ROV system GUI Report

1. INTRODUCTION

1.1 Abstract

This project presents a modular, GUI-based control system for a Remotely Operated Vehicle (ROV), integrating several advanced features including multithreaded video stitching, stereo vision depth estimation, and serial communication with an Arduino-powered control interface. The GUI is developed using PyQt6 and features dedicated sub-windows for manual and autonomous control, live camera feed with screenshot capability, stitched video playback, and stereo vision processing with depth computation. The system employs a clean OOP architecture, threading for responsive video operations, and serial protocols for real-time interaction with hardware. Despite some limitations in feature completion, the project showcases modular software design, interface engineering, and the integration of computer vision with interactive GUI element

Attached too is a visual representation of each decision the user determines, in case of the task to be implemented, the outputs, the test cases, …etc.

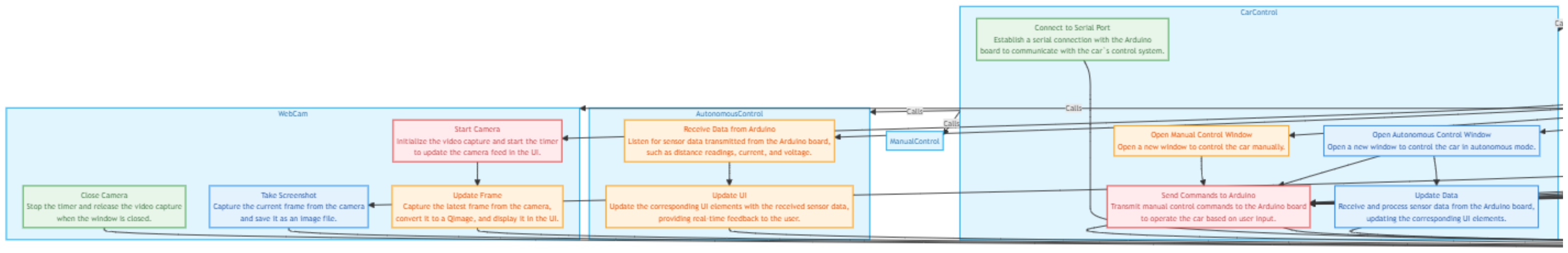
The hardware part was implemented but not attached.

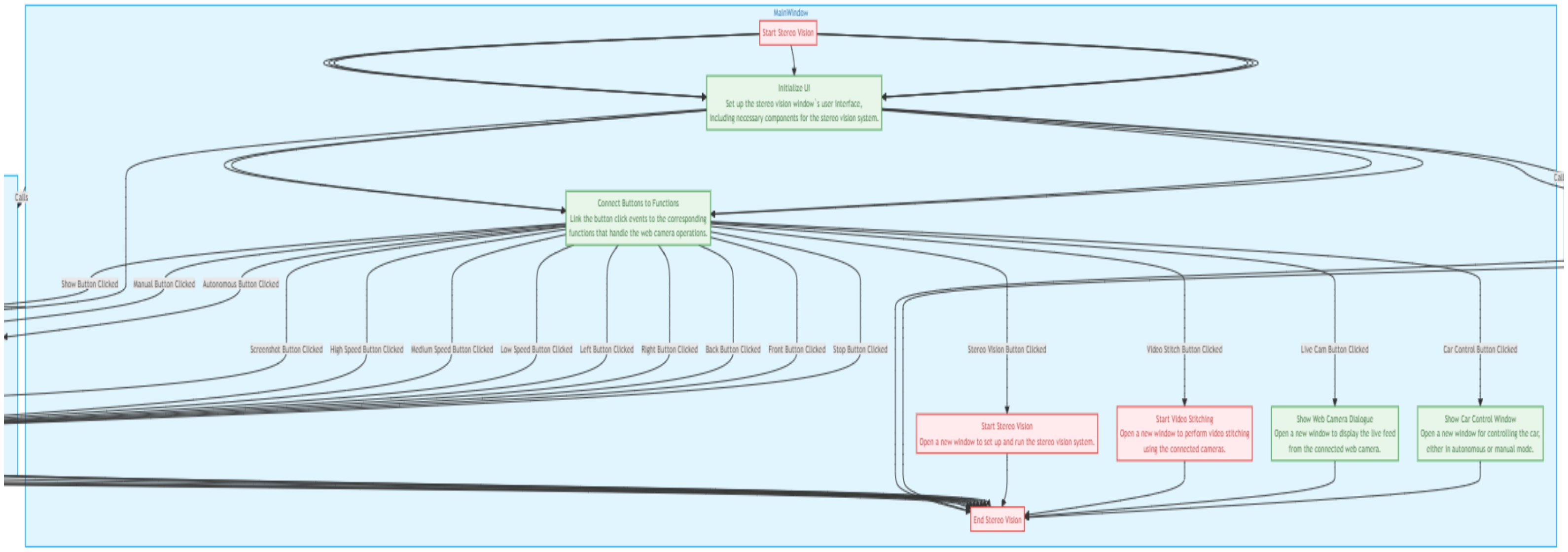
Table of Content:

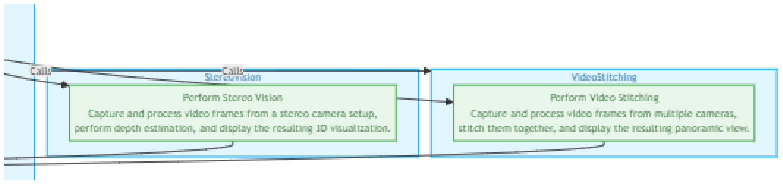
2. Software

2.1 GUI

2.1.1 Flowchart


WebCam
Start Camera
Initialize the video capture and start the timer to update the camera feed in the UI.
Update Frame
Capture the latest frame from the camera, convert it to a QImage, and display it in the UI.
Take Screenshot
Capture the current frame from the camera and save it as an image file.
Close Camera
Stop the timer and release the video capture when the window is closed.
AutonomousControl
Receive Data from Arduino
Listen for sensor data transmitted from the Arduino board, such as distance readings, current, and voltage.
Update UI
Update the corresponding UI elements with the received sensor data, providing real-time feedback to the user.
ManualControl
Calls
Calls
CarControl
Connect to Serial Port
Establish a serial connection with the Arduino board to communicate with the car`s control system.
Calls
Open Manual Control Window
Open a new window to control the car manually.
Open Autonomous Control Window
Open a new window to control the car in autonomous mode.
Send Commands to Arduino
Transmit manual control commands to the Arduino board to operate the car based on user input.
Update Data
Receive and process sensor data from the Arduino board, updating the corresponding UI elements.

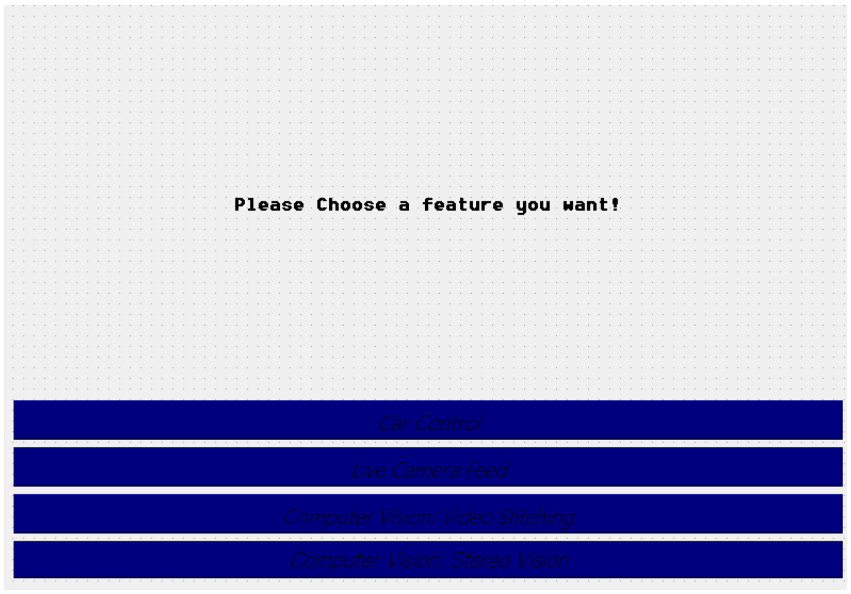

MainWindow
Start Stereo Vision
Initialize UI
Set up the stereo vision window`s user interface, including necessary components for the stereo vision system.
Calls
Connect Buttons to Functions
Link the button click events to the corresponding functions that handle the web camera operations.
Show Button Clicked
Manual Button Clicked
Autonomous Button Clicked
Screenshot Button Clicked
High Speed Button Clicked
Medium Speed Button Clicked
Low Speed Button Clicked
Left Button Clicked
Right Button Clicked
Back Button Clicked
Front Button Clicked
Stop Button Clicked
Stereo Vision Button Clicked
Video Stitch Button Clicked
Live Cam Button Clicked
Car Control Button Clicked
Start Stereo Vision
Open a new window to set up and run the stereo vision system.
Start Video Stitching
Open a new window to perform video stitching using the connected cameras.
Show Web Camera Dialogue
Open a new window to display the live feed from the connected web camera.
Show Car Control Window
Open a new window for controlling the car, either in autonomous or manual mode.
End Stereo Vision


Calls
StereoVision
Calls
Perform Stereo Vision
Capture and process video frames from a stereo camera setup, perform depth estimation, and display the resulting 3D visualization.
VideoStitching
Perform Video Stitching
Capture and process video frames from multiple cameras, stitch them together, and display the resulting panoramic view.

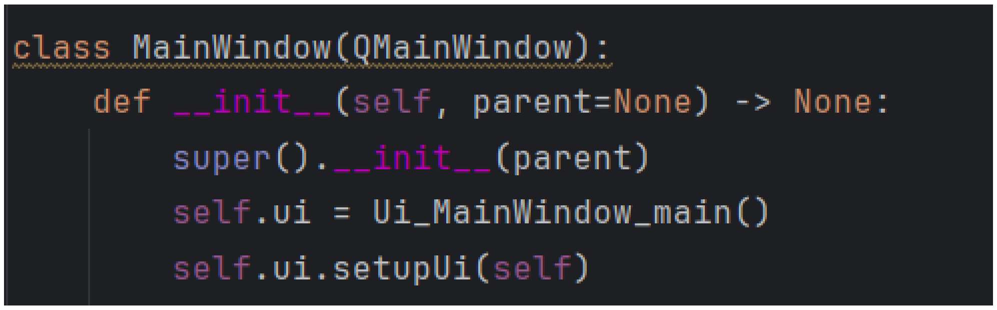
2.2 Main Window Features

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.

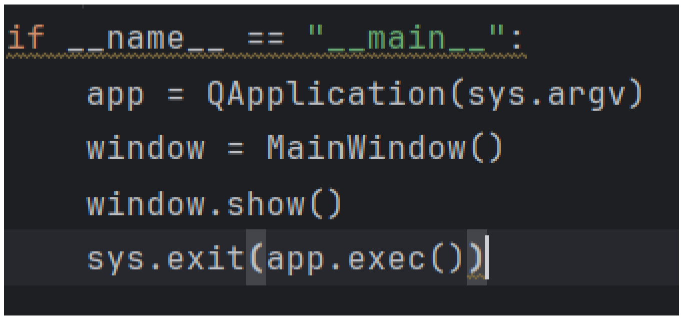

Please Choose a feature you want!
Car Control
Live Camera Feed
Computer Vision: Video Stitching
Computer Vision: Stereo Vision

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.



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

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



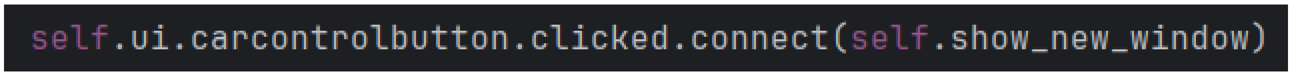
```python
if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec())
```

This snippet is the most important part of the code. Without it the code will not be running sets up the execution environment for your PyQt application.

2.3 Sub Windows Features

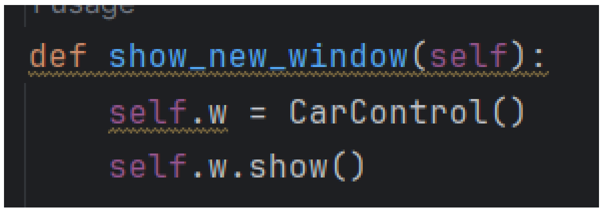
2.3.1 Car Control

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



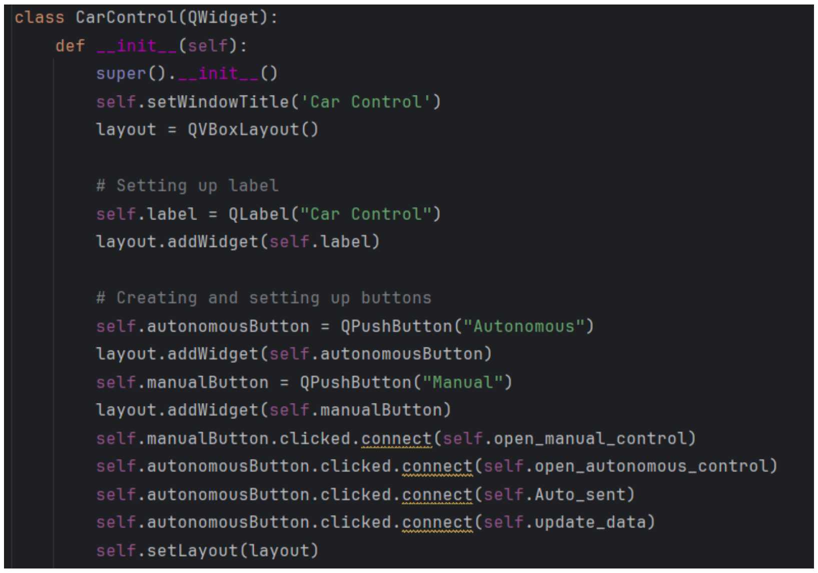
```python
self.ui.carcontrolbutton.clicked.connect(self.show_new_window)
```

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.



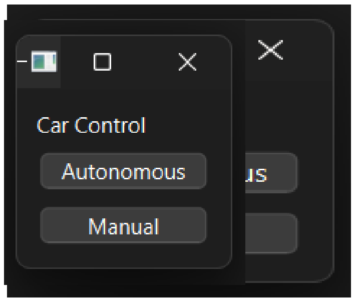
```python
def show_new_window(self):
    self.w = CarControl()
    self.w.show()
```

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



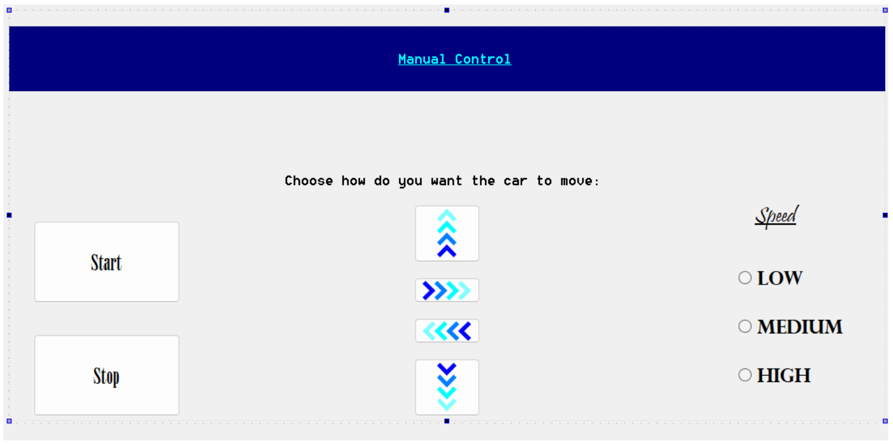
```
class CarControl(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('Car Control')
        layout = QVBoxLayout()

        # 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)
```

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.

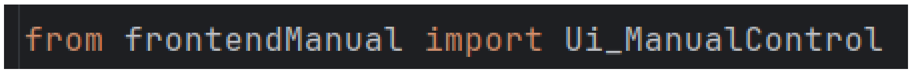


Manual Mode:

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.

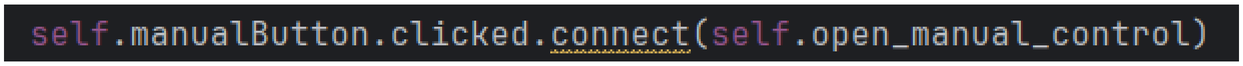


A blue frame has been added to enhance the design. Two push buttons were included for controlling the car: one to start and one to stop the car. Additionally, four push buttons indicate

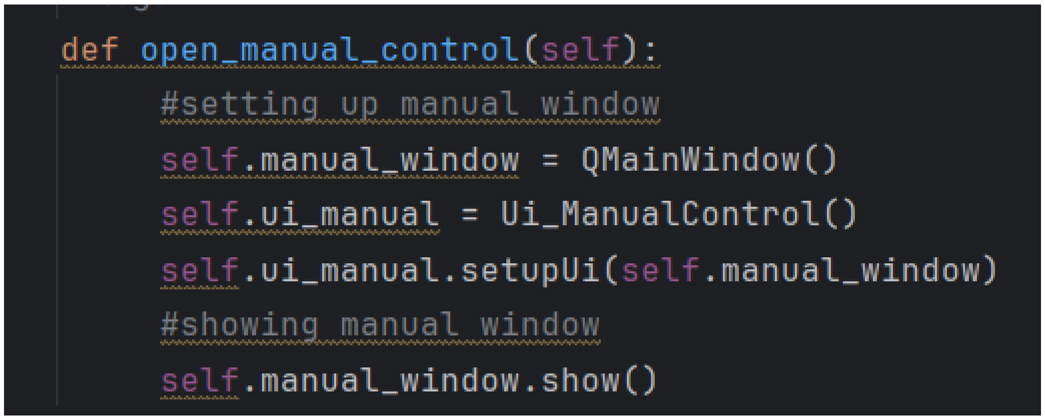


```
from frontendManual import Ui_ManualControl
```

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.



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

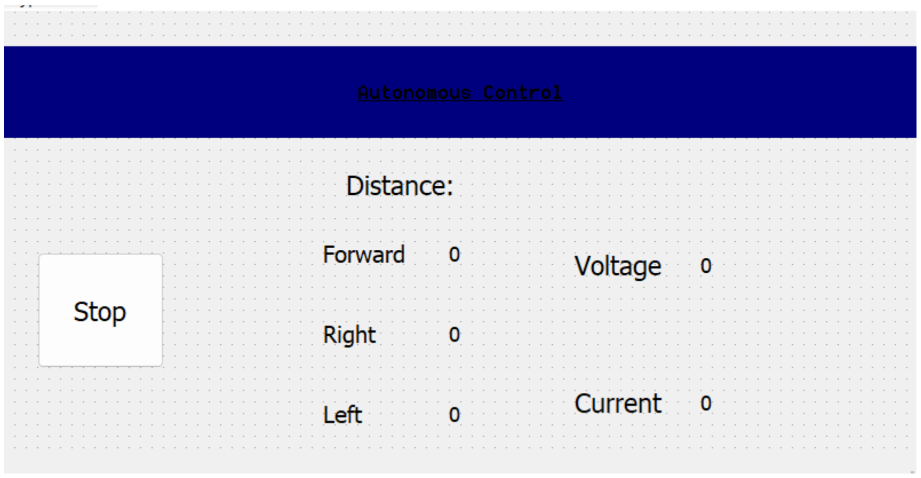


```
def open_manual_control(self):
    #setting up manual window
    self.manual_window = QMainWindow()
    self.ui_manual = Ui_ManualControl()
    self.ui_manual.setupUi(self.manual_window)
    #showing manual window
    self.manual_window.show()
```

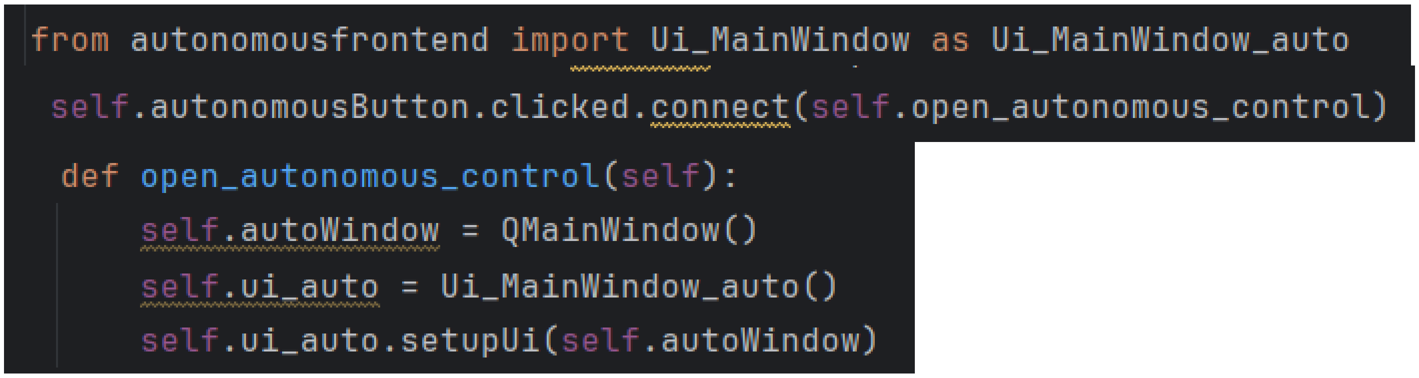
The pictures above illustrate how the manual control window was implemented and connected to the car control window. The frontendManual file, which contains the manual control window design, is imported to access its main window class. When the "Manual" button is clicked in the car control window, it 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.

Autonomous mode:

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.



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.

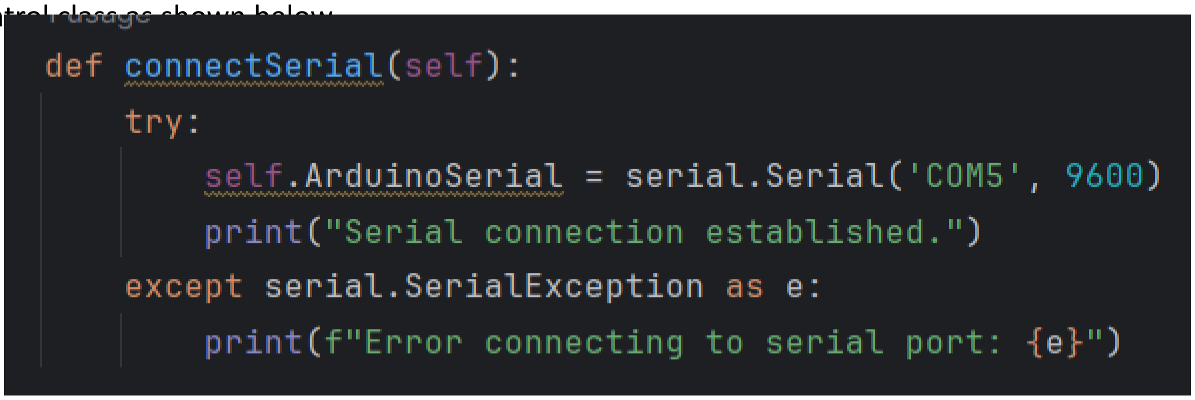


```python
from autonomousfrontend import Ui_MainWindow as Ui_MainWindow_auto
 self.autonomousButton.clicked.connect(self.open_autonomous_control)
  def open_autonomous_control(self):
      self.autoWindow = QMainWindow()
      self.ui_auto = Ui_MainWindow_auto()
      self.ui_auto.setupUi(self.autoWindow)
```

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:

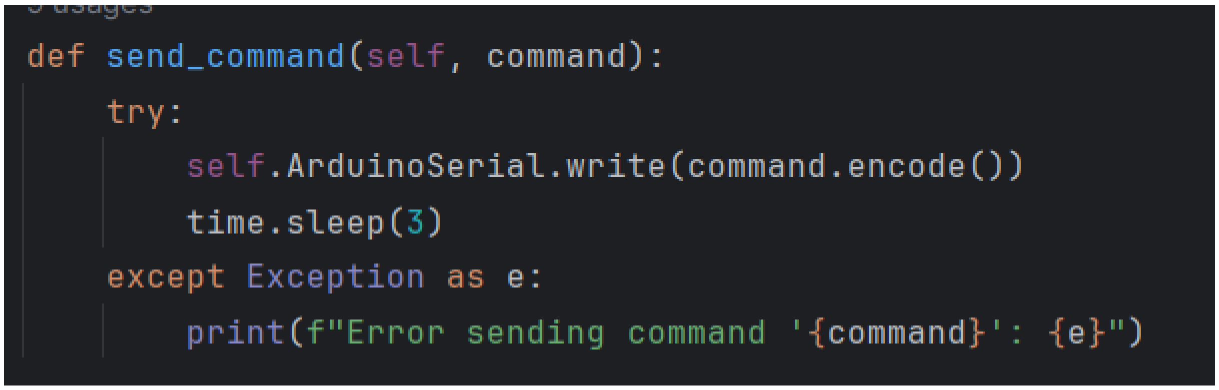
To realistically communicate with the serial connection between the Arduino and the GUI is established. This is done by sending and receiving messages from and to the Arduino. First thing to do is import serial in the python file. Next, serial must be connected, and this is done by a function called ‘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}")
```

‘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.

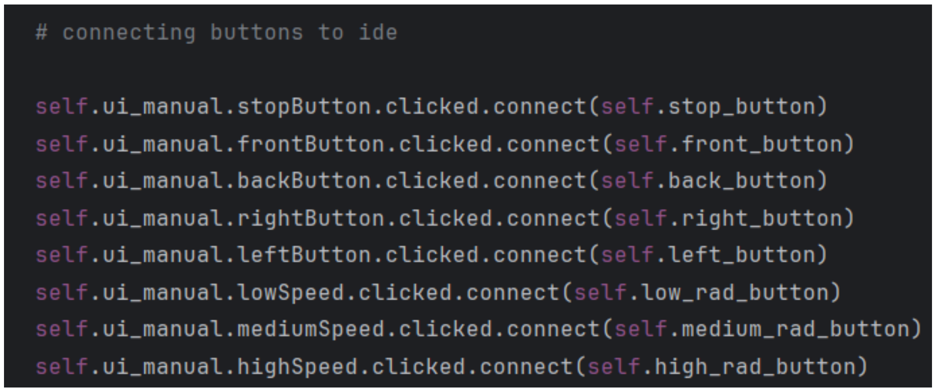


```python
def send_command(self, command):
    try:
        self.ArduinoSerial.write(command.encode())
        time.sleep(3)
    except Exception as e:
        print(f"Error sending command '{command}': {e}")
```

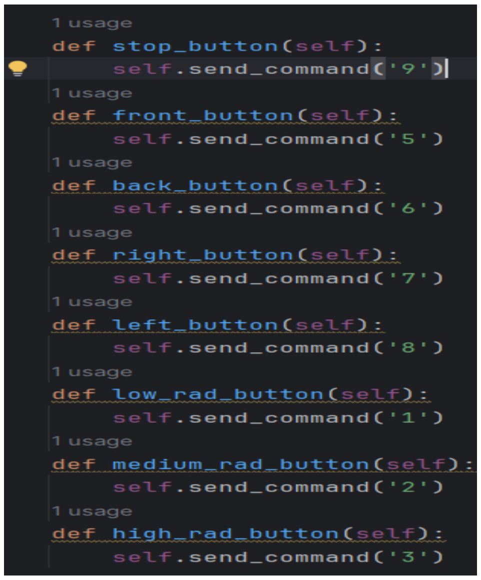
When writing data to a serial port using PySerial, it is essential to ensure that the data is in the correct format. Serial communication typically requires data to be in bytes, not strings. Python's .encode() method is used to convert a string into bytes, which is necessary for sending data over a serial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

How is this used in buttons connection and displaying outputs?

Manual Buttons:



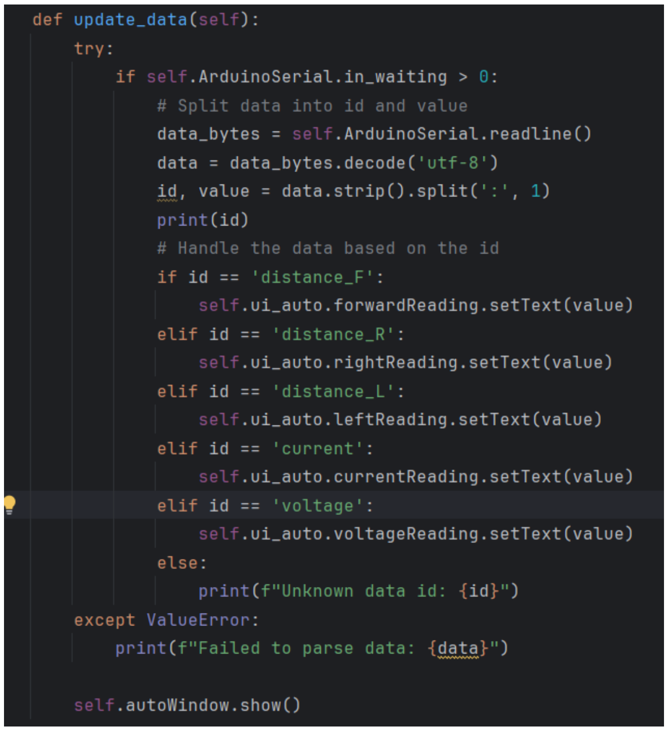
```
# connecting buttons to ide

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)
```



```
1 usage
def stop_button(self):
    self.send_command('9')
1 usage
def front_button(self):
    self.send_command('5')
1 usage
def back_button(self):
    self.send_command('6')
1 usage
def right_button(self):
    self.send_command('7')
1 usage
def left_button(self):
    self.send_command('8')
1 usage
def low_rad_button(self):
    self.send_command('1')
1 usage
def medium_rad_button(self):
    self.send_command('2')
1 usage
def high_rad_button(self):
    self.send_command('3')
```

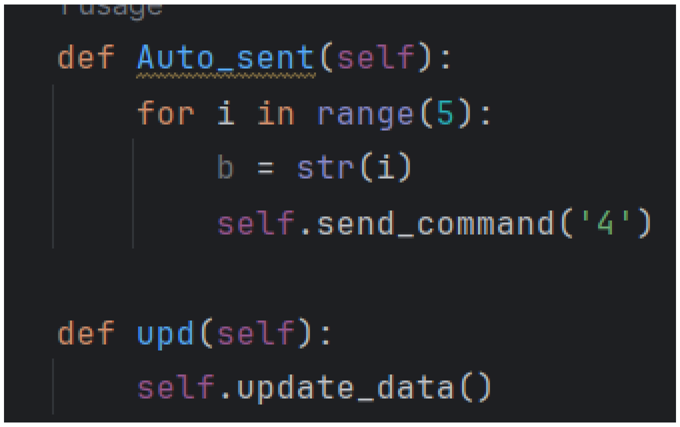
Each button clicked a function is called to send a certain command to the Arduino. How the car executes these commands is controlled by the arduino.

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()
```

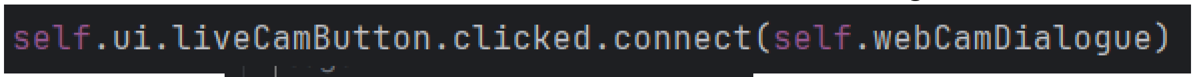
In this function data is received from the serial and text is set in the autonomous window. .readline reads data in bytes so it needs to be decoded. Data is sent by a specific format which is the name of the object reading is taken from then a colon then the actual reading. These are splitted and the ‘:’ 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.

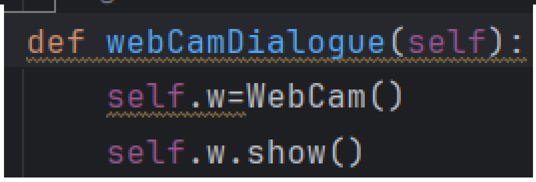

```
def Auto_sent(self):
    for i in range(5):
        b = str(i)
        self.send_command('4')

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

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

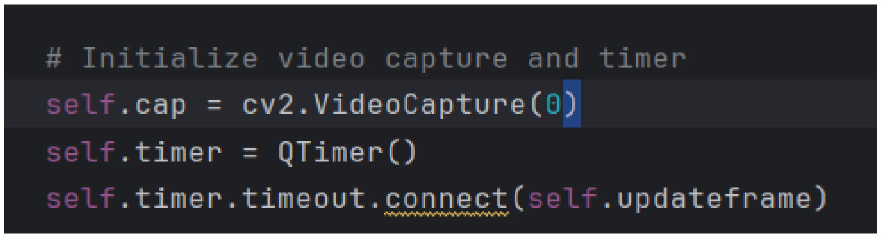
2.3.2 web cam

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.

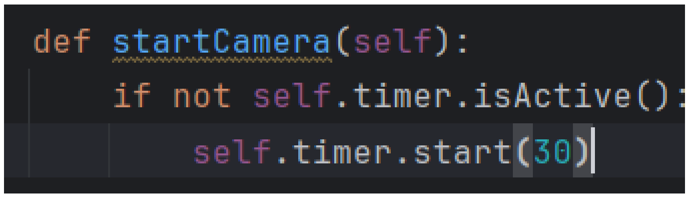

```
self.ui.liveCamButton.clicked.connect(self.webCamDialogue)
```


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

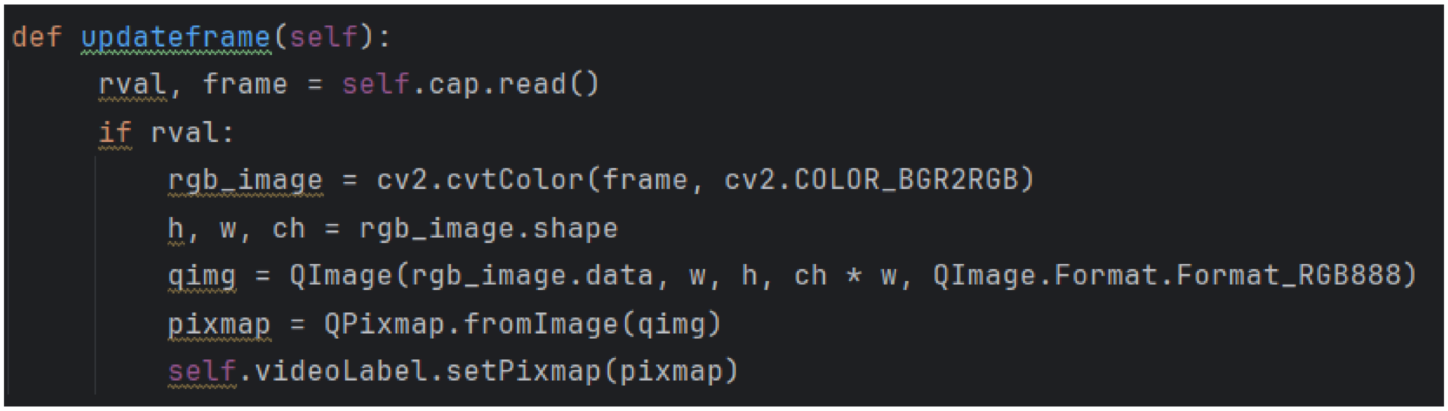
Two push buttons are added: the first is to show the live camera feed and the second is the screenshot button. When the show button is clicked a function called 'startcamera' is executed, when the screenshot button is clicked a function called 'take screenshot' is executed.


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

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.

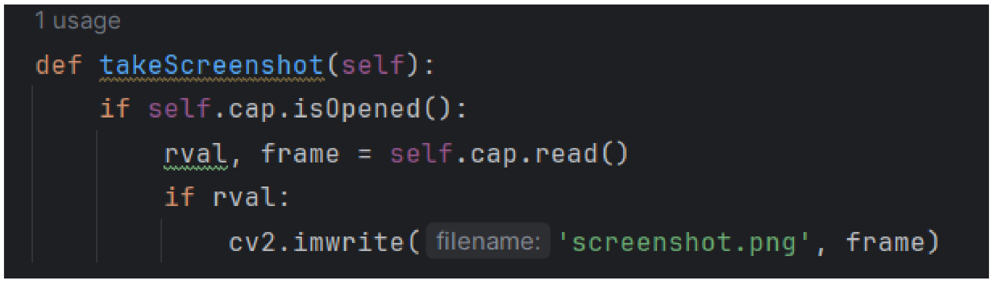

```
def startCamera(self):
    if not self.timer.isActive():
        self.timer.start(30)
```

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)
```

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.

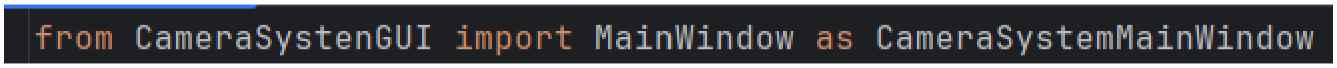


```
1 usage
def takeScreenshot(self):
    if self.cap.isOpened():
        rval, frame = self.cap.read()
        if rval:
            cv2.imwrite( filename: 'screenshot.png', frame)
```

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

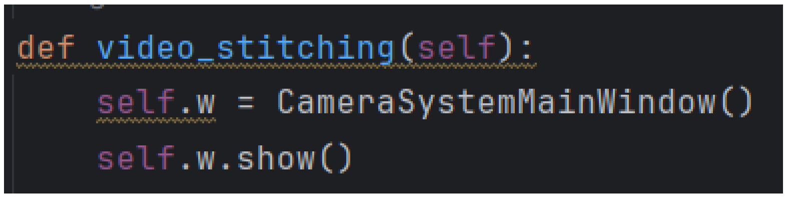
2.3.3 Video stitch

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



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

Importing a class from another file into the backend file.

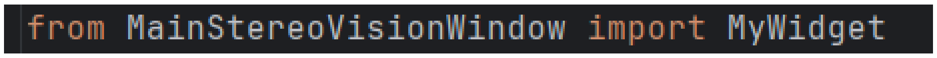


```
def video_stitching(self):
    self.w = CameraSystemMainWindow()
    self.w.show()
```

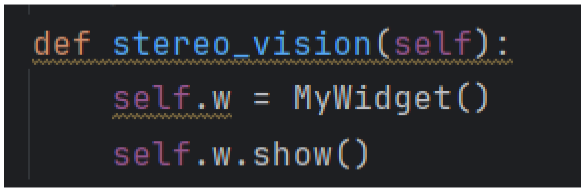
Showing an execution of this class.

2.3.4 stereo vision

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



```
from MainStereoVisionWindow import MyWidget
```

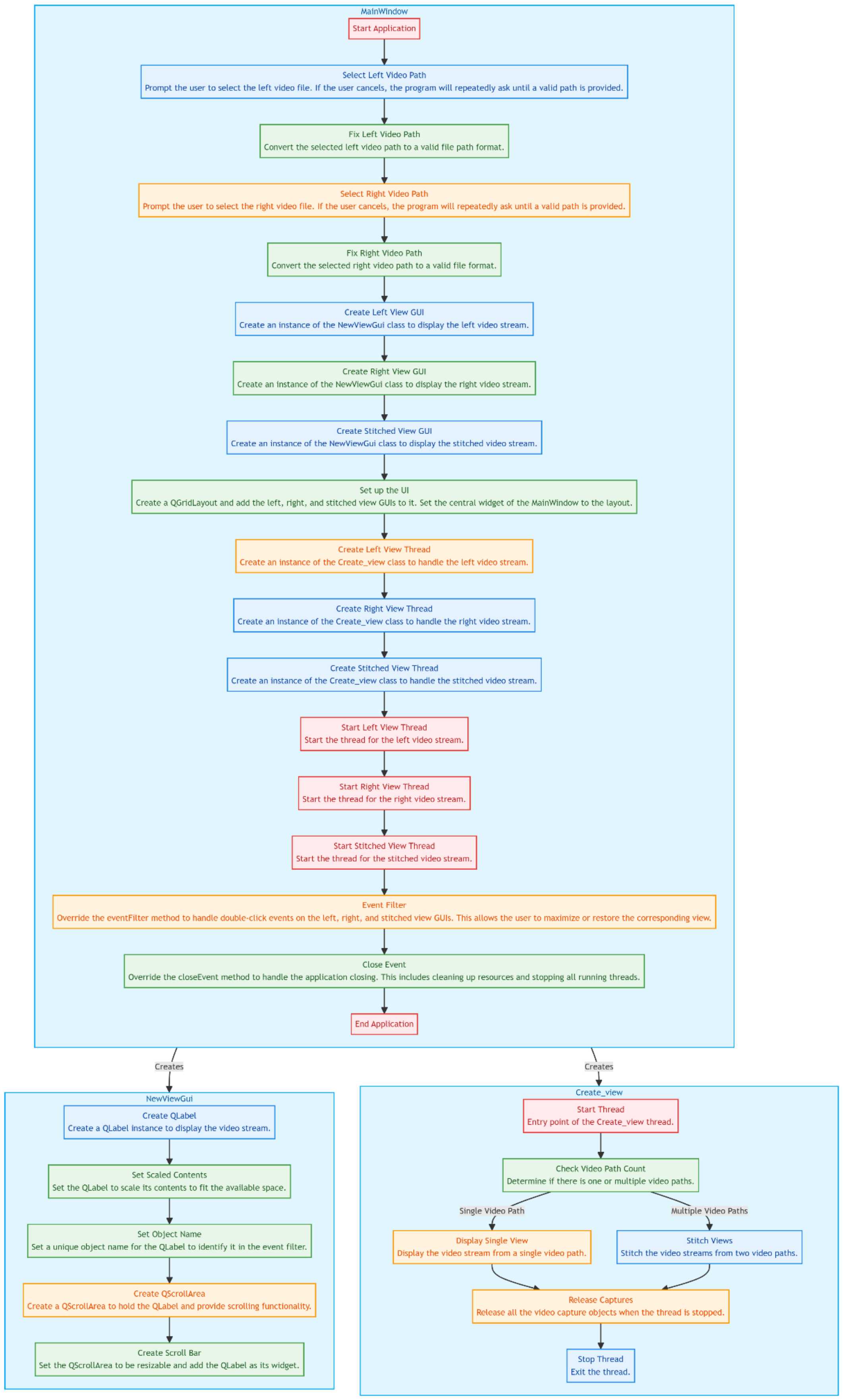


```
def stereo_vision(self):
    self.w = MyWidget()
    self.w.show()
```

3. Software Tasks

3.1 Flowchart

3.1.1 Video Stitching Flowchart



Start Application

Select Left Video Path
Prompt the user to select the left video file. If the user cancels, the program will repeatedly ask until a valid path is provided.

Fix Left Video Path
Convert the selected left video path to a valid file path format.

Select Right Video Path
Prompt the user to select the right video file. If the user cancels, the program will repeatedly ask until a valid path is provided.

Fix Right Video Path
Convert the selected right video path to a valid file format.

Create Left View GUI
Create an instance of the NewViewGui class to display the left video stream.

Create Right View GUI
Create an instance of the NewViewGui class to display the right video stream.

Create Stitched View GUI
Create an instance of the NewViewGui class to display the stitched video stream.

Set up the UI
Create a QGridLayout and add the left, right, and stitched view GUIs to it. Set the central widget of the MainWindow to the layout.

Create Left View Thread
Create an instance of the Create_view class to handle the left video stream.

Create Right View Thread
Create an instance of the Create_view class to handle the right video stream.

Create Stitched View Thread
Create an instance of the Create_view class to handle the stitched video stream.

Start Left View Thread
Start the thread for the left video stream.

Start Right View Thread
Start the thread for the right video stream.

Start Stitched View Thread
Start the thread for the stitched video stream.

Event Filter
Override the eventFilter method to handle double-click events on the left, right, and stitched view GUIs. This allows the user to maximize or restore the corresponding view.

Close Event
Override the closeEvent method to handle the application closing. This includes cleaning up resources and stopping all running threads.

End Application

Create QLabel
Create a QLabel instance to display the video stream.

Set Scaled Contents
Set the QLabel to scale its contents to fit the available space.

Set Object Name
Set a unique object name for the QLabel to identify it in the event filter.

Create QScrollArea
Create a QScrollArea to hold the QLabel and provide scrolling functionality.

Create Scroll Bar
Set the QScrollArea to be resizable and add the QLabel as its widget.

Start Thread
Entry point of the Create_view thread.

Check Video Path Count
Determine if there is one or multiple video paths.

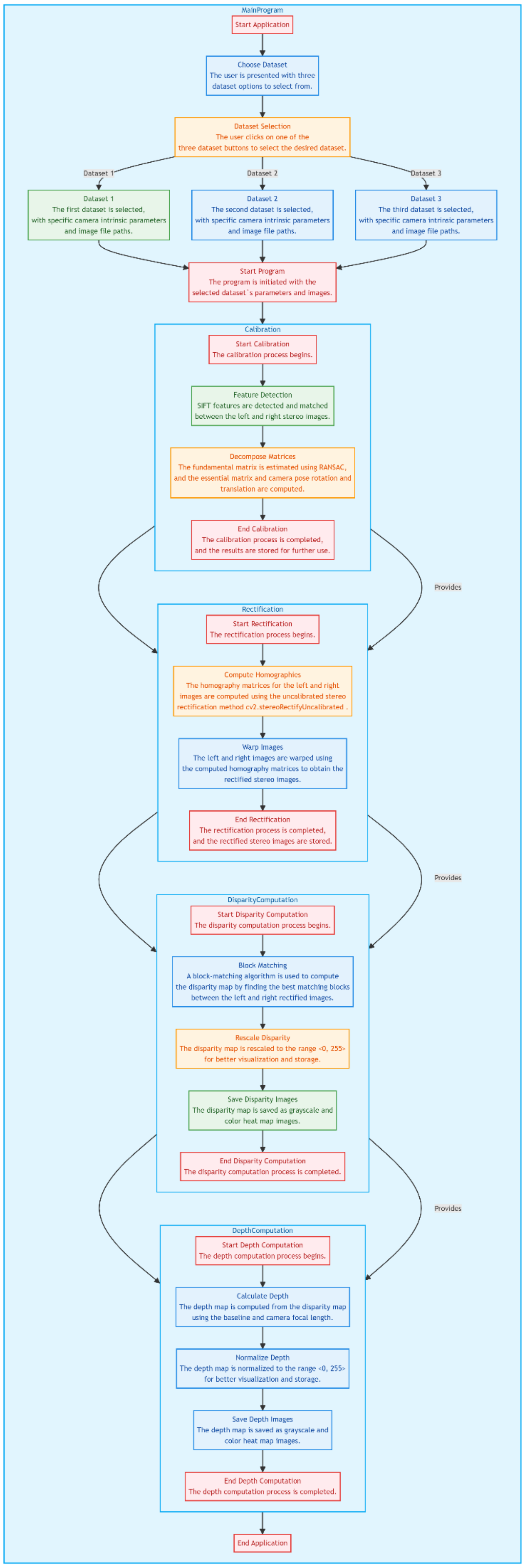
Display Single View
Display the video stream from a single video path.

Stitch Views
Stitch the video streams from two video paths.

Release Captures
Release all the video capture objects when the thread is stopped.

Stop Thread
Exit the thread.

3.1.2 Stereo Vision flowchart


MainProgram
Start Application
Choose Dataset The user is presented with three dataset options to select from.
Dataset Selection The user clicks on one of the three dataset buttons to select the desired dataset.
Dataset 1
Dataset 2
Dataset 3
Dataset 1 The first dataset is selected, with specific camera intrinsic parameters and image file paths.
Dataset 2 The second dataset is selected, with specific camera intrinsic parameters and image file paths.
Dataset 3 The third dataset is selected, with specific camera intrinsic parameters and image file paths.
Start Program The program is initiated with the selected dataset`s parameters and images.
Calibration
Start Calibration The calibration process begins.
Feature Detection SIFT features are detected and matched between the left and right stereo images.
Decompose Matrices The fundamental matrix is estimated using RANSAC, and the essential matrix and camera pose rotation and translation are computed.
End Calibration The calibration process is completed, and the results are stored for further use.
Provides
Rectification
Start Rectification The rectification process begins.
Compute Homographies The homography matrices for the left and right images are computed using the uncalibrated stereo rectification method cv2.stereoRectifyUncalibrated .
Warp Images The left and right images are warped using the computed homography matrices to obtain the rectified stereo images.
End Rectification The rectification process is completed, and the rectified stereo images are stored.
Provides
DisparityComputation
Start Disparity Computation The disparity computation process begins.
Block Matching A block-matching algorithm is used to compute the disparity map by finding the best matching blocks between the left and right rectified images.
Rescale Disparity The disparity map is rescaled to the range <0, 255> for better visualization and storage.
Save Disparity Images The disparity map is saved as grayscale and color heat map images.
End Disparity Computation The disparity computation process is completed.
Provides
DepthComputation
Start Depth Computation The depth computation process begins.
Calculate Depth The depth map is computed from the disparity map using the baseline and camera focal length.
Normalize Depth The depth map is normalized to the range <0, 255> for better visualization and storage.
Save Depth Images The depth map is saved as grayscale and color heat map images.
End Depth Computation The depth computation process is completed.
End Application

3.2 Main Window Features

3.3 Sub Windows Features

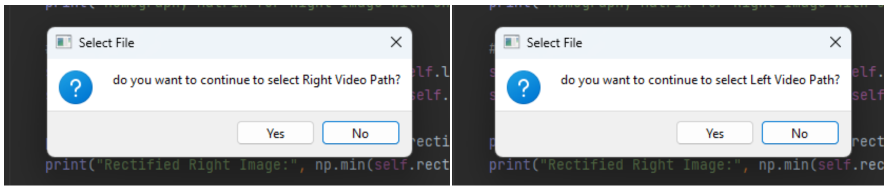
3.3.1 Video Stitching

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



Select File

do you want to continue to select Right Video Path?

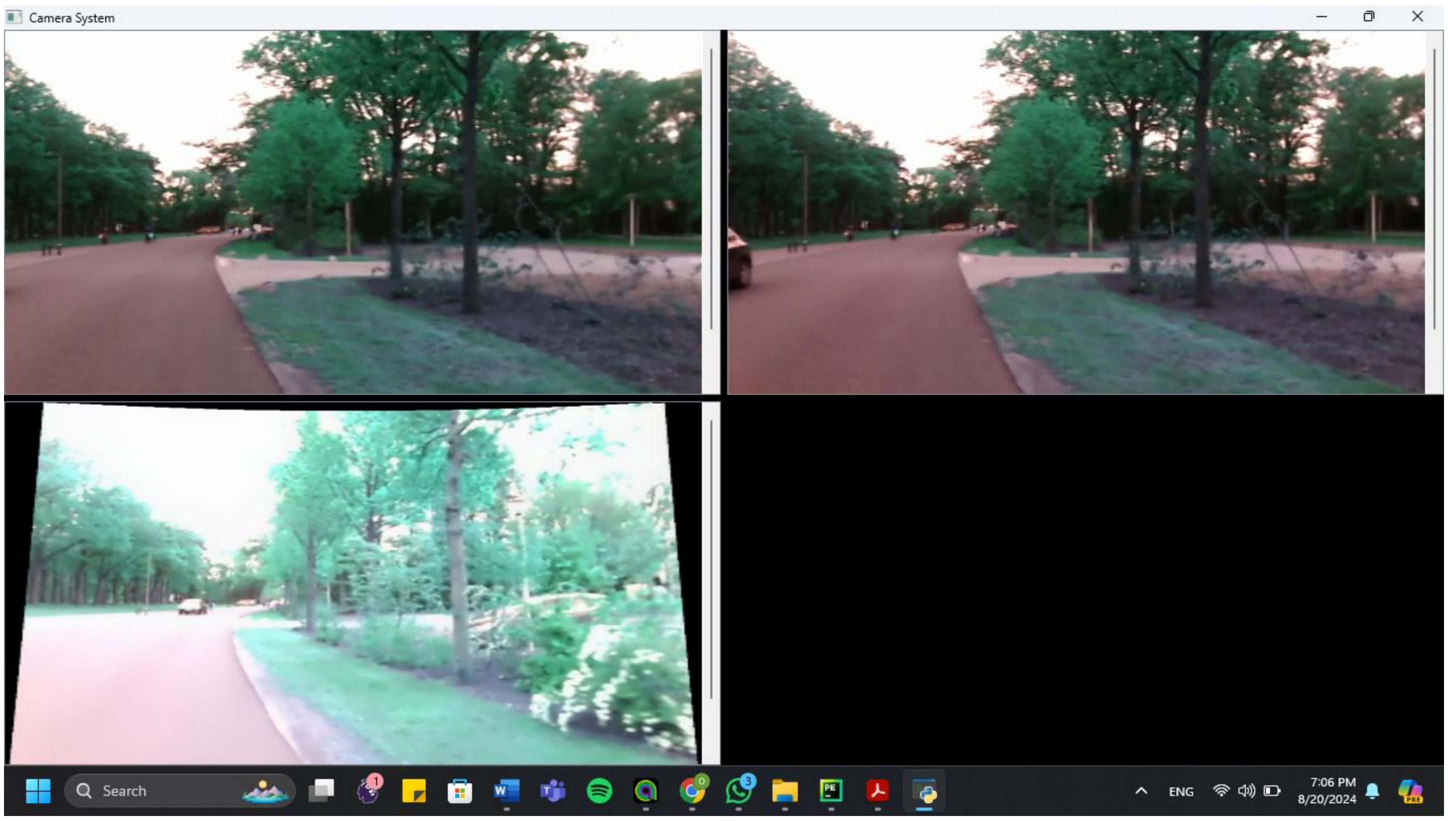
Select File

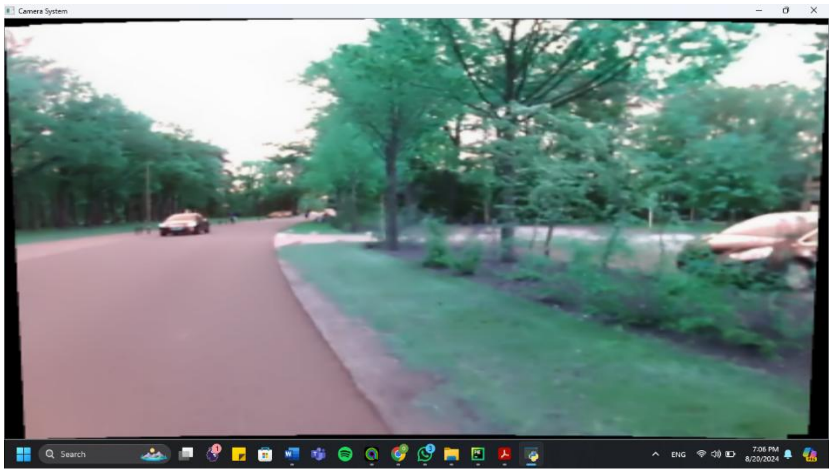
do you want to continue to select Left Video Path?

```
print("Rectified Right Image:", np.min(self.rect
```

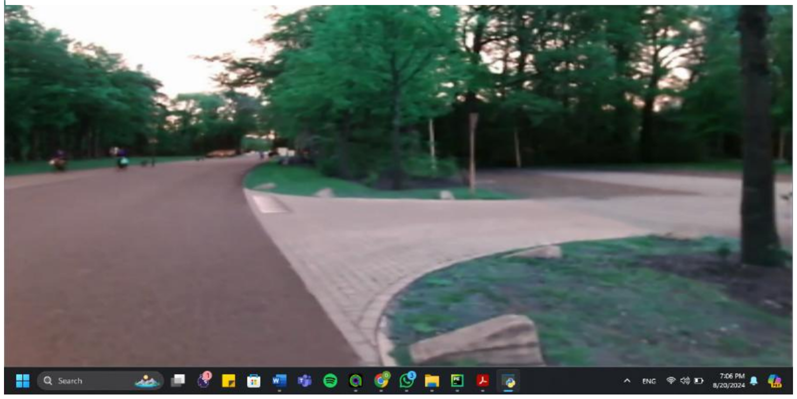
2. The Video Stitching window

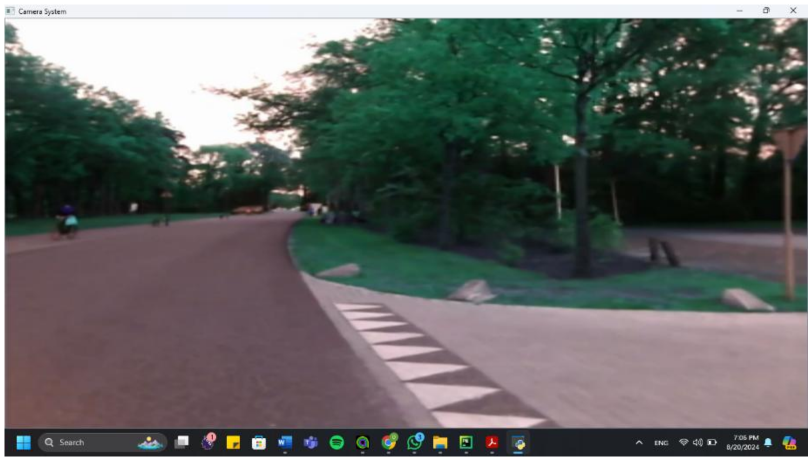
A video stitching window is then displayed containing the left view video and the right view video and the stitched view





Stitched view





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.

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. The Algorithm

3.1 The Frontend algorithm

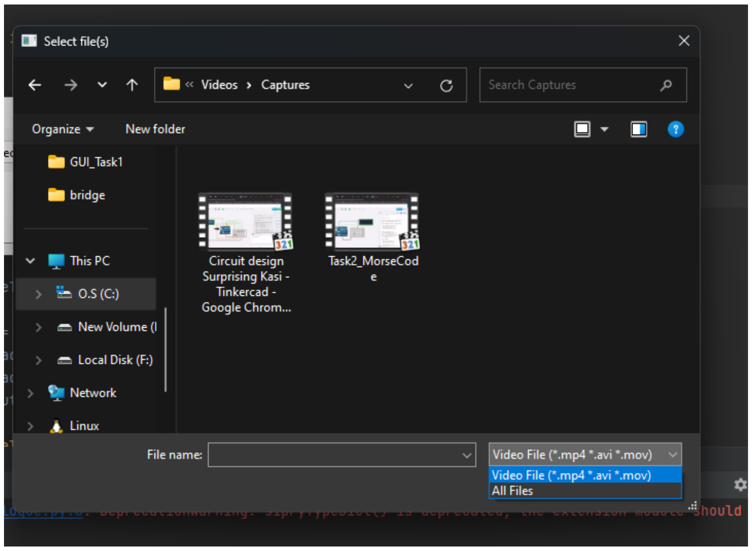
Program is started with file explorer window implemented in the main window (MainCameraSystemGUI) in an independent function (SelectVideopath) to be called twice for the left and the right views.

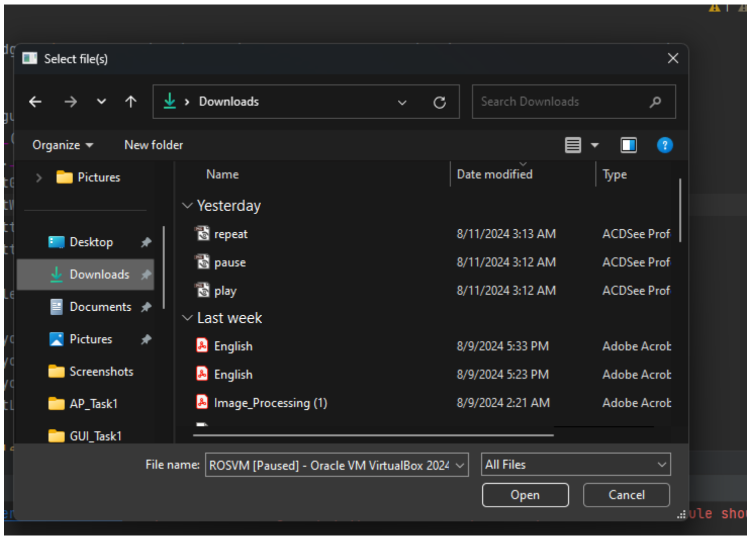
If the user hit cancel in the message box appeared Infront, the program won't shut displaying the same message box once more until the user choose a sufficient video path.

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



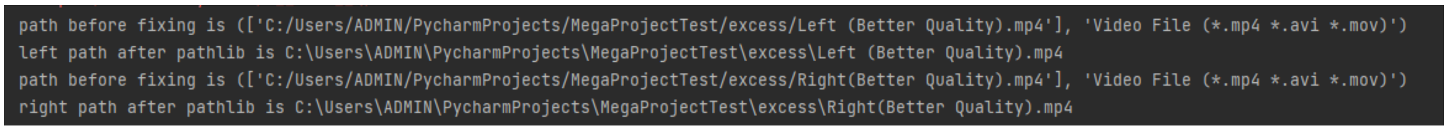

Select file(s)
Downloads
Search Downloads
Organize
New folder
Pictures
Desktop
Downloads
Documents
Pictures
Screenshots
AP_Task1
GUI_Task1
Name
Date modified
Type
Yesterday
repeat 8/11/2024 3:13 AM ACDSee Prof
pause 8/11/2024 3:12 AM ACDSee Prof
play 8/11/2024 3:12 AM ACDSee Prof
Last week
English 8/9/2024 5:33 PM Adobe Acrob
English 8/9/2024 5:23 PM Adobe Acrob
Image_Processing (1) 8/9/2024 2:21 AM Adobe Acrob
File name: ROSVM [Paused] - Oracle VM VirtualBox 2024
All Files
Open
Cancel

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

a. The path is passed into a fixing function using pathlib to remove any excess string attached to the path
b. 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.



```
path before fixing is (['C:/Users/ADMIN/PycharmProjects/MegaProjectTest/excess/Left (Better Quality).mp4'], 'Video File (*.mp4 *.avi *.mov)')
left path after pathlib is C:\Users\ADMIN\PycharmProjects\MegaProjectTest\excess\Left (Better Quality).mp4
path before fixing is (['C:/Users/ADMIN/PycharmProjects/MegaProjectTest/excess/Right(Better Quality).mp4'], 'Video File (*.mp4 *.avi *.mov)')
right path after pathlib is C:\Users\ADMIN\PycharmProjects\MegaProjectTest\excess\Right(Better Quality).mp4
```

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

a. 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
b. 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

3.2 The 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.

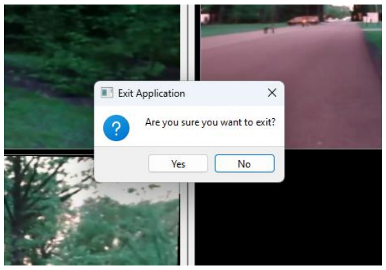
Unfortunately the library doesn't handle all stitching frames correctly , also It takes time for stitching , slower than the supposed smooth motion of the video capture, making the stitched view window lagging than the other windows , even after implementing a few method to speed up the stitching process 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.

3.3 The Closing 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



4. **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/

3.3.2 Stereo Vision

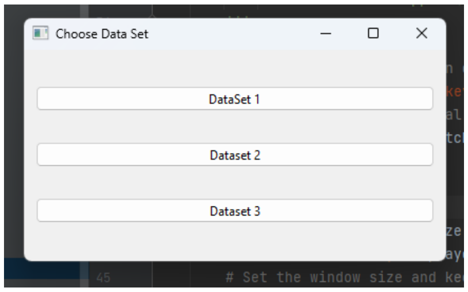
1. The initialization

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.



2. The Algorithm

2.1 The Main window

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

2.2 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

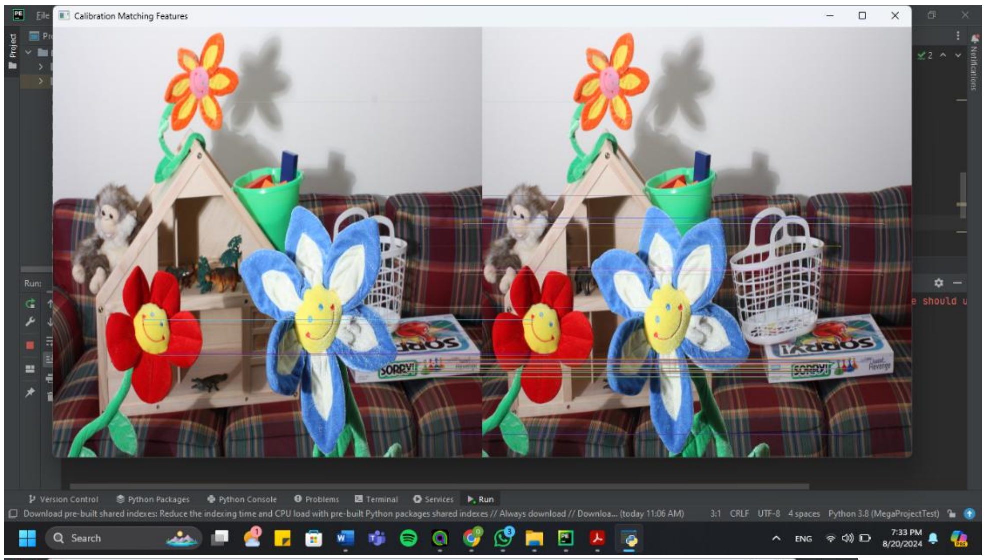
The Epipolar lines in the utility class is first detected using BFmatcher and the matches are saved in a list

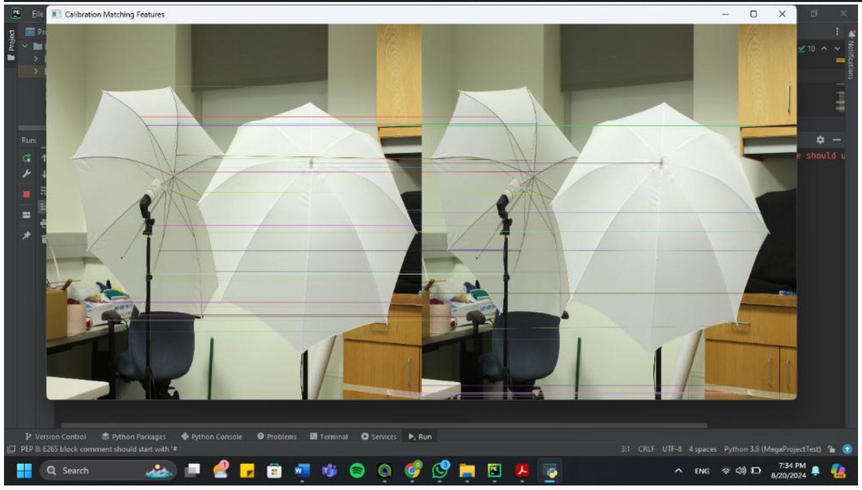
Then function ‘drawMatches’ is called by giving it the matched list, the image and the image_data. Another image then is outputted with the epipolar lines between all the matches is displayed.

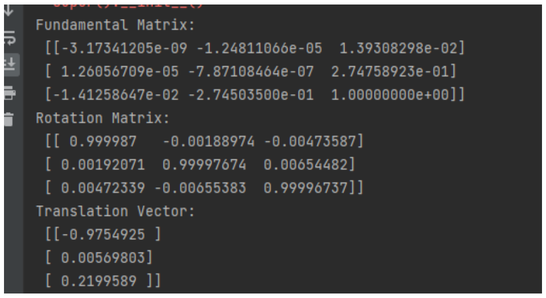
2.3 Calibration Algorithm

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







```
Fundamental Matrix:
 [[-3.17341205e-09 -1.24811066e-05  1.39308298e-02]
 [ 1.26056709e-05 -7.87108464e-07  2.74758923e-01]
 [-1.41258647e-02 -2.74503500e-01  1.00000000e+00]]
Rotation Matrix:
 [[ 0.999987   -0.00188974 -0.00473587]
 [ 0.00192071  0.99997674  0.00654482]
 [ 0.00472339 -0.00655383  0.99996737]]
Translation Vector:
 [[-0.9754925 ]
 [ 0.00569803]
 [ 0.2199589 ]]
```

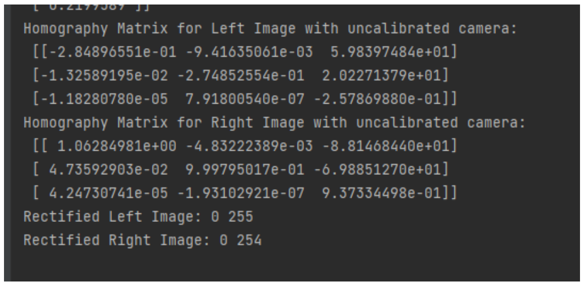
2.4 Rectification Algorithm

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).

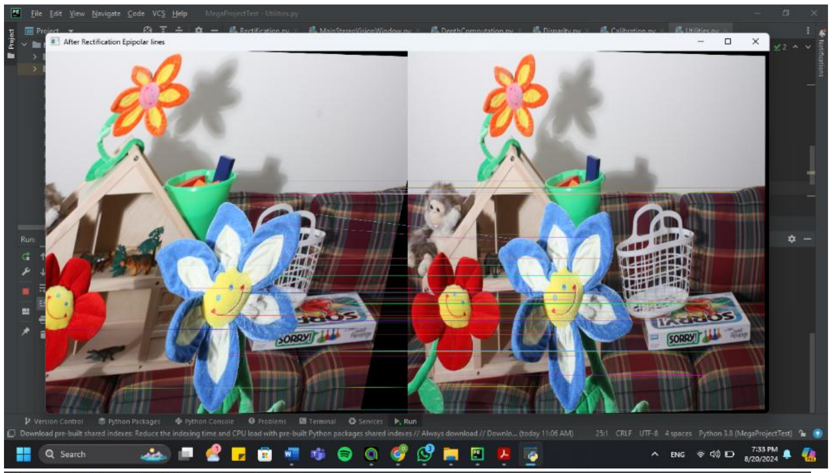
The function output is a set of homogonous matrices that are printed in the console. They are used to estimate the rectified images after that.

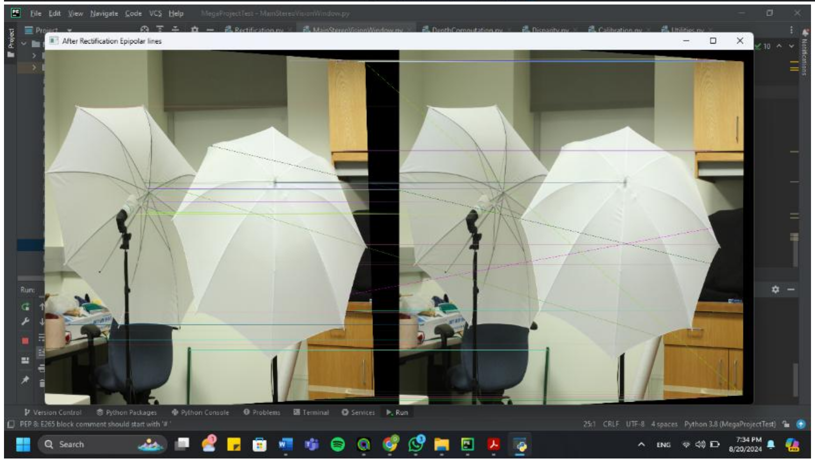


```
Homography Matrix for Left Image with uncalibrated camera:
 [[-2.84896551e-01 -9.41635061e-03  5.98397484e+01]
 [-1.32589195e-02 -2.74852554e-01  2.02271379e+01]
 [-1.18280780e-05  7.91800540e-07 -2.57869880e-01]]
Homography Matrix for Right Image with uncalibrated camera:
 [[ 1.06284981e+00 -4.83222389e-03 -8.81468440e+01]
 [ 4.73592903e-02  9.99795017e-01 -6.98851270e+01]
 [ 4.24730741e-05 -1.93102921e-07  9.37334498e-01]]
Rectified Left Image: 0 255
Rectified Right Image: 0 254
```

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.





2.5 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.

2.6 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. 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/