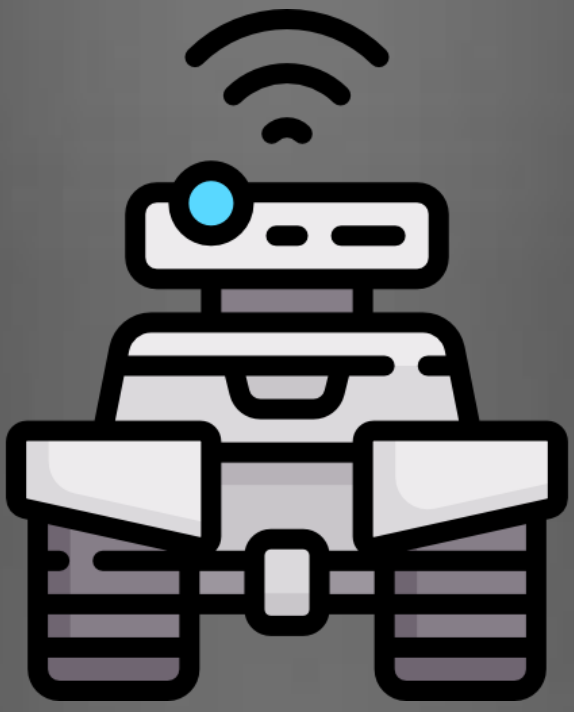
Mega Project Report



Threewheelerz

By

Mohamed khamis

Habiba Shaker

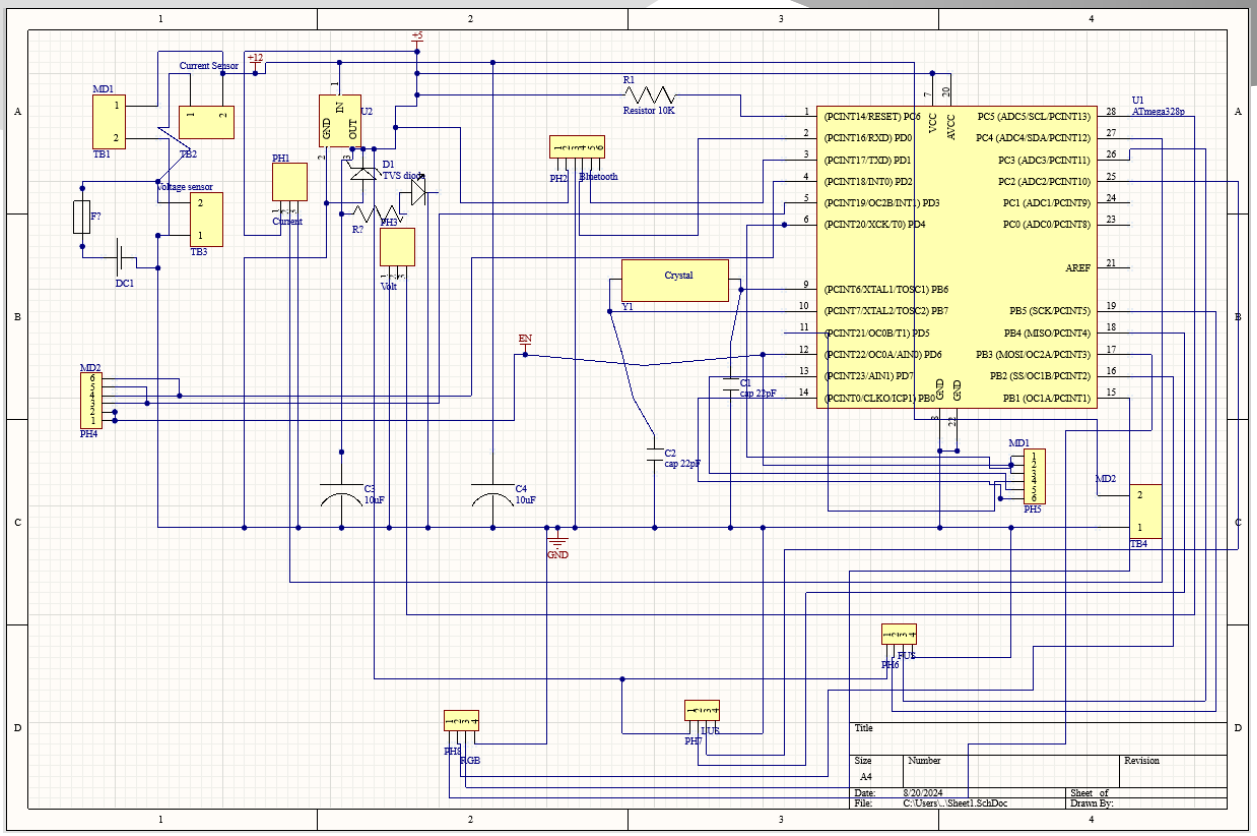
Amira Mohamed Taha

Electrical System:

PCB:

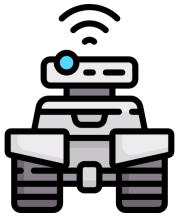
For this car to work a pcb was needed to connect the microcontroller with motors drivers connections, and sensor reading also convert supply to 5v for the microcontroller and maintain protection.

PCB schematic:



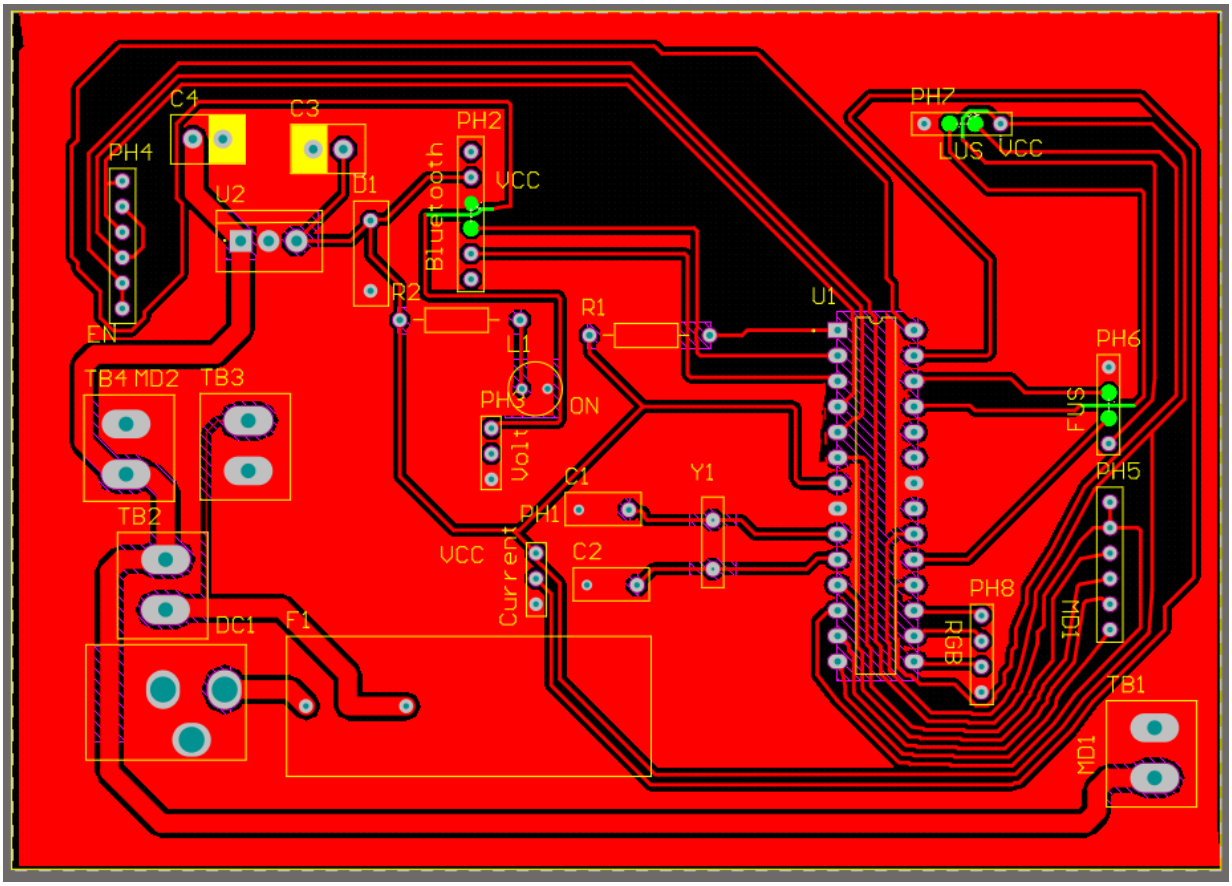
Schematic contains:

1. Microcontroller Atmega328p
2. Crystal 16MHZ
3. Capacitor 22pF
4. Pin headers for connecting:
    - Motor divers logic input
    - Front and side ultrasonic
    - RGB
    - Bluetooth module
    - Voltage sensor
    - Current sensor

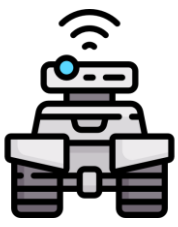


5. Terminal blocks for connecting:
   - Supply for motor drivers
   - Input for current sensor
   - Input for voltage sensor
6. Linear voltage regulator
7. Fuse
8. TVS diode
9. DC jack

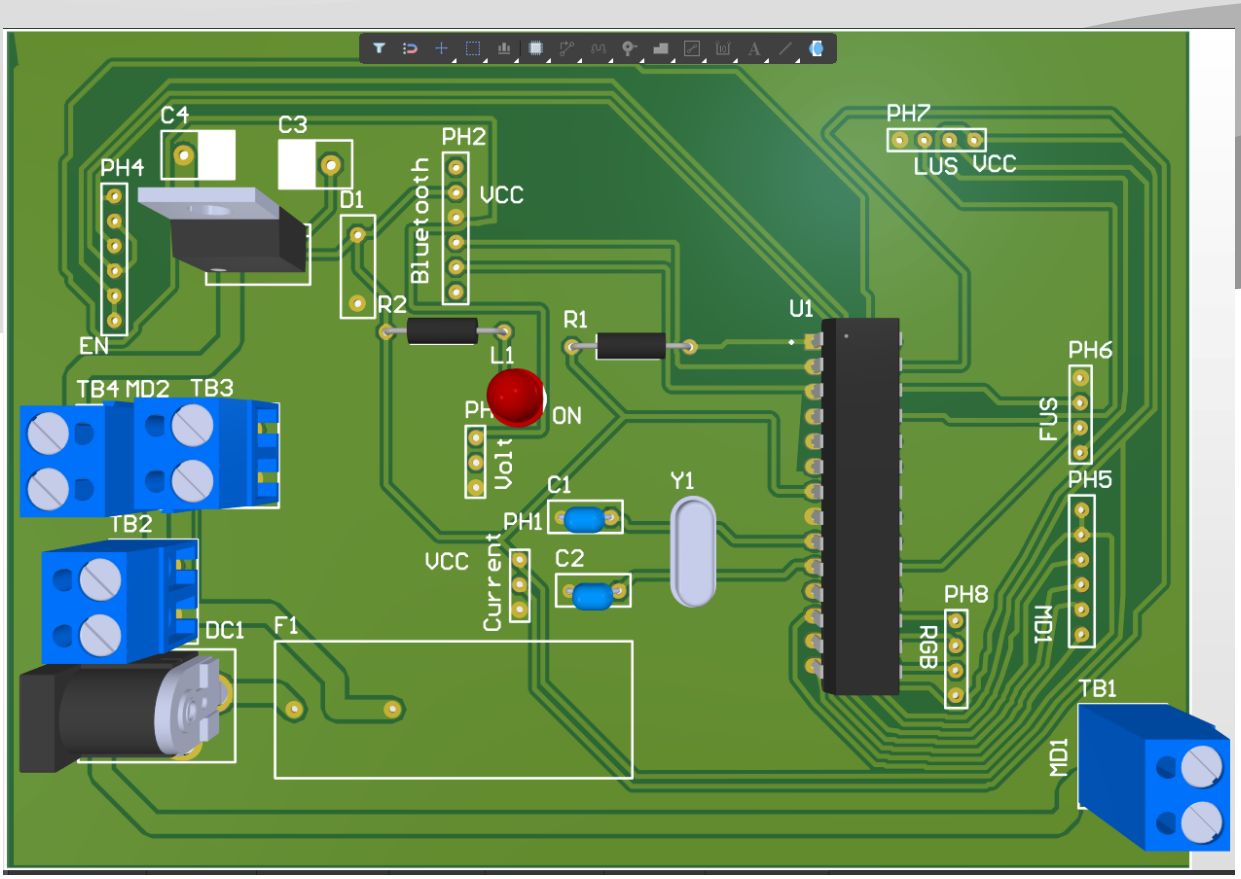
Components need to be arranged in a way each connector is near its component and be able to rout on single layer and appropriate clearance



- This is the design we came up to motor driver supply connector an pin headers is in edge of pcd, voltage and current sensor input and reading connector are near each other and rgb facing back
- Routing thickness is 0.5 mm except for motor driver supply is 2mm to withstand hight current
- Clearence was maintained 0.3 mm

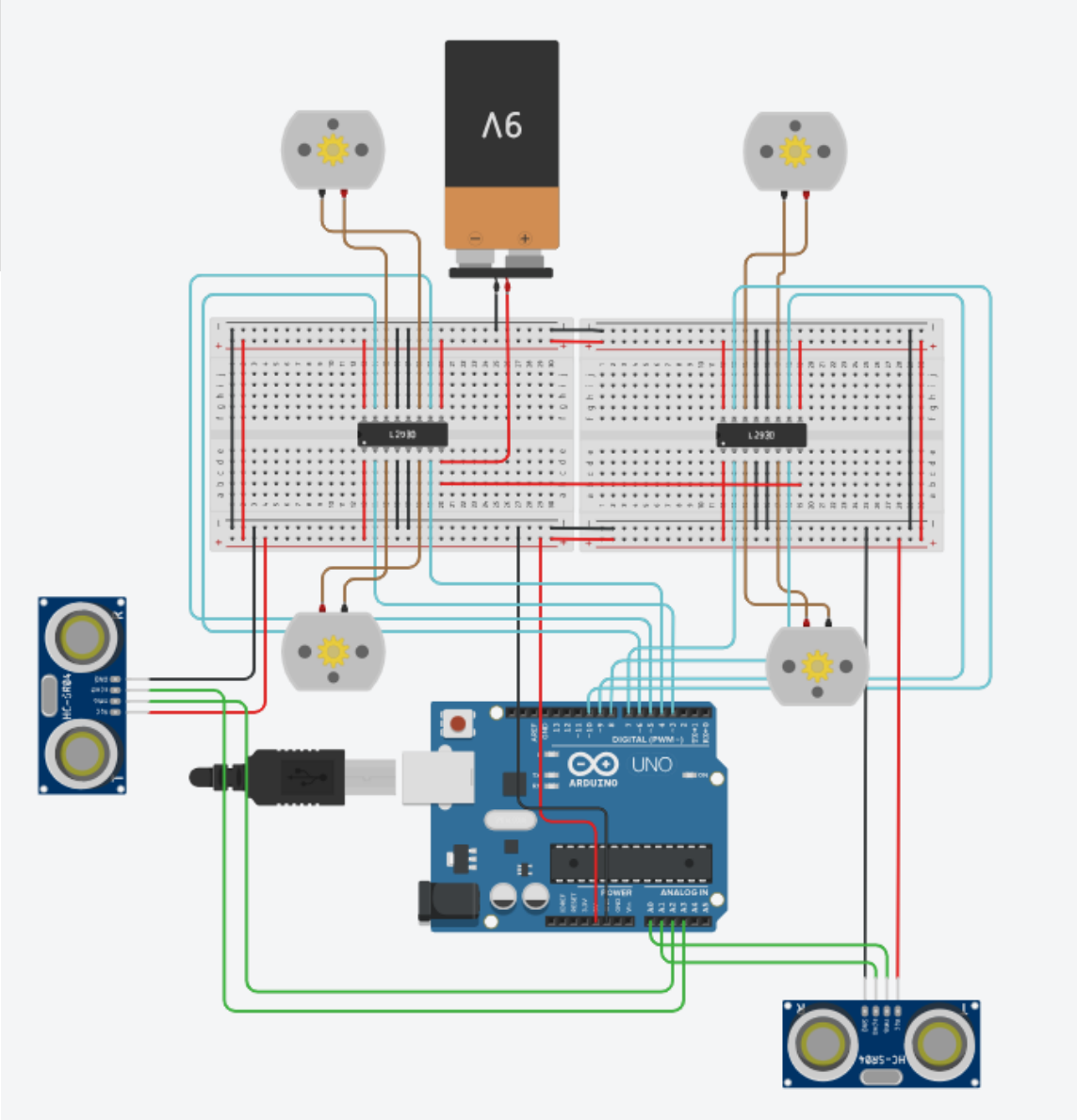


This is the final design in 3D:



Firmware:

Circuit Schematic on Tinkercad:



Here is the link for the simulation:

https://www.tinkercad.com/things/hb7D3SsHG8o-stunning-esboo/editel?sharecode=gVFfFbH5X_ApJPINC1msti89HeGr8eeFjMuMNg-cg8s

The car control final code explanation:

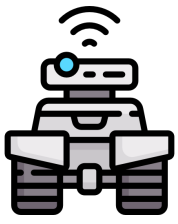
- The motor control pins are defined for the front-left (FL), front-right (FR) and back motors (B) controlled together with the same pins.
- Define the speed pin and initialize the speed modes.
- Two ultrasonic sensors are used, side sensor is connected to the defined pins “TRIG_SIDE” and “ECHO_SIDE” and front sensor is connected to pins “TRIG_FRONT” and “ECHO_FRONT”.
- Set the motors pins mode as outputs and Set the trigger pins as output and echo pins as input using “ pinMode()”
- Initialize the PID variables, the mode by -1 that indicates no mode selected yet, the motor speed and the front distance to detect obstacle.
- Define the voltage and the current pins and set them as inputs and initialize their values by zero.
- Define the RGB LED pins and set them as outputs.
- Begin communication using Serial.begin(9600), that allows the arduino to interact with the python code.
- Main Loop:
    1. Check the Received Commands: If a command is received from the GUI, arduino read it based on the following:
        - Manual mode commands:
            - 'F' (Forward): Moves forward.
            - 'B' (Backward): Moves backward.
            - 'R' (Right): Rotate right.
            - 'L' (Left): Rotate left.
            - 'S' (Stop): Stop all motors.
        - Mode selection commands:
            - 'M': Set the mode to manual (mode = 0).
            - 'A': Set the mode to autonomous (mode = 1).
        - Speed control commands:
            - 'l' (low): Set the speed to low (turn on green LED).
            - 'm' (medium): Set the speed to medium (turn on blue LED).
            - 'h' (high): Set the speed to high (turn on red LED).
        - Set the distance from the wall command:
            - 's': Read the distance (setpoint) from the GUI and update the desired distance for the side sensor in autonomous mode.

    2. Sending sensor data: Every 1.5 seconds, send the following data over serial communication that appears in the frontend GUI:

        - Distance measured by the side ultrasonic sensor.
        - Current value measured by the current sensor.
        - Voltage value measured by the voltage sensor.

- Manual mode function (manualControl()): According to the received command, one of the following manual control functions takes place:

    - Moves forward ('F'): Moves the car forward.
    - Moves backward ('B'): Moves the car backward.
    - Rotate right ('R'): Rotate the car to the right.
    - Rotate left ('L'): Rotate the car to the left.
    - Stop ('S'): Stop all motors.



- Autonomous mode function (autonomousControl()):
    - Get the distance from the wall using the side ultrasonic sensor.
    - Apply PID control equation to adjust the car's position and maintain the desired distance from the wall.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\,d\tau + K_d \frac{de(t)}{dt}$$

    - Control the motors based on the PID output.

- The setpoint function (assignSetPoint()):
    - Read the desired distance value from the GUI, convert it to a float, and update the setpoint to the desired distance from the wall.

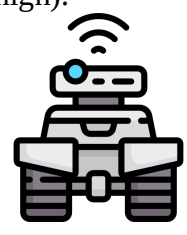
- Motor control function based on PID output (controlMotors(float PIDOutput)):
    - Calculate the motor speed by the absolute value of the PID output.
    - Constrain the motor speed in a range of (0 to motorSpeed).
    - If the output is positive, rotate right; if negative, rotate left and then moves forward.

- Ultrasonic distance measurement function (getUltrasonicDistance(int trigPin, int echoPin)):
    - Measure the time taken for the pulse to travel to the wall and back.
    - Convert the time into distance in cm and return its value.

- Obstacle Detection function (obstacleDetected()):
    - Measure the distance using the front ultrasonic sensor, if the distance is less than the defined obstacle distance, return true to indicate an obstacle.

- Voltage reading function (voltRead()):
    - Read the voltage from the voltage sensor, convert the analog reading to a voltage value using voltage divider rule and return its value.

- Current reading (current()):
    - Read the current value from the current sensor, convert the voltage reading to a corresponding current value using the sensitivity factor and return the value.
    -
- Speed and RGB LED control function (setSpeed(char speed)):
    - Change the motor speed based on the selected level of the speed (low, medium, high).
    - Control the RGB LEDs based on the chosen speed (Green for low speed, Blue for medium speed, Red for high speed)



Software Tasks' Algorithms:

Camera feed implementation:

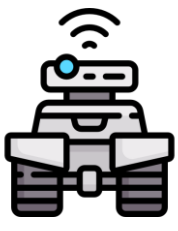
To implement the live camera feed feature, I began by researching and reviewing various online resources and documentation to deepen my understanding of how live camera feeds work, particularly within the context of Python and PyQt6. This initial research phase allowed me to conceive the implementation strategy, which I then captured in a short pseudocode. The pseudocode helped me brainstorm the necessary steps, including how to initialize the camera feed, interface with the laptop's webcam, and efficiently display the feed in the GUI using PyQt6.

Initialization:

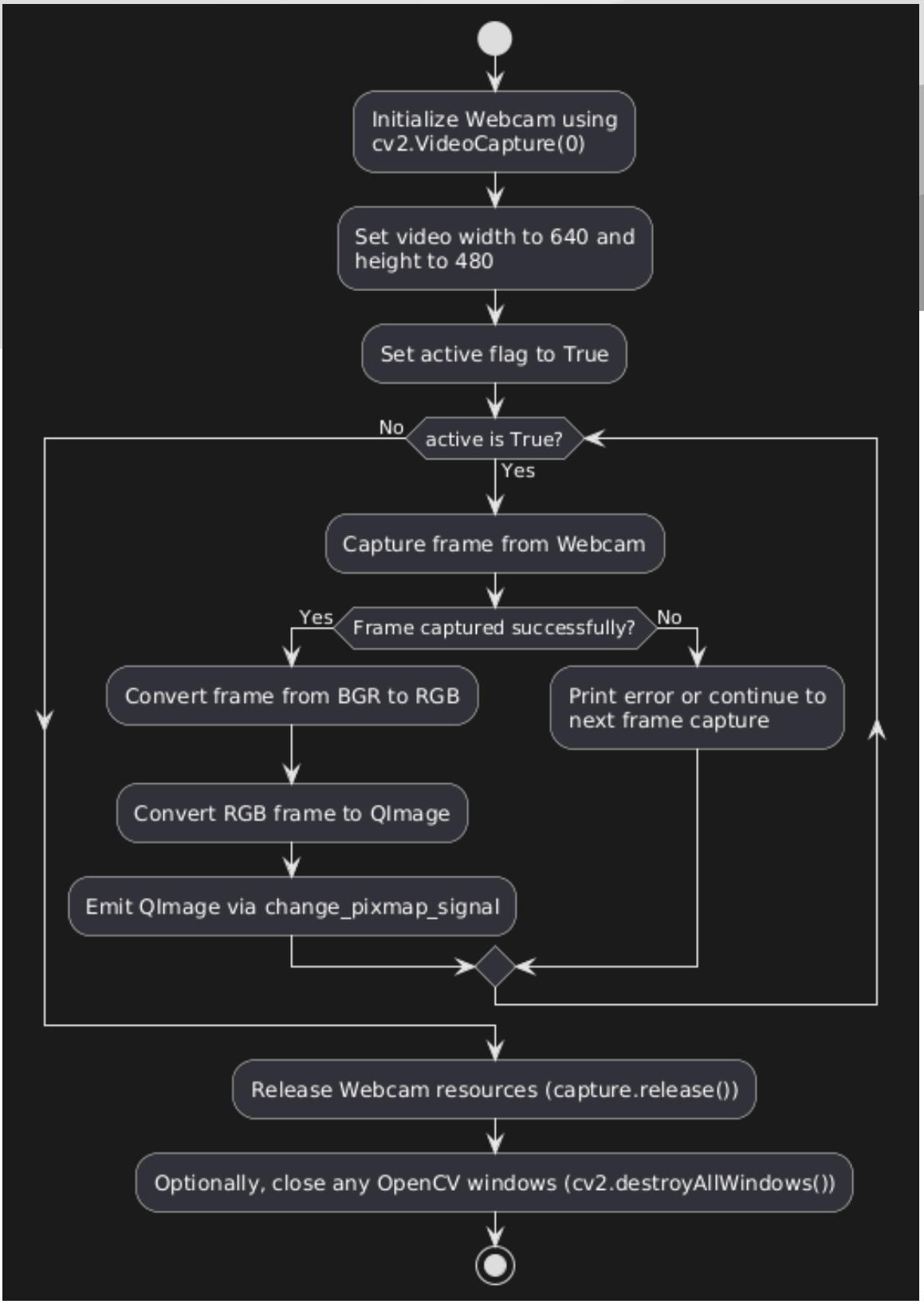
- I accessed the webcam of laptop using OpenCV(cv2.videocapture(0)). This function enables the default build in camera(WebCam).
- I set up the video frame ('width' & 'height') to fixed dimensions 640x480 pixels

Frame capture:

- Continuously capture frames from the webcam once webcam is successfully initialized and active
- Convert each captured frame from BGR (default format used by OpenCV) to RGB format, which is compatible with PyQt6.
- Convert the RGB frame into a QImage format suitable for display in a PyQt6 widget.
- Emit the QImage using a signal (change_pixmap_signal) to update the GUI.



Flow chart:


Initialize Webcam using cv2.VideoCapture(0)
Set video width to 640 and height to 480
Set active flag to True
No
active is True?
Yes
Capture frame from Webcam
Yes
Frame captured successfully?
No
Convert frame from BGR to RGB
Print error or continue to next frame capture
Convert RGB frame to QImage
Emit QImage via change_pixmap_signal
Release Webcam resources (capture.release())
Optionally, close any OpenCV windows (cv2.destroyAllWindows())

Screenshot button:

Check Webcam Status:

- Ensure the webcam is active and capturing frames.
- If the webcam is not active, print a message and exit the function.

Capture Frame:

- Capture a single frame from the active webcam stream by using self.capture.read()
- If the frame capture is successful, proceed to save the image.

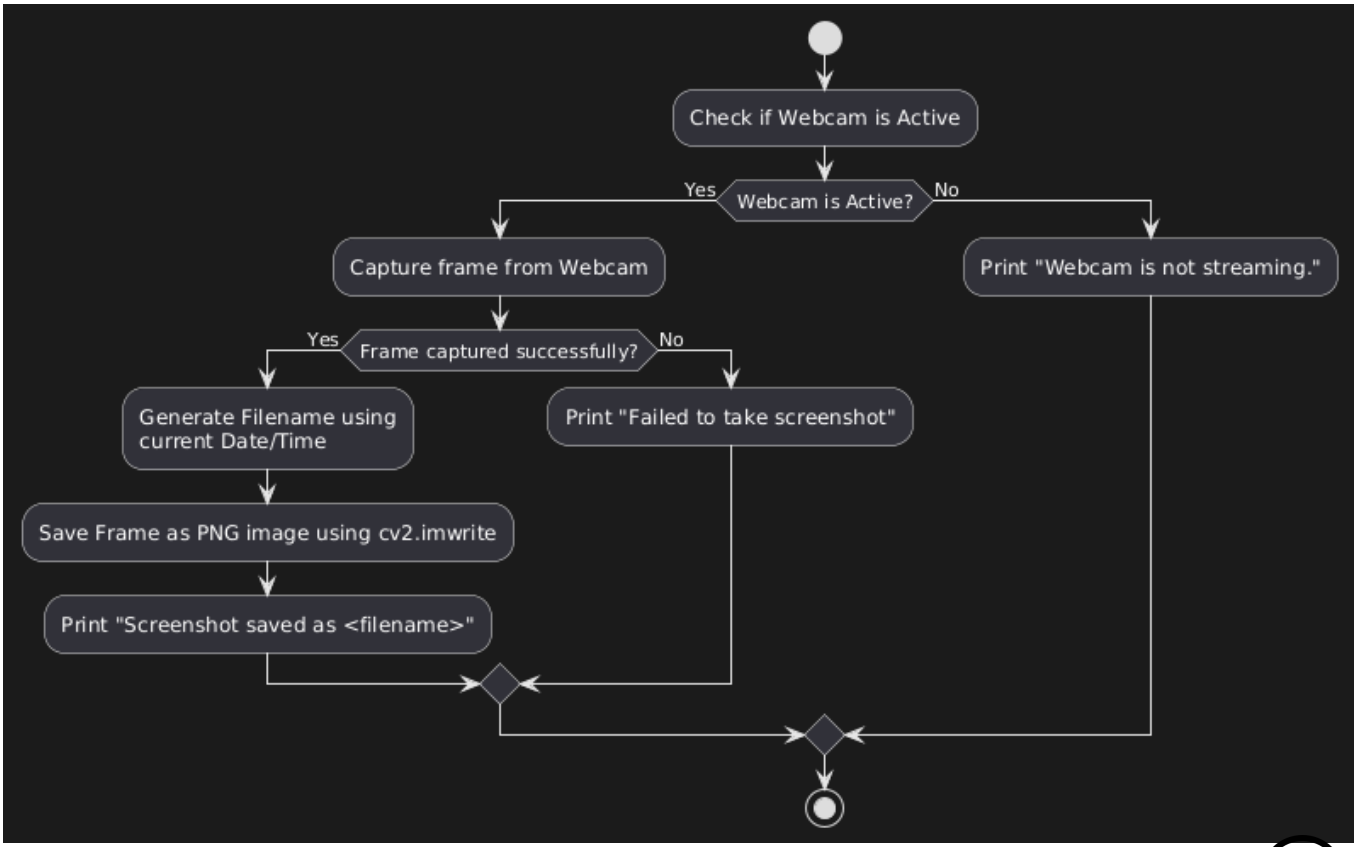
Save the Frame:

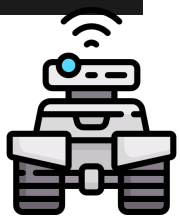
- Generate a filename for the screenshot based on the current date and time.
- Save the captured frame as a PNG image file using OpenCV's cv2.imwrite.

Notify User:

- Print a message indicating that the screenshot has been saved successfully

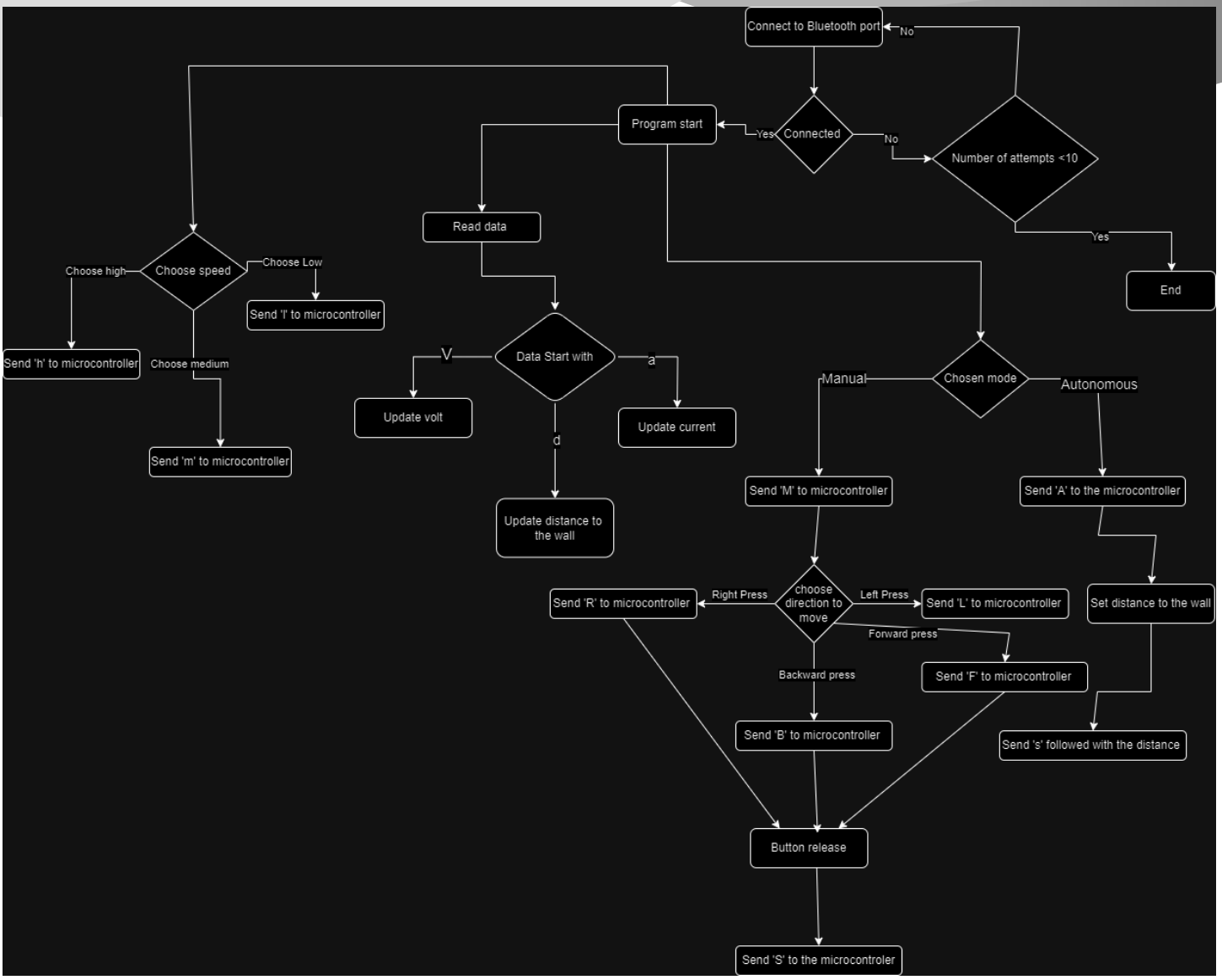
Flowchart:

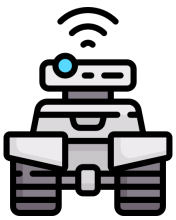




GUI Features:

Backend Flowchart:





Main Windows Features:

1. Choose Mode
   - Dropdown Menu: Allows the user to select the control mode for the car (manual or autonomous).
   - Set Mode Button: Confirms and sets the selected mode from the dropdown menu.

2. Manual Control
   - Up Arrow: Moves the car forward.
   - Down Arrow: Moves the car backward.
   - Left Arrow: Turns the car to the left.
   - Right Arrow: Turns the car to the right.

3. Camera Feed Display
   - Central Display Area: This is the area where the live camera feed from the car would be displayed.

4. Action Buttons
   - Screenshot Button: Captures the current frame from the camera feed and saves it as an image file.

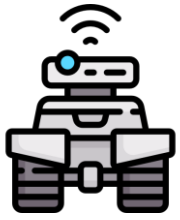
5. Speed Control
   - Dropdown Menu: Allows the user to choose the speed (Low , Medium & High) at which the car operates.
   - Set Speed Button: Confirms and sets the selected speed for the car.

6. Sensor Readings
   - Voltage Display: Shows the current voltage (e.g: 0.00 V).
   - Current Display: Shows the current drawn by the car (e.g: 0.12 A).
   - Distance to the Wall: Displays the distance from the car to an obstacle (e.g: 7.55 cm).

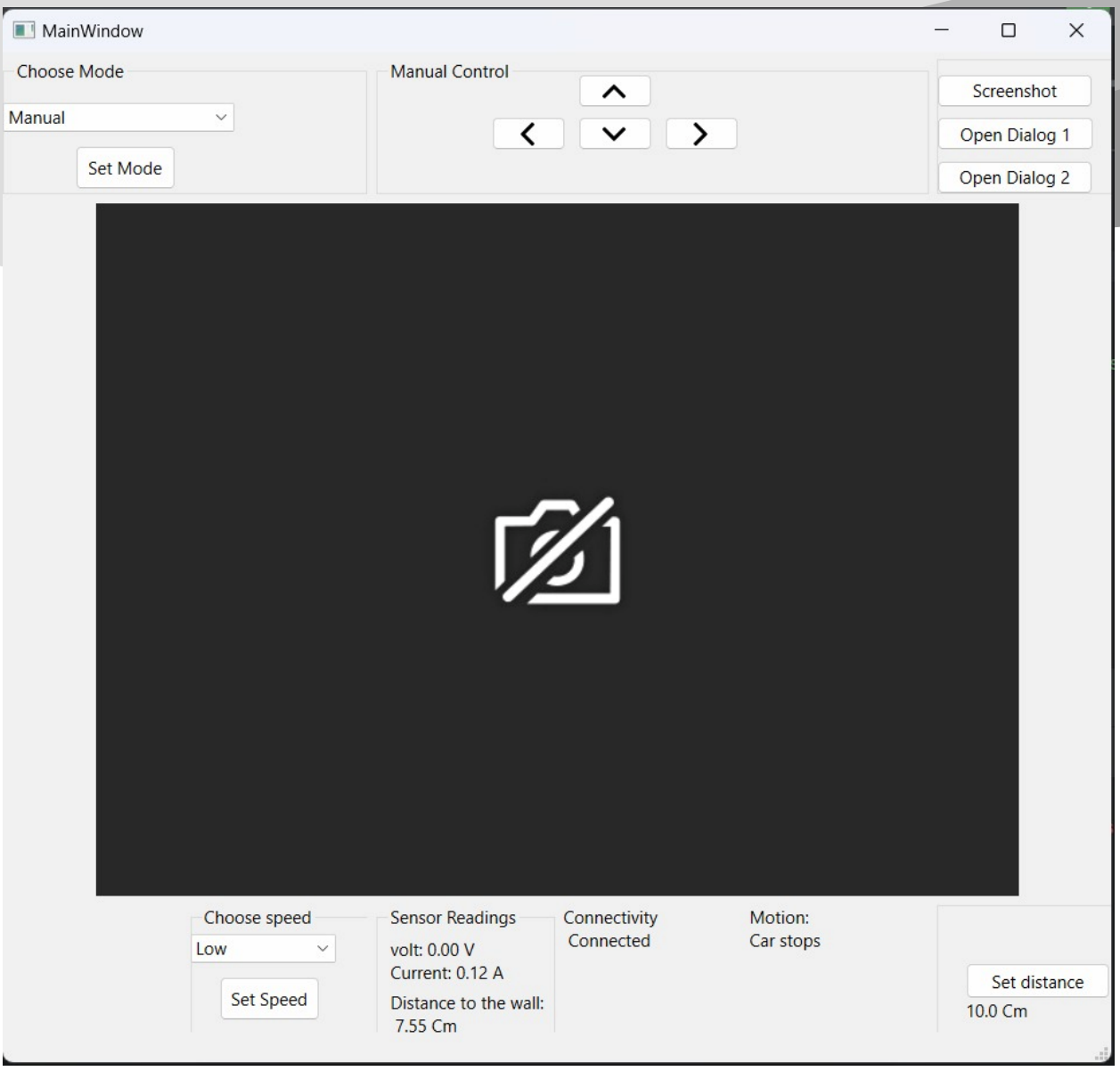
7. Connectivity Status
   - Connectivity Indicator: Shows whether the car is connected (e.g: "Connected").

8. Motion Status
   - Motion Indicator: Displays the current motion status of the car (e.g: "Car stops").

9. Distance Control Section
   - Set Distance Button: Allows the user to set a specific distance for the vehicle to maintain from an obstacle.
   - Distance Input Field: The field next to the button where the user can enter the desired distance (e.g., 10.0 cm).



Screenshot:


MainWindow
Choose Mode
Manual
Set Mode
Manual Control
Screenshot
Open Dialog 1
Open Dialog 2
Choose speed
Low
Set Speed
Sensor Readings
volt: 0.00 V
Current: 0.12 A
Distance to the wall:
7.55 Cm
Connectivity
Connected
Motion:
Car stops
Set distance
10.0 Cm