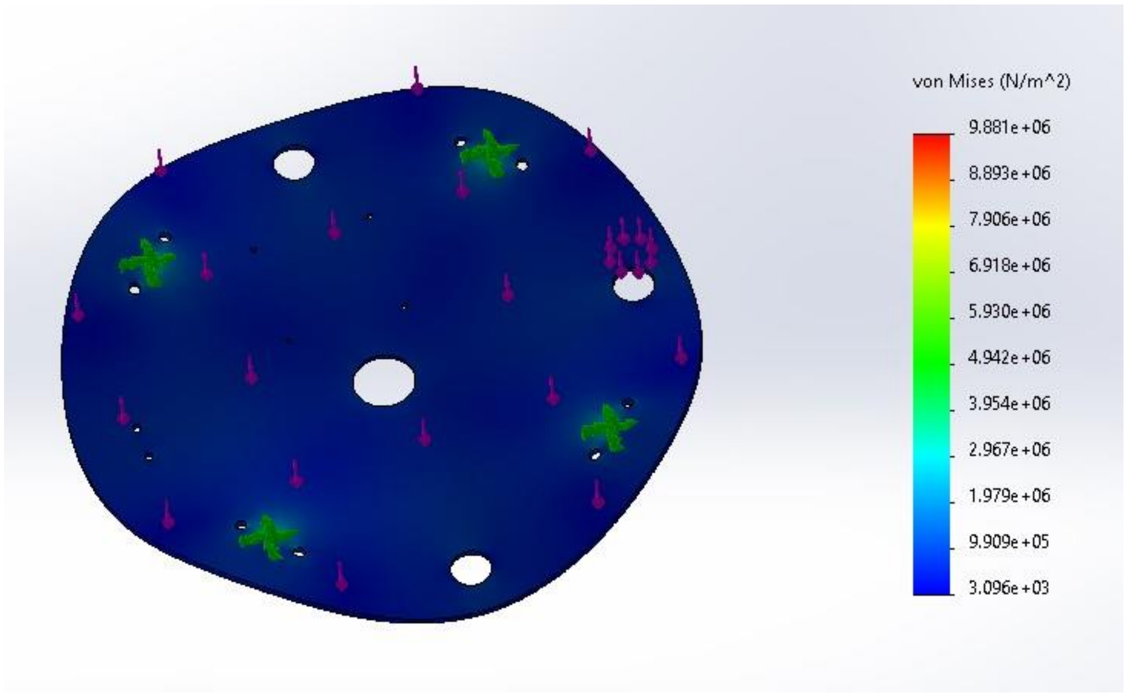


Figure 16: Upper Part Analysis

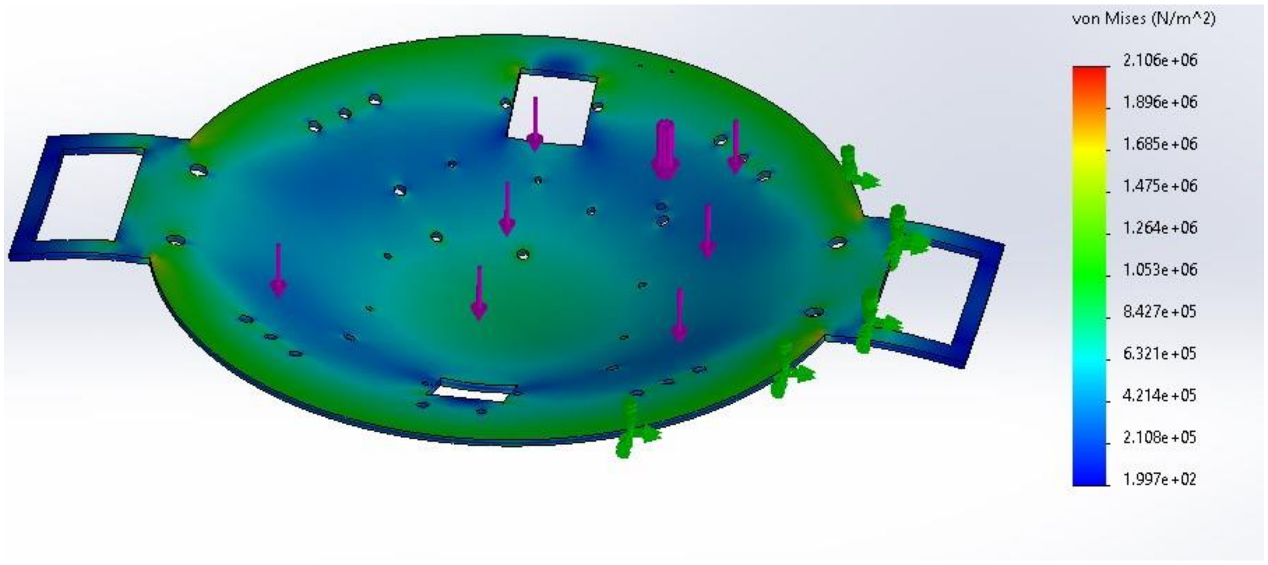


Figure 17: Bottom Part Analysis

3 Block Diagram :

Here the block diagram describes the connection of the actuators and controllers and sensors

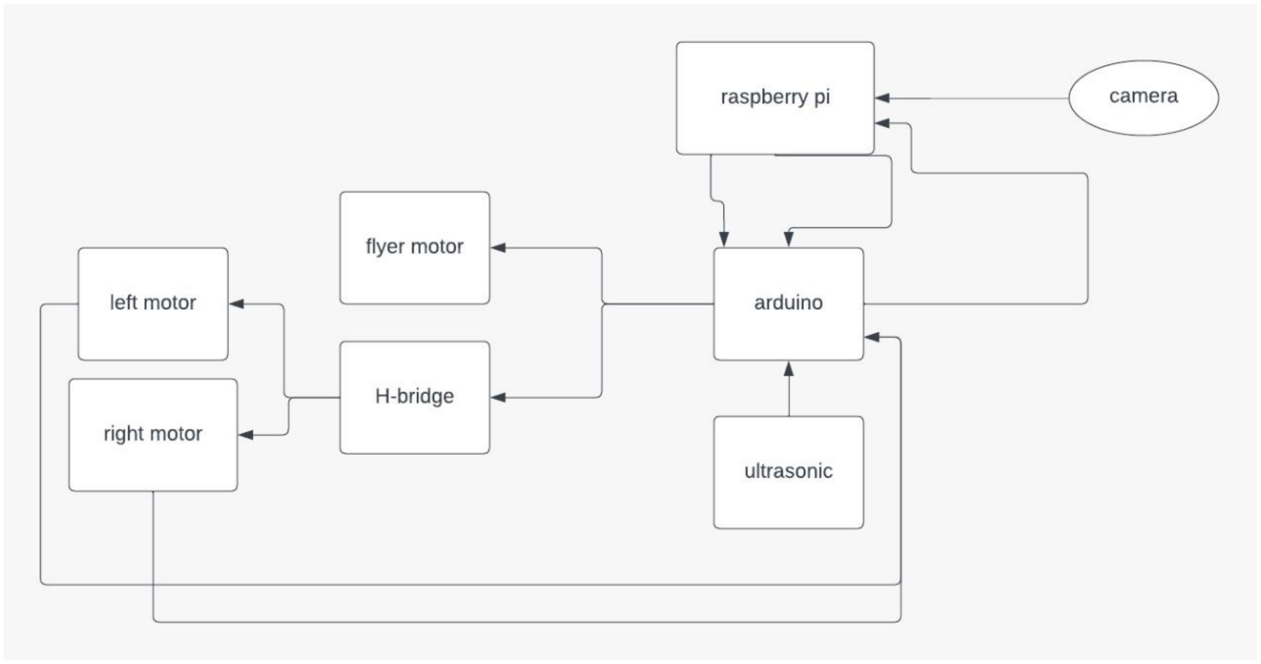


Figure 18: Block Diagram

4 Software Architecture :

Here we demonstrate the communication between each part in the robot

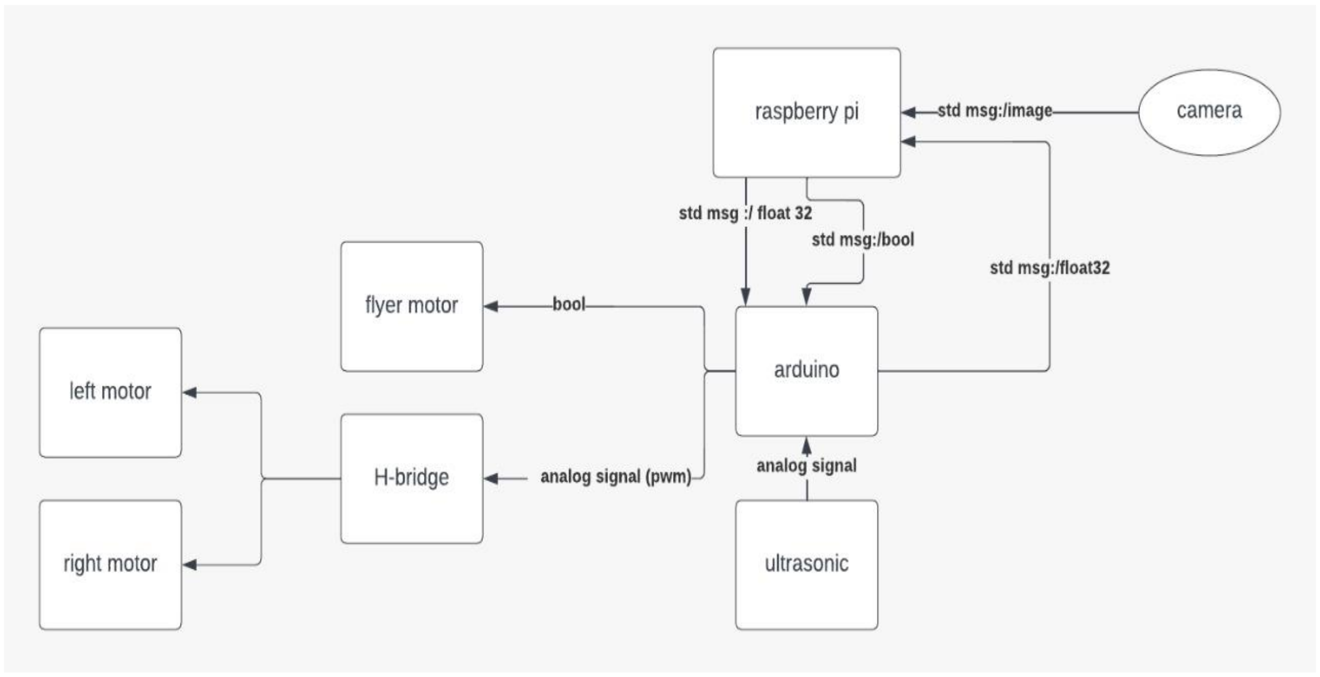


Figure 19: Software Architecture

5 Flow Chart :

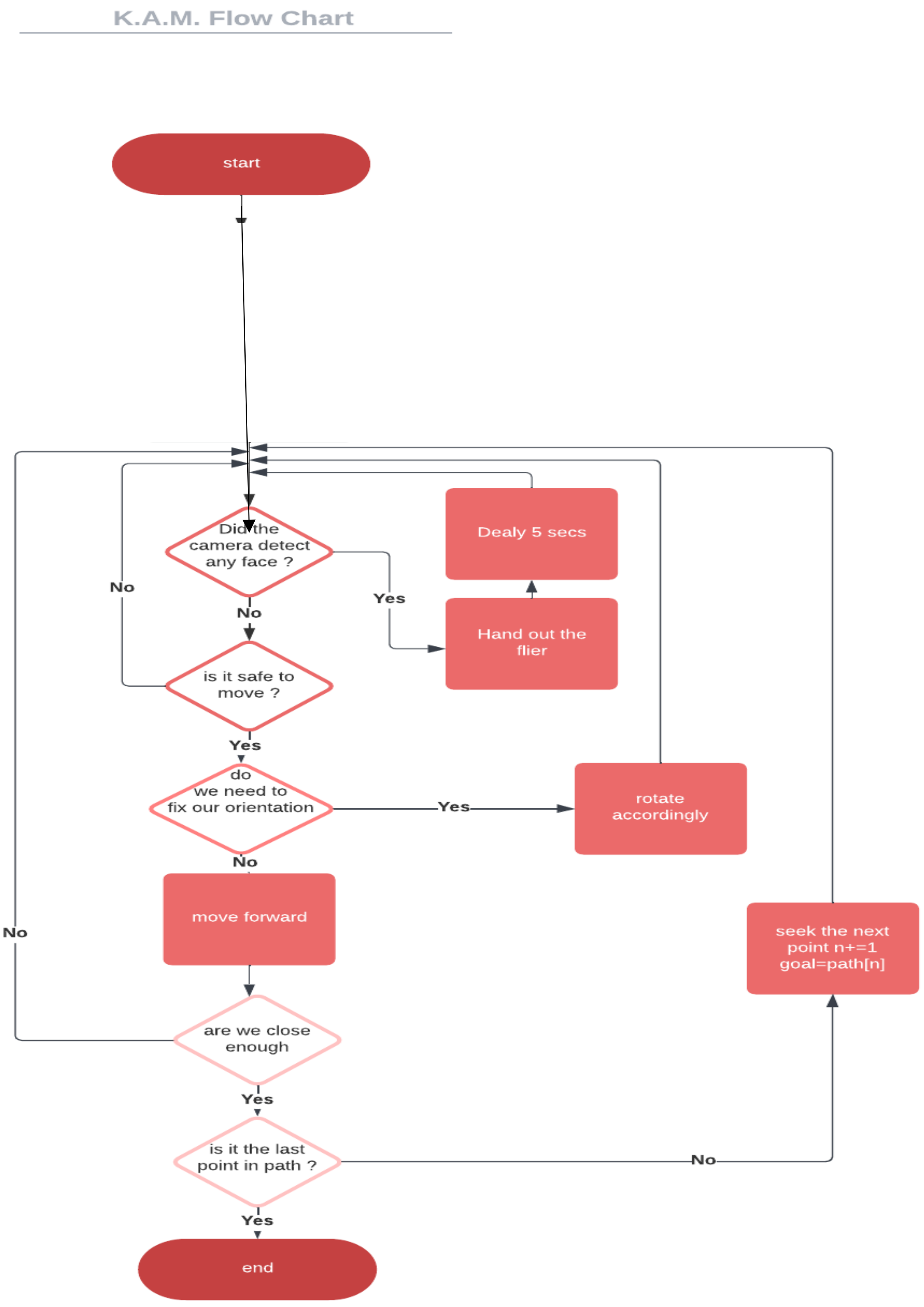


Figure 20: Flowchart

6 Actuator sizing :

We made Actuator sizing on solid works and we have load of 0.445 N.m (4.5 kg.sec) so we will use 2 DC motors of 8 kg.sec

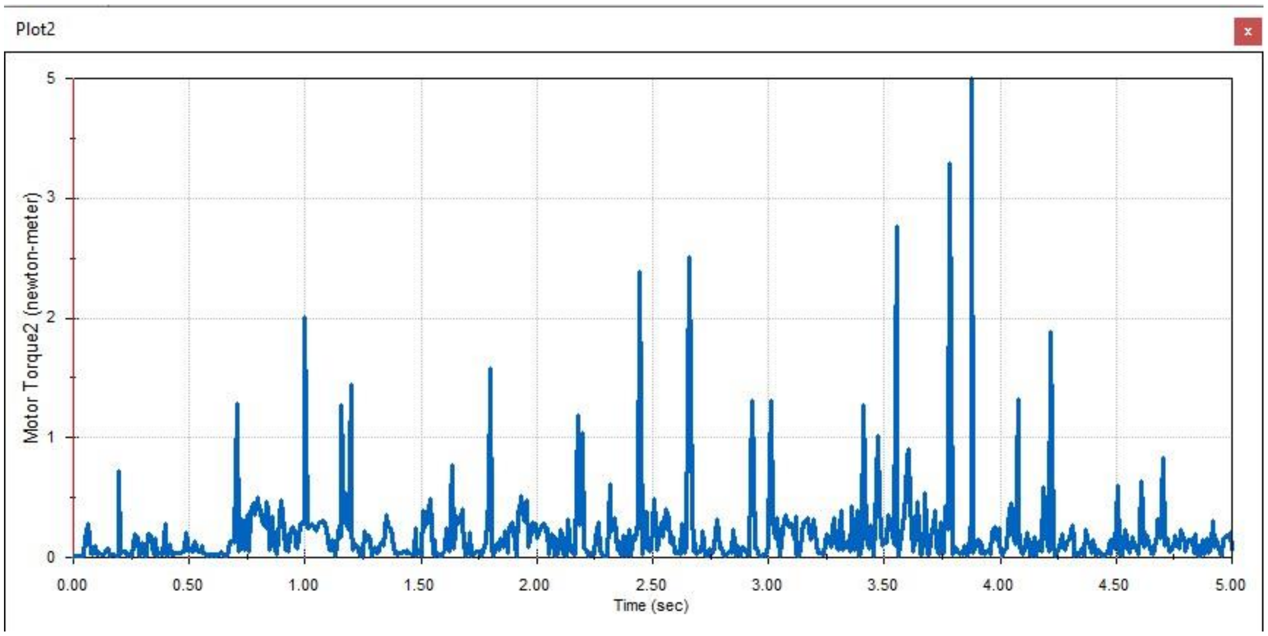
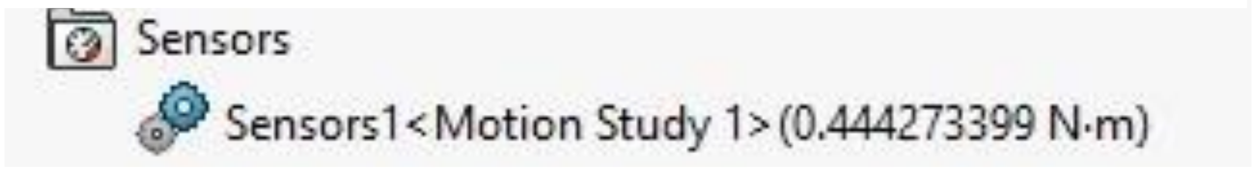


Figure 21: Actuator Sizing



Sensors
Sensors1<Motion Study 1>(0.444273399 N·m)

Figure 22: Torque Needed

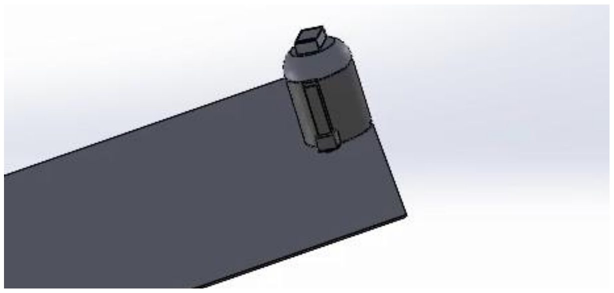


Figure 24: Start of simulation

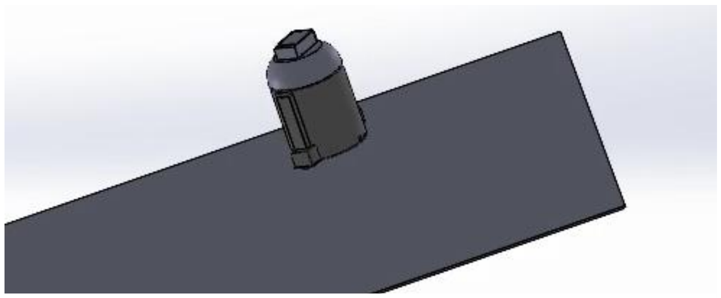


Figure 23: End of Simulation

7 Simulation :

We designed the robot to move in a path till it finds a person that is detected by the camera module then it stops and hand out a flyer, a proximity sensor that protects it from hitting anything in the path.

And we made this path by determining way trajectory.

Figure 25: Robot URDF

The first problem we faced was that the wheels was moving in opposite direction due to no exporting the URDF directly so we had to adjust the wheel joint and changed the sign of one of the wheels to be able to move directly in the same direction which was the right motor

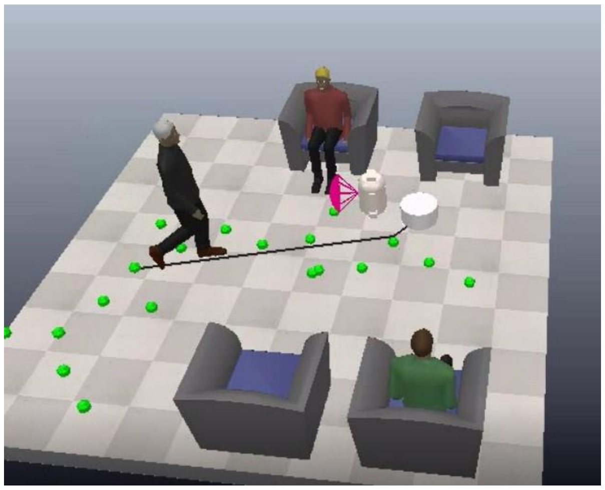


Figure 26: Simulation Environment

Then we adjusted our environment to be similar to the actual environment, so we had the robot move in a loop between the chairs

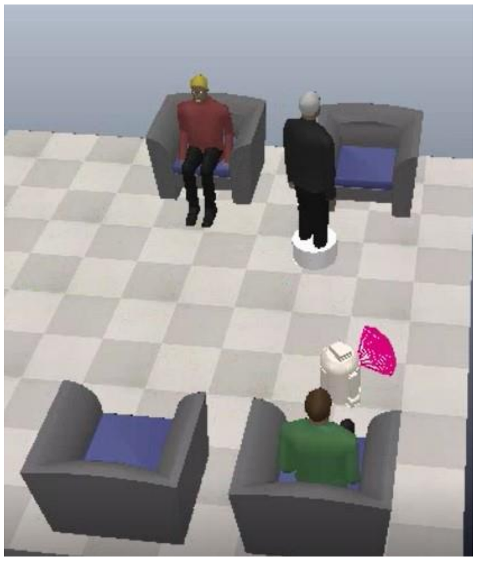


Figure 27: Simulation in Coppelia

The robot moves around the chairs in order to detect a face to hand the flyer to the client

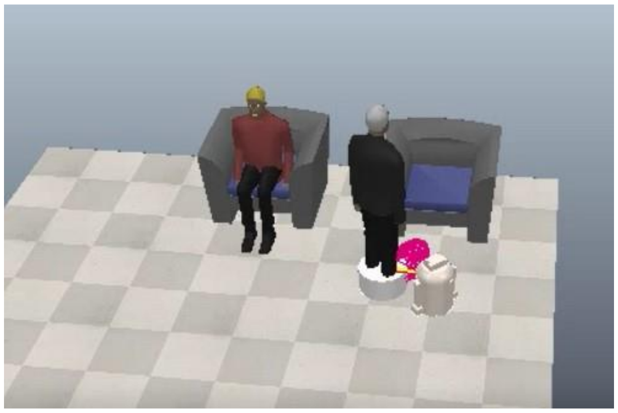


Figure 28: UltraSonic Detection

If the robot's path got blocked it will stop because of the proximity sensor in the robot and will wait till the person moves out of the way then proceed its cycle

7.1 Left Motor Code :

```
function speed_callback(msg)

 commandedVelocity = msg.data

end

function sysCall_init()

 commandedVelocity = 0.0

 jointHandle = sim.getObject('.')

 publisher = simROS.advertise('left_motor/actual_speed',

'std_msgs/Float32')

 subscriber = simROS.subscribe('left_motor/setpoint_speed',

'std_msgs/Float32', 'speed_callback')

end

function sysCall_actuation()

 sim.setJointTargetVelocity(jointHandle, commandedVelocity)

end

function sysCall_sensing()

 actualJointSpeed = sim.getJointVelocity(jointHandle)

 simROS.publish(publisher, {data=actualJointSpeed})

end

function sysCall_cleanup()

 simROS.shutdownPublisher(publisher)

 simROS.shutdownSubscriber(subscriber)

end
```

7.2 Right Motor Code :

```
function speed_callback(msg)

 commandedVelocity = - msg.data

end

function sysCall_init()

 commandedVelocity = 0.0

 jointHandle = sim.getObject('.')

 publisher = simROS.advertise('right_motor/actual_speed',

'std_msgs/Float32')

 subscriber = simROS.subscribe('right_motor/setpoint_speed',

'std_msgs/Float32', 'speed_callback')

end

function sysCall_actuation()

 sim.setJointTargetVelocity(jointHandle, commandedVelocity)

end

function sysCall_sensing()

 actualJointSpeed = sim.getJointVelocity(jointHandle)

 simROS.publish(publisher, {data=actualJointSpeed})

end

function sysCall_cleanup()

 simROS.shutdownPublisher(publisher)

 simROS.shutdownSubscriber(subscriber)

end
```

In order to adjust the speed in direction of robot we ran 2 python code in order to navigate and control the robot and to enable us to stop the robot when the proximity sensor detects any thing

7.3 Wheel_odometry_localizer code :

```
import rospy

import math

from std_msgs.msg import Float32, Float32MultiArray


WHEEL_RADIUS = 0.0325

WHEEL_DISTANCE = 0.2


def pi_2_pi(angle):

    # a function to wrap the angle between -pi and pi (for numerical stability)

    return (angle + math.pi) % (2 * math.pi) - math.pi


def forward_model(wl, wr):

    # units are m/s and rad/s

    xdot = (wr * WHEEL_RADIUS + wl * WHEEL_RADIUS) / 2

    thetadot = (wr * WHEEL_RADIUS - wl * WHEEL_RADIUS) / WHEEL_DISTANCE

    return xdot, thetadot


def left_wheel_speed_callback(msg):
    global wl
    wl = msg.data

def right_wheel_speed_callback(msg):
    global wr
    wr = msg.data



if __name__ == "__main__":

    # Initialize the ROS node
    rospy.init_node("wheel_odometry_localizer_node")

    # Initialize wheel speed values
    wl, wr = 0, 0

    # Publisher for the pose topic
    pose_pub = rospy.Publisher("/wheel_odometry_localizer/pose", Float32MultiArray,
queue_size= 10)

    # Subscriber to the velocity commanding topic
    rospy.Subscriber("/sim_ros_interface/left_motor/actual_speed", Float32,
left_wheel_speed_callback)
    rospy.Subscriber("/sim_ros_interface/right_motor/actual_speed", Float32,
right_wheel_speed_callback)
```

```
# Initialize time for integration

t_start = rospy.get_time()

t_prev = rospy.get_time()


# Initialize variables for trapezoidal integration

xdot_prev, thetadot_prev = 0, 0


# Initialize integration values

x, y, theta = 0, 0, 0


rospy.loginfo("Connecting to CoppeliaSim topics")

rospy.loginfo("Accessing wheel speeds")


while not rospy.is_shutdown():

    xdot, thetadot = forward_model(wl, wr)


    # calculate dt

    t_start = rospy.get_time()

    dt = t_start - t_prev


    # do the integration (trapezoidal)

    x += (xdot + xdot_prev)/2 * math.cos(theta) * dt

    y += (xdot + xdot_prev)/2 * math.sin(theta) * dt
    theta += (thetadot + thetadot_prev)/2 * dt
    theta = pi_2_pi(theta) # wrap-to-pi
    t_prev = t_start

    # equate variables for next iteration
    xdot_prev = xdot
    thetadot_prev = thetadot

    # initialize ros message to be published
    msg = Float32MultiArray()
    msg.data.append(x)
    msg.data.append(y)
    msg.data.append(theta)
    pose_pub.publish(msg)
```

7.4 Navigator code :

```
import rospy
import math
from std_msgs.msg import Float32, Float32MultiArray, Int32

WHEEL_RADIUS = 0.0325
WHEEL_DISTANCE = 0.2
```

```
DIST_THRESHOLD = 0.1

YAW_THRESHOLD = 0.1

THETADOT_NAV = 0.85

XDOT_NAV = 0.2


TRAJECTORY=[

    [0, 0.5],

    [0,   1],

    [0, 1.5],

    [0,   2],

    [0.5, 2.5],

    [1,   2.5],

    [1.5, 2.5],

    [2,   2.5],

    [2.5, 2.5],

    [2.5, 2],

    [2.5, 1]

]


def inverse_model(xdot, thetadot):

    # units are m/s and rad/s

    wl = (xdot - thetadot * WHEEL_DISTANCE / 2) / WHEEL_RADIUS

    wr = (xdot + thetadot * WHEEL_DISTANCE / 2) / WHEEL_RADIUS

    return wl, wr


def pose_clbk(msg):
    global x, y, theta
    x = msg.data[0]
    y = msg.data[1]
    theta = msg.data[2]


def euclidean_distance(x_, y_, goal_point_):
    dist_x = goal_point_[0] - x_
    dist_y = goal_point_[1] - y_
    dist = math.sqrt(dist_x**2 + dist_y**2)
    return dist


def sign(x):
    if x >= 0:
        return 1
    else:
        return -1


def pi_2_pi(angle):
    # a function to wrap the angle between -pi and pi (for numerical stability)
    return (angle + math.pi) % (2 * math.pi) - math.pi


def steering_angle(x_, y_, theta_, goal_point_):
    return pi_2_pi(math.atan2(goal_point_[1] - y_, goal_point_[0] - x_) - theta_)


if __name__ == "__main__":
    rospy.init_node("navigator_node")


    # Initialize pose values
    x, y, theta = 0, 0, 0


    # Subscriber to the velocity commanding topic
    rospy.Subscriber("/wheel_odometry_localizer/pose", Float32MultiArray, pose_clbk)


    # Setup wheel speed publishers
    left_wheel_speed_pub = rospy.Publisher("/sim_ros_interface/left_motor/setpoint_speed", Float32,
queue_size=10)
    right_wheel_speed_pub = rospy.Publisher("/sim_ros_interface/right_motor/setpoint_speed", Float32,
queue_size=10)


    # Setup ROS rate
    rate = rospy.Rate(10) # 30 Hz


    rospy.loginfo("Navigation node successfully spawned")


    while not rospy.is_shutdown():


        rospy.loginfo("Starting to traverse the path")


        for goal_point in TRAJECTORY:
            while euclidean_distance(x, y, goal_point) > DIST_THRESHOLD and not rospy.is_shutdown():
```

```
        # Read proximity sensor
        proximity_msg = rospy.wait_for_message("/sim_ros_interface/proximity_sensor/state", Int32)
        if proximity_msg.data != 0:     # proximity sees something
            xdot = 0
            thetadot = 0
        # Safe to navigate
        elif abs(steering_angle(x, y, theta, goal_point)) > YAW_THRESHOLD:
            xdot = 0
            thetadot = THETADOT_NAV * sign(steering_angle(x, y, theta, goal_point))
        else:
            xdot = XDOT_NAV
            thetadot = 0
        lw_speed, rw_speed = inverse_model(xdot, thetadot)
        left_wheel_speed_pub.publish(Float32(lw_speed))
        right_wheel_speed_pub.publish(Float32(rw_speed))
        rate.sleep()
    rospy.loginfo("Robot navigation finished or terminated!")
    left_wheel_speed_pub.publish(Float32(0))    # publish zero speeds
    right_wheel_speed_pub.publish(Float32(0))
    break
```

8 PID :

8.1 Code :

```
#include "PID_v1.h"

#include "TimerOne.h"

#include "ros.h"

#include "std_msgs/Float32.h"

#include "std_msgs/Bool.h"

#include "NewPing.h"




#define KP1 0

#define KI1 10

#define KD1 0


#define KP2 0

#define KI2 10

#define KD2 0


//ENCODER

#define ENCODER1_PIN 2

#define ENCODER2_PIN 3

#define ENCODER_PULSE_PER_REV 10




//MOTOR 1

#define MOTOR_A1_PIN 7
```

```
#define MOTOR_B1_PIN 8


//MOTOR 2

#define MOTOR_A2_PIN 4

#define MOTOR_B2_PIN 9


#define PWM_MOTOR_1 5

#define PWM_MOTOR_2 6


#define EN_PIN_1 A0

#define EN_PIN_2 A1


#define trig 10

#define echo 11


 NewPing sonar(trig,echo,20);


// Timer specific macros

#define TIMER_PERIOD_MS 5


ros::NodeHandle nh;



bool face_det = false;
```

```
int counter1;
int counter2;

double speed_setpoint = 80;
double usSpeed1 = 0;
double usSpeed2 = 0;

double rpm1;
double rpm2;

long duration;
int distance;

void boolcb(const std_msgs::Bool&msg)
{
  face_det = msg.data;
}

ros::Subscriber<std_msgs::Bool> subscriber("/face_detector/status",&boolcb);

PID motor1_PID(&rpm1,&usSpeed1, &speed_setpoint, KP1, KI1, KD1, DIRECT);
PID motor2_PID(&rpm2,&usSpeed2, &speed_setpoint, KP2, KI2, KD2, DIRECT);

void encoder1_ISR(){
  counter1++;
  //Serial.println("Encoder1");
```

```
}


void encoder2_ISR(){
  counter2++;
  //Serial.println("Encoder2");
}


void timerISR(){


  detachInterrupt(digitalPinToInterrupt(ENCODER1_PIN));
  detachInterrupt(digitalPinToInterrupt(ENCODER2_PIN));


  //Serial.print("Counter: ");
  //Serial.println(counter2);


  rpm1 = ((double(counter1) / ENCODER_PULSE_PER_REV)*200)*60;
  rpm2 = ((double(counter2) / ENCODER_PULSE_PER_REV)*200)*60;


  //Serial.print("Motor two RPM:");
  //Serial.println(rpm1);


  counter1 = 0;     // Reset the counter
  counter2 = 0;     // Reset the counter


  attachInterrupt(digitalPinToInterrupt(ENCODER1_PIN), encoder1_ISR, RISING);
  attachInterrupt(digitalPinToInterrupt(ENCODER2_PIN), encoder2_ISR, RISING);
```

```
}


void sensor(){


Serial.println(sonar.ping_cm());

delay(100);

if((sonar.ping_cm()<9)&&(sonar.ping_cm()>0))

{

  usSpeed1=0;

  usSpeed2=0;

  }


}


void setup() {


  nh.initNode();

  nh.subscribe(subscriber);




  pinMode(MOTOR_A1_PIN, OUTPUT);

  pinMode(MOTOR_B1_PIN, OUTPUT);


  pinMode(MOTOR_A2_PIN, OUTPUT);

  pinMode(MOTOR_B2_PIN, OUTPUT);
  pinMode(PWM_MOTOR_1, OUTPUT);
  pinMode(PWM_MOTOR_2, OUTPUT);

  pinMode(EN_PIN_1, OUTPUT);
  pinMode(EN_PIN_2, OUTPUT);

  pinMode(ENCODER1_PIN, INPUT);
  pinMode(ENCODER2_PIN, INPUT);



  attachInterrupt(digitalPinToInterrupt(ENCODER1_PIN), encoder1_ISR, RISING);
  attachInterrupt(digitalPinToInterrupt(ENCODER2_PIN), encoder2_ISR, RISING);

  Timer1.initialize(5000);        // initialize timer1, and set a 1/2 second period
  Timer1.attachInterrupt(timerISR);

  motor1_PID.SetMode(AUTOMATIC);
  motor2_PID.SetMode(AUTOMATIC);
  digitalWrite(MOTOR_A1_PIN, 0);
  digitalWrite(MOTOR_B1_PIN, 1);
  digitalWrite(MOTOR_A2_PIN, 1);
  digitalWrite(MOTOR_B2_PIN, 0);

  digitalWrite(EN_PIN_1, HIGH);
  digitalWrite(EN_PIN_2, HIGH);
```

```
//  analogWrite(PWM_MOTOR_1, 80);
//  analogWrite(PWM_MOTOR_2, 80);

//  analogWrite(PWM_MOTOR_1, usSpeed1);
  //analogWrite(PWM_MOTOR_2, usSpeed2);

  Serial.begin(9600);
  delay(100);
}

void loop() {

 Serial.print("MOTOR TWO: ");
 Serial.println(rpm2);

 Serial.print(" MOTOR ONE: ");
 Serial.println(rpm1);

 motor1_PID.Compute();
 motor2_PID.Compute();

 sensor();

if(face_det==1)
{
  Serial.println(face_det);
  digitalWrite(12, LOW);
  digitalWrite(13, HIGH);
  }

 analogWrite(PWM_MOTOR_1, usSpeed1);
 analogWrite(PWM_MOTOR_2, usSpeed2);


 nh.spinOnce();
 delay(100);

//Serial.print("GIVEN SPEED: ");
//Serial.println(usSpeed1);
//Serial.println(usSpeed2);

}
```

9 Raspberry pi :

9.1 Face detection Code :

```
#!/usr/bin/env python

import cv2
import rospy
import numpy
from std_msgs.msg import Int32
from sensor_msgs.msg import Image

def imageMessage_callback (msg):
    global vid_path
    vid_path = msg.data

if _name_ == "_main_":

    # Load the face detector model
    face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

    # Load the demo video
    rospy.Subscriber("/sim_ros_interface/image", Image, imageMessage_callback)
    cap = cv2.VideoCapture(0)

    # Initialize ROS node
    rospy.init_node("face_detector_node")
    pub = rospy.Publisher("face_detector/status", Int32, queue_size=10)

    while cap.isOpened():
        ret, frame = cap.read()

        # Reset each frame
        face_detected_in_this_frame = False

        if ret:

            image_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

            # scale_factor: Parameter specifying how much the image size is reduced at each image scale.
            scale_factor = 1.3
            # min_neighbours: Parameter specifying how many neighbors each candidate rectangle should
have to retain it.
            min_neighbours = 5

            # Perform the faces detection
            faces = face_cascade.detectMultiScale(image_gray, scale_factor, min_neighbours)

            for (x,y,w,h) in faces:

                # Draw a rectangle at the detected location of the face
                cv2.rectangle(image_gray, (x,y), (x+w,y+h), (255, 255, 255) ,2)

                # Set the bool face_detected_in_this_frame to True
                face_detected_in_this_frame = True

            # Publish the status of the detector
            if face_detected_in_this_frame:
                pub.publish(Int32(1))
            else:
                pub.publish(Int32(0))

            # Show the frame
            cv2.imshow('frame', image_gray)
```

```
# Check if the user has pressed ESC key
c = cv2.waitKey(1)

if c == 27:
    cv2.destroyAllWindows()
    break   # exit if ESC is pressed
```

10 Manufacturing :

We fixed the h-bridge with screws onto the base

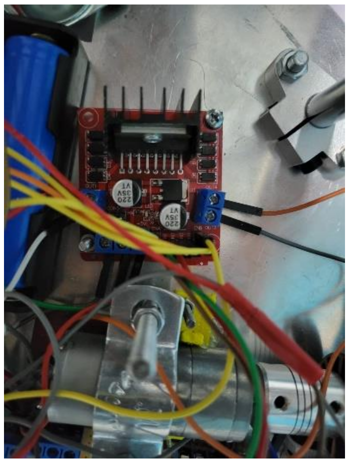


Figure 29: Real H-Bridge Fixation

The fixation of the Arduino is also with screws

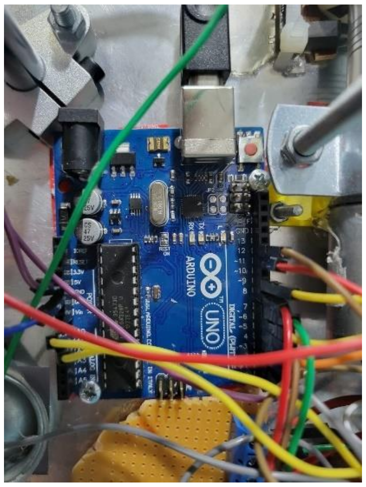


Figure 30: Arduino Fixation

We fixated the Motor with a 3d printed part and a metal part on top to keep it from moving

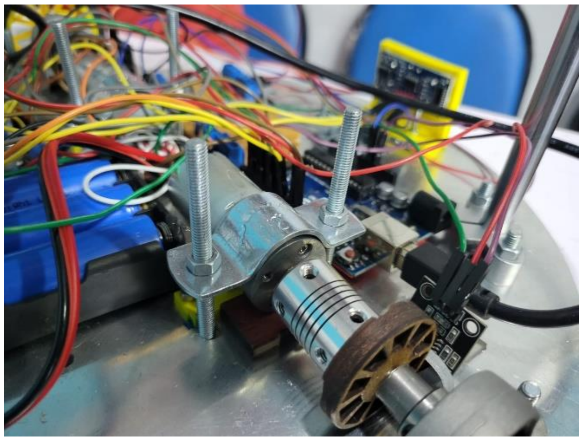


Figure 31: Motor Fixation

We Fixated the wheels with a coupler to the shaft and fixated the motor to the shaft with a coupler

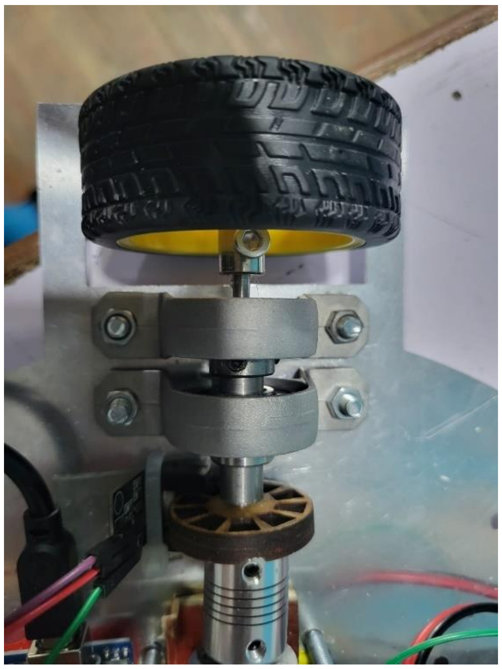


Figure 32: Wheel Fixation

11 Circuit :

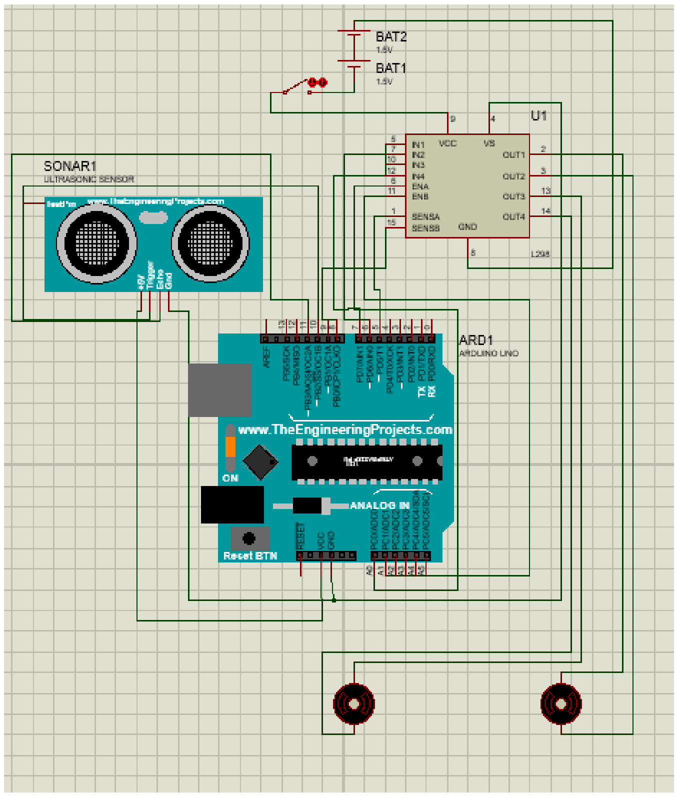


Figure 33: Wiring Of Robot

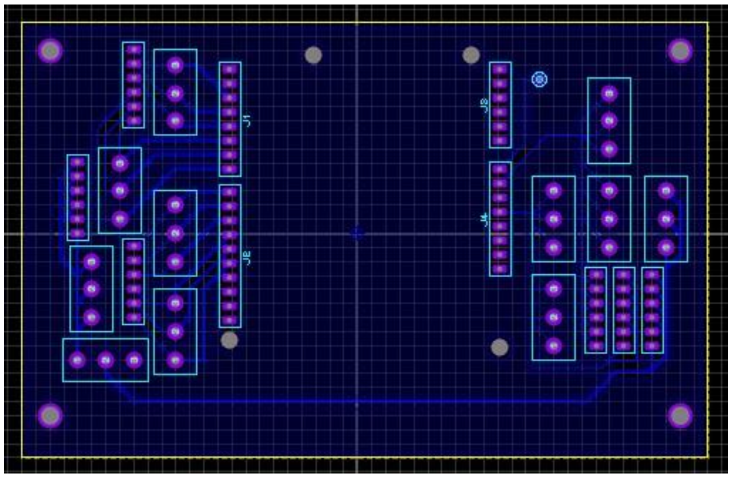
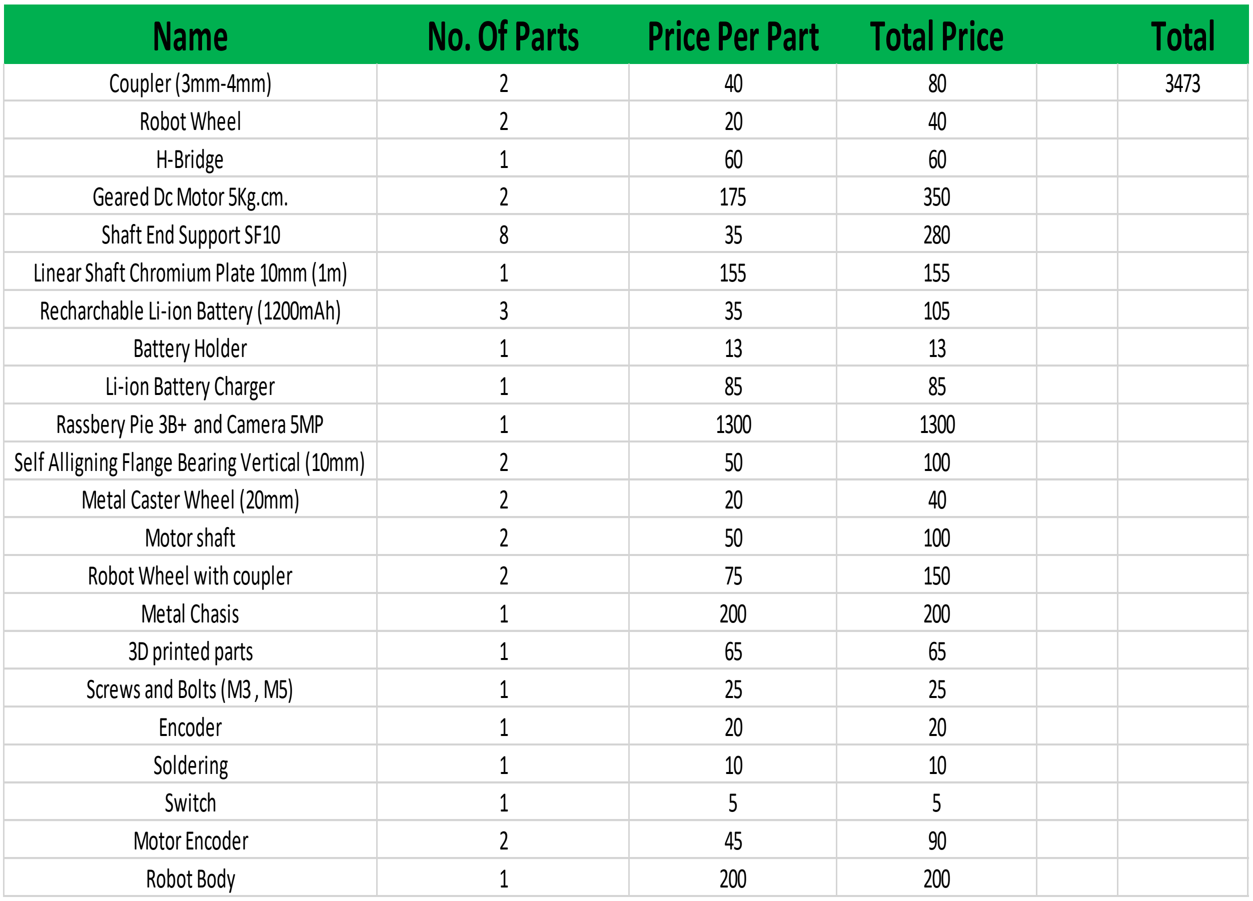


Figure 34: PCB

12 B.O.M :



Name	No. Of Parts	Price Per Part	Total Price		Total
Coupler (3mm-4mm)	2	40	80		3473
Robot Wheel	2	20	40		
H-Bridge	1	60	60		
Geared Dc Motor 5Kg.cm.	2	175	350		
Shaft End Support SF10	8	35	280		
Linear Shaft Chromium Plate 10mm (1m)	1	155	155		
Recharchable Li-ion Battery (1200mAh)	3	35	105		
Battery Holder	1	13	13		
Li-ion Battery Charger	1	85	85		
Rassbery Pie 3B+ and Camera 5MP	1	1300	1300		
Self Alligning Flange Bearing Vertical (10mm)	2	50	100		
Metal Caster Wheel (20mm)	2	20	40		
Motor shaft	2	50	100		
Robot Wheel with coupler	2	75	150		
Metal Chasis	1	200	200		
3D printed parts	1	65	65		
Screws and Bolts (M3 , M5)	1	25	25		
Encoder	1	20	20		
Soldering	1	10	10		
Switch	1	5	5		
Motor Encoder	2	45	90		
Robot Body	1	200	200		

Figure 35: Bill of Materials