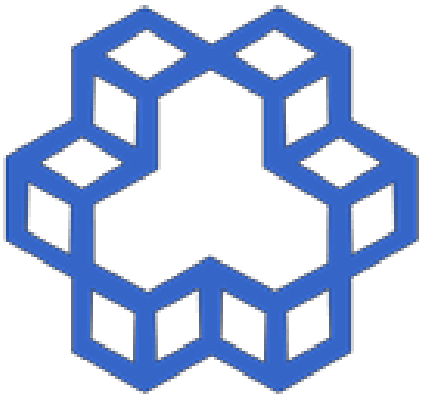


K. N. Toosi
University of Technology

Introductory Robotics Course
Project and Lab

Smart Hospital Group

Professor: Dr. Ali Najafi Ardakani

Term 4021

In the name of God, soul and wisdom

Review

Smart hospitals are an important development in the field of healthcare. These hospitals improve clinical and management processes by using information technologies and digital networks. These optimizations are made in order to improve the quality of medical services and the comfort of patients and hospital staff.

Smart hospitals have features that contribute to the development of optimized systems:

1. Data Collection and Insights: These hospitals are capable of collecting data from the hospital's internal systems. This data is then converted into insights by machine learning and data analysis software. Doctors, nurses, and hospital staff also have access to this insight.
2. Patient Admission Kiosk: These kiosks make the receptionist's task easier and reduce the waiting time of patients by automating repetitive parts of the registration process.
3. Digital scheduling system: Implementing a digital scheduling system can drastically reduce hospital costs and help staff manage time.
4. Educational games: Using monitors for educational games reduces children's anxiety and makes the hospital atmosphere more relaxed.
5. Electronic medical records: Electronic records instead of papers, make it easier to work and reduce the transmission of disease between clients and the doctor.

Overall, smart hospitals facilitate the improvement of the quality of healthcare services and improve the patient experience by integrating modern technologies.

Table of contents

Division of duties

At first, according to their expertise and ability, the members were categorized into different groups, the division is as follows:

Image Processing Group: Mr. Aghazadeh, Mr. Elmi, Mr. Ghaffarzadeh, Mr. Sharifi and Mr. Maleki

Curtain and Smart Light Group in the Patient's Room: Ms. Faraji and Ms. Gol Nabi

Chassis Design Group: Messrs. Rahi, Sharafi and Nalbandian

Voice Assistant and Easy-to-Use Robot Group: Padash and Ershad

Engine Group: Mr. Padash and Mr. Samieinia

Procurement and Procurement Officer: Ramtin Yavariri

Responsible for writing the report and making the PowerPoint: Mr. Sharifi and Mr. Samieinia

Image Processing

In performing image processing, we performed the following steps:

1. Preparation of Images
2. Editing images
3. Image Repair
4. Color Image Processing
5. Multiple Resolution Processing
6. Compression of Images
7. Image Structure Processing
8. Image segmentation
9. Representation and Extraction of Images Properties
10. Object Detection in Images

In the following, we will explain all these steps:

A. Preparation of Images

To process images, we need digital images, which can be done using cameras.

B. Editing images

After preparing the digital images, we apply modifications to it to prepare it for image processing.

C. Image Repair

At this stage, we took steps to eliminate the noise.

D. Color Image Processing

In this stage of color image processing, various processes are performed on RGB and RGB color images.

E. Multiple Resolution Processing

Multiple resolution is a type of representation of images at multiple scales. Using this method, images with very large dimensions such as aerial and satellite images

can be compressed in such a way that they can be easily displayed in a short period of time.

F. Image Compression

In order to transfer images to other devices or to store them in existing memory, we need to compress the images because the original images are large in size.

G. Image Structure Processing

In order to represent and describe the shapes in the images, it is necessary to identify the components of the images. Image structure processing, which involves a set of mathematical processes, makes it possible to identify the components that make up the images

For example, by processing the structure of the image, the edges and lines in the images can be blurred or their image can be intensified.

H. Crop Images

By cutting the images into smaller parts, the analysis of the images can be simplified. In other words, the act of cutting the image causes the computer to pay more attention to the more important parts and ignore the less important parts of the image when analyzing the image.

I. Representation and Extraction of Images Properties

After cutting the images into smaller parts, the next step is to represent the images and describe them by extracting information from them. Using this step, the characteristics and characteristics of the images and their quantitative information are determined for further processing by the computer.

J. Object Detection

After the visualization and extraction of the images properties, this information is given to an automated system to assign a label to the object in the image.

For example, the system can identify objects in images and divide them into different categories such as "machine," "human," "animal," and "plant" using predefined labels.

What are the methods of image processing?

In response to the question of what image processing is, we explained that this field is used to improve image quality, remove certain parts of images, identify objects in images, and even create new ones.

Image processing is a complex field, and there are different algorithms and methods in this field. In this section, we will discuss some of the common image processing issues.

Robot detection with the help of Aruco:

One of the most famous and effective methods for detecting and determining the position of objects is in computer vision. This method uses markers with special patterns called "Aruco markers" that can be used to detect and calculate the position and direction of objects in an image or viewpoint camera. (Figure 3)

The main purpose of using Aruco markers is to determine the position and angle relative to the camera. These markers include simple patterns such as frames with black and white pixels that create specific patterns. These patterns are designed in a very special way to be easily recognized and identified by computer vision systems, such as cameras.

Some of the benefits of Aruco include:

- ✓ Accurate and reliable: Aruco can detect markers with high accuracy in various lighting conditions.
- ✓ Fast and efficient: Aruco can track indicators in real-time, making it suitable for high-speed applications.
- ✓ Flexible: Aruco can be used with different types of markers and cameras.

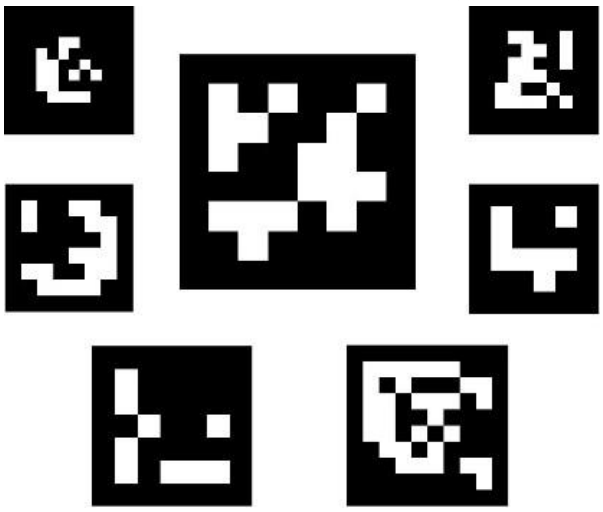


Figure 3- Examples of Aruco

In this method, Aruco is attached to the bot and then detected Aruco with the help of the functions in the opencv library. This method has a much better performance than the previous two methods in terms of reliability and speed, and this method is chosen to continue the project.

In-Room Image Processing

After the robot enters the room, we guide the robot with the help of a guide line drawn to the patient's bed.

Parts Used:

- Physical Components: One ESP32 Board, L298N Motor Driver, Mobile, IP Webcam Software , Jumper Cable, 12V DC Motor (2 Pcs), Metal Chassis, Wheels, Power Supply
- Knowledge of Image Processing via OpenCV in Python
- Arduino IDE Knowledge
- Knowledge of the different aspects of the PID control system to adjust the PID system
- L298N Engine Driver Knowledge

Introduction:

A line follower robot that independently follows a black line/path on a white background with the help of image processing using OpenCV.

Activity Performed:

The phone's camera records the current frame and sends it to the laptop for image processing, OpenCV determines whether the black line is in the center or left or right and sends this information to the Arduino IDE. The left should move and vice versa.

- To control the direction of rotation of the robot, the change is introduced to the speed of the motors.
- The robot will rotate to the right if the speed of the right wheel is slower than the left wheel, and vice versa.
- The robot will move forward if both motors have the same speed.
- L298N is a motor driver that is used to drive motors.

Image Processing Using OpenCV

Overview

(The direction of the robot is from the right side of the screen to the left side of the screen. The top direction in the image corresponds to the right side of the robot and vice versa.)

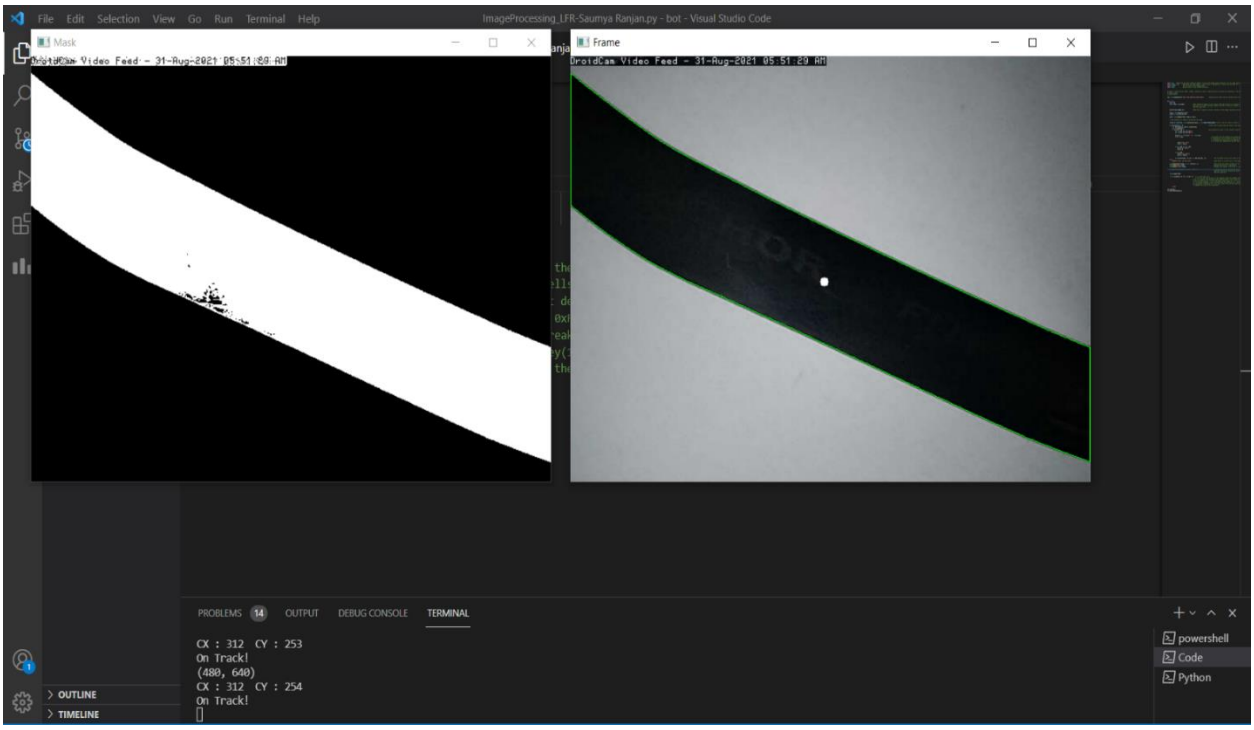
A. Direct Movement

1. The main frame taken by the camera. The white dot is used to determine the approximate center of the contour of the black line in the frame.

2. The same frame with the image overlay applied to it, shows what the robot sees when looking at the frame.

Because the black line is roughly in the middle of the screen, the robot has instructions to go straight.

```
CX : 312  CY : 253
On Track!
(480, 640)
CX : 312  CY : 254
On Track!
```



B) Moving Right

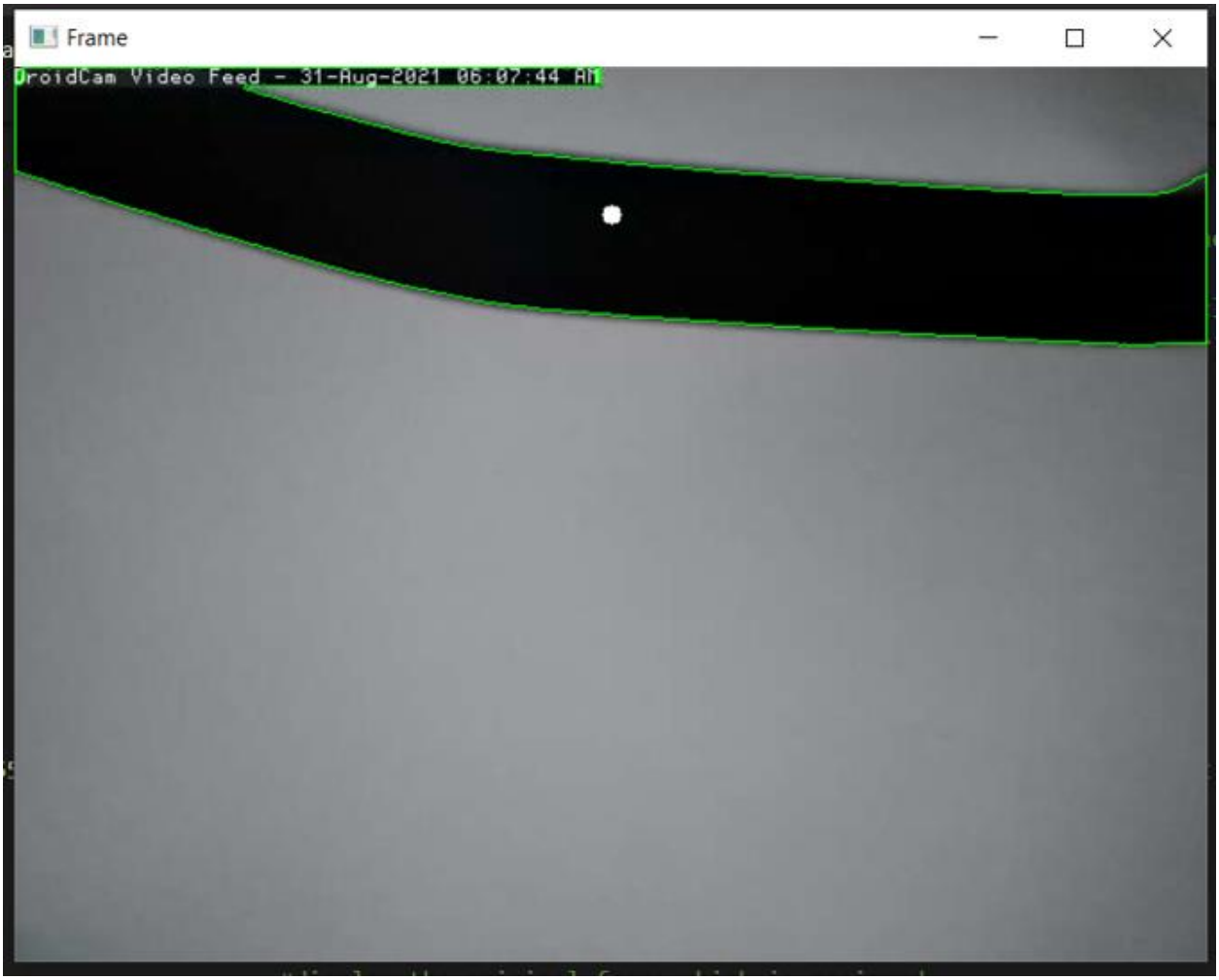
```
if cy <=190 :
    print("Turn Right")
    error = 190-cy
```

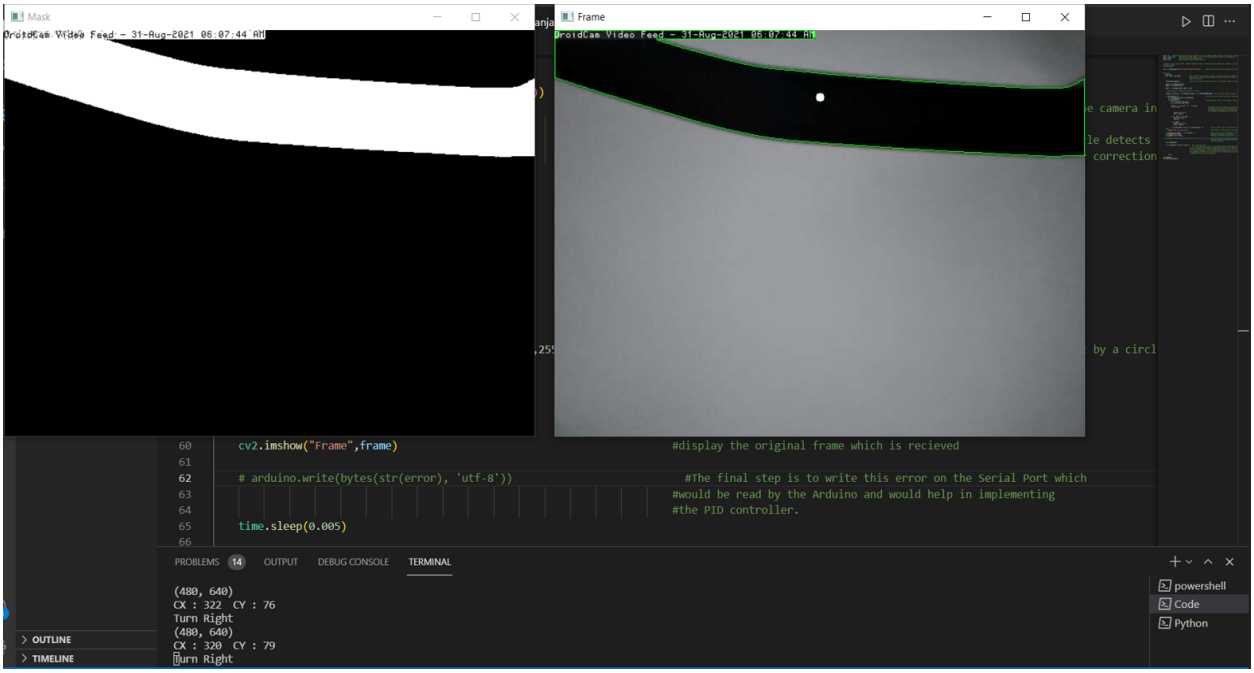
- Cy is a variable that is the height of the central point from the top of the screen.
- Since the camera is recorded in portrait mode and image processing is done in landscape mode.

The orientation (top) in the attached images corresponds to the (right) side of the screen.

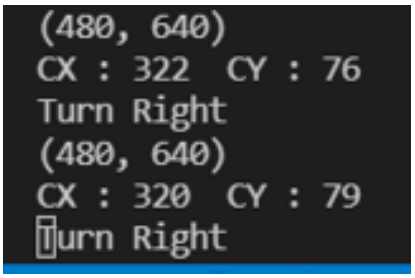
The direction (top) in the attached images corresponds to the (right) page.

- So, if the black line is at the top (right) of the frame, it should go to where the black line is, i.e., to the right, the robot will receive instructions to turn to the right.

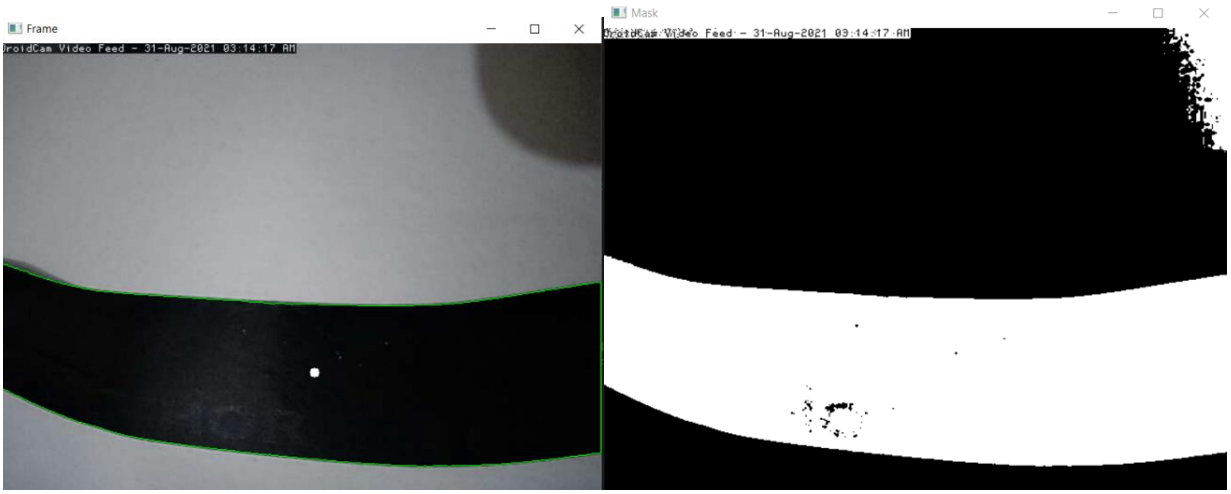

Frame
DroidCam Video Feed - 31-Aug-2021 06:07:44 AM



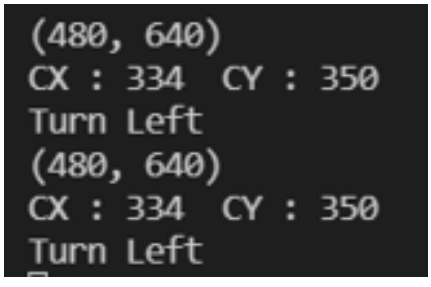
Since the center location of the dot is 76 pixels from the top of the screen (which corresponds to 76 pixels from the right side of the camera frame), the instruction should be to the right to maintain the black line in the middle of the camera frame.



The same processing is done to move to the left.


Frame
DroidCam Video Feed - 31-Aug-2021 03:14:17 AM
Mask
DroidCam Video Feed - 31-Aug-2021 03:14:17 AM


Mask
DroidCam Video Feed - 31-Aug-2021 03:14:17 AM
Frame
DroidCam Video Feed - 31-Aug-2021 03:14:17 AM
mask = cv2.inRange(frame, high_b, low_b)
contours, hierarchy = cv2.findContours(mask, 1, cv2.CHAIN_APPROX_NONE)
if len(contours) > 0 :
c = max(contours, key=cv2.contourArea)
M = cv2.moments(c)
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
(480, 640)
CX : 334 CY : 350
Turn Left
(480, 640)
CX : 334 CY : 350
Turn Left
OUTLINE
TIMELINE


(480, 640)
CX : 334 CY : 350
Turn Left
(480, 640)
CX : 334 CY : 350
Turn Left

Since the dot is at the bottom of the screen, the instructions are to move to the left.

- Black Line Detection Using Masks and Lines

```
low_b = np.uint8([51,51,51])
high_b = np.uint8([0,0,0])

mask = cv2.inRange(frame, high_b, low_b)

# used masking as a method to preprocess the image

contours, hierarchy = cv2.findContours(mask, 1, cv2.CHAIN_APPROX_NONE)
```

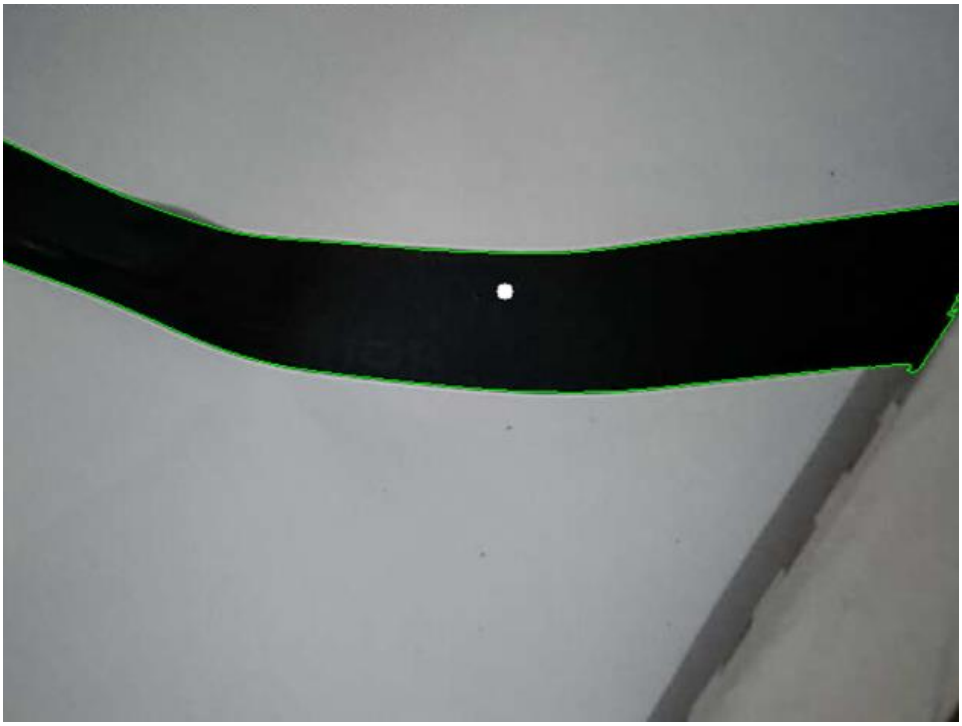
Use Contour to find the outline of the black areas in the mask frame created from the original frame.

```
if len(contours) > 0 :
    c = max(contours, key=cv2.contourArea)
```

If black areas are found, find the contour that has the most area. (This is done to avoid focusing on other dark objects in the vicinity of the path)

```
cx = int(M['m10']/M['m00'])
cy = int(M['m01']/M['m00'])
```

Calculating the center of the largest black area detected using the .moments white circle method



- Rotation conditions

```
if cy >= 290 :


    print("Turn Left")
    error = 290-cy

if cy < 290 and cy > 190 :
    print("On Track!")
    error =0

if cy <=190 :
    print("Turn Right")
    error = 190-cy
```

"Center" is defined in the range of 190 to 290 pixels.

If the y-point coordinates are more than 290, i.e., 290 pixels from the top of the screen, the error variable will include the correction necessary to rotate the bot to the left.

If the y-point coordinates from the top are less than 190 pixels, and the robot rotates to the right, the error variable will include the correction necessary to rotate the robot to the right.

If the robot's y-coordinates are between 190 and 290 pixels, the robot will continue to move straight and the error variable contains a value of 0, that is, no correction is required.

- Confirm

```
cv2.drawContours(frame, c, -1, (0,255,0), 1)
cv2.imshow("Mask",mask)
cv2.imshow("Frame",frame)
```

Finally, to see if the camera correctly detects, we print the contour lines along with the frame.

We also print the mask and the main frame along with an identification circle to trace the center of the detected contour and provide us with the required adjustment rate.

```
cv2.circle(frame, (cx,cy), 5, (255,255,255), -1)
```

It is used to draw a white feathered circle with a radius of 5 pixels in the center of the contour for better detection vision.

```
arduino.write(bytes(str(error), 'utf-8'))
```

The final step is to send this error data to the Arduino IDE to control the PID.

Arduino Code and PID Setup

```
// motor one
int enA = 5;        //l298n's en_A port connected to the pwm pin (5) of the arduino uno
int in1 = 6;        //direction control pin in1 connected to digital pin (6) of the arduino uno
int in2 = 7;        //direction control pin in2 connected to digital pin (7) of the arduino uno

// motor two
int in3 = 9;        //similarly for motor two
int in4 = 8;
int enB = 10;       //port 10 is also a pwm port so that we can control the speed of the motors via enA and enB pins
```

These are the connections that must communicate with the Arduino Uno board .

Speed control pins mean. ENA and ENB are used to turn the engines on and off and control its speed.

Pulling these pins up causes the motors to rotate and stop them by pulling it at LOW . But, with pulse width modulation (PWM), we can actually control the speed of the motors.

This is why enA and enB are connected to PWM pins to regulate the rotation speed of the motors.

```
const uint8_t max_a = 100 ;
const uint8_t max_b = 100 ;
const uint8_t base_a = 60 ;
const uint8_t base_b = 60 ;
```

The maximum engine speed is A and (0-255)100 B.

The base speed of motors is A and 60 B A (0-255).

```
float pid, err, previous_error = 0;
float pid_p = 0;            //proportional part of the pid
float pid_i = 0;            //integral part of the pid
float pid_d = 0;            //differential part of the pid

double kp = 1;              //proportional factor
double ki = 0;              //integrating factor
double kd = 0.2;            //differentiating factor
```

Here we set up the initial errors.

The kp, ki, and kd values that I got from trial and error can vary for different people.

These were the values that my model performed best at base speed of 60.

The values of kp, ki, and kd are the factors by which proportional, integral, and differential errors affect the final error.

```
void PID(int error)
{
  Prev_time = curr_time;
  curr_time = millis();
  elapsedTime = (curr_time - Prev_time) / 1000;
```

millis ( ) returns the number of milliseconds that have been spent since the Arduino board started running the current program.

elapsedTime is calculated by sublayering curr_time and Prev_time.

```
pid_p = kp * error;
```

Proportional Error

This factor requires an error to make a correction, if there are 0 errors, no corrections will be made.

This creates a steady state error and cannot guarantee that the bot is following the correct path, so we need other PID control parts as well.

```
pid_i = pid_i + (error * elapsedTime);
```

Merge error

This small area below adds the error graph in a very short amount of time to the previous total error.

The integrated section is used to reduce the steady mode error by adding up all the previous errors.

```
pid_d = kd * ((error - previous_error) / elapsedTime);
```

Derivative error

If the error is decreasing, its graph will have a negative slope and hence the derivative will be negative which when added to the total correction means that the corrections will also decrease.

In case the robot gets too close to the center point, this will help slow down the motors.

```
pid = pid_p + ki * pid_i + pid_d;
```

Then I calculate the final error by summing up all the errors.

Applying final modifications to the speed of the robot motor.

```
speed_A = base_a + pid;
//adjusting the speed of motor A with the help of PID corrections
speed_B = base_b - pid;
//adjusting the speed of motor B with the help of PID corrections
```

```
//setting a limit for the maximum speed a motor can turn on
  if (speed_A > max_a) {
    speed_A = max_a;
  }
  if (speed_B > max_b) {
    speed_B = max_b;
  }

  if (speed_A < 0) {
    speed_A = 0;
  }
  if (speed_B < 0) {
    speed_B = 0;
  }
```

This ensures that the speed of the motors never exceeds the value of 100 that I had set before, and I limit them to the maximum if I do.

```
void setup() {
  // All the arduino pins used are outputs
  pinMode(enA, OUTPUT);
  pinMode(enB, OUTPUT);
  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);
  pinMode(in3, OUTPUT);
  pinMode(in4, OUTPUT);
  pinMode(13, OUTPUT);
```

All Arduino pins are used as outputs where the Arduino (0V or 5V) (off or on) on the in1, in2, in3 and in4 ports and for the enA and enB pins provide the PWM output . It can range from 0 to 255 as these pins control the speed of the robot.

```
curr_time = millis();
Serial.begin(115200);
```

Set the current time and start communicating with the Arduino UNO board with a wind rate of 115200 bits per second.

```
void moveForward(int speed1, int speed2) {

  digitalWrite(in1, HIGH);
  digitalWrite(in2, LOW);

  analogWrite(enA, speed1); // Speed Range 0-255

  digitalWrite(in3, LOW);
  digitalWrite(in4, HIGH);

  analogWrite(enB, speed2);
}
```

It takes speed 1 and speed 2 as inputs and puts enA and enB equal to them (which are the speed of motor 1 and motor 2)

For Engine 1,

If in1 is low and in2 is low, motor 1 will stop.

Since in1 is high and in2 is low, motor 1 rotates forward at speed 1

For motor 2,

Since the in3 is low and the in4 is high, the motor 2 rotates forward at a speed of 2

Loop ()

```
int errors = Serial.readString().toInt();
```

Variable errors error sent via serial port by<= OpenCV stores this field

```
arduino.write(bytes(str(error), 'utf-8'))
```

```
PID(errors);
```

Now this error is applied by the PID control function described earlier.

```
  speed_A = base_a + pid;
  //adjusting the speed of motor A with the help of PID corrections
  speed_B = base_b - pid;
  //adjusting the speed of motor B with the help of PID corrections
```

which determines the speed of the A and B motors.

```
moveForward(speed_A, speed_B);
```

And finally, the L298N receives these values via the Arduino range to turn the motors.

This process is repeated over and over again with feedback in the loop until the Arduino is discontinued.

Connections

Arduino Uno L298N

pin 5 - enA

pin 6 - in1

pin 7 - in2

pin 8 - in3

pin 9 - in4

pin 10 – enB

Arduino GND Pin to GND L298N Pin (for Common Ground)

L298N to Li-Po Battery Pack

Positive terminal - Vcc (+12 V pin)

Negative terminal - GND (+5 V pin)

USB port of laptop to Arduino (for Arduino power)

Port Used: COM4 Port

Tasks that still need to be done

- Setting up more PID to make the robot smoother
- Using the ESP32 camera module instead of the phone camera to make the robot lighter

Technical problems ahead

- During image processing, at high speeds, sometimes the robot will swerve to one side even after detecting the black line, due to the large difference in the maximum and speed of the base motor.
- PID Setup

Smart curtains and lighting in the patient's room

The goal of this part of the project was for the LDR sensor to move the curtain to the right or left according to the light it receives. If I want to give a general explanation of how it works, it is that a motor is connected to the shaft of the curtain, and according to the clockwise or anticlockwise rotation of the motor, the curtain should be moved to the right or left, which is a part of this project It was related to the Arduino code that we defined a variable called the LDR value , which shows the amount of light received by the LDR.

Next, we selected different intervals for the LDR value and finally defined the rotation of the engine with digital write (i.e., its high, low) and counterclockwise (i.e., changing its arguments (LOW, HIGH).

The equipment used in this section is:

ESP32 board, L298N driver, some jumper wires, 100 kOhm resistor, board, LM35 sensor, 12V gearbox motor, pole, gear, camshaft and cam

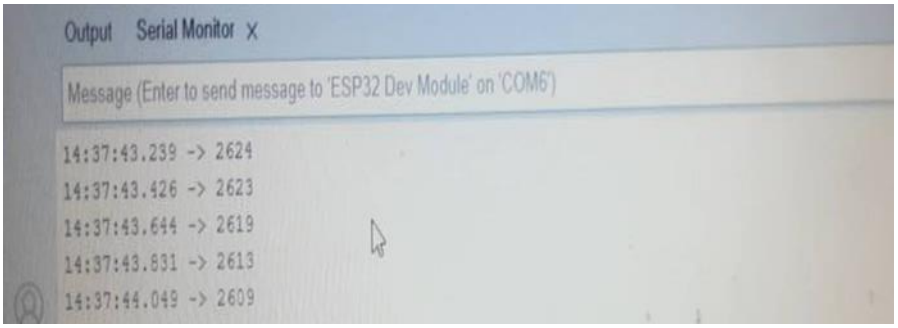
Steps :

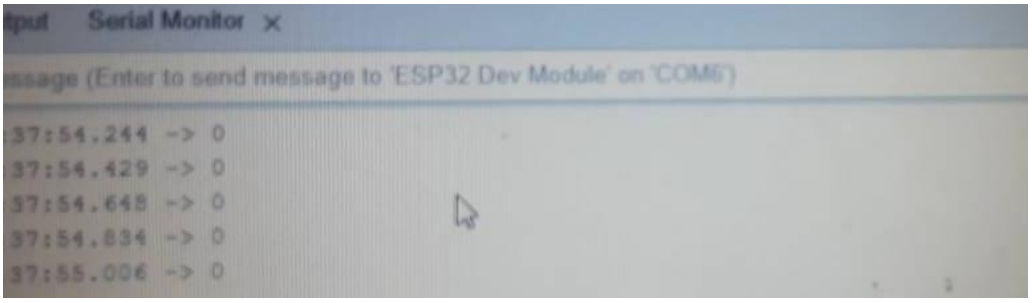
First, we put the ESP32 board on the board. Then we close the LDR circuit.

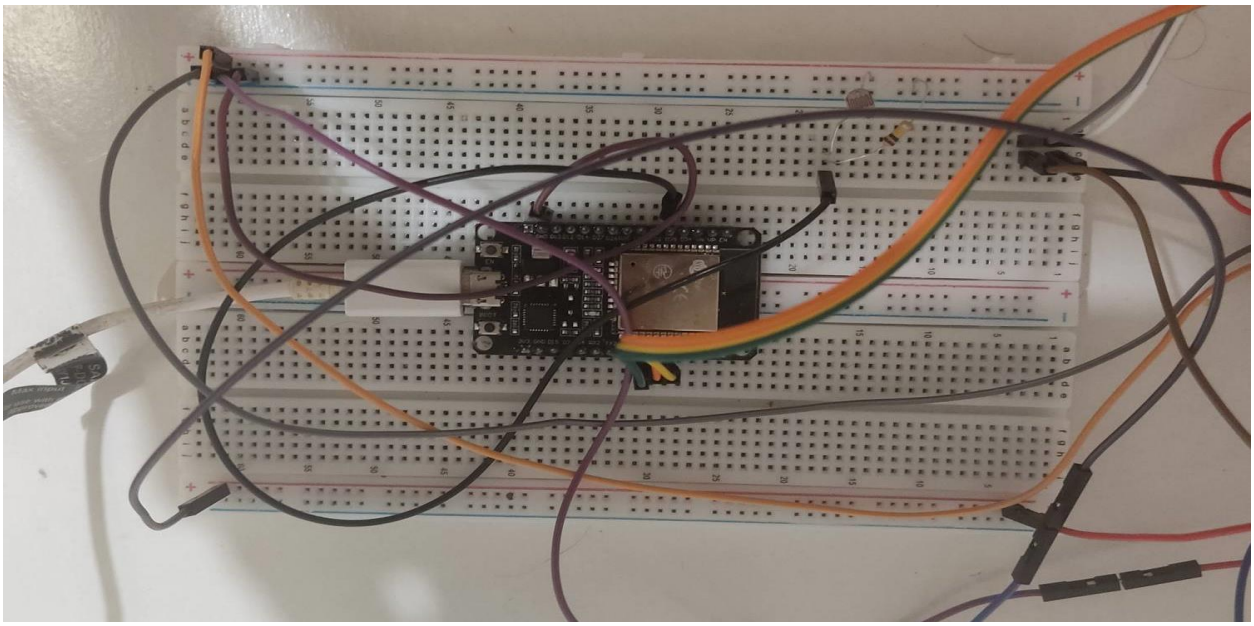
We put the LDR on the board, one end of which is connected to the positive end of the board (which I consider the positive side of the right side of the board to be +5 volts) and the negative part is ground and the positive left side is for the lithium battery or power supply). I connect a jumper wire from the LDR connection and the resistor (according to the code that I defined digital pin 35) to the digital pin 35, ESP32.The driver L298N which has a 12-volt pin is the same as the positive input of the battery.The driver with the ground pin that connects to the negative end of the board (the negative part on the right side of the board).The driver that has a 5-volt output is connected to the positive part of the right side of the board.And finally the positive head of the source We connect the power to the positive side of the left side, and the negative end is connected to the negative end of the common board board.We connect the input of the L298N driver to the positive side of the left side of the board.

Finally, we connect the ground- ESP32 to the common negative head of the board.

When we run the code, the following results are obtained, which in high light represents approximately the number 2600 and if it goes dark, it displays the number 0.







In the next part, we are going to close the engine circuit, the output L298N is connected to the two ends of the motor, according to the code, enable L298N is

connected to digital pin 18 ESP32, and pin 1,2 L298N is connected to digital pin 19,21 ESP32.

Finally, according to the following code

```
int LIGHT_SENSOR_PIN = 35; // ESP32 pin GIOP35 (ADC0)

int motor1Pin1 = 21;
int motor1Pin2 = 19;
int enable1Pin = 18;
// Setting PWM properties
const int m_freq = 10000;
const int m_pwmChannel = 0;
const int m_resolution = 8;
int LDRValue;

void setup() {
  Serial.begin(9600);
  while (!Serial) {x
    delay(10);
  }
  pinMode(LIGHT_SENSOR_PIN, INPUT);

  // sets the pins as outputs:
  pinMode(motor1Pin1, OUTPUT);
  pinMode(motor1Pin2, OUTPUT);

  // configure LED PWM functionalitites
  ledcSetup(m_pwmChannel, m_freq, m_resolution);

  // attach the channel to the GPIO to be controlled
  ledcAttachPin(enable1Pin, m_pwmChannel);

}

void loop() {
  LDRValue = analogRead(LIGHT_SENSOR_PIN); // read the value on analog pin
    Serial.println(LDRValue);
   // We'll have a few threshholds, qualitatively determined
  if (LDRValue < 40) {
    // Stop the DC motor
    digitalWrite(motor1Pin1, LOW);
    digitalWrite(motor1Pin2, LOW);
```

```
    digitalWrite(motor1Pin1, LOW);
    digitalWrite(motor1Pin2, HIGH);
    ledcWrite(m_pwmChannel, 200);


  }
  else if (LDRValue > 40 && LDRValue < 2000) {
    digitalWrite(motor1Pin1, HIGH);
    digitalWrite(motor1Pin2, LOW);
    ledcWrite(m_pwmChannel, 200);
  }
  else if (LDRValue > 2000 && LDRValue < 2800) {
    digitalWrite(motor1Pin1, LOW);
    digitalWrite(motor1Pin2, HIGH);
    ledcWrite(m_pwmChannel, 200);
  }
  else if (LDRValue > 2800 && LDRValue < 3600) {
    digitalWrite(motor1Pin1, LOW);
    digitalWrite(motor1Pin2, HIGH);
    ledcWrite(m_pwmChannel, 200);
  }
  else {
    digitalWrite(motor1Pin1, LOW);
    digitalWrite(motor1Pin2, HIGH);
    ledcWrite(m_pwmChannel, 200);
  }
  delay(100);
}
```

The following results are obtained that if the amount of light received by the LDR is less than 2000, the clockwise motor and the curtain move to the right, and if it is more than 2000, the anticlockwise motor and the curtain move to the left.

How to fasten the motor to the curtain shaft

Chassis Design

At first, the chassis design team presented several designs, but due to the limitation of laser cutting, all the designs were rejected and not implemented, so it was necessary to present an alternative design, which was sent to the workshop after three days of design without the implementation of filth at the depth of Z, and the chassis was delivered.

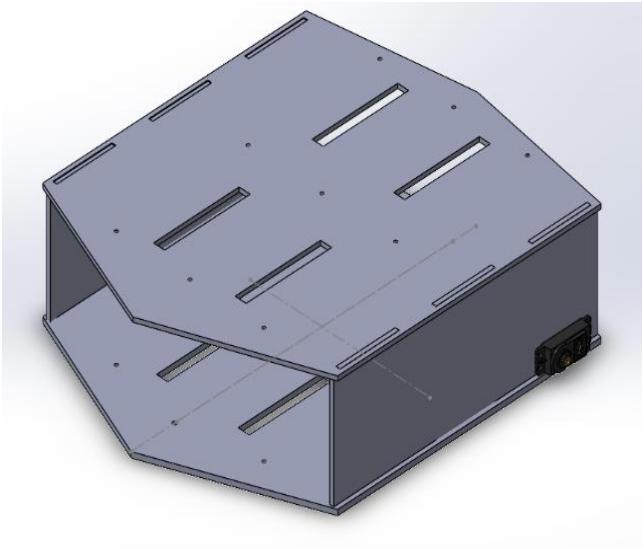


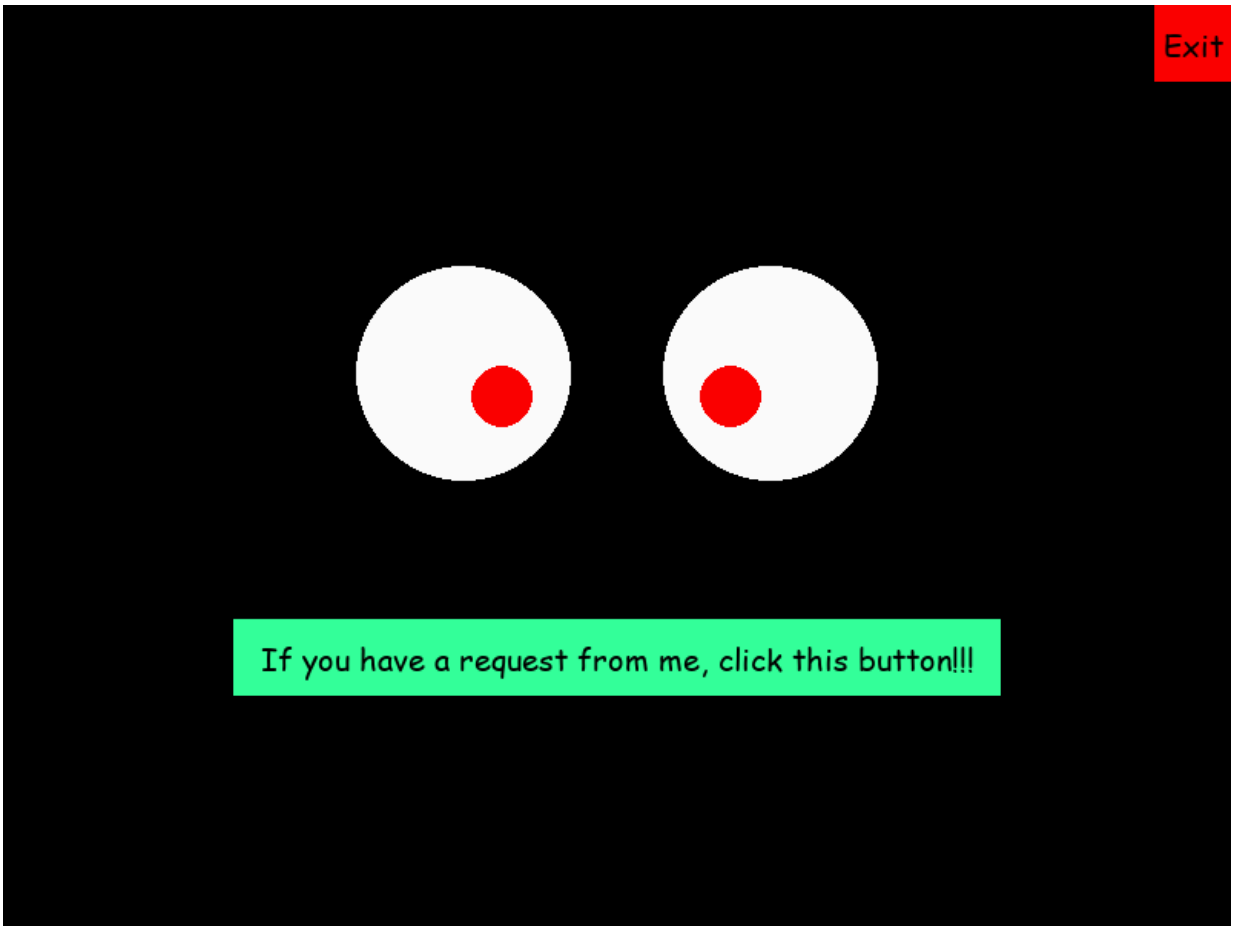
Figure - Modeling the Robot Body

Shape - Chassis Body Design

Voice assistant and easy to use in the bot

In this section, we wanted our smart nurse robot to ask a number of questions related to the patient's current condition and environmental conditions in its LCD, so that the user can choose the option according to that question. Also, the robot can read questions to the patient in speaking mode. This is possible with the help of Python and the Google Text-to-Speech and pygame libraries . Google Text-to-Speech Library It converts any English text to voice and saves it as an mp3 file , which we can use to read the questions audibly. We use the pygame library to build the user interface and the questions and options to be displayed in separate slides, respectively. We also design a smiley face as a robot face that moves the mouse and forms trigonometric relationships of the position of the smiley's pupil. Change

For example, consider the following figure:




```
#ساخت صفحه محیط گرافیکی
screen = pygame.display.set_mode((screen_x,screen_y), pygame.FULLSCREEN) #Lcd size:800x480

#رنگ های استفاده شده در محیط گرافیکی (Red,Green,Blue)
backgroundcolor = (0,0,0) #Black
red = (250,0,0)
white = (250,250,250)

def draw_eye(eye_x, eye_y):
    mouse_x, mouse_y = pygame.mouse.get_pos()

    distance_x = mouse_x - eye_x
    distance_y = mouse_y - eye_y

    distance = min(math.sqrt(distance_x**2 + distance_y**2), 30)
    angle = math.atan2(distance_y, distance_x)

    pupil_x = int(eye_x + (math.cos(angle) * distance))
    pupil_y = int(eye_y + (math.sin(angle) * distance))

    pygame.draw.circle(screen,white,(eye_x,eye_y),70)
    pygame.draw.circle(screen,red,(pupil_x,pupil_y),20)

def text_objects(text, font):
    textSurface = font.render(text, True, (0,0,0))
    return textSurface, textSurface.get_rect()
```

Engine

Initially, the SERVO 360 MG995 motor was considered, but the team realized after testing that this motor is not suitable for transferring power to the wheels and cannot move the robot. This is because this motor is more suitable for dynamic arms.

Figure - SERVO 360 MG995

To achieve this, 12-volt geared motors were used, which provided adequate power.

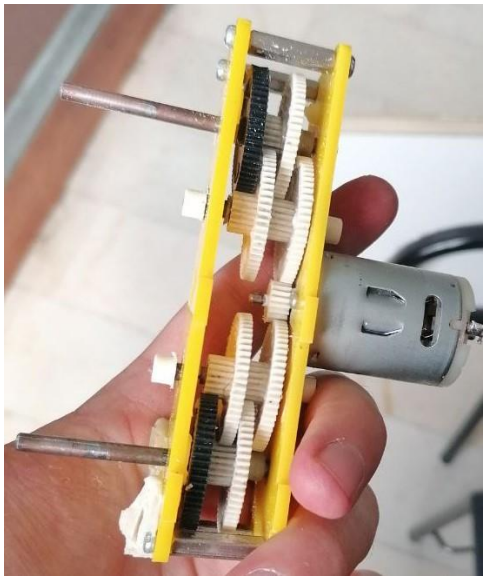


Figure - 12 Volt Gearbox

It was necessary to make modifications to the chassis, which included drilling into the body. However, a fundamental issue we encountered was that the motor shaft did not match the wheels we had purchased. We had to either mill the wheels or use adhesive to adjust the shaft size to fit.

Issues Encountered

Image Processing Team

- During image processing at high speeds, the robot sometimes veered to one side even after detecting the black line, due to significant differences in the maximum speed and base motor speed.
- Tuning the PID controller.
- Voltage fluctuations caused the ESP32 board to burn out.

Smart Curtain and Lighting Team in Patient Room

- Encountered issues while running the code, resulting in the driver shutting down.

Chassis Design Team

- Modifications to the chassis were necessary due to the motor replacement.
- Lack of a filter in the Z-axis depth.
- Design breakdown.

Voice Assistant and User-Friendly Robot Team

- While running the code, we faced issues that prevented us from displaying images on the LCD.

Motor Team

- Initially, the SERVO 360 MG995 motor was considered, but the motor team realized after testing that this motor was not suitable for transferring power to the wheels and could not move the robot, as it is more appropriate for dynamic arms. Therefore, 12-volt geared motors were used, which provided adequate power.