InteracTable

Author: Suann Chi, Isha Iyer, Tanushree Mediratta : Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*— This paper outlines the preliminary design of our capstone design project, InteracTable. InteracTable is a proof of concept prototype of a portable system that can turn any surface into an interactive touch screen. Current commercially available capacitive touchscreen tables are very costly and are immobile [1]. Our design overcomes these limitations by using computer vision algorithms to track the location of a user's finger and piezo sensors to detect the vibrations propagating through the medium from a tap on the table.**

***Index Terms*— Algorithms, Collaboration, Computer Vision, Detection, Interactive, Piezo Sensors, Tracking, Touch Screen**

I. Introduction

INTERACTABLE is a portable product that will revolutionize the way people work. Any table surface can become a user's laptop screen while using the InteracTable. The size of the work surface will be adjustable to different tables, and best of all, it can accommodate multiple users for the best collaborative experience. InteracTable's design consists of a projector connected to the user's laptop, whose screen will be projected onto a flat table top. Computer vision will be used to track a red dot sticker placed on a user's finger. The projected screen will be captured by a webcam and these images are then sent back to the laptop for processing. A piezo sensor connected to an Arduino will detect a tap on the table. For this proof of concept prototype, we will constrain the system to work with only one finger. Our system must detect the location of a red dot with 100% accuracy. Must map the detected location of the red dot in the projected image to the laptop screen dimensions within the radius of the red dot - 0.64 cm. Must identify a tap on the surface with 100% accuracy in order to achieve a usable system that works in real time. Must achieve a system response time of 1 second.

There are existing solutions to the problem of collaboration including Google Drive and capacitive touch screen tables. However, these solutions do not expand to all applications that engineers tend to use in a collaborative setting. Our product is meant for software that is not inherently collaborative in nature such as most code editors, offline Microsoft products and browsing the internet. Further, touch screen tables also do not serve as a good alternative because they cost thousands of dollars and they are not portable [1]. Thus, InteracTable is a competitive product because it is low-cost, portable and user-friendly.

II. Design Requirements

We are aiming for an accuracy rate of 95% since this is a value we have seen for many commercially available capacitive touchscreens [2]. We will verify our design by the following metrics:

1. The detected coordinate of the red dot should be within the radius of the dot. We will test this by plotting the detected coordinate on a set of test images and visually identifying if the detected coordinate is within the radius of the red dot. This metric determines the usability of the system.
2. On detecting a coordinate, we should be able to compare the coordinate detected to the coordinates of the projected image so that we can determine whether or not a button was selected. Our GUI will display a calibration screen on system start to get the coordinates of the boundaries of our projected screen. The calibration mode asks the user to place the red dot in the top left, center and bottom right corners of the projected screen. These coordinates will be used to map a "tap" on the projected screen to the laptop screen dimensions. The mapping is necessary to determine if the "tap" occurred within the bounds of a button so the appropriate response is triggered. To test this metric, we will measure the distance between the red dot on a user's finger selecting a button and the corresponding mapped point on the laptop GUI screen.
3. The piezo sensors should detect a tap with 100% accuracy - no false positives or false negatives detected. A tap should also be detected with reasonable pressure and effort. This qualitative metric can be realized with user testing. We plan to ask several users to test our system to determine our tap accuracy rate.
4. We would like the response time for our system to be within 1 second. We will test this time by using the Python time.time() module to take the difference between the moment a tap is detected and a GUI response is triggered. This response time is very important since we want our system to be comparable to working with other collaborative tools like Google Docs.

III. Architecture and/or Principle of Operation

Our system can be broken down into three main functional subsystems:

1. Location Detection subsystem
2. Tap Detection subsystem
3. GUI subsystem

A projector projects the GUI on to a poster board. A webcam monitors the projected GUI. Five piezo sensors are attached to the back of the poster board. When the user taps on the flat surface onto which our screen is projected, the vibrations generated by the tap will be picked up by the piezo sensors. This analog signal will then be processed by an Arduino. A script will be used to establish a threshold value for each sensor that would indicate whether a tap occurred or not. This produces a binary output to the serial port, with "1" being a detected tap. The binary result will then prompt the camera to capture the frame in which this tap was detected. This frame will then be analysed using color detection in the Lab color space. The specific location for tap will be calculated by finding the center of the cluster of red pixels that will be classified using an SVM for dynamic thresholding. After this, the detected coordinate of the red dot will be used to then calculate the corresponding coordinates in the GUI to check which button was being tapped by the user. Once this has been confirmed, the appropriate GUI response will be generated.

We have changed our system block diagram since the first iteration of our design report. Our system does not include MATLAB as we made the decision to write all of our software implementations in Python, as recommended by our course staff. This decision was made to prevent any lag that may occur when transferring data between MATLAB and Python. We now have a unidirectional transfer of data from the location detection subsystem to the GUI subsystem, streamlining the data transfer process. We have also changed our system to use an Arduino in place of a Raspberry Pi. Since we had a working behavioral model of the tap detection subsystem implemented with an Arduino, we decided to continue to use this model in our final design to save time.

Fig. 1. System setup in lab.

IV. Design Trade Studies

A. Tap and Location Detection Systems

System design selection

We had initially identified two different approaches that could potentially solve the problem of detecting where a tap occurred:

a. *Using cameras for detection and tracking algorithms*: The issue with this approach was that it required multiple cameras at different angles for us to distinguish between when the user's finger hovered over a button versus when the user actually tapped a button. We thought of overcoming this problem using a kinect to generate depth maps. However, Microsoft has stopped manufacturing kinects and they would only have been compatible with Windows machines, which was undesirable.
b. *Using piezo sensors to detect vibrations*: This approach included finding the intersection points of several hyperbolic functions computed using waves created due to the vibrations generated by a user's tap. However, this method restricted us to a user interface that required widely spaced buttons in order for the sensors to accurately detect the location of the source of the vibration, i.e., the location of the tap.

Thus, after carefully weighing the pros and cons of the above-mentioned methods, we decided to create a system that would make use of computer vision algorithms to locate the user's finger and a piezo sensor circuit to detect whether a tap occurred or not. This way we could take advantage of both approaches while avoiding their respective drawbacks.

Detection algorithm selection

Now that we had decided to use computer vision to solve the problem of localizing the user's finger, our next step was to decide on an algorithm that would allow us to accurately track the user's finger, which serves as an input to our system. The simplest solution we could think of was to place a bright red dot on the user's finger tip. Now the decision process was two-fold:

a. *Tracking versus Detection:*
   We would either implement a tracking algorithm, Lucas Kanade in our case, or a detection algorithm that would find the red dot in a single frame in which the tap occurred [8]. After careful consideration, we realized that it would be computationally more efficient to localize the user's finger in one frame instead of trying to keep track of it over several frames We found the complexity of the Lucas-Kanade Algorithm to be O(n^3) and the detection algorithms to be O(n^2)[7].
b. *Color detection versus Circle detection:*
   Having concluded that we wanted to implement detection instead of tracking, our next decision required us to choose between color detection and circle detection.

   Color detection seemed to be an obvious choice since the "object" that we would be detecting would be a red dot [10]. However, we did not know how the projected light rays from the projector would affect the red color of the dot. Hence, we experimented with three different color spaces- RGB, HSV, Lab. RGB completely failed as expected, whereas Lab performed the best. The preliminary results are shown in figure 4.

   Since we were not sure how robust color detection would be, we also implemented circle detection. This algorithm was chosen because we knew that the contour of the circular dot would remain unaffected for the most part. The circle detection algorithm was a tweaked version of the Determinant of Hessian blob detection algorithm. The difference between our implemented algorithm and the original one was the fact that the scale of the blur was fixed since the radius of the circle was known. This reduced the number of iterations that would have been required for scale selection. The preliminary results for circle detection can be seen in figure 5.

Based on our initial results, we have decided to stick with Lab color detection as it seems to be the most promising out of the three methods [11]. To improve the robustness of our results, we will be implementing dynamic thresholding using an SVM that would help classify red pixels versus non-red pixels. This would replace our current hard thresholding which has a high chance of failing if there is a change in the testing environment.

Programming language selection

Our initial game plan was to implement all our computer vision algorithms in MATLAB and other data handling in Python. However, we decided to act on the feedback given to us by our staff and implement all of our code in Python. This would reduce any latency that could have arised when interfacing between MATLAB and Python and better help us achieve our goal of a 1 second response time.

B. Display System

GUI color scheme selection

After taking a few test images in a setup that closely mimicked our demo environment, we noticed that projecting a darker color for a button was occluding the red dot which reduced the accuracy of both our color and circle detection algorithms. The fact that we would only try to detect the location of the red dot when the user tapped on a button lead to our decision of inverting the GUI color scheme. This means that we would now have white hues for the buttons and darker hues for the background. However, after collecting test user feedback, we realized that this user interface was bland and lacking user engagement. Thus, we created a second iteration of the GUI which is more visually appealing and improves our user experience. This new interface now walks the user through different scenarios in a comic strip style. While one story depicts why we need InteracTable, the other describes what our system looks like and how it works.

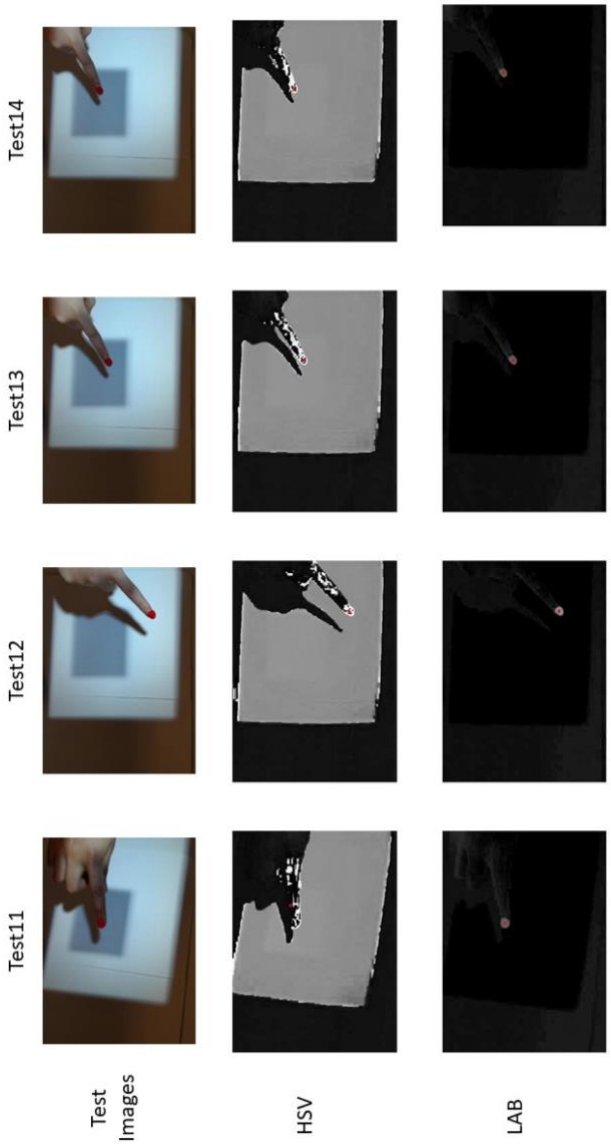

Fig. 4. Preliminary results for color detection.

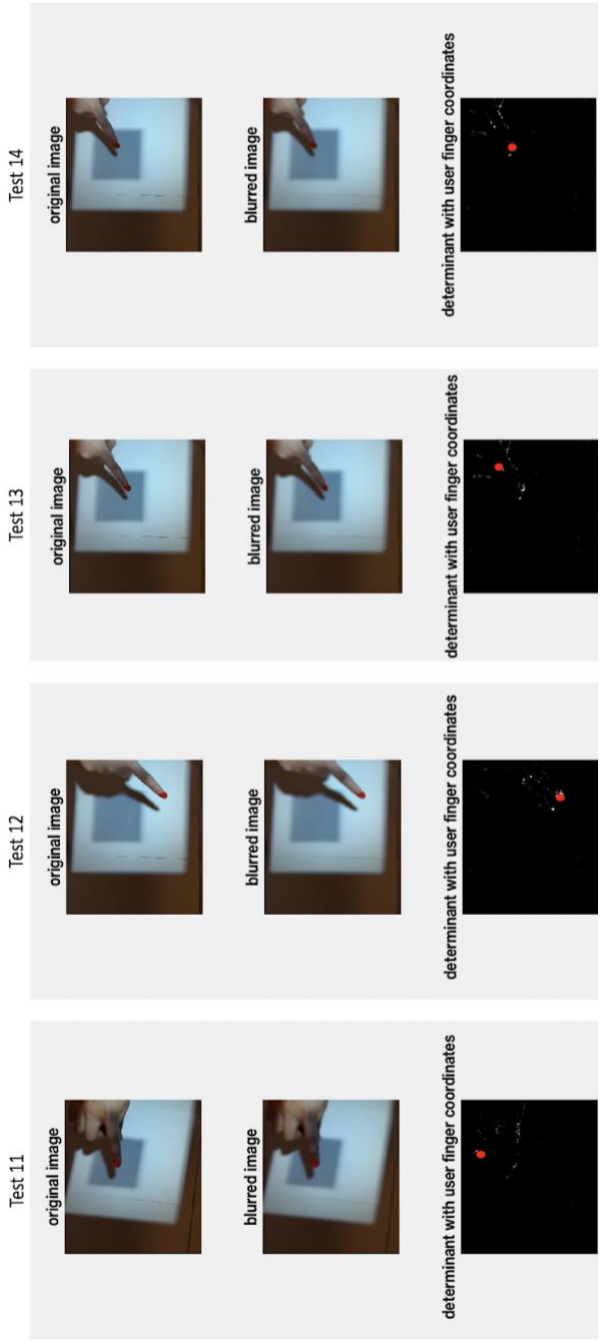

Fig. 5. Preliminary results for circle detection.

C. Metrics and Validation

Metric 1: This determines if the location detection subsystem detects the centroid of the red pixels in a test image within the radius of the red dot sticker in the image. In order to test this metric, fifty pictures were taken of the user's finger with a red dot, and run through the location detection subsystem to find the centroid of the cluster of detected red pixels. Every time, the program successfully detected the correct coordinate within the boundaries of the red dot sticker within the image. Fifty data points were deemed sufficient to test this metric since every image was correct, so it was deemed redundant to take more data points.

Metric 2: This metric aimed to minimize the distance between the center of the red dot and the mapped coordinate plotted by the system. Ideally, this distance should be within the radius of the red dot, which is 0.64cm. To test Metric 2, the poster board was tapped twenty times in different locations, and the distance was measured in centimeters using a ruler. Twenty data points are sufficient to test this metric given that there are only so many places a user can tap on the poster board. In Fig. 6, every data point below the red line is within dot radius. Based on the data collected, our system passes this metric fifty percent of the time. While this is not ideal, the furthest points were at most 1.5 cm, which translates to a few pixels on a laptop surface. The projector, camera, and poster board angles all contribute to this skewed mapping. To accommodate this margin of error, there is a buffer of a few pixels around each button so correct GUI responses will still be triggered even if the mapped coordinate is not exact.

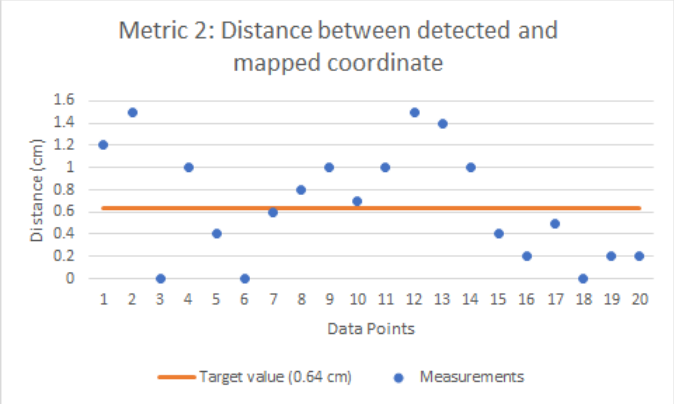

Fig. 6. Measurements for Metric 2.

Metric 3: The tap detection subsystem accuracy rate is limited by the sensitivity of the piezo sensors. The piezo sensors accurately detect the vibrations from a tap when the tap occurs close to the sensor. When we tested the tap detection subsystem with a single piezo sensor at the center of the poster board, the resulting accuracy rate was 30% with thirty test points and three test users, significantly below the target value of 100%. To improve this accuracy rate, we placed four more sensors along the perimeter of the poster board in the configuration shown in Fig. 7. This configuration of five piezo sensors results in an accuracy of 100% determined by testing 50 taps over the entire surface area of the poster board.

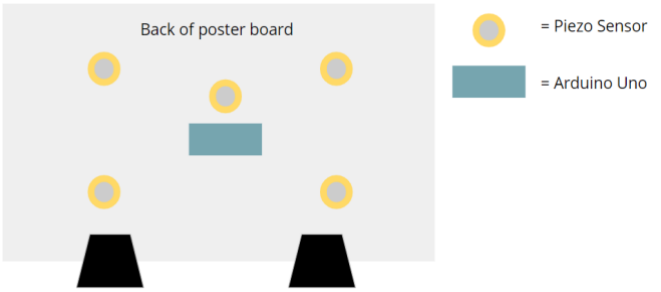

Fig. 7. Configuration of piezo sensor circuit on the back of the poster board.

Metric 4: The desired system response time was 1 second, mapped as the red line in Fig. 8. As shown in Fig. 8, we were unable to achieve this time. The achieved system response time is 1.696 seconds on average. The response times were computed using the Python time.time() module, finding the difference between the time a tap was detected to the time the appropriate GUI response is triggered. The limiting latency is due to delays in sending data from the Arduino to the Python GUI script and the time taken to predict each pixel using the trained color detection SVM.

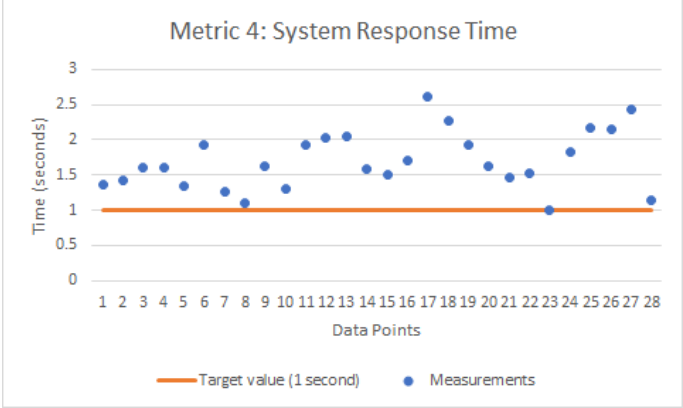

Fig. 8. Measurements for Metric 4.

V. System Description

A. Tap Detection Subsystem

The tap detection system consists of the piezo sensor circuit, which detects the resulting vibrations through the poster board from a tap on the board and an Arduino Uno, which notifies the GUI subsystem when a tap is detected. When the system boots up, the average voltage value of each sensor at rest is used to determine an appropriate threshold voltage. Any voltage beyond this threshold is designated to be a "tap". When a "tap" is detected, the Arduino writes a "1" to the serial port. Otherwise, it writes a "0".

B. Location Detection Subsystem

The location detection system consists of the webcam and the laptop. When the laptop receives tap confirmation, the webcam takes a picture of the user's finger against the projected GUI, using the OpenCV library for Python. Once the picture is saved, it is analyzed by the trained SVM to detect the centroid of the red pixels in the image. The SVM was trained using the scikit-learn package, using data collected with basic color thresholding in the LAB colorspace. The LAB colorspace conversion was done using the scikit-image package. The coordinates of the detected centroid is passed to the GUI subsystem.

C. GUI Subsystem

The display system consists of the laptop and the projector. The projector displays the laptop screen via an HDMI cable. Its focus can be adjusted depending on the distance between itself and the poster board for greater clarity. Additionally, there is a keystone to adjust for vertical tilt of the projected screen. The GUI itself is written using the PyQt library. When the location detection subsystem passes over the detected coordinates of the user's finger, the GUI subsystem determines whether or not the finger is over a button. To identify which button was selected, the GUI uses the dimensions of the projected screen that are gathered with the initial system calibration on start up. These dimensions are used to calculate the proportional coordinate on the laptop screen. If the coordinate is identified as within the set bounds of a button, the projected display is updated appropriately.

VI. Project Management

A. Schedule

Please find the schedule in Fig. 9. on page 9.

B. Team Member Responsibilities

TABLE I. Team Member Responsibilities

Name	Tasks	
	Primary Completed Tasks	***Secondary Tasks***
Isha	Color Detection Algorithms in Matlab. Compare Color Spaces RGB, HSV, LAB and choose best for our system. Set up preliminary GUI for Matlab to Python pipeline (this pipeline is no longer a part of our design)	Building the final piezo sensor circuit
	Color Detection implemented in Python and train SVM for automating color thresholding in Python Real time detection with camera and projector setup	
Tanushree	Circle Detection algorithm in Matlab. Initial GUI using PyQt with required functionality. Final GUI implementation using PyQt.	Building the final piezo sensor circuit
Suann	Initial Piezo Sensor Circuit and testing for threshold Arduino sending data to laptop Final GUI design Lucas-Kanade	Help Isha with testing the detection algorithms work with our camera and projector setup.
All	Set up mechanical parts Test for Metric 1: detected coordinate is within red dot Test for Metric 2: detected pixel coordinate is within the GUI button Testing Metric 3: low latency Integrate tap detection and location detection Write final paper	Write the final GUI Reading sensor data from the piezo sensor circuit to decide threshold for tap detection Testing the final system

C. Budget

Please find the Table II, the table of parts, on page 10.

D. Risk Management

The following are some technical challenges and design risks we identified at the beginning of the semester.

We recognize that surfaces can be of varying dimensions. To accommodate these varied dimensions of our projected GUI, we will add a calibration screen to determine the coordinates of the corners of the projected gui.

We were aware that we may face an issue in finding the perfect illumination under which the projected screen is clearly visible and the red dot on the finger is not obstructed by the colors of the projected screen. When buying the projector, we made sure to look for reviews of the projector being used in daylight so that we could make sure the image is clearly visible under room lights.

We were concerned about facing an issue with the camera tracking the color red since we did not know how the projector light may change the color of the red dot we are tracking. A hand may also distort the projected screen by interfering with the light rays. This is the reason why we decided to track the shape of the dot in conjunction with the color. After setting up our preliminary test environment, we quickly realized that there was negligible distortion from the hand and little to no color change in the red dot from the projected light.

We expected that there may be delays in processing data in real time. We were able to manage this by using a trained SVM for color detection. We also minimized delays in sending data over

a serial port by designing out the connection from Matlab to Python. There is only a one-way data transfer from the Arduino to the Python gui detection subsystem.

Before purchasing the Raspberry Pi, we were under the assumption that we could use it to read analog data just like an Arduino. That is not the case. To save time, we switched to using an Arduino to read sensor data. In the case that we were unable to use the piezo sensors, we had a fallback plan of using an accelerometer built in to a cheap Android phone. This would have given us similar results in detecting the vibrations from a tap on our demo surface.

After our design presentation, we were advised to change our design to omit the use of Matlab alongside Python since there could be delays in sending data from Python to Matlab and vice versa. We decided to write our detection algorithms in Python to omit any possible delays.

In order to prevent spending too much time implementing algorithms so that we can move on to other parts of the project, we set a hard deadline of March 1st to make our conclusions on which algorithm would be best to use. This deadline was to prevent the risk of falling behind schedule so that we can make sure we have a working demo by the end of the semester.

VII. Related Work

A touchscreen table is the most comparable product to our design. However, touchscreen table are very expensive - they can cost thousands of dollars - and they are not portable [3]. Our design can be implemented at any table. If we had the budget to buy a very small projector and camera, our prototype would be easier to transport to different surfaces.

We researched other similar projects done by professors at CMU. One team of researchers made spray paint, named Electrick, that can turn any surface into a touchscreen [4]. Our design is not as permanent as paint.

Professor Chris Harrison built a device that "acoustically couples mobile devices to surfaces" using several piezo sensors [5]. The results from this research project spurred us to decide to use piezo sensors for our own implementation.

For the project OmniTouch, Professor Chris Harrison also created a handheld device that projects a screen on a person's arm and detects a tap on the buttons on the arm [6]. This localizes a tap on the table using depth maps to determine the location of a finger and if a finger taps the arm.

VIII. Summary

Was your system able to meet the design specifications ? Describe very briefly the limits on your system's performance, and any obvious things you can do to improve the system performance if you had more time.

A. Future work

Our system was able to meet design specifications 1, 2 and 3. We were not able to achieve Metric 4, a 1 second response time, exactly. However, our average response time of 1.696 seconds does not result in noticeable system lag in real time. Thus, we found that this achieved response time was acceptable. If we had more time, we would collect a smaller data set of training data so that the time to make a prediction on a single pixel is reduced. Additionally, we would try to parallelize the predictions to cut down on time further. We would also port the piezo sensor circuit to a Raspberry Pi to omit the data transfer over a serial port.

B. Lessons Learned

The most important lessons we learned were to scope out potential project ideas well, schedule enough time to conduct research and set hard deadlines. In addition to this, we had to build in enough slack time. We found it is not prudent to use PyQt with Mac for this application area. The Mac OS adds a lot of processing delay and is slow to update the displayed PyQt GUI. Most importantly, we learned that we had to leave enough time at the end to fix unexpected issues.

References

[1] "3M C4667PW 46" Projected Capacitive 60 Points Multi-Touch Display," *neweggbusiness.com*. Available: https://www.neweggbusiness.com/Product/Product.aspx?Item=9B-0WX-000M-00034&ignorebbr=1&source=region&nm_mc=KNC-GoogleBiz-PC&cm_mmc=KNC-GoogleBiz-PC-_-pla-_-Interactive+Digital+Signage-_-9B-0WX-000M-00034&gclid=EAIaIQobChMI_eDzvfDo4AIVjRyGCh2O0gvPEAkYAiABEgJOZ_D_BwE&gclsrc=aw.ds. [Accessed Mar. 4 2019]

[2] S. Hoober, "Common Misconceptions About Touch," *uxmatters.com,* Mar. 18, 2013. [Online]. Available:https://www.uxmatters.com/mt/archives/2013/03/common-misconceptions-about-touch.php. [Accessed Mar. 4 2019].

[3] "touchscreen table," *google.com*. Available: https://www.google.com/search?q=touchscreen+table&client=ubuntu&hs=gcy&channel=fs&source=lnms&tbm=shop&sa=X&ved=0ahUKEwj2863BxeLgAhXkqFkKHeIQCcYQ_AUIDigB&biw=773&bih=795. [Accessed Mar. 4 2019].

[4] A. Liszewski, "Scientists Figure Out How to Turn Anything Into a Touchscreen Using Conductive Spray Paint" *Gizmodo*, May 8, 2017. [Online], Available: https://gizmodo.com/scientists-figure-out-how-to-turn-anything-into-a-touch-1795016303. [Accessed Mar. 4, 2019].

[5] Xiao, R., Lew, G., Marsanico, J., Hariharan, D., Hudson, S., and Harrison, C. 2014. Toffee: Enabling Ad Hoc, Around-Device Interaction with Acoustic Time-of-Arrival Correlation. In Proceedings of the 16th International Conference on Human-Computer Interaction with Mobile Devices and Services (Toronto, Canada, September 23 - 26, 2014). MobileHCI '14. ACM, New York, NY. 67-76. [Online], Available: http://chrisharrison.net/projects/toffee/ToffeeCMU.pdf. [Accessed Mar. 4, 2019].

[6] Harrison, C., Benko, H., and Wilson, A. D. 2011. OmniTouch: Wearable Multitouch Interaction Everywhere. In Proceedings of the 24th Annual ACM Symposium on User interface Software and Technology (Santa Barbara, California, October 16 - 19, 2011). UIST '11. ACM, New York, NY. 441-450. [Online], Available: http://chrisharrison.net/projects/omnitouch/omnitouch.pdf. [Accessed Mar. 4, 2019].

[7] "What is the computational complexity of Lucas-Kanade algorithm?" *stackoverflow.com,* Mar. 14, 2014. [Online].

Available: https://stackoverflow.com/questions/21111318/what-is-the-computational-complexity-of-lucas-kanade-algorithm. [Accessed Mar. 3, 2019].
[8] Levin, G. "Computer Vision for Artists and Designers: Pedagogic Tools and Techniques for Novice Programmers". *Journal of Artificial Intelligence and Society*, Vol. 20.4. Springer Verlag, 2006. [Online]. Available: http://www.flong.com/texts/essays/essay_cvad/. [Accessed Mar. 3, 2019].
[9] M. Grusin, "Serial Peripheral Interface (SPI)," *learn.sparkfun.com*. Available: https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/all. [Accessed Mar. 4, 2019].
[10] U. Sinha, "Color Spaces," [Online]. Available: http://www.aishack.in/tutorials/color-spaces-2/ [Accessed Mar. 4, 2019].
[11] MathWorks, "Understanding Color Spaces and Color Space Conversion," *mathworks.com.* Available: https://www.mathworks.com/help/images/understanding-color-spaces-and-color-space-conversion.html. [Accessed Mar. 4, 2019].

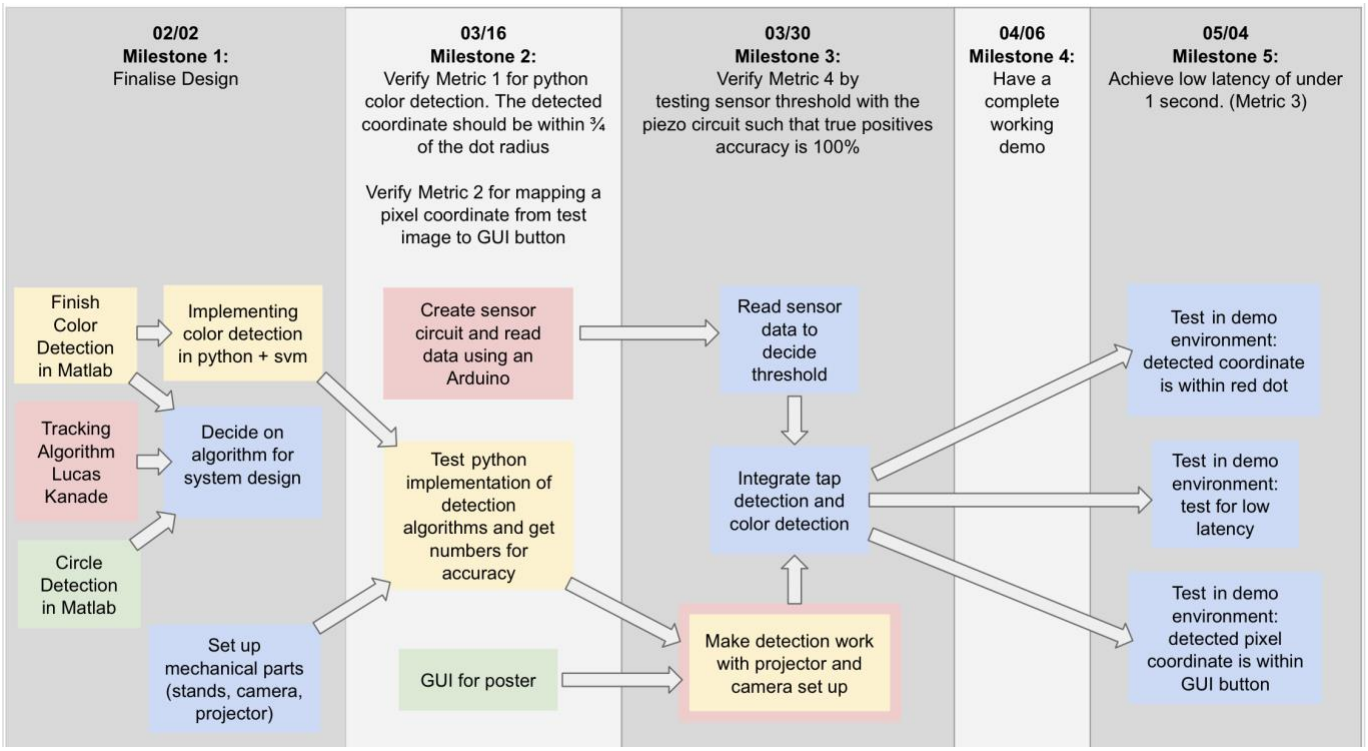

Fig. 9: Semester schedule with dependencies. Yellow indicates Isha's tasks, pink indicates Suann's tasks, green indicates Tanushree's tasks and blue indicates tasks done by all team members. The boxes with colored borders indicate primary and secondary task assignments.

TABLE II. TABLE OF PARTS AND BUDGET

Item	Budget		
	Cost	***Quantity***	***Source***
Webcam	$67.93	1	Amazon
Projector (Black)	$79.99	1	Amazon
Camera Tripod (1 pack)	*$6.99*	*1*	*Amazon*
Projector Tripod (50 inch tripod only)	$29.98	2	Amazon
Red Dot Sticker	$7.99	1	Amazon
USB Hub (4-port)	$9.49	1	Amazon
USB Hub (Anker)	$9.99	1	Amazon
Raspberry Pi	*$34.49*	*1*	*Amazon*
Raspberry Pi Charging Cord	*$8.47*	*1*	*Amazon*
Sparkfun Piezo Sensors	$25.21	10	Sparkfun
Macbook HDMI adapter	$19.99	1	Amazon
Macbook USB adapter	$8.99	1	Amazon
16 GB Micro SD Card	*$12.80*	*2*	*Amazon*
Poster clamps	$98.93	1	Amazon
Arduino Uno	0	1	
Matlab 2018b	0		CMU Software License
Python3	0		https://www.python.org
Scikit-image for Python colorspace conversions	0		http://scikit-image.org
Scikit-learn for Python SVM	0		https://scikit-learn.org
PyQt5 for GUI and PyQt5-sip needed for PyQt	0		https://www.riverbankcomputing.com/software/pyqt/download5
OpenCV for accessing webcam	0		https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_tutorials.html
Personal Laptops	0		
pigpio library	*0*		http://abyz.me.uk/rpi/pigpio/python.html
sockets library	0		https://docs.python.org/3/library/socket.html

a. Items that are italicized were bought but not used.

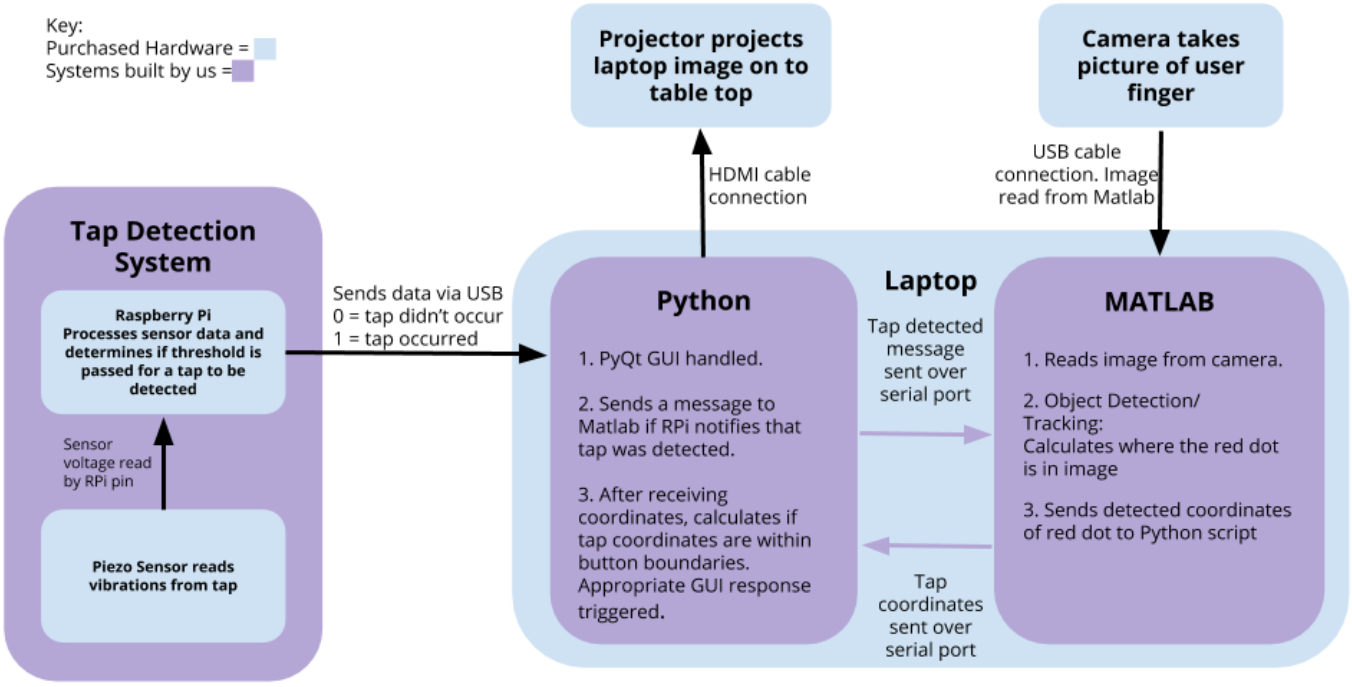

Fig. 2. Initial block diagram.

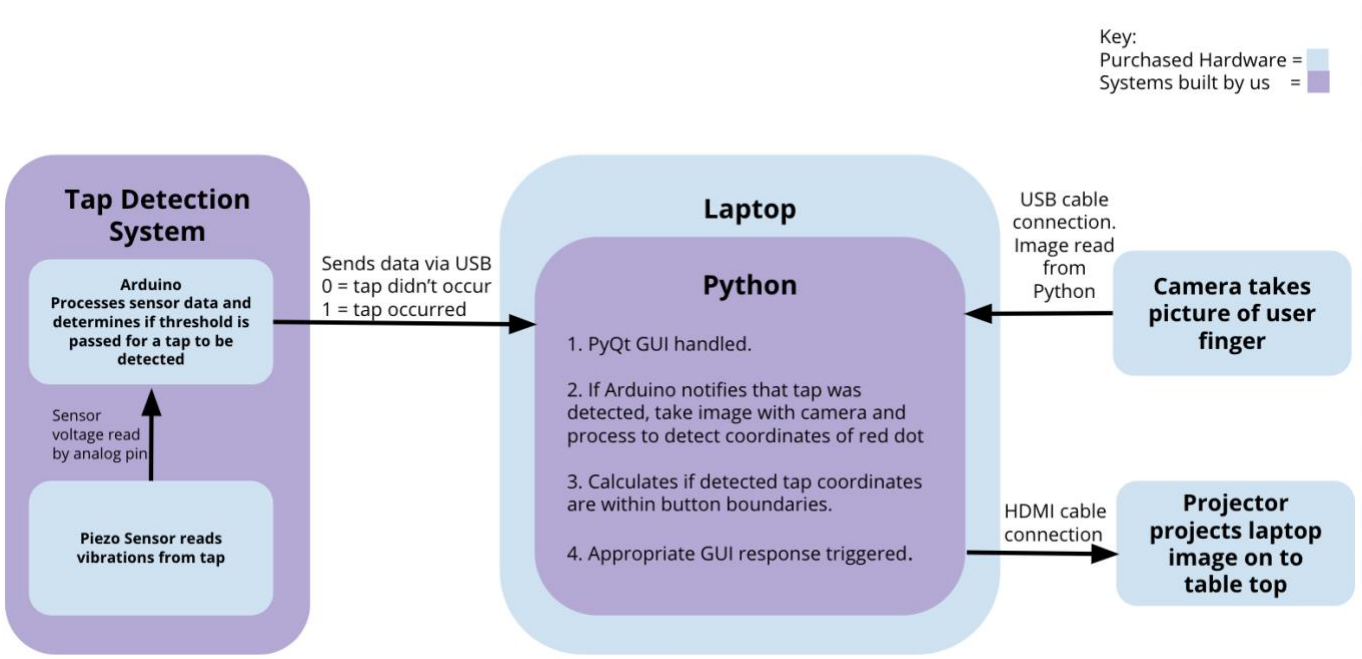

Fig. 3. Final system block diagram.