Daniel Stoller

Dr. Dickerson

Junior Design

Project 3 Final Report

Design Overview

- **High-level description of your design and its purpose.**

For my design, I created a 3D color matching game that takes roll and pitch orientations from an IMU as the inputs. This game has 4 difficulty levels and a tutorial mode, each with different features to either help you min or to make the game harder. The IMU is used to gather data about orientation based on the gyroscope and accelerometer in it by reading then filtering the data using a Kelman Filter. The pitch and roll are printed serially and will be read by the python code. The game utilizes PyGame, TKinter, PyOpenGL, and PySerial, with PyGame creating the windows, TKinter taking inputs from the settings menu, PyOpenGL to model the IMU in a 3D World, and PySerial to read the data from the IMU.

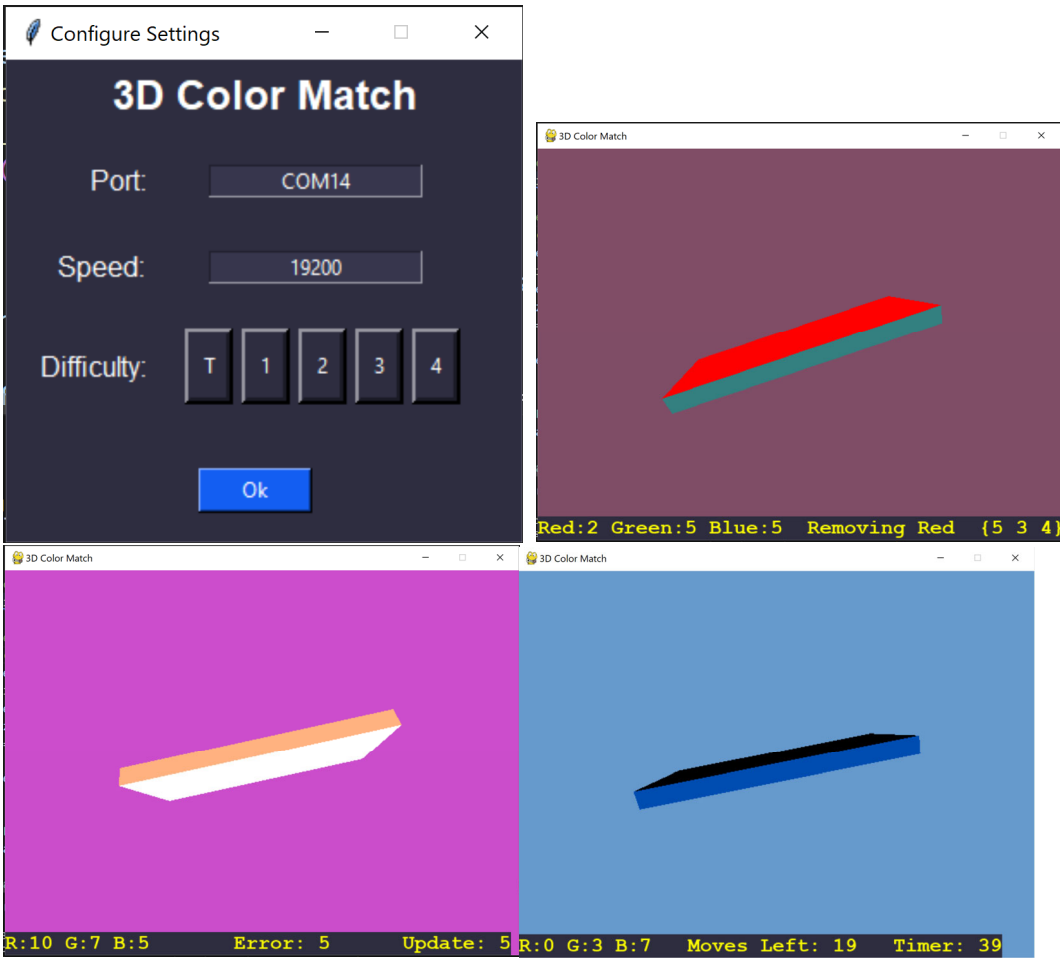
The game choses a random background color and your job is to match the color of the background by increasing and decreasing the RGB values of the model. Different combinations of red, green, and blue make different colors and by changing the ratio of the 3, different colors are created. Depending on the orientation of the IMU, the user is able to increase or decrease the amount of red, green, or blue in the color to a value between 0 and 9.

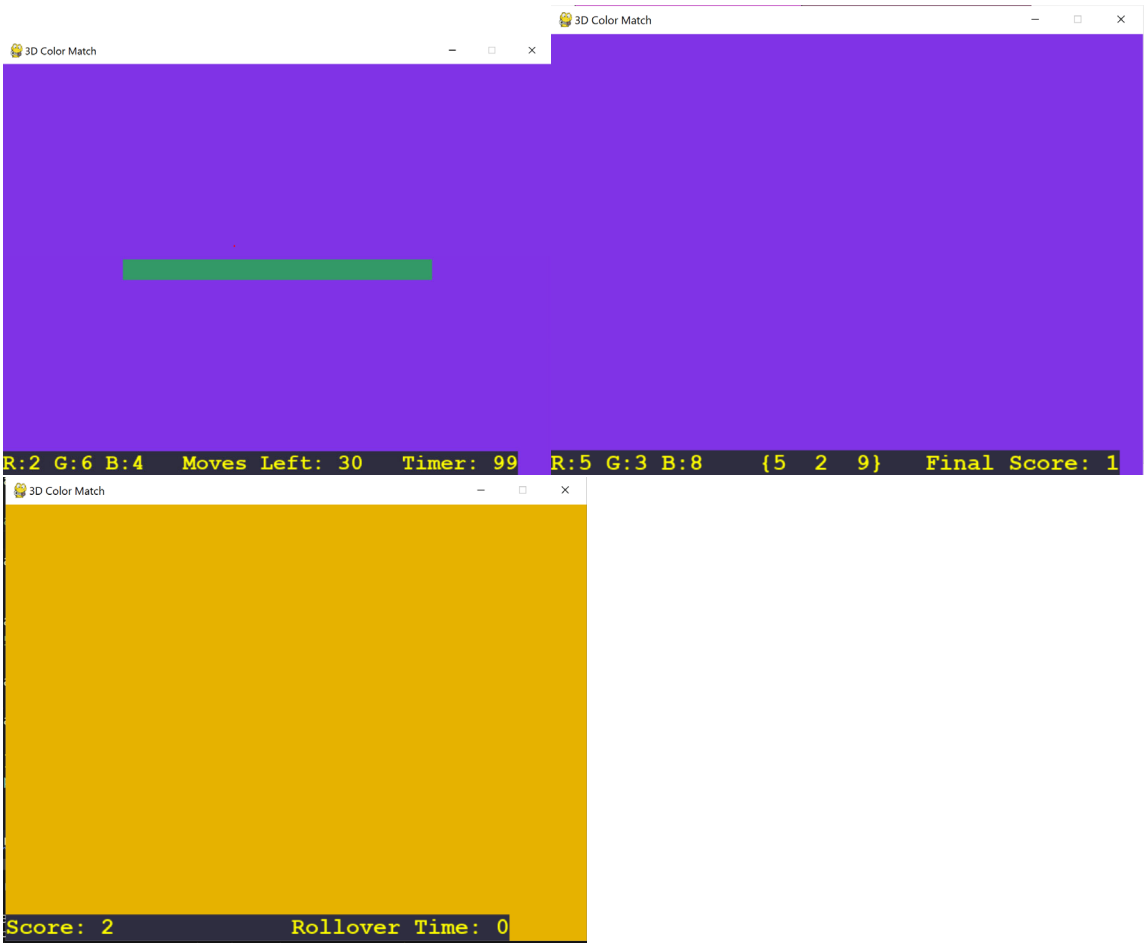
There are 6 different difficulties/modes a user can chose when playing this game:

- Difficulty 0 (no input): No color game. Shows the pitch and roll of the board
- Difficulty T (Tutorial): Shows user how to play the game. First, the instructions will be displayed at the bottom. Then, the user will be given the RGB values of their background on the bottom right and they have to just copy those numbers. In this difficulty only, the color of the top and bottom of the board corresponds to the zone that you are in, and text will display at the bottom saying what the action of that zone is. For example, if you rotate the board into the increase green zone, the bottom of the board will turn green and the text at the bottom will say “Increasing Green.” This mode is to help users learn the controls and purpose of this game and start understanding what colors they should change to achieve the desired color
- Difficulty 1 (Errors): Shows the user the amount of changes they are away from winning. The amount of changes are displayed as “error” and the error is updated every 10 seconds. For example, if the board’s red needs to be increased by 2 and their green needs to be decreased by 1, their error is shown as “3.” There is no way of losing in this difficulty
- Difficulty 2 (Normal): This is the normal game with no assists and no way of losing. The amount of time the user has spent on a given color is displayed
- Difficulty 3 (Timer): This is the first difficulty where the user can lose. There is a 60 second timer given and if the user doesn’t make that 60 second timeframe, they lose. The leftover seconds are rolled over to the next round to incentivize finishing early. When starting the next round with extra seconds, the given seconds are used before the extra seconds. For example, if I finish the

first round with 30 seconds left, my next timer would show 90 seconds. If I finish that second game with 75 seconds left, only 45 seconds would roll over instead of 75 or 60, giving you 105 second next round. This is because the rolled over seconds is just the allotted 60 seconds and 15 of them were used to complete that round.

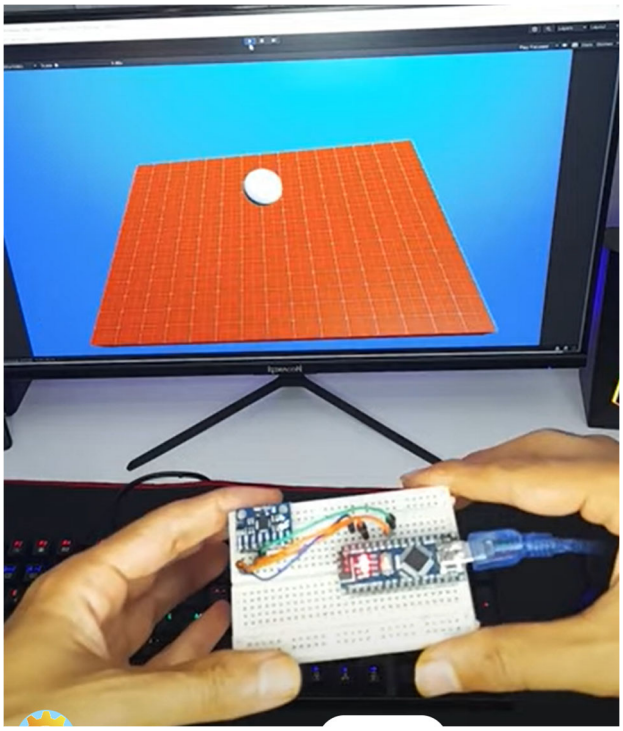
- Difficulty 4 (Move Counter and Timer): This difficulty gives you a certain number of moves along with a 60 second timer that does not rollover. In any given color, you are only allowed to make 30 moves. If you make 30 changes and don't achieve the desired color, you lose. Brightening and darkening (changing R, G, and B by 1 together) counts as 1 move.





- **Original design concepts that you considered (at a high level) and then your final design. This document is not only a description of the technical aspects of your design, but it is also a digest of your design process.**

The original concept (or inspiration) for this design project was to create a 3d model of a ball rolling on top of the IMU model. This would involve either learning a way to attach 2 objects in OpenGL to have them tilt synchronously and then translating across the x and y plane at various speeds depending on the angle of the pitch or roll. Upon more research and a greater understanding of OpenGL's capabilities, I learned that this project wasn't ideal for OpenGL. This is because OpenGL places objects in a 3d world and OpenGL moves and rotates the world around the camera, but the camera doesn't move when simulating the IMU. Instead, the world is being rotated different directions in front of you. OpenGL would not be ideal for this project because the cube would have to be moved within the 3D space and from my research it didn't seem like that was possible. To conclude, OpenGL moves and tilts the world around the camera, but from my research it does not seem possible or simple to move objects within the space because all edges, surfaces, and vertices are predefined and I could not figure out a way to effectively change these values to move the cube within space.



- **Clearly explain how your project expands and builds on previous works (i.e. justifications for a 6-week project)**

This project was a challenge because the only experience I had with an aspect of this project was pinning the IMU and coding in Arduino. I have never worked with sensors in Arduino before, never learned how to code in Python, and never 3D modeled before. This project required learning how IMU sensors gather and filter data, how 3D modeling with PyOpenGL is done, basic python syntax and Linux operations, and researching different functions and uses for functions in PyGame, PyOpenGL TKinter and PySerial. This involved many hours of research to get started with my project because I have never coded in Python before so this project involved learning the syntax and understanding what each function in the base code does so I can understand how I will be able to modify it and add to this code.

This project was very useful for my future because the experience I got in coding Python and retrieving IMU data will be invaluable to me. Python is used for almost every Software Engineering job and I have never worked with it before, so having this experience with coding in Python will boost my learning and can show companies that I have coded in Python before. More important that understanding Python syntax, I proved the ability to learn and use python libraries without prior learning or experience. Python is a powerful coding language because of the sheer amount of libraries that are available that can be used for a wide variety of purposes . Having the ability to utilize a library without prior experience will prove useful to me for both my education and my profession.

Additionally, finding an accurate and effective IMU reader for the MPU6050 and understanding the capabilities of the IMU will be useful for Micromouse, where we were starting to use an IMU to calculate the positioning of the mouse within the maze by the gyroscope and accelerometer to track the speed and turns. This experience helped me better understand what IMU values we will need to achieve this.

- **Cite your references and include images/schematics**

IMU Model Python Code: https://github.com/MA-Lugo/PyIMU_3Dvisualizer

IMU Reader with Kelman Filter: https://github.com/CarbonAeronautics/Part-XV-1DKalmanFilter/blob/main/ArduinoCode

Preliminary Design Verification

- **Write a summary of preliminary testing results from your prototypes in this section.**

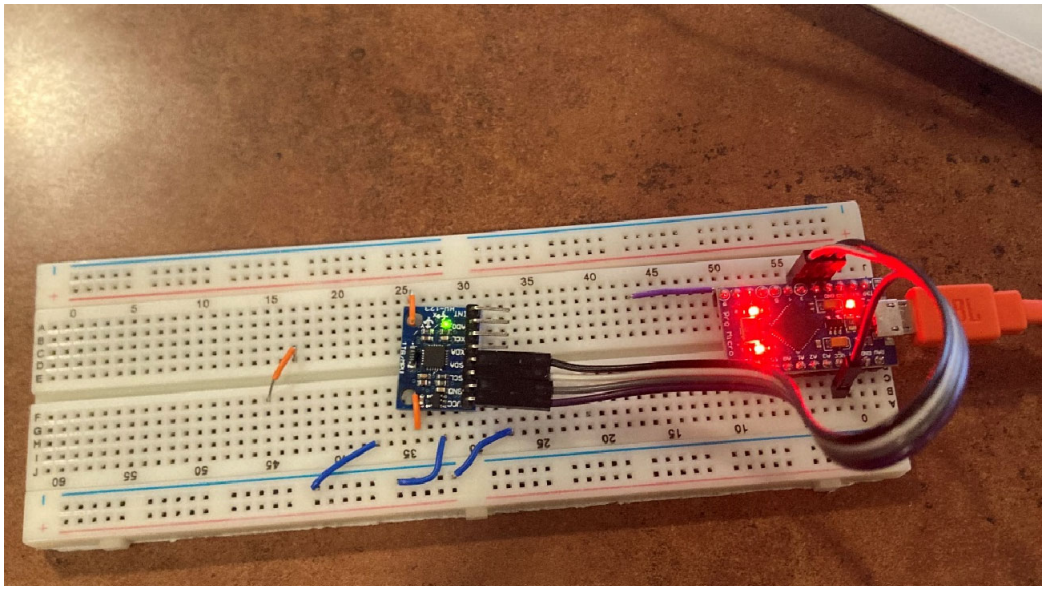
Preliminary testing results involved verifying the code I was using on GitHub and verifying the pinout and accuracy of the MPU-6050 IMU.

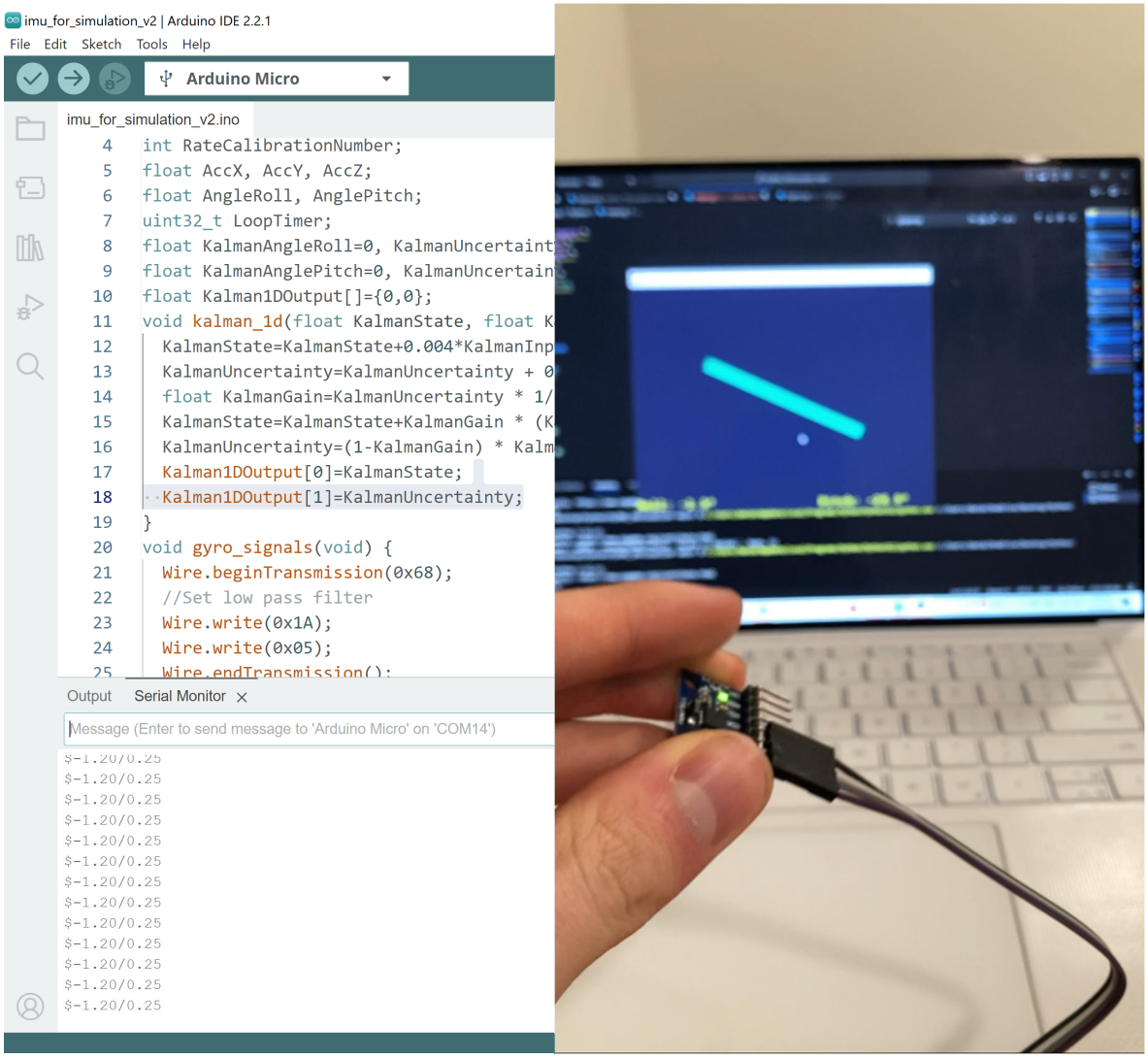
- **What did you to do verify your design before moving into the final production phase (e.g. breadboards, test programs, etc)?**

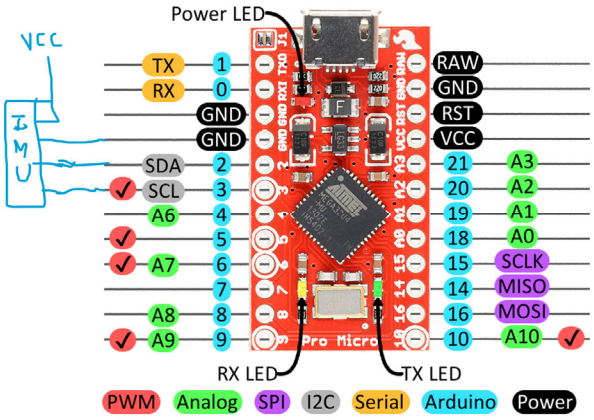
The milestones I needed to complete to start this project included achieving accurate angle readings from the IMU, installing all necessary software, verifying the model code's effectiveness, and integrating my serial outputs with PySerial to transfer the data from Arduino to the model.

IMU:

I verified and tested the IMU's pinout and tried reading basic values. The built-in reading data function for the MPU6050 showed that the pinout was correct, but the output was less than ideal. It was inaccurate, drift overtime, and very fidgety. Changing the yaw of the IMU would break the center point and shift all the values over. At first, I tried to fix this by changing the error values within the equations to help stability, but this proved overall ineffective. The GyroAngle kept linearly decreasing when idle, and gyro angle adds upon the previous gyroangle, so to counterbalance this, I added a precise value that would counteract the slope of the linearly decreasing angle. This proved effective for getting stable values when idle, but after big motions, the starting point would be way off from 0 degrees. I looked further into filtering that would better integrate the accelerometer and gyroscope and found the Kelman filter. This filter has more of a predictive nature to it and better looks at the past in order to determine here the MPU6050 is the present.



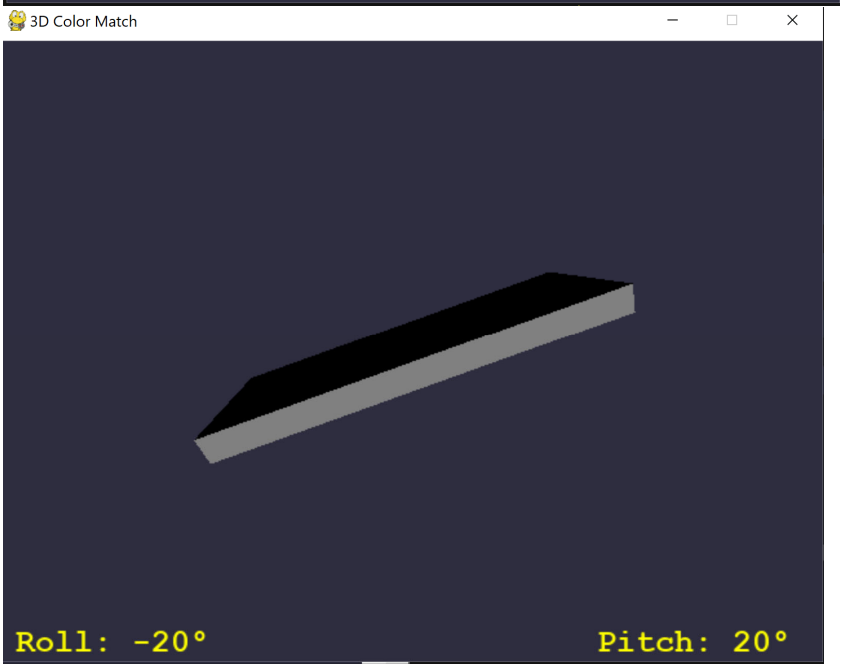




Python, Model, and Serial:

I first downloaded Python and all necessary libraries. Then, I ran the code on my computer to verify the success of my installations and the code itself. I was able to open the model and connect the serial,

but none of the serial values were being read. To fix this, I had to create a parser that would extract the data from the serial input, but wouldn't read data that didn't represent the roll and pitch angles. Once that was completed, I had to change the sensitivity of the rotations from the input angles so the model wouldn't over-rotate.



- **What steps did you take, if any, to determine the feasibility of your design?**

Determining the feasibility was based on learning of OpenGL's capabilities and the IMU's accuracy. I found during my prototyping that OpenGL's capabilities were less user friendly that I had expected and the IMU originally had very inaccurate readings. During feasibility testing and research, I was able to find

better solutions to filtering my IMU data and discover how I could pivot my project to something that was better suited for the software I was using. This was done through watching videos on youtube, looking at Github code, and searching with ChatGPT. The results allowed me to find better solutions and expand my understanding.

- **Include photos of the initial, assembled prototype, and photos from any testing results if available. This includes verification of hardware, software and enclosures.**

Design Implementation

- **Include an overview of the overall system, described at a high level. Include a listing of all the relevant subcomponents developed/used in creating the final software/hardware**

This design involves serial connection between the Arduino and the Python code, then models based on the data. The game was made by modifying the current features of the code then adding classes and functions of my own including the Background and Color Factor classes. Background tracks the color of the background and any other game statistic such as the background color, timer, and moves left. Color Factor tracks and decides the changes in the color of the board. ConfWindw opens the settings window and gets user inputs, RunApplication takes those inputs and stores them in the game's settings. InitPygame opens OpenGL Window and InitGL draws the world, randomizes the background color, and choses where the camera is. DrawGL draws the model and text at the bottom of the screen, updating the roll and pitch values to the model's orientation and showing different messages on the bottom based on the difficulty chosen. Read data takes the serial input and parses the Pitch and Roll by utilizing the findall function which extracts all positive and negative numbers from strings that have start with "$" and have 2 positive or negative numbers divided by a "/." Finally, the main creates the settings window, starts the OpenGL window and initializes it, tests the serial connection and its ability to read the data, then repeatedly updates the model. In the repeating while loop within the main, if the background and color factor are the same, we reinitialize the OpenGL world with a new background and reset values. If they weren't matching and the timer is up or the user ran out of moves, print the correct RGB values and the final score and close the window.

- **If there is anything else that is notable about your design, or design practices, please include it in this section. Discuss your design process in developing the system and any challenges that came about**

Best design practices included commenting, creating adjustable variables, and organizing my code into distinct classes and functions. Understanding the code and what each function does was a large task and quite grueling but it was important because if there was a problem that needed debugging or I wanted to modify or duplicate an aspect of the code, I needed to understand how they did it in the first place. I put comments all over the entire file so I could go back and understand exactly what is happening in every function, class, or loop. This makes it easier for me to go back and read and makes it easy for a future employer to understand exactly what was going on in the code.

Whenever I chose a value that I thought I might want to adjust later, I created a variable for it and put the comment "####ADJUSTABLE####" above it. Some examples of values I expected I might want to tweak later included the game timer, the allowed number of moves, the range of values for RBG, the roll and pitch boundaries for each color zone, and the amount of time required to be in a zone before adjusting the RGB values.

Finally, I organized the locations of my code and made functions for common actions. This was important to do because this game has 5 different difficulties and some difficulties have certain features while some don't, so having functions for everything makes it easy to track and debug my own code, and it makes all the necessary switch cases more compact and efficient. By making functions for everything, it allowed my code to be smaller and to fix the root of the problem easier. Some examples of me doing this include making a function to restart the game after a score or starting, making 4 separate timer-related functions, and creating color modifying functions which are each very long because there are 9 different color zones.

```
class Background:
    r = 0
    g = 0
    b = 0
    score = 0
    start_time = 0
    ####Adjustable####
    timer = 60
    rollover_time = 0

    #Used so after update period is over, error will only be updated once
    #Can't be 0
    error_timer = 1
    error_delay = 10
    error = 0

    ####Adjustable####
    total_moves = 30
    moves = 0
```

```
class ColorFactor:
    #R, G, and B must be between 0 and 1
    r = 0.5
    g = 0.5
    b = 0.5
    ####Adjustable####
    change_amount = 0.1

    #Stores chars representing the different orientations
    #Will be compared with current orientation to see if color should be changed
    #Increasing and decreasing share the same char because it's impossible to get from increase to decrease without going thr
    prev = 'c'
    #stores amount of time in a given orientation
    time = 0
```

```
#RGB = (0.5, 0.5, 0.5)
def reset_colors(self): ···

#Change color based on orientation and time count
def color_handler(self): ···

def update_color(self, color, increase): ···

def compareBackground(self): ···

def explainAction(self): ···
```

Design Testing

- **Explain your test plan, the procedures you use to test your design and the outcomes from those test.**

After initial testing of the IMU and the simulation software, testing was not too difficult. The main 2 ways I debugged included using print statements to see how variables change based on certain actions and inputs, and using “except Exception as e” which would print what went wrong and which exception was triggered.

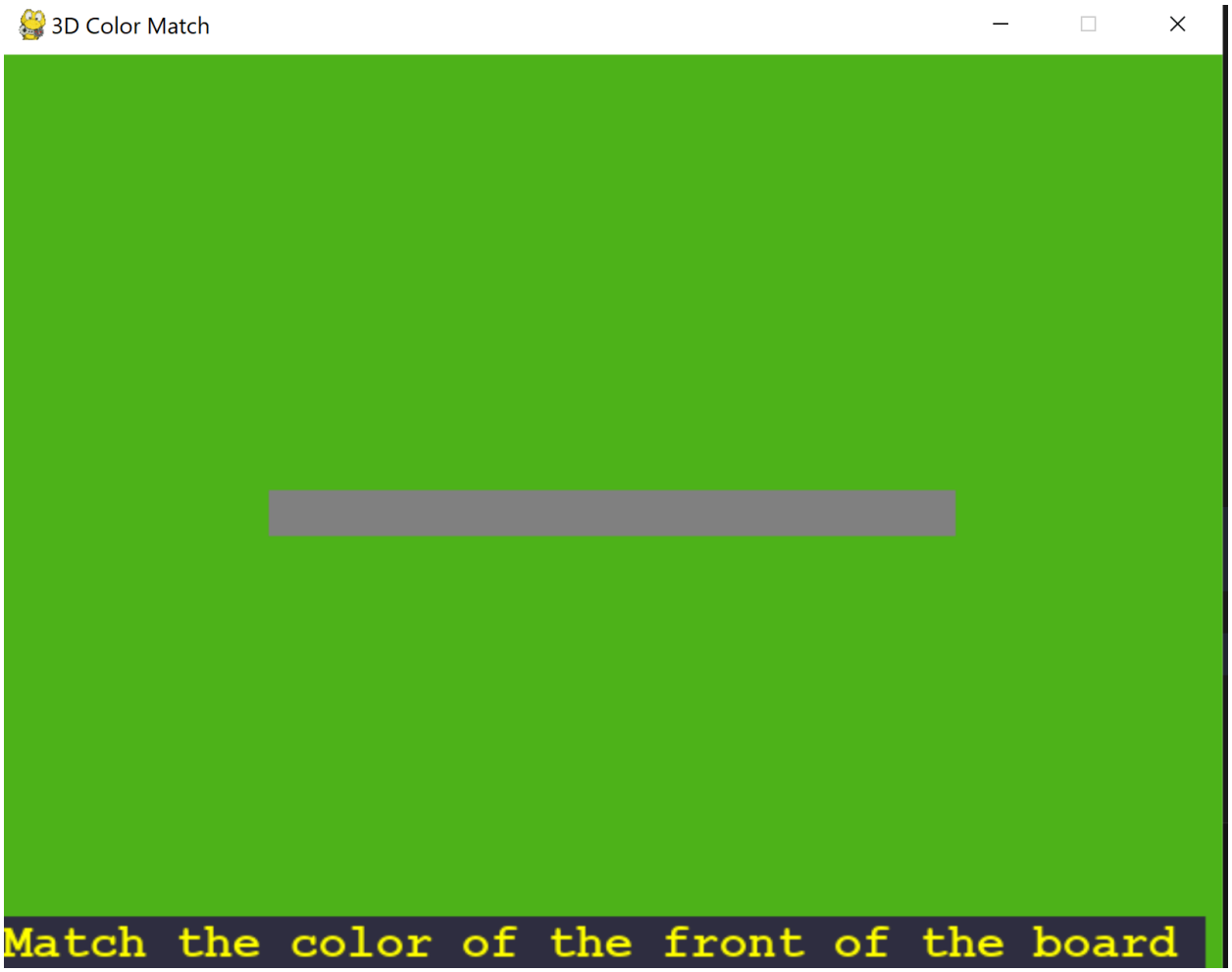
```
#If an error was caused
except Exception as e:
    print(f"exception: {e}")
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT)
    DrawText("Sorry, something is wrong :c")
    pygame.display.flip()
    time.sleep(5)
```

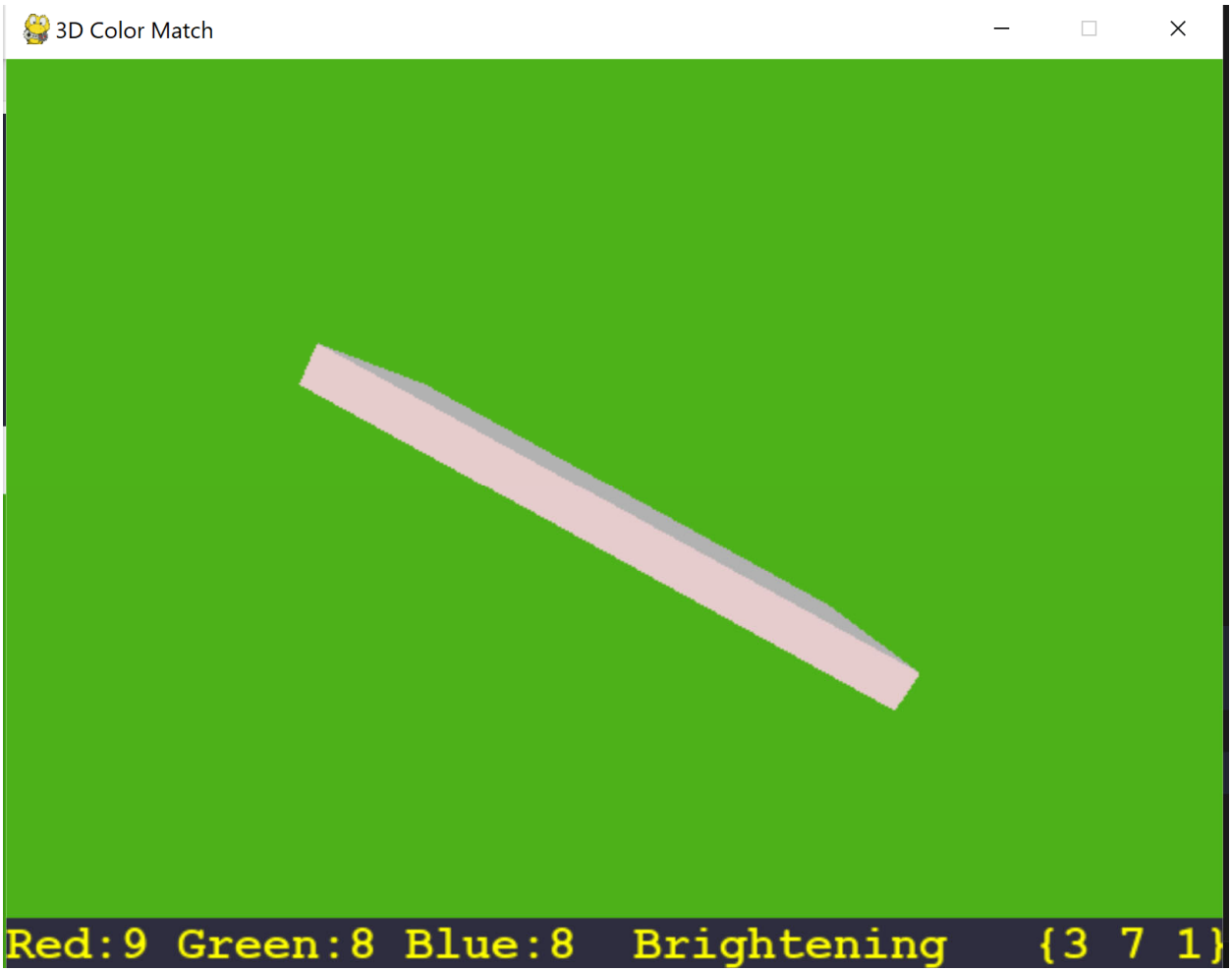
The main form of testing that was necessary was user testing. I wanted my game to be very user friendly and for it to be played without being given an explanation on how to play beforehand. I asked all my family members to play and asked for feedback. Some changes that came from having them test the game included:

1. Tutorial mode which shows the bounds of each color zone and gives a written explanation of what each zone does. This was necessary because when showing my mom, she couldn’t visualize where each zone was and didn’t know how far to tilt in order to change. This led to her both thinking she was changing when she wasn’t tilting it far enough, and also accidentally tilting it while holding it still causing the color to change. The tutorial helped get the user used to the game. Also, being given the answer key to the given colors helps the user better understand how RGB colors work and help them predict the values in the future.

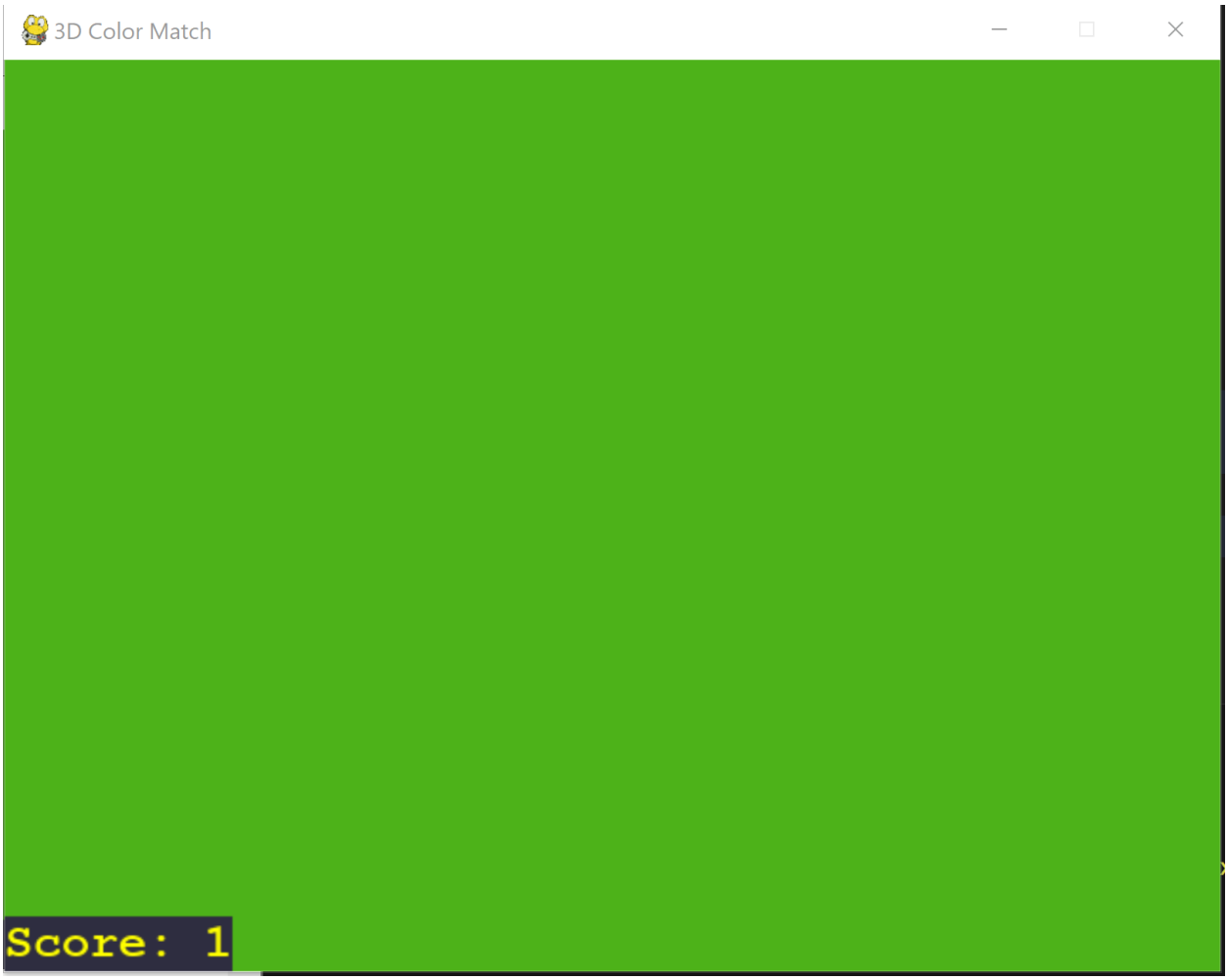
2. Changing the bounds of each color zone and making the lighten and darken color zones the smallest ones. I found that it was very easy for users to hit the up, down, left, and right zones, but the diagonal zones were more difficult. This is because people can easily tilt in 1 direction and keep it straight in the other. To fix this, I increased the area of the diagonal zones.
3. Watching the users play the error mode made me realize they were just making 1 change then waiting for the short update period to see if their 1 change helped them or not. I didn't' like this type of play because it didn't involve them thinking. I increased the update period to take away some of the effectiveness of displaying the error
4. I found that I gave too many moves in difficulty 5, so I decreased it from 50 to 30.

Watching my family play gave me better information that I could've gotten myself because I fully understand how this game works because I was the one that created every aspect of it. Seeing someone who knows nothing about this game


3D Color Match
Match the color of the front of the board


3D Color Match
Red:9 Green:8 Blue:8 Brightening {3 7 1}


3D Color Match
Red:3 Green:7 Blue:7 Darkening {3 7 1}



- **If you had unsuccessful attempts, include write-ups of those attempts and how you remedied those issues.**

Some issues or unsuccessful attempts I had were from trying to figure out what python is capable of doing. I wanted to create toggle buttons for the settings screen, but it's not intuitive how to do it. I did a lot of research on youtube and google images to find one that was similar to what I wanted, but found nothing. ChatGPT ended up being the answer to my problems in the end, as it led me down the right path to figuring out what function and values I needed to use to do this. I did a similar thing for creating the title in the settings page too.

One issue I never solved was putting text on the OpenGL screen. I spent hours researching and debugging to find a way to place more text than just the line at the bottom, but I wasn't even close. The text at the bottom is a bitfield that was converted from a string and is directly drawn right in front of the camera and doesn't rotate. Analyzing the code, however, gave me no answers as to why the text was on the bottom of the screen and how I could move it to the top. All the solutions I found were either inconsistent and accidentally affected the board or the bottom text in an undesired way, or caused the texts to rotate along with the board. Being able to place text wherever on the screen would have been great because I could have put the words "Red", "Green", "Blue", "White", and "Black" on the screen in their corresponding zones. I could have also had enough space to write the

full words of the colors instead of “R: G: B:” At some point, I decided I had to make do with what I had and figured out different ways to communicate how to play, hence the creation of the tutorial mode.

- **Make sure to document your debugging process and any challenges that came about.**

Debugging was quite difficult at first because I didn’t understand python and I was having lots of software installation issues. It took many hours of trying different solutions to finally get everything working as it should be, but figuring out the problem took me asking friends, TA’s, and the internet what was wrong and still not getting an answer.

My debugging process was play testing and print statements. I understoon what each value did and how it would be affected by any action and by tracking the print statements, I was able to pinpoint the location of any given error. By doing my project in parts, adding small aspects one at a time, I was able to keep errors at a minimum.

Summary, Conclusions, and Future Work

Write a brief summary of your project and your conclusions from this assignment. Include a detailed discussion of changes you would make to improve the design if you were to do another design iteration.

3D color matcher is a fun game that aspires to be user friendly in its clarity, accessibility, interface, and difficulty level. This project was a result of constantly improving my previous iterations and adding any feature I can think of. It started as just a IMU model, then a color matching game, then added difficulties with different rules, then after seeing user testing added a tutorial mode. This project involved creativity, resourcefulness, and predictive thinking.

The only improvement I would make si adding text anywhere on the screen so I could have enough room to permanently show the score, pitch and roll, better signify the color zones, and be able to spread out my words more.

If I were to do another design iteration, I would figure out a way to have a cube sliding along the top surface of the model and make the RGB zones on the top surface. Having the cube within that zone will be the trigger for changing color instead of the orientation of the board. They are essentially the same thing, as tilting it a specific direction will cause a specific action, but it would be more user friendly because they would be able to see which zone the cube was currently in as opposed to the system now where you can’t see what color zone you’re in.