Student: Joel Eliston

Professor: Dr. Tejaswi Gowda

AME 494/598: Programming the Internet-of-Things

03 December 2023

Quadcopter controlled by Hand Gesture

Introduction

In this innovative project, the primary goal is to harness the capabilities of a hand pose estimation model, Posecaster, to seamlessly interface human hand gestures with the dynamic control of a quadcopter. Leveraging the power of Posecaster's hand landmark detection from a webcam, the project unfolds a sophisticated interaction paradigm between human gestures and unmanned aerial vehicles. The real-time coordinates of hand landmarks are transmitted to a JavaScript code through a WebSocket, where advanced processing isolates the precise position of the index finger. Subsequently, this information is relayed to MATLAB via a socket connection. In the MATLAB environment, the received hand gesture coordinates serve as dynamic control inputs for a quadcopter simulation running in Simulink. This integration offers a tangible demonstration of how cutting-edge computer vision, web communication, and simulation technologies converge to create a hands-free and intuitive control system for aerial vehicles.

Model design

1. Model Overview

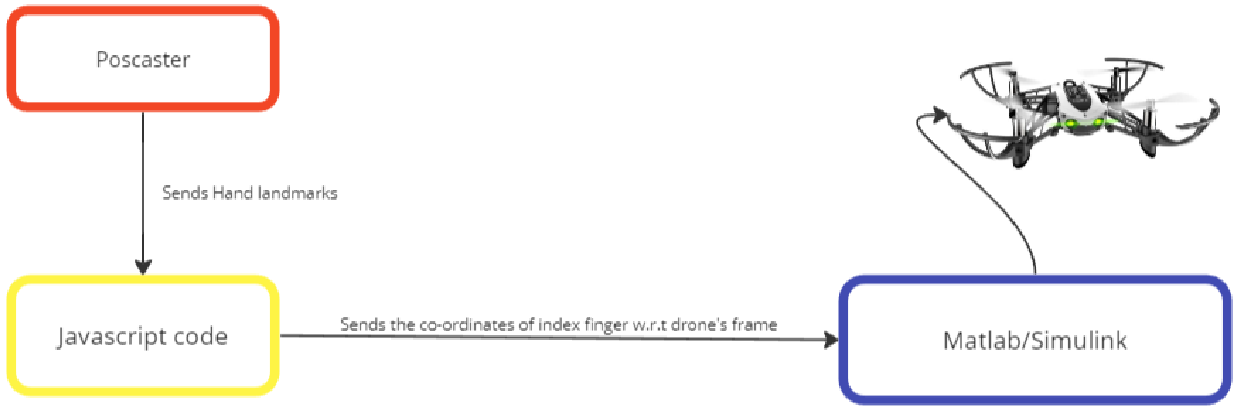
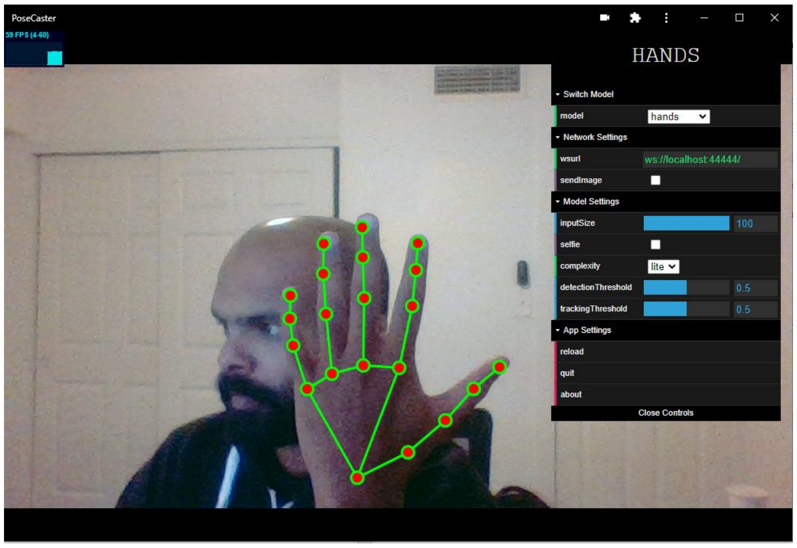


Figure 1:Overview

The model seamlessly integrates several key components to achieve hand gesture-controlled quadcopter navigation. The Posecaster model captures intricate hand landmarks through a webcam, and these coordinates are transmitted in real-time to a JavaScript code using a WebSocket connection. Within the JavaScript environment, a processing algorithm extracts the crucial index finger position from the all of hand landmarks. This information is then communicated to MATLAB through a socket connection. In MATLAB, the received hand gesture coordinates serve as dynamic control inputs for a quadcopter simulation embedded within Simulink. This simulation not only interprets the precise positioning but also provides a responsive feedback loop, orchestrating a seamless interaction between the hand position and the virtual drone.

2. Poscaster



model	hands
wsurl	ws://localhost:44444/
sendImage	☐
inputSize	100
selfie	☐
complexity	lite
detectionThreshold	0.5
trackingThreshold	0.5

Figure 2:Posecaster

Posecaster, a user-friendly hand pose estimation tool, leverages the capabilities of the popular Mediapipe framework to deliver precise and real-time hand landmark coordinates from webcam inputs. Operating seamlessly through a straightforward interface, Posecaster utilizes the WebSocket protocol, specifically wurl (ws://localhost:44444/), to transmit the detected hand landmarks to external applications. With a focus on simplicity and accessibility, Posecaster provides an efficient solution for integrating hand gesture control into diverse applications, including interactive systems, games, and drone navigation projects.

3. JavaScript code

This JavaScript code acts as a bridge between Posecaster's hand pose data and the MATLAB simulation, making the translation and transmission of coordinates to enable real-time control of the quadcopter.

- ***WebSocket Communication:***

The JavaScript code establishes a WebSocket server (wss) to receive hand pose coordinates from the Posecaster model.

Hand landmarks are transmitted in JSON format over the WebSocket from Posecaster.

- ***Data Processing:***

The received data is processed to extract the coordinates of the index finger, essential for controlling the quadcopter.

If the received data is empty (indicating no hand pose detected), default coordinates (0,0) are assigned.

- ***Coordinate Adjustment:***

The (0,0) coordinates from Posecaster correspond to the top-left corner of the image. In this code, these coordinates are adjusted to have (0,0) at the image center.

The adjusted coordinates are then prepared for transmission to MATLAB.

- ***TCP Communication with MATLAB:***

The JavaScript code establishes a TCP client (tcpClient) to communicate with MATLAB using socket connections. Upon receiving a "send" signal from MATLAB, it transmits the processed and adjusted hand pose coordinates to MATLAB for further simulation.

4. MATLAB code:

This MATLAB code facilitates the integration of hand gesture-controlled coordinates received from the JavaScript code into the Simulink model, making a seamless interaction between real-time hand gestures and the simulated drone. The communication scheme ensures accurate and responsive control within the Simulink environment.

- ***Simulation Initialization:***

The MATLAB code begins by opening the Simulink model (parrotMinidroneCompetition), configuring simulation parameters, initiating the simulation and start the simulation. In this section, the MATLAB code is paused for 7 seconds in order to let the drone move to the center of the environment (x=2, y=2). Since the environment has few obstacles in the simulation, this ensures that the drone will not collide with any of it while tracking the hand position.

- ***Communication Setup:***

The code establishes a TCP server (tcpip) on the local host, waiting for a client (JavaScript) to connect. This server listens on port 8999.

- ***Simulation Control Loop:***

The main loop runs for a specified number of steps (1200 in this case) to demonstrate the functionality. Within each iteration, a signal "send" is sent to the connected client (JavaScript) using fwrite(s, 'send').

- ***Data Reception:***

A nested loop waits until data is received from the connected client through the TCP connection. The received data is read, converted from binary to a numerical matrix, and processed.

- ***Coordinate Adjustment:***

The received coordinates are adjusted based on the specific requirements of the Simulink model. Notably, x_pos and y_pos are updated and rounded.

- ***Simulink Parameter Update:***

The adjusted coordinates are then used to set parameters in the Simulink model, influencing the drone's position (x_pos and y_pos+1.2).

- ***Simulation Step Execution:***

The Simulink model is commanded to execute a single step (continue) and then paused (pause) for a specified duration. This is done so that the drone in the simulation moves to the new co-ordinate.

- ***Loop Termination and Cleanup:***

The loop iterates until the predetermined count is reached, after which the server is closed, and the Simulink simulation is stopped.

5. Simulink

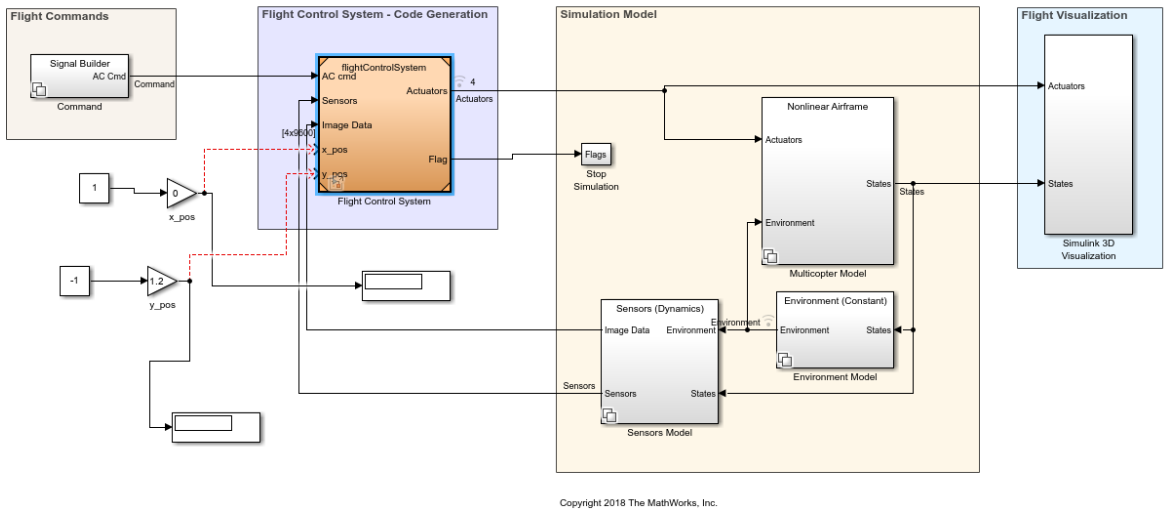


Figure 3:parrotMinidroneCompetition Simulink block

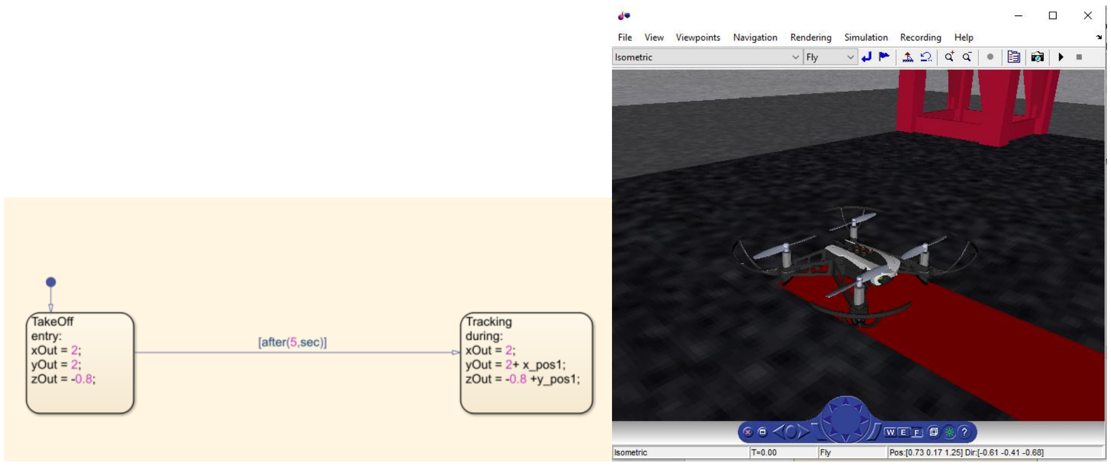


Figure 4: State flow chart and Drone simulation

The modified Simulink model, based on the "Parrot Minidrones Support from Simulink" add-on example, introduces key alterations to enable dynamic drone control. Two gain blocks, namely x_pos and y_pos, are incorporated to receive position coordinates from MATLAB. These coordinates serve as inputs for a Stateflow chart embedded within the model. The Stateflow chart dictates the drone's behavior, guiding it to a central position (2,2, -0.8) during initialization to avoid obstacles. Subsequently, as the MATLAB code sends updated coordinates, the Stateflow chart transitions, instructing the drone to move to the newly received position. The simulation operates on a time delay, with MATLAB pausing for 3 seconds to allow the drone to execute its movement based on the received coordinates. This straightforward model modification ensures precise coordination between MATLAB and the Simulink environment, facilitating a controlled and responsive simulation of the drone's movements.

Discussion and Conclusion:

The initial plan for the project aimed at creating a real-time system using Python, AWS, and MATLAB to control a Parrot Anafi drone based on hand movements captured by a webcam. However, influenced by the professor's suggestion to explore Posecaster, the project took a different route. The final implementation involved a JavaScript code receiving hand landmark data from Posecaster, isolating the index finger position, and transmitting it to MATLAB for simulation, using a simplified approach.

The designed system faced challenges, particularly in achieving perfect real-time responsiveness. MATLAB's pause during simulation updates introduced delays, and variations in coordinates sometimes caused inadequate simulation intervals. The pixel-to-distance conversion algorithm also revealed limitations, demanding refinement for precise tracking. Despite these challenges, the project provided valuable insights into data transmission using websockets and sockets. The readings and workshops in AME 494/598 significantly influenced the project, particularly in understanding WebSocket and socket communication. Challenges encountered, like synchronizing pauses and message handling, enhanced practical problem-solving skills.

Looking forward, the experience gained from this class and project forms a foundation for future research. The focus will be on refining real-time communication, addressing synchronization challenges, and exploring methods to minimize delays in simulated drone movements. The goal is to enhance the responsiveness of the system, making it more seamless and robust for potential applications in drone control through hand gestures.