TECHNICAL REPORT FOR AQUAPHOTN’S MEGATRAINING PROJECT

TEAM 6

1-Introduction

This Mega Project is a simulation of the main activities needed to be ready for a Robot The main objective is to create a flexible and effective remotely operated vehicle that can be operated via a comprehensive GUI interface and switch between manual and autonomous modes The project brings together hardware design, firmware programming, and advanced software techniques, particularly in image processing.

1. Hardware and Firmware

In the hardware aspect, you will design and fabricate a PCB that forms the core of the ROV's control system. This PCB will house a bare microcontroller circuit, motor drivers for controlling two DC motors, and feedback sensors to monitor the car's voltage and current. The car will be powered by a battery, making it fully mobile.

In firmware, the car will operate in two modes:

• Manual Mode: This mode enables the user to move the car in any direction using a GUI and wireless control.
• Autonomous Mode: Using PID control to guarantee it travels parallel to the wall, the vehicle will keep a certain distance from it.

2. Graphical User Interface (GUI)

The GUI serves as the ROV's command center, giving users access to real-time video feeds, sensor readings, and vehicle control. With several windows for distinct tasks, the interface will be flexible, modular, and easy to use. The graphical user interface will integrate:

• Car Control: Move the vehicle in either manual or automated mode.

• Live Camera Feed: See and take pictures with the vehicle's cameras.

•Sensor Readings: Display real-time data from voltage, current, and ultrasonic sensors.

•Video Stitching and Stereo Vision: Advanced image processing tasks to merge video feeds and extract 3D information from 2D images.

3. Computer Vision Tasks

These tasks involve using image processing techniques to achieve specific goals:

- Video Stitching: Combine two video streams (e.g., from left and right cameras) into a single, larger video, useful for enhanced visual coverage.
- Stereo Vision: Extract 3D information from the environment, helping in navigation and object measurement underwater.

2-Disscusion

❖ Hardware

Overview of the fabricated PCB

The PCB contains a Protection and regulation for the input power using mosfet (IRF5305) and voltage regulator (LM7805), Capacitor for filtering noises, microcontroller (Atmega 328p), Motor drive (L293D), Voltage sensors using voltage divider, Current sensor using (ACS712), Headers for connecting the Bluetooth module and ultrasonic sensor and to program the Atmega using Arduino as ISP

Procedures

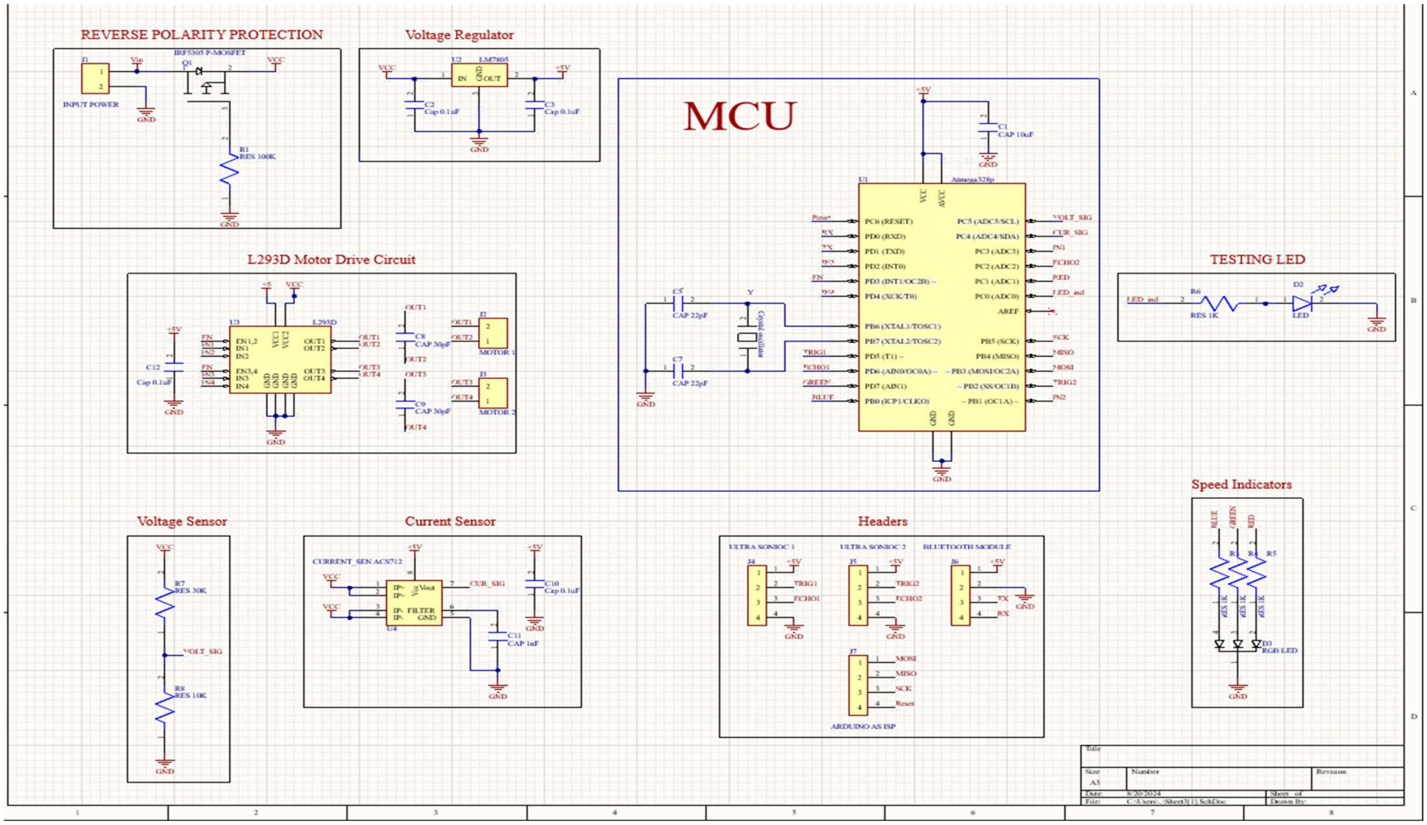
As we knew what exactly the circuit we needed to do and what is the purpose from this circuit we started some steps to make this PCB

1. Searched for all the components needed to make this circuit and understanding everything about it
2. After knowing what exactly the circuit we need to do and which components we will need We opened **Makers website** to make sure that all components exist and if not look for another option for the un available component
3. Opened Altium designer started doing the Symbols and foot prints for the Components

   (ALL the footprints were done using their datasheet provided in the makers website)
4. Started connecting all the components in the schematic Using Net Lablea or wires
5. we met and bought the components and printed the footprints and checked that all the footprints are correct
6. Started the PCB design by first setting the Design rules for the clearance, Track width, Polygon connection ….
7. Start Routing the PCB
8. Met and fabricated the PCB
9. Checked with Avo that there is no short circuit or any fault in the tracks
10. Connecting the power and test all the PCB

Now let’s go in detailed with Schematic connections, Layout Design & Routing

Schematic connection



Firstly, in this circuit we made our own modules for everything nothing is premade module.

1. **Reverse Polarity Circuit**: As a protection for the circuit form any human mistake while connecting the terminals of the battery this circuit was made using

   P-channel MOSFET (IRF 5305) : This mosfet is chosen according to it’s

   - VDs(BR) : which is equal 55V so that is very good as the max voltage will be supplied is 12V put we choose this mosfet for a more safety margin
   - VGs(TH) : which is equal to -2V Min and -4V max so we connected the gate to **100Kohm resistor to make sure that VG=0** and the Vs =12V

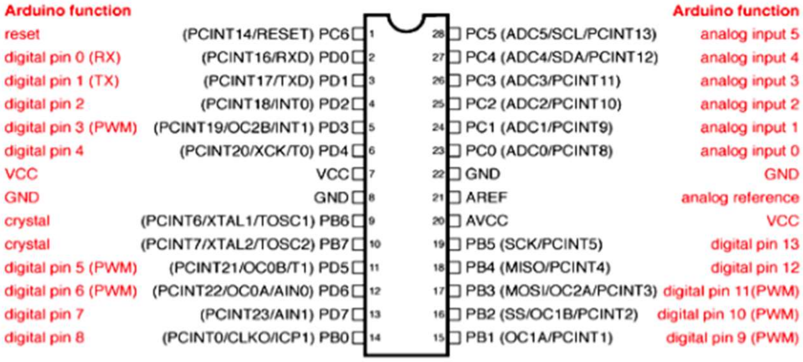
   So, in case of the correct polarity connection the mosfet will be open as

VGs = -12V

- ID : Continuous Drain Current which is equal 31 A and our circuit only 2.5A so it will be safe
- RDs(on): which is equal 60 mOhms so it will not effectively affect the power needed in this circuit

2. **Voltage regulation Circuit:** In our circuit we need a 12v for the motors and also, we need only 5V to be supplied to the Atmega as more that 5.5V will damage the microcontroller So we used
   - LM7805 Voltage Regulator that will regulate the 12 V to 5V also
   - 2 Capacitors 100nF as specified in the data sheet that we should use a capacitors with this regulator to filter any noises
3. **ATMega 328 P-PU:** This is the used microcontroller which is the brain of the whole mega project this microcontroller is the same as the used in the Arduino UNO according to its data sheet its basic connection is
   - AT pin 9, 10 is a 16 MHZ crystal oscillator and a two 22pF capacitors to generate a stable frequency signal
   - A 10uf decoupling Capacitor between the VCC and GND of the IC
4. **Atmega Pins connection**: other components like the motor drive the ultrasonic sensor and is connected to the pins is as the firmware team choose a specific pin while making the Code and we use Arduino to Atmega mapping to connect the right pins

ATMega328P and Arduino Uno Pin Mapping



Arduino function					Arduino function
reset	(PCINT14/RESET) PC6	1	28	PC5 (ADC5/SCL/PCINT13)	analog input 5
digital pin 0 (RX)	(PCINT16/RXD) PD0	2	27	PC4 (ADC4/SDA/PCINT12)	analog input 4
digital pin 1 (TX)	(PCINT17/TXD) PD1	3	26	PC3 (ADC3/PCINT11)	analog input 3
digital pin 2	(PCINT18/INT0) PD2	4	25	PC2 (ADC2/PCINT10)	analog input 2
digital pin 3 (PWM)	(PCINT19/OC2B/INT1) PD3	5	24	PC1 (ADC1/PCINT9)	analog input 1
digital pin 4	(PCINT20/XCK/T0) PD4	6	23	PC0 (ADC0/PCINT8)	analog input 0
VCC	VCC	7	22	GND	GND
GND	GND	8	21	AREF	analog reference
crystal	(PCINT6/XTAL1/TOSC1) PB6	9	20	AVCC	VCC
crystal	(PCINT7/XTAL2/TOSC2) PB7	10	19	PB5 (SCK/PCINT5)	digital pin 13
digital pin 5 (PWM)	(PCINT21/OC0B/T1) PD5	11	18	PB4 (MISO/PCINT4)	digital pin 12
digital pin 6 (PWM)	(PCINT22/OC0A/AIN0) PD6	12	17	PB3 (MOSI/OC2A/PCINT3)	digital pin 11(PWM)
digital pin 7	(PCINT23/AIN1) PD7	13	16	PB2 (SS/OC1B/PCINT2)	digital pin 10 (PWM)
digital pin 8	(PCINT0/CLKO/ICP1) PB0	14	15	PB1 (OC1A/PCINT1)	digital pin 9 (PWM)

Digital Pins 11,12 & 13 are used by the ICSP header for MOSI, MISO, SCK connections (Atmega168 pins 17,18 & 19). Avoid low-impedance loads on these pins when using the ICSP header.

5- Headers: Used for connecting the external components like the sensors to PCB

- 2 Headers for Ultra Sonic Sensors connection
- Header for Bluetooth module
- Header for programing the ATmega using Arduino as ISP

6 – ACS712 Current Sensor: This sensor measures the current flowing through its IP+ and IP- terminals and converts this current into an analog signal at the Vout (CUR_SIG) pin, which can be read by a microcontroller.

7- Voltage divider for sensing Volt: configuration with two resistors, R7 (30kΩ) and R8 (10kΩ). This setup is designed to scale down the input voltage (VCC) to a lower voltage that can be safely read by a microcontroller, when VCC is 12V, the voltage signal (VOLT_SIG) will be 3V.

8 - RGB LED: Used for indicating the speed of the motors

9 – RED LED: Used for testing the circuit after it is fabricated

10 – MOTOR Drive: We choose L293D as it has internal flyback diodes to protect against voltage spikes and it was available on makers website. The motor has 2VCC pins one for 5 volt and the other can handle 4.5 to 36volt but we will put two small dc motors, therefore a voltage between 6V and 12V is generally ideal.

Layout Design & Routing

Design

Routing

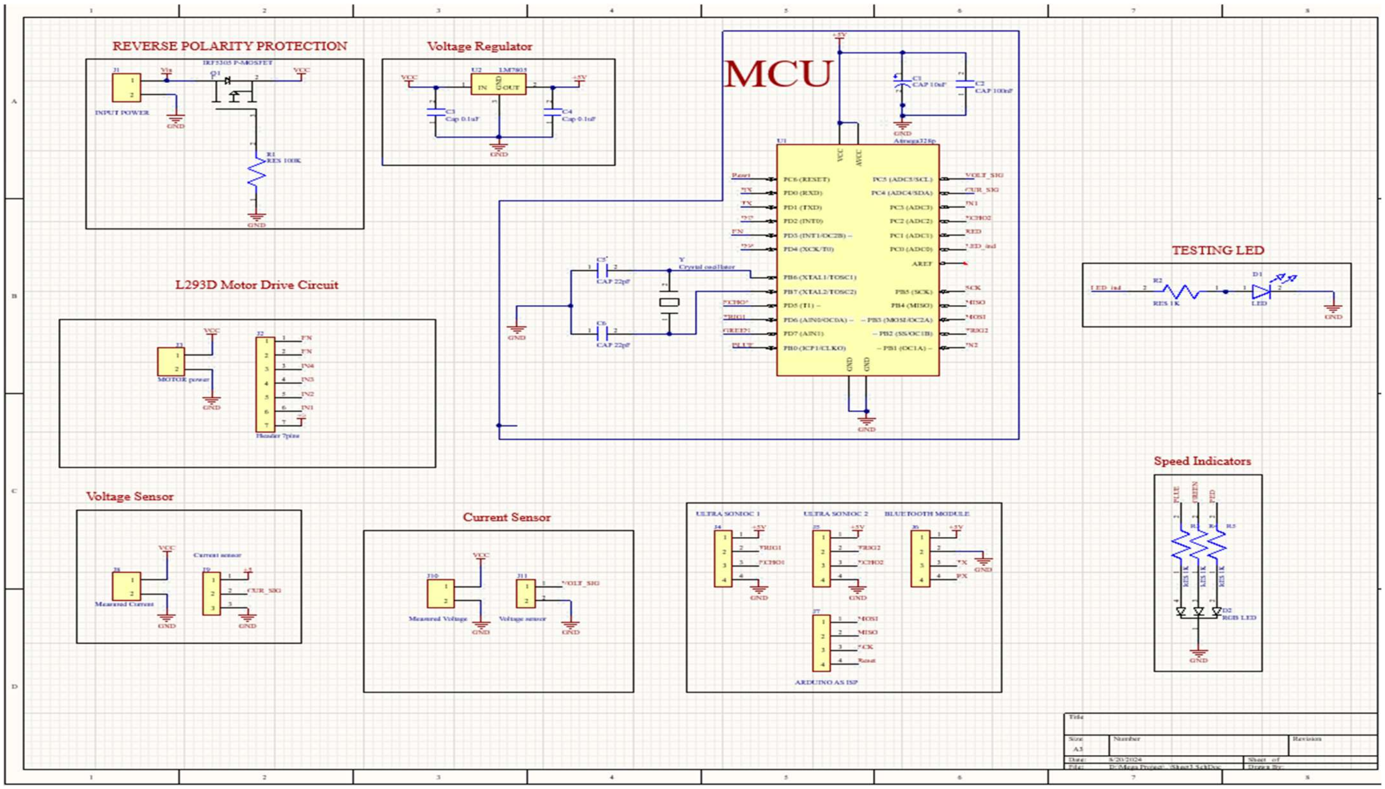
After assigning the appropriate footprints to the components, the routing process began.

- The first step was the optimal placement of components, where I aimed to achieve the best layout by considering the electrical characteristics, such as positioning the capacitor as close to the power source as possible. Additionally, I minimized the intersection of the rat's nests to simplify the routing process.
- Next, I established the design rules: a 1mm track width with a clearance of 0.7mm between copper-to-copper and track-to-copper. For power tracks, such as Vin and the motor tracks, I increased the track width to 1.5mm to accommodate the higher current these tracks would carry.
- I proceeded with routing on the bottom layer, experimenting with various methods to achieve the most efficient routing layout. Due to the complexity of creating our custom motor driver, voltage, and current modules, I added a few fly wires on the top layer to simplify the design.
- Finally, I created two polygons, one for VCC and one for GND, ensuring they passed through the pins of the respective nets as efficiently as possible. Upon running the Design Rule Checker (DRC), I identified some issues, particularly with the GND polygon, which did not fully connect to all the net's pins. I resolved these issues by clearing the surface for the polygon and removing any dead copper. For the top overlay, I added outlines to assist with soldering and to guide the end user of the board.

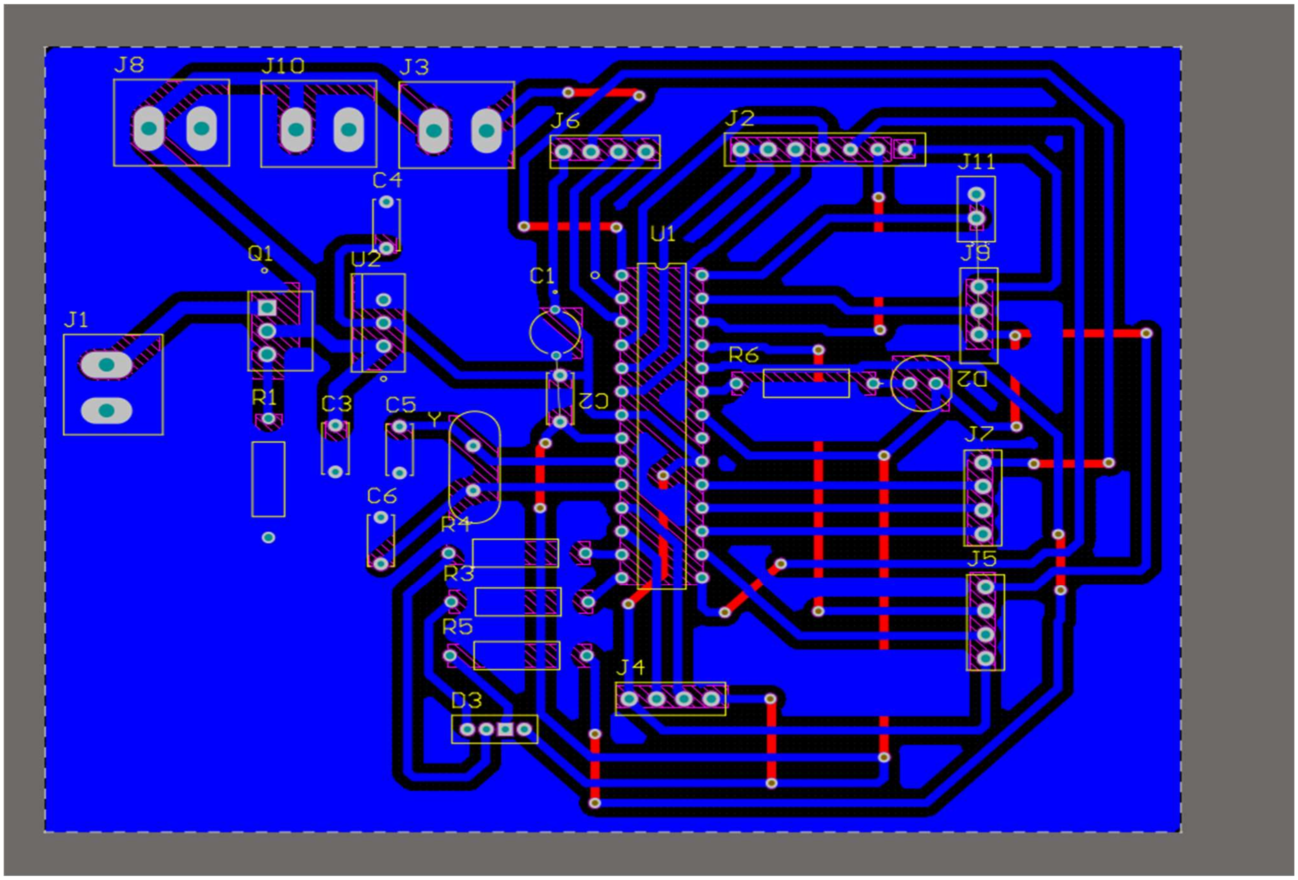
Spare Circuit

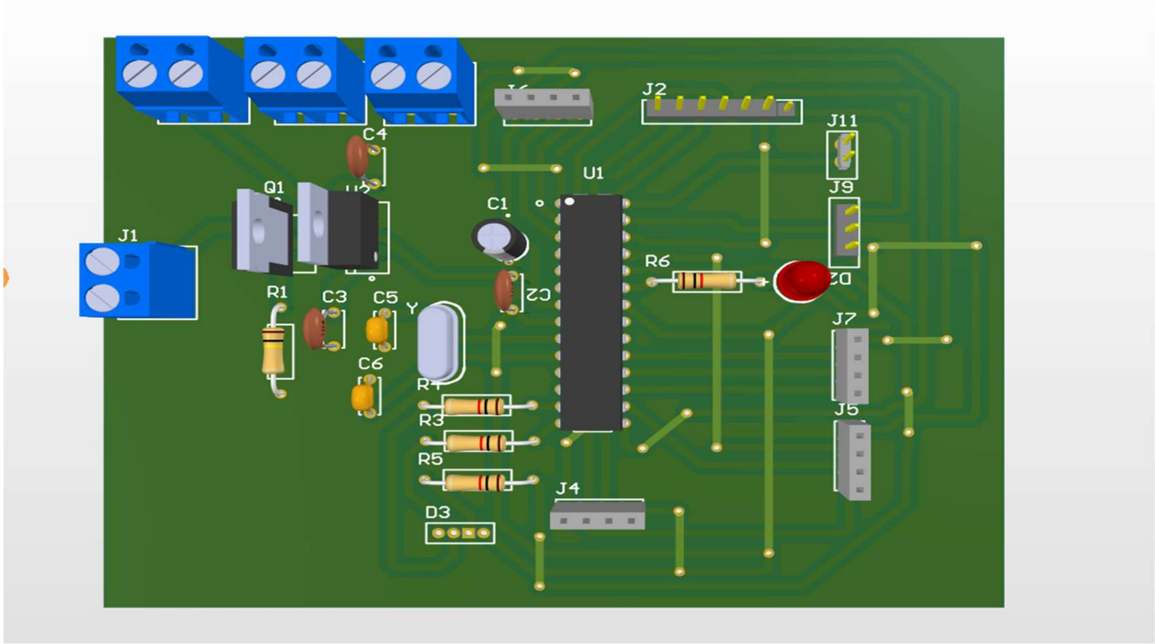
We decided to make another PCB design Put this time using the premade modules so it will be easier in fabrication and in everything this PCB was a spare because if there is any fault happen, we can fabricate and make another and easy PCB quickly

Schematic



PCB layout


J8
J10
J3
J6
J2
J11
C4
Q1
U2
U1
C1
J9
J1
R6
D2
R1
C3
C5
Y
C2
J7
C6
R4
R3
J5
R5
J4
D3


J2
J11
C4
Q1
U1
C1
J9
J1
R6
D2
C2
R1
C3
C5
Y
J7
C6
R4
J5
R3
R5
J4
D3

Software tasks' Algorithms:

Arduino Code Algorithm

Initialization:

- Initialize the microcontroller, sensors, motors, and user interface components.
- Configure the variables required for PID control, ultrasonic sensor reading, and motor control.

Inputs from user:

- Ask the user to choose the car's speed (low, medium, or high).
- Ask the user to select the mode of operation (manual or autonomous).

In manual mode

Give the driver the ability to stop, turn the car left or right move forward or backward.
Execute the corresponding motor control commands based on user input.
Press '0' to allow the user to go out of manual mode.

In autonmous mode

The code measures the separations between obstacles in the autonomous mode by analyzing data from ultrasonic sensors. To keep the car moving at a safe distance from obstacles, it computes errors and applies a PID controller.

PID Control

The PID function determines the errors for the left and right ultrasonic, adjusts the car's movements based on the errors, and makes corrections by turning left or right as needed to keep moving in path parallel to walls.

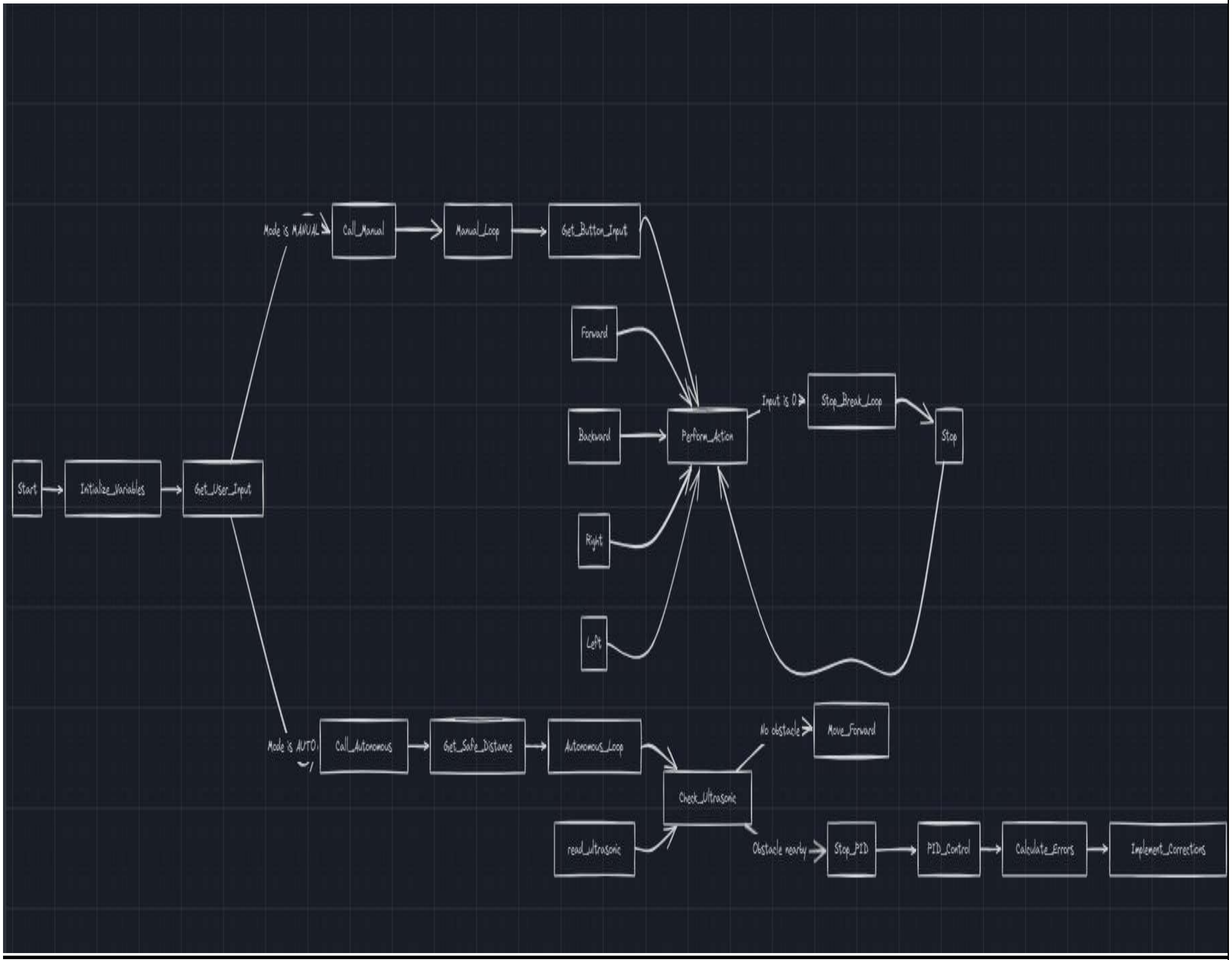
Ultrasonic Sensor Reading

To calculate the distance between car and walls, the read_ultrasonic function reads data from the ultrasonic sensors located at sides of car.

RGB LED

LED has 3 colors depending on speed of motor ,low :red color,medium :blue color,high : green color

Flow Chart


Start
Initialize_Variables
Get_User_Input
Mode is MANUAL
Call_Manual
Manual_Loop
Get_Button_Input
Forward
Backward
Right
Left
Perform_Action
Input is 0
Stop_Break_Loop
Stop
Mode is AUTO
Call_Autonomous
Get_Safe_Distance
Autonomous_Loop
Check_Ultrasonic
No obstacle
Move_Forward
read_ultrasonic
Obstacle nearby
Stop_PID
PID_Control
Calculate_Errors
Implement_Corrections

❖ GUI Features:

Our gui has the following features:

Manual Mode

In the manual mode, the GUI allows users to control the car's movements directly. When this mode is selected, the car's color changes to indicate that it is in manual mode. A new menu appears with directional buttons, enabling the user to choose the direction the car should move. This mode gives the user total control over the car's movements, making it perfect for hands-on control.

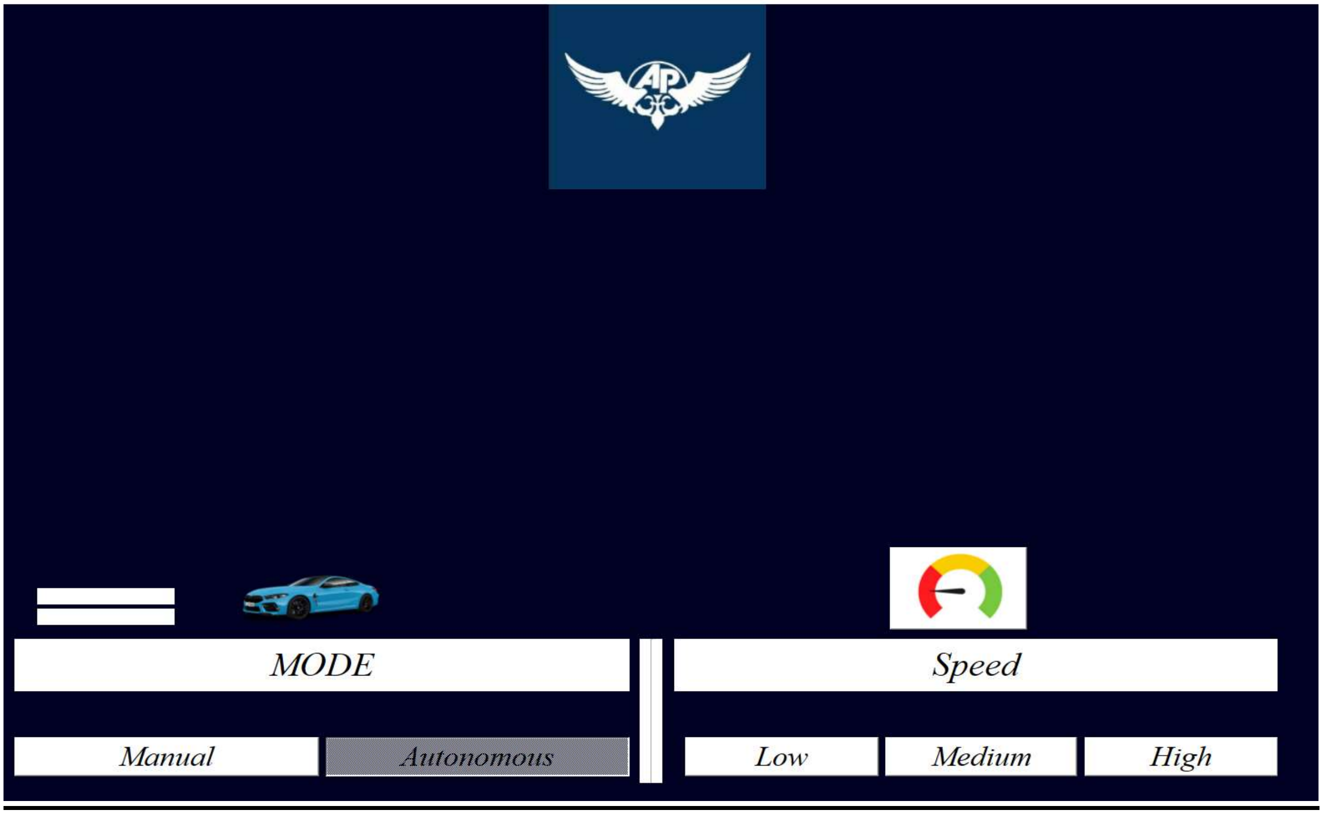
Autonomous Mode

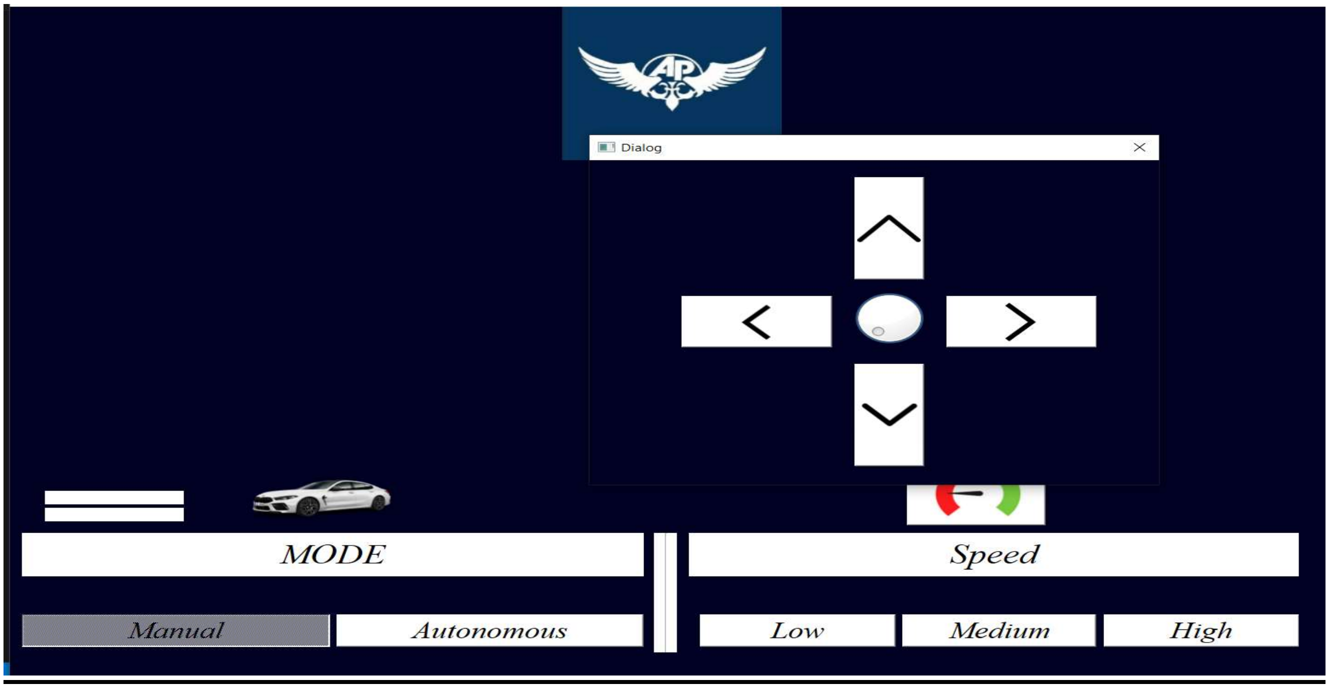
In Auto Mode, the car operates autonomously based on user inputs. Selecting this mode changes the car's color to signify the switch to autonomous control. A new window opens where the user can input the desired distance the car should travel.

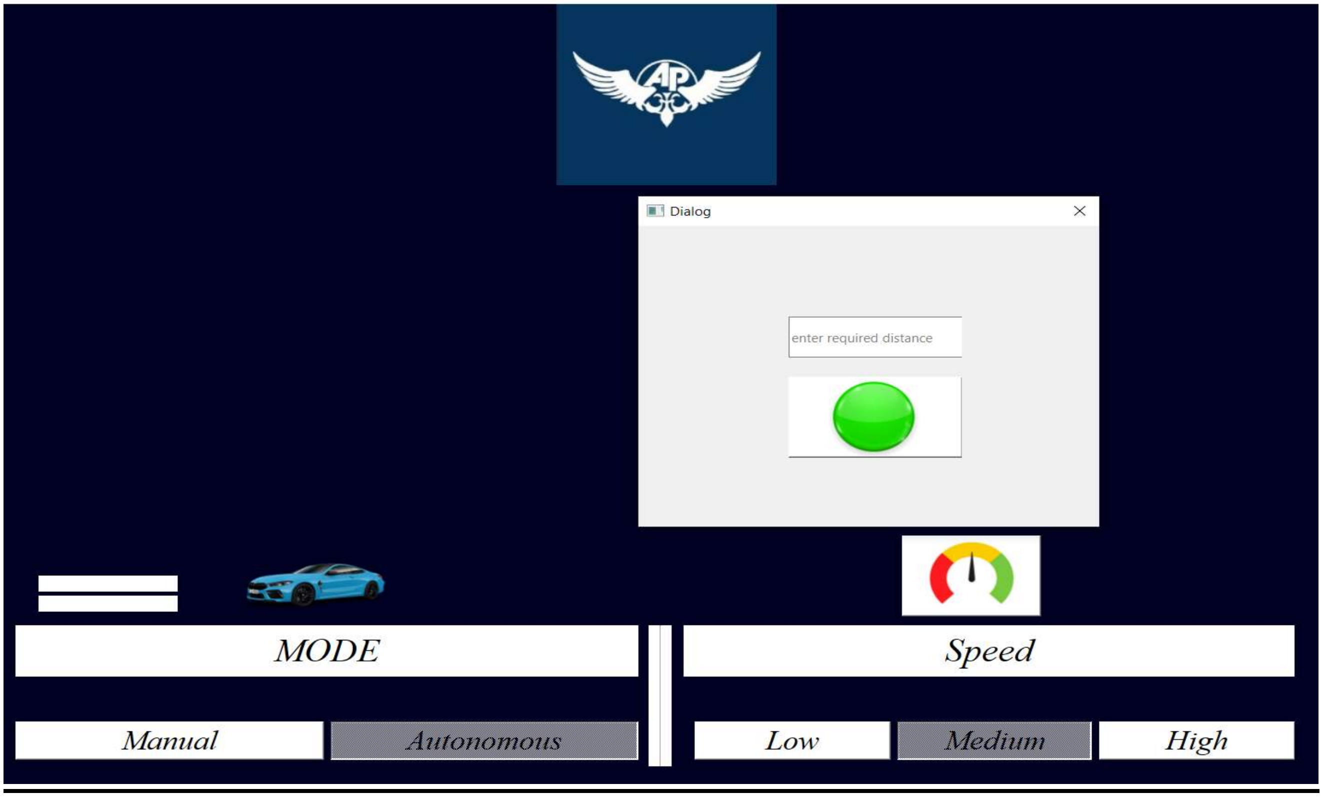
Speed

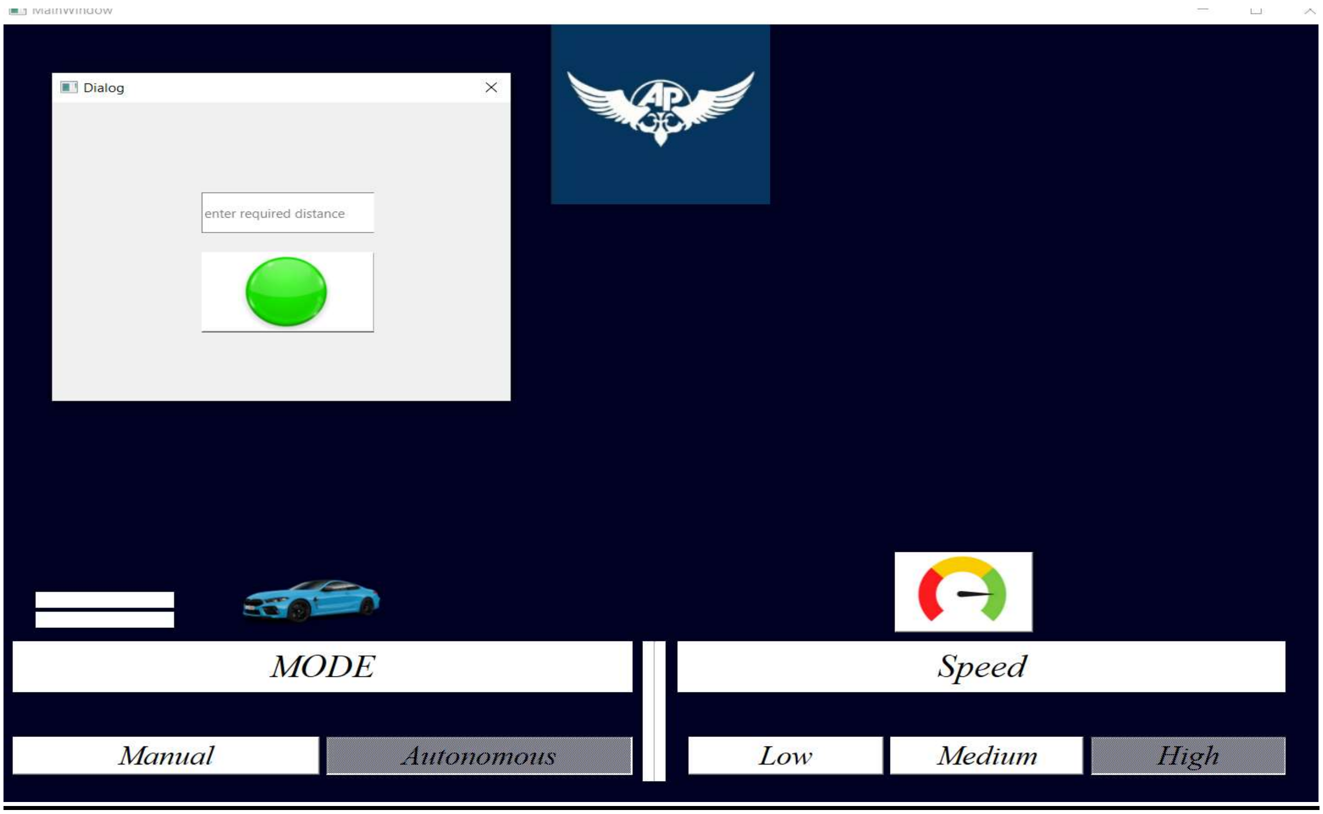
The user can choose from three speed levels: Low, Medium, and High. The speed indicator reflects the selected speed, adjusting color to match the selected level.

Main Window Features






Dialog
enter required distance
MODE
Speed
Manual
Autonomous
Low
Medium
High


MainWindow
Dialog
enter required distance
MODE
Speed
Manual
Autonomous
Low
Medium
High

Sub Windows Features

Integration in the GUI

1. **Button Actions**:
    - **Manual Mode and Auto Mode**:
        - Buttons for switching between Manual and Auto modes (`manual_pushButton` and `auto_pushButton`) change the appearance of the car icon and open the respective dialogs. The `change_car_icon_color` method updates the car icon when Auto mode is selected, and `return_car_icon_color` reverts the icon color when Manual mode is selected.
    - **Speed Selection**:
        - Buttons for speed selection (`low_pushButton`, `medium_pushButton`, `high_pushButton`) are connected to the `changeSpeed` method. Each button triggers a signal that sends the corresponding speed command to the Arduino.
2. **Dialogs**:
    - **Manual Control Dialog**:
        - Opens a dialog window with directional buttons (`up_pushButton`, `down_pushButton`, `left_pushButton`, `right_pushButton`). Each button is connected to the `sendCommand` method, which sends the respective command to the Arduino via the serial thread.
    - **Auto Mode Dialog**:
        - Opens a dialog window where users can input a distance and select a speed level. The `sendSafeDistance` method reads the input, validates it, and sends it to the Arduino. The speed level can be adjusted using the UI, and corresponding speed commands are sent to the Arduino.
3. **Serial Communication**:
    - `SerialThread`:
        - Manages serial communication in a separate thread to avoid blocking the GUI. It reads data from the Arduino and emits signals (`data_received`) that update sensor values in the UI.
    - **Sending Commands**:
        - Commands for speed and mode are sent to the Arduino using the `sendCommand` method. This method is called from various UI actions, including speed button clicks and dialog button actions.
4. **Sensor Updates**:
    - `updateSensorValues`:
        - This method updates the UI with data received from the Arduino, such as current and voltage readings. It processes incoming serial data and updates the corresponding QLabel widgets in the main window.
5. **Event Handling**:
    - `closeEvent`:
        - Ensures the serial thread is stopped and the serial port is closed when the application is closed.

Computer vision codes:

Video stitching algorithm:

This algorithm is used to stitch frames from two different videos in order to combine two views into one view using opencv library

The algorithm passes through many processes including: loading videos into the algorithm using file explorers ,stitching creation , frame resizing,multithreading and finally combining both videos

1-**video loading:** videos loaded using cv2.VideoCapture() to read every frame in both videos

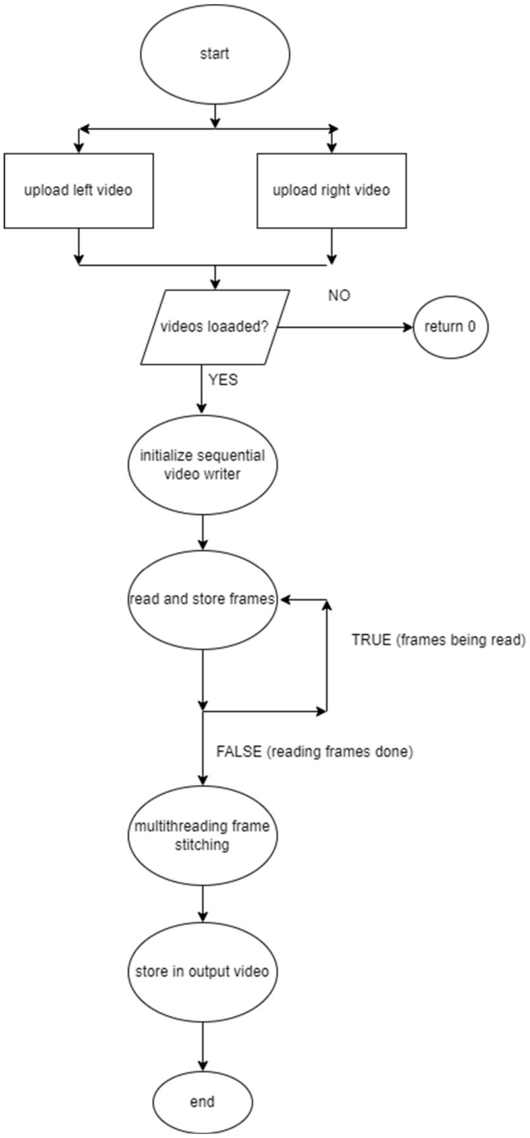
2-**cv2.stitcher,Create() and panorama algorithm:** are used to combine left and right views into a single panoramic view

3-**frame resizing:** after stitching is complete,stitched frames should be resized to match eachother to have suitable dimensions

4**- combining videos (main function):** finally videos are combined using combineVideo function which loads both videos and combine them sequentially and then loads them into the output video with suitable format

5-**multithreading**: multithreading is also used to improve performance and enhancing processing speed by using parallel processing (multi frames are processed at a time)

Code sequence and flow chart:


start
upload left video
upload right video
videos loaaded?
NO
return 0
YES
initialize sequential video writer
read and store frames
TRUE (frames being read)
FALSE (reading frames done)
multithreading frame stitching
store in output video
end

Stereo vision algorithm:

Stero vision lgorithm is used to process 2D images and extract 3d information from them as depth by comparing multi image parameters taken from different views.

The algorithm starts by loading two images pathes for processing, then passes through multiple steps as calibaration, rectification ,correspondence and depth image compution

1-calibaration process :

a-SIFT(scale invariant feture transformer) algorithm used to detect key points and descriptors in both images and for feature matching

b-Flann: then flann lgorithm(based on nearest neighbor searching algorithm) is used to match those key points from sift algorithm

c-low’s ratio test: used s a filter to save only good matches based on the match ratio(0.6 in my code)

then matches are shown on both images

d-fundmental matrix:fundmental matrix is a 3x3 matrix computed to show epipolar geometry in both images ,we can get fund. Matrix using key points coordinates which are used as parameters for cv2.findFundmentalMat() function

e- essential matrix: used to encode translation and rotation that happened between both views (left and right images) , to calculate essential matrix,we need fund. Matrix to be calculated and both cameras focal length and principal points should be known(I used values from sample files), then from essential matrix , rotation and translation should be calculated easily using cv2.recoverpose() which takes fund matrix ,keypoints coordinates and camera parameters as its parameters (essential mat= cam1(transpose) * fundmental mat * cam0)

2-rectification process:

Rectification process changing imge perspective to align images and epipolr lines together on same horizontal plane by using matrix transformation/rotation using cv2.stereoRectifyUncalibrated function which also outputs homography matrix

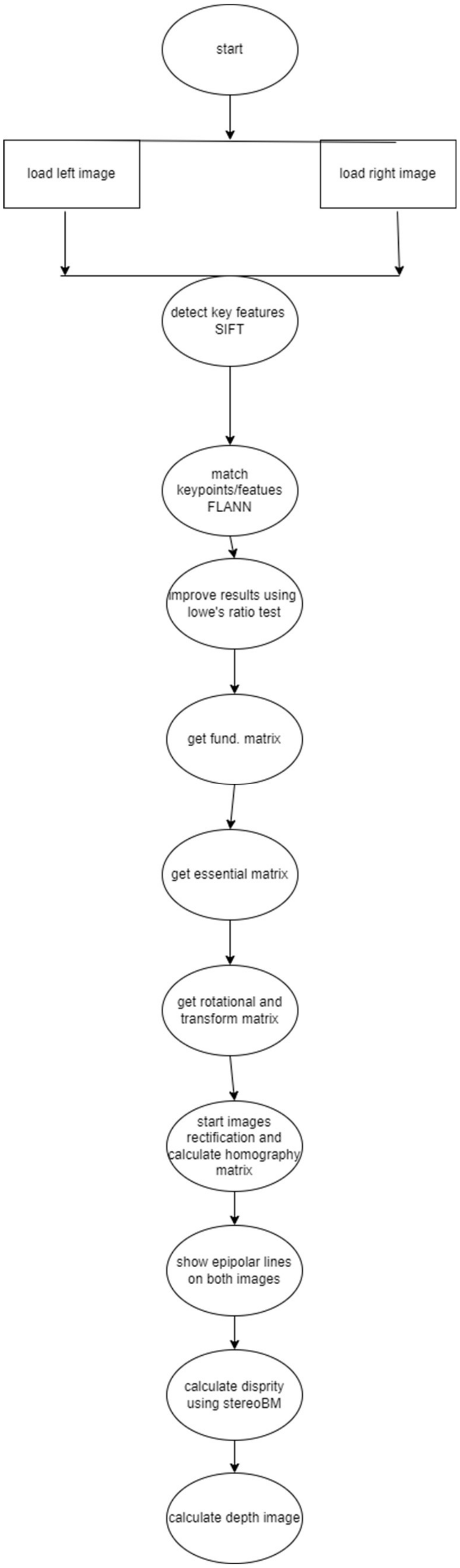
a-Epipolar lines : computed on both images using cv2.computeCorrespondEpilines() function, its used visiualize epipolar geometry on both images and show the diiference between two views depending on one focus point

3- correspondence:

In this process , disparity map ,which shows the horizontal distance difference between same points in two different views is calculated and visuilaized using stereoBm (stereo block matching)algorithm using suitable number of disparities(number of search pixels) and block size and then normalization is made for better visuiliation

4-**depth image compution:** depth image is used to show close and far objects from the camera as grayscal image using the equation depth =(focal length * baseline/disparity)

Flowchart and code sequence:


start
load left image
load right image
detect key features SIFT
match keypoints/featues FLANN
improve results using lowe's ratio test
get fund. matrix
get essential matrix
get rotational and transform matrix
start images rectification and calculate homography matrix
show epipolar lines on both images
calculate disprity using stereoBM
calculate depth image