Electrical team Mega Project

Team 3

2024-08-20

Abstract

This project aims to develop a wireless-controlled car with both manual and autonomous modes of operation. The car will utilize three ultrasonic sensors, an L298N motor driver, an HC-05 Bluetooth module, and two DC motors to achieve the desired functionality. In the manual mode, the user will be able to control the car wirelessly using a graphical user interface (GUI). In the autonomous mode, the car will be capable of moving parallel to a wall at a fixed distance, which can be set through the GUI. The distance to the wall will be maintained using an ultrasonic sensor and a PID control method, ensuring a stable and consistent distance. The project also includes the design and fabrication of a printed circuit board (PCB) or multiple PCBs to house the various components and provide a compact and efficient solution. In manual and autonomous modes, software tasks such as: video stitching, live camera feed with screenshot button and stereo vision and these are displayed by the GUI was included to make sure that the car moves in a proper way.

Table of Contents

1 Electrical system

1.1 Approach

To design our PCB for the Mega project, we started by creating detailed schematics that outline all necessary components and their connections. Then we made the Footprints of each component that we bought from Makers or Electra to make sure that the dimensions is right. Next, we used Altium designer to layout the board, carefully considering component dimensions and placement for optimal performance and ease of assembly. We'll then prototype the PCB, focusing on precise soldering and testing to identify and resolve any issues.

1.2 Components selection

1.2.1 Microcontroller selection

The Atmega328P-PU (shown in Figure 1) is relatively inexpensive microcontroller. It was chosen for our application as it is compatible with Arduino libraries and development tools which simplifies programming and debugging. It has low power consumption compared to other microcontrollers which is beneficial for our battery-operated project. There is a large community of users and extensive documentation available for the ATmega328P-PU, which can be very helpful for troubleshooting and solving any problem we face.
- 6 PWM channels.
- 6-channel 10-bit ADC.
- Timers/Counters: Two 8-bit and one 16-bit Timer/Counter.
- USART: Programmable Serial USART.
- SPI: Master/Slave SPI Serial Interface.
- I2C: Byte-oriented 2-wire Serial Interface.

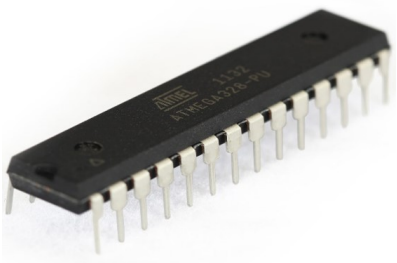


Figure 1: Atmega 328P-PU

1.2.2 Ultrasonic Sensors (HC-SR04)

• it is used for non-contact distance measurements. This sensor operates by emitting an ultrasonic sound pulse and measuring the time it takes for the echo to return to calculate the distance to an object. This sensor provides 2cm to 400cm of non-contact. This sensor includes a transmitter, receiver, and a control circuit. There are four pins Vcc, Trig, Echo(receive) and GND.
-Operating Voltage: 5 Volt DC.
-Operating Current: 15 mA.
-Operating Frequency: 40 HZ.
-Measuring Angle: [15-30].

-Ranging Distance: [2cm-4m].
-Trigger input signal: 10us TTL pulse.
-Echo Output Signal: input TTL lever signal and the range in proportion.

Figure 2

1.2.3 DC geared motor

• this motor has a double-shaft.it is combination of a motor and gearbox. The addition of a gearbox to a motor reduces the speed while increasing the torque output.
-the dual shaft turns up to about 600 rpm.
-Powered by [3,12] volts.
-draws an average of 190 mA of current (max. 250 mA).
-Speed: 600 (+,-) 10 rpm at 12 volt.
-Gearbox: 1:48

Figure 3

1.2.4 L293D

• channel motor driver IC used for controlling the direction and the speed of two DC motors simultaneously and independently.it includes built-in diodes to protect against back EMF from the motors.it also includes thermal shutdown protection to prevent damage due to overheating. The 4 center pins connected and used for heatsinking.
-Output Current: each channel can deliver up to 600 mA of continuous current.
-Maximum current: 1.2 Amp.
-Input Voltage: 7 Volt.
-Enable Voltage: 7 Volt.
-Vs: [4.5,36] Volt.
-Vss: [4.5,36] Volt.
At last, it was confirmed by all mates that the L298N module will be used. Because the L293D IC did not match our PCB size and the tracks.

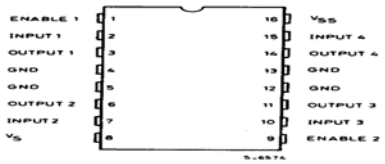


Figure 4

1.2.5 L298N module

The circuit will allow you to easily and independently control two motors of up to 2A each in both directions. It is ideal for robotic applications and well suited for connection to a microcontroller requiring just a couple of control lines per motor. It can also be interfaced with simple manual switches, TTL logic gates, relays, etc. This board equipped with power LED indicators, on-board +5V regulator and protection diodes.

- Input Voltage: 3.2V 40Vdc.
- Driver: L298N Dual H Bridge DC Motor Driver
- Power Supply: DC 5 V - 35 V
- Peak current: 2 Amp
- Operating current range: 0 36mA
- Control signal input voltage range:
- Maximum power consumption: 20W (when the temperature T = 75 ℃).
- Storage temperature: -25 ℃ +130 ℃.
- On-board +5V regulated Output supply (supply to controller board i.e. Arduino).
- Size: 3.4cm x 4.3cm x 2.7cm

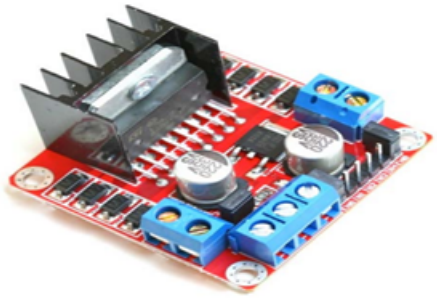


Figure 5

1.2.6 RGB LED

combine red, green, and blue light to produce a wide range of colors and can be controlled using PWM to adjust the brightness of each color channel.
- Red:
Forward Voltage: [1.8,2.2] volt.
Forward Current: 20 mA.
Reverse Current: 10 uA.
Operating Temperature: [-25,85] c.
- Green:
Forward Voltage: [3,3.4] volt.
Forward Current: 20 mA.
Reverse Current: 10 uA.

Operating Temperature: [-25,85] c.
- Blue:
Forward Voltage: [3,3.4] volt.
Forward Current: 20 mA.
Reverse Current: 10 uA.
Operating Temperature: [-25,85] c.
And for all the maximum current is 30 mA.

Figure 6

1.2.7 Heat Sink

• The LM7805's internal components (mainly the pass transistor) convert the excess energy into heat. If not effectively managed, this heat can cause the regulator to overheat, potentially leading to thermal shutdown or permanent damage. Also, the regulator includes a thermal shutdown feature that disables the regulator if it gets too hot. While this protects the regulator, it stops the circuit from functioning until the regulator cools down.so we need a heat sink. And we choose the aluminum heatsink u-shape for TO-220 transistor. **How a Heatsink Works:** A heatsink increases the surface area available for heat dissipation. This allows more heat to be transferred from the regulator to the surrounding air. And are made of materials with high thermal conductivity, like aluminum or copper, which efficiently transfer heat away from the regulator. The effectiveness of a heatsink is measured by its thermal resistance (°C/W), which indicates how effectively it transfers heat. Lower thermal resistance means better heat dissipation.
• The protection requirements for the microcontroller (ATmega328P) were researched, and a report was prepared on the findings.

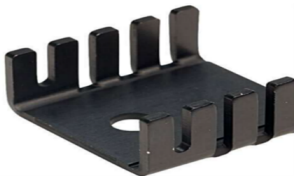


Figure 7

1.3 Schematic design

After the selection of the components that is going to be used in our project the schematic design was made to make sure that the components function together safely as shown in Figure ().

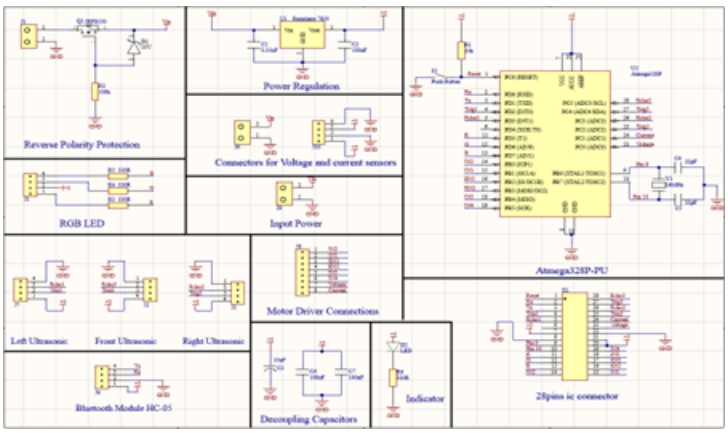


Figure 8

1.4 Reverse Polarity Protection

1.4.1 Components Used

100k Resistor is used to keep the gate voltage at a known level (ground) when there is no input and to make sure that any residual charge on the gate is discharged to ground which prevents the MOSFET from accidentally turning on when the circuit is off. Since the MOSFETs have a maximum gate-source voltage rating exceeding this voltage can damage the MOSFET therefore a **10V Zener Diode** is used for protection of MOSFET by clamping the overvoltage at the gate to a maximum of 10V. The 10V Zener diode ensures that the gate voltage stays within safe limits. **IRFN530 MOSFET** acts as a switch that controls the flow of current in the circuit. It is used as it has low On-Resistance (0.2 ohms) which ensures minimal voltage drop and power dissipation when the MOSFET is on.

1.4.2 Working Principle

When the power supply is connected correctly, the gate of the MOSFET receives the appropriate signal to turn on which allows current to flow from the drain to the source, powering the circuit. If the power supply is connected in reverse the MOSFET remains off as the 100k resistor ensures the gate is pulled to ground, preventing accidental turn-on. The zener diode protects the gate from any overvoltage that might occur due to incorrect connections.

1.5 Power Regulation

1.5.1 Components Used

The 7805 Regulator is used for providing a stable and regulated 5V output, which is ideal for powering microcontroller, sensors, LEDs and the Bluetooth module. It can deliver up to 1.5A of current with proper heat sinking, which is sufficient for many applications. It has internal current limiting, thermal shutdown, and safe area protection, making it robust and reliable.
7805 Regulator specifications
- Output Voltage: 5V DC (±2% tolerance)
- Input Voltage Range: 7V to 35V (7V to 25V is more common for stable operation), in

our case 9v will work well.
- Maximum Output Current: 1.5A (with proper heat sinking) - Operating Temperature: 0°C to 125°C.
The 0.33µF capacitor is placed at the input of the 7805 regulator to filter out high-frequency noise from the power supply. This ensures that a smooth DC voltage enters the regulator, which is crucial for its stable operation.
The 100nF capacitor is placed at the output of the 7805 regulator to smooth the DC voltage output, reduce ripple, and provide a stable voltage supply to the load.
(The values of the previous capacitors is recommended by the manufacturer in the datasheet as shown in the following Figure)

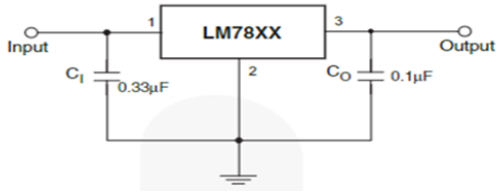


Figure 9: Recommended connection of the 7805

1.6 Decoupling capacitors

Since the microcontroller is a sensitive IC which requires a stable 5 DC voltage to operate efficiently so a 100nF capacitor is used between Vcc and ground also between AVcc and the ground in the ATmega328P-PU microcontroller (should be placed as close as possible to the microcontroller) to filter out high-frequency noise from the power supply, ensuring a stable voltage level for the microcontroller. The capacitor absorbs voltage spikes and transients that can occur when the microcontroller switches states, protecting the circuit from potential damage.

1.7 System Clock

1.7.1 Components Used

The 16MHz crystal is the standard clock speed for the ATmega328P-PU. It is widely used and well-documented, making it easier to find support and resources for our project. At 16MHz, the microcontroller can perform up to 16 million instructions per second (MIPS), providing a good balance between performance and power consumption. It is suggested by the manufacturer in the datasheet as shown in Table 1 to use 22pF capacitors to that the ATmega328P-PU operates at a reliable clock speed with stable oscillation, providing accurate timing for serial communication UART (e.g., with the HC-05 Bluetooth module). and overall performance.

Frequency Range (MHz)	Recommended Range for Capacitors C1 and C2 (pF)	CKSEL3..1
0.4 to 0.9	–	100[1]
0.9 to 3.0	12 to 22	101
3.0 to 8.0	12 to 22	110
8.0 to 16.0	12 to 22	111

Figure 10

1.8 PCB Design and Fabrication

1.8.1 Track Clearance and Width

According to JLCPCB's guidelines, the minimum clearance between tracks is typically around 0.127mm. However, 0.7mm clearance is used which provides additional safety margins, making the PCB more robust and easier to manufacture at home. According to JLCPCB Design Rules a minimum track width of 0.2mm. So a 1.2mm track is used to handle higher current loads ensuring that the PCB can handle the power requirements of the motors and other high-current components without overheating. Wider tracks also help in dissipating heat more effectively also it makes the soldering easier.

1.8.2 Ground Polygon

Using a polygon for the ground plane helps in reducing electromagnetic interference (EMI) and provides a stable reference voltage for the circuit. It also simplifies the routing of ground connections.

1.8.3 Vias for Jumper Wires

In a 10x10cm single-layer PCB, routing all connections on one side can be challenging. Vias allow for the insertion of jumper wires, providing a way to route signals that cannot be accommodated on the top layer. It simplifies the design and avoids the need for a double-layer PCB.

1.8.4 Pads and Vias size

We make sure to make the size of the pads and vias as big as possible to make the drilling and soldering easier and preventing the tracks from being corrupted.

1.8.5 Routing

The Routing of the components in a 10x10cm single layer were made probably to make sure that each component in the right place as shown in the following figures

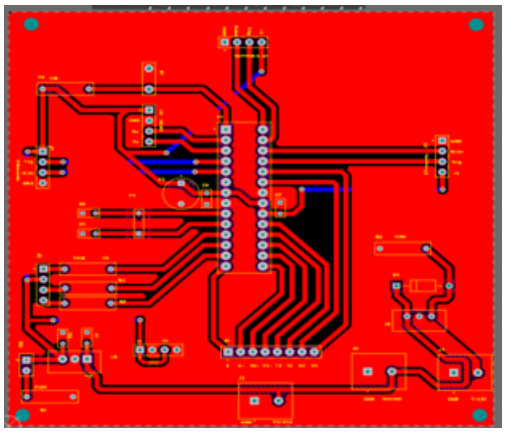


Figure 11: PCB Layout

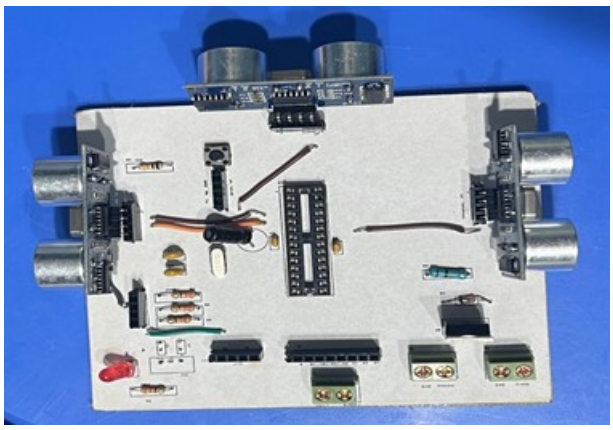


Figure 12: Right placement of the components

1.8.6 Steps for Creating a Single-Layer PCB using through hole components at Home

1.Design Printing: Print the PCB design on glossy paper to scale, ensuring that the holes in the pads are visible. Top Overlay Printing: Print the top overlay sticker to scale for easier component placement later.
2. PCB Cleaning: Clean the single-layer PCB with dish wire to remove any oxidation and ensure a smooth surface for the transfer process.
3.Ironing the Design: Place the glossy paper with the printed design onto the copper side of the PCB. Use an iron to transfer the ink from the paper to the copper surface.
4.Etching Process: Submerge the PCB in an etching solution, such as Ferric Chloride (Hexahydrate), to remove the unused copper. Only the tracks and polygons from the design should remain.
5. Hole Drilling: Drill the holes in the PCB where indicated by the design to accommodate component leads.
6. Sticker Application: Apply the top overlay sticker to the opposite side aligned with the drilled holes in the PCB to assist with component placement.
7. Component Placement: Place each component on the PCB then solder it.
8. Testing: Use a Avometer to check that each pin of the components is connected only to its designated track and that there are no short circuits between tracks.

1.9 Considerations

At first 3 days of the Mega project we tried to design all the modules that where going to be used in the project but we faced a problem routing all of this components on a 10x10cm single layer PCB. The schematics of the first design and all Footprints are shown in the following Figures.

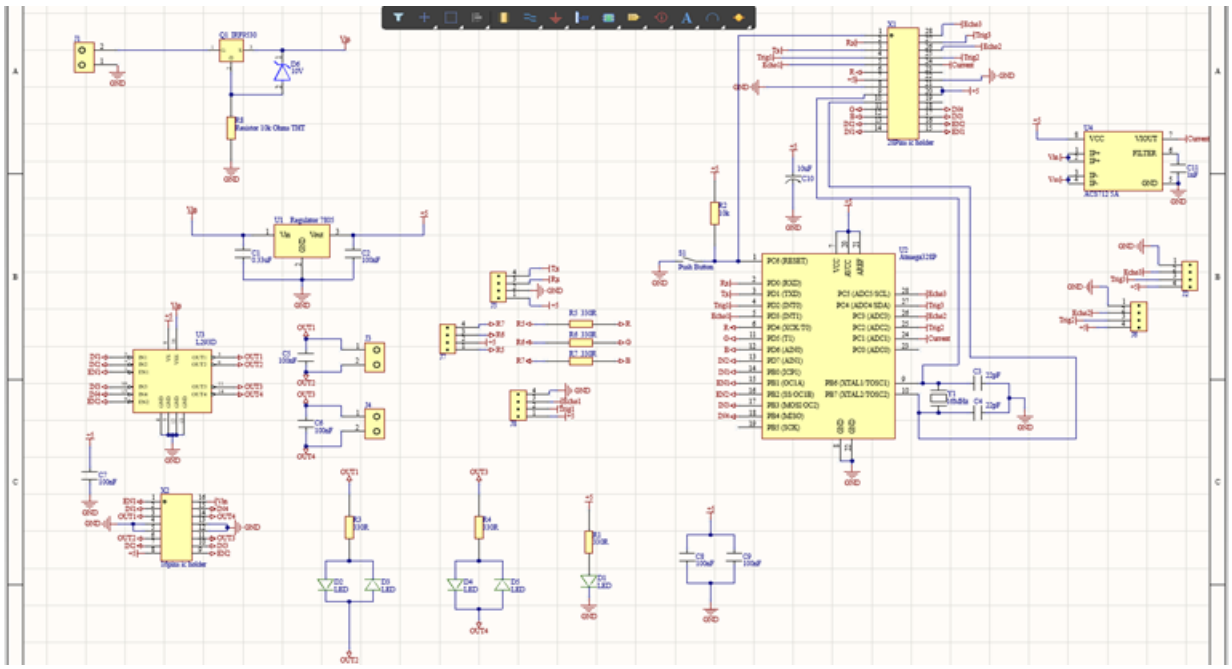


Figure 13: Schematic of all components including (L293D and ACS712 current sensor)

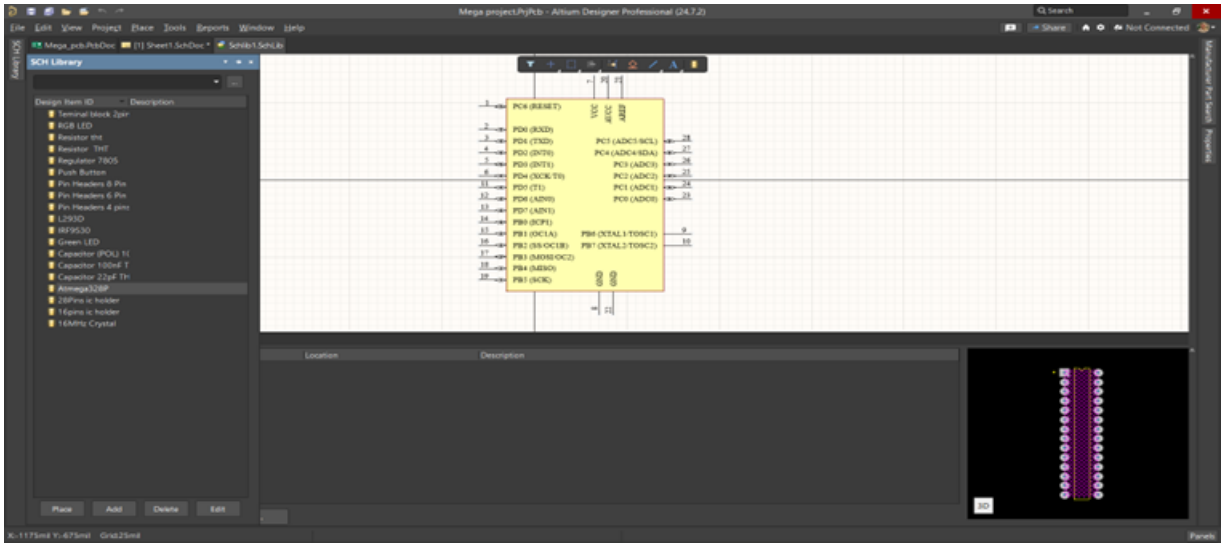


Figure 14: List of all Footprints made

1.10 Power Calculations

- ATmega328P-PU: Approximately 10-20mA.
- 3 x Ultrasonic Sensors (HC-SR04): Each sensor draws about 15mA, so total is 45mA.
- HC-05 Bluetooth Module: Approximately 30mA.
- L293D IC: Approximately 60mA (without motor load).
- RGB LED with 3 x 330 Resistors: Approximately 20mA per color, so total is 60mA.
- Green LED with 330 Resistor: Approximately 10mA.
- 2 x Geared DC Motors: Each motor draws 190-250mA, so total is 380-500mA operates in range 3 12Volts.

Total Current Draw is 725mA (The voltage requirements were considered in the components section)

1.10.1 Battery Capacity

our battery pack consists of 3 x 3.7V 1500mAh batteries in series, providing a total voltage of 11.1V and a capacity of 1500mAh. Assuming the 7805 regulator has an efficiency of around 70% (typical for linear regulators), the effective capacity is reduced to 1050mAh. So our 3 x 3.7V 1500mAh batteries should be sufficient to power your setup for approximately 1.45 hours under continuous operation.

2 Firmware system

2.1 Approach

First thing we wanted to do is to establish a system for the car to move by sending a signal to the ATmega328p to do a certain function so, we created functions for every command in the car as : **auto motion , turn right , turn left ,speed select , move forward ,...and so on** and after creating this functions we want to select only one so,we made a switch case to select one according to the signal sent from GUI or from Serial monitor.

2.2 Flowchart

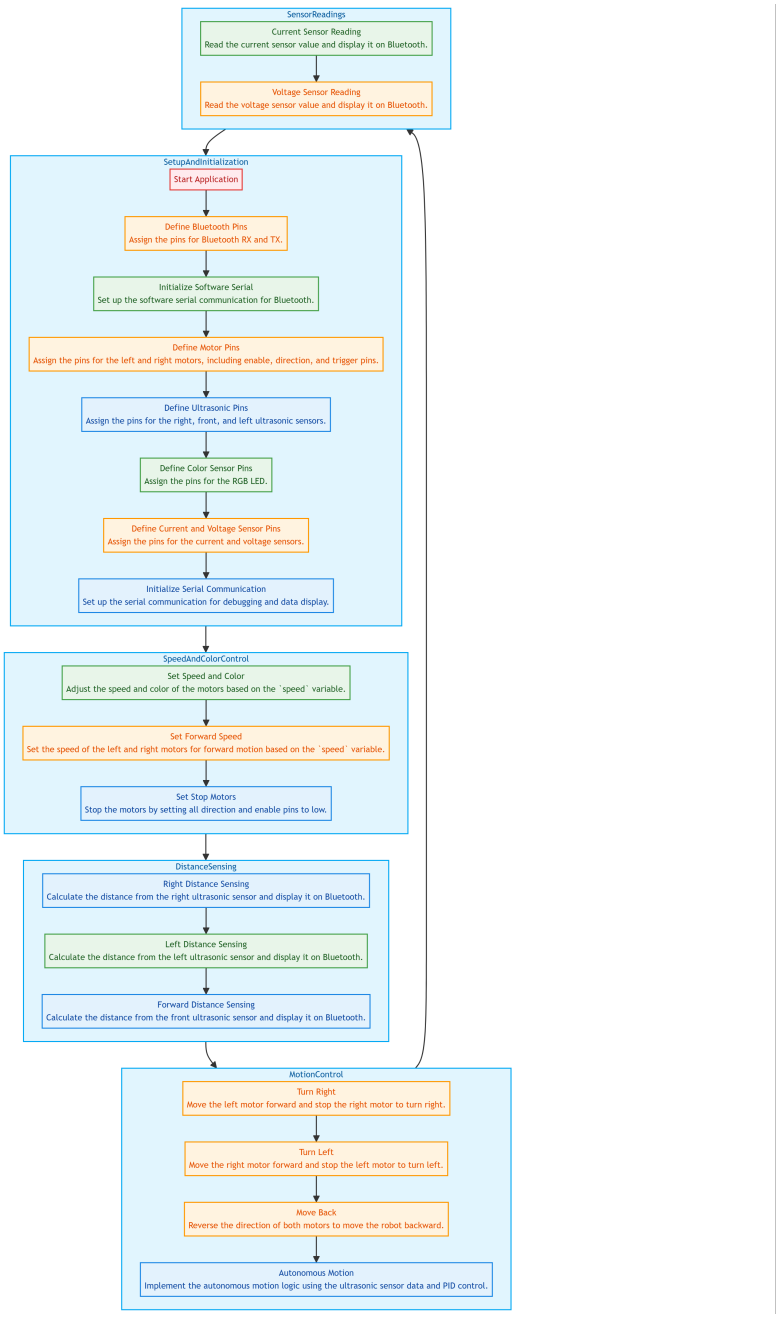


Figure 15: Arduino code flowchart

2.3 Explanation

We first have to define the Bluetooth module to the IDE and setting pins to it .

```
// C++ code
// 1&2 for left motor
//3&4 for right motor

// defining bluetooth pins
const int BTRX  = 1;
const int BTTX = 0;

// We use software serial to avoid conflicts
// with the default RX/TX  pins of the Arduino board
#include <SoftwareSerial.h>
SoftwareSerial SerialBT(BTRX,  BTTX);  // to establish the bluetooth
```

Figure 16: setting Bluetooth

Then we defined the pins of the motor and the sensors and initializing some variables for using them in voltage sensor and current sensor and the ultrasonic reading and defining speed variable to be controlled later .

```
//define the used pins
# define E_R 11
# define E_L 10
# define I1 9
# define I2 8
# define I3 12
# define I4 13
# define T_L 2
# define T_R A2
# define T_F A5
# define Echo_R A3
# define Echo_F A4
# define Echo_L 3
# define Red 5
# define Blue 6
# define Green 7
# define C_S_Pin A1
# define V_S_Pin A0
const float referenceVoltage = 5000.0; // 5V to be used for voltage sensor
int speed = 0; // t initiallize the speed the modify it to elect it later
long duration_R, duration_L, duration_F, distance_F, distance_R, distance_L; //to be used with ultrasonic
int direction;  //1for left ,2 for right because tinker doesn't compare strings
float vcc = 5.0;     // Voltage of the Arduino board
float adcResolution = 1024.0;  // ADC resolution (10-bit)
float currentSensorOffset = 2.5; // Sensor offset voltage
float currentSensorScale = 0.185; // Sensor scaling factor (185 mV/A)
```

Figure 17: defining pins and initializing variables

Then we defined each pin whether it's input or output and begin the serial using Bluetooth or from computer.

```
39  void setup()
40  {
41    SerialBT.begin(9600);//to set up the bluetooth
42    pinMode(E_R,OUTPUT);
43    pinMode(E_L,OUTPUT);
44    pinMode(I1,INPUT);
45    pinMode(I2,INPUT);
46    pinMode(I3,INPUT);
47    pinMode(I4,INPUT);
48    pinMode(T_L,OUTPUT);
49    pinMode(T_F,OUTPUT);
50    pinMode(T_R,OUTPUT);
51    pinMode(Echo_R,INPUT);
52    pinMode(Echo_F,INPUT);
53    pinMode(Echo_L,INPUT);
54    pinMode(Red,OUTPUT);
55    pinMode(Blue,OUTPUT);
56    pinMode(Green,OUTPUT);
57    Serial.begin(9600);
58  }
```

Figure 18: defining pins and initializing variables

After that we started Creating the functions ; first function is the **speed colour()** which controls the colour of LED according to speed selected.

```
55  void speed_colour(){//function to set the color of RGB LED according to speed
56    if (speed==1){
57      analogWrite(Green,255);
58      analogWrite(Blue,0);
59      analogWrite(Red,0);
60    }
61    else if (speed==2){
62      analogWrite(Green,0);
63      analogWrite(Blue,255);
64      analogWrite(Red,0);
65    }
66    else if (speed==3){
67      analogWrite(Green,0);
68      analogWrite(Blue,0);
69      analogWrite(Red,255);
70    }
71    else{
72      analogWrite(Green,0);
73      analogWrite(Blue,0);
74      analogWrite(Red,0);
75    }
76  }
```

Figure 19: Speed colour function

The next function is **Forward speed()** which controls the speed of motors from three speeds according to the selected speed.

```
77  void Forward_speed(){ //function to set the speed of motors according to speed
78    if(speed==1){
79      digitalWrite(I1,HIGH);
80      digitalWrite(I2,LOW);
81      digitalWrite(I3,HIGH);
82      digitalWrite(I4,LOW);
83      analogWrite(E_R,115);
84      analogWrite(E_L,85);
85    }
86    else if(speed==2){
87      digitalWrite(I1,HIGH);
88      digitalWrite(I2,LOW);
89      digitalWrite(I3,HIGH);
90      digitalWrite(I4,LOW);
91      analogWrite(E_R,200);
92      analogWrite(E_L,170);
93    }
94    else if(speed==3){
95      digitalWrite(I1,HIGH);
96      digitalWrite(I2,LOW);
97      digitalWrite(I3,HIGH);
98      digitalWrite(I4,LOW);
99      analogWrite(E_R,255);
100     analogWrite(E_L,225);
101   }
102   else {
103     digitalWrite(I1,LOW);
104     digitalWrite(I2,LOW);
105     digitalWrite(I3,LOW);
106     digitalWrite(I4,LOW);
107     analogWrite(E_R,0);
108     analogWrite(E_L,0);
109   }
```

Figure 20: Forward speed function

We initialized the PID constants then implemented the sensors functions to read distance from (right,left and forward).

```
131 void Right_distance(){// function to calculate distance from ultrasonic on the right
132   digitalWrite(T_R, HIGH);
133   delayMicroseconds(10);
134   digitalWrite(T_R, LOW);
135   duration_R = pulseIn(Echo_R, HIGH);
136   distance_R=(duration_R/2.0)*0.0343;
137   SerialBT.print("distance_R: "); //used BT to show results on bluetooth
138   SerialBT.print(distance_R);
139 }
140 double kp_L = 1.0, ki_L = 0.1, kd_L = 0.05; // PID constants without testing , should be modified by testing
141 double setpoint_L = 35.0; // Desired distance in cm
142 double input_L, output_L, error_L, last_error_L = 0.0;
143 double integral_L = 0.0, derivative_L = 0.0;
144 void Left_distance(){// function to calculate distance from ultrasonic on the left
145   digitalWrite(T_L, HIGH);
146   delayMicroseconds(10);
147   digitalWrite(T_L, LOW);
148   duration_L = pulseIn(Echo_L, HIGH);
149   distance_L=(duration_L/2.0)*0.0343;
150   SerialBT.print("distance_L: ");//used BT to show results on bluetooth
151   SerialBT.print(distance_L);
152 }
```

```
153  double kp_F = 1.0, ki_F = 0.1, kd_F = 0.05; // PID constants  without testing , should be modified by testing
154  double setpoint_F = 35.0; // Desired distance in cm
155  double input_F, output_F, error_F, last_error_F = 0.0;
156  double integral_F = 0.0, derivative_F = 0.0;
157  void Forward_distance(){// function to calculate distance from ultrasonic in front of the car
158    digitalWrite(T_F, HIGH);
159    delayMicroseconds(10);
160    digitalWrite(T_F, LOW);
161    duration_F = pulseIn(Echo_F, HIGH);
162    distance_F=(duration_F/2.0)*0.0343;
163    SerialBT.print("distance_F: ");//used BT to show results on bluetooth
164    SerialBT.print(distance_F);
165  }
```

Figure 21: sensors

Then we made functions to make the car turn left or right by stoping one motor and operating the other.

```
162  void Turn_Right(){ // function to move the left motor and stop the right one to turn right
163    digitalWrite(I1,HIGH);
164    digitalWrite(I2,LOW);
165    digitalWrite(I3,LOW);
166    digitalWrite(I4,LOW);
167    analogWrite(E_R,0);
168    analogWrite(E_L,255);
169    delay(100);
170
171  }
172  void Turn_Left(){// function to move the right motor and stop the left one to turn left
173    digitalWrite(I1,LOW);
174    digitalWrite(I2,LOW);
175    digitalWrite(I3,HIGH);
176    digitalWrite(I4,LOW);
177    analogWrite(E_R,255);
178    analogWrite(E_L,0);
179    delay(100);
180
181  }
```

Figure 22: turning functions

Then we made a function to reverse the direction of motors and move back .

```
183  void move_back(){ // function to reverse directions of motors to move back
184    digitalWrite(I1,LOW);
185    digitalWrite(I2,HIGH);
186    digitalWrite(I3,LOW);
187    digitalWrite(I4,HIGH);
188    analogWrite(E_R,200);
189    analogWrite(E_L,170);
190  }
```

Figure 23: moving back

After that we implemented autonomous function for the automatic motion of car depending on sensors where the car adjust it's direction according to the PID equation.

```
void Auto_motion(){// function for autonomous motion
  Forward_distance();//called it to know the forward space
  Left_distance();//called it to know the left space
  Right_distance();//called it to know the right space
  Forward_speed();//to move in straight line
  delay(250);
  if(distance_F<35){ // check forward limits if smaller than 35 it moves back and controlled by PID
    error_F = setpoint_F - distance_F;

  // Calculate the PID terms
  integral_F += error_F * ki_F; // Integral term (0.1 is the time step in seconds)
    derivative_F = (error_F - last_error_F) / 0.1; // Derivative term (0.1 is the time step in seconds)

  // Calculate the PID output
    output_F = (kp_F * error_F) + (ki_F * integral_F) + (kd_F * derivative_F);
    distance_F=output_F;
  // Update the last error
    last_error_F = error_F;
    move_back();
  }
  else if (distance_R<35){// check right limits if smaller than 35 it moves left and controlled by PID
    error_R = setpoint_R - distance_R;

  // Calculate the PID terms
  integral_R += error_R * ki_R; // Integral term (0.1 is the time step in seconds)
    derivative_R = (error_R - last_error_R) / 0.1; // Derivative term (0.1 is the time step in seconds)

  // Calculate the PID output
    output_R = (kp_R * error_R) + (ki_R * integral_R) + (kd_R * derivative_R);
    distance_R=output_R;
  // Update the last error
    last_error_R = error_R;
    Turn_Left();
  }
  else if(distance_L<35){// check left limits if smaller than 35 it moves right and controlled by PID
    error_L = setpoint_L - distance_L;

  // Calculate the PID terms
  integral_L += error_L * ki_L; // Integral term (0.1 is the time step in seconds)
    derivative_L = (error_L - last_error_L) / 0.1; // Derivative term (0.1 is the time step in seconds)

  // Calculate the PID output
    output_L = (kp_L * error_L) + (ki_L * integral_L) + (kd_L * derivative_L);
    distance_L=output_L;
  // Update the last error
    last_error_L = error_L;
    Turn_Right();
  }
}
```

Figure 24: Autonomous function

Finally , we implemented functions that read volt and current from one pin and modify this reading through each sensor equation to match the real value.

```
void current_sensor(){//reads the current sensor
  int adcValue = analogRead(C_S_Pin);
  float currentValue = (adcValue / adcResolution * vcc - currentSensorOffset) / currentSensorScale;// Calculate the current value
  SerialBT.print(currentvalue)
}
void volt_sensor(){// reads the volt sensor
  int sensorValue =analogRead(V_S_Pin);

  // Convert the sensor value to voltage
  float voltage = (sensorValue * referenceVoltage) / 1024.0;
  SerialBT.print(voltage);
}
```

Figure 25: volt and current

After finishing the functions ,the loop takes input from user and selects which function to execute.

```
int pro=0;
void loop() {
  int fun = SerialBT.parseInt();//takes input from bluetooth to call a function later from switch case
  if (fun!=0){//to save the called function in varialble(pro) untill calling another
    pro=fun;
  }
  SerialBT.println(pro);// to print which process we are calling
    // Handle the different cases non-blockingly
  switch (pro) {
    case 1:
      speed = 1;
      speed_colour();
      break;
    case 2:
      speed = 2;
      speed_colour();
      break;
    case 3:
      speed = 3;
      speed_colour();
      break;
    case 4:
      Auto_motion();
      current_sensor();
      volt_sensor();
      break;
    case 5:
      Forward_speed();
      break;
    case 6:
      move_back();
      break;
    case 7:
      Turn_Right();
      break;
    case 8:
      Turn_Left();
      break;
    case 9:
      stop();
      break;
  }
  delay(100);
}
```

Figure 26: The loop

And here is a simulation link and screen shot for the functions on tinker cad: https://www.tinkercad.com/things/24p6XnmEoFc-project

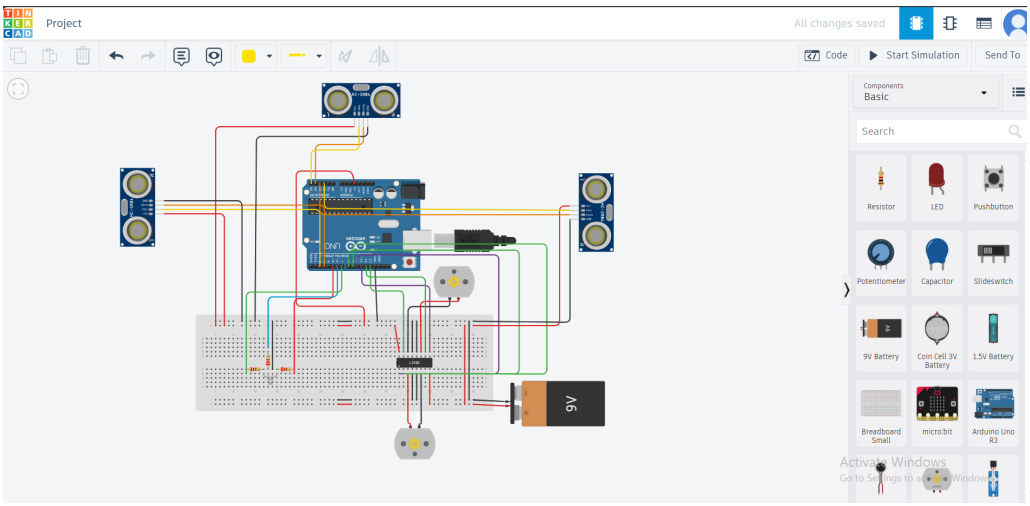
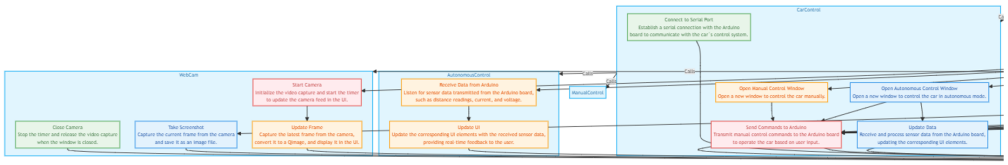


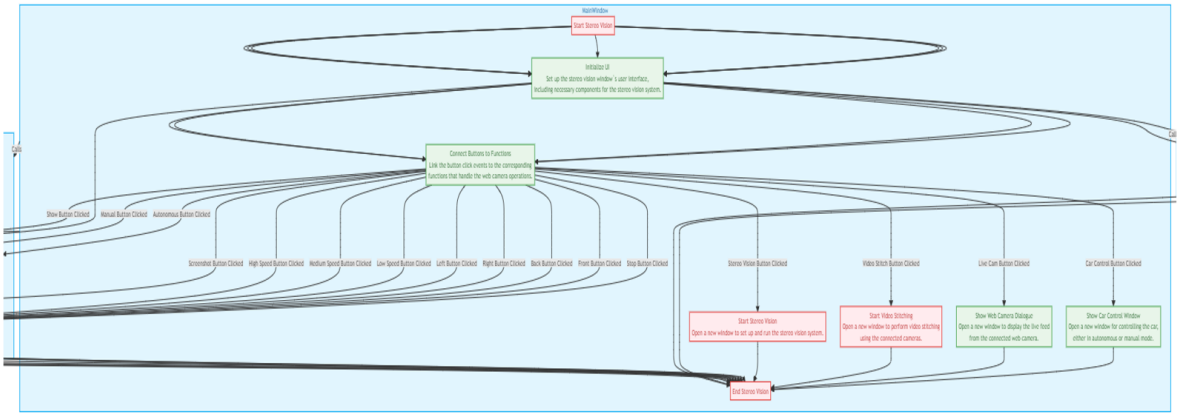
Figure 27: The loop

3 Software

3.1 GUI

3.1.1 Flowchart





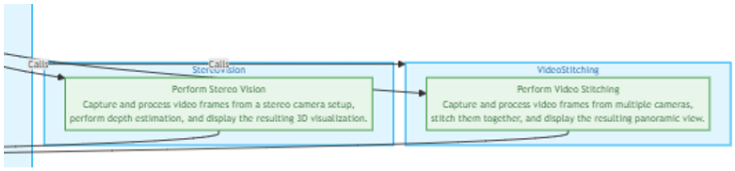


Figure 28: GUI Flowchart

3.1.2 Main Window Features

The main window of the GUI includes 4 push buttons each for different Feature. This was done by Qt Designer app and converted to a python file named frontendtest.py.

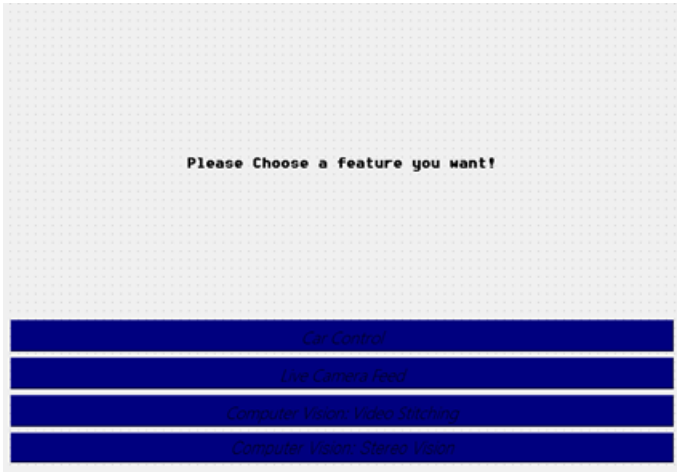


Figure 29: Main window

The above picture shows the design of the main window. A label was added to the main window and changed to the shown text, Font, Font size and colors were adjusted accordingly. The layout of the window was set to a vertical layout. Moreover, four push buttons were added each one for a different feature that once clicked on, a sub window will be shown. Adjusted the colors, font styles to the push buttons to be as neat as possible. Finally used the command "pyuic6 -x test.ui (name of the designer file) -o frontendtest.py (name of the python file)". For managing the backend, a backend file was created named "backendtest1.py". To run the file and the above picture appears, "from frontendtest import Ui_MainWindow as Ui_MainWindow_main" this sentence was written.

```
class MainWindow(QMainWindow):
    def __init__(self, parent=None) -> None:
        super().__init__(parent)
        self.ui = Ui_MainWindow_main()
        self.ui.setupUi(self)
```

Figure 30

In the above picture it initializes the MainWindow class and sets up the user interface.

```
if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec())
```

Figure 31

3.2 Sub Windows Features

3.2.1 Car control

On the main window there is a button named "Car Control" which once clicked on by the mouse it executes another sub-window.

```
self.ui.carcontrolbutton.clicked.connect(self.show_new_window)
```

Figure 32

The picture above shows how the car control button works. When it is clicked on it calls a function named "show_new_window". The below picture shows the new function.

```
def show_new_window(self):
    self.w = CarControl()
    self.w.show()
```

Figure 33

This function calls a class named "CarControl" and shows the execution of this class.

```python
class CarControl(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('Car Control')
        layout = QVBoxLayout()

        # Setting up label
        self.label = QLabel("Car Control")
        layout.addWidget(self.label)

        # Creating and setting up buttons
        self.autonomousButton = QPushButton("Autonomous")
        layout.addWidget(self.autonomousButton)
        self.manualButton = QPushButton("Manual")
        layout.addWidget(self.manualButton)
        self.manualButton.clicked.connect(self.open_manual_control)
        self.autonomousButton.clicked.connect(self.open_autonomous_control)
        self.autonomousButton.clicked.connect(self.Auto_sent)
        self.autonomousButton.clicked.connect(self.update_data)
        self.setLayout(layout)
```

Figure 34

In this class, the __init__ method initializes the class but only calls the base class constructor using super().__init__(). A new sub-window is implemented programmatically, rather than using the Qt Designer application. The window title is set, the layout is configured to be vertical, and two push buttons are added to represent the two modes of the car: Autonomous and Manual. The design of the sub-window is illustrated in the picture below.

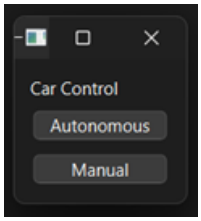


Figure 35

Each button send you to another sub-window which are both done by the Qt designer app. A function named "open_manual_control" executes when the manual button is clicked. The below image shows the design of the manual window.

Figure 36

A blue frame has been added to enhance the design. Two push buttons were included for controlling the car: one to start and one to stop the car.

Additionally, four push buttons indicate the car's movement directions: forward, left, right, and backward. The icons on these buttons, representing the movement directions, are downloaded from the internet. Furthermore, three radio buttons were added to allow the user to select the car's speed. This designer file was converted to python file using the same command as before but with different file names.

```
from frontendManual import Ui_ManualControl
```

```
self.manualButton.clicked.connect(self.open_manual_control)
```

Figure 37

```
def open_manual_control(self):
    #setting up manual window
    self.manual_window = QMainWindow()
    self.ui_manual = Ui_ManualControl()
    self.ui_manual.setupUi(self.manual_window)
    #showing manual window
    self.manual_window.show()
```

Figure 38

The pictures above illustrate how the manual control window was implemented and connected to the car control window. The frontendManual file, which contains the manual control window design, is imported to access its main window class. When the "Manual" button is clicked in the car control window, it triggers a function that displays the manual control window.In this function, an instance of the MainWindow class from frontendManual is created. The user interface for this window is then set up, and the manual control window is displayed.

The other mode of the car control is the autonomous mode. A Qt designer is used for the design of its window as shown in the below image.

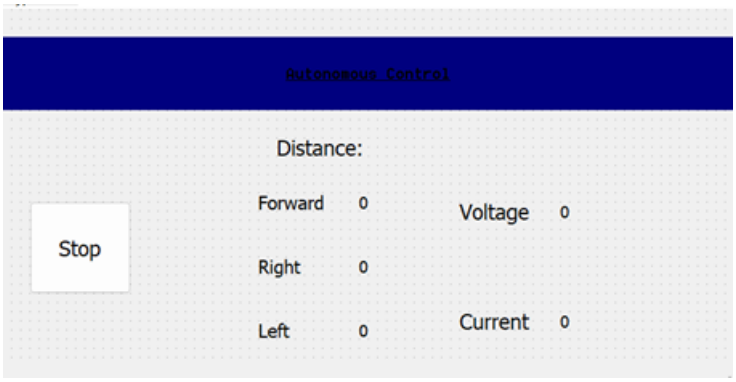


Figure 39

A blue frame has been added to enhance the design. A push button is added to stop the car once clicked in it. Lots of labels are added to the window indicating readings such as: distance (forward, right, and left distances), voltage, and current. This designer file was converted to python file using the same command as before but with different file names.

```
from autonomousfrontend import Ui_MainWindow as Ui_MainWindow_auto
```

```
self.autonomousButton.clicked.connect(self.open_autonomous_control)
```

```
def open_autonomous_control(self):
    self.autoWindow = QMainWindow()
    self.ui_auto = Ui_MainWindow_auto()
    self.ui_auto.setupUi(self.autoWindow)
```

Figure 40

The pictures show how the window is shown once clicked in the autonomous button. It is implemented just like the manual implementation. Serial connection of the Arduino and GUI: To realistically communicate with the serial connection between the Arduino and the GUI is established. This is done by sending and receiving messages from and to the Arduino. First thing to do is import serial in the python file. Next, serial must be connected, and this is done by a function called 'connectSerial' in the CarControl class as shown below.

```python
def connectSerial(self):
    try:
        self.ArduinoSerial = serial.Serial('COM5', 9600)
        print("Serial connection established.")
    except serial.SerialException as e:
        print(f"Error connecting to serial port: {e}")
```

Figure 41

'COM5' is the port to be connected to, '9600' is the baud rate which must be matched to the baud rate in the Arduino. If there is no serial connection is established a message will be printed. After connecting the serial commands must be sent to the Arduino to understand using function send_command.

```python
def send_command(self, command):
    try:
        self.ArduinoSerial.write(command.encode())
        time.sleep(3)
    except Exception as e:
        print(f"Error sending command '{command}': {e}")
```

Figure 42

When writing data to a serial port using PySerial, it is essential to ensure that the data is in the correct format. Serial communication typically requires data to be in bytes, not strings. Python's .encode() method is used to convert a string into bytes, which is necessary for sending data over a serial connection. How is this used in buttons connection and displaying outputs?

```python
# connecting buttons to ide

self.ui_manual.stopButton.clicked.connect(self.stop_button)
self.ui_manual.frontButton.clicked.connect(self.front_button)
self.ui_manual.backButton.clicked.connect(self.back_button)
self.ui_manual.rightButton.clicked.connect(self.right_button)
self.ui_manual.leftButton.clicked.connect(self.left_button)
self.ui_manual.lowSpeed.clicked.connect(self.low_rad_button)
self.ui_manual.mediumSpeed.clicked.connect(self.medium_rad_button)
self.ui_manual.highSpeed.clicked.connect(self.high_rad_button)
```

Figure 43

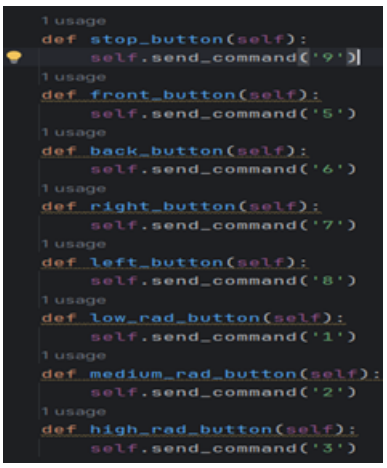


Figure 44

Each button clicked a function is called to send a certain command to the Arduino. How the car executes these commands is controlled by the arduino.

In autonomous data is received from the arduino.

```
def update_data(self):
    try:
        if self.ArduinoSerial.in_waiting > 0:
            # Split data into id and value
            data_bytes = self.ArduinoSerial.readline()
            data = data_bytes.decode('utf-8')
            id, value = data.strip().split(':', 1)
            print(id)
            # Handle the data based on the id
            if id == 'distance_F':
                self.ui_auto.forwardReading.setText(value)
            elif id == 'distance_R':
                self.ui_auto.rightReading.setText(value)
            elif id == 'distance_L':
                self.ui_auto.leftReading.setText(value)
            elif id == 'current':
                self.ui_auto.currentReading.setText(value)
            elif id == 'voltage':
                self.ui_auto.voltageReading.setText(value)
            else:
                print(f"Unknown data id: {id}")
    except ValueError:
        print(f"Failed to parse data: {data}")

    self.autoWindow.show()
```

Figure 45

In this function data is received from the serial and text is set in the autonomous window. .readline reads data in bytes so it needs to be decoded. Data is sent by a specific format which is the name of the object reading is taken from then a colon then the actual reading. These are splitted and the ':' is the delimiter a variable id is the name, the variable value is the reading. By checking the id if it matches the name so this object is set according to the value, if it doesn't match the name a message is printed as shown.

```
def Auto_sent(self):
    for i in range(5):
        b = str(i)
        self.send_command('4')

def upd(self):
    self.update_data()
```

Figure 46

This command is sent to the serial to understand that the GUI needs to receive readings.

3.2.2 Web Cam

A class called webcam used to show live cam and used to take a screenshot when the user wants. It is a sub-window from the main window once live camera feed button is clicked a Qdialogue is executed.

```
self.ui.liveCamButton.clicked.connect(self.webCamDialogue)
```

```
def webCamDialogue(self):
    self.w=WebCam()
    self.w.show()
```

Figure 47

Two push buttons are added: the first is to show the live camera feed and the second is the screenshot button. When the show button is clicked a function called 'startcamera' is executed, when the screenshot button is clicked a function called 'take screenshot' is executed.

```
# Initialize video capture and timer
self.cap = cv2.VideoCapture(0)
self.timer = QTimer()
self.timer.timeout.connect(self.updateframe)
```

Figure 48

Using OpenCV VideoCapture object is used to show the default camera of the laptop this is indicated by the number 0. self.timer.timeout refers to the signal emitted by the QTimer object when the time runs out.

```
def startCamera(self):
    if not self.timer.isActive():
        self.timer.start(30)
```

Figure 49

This function updates the frame every 30 milli seconds as when 30 milli seconds pass it connects to updateframe function.

```
def updateframe(self):
    rval, frame = self.cap.read()
    if rval:
        rgb_image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        h, w, ch = rgb_image.shape
        qimg = QImage(rgb_image.data, w, h, ch * w, QImage.Format.Format_RGB888)
        pixmap = QPixmap.fromImage(qimg)
        self.videoLabel.setPixmap(pixmap)
```

Figure 50

First of all, it captures a frame which gives two outputs: first one is a boolean True or False and the second is the captured frame in numpy array form. If it is True, frame captured is converted to be RGB. Then dimensions of the frame are the ouputs of .shape which are height, width and color channel. Then the Qimage is used to create an image from the raw RGB data. QPixmap is a PyQt class that displays images in a widget so the variable pixmap converts an image to a pixmap. Lastly, it updates the Qlabel with the pixmap.

```
1 usage
def takeScreenshot(self):
    if self.cap.isOpened():
        rval, frame = self.cap.read()
        if rval:
            cv2.imwrite( filename: 'screenshot.png', frame)
```

Figure 51

Simply, when the videocapture is opened a frame is captured once screenshot button is clicked in and saved.

3.2.3 Video stitch

When video stitch button is pressed it sends the user to another window.

```
from CameraSystenGUI import MainWindow as CameraSystemMainWindow
```

Figure 52

Importing a class from another file into the backend file.

```
def video_stitching(self):
    self.w = CameraSystemMainWindow()
    self.w.show()
```

Figure 53

Showing an execution of this class.

3.2.4 Stereo vision

When stereo vision button is pressed it sends the user to another window.

```
from MainStereoVisionWindow import MyWidget
```

```
def stereo_vision(self):
    self.w = MyWidget()
    self.w.show()
```

Figure 54

3.3 Software tasks

3.3.1 Flowchart

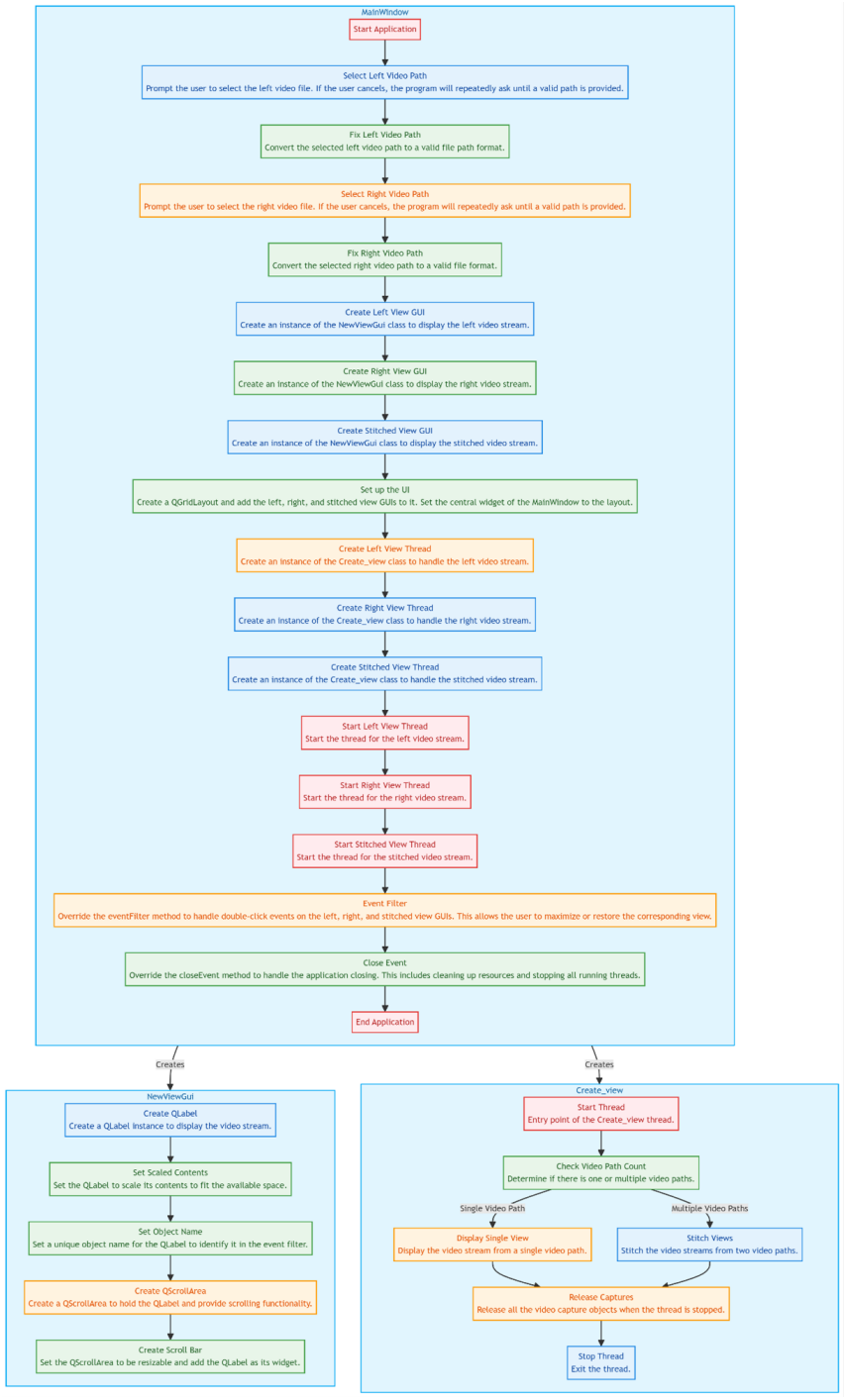


Figure 55: Video Stitching Flowchart

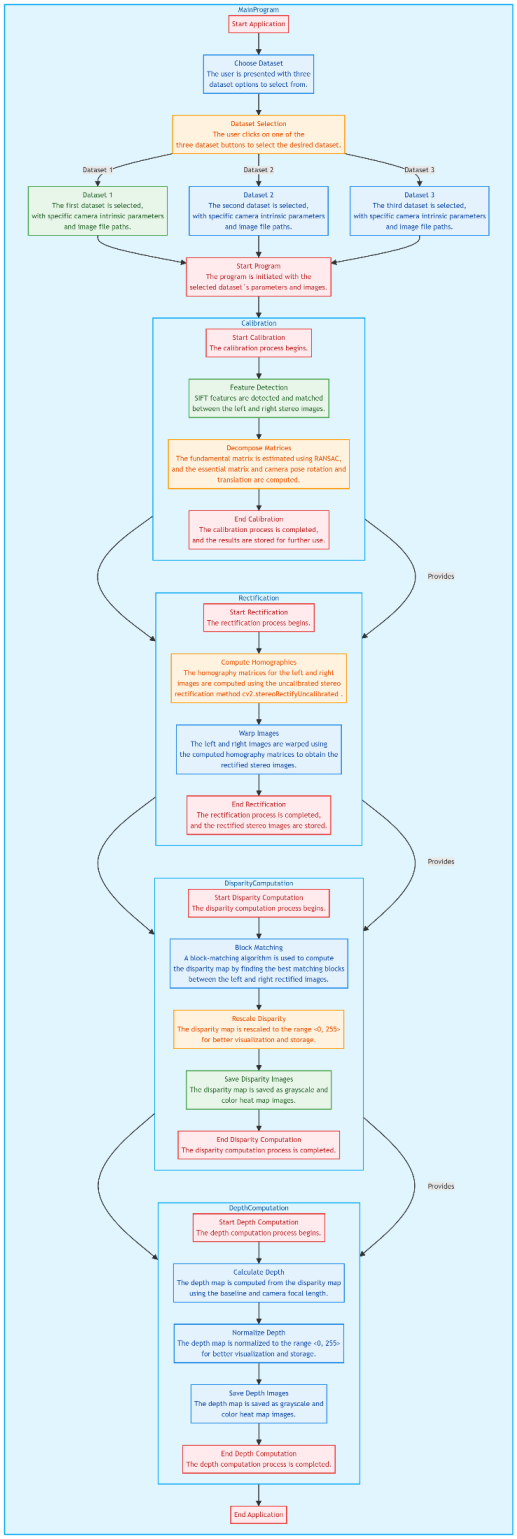


Figure 56: Stereo Vision flowchart

3.4 Video Stitching

3.4.1 The initialization

When the video stitching is initialized from the main window automatically a message box is opened to tell the user to open the video view on the left This step can be replaced with opening the right camera or start reading its

content Next another message box is opened for the right video view that also can later be replaced with a right camera

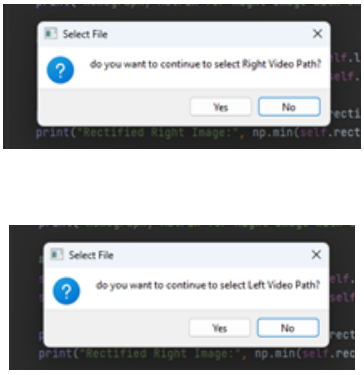


Figure 57

3.4.2 The Video Stitching window

A video stitching window is then displayed containing the left view video and the right view video and the stitched view.

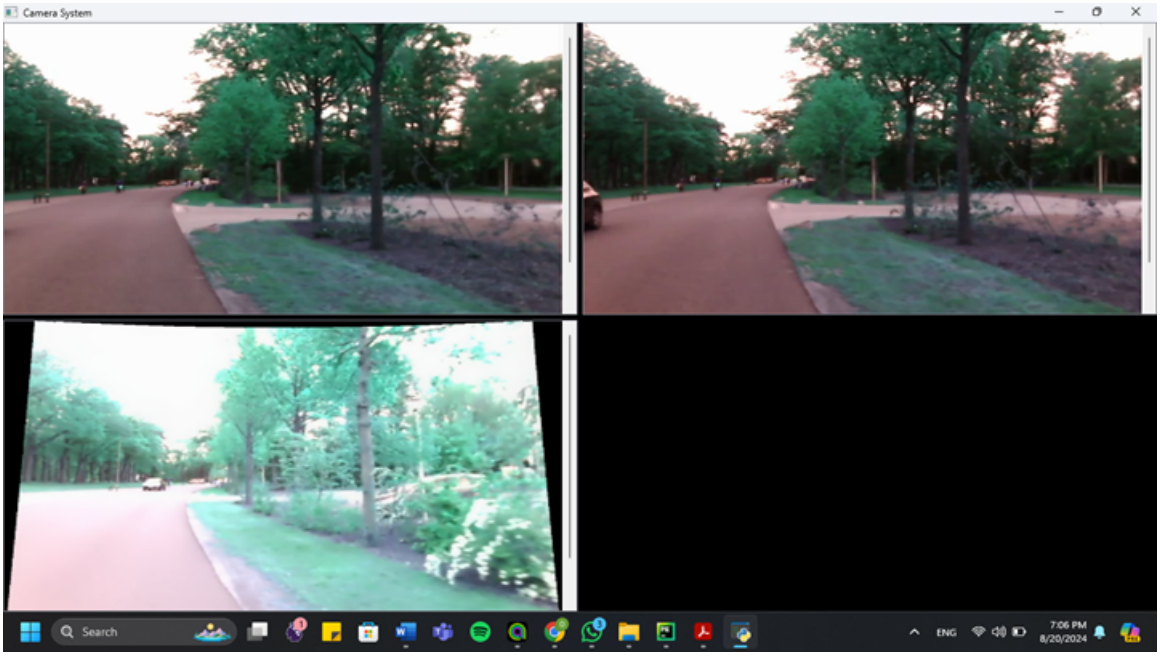


Figure 58

There is a scroll bar for each window to show the rest of the view that is omitted due to the small window size There is an option for maximizing each view window to fit all the video stitching window for closer inspection by double clicking left mouse button By double clicking again the maximized view is returned into normal view and all the other view windows are displayed once more

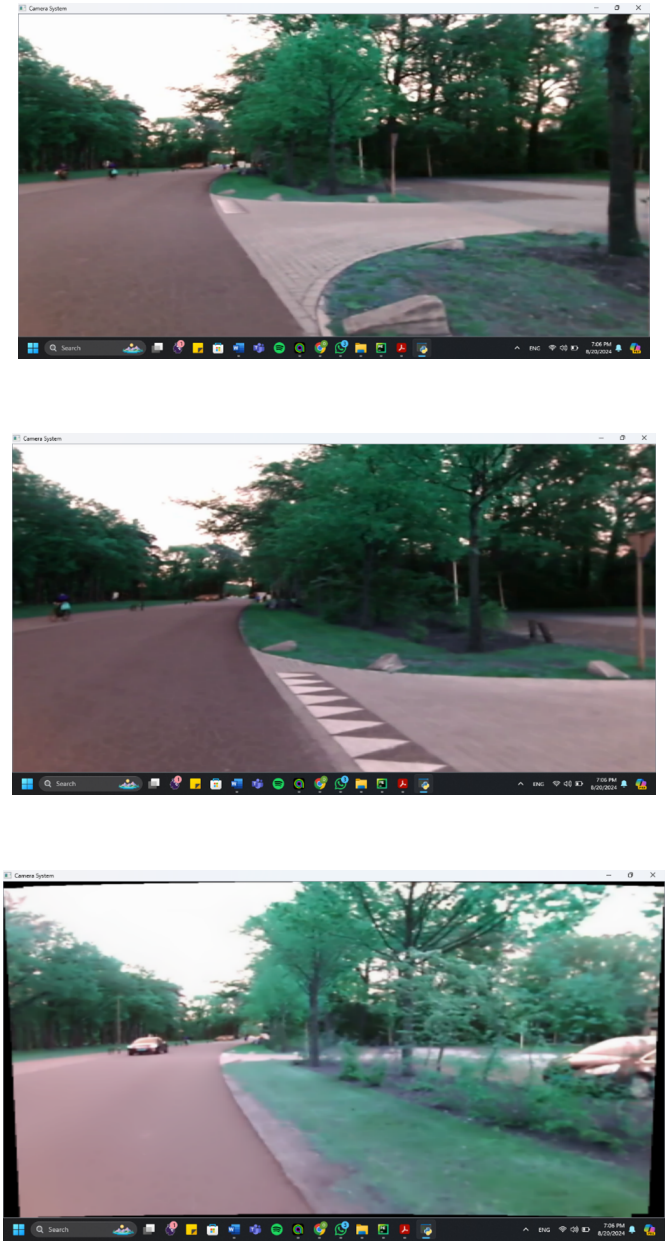


Figure 59

The MainCameraSystemWindow (videoStitiching window) consists of 2*2 subset windows to display all the views and for the last view ([1][1]) is an empty view as 3*1 or 1*3 windows were not a well displayed windows or interface to the user.

3.4.3 Frontend Algorithm

Program is started with file explorer window implemented in the main window (MainCameraSystemGUI) in an independent function (SelectVideopath) to be called twice for the left and the right views. If the user hit cancel in the message box appeared Infront, the program won't shut displaying the same message box once more until the user choose a sufficient video path. The program is only stopped by stopping terminating the whole GUI program.

In the file explorer there are 2 options displayed for the path showed to the user

- Video path: which is the sufficient path for a video (.mp4,.mov,.avi) for user restriction
- All files: for showing all files in the folder if the user wants

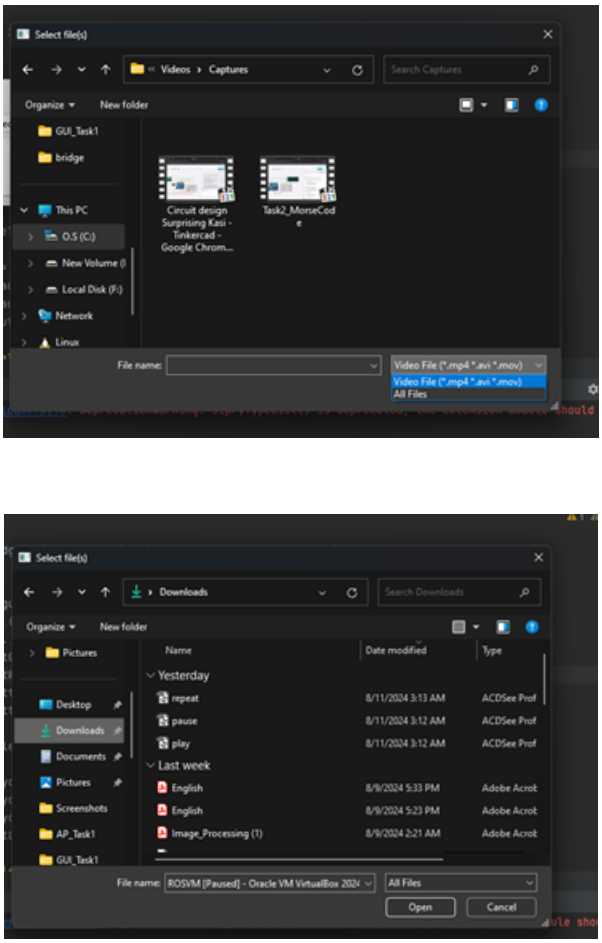


Figure 60

The path then is passed into a fixing function as the path read from the file explorer can't be directly sent and opened by OpenCV as it doesn't identify this path. The path is fixed by implementing 2 methods

1. The path is passed into a fixing function using pathlib to remove any excess string attached to the path
2. The new Path is passed into another function to remove any backslashes into forward slashes as this is how OpenCv reads the passes

The pass is now fixed and we can start opening each view captures.

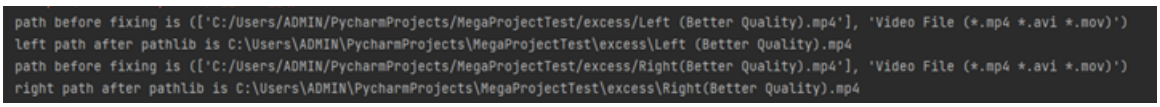


Figure 61

For each path a QThread is opened to implement Multiple Threading.

1. The frontend : for the new view window, a new instance is created is created from New_View class to fill all the attributes needed that are constant for each view, so a class was created for avoiding repetition

2. The backend : For the 3 views opened, each view thread creates an instance of class called createViewThread Which implements the function needed for the displaying if it the video view or to implement stitching method then display this view in the New_View Window

After finishing the backend Algorithm after that, the CameraSystemWindow is displayed This window has 3 views for the 3 New_View instances that was created For each New_Window , there was an attribute for its name , the object name , the scroll bar of the window view and finally the Window state, which is a variable from (Camera_State) enum class which contains 2 constants (Normal , Maximized) , and by default it is started with 'Normal' for the Window navigation later. At creating the new ViewThread, The Displaying and Stitching backend algorithm is implemented

3.4.4 Backend Algorithm

At first the view information is saved in the initialization. Then, the thread is started from the frontend and run method is started. 'Run' method identifies if the path array pathed to the class has 1 or more paths, if it was 1 path, then it's a normal view display, else, that means that there are more than 1 view to be stitched together and displayed after the implementing the stitching algorithm. If the path array was only 1 path, (display_view_stream) function is called to open the capture and display all video frames till the video ends. If the path array was more than 1, (stitch_view_stream) is called, the function starts with opening both videos capture and getting frame by frame from the captures then calling (stitch_frames) function to stitch both frames into one frame then display it Stitching function Algorithm is implemented using Stitch built-in library for simplicity, by calling Stitcher.create() and creating both frames into one larger frame. Unfortunately the library doesn't handle all stitching frames correctly , also It takes time for stitching , slower than the supposed smooth motion of the video capture, making the stitched view window lagging than the other windows , even after implementing a

few method to speed up the stitching process like : decreasing video quality , dropping a frame after each 2 stitching frames formed , also estimating a special wait time of the wait key by calculating the synchronization time of both video frames and their data.

3.4.5 Close Algorithm

After finished all the display and if the user wants to quit this video, the x button is hit. After hitting the x button, a message button is showed for ensuring if the user want to close the window. If yes was pushed, the program calls function 'close_event' which ensures closing all threads and releasing all captures for clean finish and memory optimization

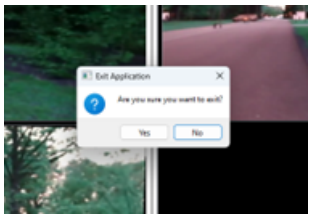


Figure 62

3.4.6 References

i. https://youtu.be/-9MXhM_HmxE?si=5HcXsg3WB7rlzT7p
ii. https://youtu.be/hGhqPpVZR4A?si=N7rBjEJAEkyhQKYa
iii. https://docs.opencv.org/2.4/modules/stitching/doc/stitching.html
iv. https://stackoverflow.com/questions/72022176/warning-cant-open-read-file-check-file-path-integrity
v. https://www.geeksforgeeks.org/python-play-a-video-using-opencv/

3.5 Stereo vision

3.5.1 Initialization

After choosing Stereo vision from the main window a dialogue box is opened for the user to choose Which data set to continue with between dataset 1,2,3. Each set data is initialized in the program, at the user choosing aa data set to test the program with, the data is sent to the function that starts the program. The function calls calibration process at first, after implementing the calibration algorithm data, the rectification process is called with the data needed and with a calibration instance to take the variables output from

the calibration process needed in rectification, same after that for disparity process and at last the depth computation process.

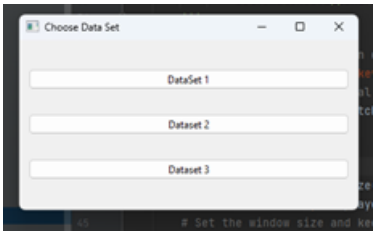


Figure 63

3.5.2 Main window algorithim

A StereoVisionProcess abstract class (ABC) is constructed in which all the 4 processes inherit from it. Each class have the function create which is abstract in the superclass. aster the user chooses which data set to continue with, an instance of Calibration class is created with the parameters it needed then create function is called to start the process. The same is implemented for each process Create function in each process class is implemented to start the class implementation and calling all its functions in a correct and fixed order

3.5.3 Utility Class (Detecting and drawing epipolar lines) Algorithm

A Utility class is also created that has one static function(draw_epipolar_lines) to be called from any class needs to draw the epipolar_lines instead of writing the same function more than 1 time (Calibration process and Rectification process) At first detecting and drawing the epipolar lines is called first from the utility class. Then a function to decompose both fundamental and essential matrices is called and the matrices are then printed and displayed in the console The Epipolar lines in the utility class is first detected using BF-matcher and the matches are saved in a list Then function 'drawMatches' is called by giving it the matched list, the image and the image_data. Another image then is outputted with the epipolar lines between all the matches is displayed.

3.5.4 Calibration Algorithm

After calling the 'detect_and_draw epipolar_lines' from utility class and returning the keypoints of each image and the matched points, Decompose Matrix function is called passing the keypoints and the matched points to it to extract the points from the matched list and decompose the fundamental

matrix using built-in function (FindFundamentalMat) in OpenCv. Then the essential matrix using dot product of the keypoints and the fundamental matrix. lastly, it extracts the inliers and call 'recoverPose' function to calculate the rotation and the translation vectors. At last, the algorithm returns to the main class to implement rectification process next

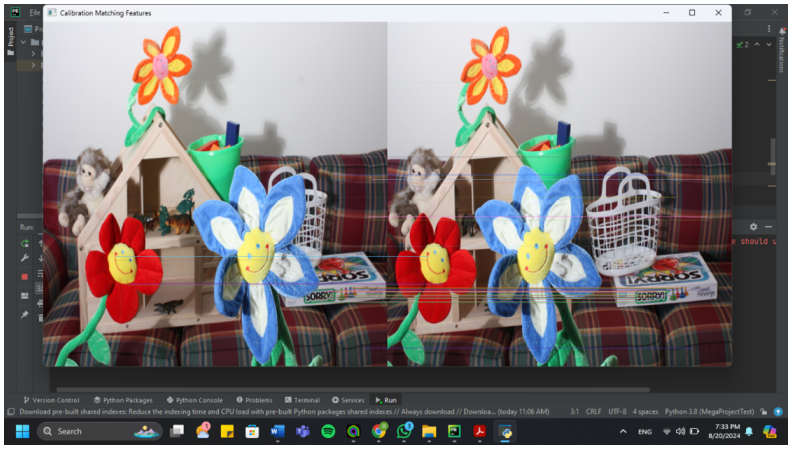


Figure 64

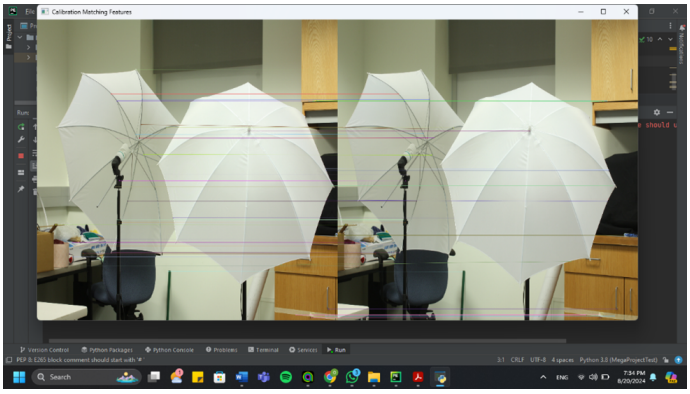


Figure 65

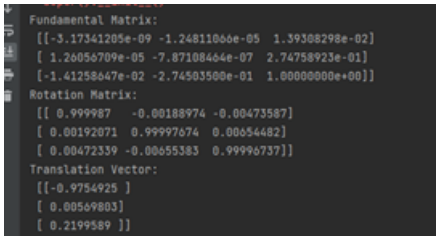


Figure 66

3.5.5 Rectification Algorithm

The rectification process is used to remeasure the matches extracted from the previous data, as some of these matches can be incorrect due to connecting only 2 points of the inliers for a matching feature, in which this match may

be incorrect and the 2 points only happen to be in the same epipolar line but they are not matching. Therefore, the rectification process is implemented to decompose both left and right images but rectified for better matches The algorithm used 'StereoUncalibrated' function as 'StereoRectify' function which is supposed to utilize the previous estimated data don't work and give wrong outputs of whole black or white images. The Uncalibrated function takes the inliers and the fundamental matrix resulted from the previous step (Calibration). The function output is a set of homogonous matrices that are printed in the console. They are used to estimate the rectified images after that.

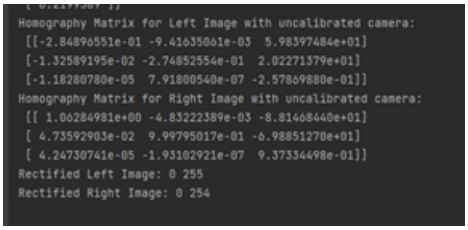


Figure 67

At last, 'detect_and_draw_epipolar_lines' is then called once more, from the utility class, to detect the matched points and draw them and display the outputted image. Finally, the rectification process returns to the main window and proceeds to disparity computations.

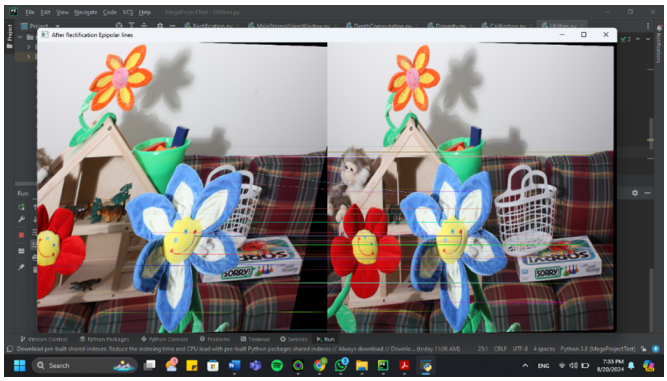


Figure 68

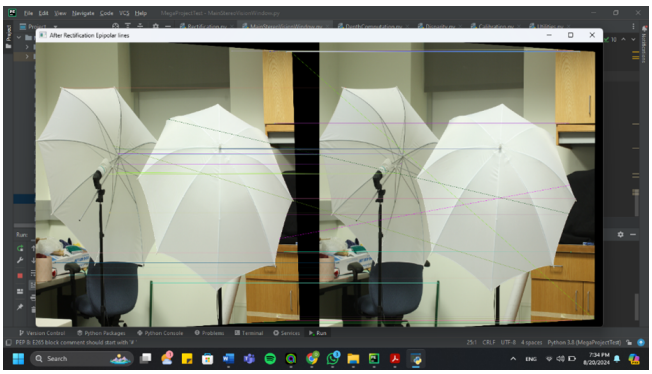


Figure 69

3.5.6 Correspondence (Disparity) Algorithm

this class is the 'block_matching' function, which compares small blocks (windows) from the left and right images to find the best matching blocks. The difference in the horizontal position of these matching blocks is recorded as disparity. The disparity map is then rescaled to a grayscale image for visualization, and optionally, a color heatmap is generated to provide a more detailed view of the disparities. Finally, the computed disparity maps are saved as image files, completing the process before the depth computation can begin.

3.5.7 Depth Computation Algorithm

The 'depthComputation' class is responsible for converting the disparity map into a depth map, which provides the distance of objects from the camera in the scene. The key method in this class is 'calculate_and_display_depth', which calculates the depth by using the camera's focal length and the baseline distance between the cameras. The depth is computed as the ratio of the baseline times the focal length to the disparity values. The resulting depth map is then normalized and saved as both a grayscale image and a color heat map for visualization. This class concludes the stereo vision process by providing a detailed depth map, essential for understanding the 3D structure of the scene and estimating the object depth from the installed camera.

3.5.8 References

i. https://youtu.be/S-UHiFsn-GI?si=D94UTnRJScMqW_dG
ii. https://youtu.be/EkYXjmiolBg?si=QPG-egI548ceOa0D
iii. https://youtu.be/KOSS24P3_fY?si=915wx5GngW2M4lRD
iv. https://stackoverflow.com/questions/36172913/opencv-depth-map-from-uncalibrated-stereo-system

v. https://www.educba.com/opencv-get-image-size/