TECHNICAL REPORT FOR AQUAPHOTN’S MEGATRAINING PROJECT

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 vehicle 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 voltage, current, and ultrasonic sensor data.
- 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-Discussion

Hardware

❖ 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 autonomous mode

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

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 a path parallel to walls.

Ultrasonic Sensor Reading

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

RGB LED

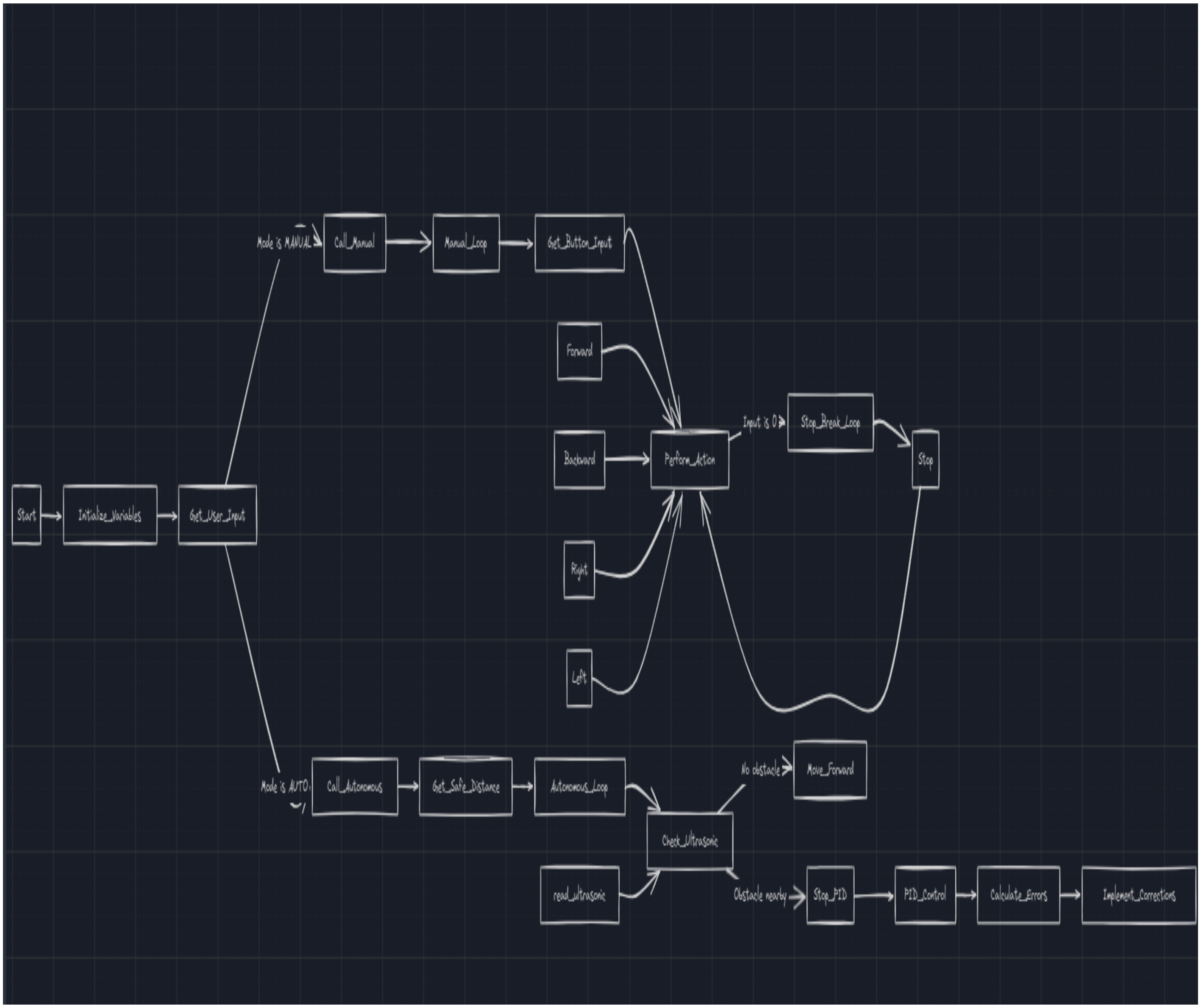
LED has 3 colors depending on the speed of the motor

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
read_ultrasonic
No obstacle
Move_Forward
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.

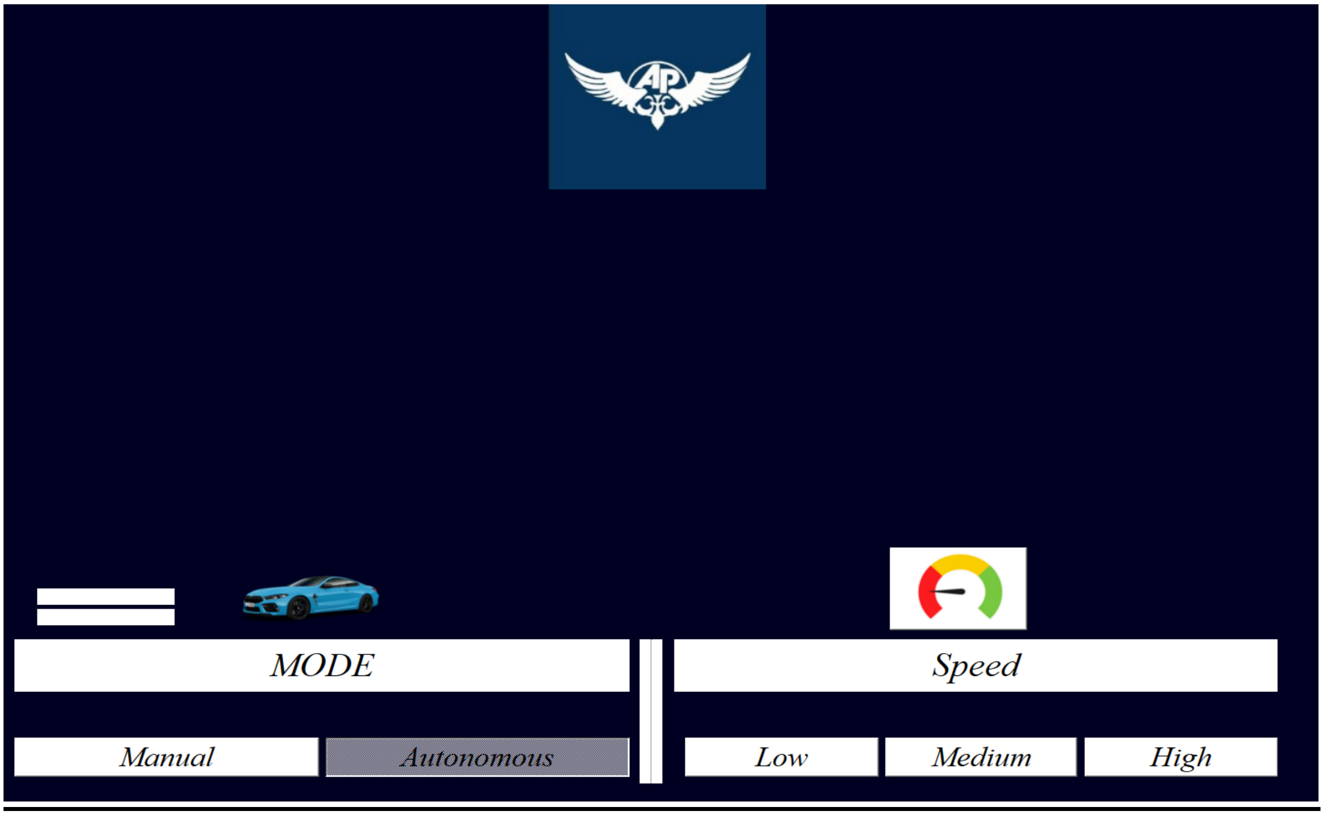
Autonomous Mode

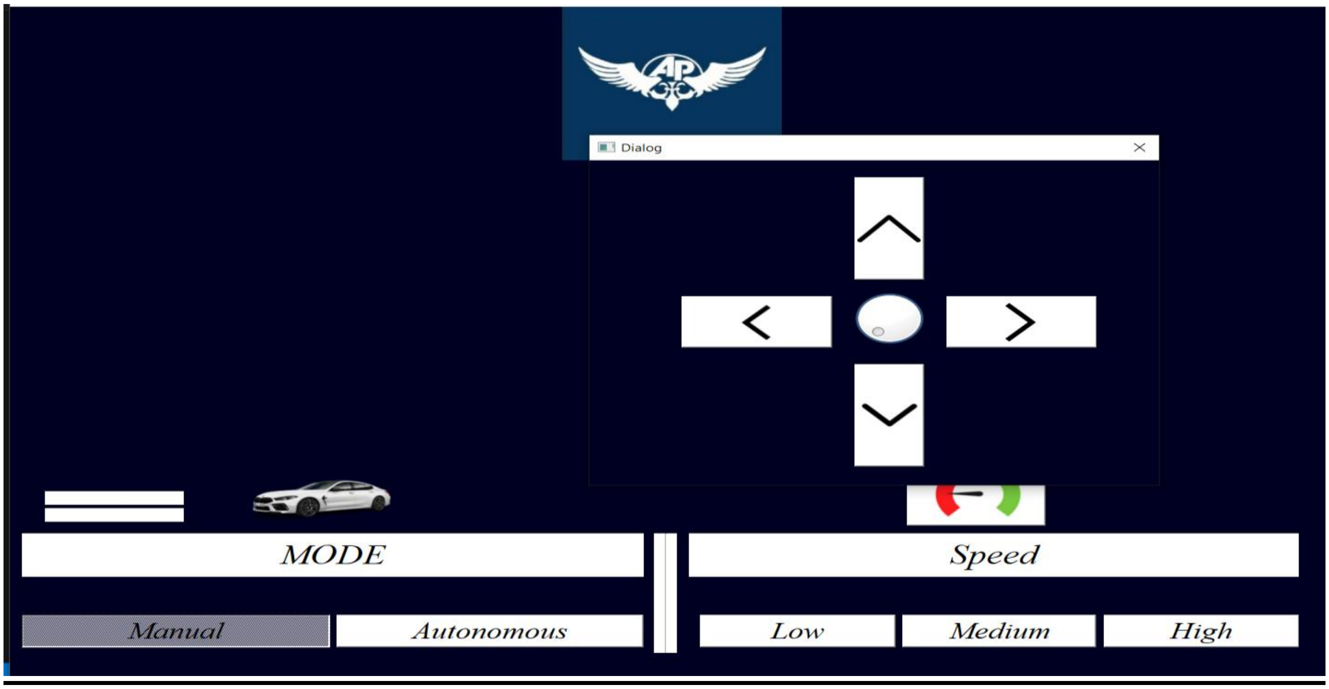
In Auto Mode, the car operates autonomously based on user inputs. Selecting this mode changes the car's color to signify the switch to autonomous control. A new window opens where the user can input the desired distance the car should travel.

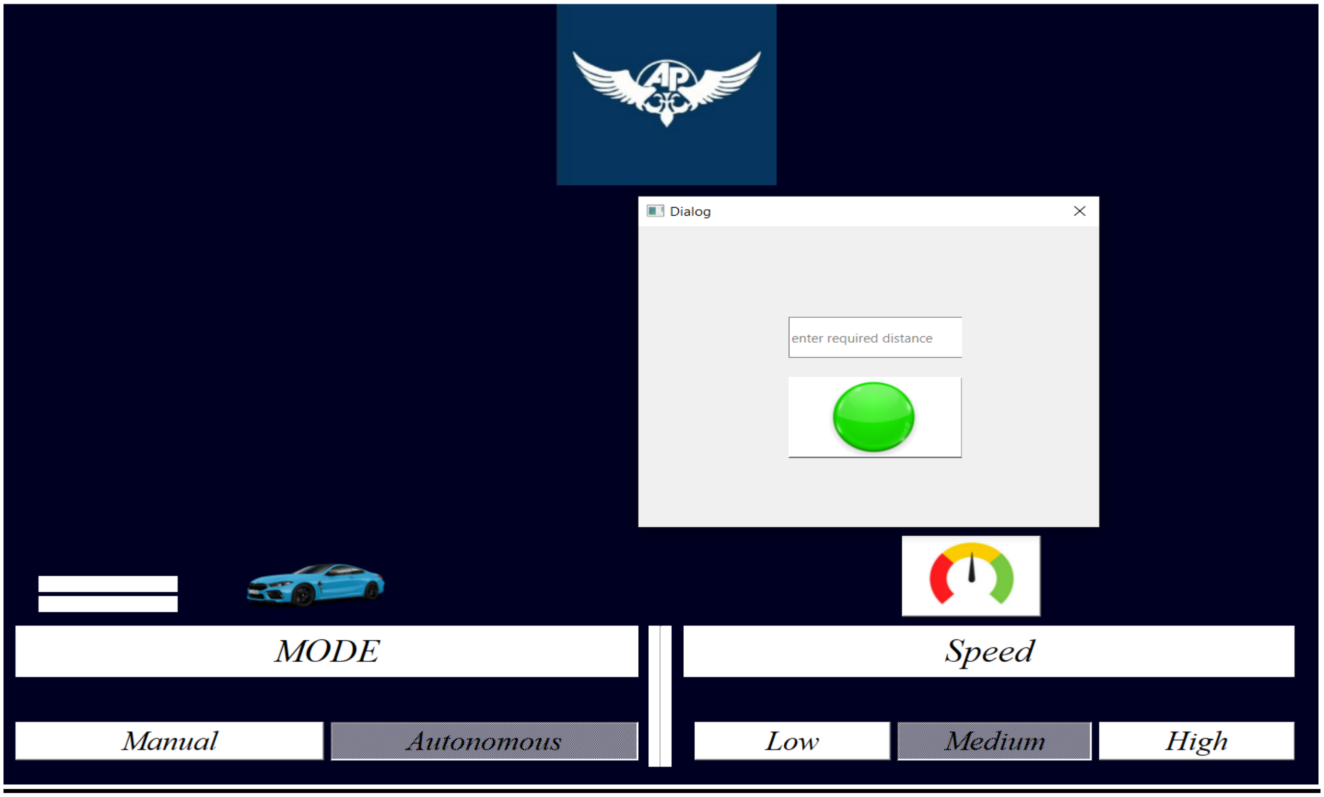
Speed

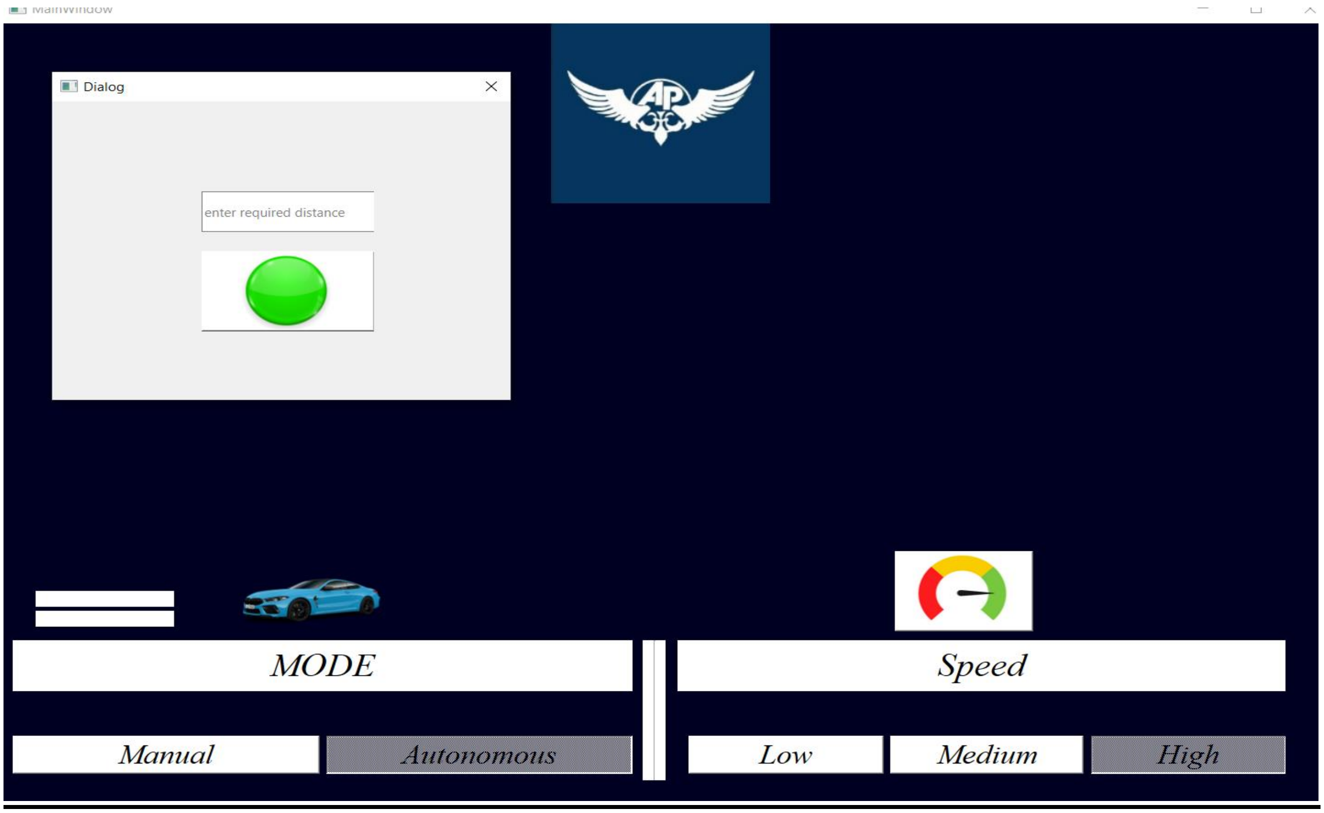
The user can choose from three speed levels: Low, Medium, and High. The speed indicator reflects the selected speed, adjusting the color to match the selected level.

Main Window Features




Dialog
MODE
Speed
Manual
Autonomous
Low
Medium
High


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**:
    - `close event`:
        - 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

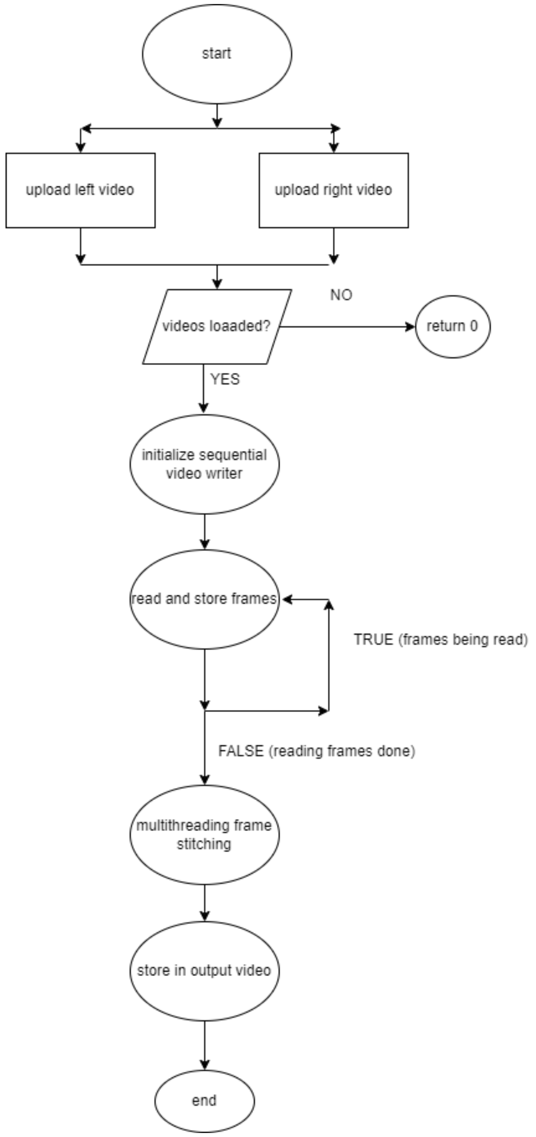
2-**cv2.stitcher,Create() and panorama algorithm:** are used to combine left and right views into a single panoramic view

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:



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

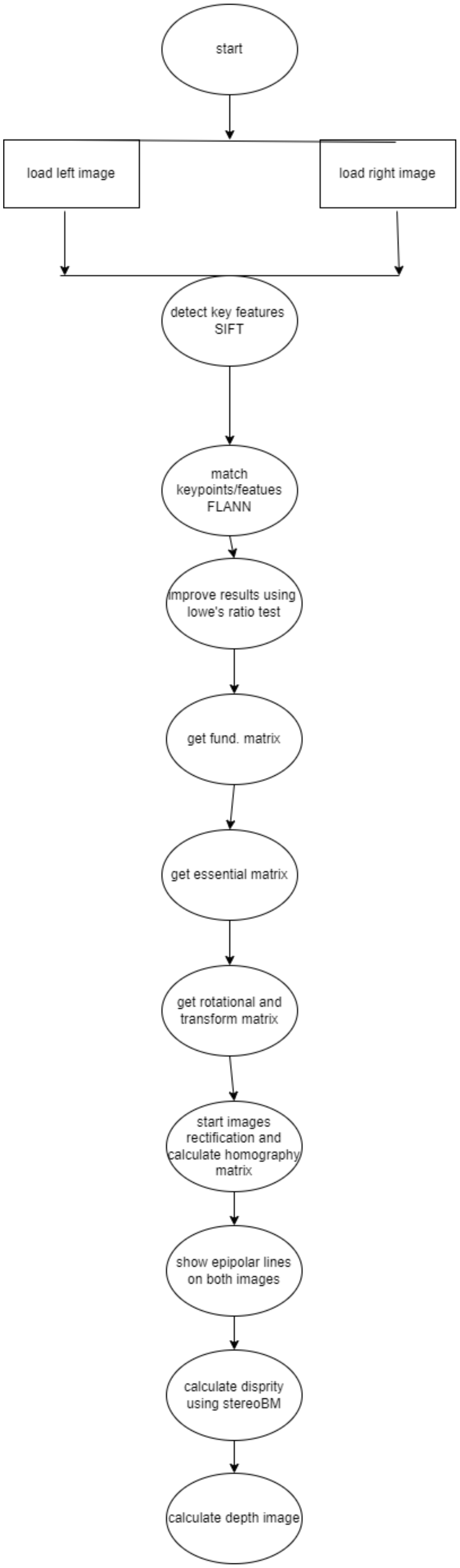
a-Epipolar lines : computed on both images using cv2.computeCorrespondEpilines() function, its used visiualize epipolar geometry on both images and show the diiference between two views depending on one focus point

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 dispriity using stereoBM
calculate depth image