Sense HAT Controlled Pictionary Game

Nicholas Clavette

Johann Karl Muller

Introduction:

The Sense HAT Controlled Pictionary Game was conceptualized for a final project in ELE408/409 Embedded Systems with Professor Bin Li. It uses multiple embedded systems approaches learned during the term such as Python programming, Web Programming, GUI (Graphic User Interface) Programming, and Socket Programming to produce a final working product.

Project Description:

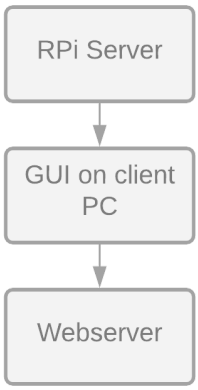
Pictionary is a drawing game where players take turns drawing for their team. The image to be drawn is randomly picked and one member of the team must start drawing while the rest of his team tries to guess what it is. If their team guesses correctly what the player is drawing within the given time, then their team will receive a point. It is a very fun game, and it could be expanded to have more interactive controls such as a drawing "wand". This was the basic idea that pushed Johann and I to decide that we could use IMU (Inertial Movement Unit) motion data from the Sense HAT to control drawing on a screen.

Project Motivation:

Motion tracking is a difficult problem to crack. In the real world, there is always some error being integrated into the estimated position of an object. This project was an opportunity for us to dip our feet into the reality of motion tracking algorithms, and gain perspective on how fruitful solving this problem could be. It is also fun to play games with friends and family, and in a time of social distancing, the ability to play a game over a network together seems very comforting.

Detailed Discussion and Results:

We decided that the Raspberry Pi could be a server, whose only job is to connect to a client, and send IMU data as quickly as possible. A client PC will then connect to this server through socket connection and a GUI application receives the gyroscope data from the Raspberry Pi as it is transmitted. This GUI application also has a drawing system, which uses a canvas to produce lines on it, correlating to the received gyroscope data. The image is then captured from this canvas and saved into a jpg file that will serve to update the image being transmitted over the web to allow the team members to see what is being drawn in real time and start guessing.



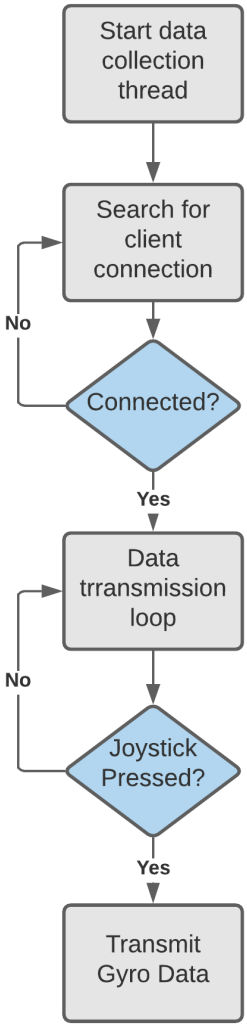
The web-socket allows users to connect to a chat and send guesses while the image is being updated into the screen.

Raspberry Pi Server:

As the motion of the Sense HAT would be our data to draw on the screen, we needed to receive this data as quickly as possible. The IMU on the Sense HAT produces gyroscope data by measuring its rotational acceleration on a pitch, roll, and yaw axis and sums these accelerations over time, keeping track of the angles it is currently at. The Sense HAT documentation states that it is ideal to sample the IMU for gyroscope data as quickly as possible, to keep the latest rotational data as accurate as possible.

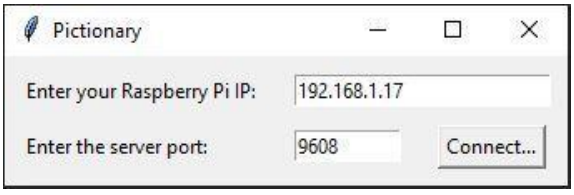
To do so, the Raspberry Pi has a separate thread that is always running, overwriting a global variable with the current gyroscope data. The main thread of our server simply waits for the joystick button to be pressed and held, and then samples this global variable and sends it to the connected client.

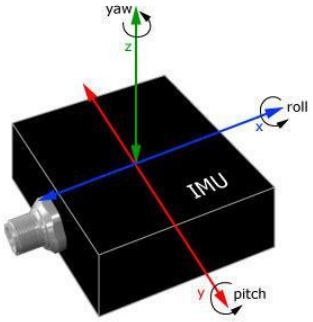
Keeping the program as simple as possible on the Raspberry Pi was extremely important, as we needed to receive this data as quickly as possible, and the Raspberry Pi is not a very powerful computing platform.



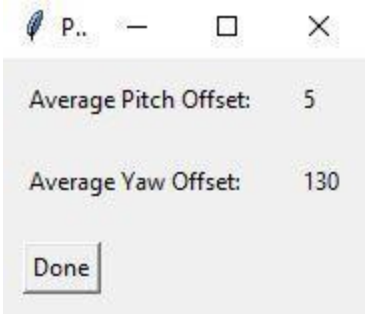
Client-Side Application

The client-side application is a GUI which allows the user to connect to the Raspberry Pi server with its IP and port number. The GUI will warn the user if the connection could not be established.



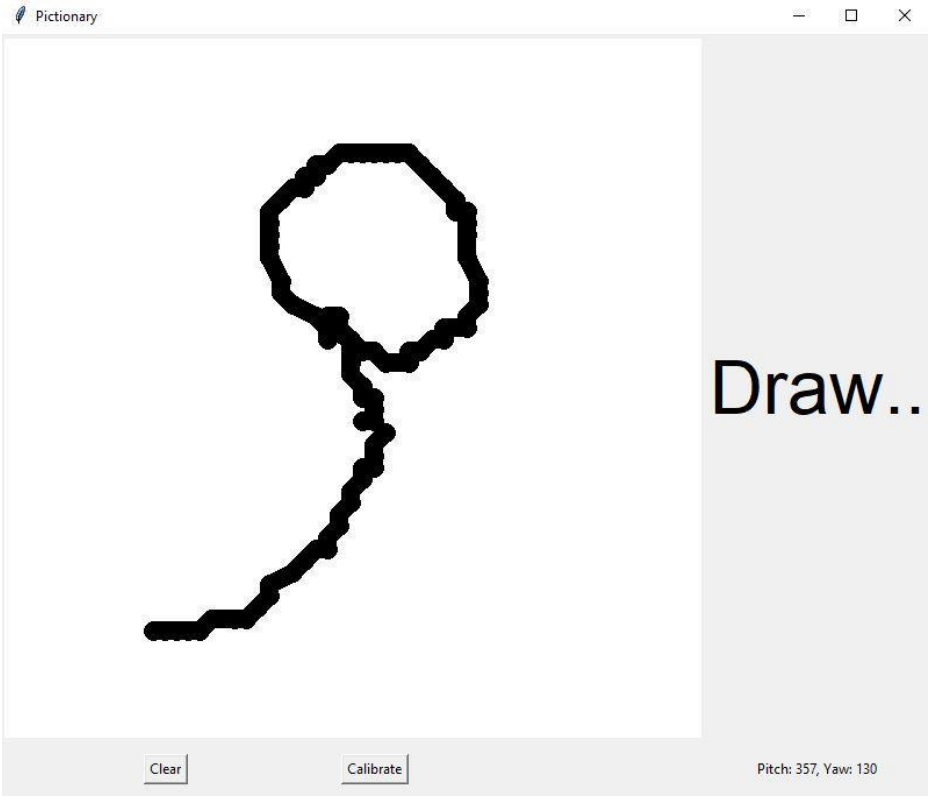


Once connected, the application brings the user to a calibration page, which allows the player to set their yaw offset. The image to the right show's which axis correlates to which type of rotation. Our application only uses pitch and yaw, where pitch rotation will allow vertical drawing of the lines, and yaw rotation will allow horizontal drawing of the lines. Since we are all sitting flat, respective to the horizon, the pitch will always be the same for any player. This is not the case for yaw, however. Since each of us may be sitting a different direction with respect to the magnetic compass of the Earth, we must account for the yaw offset for each player. The pitch and yaw data are also put through a transfer function, which maps the incoming data to a pixel range



$(0 <= x < 600, 0 <= y < 600)$

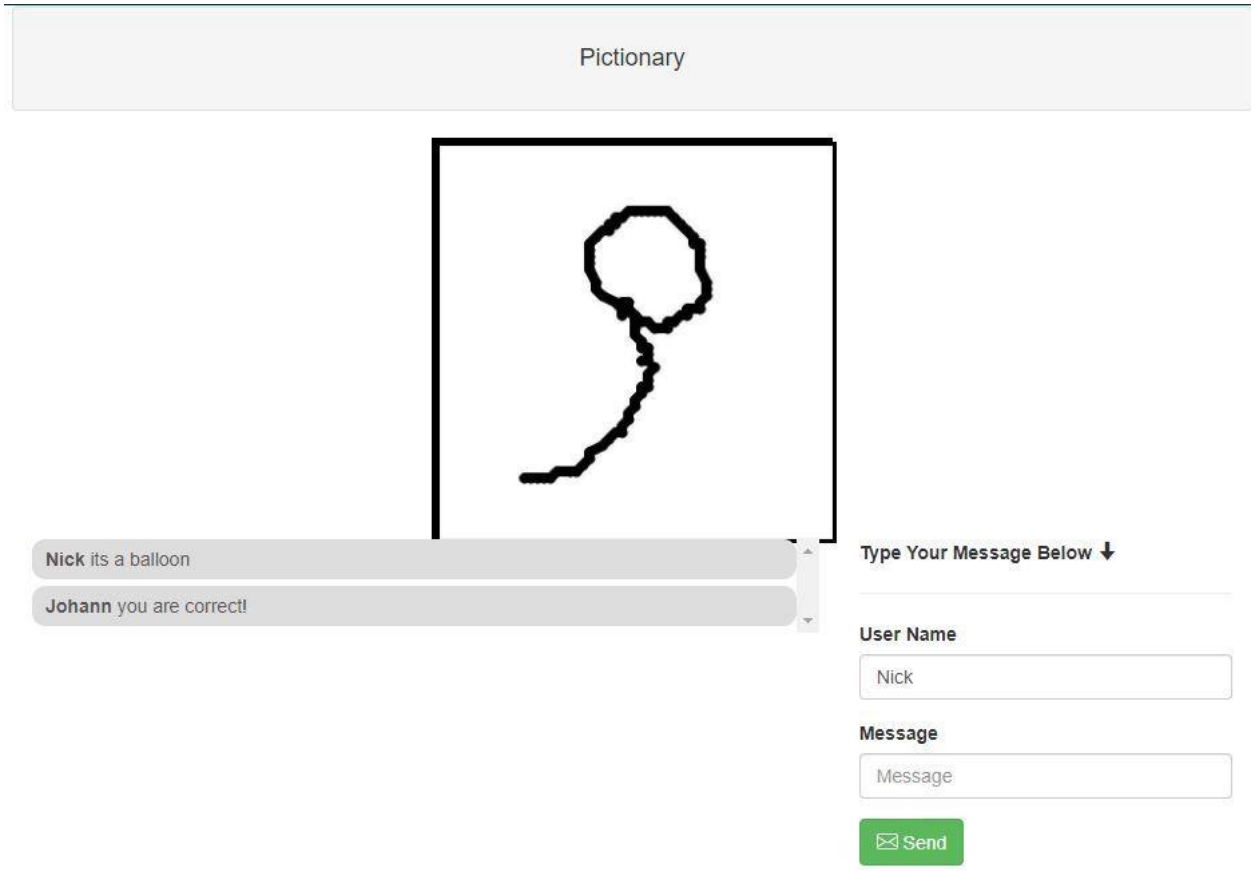
The client application uses a thread to constantly be checking for incoming gyroscope data. When data is received, it fills a python list with the data. We build a queue of gyroscope data this way. The main thread continuously monitors this queue, and operates on the first element of the queue, and then deletes it. This allows for no data loss if the drawing action takes more time than expected. The main thread can always catch up to the queue once the player releases the joystick button and stops sending data. Further improvements can be implemented to increase precision.



Above is an attempt to draw a balloon. It works surprisingly well right!

Web Server

A web server can be hosted by the player who is actively drawing. The server monitors an image file which is updated by the GUI. When the image is updated, it too updates the image on the website. Players can then guess through the chat what the drawing is. The web server was programmed in python using Flask and Flask-socketio libraries and the website was developed with HTML, CSS, JavaScript and JQuery and it was only tested on a local network. If time had permitted, we could have implemented the random system to pick a theme or word to be drawn, as well as a system to "read" guesses and alert users when the guess was right. Below is an image of the final web site.



Conclusion:

This was a fun project to work on. It was challenging to find the best way to transmit the IMU data over the network and pushed us to think outside the box. Unfortunately, machine learning was unable to be implemented due to a lack of time from the busy schedules of the developers. However, we recognize the potential for machine learning with this type of application to

produce smoother translations between incoming gyroscope data and lines on the screen. Machine learning could also be used to take in acceleration data as well, enabling drawing to be performed using planar motion along with rotational motion. We believe this project is a great starting platform to design and test these algorithms on.