5AID0 Autonomous Vehicles Conquering the World

Final Lab Report

Quartile 2 - 2024-2025

Full Name	Student ID	Study
Janssen, Jelle	[redacted]	Computer Science and Engineering
[redacted] Vlad	[redacted]	Automotive Technology
[redacted] Luca	[redacted]	Automotive Technology
[redacted] Savvas	[redacted]	Automotive Technology
[redacted] Thomas	[redacted]	Automotive Technology
Wejbora, Tobias	[redacted]	Mechanical Engineering

Eindhoven, March 23, 2026

Contents

1 Introduction

In a world where humans have long vanished, and robots have become the custodians of Earth, history remains a fragile thread connecting the past to the present. This project envisions a scenario in the year 7984, where three autonomous robots embark on an extraordinary mission to recover lost fragments of human history from a fabled land known as the "Lost Continent," shown in Figure 1. Their objective is to locate and identify photographs of significant individuals from humanity's past and to ensure that the lessons of history are preserved to guide future decision making.

The robots operate under a guiding principle: "Those who cannot remember the past are condemned to repeat it." To achieve their goal, they traverse the Lost Continent equipped with sensors, cameras, and wireless communication technology. Their task is to locate and photograph images of notable historical figures, accompanied by their fellow explorer robots. The captured images are then transmitted wirelessly to their "Home Continent" - represented by a laptop - for further analysis.

For this project, the historical figures represented in the Lost Continent are the last four U.S. Presidents, whose leadership played pivotal roles in shaping human history. These photographs, mounted on stands within a designated exploration area, serve as relics of humanity's achievements and challenges. By identifying these individuals and accessing their corresponding Wikipedia entries, the robots aim to piece together critical lessons from the past.

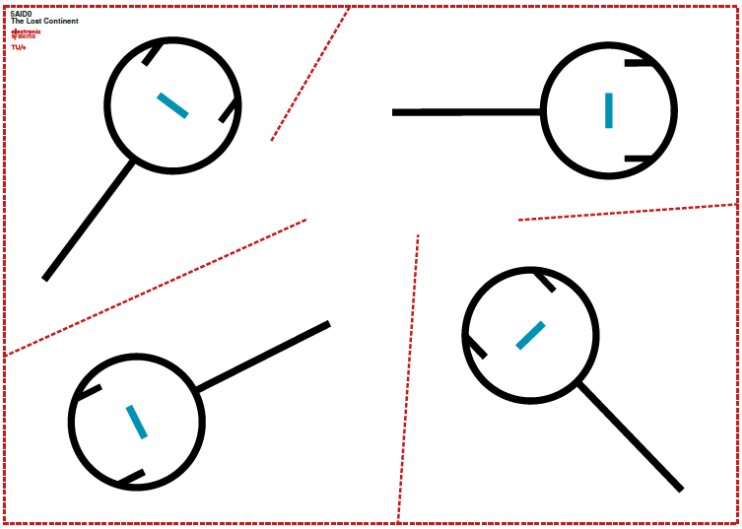

Figure 1: Lab 1 Track

To maintain the integrity and challenge of the mission, the robots must operate under strict rules:

- The robots may start at arbitrary locations on the map, even without their line-following sensors being initially aligned with a path.
- Adding additional sensors to the robots is not permitted.
- The robots must function autonomously, without manual control via the wireless network or any form of human intervention during the exploration process.

This report outlines the design choices and implementation of the robots' mission, detailing the exploration, image capture, identification, and data retrieval processes. Leveraging advanced image recognition techniques and Wikipedia integration, the project demonstrates a creative approach to preserving the legacy of human civilization, even in a post-human world ruled by intelligent machines.

2 Design Choices

2.1 Navigation

This section discusses the difficulties that the team encountered when implementing the navigation logic and the design choices the team has made based on those difficulties.

The first issue that arose was making left turns when there is a wall in front. Since the angles were not always the same, a solution had to be found where the robot could handle different angles. Multiple ideas were tested, one of which was measuring multiple angles with the ultrasound to find the angle of the turn and turning that amount. However since the ultrasound was not reliable enough this was not an option. So the team came up with the solution of always turning 90 degrees and then letting the correction system, which is used for standard navigation, correct for the inaccuracies.

The second issue was knowing which parking spot a robot is entering. This is needed since the robot needs to turn a different amount depending on which parking spot it is entering. This was solved by keeping track of which exits it has seen. If the robot sees the straight line exit first, it knows it entered in between a parking spot and the straight line. If the robot sees a parking spot first and then the straight line, it knows it entered in between the two parking spots. Based on this information we know in which parking spot each robot will end up and thus with what amount of degrees it has to turn to face the camera.

The third issue arose when trying to exit the room. First the idea was to let the robots drive towards the wall that was closest and then following this wall, however this did not work since the robot did not know at which angle it was approaching a wall and it could thus not align itself correctly. That is when the idea came to drive around the circle and exit when the distance to the wall on the right was smallest. This way the robot would immediately be aligned with the wall in the right direction.

2.2 Wireless Communication

The development of the wireless communication started with improving the script of lab 2. We did this by adding acknowledgments from the laptop after each column and to help with resending of columns the column numbers was also transmitted before each column. After that, the sub-team started integrating facial recognition into the communication code of the laptop. During this it became clear that the best solution was to combine the scripts into one big script and save the received image into a file. The file would be read and analyzed after receiving the image.

While integrating the picture-taking and sending code with the navigation code another problem arose. For some reason sending does not work using Serial when combined with the $ServoWithoutTimer5.h$ library, because of this the receiving of acknowledgments switched from $Serial.read()$ to using the register $UART0$. After integrating the code the sub-team started on robot-robot communication. During this another similar problem arose. All Serial functions did not work properly and were not sending and receiving data. Due to this, it was decided to create our own version of Serial, which would only have the necessary parts. This solution seemed to work and later on, the team expanded the code to look more like Serial. For robot-robot communication, it was decided to go for a simple approach where robots would send bytes of data to each other. Each byte was a specific message. To avoid conflict with the sending of frames the team chose to have the camera robot switch to unicast for the sending of frames.

2.3 Facial Recognition

When the camera bot enters the straight black line of a designated circle, it will stop and activate its camera to capture an image. Once the image is successfully captured and transmitted to the "Home continent", laptop, a Python-based algorithm will process the photograph to recognize the facial characteristics of the depicted celebrity.

The algorithm will compare the captured image with a predefined presidential database of facial characteristics created by the group. By analyzing and matching facial features, the system will identify the celebrity. Upon successful identification, the program will automatically open the corresponding Wikipedia page to display detailed information about the recognized celebrity.

This process ensures an efficient and accurate method for identifying the individuals depicted in the photographs and provides contextual historical knowledge to the robots for their mission.

3 Implementation

3.1 Navigation

The navigation works by following the right wall, this way the robot will enter every room and sense every line. This is done by measuring the distance in front first, then driving 50% of this distance and continuously scanning right. In this way, it is ensured that the robot does not drive into the wall in front, and it is able to correct the distance to the right wall. If the distance to the right wall is suddenly very large it knows it has reached the end of the wall and it has to do a 180 degree turn around the wall.

Because of the way the robots navigate around the rooms, they will always enter the line on the circle. In 3 out of the 4 circles it enters the circle with the straight line on the left and the first parking spot on the right, in the other case it enters in between the 2 parking places. Therefore the camera robot was programmed to drive clockwise and the explorer robots were programmed to drive counterclockwise. To be able to have no collisions the team did make the assumption that the unique case would be the first room the robots would enter and the camera robot would enter first. When the camera robot sees the line, it will find its exit and drive to the end of the line for the picture. When an explorer robot finds the line, it will skip the first parking spot and enter the second. Since the second robot enters the line at the same location, this ensures that the second robot can enter the first parking spot without collisions.

To exit the line the robots will drive around on the circle and when the distance towards the right is smaller than the set threshold, it will switch into a state to exit the room. This way it is ensured that the robots are aligned with the wall and they are not driving around in the middle of the room. When the distance to the right is suddenly very large, it knows that it has to do a 180 degree turn around the wall just like in the standard navigation, however in this case it also means the robot had found a new room and then switches to the first state where it is also looking for a line.

3.2 Wireless Communication

The communication is split into two parts: robot-robot communication and robot-laptop communication. The robots use a custom simplified version of the HardwareSerial.h library. Our version works by reading and writing to the registers of the Arduino and uses the built-in ISR interrupt for receiving data. Just like the hardware serial it also uses a buffer for receiving data. However, our version can only transmit bytes and strings. The code on the laptop uses Python and the Serial library to transmit and receive data.

The robot-to-robot communication is implemented as follows. When one of the non-camera robots reaches a spot to take a picture they will transmit a specific byte to indicate that they are ready to take a picture. When the first robot reaches the desired spot it will send a byte with the value 0xAB and when the second robot reaches its spot it will send a byte with the value 0xAC. When the camera robot reaches the end of the straight line and turns the robot will check if it has received the two bytes from the other robots. If the camera robot has not received those bytes it will wait until it has received them. Once the camera robot receives the bytes it will start taking a picture and sending the data to the laptop. When the camera robot is finished taking the picture it will send a byte with the value 0xFA to the other robots to indicate that the picture is taken and that they can go back to searching for lines.

Robot-to-laptop communication is only used for sending pictures. When the camera robot wants to send a picture it will first change its casting mode from broadcast to unicast. Then when it is done sending a picture the casting mode will be set back to broadcast. The sending of the picture starts by sending the string "VSY" over the network to signal that sending has started. After that the robot will start taking a picture, but will only save the first 10 columns. Then the robot will send those 10 columns to the laptop. This will repeat itself until the whole image is transmitted. A column is sent by first sending the string "COL" and then the number of the column. After that the actual data is sent. The data is sent in chunks of 64 bytes. Each chuck starts with the byte 0xAA and the length of the chunk. After sending all the data the robot will wait for an acknowledgment from the laptop that the row has been received. If the acknowledgment is not received after one second the robot will retransmit the same row again. This will repeat itself for the next 10 rows and until the whole image is transmitted. When a picture is taken for the first time the left part of the picture can be tinted white and or green, due to the camera still adjusting. To avoid this the robot is start the image-sending process over again after the first 80 columns are sent.

3.3 Facial Recognition

This section discusses the face recognition process used in a Python program to identify celebrities. When triggered, the system is tasked with recognizing the person in the image and displaying their information from Wikipedia. To achieve this, the program utilizes Python libraries such as `face_recognition` and `webbrowser`.

The first step involves building a database, as seen in Figure 2 of known celebrities by providing their images to the program. The `face_recognition` library is used to analyze the facial features of the celebrities. These features are stored as unique face encodings, which are numerical representations of the face's characteristics. The program then compares any new image against the stored encodings to identify the person in the picture.

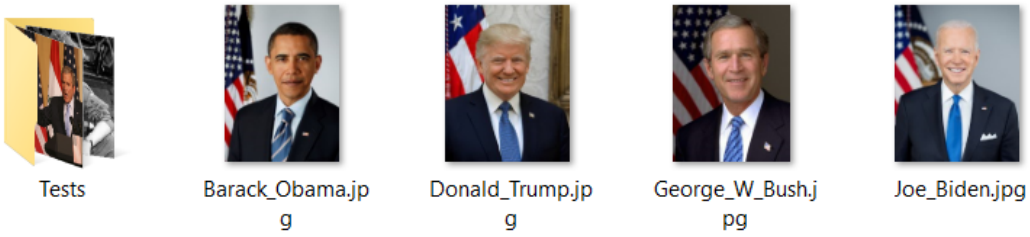

Figure 2: Presidential Database

When the system is ready to recognize a celebrity, it uses the `face_recognition.face_encodings()` function to detect faces in the input image. The program extracts the facial features from the detected face and compares them to the database of known celebrities using the `face_recognition.compare_faces()` function. If a match is found, the program identifies the celebrity by `Name`, as seen in Figure 3 and Figure 4.

Figure 3: Face recognition algorithm on very old picture of George W. Bush

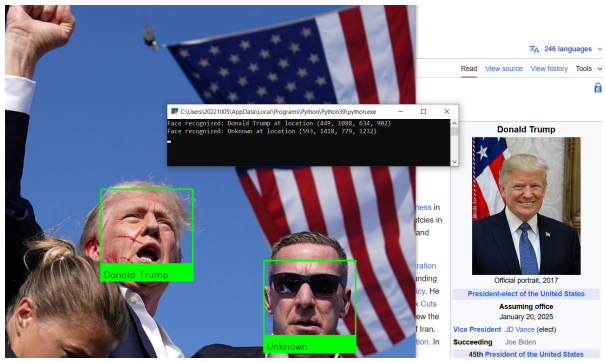

Figure 4: Face recognition algorithm on picture of Donald Trump with face encodings

To display additional information, the program then opens the celebrity's Wikipedia page using the `webbrowser.open_new_tab()` function. The URL for the Wikipedia page is dynamically constructed based on the recognized celebrity's name (for example, `https://en.wikipedia.org/wiki/Barack_Obama`). If no match is found, the program will output `"No faces recognized"`, and `"Unknown"` for any additional unrecognized people in the picture.

This process involves capturing an image, detecting and comparing facial features, and then accessing a Wikipedia page if a match is found.

4 Results

The results presented during the final video will be discussed within this section in detail; breaking down how the individual components performed.

4.1 Navigation

Regarding the navigation of the robots, only one quadrant of the continent was traversed. Furthermore, due to one of the explorer robots being faulty hardware wise, only the camera robot and one explorer robot were used during the demonstration. Nevertheless, the robots performed as expected where they were placed hugging a wall. Once initialized, the ultrasonic sensor detected the surrounding walls and the robots traversed the continent counterclockwise until the infrared sensors detected a pathway to the celebrities. Once the camera robot detected a line, it rotated counterclockwise, following the circular line. The camera robot accurately detected the straight line and positioned itself perpendicular at the end of the line. The explorer robot also accurately detected the circular line rotating clockwise. The explorer robot was able to track the number of parking spaces well, and it avoided the first one that it encountered. Once at the correct parking space, the explorer robot faced the camera robot as intended.

4.2 Wireless Communication

The results of wireless communication at the end of the project were promising. A solid connection between the laptop, camera robot, and explorer robot was obtained. This allowed for the navigation sequence as described above and the facial recognition algorithm described below. More specifically, the sending of packets of information between the robots was seamless and reliable which allowed for the robots to communicate accurately when they were in position. This then allowed for the image to be captured and transmitted from the camera robot to the laptop. The image transmission took approximately 90 seconds.

4.3 Facial Recognition

Once the picture was transmitted, the facial recognition algorithm was already running in the background, ensuring an extremely fast identification process once the image had been saved. The algorithm successfully recognised all the presidents, even under challenging conditions. It accurately identified faces from different angles and was able to handle obstructions like glasses or variations in key facial features. Additionally, it showed remarkable resilience in recognising altered images, including those with colour modifications and representations of the presidents at a much younger age. After the identification process, the Wikipedia page of given president was always opened.

5 Evaluation of Results

5.1 Navigation

As part of the navigation testing, the goal was to have the robot enter a room, detect the black line and move into either position 1, the first small line encountered, position 2, the second small line encountered, or position 3, the big line used for positioning the Camera Robot. This was experimented and found to be working as expected for the non-camera robots. After positioning, the robots individually successfully exited the room as planned. However, when testing the Camera Robot's ability to enter the line, some significant difficulties were encountered. While it did enter the line most of the time, its navigation was inconsistent, likely due to the hardware issues with the robot. This led to additional troubleshooting efforts and delays.

5.2 Wireless Communication

The wireless communication system faced some challenges, primarily due to a faulty xBee shield, which had affected the coordination between the three robots. Despite this, successful implementation of communication between the Explorer robots and the Camera Robot has been achieved. When the normal robots entered position 1 and 2, they would send a signal to the Camera Robot, allowing it to recognize that they were in place and enabling it to proceed with capturing the image. Before the final presentation, an experiment has been conducted where an Explorer Robot was put on the circle and where it navigated to its correct position. It would then send out the message and this has been received correctly.

5.3 Facial Recognition

The facial recognition system was tested by having the Camera Robot take pictures from different angles and then processing these images for identification. Despite variations in positioning, the system consistently recognized the faces and correctly matched them. After identification, the computer successfully opened the corresponding Wikipedia page for the detected face. This demonstrated that the facial recognition system works.

5.4 Final Product

In the final testing phase, the Camera Robot was repaired on the last day and due to the limited time available, there was not enough opportunity to fully integrate and test all the adjustments. As a result, instead of successfully navigating through all four rooms as initially planned, the system was only able to enter one room with one Explorer Robot and the Camera Robot.

5.5 Potential Improvements

Many of the issues encountered could have potentially been avoided if the equipment had functioned from the start. The malfunctioning xBee shield and the broken Camera Robot were major setbacks that significantly reduced the available testing time. If the hardware had been working properly, more focus could have been placed on refining the navigation, communication, and integration between components earlier in the process.

5.6 Conclusion

Although individual components performed well, the final integration was impacted by hardware failures and time constraints. The last-minute repair of the Camera Robot left too little time to fully adjust the code, which resulted in the system only navigating one room instead of four. Despite this, key functionalities worked correctly. Future improvements should focus on earlier hardware verification, better contingency planning for unexpected failures, and more extensive system-wide testing to ensure full integration and functionality within the given timeframe.

6 Individual contributions

[redacted]

During the whole project I mainly focused on the navigation of the robot. So I contributed on the line following system, wall navigation and positioning of the robot. Together with Vlad and Thomas, we came up multiple solutions for problems related to navigation due to which the robot was reliably getting into its position.

[redacted]

Having focused on movement and navigation of the robot during labs 1 and 3, I continued working on this part of the systenm for the final project. I was mainly involved with the movement of the robot inside the continent, how it detects the black line then follows it until it gets into the correct position. Finally, I was highly involved during final testing, when all parts of the project were combined. Due to technical issues during the last two weeks before the presentation, a lot of time was spent trying to discover the problem.

[redacted]

During the development of our system, I mainly worked on communication. During the first couple of weeks, I worked on creating a reliable program that would take and send the picture to the laptop. I also worked on the laptop's receiving code. When the code worked on its own I integrated it with the other code and debugged where necessary. After this, I worked on inter-robot communication and integrated that with the other code. Lastly, I helped the navigation team with debugging their code and fixing problems.

[redacted]

Throughout this final lab, my primary focus was on facial recognition, which involved developing the algorithm and writing all the related sections in the final report. Additionally, I collaborated with Luca on the wireless communication, ensuring a successful image transmission. In the end I helped the team with combining the whole code.

[redacted]

During the final assignment, I focused on the navigation of the robots and making the maze. Besides that, I was focused on testing the abilities of the robot.

Tobias Wejbora

During the final project, I began by researching into possible options for facial recognition. A basic script was made as a proof of concept which allowed another team member to create a fully working script for the final project. Furthermore, I worked on finding solutions to the many problems encountered while combining all features from the labs. This mainly included adapting the navigation strategy.

A Navigation

Arduino navigation code is too extensive to provide within the report.

B Wireless Communication

```
import sys
import os
import ctypes
from sys import platform
import serial
import numpy as np
if platform == "win32":
    os.environ["PYSDL2_DLL_PATH"] = "."
from sdl2 import *
import sdl2.ext
import time
from PIL import Image
import face_recognition as fr
import cv2
import webbrowser

# CONFIGURE ME! (Windows)
# Set the correct COM port used by the USB connection to the CameraBot
SERIAL_DEVICE = 'COM12'

# CONFIGURE ME! (Linux/Mac)
# SERIAL_DEVICE = '/dev/ttyACM0'

# Set the baud rate used by the serial connection
# SERIAL_BAUD_RATE = 1000000
SERIAL_BAUD_RATE = 57600

# QVGA image size settings
WIDTH = 240
HEIGHT = 320

# Short aliases for face_recognition functions
load_image = fr.load_image_file
get_encodings = fr.face_encodings
get_locations = fr.face_locations
compare_faces = fr.compare_faces
face_distance = fr.face_distance

# Load known face images and generate encodings
known_faces = {
    "Barack Obama": load_image(r".\Images\Presidents\Barack_Obama.png"),
    "Donald Trump": load_image(r".\Images\Presidents\Donald_Trump.png"),
    "George W. Bush": load_image(r".\Images\Presidents\George_W_Bush.png"),
    "Joe Biden": load_image(r".\Images\Presidents\Joe_Biden.png")
}

known_face_encodings = [get_encodings(img)[0] for img in known_faces.values()]
known_face_names = list(known_faces.keys())


def face_recognition():
    # Load input image
    input_image_path = r".\Images\output.png"
    input_image = load_image(input_image_path)

    # Detect faces and generate encodings
    input_face_locations = get_locations(input_image)
    input_face_encodings = get_encodings(input_image, input_face_locations)

    # Compare faces and check for matches
    any_match_found = False

    for face_encoding in input_face_encodings:
        matches = compare_faces(known_face_encodings, face_encoding)
        distances = face_distance(known_face_encodings, face_encoding)
        best_match_index = np.argmin(distances)
```

```
            if matches[best_match_index]:
                name = known_face_names[best_match_index]
                print(f"Match found: {name}")
                any_match_found = True

                # Open corresponding Wikipedia page
                wikipedia_url = f"https://en.wikipedia.org/wiki/{name.replace(' ', '_')}"
                webbrowser.open_new_tab(wikipedia_url)

        if not any_match_found:
            print("Not recognised")


def yuv2rgb(y, u, v):
    """Convert YUV color space values to to RGB values.
    In the yuv422 color space, 2 pixels are encoded in 4 bytes as follows:
    <Y1><U><Y2><V>
    first pixel <Y0><U><V>
    second pixel <Y1><U><V>
    see <https://paulbourke.net/dataformats/yuv/>
    """

    return (
        np.clip((298 * (y-16) + 409 * (v-128) + 128 ) >> 8, 0, 255),
        np.clip((298 * (y-16) - 100 * (u-128) - 208 * (v-128) + 128 ) >> 8, 0, 255),
        np.clip((298 * (y-16) + 516 * (u-128) + 128 ) >> 8, 0, 255)
    )

def sync(serial_conn, first=True):
    """ Wait for frame synchronization input """
    # check if we are unexpectedly skipping data
    dropped_bytes = 0

    waiting = True
    while waiting:
        # check for the anticipated sequence
        if serial_conn.read() == b'V':
            if serial_conn.read() == b'S':
                if serial_conn.read() == b'Y':
                    waiting = False
        else:
            # we got something unexpected (unless it is the first frame)
            dropped_bytes = dropped_bytes + 1
    # only on the first frame we expect to drop some data
    if not first:
        if dropped_bytes > 0:
            print(f"Warning: dropped bytes: {dropped_bytes}")

def setupSDL():
    """ Create the SDL window """
    SDL_Init(SDL_INIT_EVERYTHING)
    window = SDL_CreateWindow(b"Frame Viewer",
                    SDL_WINDOWPOS_CENTERED, SDL_WINDOWPOS_CENTERED,
                    WIDTH, HEIGHT, SDL_WINDOW_SHOWN)
    window_surface = SDL_GetWindowSurface(window)
    frame = SDL_CreateRGBSurface(0, WIDTH, HEIGHT, 32, 0x0000ff, 0x00ff00, 0xff0000, 0)
    pixels = sdl2.ext.pixels2d(frame.contents) # used to access pixels from passed source
        directly
    SDL_BlitSurface(frame, None, window_surface, None) # copy the frame to the window
        surface
    SDL_UpdateWindowSurface(window) # copy the window surface to the screen
    event = SDL_Event()

    return [pixels, frame, window_surface, window, event]

def updateSDLWindow(frame, window_surface, window, event):
    """ Update the window with new image data """
```

```
    # copy frame to the window surface
    SDL_BlitSurface(frame, None, window_surface, None)
    # copy the window surface to the screen
    SDL_UpdateWindowSurface(window)
    # check for any quit events
    while SDL_PollEvent(ctypes.byref(event)) != 0:
        if event.type == SDL_QUIT:
            return False
    return True

def cleanupSDL(frame, window):
    """ Close the window """
    SDL_FreeSurface(frame)
    SDL_DestroyWindow(window)
    SDL_Quit()

def waitForStartFrame(serial_connection):
    """ Read data from the serial port """
    # print("Waiting for SOF\n")
    while (True):
        d = serial_connection.read()
        if d == b'\xAA':
            break
    # print("Start of frame found")

    dataSize = int.from_bytes(serial_connection.read(), "big")

    # print(dataSize)

    return dataSize

def main():

    print(" +----------------------------------------+")
    print(" | XBee receive code|")
    print(" +----------------------------------------+\n")

    # Open the serial connection
    try:
        serial_connection = serial.Serial(port=SERIAL_DEVICE, baudrate=SERIAL_BAUD_RATE)
    except serial.SerialException as e:
        print(f"Exception: {e}")
        print("Serial connection not established. \nCheck port number and baudrate.")
        return

    # Setup the SDL window
    (pixels, frame, window_surface, window, event) = setupSDL()

    # create array to store the image
    bgr_image = np.zeros((HEIGHT, WIDTH, 3), dtype=np.uint8)

    # Start receiving image frames.

    # wait until the sync key is received via the serial port
    print("Syncing...\n")
    sync(serial_connection, first=True)
    print("Synced!\n")



    # Capture frames
    continue_running = True
    while continue_running:
        # for every frame column
        x = 0
        while x < WIDTH:
```

```
        waiting = True
        while waiting:
             # check for the anticipated sequence
            if serial_connection.read() == b'C':
                if serial_connection.read() == b'O':
                    if serial_connection.read() == b'L':
                        waiting = False


        col  = int.from_bytes(serial_connection.read(2), "big")

        if(col != x):
            if(col < x):
                print(f"Warning: ack lost. received column {col} twice")
            else:
                print(f"Error: column mismatch {col} != {x}")
            x = col

        y = HEIGHT - 1
        while y > 0:

            dataSize = waitForStartFrame(serial_connection)

            buf = serial_connection.read(dataSize)

            if(len(buf) == dataSize):
                i = 0
                while i + 3 < len(buf):
                    # convert pixel 1 YUV to RGB
                    (r,g,b) = yuv2rgb(buf[i+0], buf[i+1], buf[i+3])
                    # set the pixel data in the SDL window
                    pixels[x][y] = (r<<16) | (g<<8) | (b<<0)
                    # save it for the image
                    bgr_image[y, x] = (b, g, r)

                    # convert pixel 2 YUV to RGB
                    (r,g,b) = yuv2rgb(buf[i+2], buf[i+1], buf[i+3])
                    # set the pixel data in the SDL window
                    pixels[x][y-1] = (r<<16) | (g<<8) | (b<<0)
                    # save it for the image
                    bgr_image[y-1, x] = (b, g, r)

                    # update y and i
                    y -= 2
                    i += 4
            else:
                print("Error: data size mismatch")

        # send ack
        serial_connection.write(b'\x55')


        # update the window (once for every column of new pixels)
        continue_running = continue_running and updateSDLWindow(frame, window_surface,
            window, event)

        if not continue_running:
            break

        #print("done\n");
        x += 1

    # Save the image
    img = Image.fromarray(bgr_image, 'RGB')
    img.save("./Images/output.png")
    print("Image saved")

    # Perform face recognition
```

```
        face_recognition()

        if not continue_running:
            break

        # wait until the sync key is received for the next frame via the serial port
        sync(serial_connection, first=False)


    # close the window
    cleanupSDL(frame, window)

    # close the serial connection
    serial_connection.close()

    return 0

if __name__ == "__main__":
    sys.exit(main())
```

Listing 1: Python Code for wireless communication

C Face Recognition

```
import os
import numpy as np
import face_recognition as fr
import cv2
import webbrowser

def load_known_faces(known_face_paths):
    """Load known face images and compute encodings."""
    known_face_encodings = []
    known_face_names = []

    for name, path in known_face_paths.items():
        if not os.path.exists(path):
            print(f"Warning: File not found for {name}: {path}")
            continue
        image = fr.load_image_file(path)
        encodings = fr.face_encodings(image)
        if encodings:
            known_face_encodings.append(encodings[0])
            known_face_names.append(name)
        else:
            print(f"Warning: No faces found in image for {name}: {path}")

    return known_face_encodings, known_face_names

def recognize_faces(input_image_path, known_face_encodings, known_face_names):
    """Recognize faces in the input image and return matched names."""
    if not os.path.exists(input_image_path):
        print(f"Error: Input file not found: {input_image_path}")
        return None, None

    input_image = fr.load_image_file(input_image_path)
    face_locations = fr.face_locations(input_image)
    face_encodings = fr.face_encodings(input_image, face_locations)
    recognized_faces = []

    for face_encoding in face_encodings:
        matches = fr.compare_faces(known_face_encodings, face_encoding)
        distances = fr.face_distance(known_face_encodings, face_encoding)
        best_match_index = np.argmin(distances) if distances.size > 0 else None

        if best_match_index is not None and matches[best_match_index]:
            recognized_faces.append(known_face_names[best_match_index])
        else:
            recognized_faces.append("Unknown")

    return face_locations, recognized_faces, input_image

def draw_faces(image, face_locations, recognized_faces):
    """Draw rectangles around detected faces and annotate names."""
    # Convert the image to BGR for OpenCV (original is RGB)
    image_bgr = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

    for (top, right, bottom, left), name in zip(face_locations, recognized_faces):
        # Draw a rectangle around the face
        cv2.rectangle(image_bgr, (left, top), (right, bottom), (0, 255, 0), 2)

        # Draw a label with the name below the face
        cv2.rectangle(image_bgr, (left, bottom - 35), (right, bottom), (0, 255, 0), cv2.
            FILLED)
        cv2.putText(image_bgr, name, (left + 6, bottom - 6), cv2.FONT_HERSHEY_SIMPLEX, 0.6,
            (0, 0, 0), 1)

    return image_bgr

# Define paths for known faces
```

```
known_faces_paths = {
    "Barack Obama": r"Presidents\Barack_Obama.jpg",
    "Donald Trump": r"Presidents\Donald_Trump.jpg",
    "George W. Bush": r"Presidents\George_W_Bush.jpg",
    "Joe Biden": r"Presidents\Joe_Biden.jpg"
}

# Load known faces
known_face_encodings, known_face_names = load_known_faces(known_faces_paths)

# Input image path
input_image_path = r"Presidents\Tests\bush_test3.jpg"

# Recognize faces
face_locations, recognized_faces, input_image = recognize_faces(input_image_path,
    known_face_encodings, known_face_names)

if recognized_faces:
    for face, location in zip(recognized_faces, face_locations):
        print(f"Face recognized: {face} at location {location}")
        if face != "Unknown":
            wikipedia_url = f"https://en.wikipedia.org/wiki/{face.replace(' ', '_')}"
            webbrowser.open_new_tab(wikipedia_url)
else:
    print("No faces recognized.")

# Draw faces on the image
if face_locations is not None and input_image is not None:
    output_image = draw_faces(input_image, face_locations, recognized_faces)

    # Display the image in a window
    cv2.imshow("Recognized Faces", output_image)

    # Wait for a key press and close the window
    cv2.waitKey(0)
    cv2.destroyAllWindows()
else:
    print("No faces detected in the input image.")
```

Listing 2: Python Code for Face Recognition

References