Team A2 – Virtual Whiteboard

Authors: Alan Song, Andrew Huang, Brian Lane: Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—A system capable of allowing a user to control their computer cursor from a distance to access a web browser and navigate to several different pages in an office or classroom environment. The system will allow for a touchless touchscreen experience where a user can use just their hands. State of the art systems are either too expensive or require the user to be very close to the screen for functionality. Our system will allow for a user to control their device from across a standard classroom using just a camera and code that can run on their computer.**

***Index Terms*—Hand Detection, Computer Vision, Gesture Recognition, Neural Network, OS Interface, Calibration, Pose Detection, Cursor Location Transform**

I. Introduction

The Virtual Whiteboard originated from the initial idea of developing something similar to Tony Stark's Iron Man Suit user interface, where he just uses his hands to control a bunch of different things on his user interface without direct contact. However, we expanded on this idea of a touchless touchscreen and thought of how it might be practical in the real world. As students who likely spend lots of time on computers, there are a lot of downsides with prolonged computer usage including damage to the eyes from being too close to the screen and harm to the body from sitting too long. The Virtual Whiteboard which allows for a user to control their computer cursor from a distance with hand motion and gestures would allow for these problems to be mitigated since they can now stay standing and will not be close to their screens. The touchless aspect also has a sanitary benefit in this time where the pandemic is still an issue, since people who might have to use the same public computer can do so without transmitting germs. Additionally, this system would allow for students or teachers to give presentations or lectures naturally in a classroom environment while making the experience more interactive and engaging.

The most important requirements for this system are the distance at which the system is functional and making the entire experience very smooth for the user. The first requirement can be directly quantified, and we have decided on trying to make our system functional for users that are between 3 feet and 15 feet from their screen. The 3 feet minimum distance is because the average arm length is around 3 feet, so if a user is within this distance they could just reach out and use a normal touchscreen. The 15 feet maximum distance is the length of an average classroom at CMU (not lecture hall), and this would be for allowing students or teachers to give presentations from across the classroom. The smooth user experience will be expanded more upon in the design requirements, but we want to enable the user to simulate the capabilities of a mouse with their hands by using different gestures for mouse clicks and scrolling.

II. Design Requirements

The smooth user experience can be broken up into three quantitative categories.

A. Latency

One of the key aspects of a smooth experience is a user making a gesture or a motion and seeing the result of it shown immediately on screen. This will be accomplished by making our design meet as low of a latency as possible. We have decided to strive for achieving a 50 ms latency for our system. This corresponds to 20 frames per second, which means the cursor on screen should update its position 20 times each second while following user input. This latency will not be noticeable to the average human and should make the system feel like it is instantly responsive.

B. Gesture recognition accuracy

When using a mouse or a touchscreen, a user wants a click to be registered as a click 100% of the time. When using any device, the user would desire that their inputs are properly detected all the time. However even then, it is natural for users to have to click multiple times with a mouse or to tap repeatedly on a touchscreen to guarantee their input goes through. We have decided to aim for less than 10% gesture recognition error in our system. To put this into perspective, for every 10 clicks a user tries to input through hand gestures, we would guarantee that they must possibly repeat a gesture only one time for successful detection.

C. Cursor precision

Our system ultimately controls the computer's cursor, which allows for the user to interact with objects on the screen. For a standard screen size of 1920 pixels by 1080 pixels, the smallest area that a user would have to click on is 30 pixels by 40 pixels, which is the "exit" button at the top right of a browser. For all other objects on screen, there is a larger area where if the cursor is anywhere within, the object will be interacted with. We want our system to be able to track user hand motion with an error of around 30 pixels so that the user will never misclick because of our system, but only by human error.

III. Architecture and/or Principle of Operation

The block diagram in figure 1 represents the overall system architecture. The project is largely done in software with the two main hardware systems being a camera and a laptop. The camera will provide the primary input to the system in the form of image data. We purchased our camera and will not be building our own camera for this project. The camera will continuously feed this image data into the laptop and more specifically the calibration and pose estimation blocks, which are both part of a hand detection module. The rest of the system is software that runs entirely on the laptop. The laptop is also something that we own, and we will not be building a laptop for this project.

A. Hand Detection

The hand detection module includes the calibration and pose estimation blocks. The module takes in images which include the user and possible other objects in the background and isolates and identifies the user's hand and arm. The hand and arm data show up as data points in the hand detection module and will be converted into coordinate points to be sent to other parts of the system. In the calibration block, the user will map out their range of motion by drawing a circle with their arms extended to detect the largest range in which the user's hand can move. The range of motion will be sent into the cursor location transform which is part of the OS interface. The pose estimation block will continually send coordinate information about where the hand is located into both the cursor location transform and the gesture recognition module. Although the pose estimation block could potentially be used to determine gestures from the different points mapped onto the user's hand, we decided to just send a cropped version of the image with landmark coordinates into the gesture recognition module instead. The pose estimation block and calibration block will be developed entirely by us.

B. Gesture Recognition

The gesture detection module will take in hand landmark coordinates and the zoomed/enhanced image of the user's hand to determine what gesture the user is making. The gesture recognition module uses a neural network to detect the user's hand gesture among a dataset of over 20 different hand gestures, of which we only need five. The neural network will directly convert the image input into an integer output that represents the gesture detected. This gesture integer will be fed directly into the OS interface. The gesture recognition module will be developed entirely by us, although the dataset used to train will be off-the-shelf.

C. OS Interface

The OS interface module includes both the cursor location transform and the OS interface. The cursor location transform is a type of calibration for the OS interface system. Since coordinates from the hand detection module will not correspond directly to coordinates in the OS to represent the screen, a transformation is needed to map motion of the user to motion of the cursor on screen. We decided not to just map the entire range of motion to the entire size of the screen, but rather make the range of motion approximately 75% of the screen size. This way for users far from the screen they will not have to make super small and precise movements to move the cursor onto small on-screen objects. This scaling will also make the system's sensitivity like a regular mouse, where the user may have to pick up and re-adjust the mouse multiple times while using it. Once the calibration is done, the OS interface will move the cursor based on relative positioning, which is taking the difference in position between two time frames to determine how far to move. The gesture recognition module will provide the gesture which will be converted into a mouse action. The OS interface module will take the hand position, calculate where to move the cursor based on current and previous hand position, and perform certain mouse actions based on the inputted gesture integer. These simulated mouse movements and actions will be fed into the operating system of the laptop, allowing for direct control of the cursor through software.

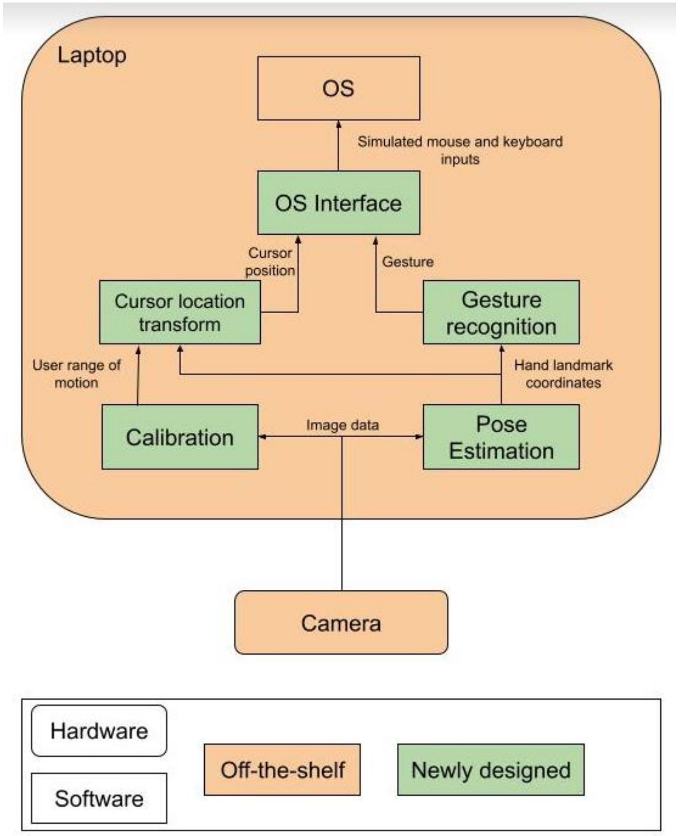


Fig. 1. Block diagram for entire system

IV. Design Trade Studies

A. Hand Detection

Multiple object detection methodologies have been used for hand tracking as the problem boils down to recognizing and sensing hand data. We considered multiple different solution approaches for hand detection: IMU, infrared sensors, ultrasonic sensors, computer vision, and Ultraleap Leap Motion Controller. Ultrasonic sensors eliminate the need for external sensors on the body, but usage of it would require complex calculations for extracting pose and location data of hand making it too difficult to work with for our use case especially given its low resolution of around 1 cm. Infrared sensors obtain sensor location with around a couple millimeters of resolution. However, the approach only gets the location of the sensors, and we would need to research and develop our own complex algorithms for pose and gesture estimation. IMU accelerometers are especially subject to drift, and small errors in measurements are exponentially multiplied during double integration for position estimation. We considered an external sensor as a calibration metric for position estimation as well as a Kalman filter, but both approaches seemed too complex given our problem statement and the effort to implement either was not outweighed by the benefits the IMU itself provided to our task. We also considered the use of an external hardware sensor called Ultraleap Leap Motion Controller that was geared specifically towards hand pose detection for AR games. While the accuracy was good at close distances, the most complex sensor the company provides has a max usage distance of around a meter; our solution statement requires around a three to fifteen feet operation distance. Thus, we ultimately decided on the use of computer vision for our detection algorithm for a multitude of reasons.

First and foremost, there has been by far the most work done in this area with regards to hand detection, so it simplifies our solution approach as well as more easily discretizes our task for hand recognition. Secondly, as we are using a neural network on the hand image for gesture recognition, the computer vision approach standardizes and feeds more seamlessly into our system pipeline. The specific library we'll be using is the MediaPipe body landmark recognition library. The library includes a pretrained model that includes facial and hand detection algorithms on images fed through a camera and allows for us to tune parameters like detection confidence and maximum number of hands to detect. With respect to the Leap Motion Controller, a computer vision approach will allow us to increase the distance with which our user can operate provided our camera has a high enough resolution. To combat low confidence detection due to motion blur when a user moves their hand in the camera view, a camera with a higher frame rate refresh will be used as well. The high frame rate camera (60 frames per second) will also meet our requirement for 50 ms latency. We also decided on a webcam because it is likely that our users will have access to their own webcam that they can use with their own laptop to run our system. Practically, it is unlikely that users have professional or super expensive cameras handy to use to get our system working on their computer.

B. Gesture Recognition

Classification of user hand gestures from image data is a perfect fit for a machine learning approach. Finding a function to discriminate between the number of hand gestures required to meet user product specifications would not be feasible by hand. The requirements for this users' product specifications of this gesture recognition include quick training of our model and low inference computational latency, as well as high accuracy. Considered approaches included deep learning in the form of a deep convolutional neural network applied to raw image data or a simpler architecture neural network that would be provided feature data in the form of landmark coordinates output by our hand pose estimation. Both approaches would be implemented with a simple supervised learning approach employing stochastic gradient descent on a multiclass cross-entropy loss function. Unsupervised learning approaches were found unfit for this product, due to the extended learning time required, as well as the saturation of available appropriate datasets including images of various hand gestures.

The final design selected is the latter of the two options: the simple neural network trained on pose estimation data. This selection was made because the simplicity of the model would allow for a much shorter inference time, as well as a predicted higher accuracy than what could be achieved from raw image data that would include background pixels, pixel alpha differences from lighting, as well as overall higher noise. This meets the requirements for a smooth user experience by lessening overhead for latency and improving classification accuracy for gesture recognition accuracy.

C. OS Interface

The OS interface and cursor location transform will be implemented in Python. Python will allow for easy connections between our design components and easy transfer of information between modules. The control of the mouse through the OS will be done using the mouse library in Python. Other libraries that could accomplish the same task of controlling the cursor include pywin32 and pyautogui. All these libraries would fit the user product requirements since they would all be able to interface with and control all aspects of the cursor with minimal overhead. However, the mouse library has excellent user-friendly wrappers that are much easier to work with than the functions in the other libraries.

We chose to use a laptop rather than something like an RPi (Raspberry Pi) because we felt that this fit our user product requirements better. The goal of our Virtual Whiteboard is to allow the user to control their own computer from a distance using hand gestures. Having on the system run on an RPi may be sufficient and cheaper for demonstration purposes, but the goal is for users to be able to run the code on their own computers and use their own webcams to utilize the system on their own. Therefore, ensuring that the system can function on one of our laptops using a webcam is what we want to verify that our system fits what we want it to do.

V. SYSTEM DESCRIPTION

A. Hand Detection

Our hand detection and pose estimation system will be implemented in a couple discrete steps. The MediaPipe hand landmark data structure for detection contains parameters x, y, and z. The first two parameters, x and y, represent the on-screen image location of the landmark wrist on the image coordinate system. The last coordinate represents the depth of the landmark detected which gives us a rough estimate of distance to camera scaling. In our overall system process, we will first have a manual calibration phase where the user will indicate their full comfortable range of motion. Given the x and y coordinates of the calibration phase, we will fit a bounding box that has dimensions of the interaction screen to the person which will serve as a mapping between the hand location in reality to on screen coordinates where we will put the mouse. Specifically, we will store the hand location within the bounding box as a tuple of distance along horizontal and vertical axes, and it is this information that we will then use in the OS/UI interaction phase to indicate where along the horizontal and vertical screen axes we want to put our mouse. As our included functionality allows the user to move in both vertical and lateral directions, we will scale the bounding box initially calibrated to the user to follow their positional change.

A thing to note in our initial testing with the MediaPipe hand landmark structures is that hand detection confidence can fall off depending on distance to the camera and other conditions like light over and undersaturation. Further testing is required on the mounted camera we bought that should fit our specs. If detection is not optimal at further distances, we will likely need to preprocess our image data through methods like histogram equalization for light exposure or implement some sort of camera tracking/zooming functionality to our pose estimation system to track the current user.

B. Gesture Recognition

Gesture recognition will be implemented in two stages. Initially a pre-designed model will be slightly altered for our purposes and trained with existing Jupyter notebook scripts to allow for the rapid creation of a functional model that can be used for testing and integration with other parts of the system. Following this, the pre-designed model will be altered further to a lower number of classified gestures and extended with two extra layers that will be initialized by values copied from the previous two layers. All weights and biases will be frozen save for these final two layers, which will be more extensively trained and tweaked to hopefully achieve higher classification accuracy than the pre-built model. The gesture recognition will be trained on AWS with PyTorch.

The gesture recognition module will take in enhanced hand images from the hand detection module. The trained model will be fed these images as frequently as possible. The outputs of the model classification will be converted to integer output and fed into the OS interface module.

C. OS Interface

As mentioned before, the OS interface, which includes the cursor location transform, takes inputs from the hand detection and gesture recognition modules, and outputs simulated cursor actions for the OS to execute. The OS interface is implemented in Python and mainly makes use of the Python mouse library.

Firstly, the cursor location transform function is implemented by taking in hand coordinate data from the hand detection calibration block. This calibration step will be implemented as follows. The calibration code will be called from the command line and executed, giving the user 10 seconds to move to the position they want to use the system from. This code will continue to accept hand coordinate inputs for about 15 seconds in which the user will move their hands through their complete range of motion. The cursor location transform function will record the maximum and minimum x and y coordinate values received within this time frame. These x and y values will be used to construct a rectangle that represents the user's range of motion. This rectangle will then be scaled to 75% of the screen's size. Screen resolution is obtained using the ctypes library to directly obtain screen size using the GetSystemMetrics() method. After the calibration, all hand positional changes will be scaled based on this calibrated rectangle of motion.

While the system is running, the cursor location transform will continue to receive hand position coordinates and convert them to coordinates on the screen. The module will keep track of the currently received position as well as the most recent position. The x and y coordinates of these positions will be subtracted, and the result put into the mouse library move function. The function mouse.move takes in x, y, absolute, and duration or steps_per_second as input. The absolute input tells the function whether the x and y values represent set coordinates on the screen or if they represent the change in x and y that the cursor must move. This is an important distinction because our cursor movement is all relative since we are not scaling the range of motion to cover the entire screen. If we were to try to implement absolute cursor movement, the user would likely struggle to reach the corners of the screen as they would have to extend their arms as far as possible. The duration and steps_per_second inputs control how frequently the movement updates happen. This will be synced to as fast as hand coordinate inputs are received to keep the overall system latency consistent. The cursor location transform handles all of the cursor movement.

The other mouse operations including clicking and scrolling will also be implemented using the mouse library and will use the gesture recognition output. The integer representation of hand gesture will be cased on, and different mouse functions will be called corresponding to what was detected. An important distinction here is the use of mouse.press and mouse.release instead of mouse.click. Press, release, and click function exactly as they sound like. Since the gestures will also be continually fed into the OS interface, there is no need to use the mouse.click function. The act of clicking will be done by changing the hand gesture and then quickly changing it back, essentially flashing the hand gesture. We do this so that clicking and holding can be implemented in a similar manner, where clicking is just holding and releasing under a very short timeframe. Currently this implementation choice is envisioned to be sufficient, but there may be problems in the future where clicks are interpreted as tiny holds and so the user may accidentally drag things on their screen instead of clicking. If this does happen, the mouse.click function will also be used and a counter can be implemented that measure the duration of the hold hand gesture in order to differentiate between clicks and holds. Scrolling of the mouse will be implemented with the mouse.wheel function. Any hand motion while the scroll gesture is detected will result in scrolling instead of mouse movement.

VI. Test and Validation

Since this is still our design, below we have outlined the tests we intend to carry out while we complete our implementation. We will have a general user story test where we measure all the metrics. This general user story will essentially walk the user through using the system and getting them to try and open a web browser and navigate through to the CMU website. There will also be other more specific tests for some of the design requirements.

A. Latency

To test latency, we will use our camera to capture both our user and the computer screen. Since the camera captures 60 frames per second, it will be able to see if we cross the 20 frames per second and 50 ms latency threshold. However, we will likely not be able to get an exact measurement down to the precise millisecond. These measurements can be done while the user runs through the other tests.

B. Gesture Recognition Accuracy

The gesture recognition accuracy will be measured by counting the number of gesture changes required in the user story and recording the number of errors that we observe or that the user reports. The number of errors divided by the total gesture count will be our gesture recognition error rate. Subtracting this from 100% will give us the gesture recognition accuracy. We will accumulate this value from many different iterations of the test with different users.

C. Cursor precision

The cursor precision will mostly be measured through an online cursor accuracy application. The user will try to control the cursor and hover over small red circular targets that are 30 pixels in diameter in a limited timeframe. We will record the score of these tests and run the test with varying sizes including smaller and larger targets. The accumulation of these results will give us the results for our cursor precision.

D. User Survey

We will also include a small user survey after they use our system to get a qualitative measure of how smooth the user thinks our system is. We will tell the user to rate their experience using our system on a scale from 1-10, with 1 being very difficult to use and 10 being perfect with no hiccups. This validation does have a quantitative score aspect, but it is more qualitative since it is based on user opinion and feedback.

VII. Project Management

A. Schedule

Our schedule was designed based on team member responsibility and how our different modules connect. The hand detection module and the gesture recognition module can be developed largely in parallel before needing to be tested together near the end. The OS interface can also largely be worked on in parallel and would just need to change the potential inputs based on modifications that are made to the hand detection and gesture recognition outputs. The schedule also leaves a lot of time near the end of the class to focus on integration and getting all our modules to work together. This time at the end also allows for lots of time to verify and test our system.

B. Team Member Responsibilities

Alan's main responsibility is to develop the OS interface module. He will develop most of the code in Python to interface with the OS and cursor. Since Alan's part largely relies on receiving inputs from the other modules, he will be designated some lighter tasks in earlier weeks to help the other team members with getting their components up and running. Although not explicitly depicted on the schedule, Alan's secondary responsibility will be to aid both Andrew and Brian with the hand detection and gesture recognition modules respectively, especially in the first weeks. In the last few weeks, he will work with the whole team on integration and testing of the entire system.

Andrew's main responsibility will be to develop the hand detection module. He will be responsible for turning the image data received from the camera into hand position data for the OS interface as well as sending enhanced and cropped hand image data to the gesture detection algorithm. Since his responsibilities overlap with the gesture detection, his secondary responsibility will be to work with Brian and ensuring that images are properly sent from the hand detection module into the gesture detection module. In the last few weeks, he will work with the whole team on integration and testing of the entire system.

Brian's main responsibility will be to develop the gesture detection module. He will be responsible for training our neural network model to receive image input and produce gesture classification output. Since the gesture detection module is largely dependent on the hand detection module, Brian's secondary responsibility will be to work with Andrew and ensure that the proper images are sent into the gesture detection module. In the last few weeks, he will work with the whole team on integration and testing of the entire system.

Budget

As shown in table 1 below, our budget is only used for our camera to capture image data and our AWS credits that will be used to train and run the neural network gesture recognition model.

Description	Model	Manufacturer	Quantity	Cost
Camera used to capture image data	C922x	Logitech	1	$99.99
AWS Credits	N/A	Amazon	3	$150
Laptops for running software components	Varies	Varies	1	$0

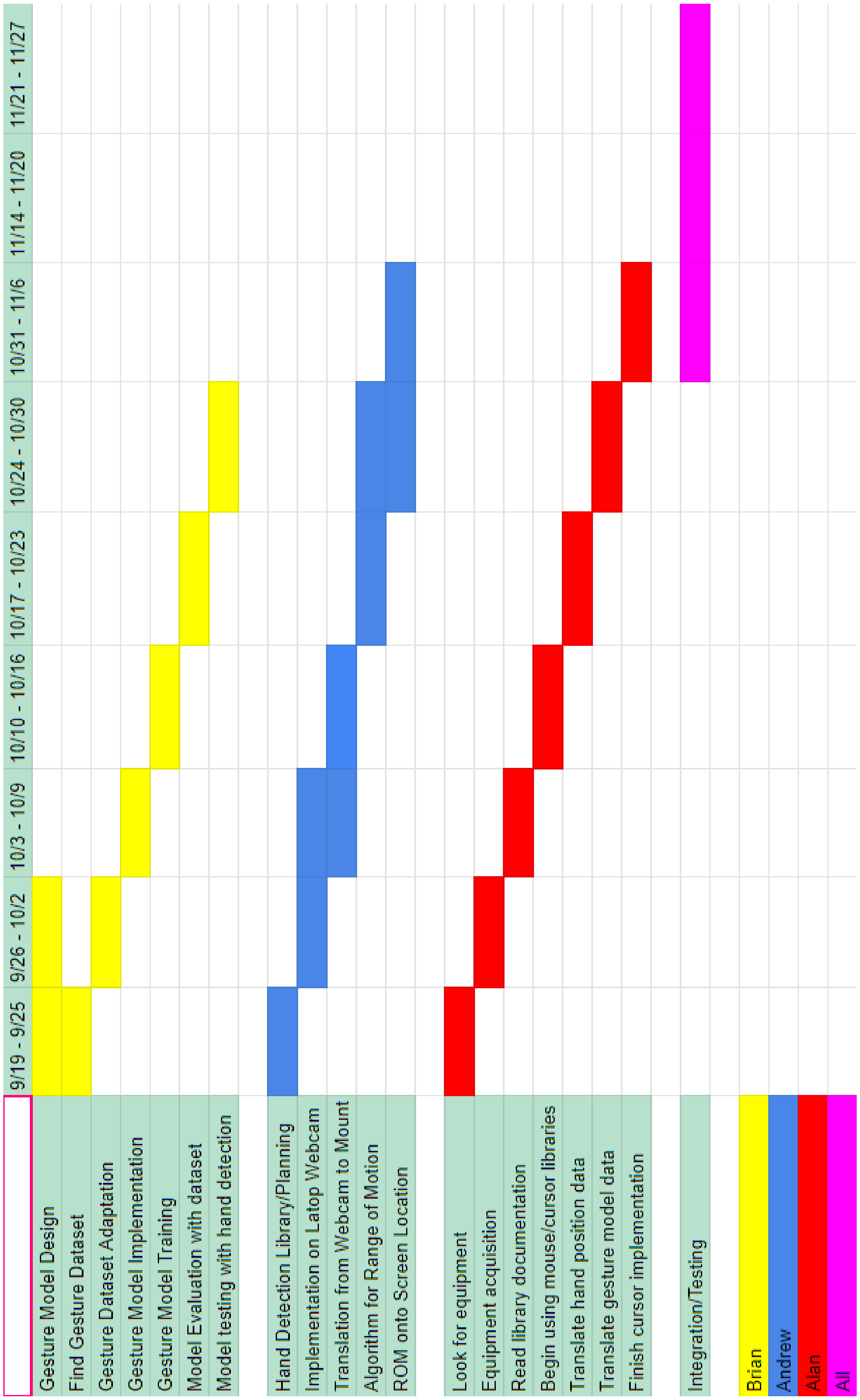
Table 1. Bill of Materials

C. *Risk Management*

The biggest risk in our project is the integration of all of our individual components and the final product meeting all of our design requirements. We foresee that integration will likely be a tough challenge, and so we have allocated sufficient time in our schedule to focus on this aspect. We decided to do a lot of our individual development in parallel so we could all come together at the end to sort out problems that came up during integration and testing. The risk of not meeting design requirements is also largely present since we are developing our modules in parallel, so even if individual testing and verification passes, once the entire system comes together, we may run into additional issues with meeting our requirements. Our resources should not be a point of risk at all since we chose a project that requires simple resources. However, this means that a lot of the success of the project will fall onto our individual responsibilities of developing the proper software. By following the proper process that we were taught to in this course, starting from the proposal to the design review, we can think about all these risks ahead of time and plan/schedule accordingly.

Glossary of Acronyms

OS	Operating System
RPi	Raspberry Pi



	9/19 - 9/25	9/26 - 10/2	10/3 - 10/9	10/10 - 10/16	10/17 - 10/23	10/24 - 10/30	10/31 - 11/6	11/14 - 11/20	11/21 - 11/27
Gesture Model Design	🟨	🟨							
Find Gesture Dataset	🟨								
Gesture Dataset Adaptation		🟨							
Gesture Model Implementation			🟨						
Gesture Model Training				🟨					
Model Evaluation with dataset					🟨				
Model testing with hand detection						🟨			
Hand Detection Library/Planning	🟦								
Implementation on Latop Webcam		🟦	🟦						
Translation from Webcam to Mount			🟦	🟦					
Algorithm for Range of Motion					🟦	🟦			
ROM onto Screen Location						🟦	🟦		
Look for equipment	🟥								
Equipment acquisition		🟥							
Read library documentation			🟥						
Begin using mouse/cursor libraries				🟥					
Translate hand position data					🟥				
Translate gesture model data						🟥			
Finish cursor implementation							🟥		
Integration/Testing							🟪	🟪	🟪
Brian 🟨									
Andrew 🟦									
Alan 🟥									
All 🟪									

Fig. 2. Detailed schedule