Magic Mice

Author: Bradley Zhou, Jade Wang, Jenny Han: Electrical and Computer Engineering, Carnegie Mellon University

Abstract—Magic Mice is a small projector laptop attachment that will display a laptop screen on a wall from 5-15 ft away and includes two main interactions for the user and the computer: a linked pen and a series of customized body gestures. This will simulate the functionality of a regular laptop but as a "tablet on the wall" and the user can interact with the laptop's wall projection as if it was a tablet. Additionally, the user has a series of customizable hand gestures that will map to Chrome macros such as refresh or previous page. This makes Magic Mice an affordable collaboration tool for team workers anywhere.

Index Terms—Hand Gestures, Laptop, Machine Learning, Open CV, Projection, Sensors

I. Introduction

The primary goal of Magic Mice is to turn a laptop screen into an interactive wall projection that will enhance the user's experience and engagement online. Named Magic Mice, this is a small projector laptop attachment that will display the screen contents on a wall from 5-15 ft away and include two main interactions for the user and the laptop: a linked pen and a series of customized body gestures. Being able to dynamically size an interface and interact with a display gives users the freedom to customize their online experience in a way that best fits their needs, whenever and wherever.

With the pandemic, most education systems have been forced to be conducted online which only makes the learning process more complicated. Teachers in more wealthy areas will use SmartBoards or a tablet to physically draw and write with a SmartPen and then screen share that to whatever video calling platform they are on. However, not all schools can afford supplying these devices to every staff member.

Thus, Magic Mice solves two problems. The first is natural collaboration between teams and the second is price point. Magic Mice will allow users the collaborative experience that the Smartboard competitors have and the portability of a screen sharing tablet. Magic Mice will report the screen contents onto any flat wall and the user will be able to interact with the projection using a pen and hand gestures. This will update the system and allow the user to collaborate with anyone freely. Finally, by only needing a pen device and projector for hardware pieces, the device is much cheaper than a Smartboard or tablet, which can cost up to 8000$ and 1000$ respectively. This makes Magic Mice a more appealing collaboration tool.

II. Design Requirements

To meet our design requirements defined in Table 1, we will have a series of systematic unit tests between each branch of code logic to ensure that all parts are reaching the metrics that we set for them. Here, I will highlight how each software component is being tested.

For our pen component, we will be testing the gyroscope and accelerometer input data and verifying that the data transmitted through the Bluetooth is the correct output dictionary of values that we want. We'll be setting 100 locations on the screen to test, a grid of 10x10 points evenly dividing the screen. The pen will be able to click within a 95% accuracy rate and click within a 5 pixel range from the actual point. This accuracy rate was chosen to leave room for error but high enough such that it does not affect the user experience. 5 pixels was chosen because that is a reasonable sizing for a web button. For the physical buttons on the pen, we'll be testing single clicks on both buttons, a click and hold for drag, and hovering. This will be done by tracking the pen movement over tracing a drawing on sketch.io, a sketching website on Chrome.

For the hand expressions, we'll be testing our gesture recognition algorithm by verifying our laptop camera input maps to the correct hand expression. We'll be splitting our original marked data set into a 20 to 80 for a test to train dataset ratio. We will be running the trained model on a 20% test data set and strive for an accuracy of >95%. As for new hand expressions that aren't in the library already, we'll verify that the system is correct 9/10 times. We will do this by taking 10 videos of 10 different people doing the hand position. These videos will be inputs into the algorithm and would need to be matched correctly 9 times. We will repeat this for 10 hand positions outside of the default. The process of 10 different people doing the same hand movement for the test will give enough variation to test the accuracy of the system.

As for our projector, we'll be testing all distance ranges from 5 - 15 ft in 1 ft increments to ensure calibration works at all reasonable distances. 15 ft was chosen as the max distance because it is slightly greater than the midpoint of an average room. Thus, users could use the projector at any point in the room depending on orientation. We'll be making sure that a 2px thick vertical line has clearly depicted edges on the projected wall. 2px was chosen because it was thin enough to potentially be blurry in a low resolution projection screen but thick enough such that the user would be able to see it. If the user does not see this 2px stroke, then it will be difficult to see.

TABLE I. Design Requirements

Component	Requirements
Pen	- >95% accuracy for location (x,y,z location and x,y,z velocity) - >95% accuracy for button to click conversion
Gesture Expression	- >95% accuracy for model on specific hand motions (fist, palm, OK hand sign) - >95% accuracy for custom hand motions - <100 ms to classify hand motion
System (GUI)	- Customize 5 gestures - Default 3 are fist, palm, OK hand sign
System (Software)	- Mouse events translate at the correct spot on the computer - <200ms latency between click and update - Chrome browser interactions and Desktop file manipulation

III. Architecture and/or Principle of Operation

Our system takes in two methods of human input, hand expressions and control of a custom sensory pen.

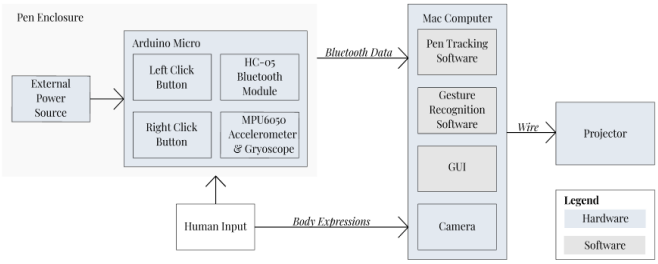

Fig. 1. Overall System Block Diagram

As shown in Figure 1 above, the sensory pen is made of an external power source and several Arduino parts that are used to sense inputs. Two push button switches for left and right mouse clicks, an BNO085 to sense accelerometer and gyroscope coordinates for pen location, and the HC-05 Bluetooth module. The HC-05 Bluetooth module communicates all of the sensor data to the Mac laptop where all data processing is done.

The sensor data is initially read as Arduino data every 200 m, and communicated to the laptop through an HC-05 Bluetooth module. This data is read as a dictionary of Python values by a pen tracking software. This software outputs respective mouse control of location, left, and right click button actions. Additionally, hand expressions image inputs are captured through the Mac laptop's built-in camera. A gesture recognition software processes these image inputs and maps them to their respective user macros. Pen mouse control outputs and user macros are effectively executed on the Mac laptop. These actions are then reflected onto the projector screen through a physical wire connected to the laptop.

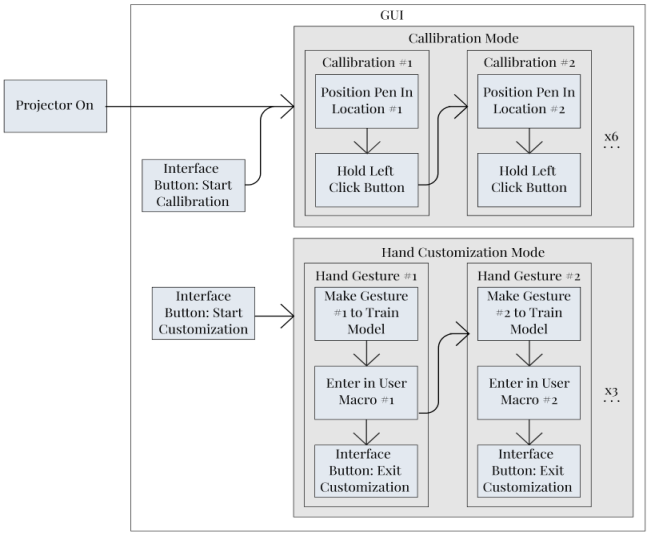

Fig. 2. GUI System Flow

The projector system is calibrated through a Mac GUI application. Before the custom sensory pen can be used, the projector must be turned on and placed at the desired distance away from the wall the laptop's screen will be projecting to. The calibration mode is activated through an interface button as shown in Figure 2. The pen is placed on 9 different locations around the border of the projected wall area shown in Figure 3 -- all 4 corners of the area, the center of the area, and the middle of each surrounding side.

Additionally, the GUI has a hand customization mode in which the user can set custom hand gestures to map to any keyboard shortcut. As shown in Figure 2, once the customization mode is entered via another interface button, the user will must make their first custom hand gesture for several seconds to train the model to recognize the gesture. The user will then have an option to exit the interface mode and or continue customizing the next hand gesture. This process will occur 3 more times for a total of 5 customizable gestures.

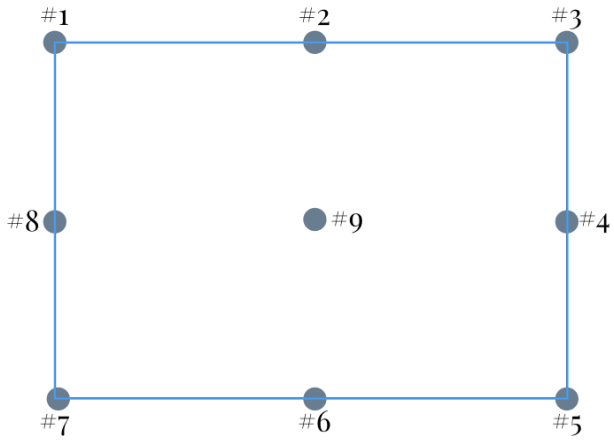

Fig. 3. Callibration points for projected screen on wall

IV. Design Trade Studies

A. Arduino Model

The model that we ultimately decided to use is the Arduino Pro Micro. We considered various models. We first looked at the de facto standard for many Arduino projects: the Arduino Uno. The limiting feature of this board is the fact it uses the ATmega328 microcontroller. This microcontroller does not have any native USB capability. USB capability is important because it allows the Arduino to be recognized as a mouse/keyboard input device. This gives us great flexibility and more options in actually creating the mouse movement events to pass through to the operating system. While the Uno does not support USB capability, the Arduino Leonardo and its ATmega32u4 does. In fact, all of Arduino's boards with the ATmega32u4 microcontroller have this desired functionality. Though the Leonardo has all the desired technical functionality we needed, it did not have the ergonomic qualifications we were looking for. Having a width of 53.3 mm, the Leonardo was much too wide to comfortably shape into a pen/stylus form. The width of an average pencil is 6 millimeters. Of course, our pen/stylus will be wider than a pencil, but holding a component the size of nine pencils in one's hand is excessively cumbersome. We then looked to the Arduino Pro Micro. The Arduino Pro Micro has the same technical specs as the Arduino Leonardo; they both use an ATmega32u4 microcontroller running at 16 MHz, have 32 KB of memory, and 2.5 KB of RAM. Having all the same technical functionality, including the desired USB capability in a much more compact size and a width of only 18 millimeters checked all the boxes we needed for a main board. The width is only three pencils, which sits significantly more comfortably in the average person's hand compared to the wider Arduino Leonardo

B. Accelerometer Model

We were considering two models of accelerometer/gyroscopes - either the BNO085 or the MPU6050. Both were sourced from Adafruit. The BNO085 is more fully featured, having an accelerometer, a gyroscope, and a magnetometer compared to the MPU6050 having just an accelerometer and gyroscope. Additionally, the BNO085 reports multiple forms of sensor data (calibrated/uncalibrated/raw ADC values). We considered the MPU6050 because it would have been simpler to work with as well as is significantly cheaper, which is one of the points we wanted to emphasize about our project. Ultimately, we decided on the BNO085 because cost is secondary relative to the other aspects of our project and having the most data to work with will help immensely in development.

C. Form Factor

The form factors we considered the most are a pen/stylus shape (like an apple pencil) or a glove with the user's fingers being the main point of tracking. The glove is a more novel and unique design that is more ergonomic as it allows the user to have his/her hands/fingers fully relaxed compared to having to actually stress muscles to grip a pen, which could lead to fatigue. A pen shape is more classic and there are many preexisting analogous examples, such as an apple pencil or a smart board pen. We decided to go with a pen shape due to the simpler implementation, the lack of ubiquity for a glove shape (people have different sized hands), and the fact that a pen can allow for more natural drawing click/drag functionality, which is useful in scenarios such as drawing.

D. Pen Tracking Software Architecture

One of the biggest points of contention we had discussed was how to organize the software functionality. We were considering three different architectures for our pen tracking software. The high level ideas we had to work around are the reading of the accelerometer sensor data, the processing of that data to make a decision on where to move the mouse, and the actual translation of that decision to an OS mouse movement event. For each of these pieces of functionality, we had to decide whether we wanted the Arduino and its microcontroller itself to be responsible or the user's computer and its much more powerful CPU to be responsible. The three options we considered are listed below:

Option 1: Have the Arduino itself perform all of these functions. The Arduino will internally keep track of the cursor's position, read the sensor values directly from the accelerometer attached to it, update the internal representation of the cursor's position, and directly translate that into a mouse movement event that is then sent directly to the user's computer via USB.

Option 2: This option is similar to the first option except for the fact we will have two Arduinos instead of one. The purpose of the second Arduino is to act as a Bluetooth receiver. One Arduino will be used as the pen/stylus and it will wirelessly (via Bluetooth) send its sensor data to the second Arduino. The second Arduino will be plugged into the user's computer via USB and will be the one that internally keeps track of the cursor's position as well as using the sensor data received over Bluetooth to update the cursor's position in real time. This Arduino also has the responsibility of turning the cursor position movements and translating them into OS events that are then sent via USB.

Option 3: We only have one Arduino and its only responsibility is to send sensor data over to the user's computer via Bluetooth. The user's computer will be running a user space program that will perform the functionality of keeping track of the cursor's position, receiving the accelerometer data, using the data to update the cursor's position, and finally translating that to an OS event to move the mouse.

Option 1 was the first one that we considered but we had to pivot away from it due to the restriction of being limited to wired USB. While it leads to a logical separation of concerns (all the code related to the pen movement is run on the pen itself) and supports the idea that the pen itself is a standalone input peripheral, it did not give us the Bluetooth wireless functionality we specified in our requirements. Option 2 was what we considered next. This architecture is commonly seen among wireless mice and keyboards that have a USB dongle required to use. This still gives us the benefit of having a logical separation of concerns, but has a higher development

complexity. There are more points of failure with this approach. Another downside to this method is the higher response time due to having another point of connection for data communication. Option 3 has multiple benefits. The biggest benefit is that the user's computer would be the system running most of the more intensive parts of the code. Macbooks have a 2.0 GHz CPU, which is a clock speed of over 100x greater than the Arduino's microcontroller. Given that we are aiming for a 125 Hz polling rate, this means that each update (reading in sensor values and updating cursor position) will have to be done in 8ms. Having the most powerful CPU possible running the hot path of the code will ensure that we can reach this polling rate requirement. A CPU running at 2.0 GHz means that each update will have

$$2 * 1024 * 1024 * 1024 / 125 = 17.18 \text{ M} \quad (1)$$

17.8 million clock cycles to perform all the calculations necessary. Additionally, because we have multiple subsystems at place (pen tracking and gesture recognition), consolidating all the code to run on the same platform is also a benefit, though this benefit is more subjective and one that we decided we valued. Ultimately, we decided to proceed with option 3 due to the aforementioned benefits.

E. Gesture recognition image processing library

We first considered using OpenPose, a CMU developed open source tool to detect body positions in real-time. We considered this because one of our team members has experience working with the tool in research projects and supports our real-time requirements. As we researched more, there were some roadblocks. First, OpenPose is very computationally intensive, requiring a powerful NVIDIA GPU to efficiently work in real-time. Given that our platform is meant to run on laptops, our workaround would have to be sending the video feed over to a remote machine that is running OpenPose, which although possible, can have reliability issues depending on the user's network. Another issue is that OpenPose exists more as a standalone tool and integrating its output into ML models to actually detect gestures is unintuitive. Given all these limitations, we looked to OpenCV next, which is more lightweight. Upon research, we discovered that while OpenPose provides much more detail, the extra detail is not necessary, as gestures can be reduced to shapes that can be extracted from grayscale images. OpenCV sacrifices data detail for much more portability. OpenCV also provides a simpler interface for integrating with machine learning: reducing the image to just grayscale sequence of pixels and passing that information through to neural networks is a very common approach for recognition models (handwriting/shape/etc). We ultimately decided these features were more important, so we moved forward with OpenCV.

F. Gesture recognition machine learning library

There is a large variety of open source machine learning libraries for Python: TensorFlow, Keras, Scikit-learn, and PyTorch, are all options that we saw upon research. Machine learning is no one in our group's specialty, but we all have a reasonable understanding that allowed us to reduce our problem to transforming a live video feed into grayscale images that can have shapes extracted from them using neural networks (pixel brightness values as inputs, whether or not there is a particular shape in the image as the output). This is a relatively surface level and more basic application of machine learning, so we chose the framework accordingly. It is not necessary to have super fine-grained control over every feature and layer of the model. The most important feature is being easy to work with while providing high performance neural network interfaces. Keras checked all these boxes, being less verbose than other libraries and frameworks while accomplishing the same tasks. Because Keras is built on top of and wraps TensorFlow, we chose Keras as our library of choice.

V. System Description

Hardware System Description

A. Arduino Pro Micro

The Arduino Micro serves as the main microcontroller board of the pen that connects all of the Arduino sensors and components together. It was chosen for its small and manageable size and laptop mouse compatibility. The Arduino Micro is based on the ATMega32U4 board with built-in USB communication, allowing the Micro to appear to a connected computer as a mouse.

B. BNO085

The *BNO085* is both the accelerometer and gyroscope sensor. It was chosen for its Arduino compatibility and manageable size. The sensor also encapsulates three sensor functionalities in one. The accelerometer is a 3-axis accelerometer that gives us coordinates about which direction is down through measurement of gravity. The gyroscope measures data on how fast the sensor is being twisted around. Finally, there is a magnetometer. This combination of these data types allows for calculation of how our sensor pen is moving through space.

C. Buttons

The sensory pen has two switch push buttons in use. A button on the left side of the pen mimics the 'left click' on a laptop mouse, and a button on the right side of the pen mimics the 'right click' on a laptop mouse. All clicks of the buttons and push/hold of the buttons are detected.

D. HC-05

The HC-05 Bluetooth module was selected as the mode of communication between the pen and software program. The Bluetooth module aggregates all of the sensory data through an Arduino script and sends all of the data to the laptop via Bluetooth communication. Bluetooth served as a simple way for us to avoid using wires for a better user experience. This specific module was also selected for its Arduino compatibility and small size.

Software System Description

Python and C++ will be the main languages we will be using for our project. This is because Python has support for all the features we plan to implement (external mouse manipulation on

Mac, OpenCV, machine learning algorithms etc). Also, all team members are proficient with Python and C++. As for speed, we believe that python will be sufficient within our requirement specifications. The slowest element would be the hand gesture recognizer, in which case we can sample the image inputs to speed up the program.

Figure 5 is the system specification for the software component of our design.

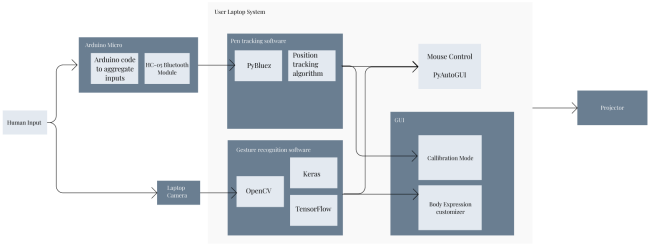

Fig. 4. Software system specification overview

As described in the figure above, most of the major computation will be done on the users Mac laptop. The laptop software system consists of a system/GUI, the gesture recognition software, and pen tracking software.

A. GUI

The GUI will be used for pen calibration, and customization of Mac laptop commands. This is what the user will interact with to set their own customized hand gestures to map to specific actions and manipulate the current dictionary of gestures. Additionally, the user can set the system to a calibration mode in case the pen click has an offset. Thus, if the pen is not synching correctly the user has a way to recalibrate.

B. System

The system will take the inputs from the gesture recognition (hand position) and pen software (click type and coordinates) and calculate the corresponding action. For gesture recognition, the system will need to map the hand position to a pre-saved Mac user macro or an error. For the pen output, our Python program will read in the Bluetooth data using the PyBluez library, a fairly commonly used Python Bluetooth API. The system will need to decipher what to do based on the click type, such as hover, long press, scroll, click, and perform that action at the correct corresponding location of the computer. After this, the system will update the projector to update the display that the user sees.

C. Gesture Recognition

The gesture recognition software will be used to input the user's Macbook camera images to specific gestures. We will do this by using OpenCV to identify hand gestures. Additionally, we will be using Keras and Tensorflow to make use of convolutional neural networks, a common type of machine learning for mapping of image inputs to an output variable -- which in our case will be user macros, such as browser page refresh, or going back a page. These will also be implemented with the same PyAutoGUI library.

D. Pen

Finally, the pen will also have a software component. This is the only component to run code separate from the laptop system. For the pen, we will have Arduino code written in C++ that aggregates all of the sensory input. This includes left and right button clicks, accelerometer data, and gyroscope data. The HC-05 module will then take all of the data and send it to the Mac laptop via Bluetooth.

VI. Project Management

A. Schedule

Figure A1 of the appendix shows our schedule for the project. In this schedule, we broke down our tasks to smaller pieces. We left time buffers in the schedule for error and testing. Also, we tried to combine simultaneous elements at the same time.

B. Team Member Responsibilities

There are a few key hardware and software elements in the design of Magic Mice: the pen, the hand gesture recognition, and the overall system. Since these three elements can be done mostly independent of each other and then fitted together at the end, we decided to split tasks this way. Each team member is responsible for the capability of a specific section, the success of that element is checked by another team member, and everyone helps in the case the team member has issues bringing the design to fruition. This way, a team member should be a specialist in a specific area of the project. Since hand expressions are the more complex element of the project, we split the testing more finely for this element.

Jenny will be in charge of the hand gesture recognition software. She will do this by linking the webcam of the computer to the algorithm and working on the gesture recognition software using open CV. She is in charge of testing the pen requirements.

Jade will be in charge of the overall system requirements. This means she will help make sure all the input elements communicate with each other and update the screen accordingly. She will also relay information to the projection on the wall. Jade is in charge of testing the hand expressions.

Bradley is in charge of assembling the hardware elements for the pen and writing the pen tracking and calibration software. He will be in charge of testing that the system updates according to inputs given. Also, he will be in charge of testing hand expressions

C. Budget

The spreadsheet of materials and costs can be found at the end of the report in Figure A2 of the Appendix.

D. Risk Management

We anticipate that parts will be broken as we iterate on our hardware design. Thus, we have ordered several backup parts in case some break. Additionally, we will be using version control and Github to manage our iterative changes and collaborative efforts.

VII. Appendix

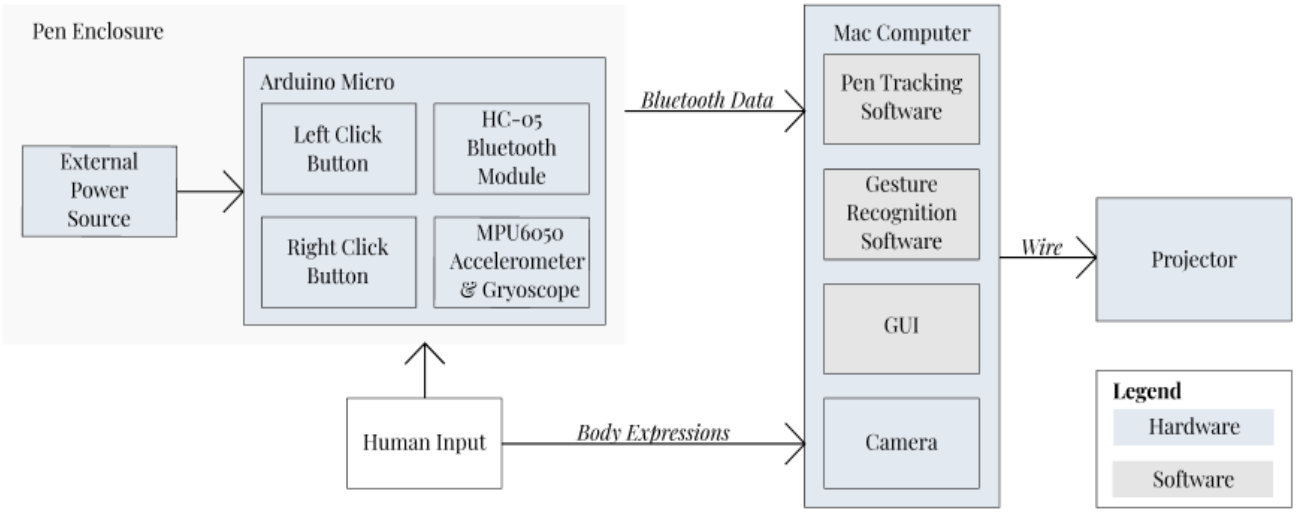

Fig. 1. Overall System Block Diagram

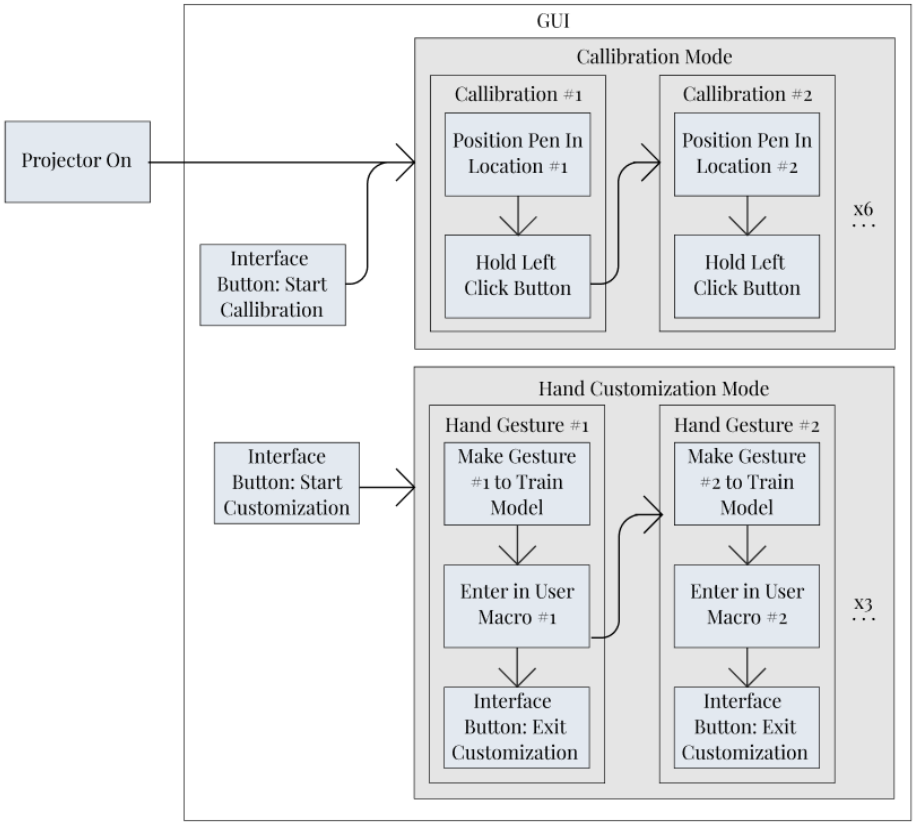

Fig. 2. GUI System Flow

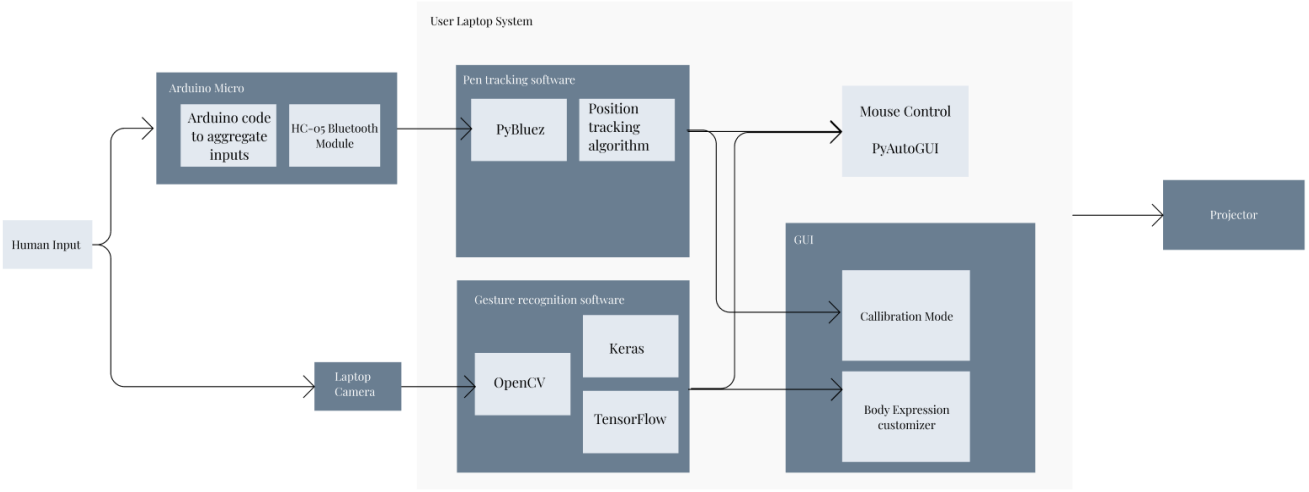

Fig. 4. Software System Specification Overview

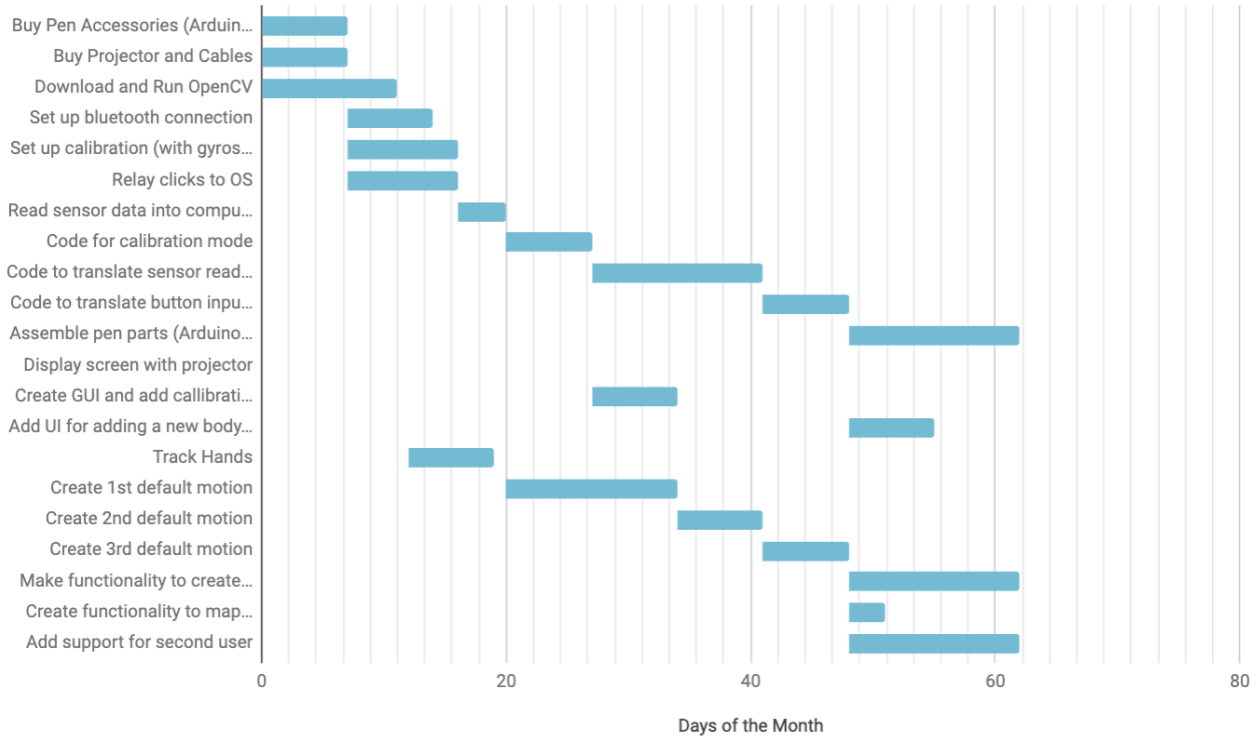

Fig A1. Gantt Chart

	Component	Link	Individual Cost	Quantity	Total Cost (tax + shipping)
2	**Pen**				
3	Arduino Pro Micro	https://www.amazon.com/Arduino-	20.7	2	43.88
4					
5	MPU6050 Accelerometer/Gyrosco	https://www.adafruit.com/product/3	6.95	2	26.21
6					
7	BNO085	https://www.adafruit.com/product/4	19.95	2	42.29
8					
9	HC05 Bluetooth module	https://www.amazon.com/HiLetgo-	7.99	2	16.94
10					
11	Buttons	https://www.amazon.com/Gikfun-6	5.98	1	6.34
12					
13	Wires	https://www.adafruit.com/product/4	0.95	3	3.02
14					
15	Breadboard	Sourced from ECE lab			
16					
17	**Projector**	https://www.amazon.com/Upgrade	59.99	1	63.59
18					
19					202.27

Fig A2. Budget