Touch TrackIR

Matthew Kuczynski, Matthew Shen, and Darwin Torres

Department of Electrical and Computer Engineering,
Carnegie Mellon University

***Abstract*—Laptops equipped with touchscreens are starting to flood the market, however they often come at a steeper price tag. To provide the touchscreen experience to someone who lacks such a laptop, we are proposing the design of a frame equipped with LEDs and photodiodes that can be slipped onto the screen of a non touch-compatible laptop to make it touch-compatible. With the frame attached, users will be able to interact with applications by using touch controls, mimicking the experience of a touchscreen laptop.**

***Index Terms*—Arduino, Infrared, LED, Photodiode, Python, Touchscreen, Windows**

I. Introduction

In the modern world, touch screens have become an accessibility feature that consumers expect on nearly all of their new devices. However, laptops seem to be one set of devices where touch screen functionality is considered optional. Therefore, consumers are left with the need to choose between other general laptop specifications and this accessibility feature. We propose to solve these issues by designing a frame that attaches to the screen of a non-touch compatible laptop and makes it work as a touch screen. Our product is called Touch TrackIR.

The ideal consumer for this product is someone who wants to make their current laptop touch screen compatible, but does not want to buy a new device, either for cost or system specification reasons. Additionally, we would like Touch TrackIR to have accessibility features that do not currently exist on the market, which would increase the importance of the product.

One of the competing technologies that exists is normal touch screen laptops like the Microsoft Surface, however the most similar technology is called AirBar. Since the AirBar product is only $80 and is a simple bar that is placed across the bottom of the screen, it is relatively cheap and lightweight. However, our reach goal is to make our product have programmable functionality so the user can customize the product for their own accessibility needs. Furthermore, Touch TrackIR removes the need to buy a new laptop if a consumer wants touch screen functionality, and it creates a cheap accessory instead.

The goal of Touch TrackIR is to take non-touch compatible laptops of a specific size and make them touch screen devices. Ideally, the product will be able to work in both indoor and outdoor environments, and the frame will be secure enough to allow movement of the laptop. Like any touch screen device, we want Touch TrackIR to respond accurately and precisely to any finger touch, and it should respond in a timely manner. In addition to single finger touch responses, we want our product to respond to the 2-finger zoom and scroll functions that users are used to on their other devices. Similar to AirBar, we want our product to remain both cheap and lightweight, however as mentioned before, we ideally want Touch TrackIR to be able to have programmable functionality.

II. Use-Case Requirements

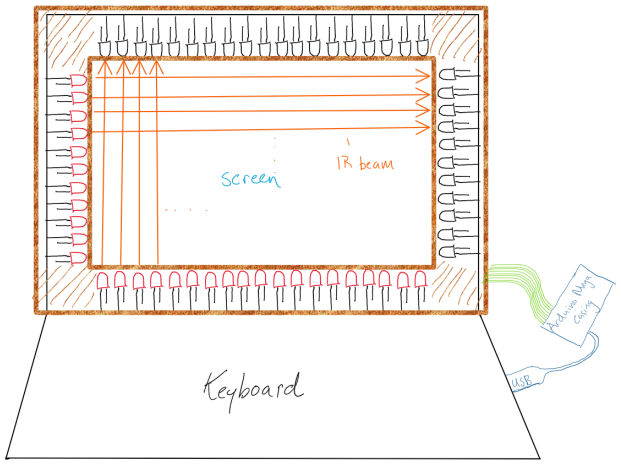
To ensure an enjoyable user experience comparable to that of a regular touchscreen laptop, we have decided on a set of quantitative requirements for the use-cases tackled by our product. For touch-precision, we want the distance between the physical point of contact with the screen (i.e. middle of finger) and the position registered by the OS to be within 0.2 inches of each other. This margin of error lies within the width of the average finger tip, and will guarantee that the registered point of contact lies directly underneath the user's finger.

For the false-positive rate, we want to ensure that at most 1 ghost touch is detected for every 5 minutes the user is idle, while for false-negative touches we are aiming for a maximum rate of 5% - that is for every 100 touches, at most 5 are not detected. Ideally, these rates would be 0, however in the early stages of our design, false-positives and false-negatives will be encountered frequently. The numbers we provide are low enough such that there will be minimal interference with the behavior that the user is expecting in response to their touches.

When it comes to the response time and refresh rate of our system, we want to make sure that our device is fast enough so that interactions do not feel "choppy" due to perceivable lag. A response time that is too low would result in large delays between a user's touch and an application updating in response to it. A refresh rate that is too low would make dragged touches less continuous, resulting in them mimicking a series of individual taps. In general, it would mean less taps can be detected per second, giving rise to a potentially higher number of false-negatives. To provide a good experience, we chose target requirements of 150 ms for response time, which is comparable to early tablets, and 15 Hz for refresh rate, which is about the bare minimum needed before continuous position updates, like when a finger is being dragged, become too slow for comfort.

Finally, our requirement for the final use-case, the weight of the frame, is a maximum of 0.5 pounds. This is the maximum amount of force the screen of our test laptop can withstand before it starts to rotate backward on its hinge. We want our frame to be lighter than half a pound so it does not tilt the screen when it is attached, especially when the user applies a force with their finger. Anything higher than that, and the hinge of the laptop experience high levels of torque that result in an unstable screen.

III. Architecture and/or Principle of Operation



Our system architecture consists of 3 main blocks:

- The Frame
- The Arduino Mega
- The Laptop

The Frame:

There are two main parts of the frame: an IR LED section and an IR Photodiode section.

For the IR LED half, the general principle of operation is that the Arduino will send select signals to decoders. These decoders will then output a high signal to some subset of MOSFETs. This signal will turn on the MOSFETs, thus allowing current to flow through a subset of IR LEDs, illuminating them.

For the IR Photodiode half, they utilize the same select signals from the Arduino as the LED array. These select signals control multiplexers which serve to read only from the Photodiodes that are positioned directly across from the illuminated LEDs.

The Arduino Mega:

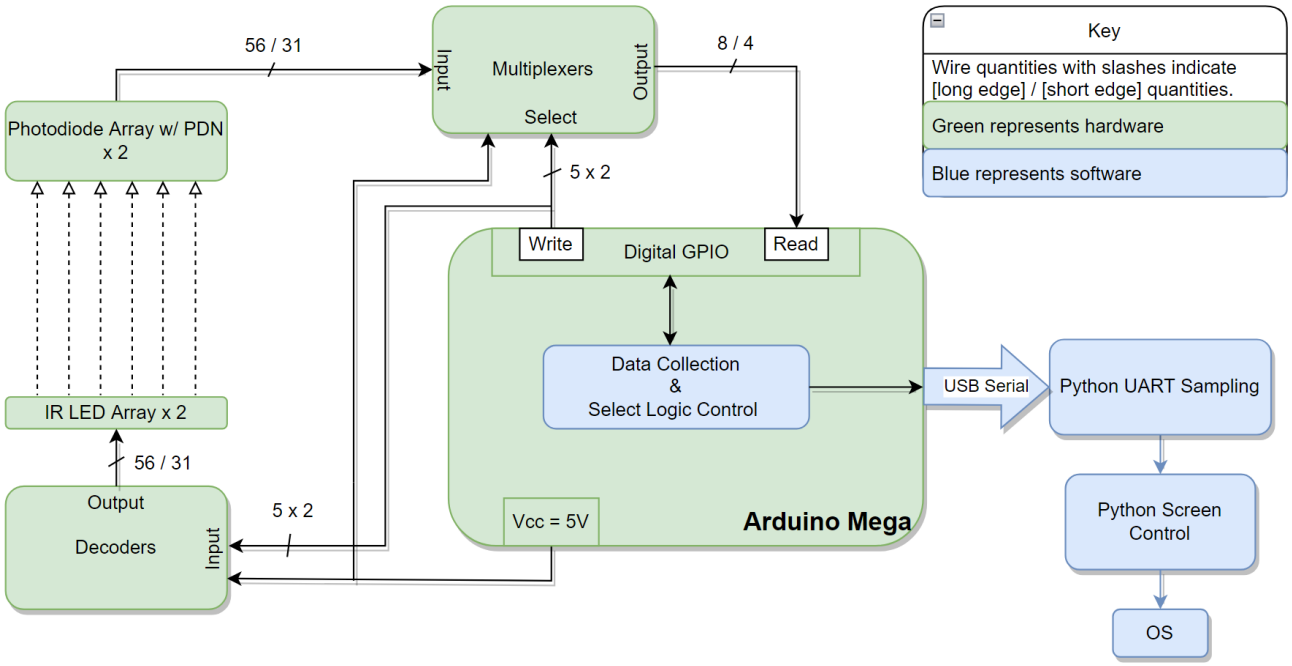
The Arduino is effectively the messenger of our architecture. It collects data from the frame and passes it to the screen control software running on the laptop.

As mentioned above, the Arduino is responsible for sending digital control signals to the hardware in the frame. It is also responsible for reading the outputs of the multiplexers mentioned above. Additionally, the Arduino lends its 5V supply and ground to the frame.

On the other end of its operation, after a single sweep through the arrays of LEDs and photodiodes, the Arduino will report this data over a USB-serial interface for translation to screen-control.

The Laptop:

The laptop will be running Python to listen on a COM port for the serial data being sent by the Arduino. Using this collected data, Python will calculate the position of a finger and pass this data off to a screen-control library.



IV. Design Requirements

Our most important design decisions are split into two main categories - frame structure and software processing, with the exception of one design requirement related to power. Starting with those related to frame structure, touch-precision is dependent on how many LEDs we can pack on each axis - the more LEDs, the higher the resolution of our grid. To meet our use-case requirement of a maximum error of 0.3 inches, we have set a design requirement of 30 LED/photodiode pairs on the vertical axis of the screen and 56 LED/photodiode pairs on the horizontal axis. This configuration results in 5.5 mm, or 0.22 inches, of spacing between the centers of each LED/photodiode. If the point of contact along an axis is calculated as the average position of activated photodiodes, we can then be precise up to 0.22/2 = 0.11 inches, as we can now register touches that occur halfway between LEDs. This is far less than our 0.3 inch goal. Moving on to false positive and false negative rates, these rely on the reliability and responsiveness of our sensors. To ensure we always get the most accurate readings, we want our frame to be strongly secured to minimize any variation that occurs in the alignment of LEDs and photodiodes due to the movement of the laptop. As a design requirement, as we handle the laptop, changing its position and orientation, we do not want deviations in values measured to exceed 0.1% error.

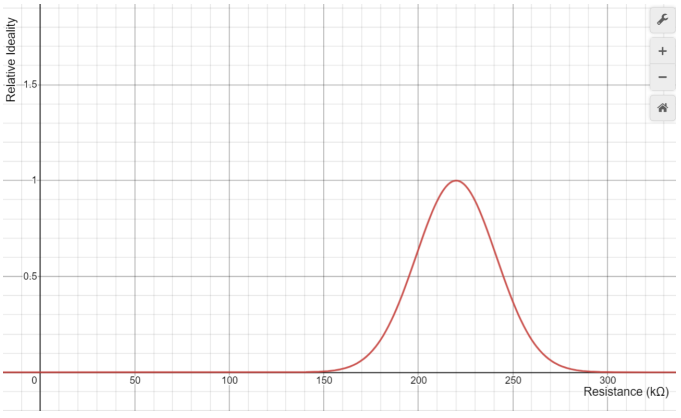
The remaining use-case requirements, response time and refresh rate, are mainly affected by the software processing design decisions. We found that our ability to meet these requirements depends on the speed of our code, so it is important that we choose a good language, use the right libraries, and maintain a high standard of quality when optimizing. Based on this, we have set the following design requirements: LED control and data collection will be written with Arduino code, while finger triangulation and screen control will be done in Python. To ensure the best performance, we will send touch control requests to the OS via python-compatible calls to the Win32 C++ API. Collectively, these all contribute to the response time and refresh rate requirements, so we can quantitatively measure their effects through our response time and refresh rate measurements.

Our final design requirement focuses on the power consumption of our components. We want a power source capable of delivering 5 Volts to satisfy the needs of our hardware. Lucky for us, this should easily be met with the Arduino's 5V pin.

V. Design Trade Studies

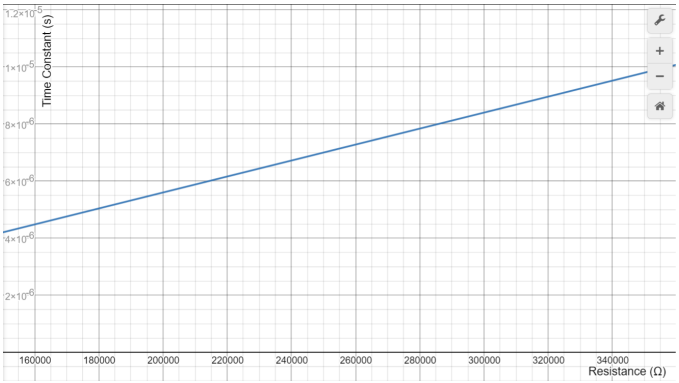
One of the major trade-offs in our design is determining a good balance between speed and sensitivity to light. The speed of the design is governed by a time constant τ=RC, where R is the resistance tied to the anode of a photodiode, and C is the sum of the input capacitance to the mux and the effective capacitance of the photodiode. As for sensitivity, there is a less well-defined governing equation, but two major aspects are ensuring the photodiodes are sensitive enough to detect light from a single LED operating at 100mA from 13 inches away (the approximate distance between the left and right edges). However, since we are turning multiple LEDs on at a time (separated by about 14 LED widths), we need to ensure that the circuit isn't too sensitive, since we only want the photodiodes to respond to the LED that is directly across from it (we will refer to an LED 14 widths down as a "neighboring LED"). Currently, we are using a 200kΩ pull-down resistor for each photodiode. In our tests, this resistor makes the photodiode sensitive enough such that when exposed to an LED 13 inches away, the worst-case voltage at the anode was measured to be 3.2V. This comfortably exceeds the value for VIH of the mux input (2.0V). However, this was a noticeable drop from a worst-case of 4.0V when using a 300kΩ resistor, but the 300kΩ resistor proved too sensitive when tested against a neighboring LED. Since the mux has a VIL of .7V, we want to ensure that neighboring LED exposure doesn't push the pull-down resistor's voltage above this. With the 300kΩ resistor, a single neighboring LED was enough to induce a voltage of .6V, which is dangerously close to VIL. Additionally, since the bottom frame runs 4 LEDs at a time, it is possible for multiple LEDs to further increase this voltage. So, we opeter for a smaller resistor. The 200kΩ experienced a mere .1V increase which we deemed safe. By our estimations, we estimate that the ideal resistance to balance this sensitivity trade-off follows a normal distribution. Given that the 200kΩ resistor overperforms regarding the neighboring LED test, we presume that something a little larger than 200kΩ is the ideal value. Additionally, given that the 300kΩ resistor nearly failed the neighboring LED test, we adjusted our standard deviation accordingly.

$$optimal\ resistance\ est. = \ exp(-\ (\frac{x-220k\Omega}{30k\Omega})^2)$$



It is also important to keep the time constant in mind, although it is likely that the serial communication will dominate the time per sweep. A graph of the time constant has been plotted below using a capacitance of 28pF (estimated from

datasheets).



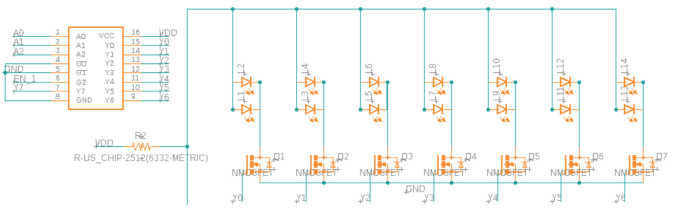
VI. System Implementation

A. *Subsystem A – Hardware*

For the hardware behind our project, there is a frame consisting of a PCB and either wooden or plastic casing. We also use the Arduino Mega to act as the brain for the frame.

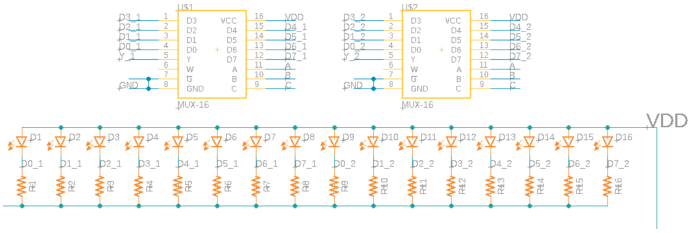
The frame consists of a PCB on each edge of the screen. The aim is to connect the four PCBs by soldering them together with header pins at the corners of the frame. The wires from the Arduino will be fed into the bottom-right of the frame where they will attach to header pins on the PCB. Signals that the left-edge PCB needs are routed through the bottom-edge PCB. Similarly, signals that the top-edge PCB needs are routed through the right edge PCB. Thus, the header pins that are used to fasten the edges together also serve a dual purpose of electrically connecting everything in the frame. We also intend to use the header pins to attach the casing.

Regarding the actual functionality of the frame, there are two main components as mentioned earlier in Section III. One component consists of two LED arrays, one along the left-edge of the screen and the other along the bottom-edge. The other is the pair of photodiode arrays, positioned along the top and right edges of the frame. Only two edges will be "active" at a given time: either the top and bottom, or the left and right. The operation of each pair is for the most part the same, with some minor differences. The left and right edges have arrays of 31 LEDs and photodiodes respectively, while the top and bottom edges have arrays of 56 of each. Below is a small subset of the left-side LED array.



The box on the left is a decoder. It receives control signals from the Arduino and feeds a high signal to one of the MOSFETs shown. In this case, a pair of LEDs is illuminated (the bottom LED array illuminates 4 consecutively). Also note the current-limiting resistor shared by the anodes of the LEDs. The main reason for having LEDs share a MOSFET is to reduce the number of components on-board.

Also see below a subset of the photodiode array:



Firstly, it is important to note that the diodes above should have their terminals reversed. The image above is from Fusion 360 PCB software, and we will just have to ensure that we solder the LEDs in the proper direction when the time comes. Each node atop a pull-down resistor is fed into a mux input, and the Arduino is responsible for instructing the mux which node to read. For the top edge, we have an additional mux whose inputs are outputs from 8 different muxes, which effectively creates a 64:1 mux, plenty for our array of 56. The reason for adding this mux was to reduce the number of wire traces. Originally, 8 wires needed to be sent back to the Arduino to be read. With the added mux, 3 more control signals are required but with only 1 wire to be read, thus reducing the number of wires by 4.

For the last piece of hardware, our Arduino Mega simply uses a USB cable to connect to the computer. As a result, it is effectively the computer that is responsible for powering the frame.

B. *Subsystem B – Python-Arduino Software Interface*

Subsystem B, which is the Python-Arduino interface, connects the information recorded in subsystem A to the screen control logic in subsystem C. This subsystem covers several important components that will be explained in this section, including the select logic control, data collection, serial communication to Python, and finger detection algorithms.

Since only a small subset of the LEDs and photodiodes will be in use at any given time, the select logic control is used to determine which LEDs are turned on and which photodiodes are being read from. This logic is written in the Arduino language and is uploaded directly on the Arduino board, which we believe is the quickest way for this to be performed. The basic idea is that we will loop over all of the LEDs and photodiodes by setting the Arduino pins that control the multiplexers and decoders. There are 13 select logic pins in total, with there being four enable pins (one for each of the two decoders on the left and bottom edges), three select pins shared by the decoders and multiplexers on the top and bottom edges, three select pins shared by the decoders and multiplexers on the left and right edges, and an additional three select pins going to the additional multiplexer on the top that is used to clean up the output logic. These pins are all set

to OUTPUT mode in the Arduino code, and they are set using a bitwise AND and shift of the loop counter.

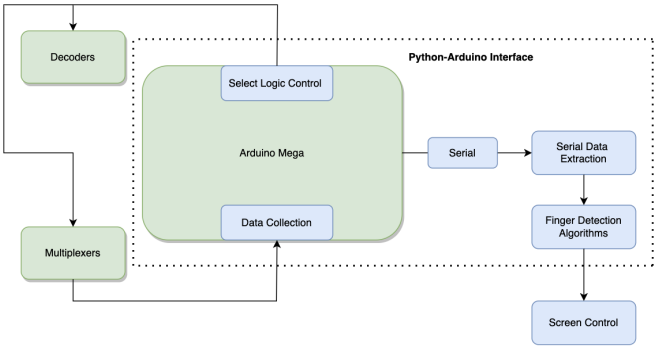


Fig. 1. Block diagram of Python-Arduino interface.

The data collection section of this subsystem requires 5 pins to be read, which correspond to the photodiode values. The right edge has four pins to be read, while the top only has one due to the extra multiplexer. Unlike for the select logic, these pins are set to INPUT mode. The photodiodes for which these pins correspond to change with the select logic, and all of the photodiodes are looped over.

While the data collection aspect is relatively simple, it ties in closely to the serial communication component of the subsystem. We have determined that the fastest way to communicate the data from the Arduino to a Python script is by using serial prints on the Arduino and the Python serial library. Each serial print contains the photodiode data with labels for the loop values, and decoding this on the Python end becomes a string parsing task. We have primarily been testing the serial communication at a baud rate of 9600, however some preliminary tests show that a faster baud rate is possible if we need to speed up our communication.

Finger detection algorithms make up the next part of this subsystem, as they make sense of the data from the serial communication. We should be able to determine the number of fingers on the screen by looking at the number of paths that are covered at any given moment, as well as any potential space between blocked paths. From calculations of the LED spacing and average finger width, we have determined that either 2 or 3 paths will be blocked by a single finger at any moment. In most cases, the blocked paths will be separated, however in the scenario where consecutive paths are blocked by multiple fingers, we expect that at least 4 paths will be blocked in one dimension, or 3 paths will be blocked in both dimensions. To determine finger location, the average of the centers of the blocked paths on each dimension will be used since this will minimize the mean squared error. The most complicated algorithms will need to determine the gestures, and these will require sampling over time. For a single touch, we expect there to be no change in finger position over time. For a swipe/scroll movement, we expect the movement to occur in a constant direction. Finally, for zoom functionality, we expect two separate blocked paths to be either moving towards each other or further apart, for zooming out and in respectively. A significant amount of testing will be needed to tune these algorithms to work with actual, physical movement.

The final component of subsystem B is to create an interface that can communicate easily with the screen control module. With the finger detection algorithms, we should be able to detect any number of fingers from one to three, and we should be able to distinguish between tap, scroll, swipe, and zoom movements. By adding the finger location to this information, the screen control module should be able to easily perform its job using functions specific to each action.

C. *Subsystem C – Screen Control*

Subsystem C is a software interface written in Python for sending requests to the OS for emulating touch controls. This is done via Microsoft's Win32 C++ API. Note that this is a C++ library and not a Python library. Although the interface will be written in Python, by using the building ctypes library, we can invoke C++ functions from the .dll files containing the instruction code for Win32 without writing any C++ code.

As of now, three main commands have been proposed to be supported by this interface: "Finger Down", "Finger Up", and "Update Position". "Finger Down" tells the OS that contact has been made at a specified pair of coordinates, and generates a new touch instance. A touch instance describes a continuous contact event, and lives as long as contact is being made with the screen. Touch instances allow us to support multi-touch gestures, as each touch can be tracked individually. "Finger Up" terminates a specified touch instance, signifying to the OS that contact is no longer being made. "Update Position" moves the point of contact of a specified touch instance to a new set of coordinates.

These three commands provide the functionality for supporting taps and drags among multiple fingers, though for our MVP we want to put a focus on being able to tap/drag with one finger. It must also be pointed out that the Win32 API also allows for a contact area to be specified. This means instead of contact being made at an exact point in space, it is spread across a group of pixels. The "Finger Down" command can take in a "radius" if a nonzero contact area is to be specified. It is up to the Windows application to decide how to interpret any touch command we give it, and how the contact area affects it.

VII. Test, Verification and Validation

We have outlined a specific test for each of the use-case and design requirements that were mentioned in the earlier sections. Almost all of the tests are quantitative, with a couple being qualitative. Tests A-F are for the use-case requirements, while tests G, H, and I are for the design requirements.

A. Tests for Touch Precision

For the touch precision requirement, we plan to conduct a test where we touch the screen 10 times in 10 different precise locations and find the maximum distance from each location to the spot calculated by the finger detection algorithms. Our goal for this test is to have the maximum distance be less than 0.2 inches from the desired location. The distance between the centers of the LEDs allows for the best possible touch precision to be 0.11 inches, so we believe that 0.2 inches is a reasonable target given that it is nearly impossible for our system to be ideal. A good touch precision is important so that the user does not have trouble touching a specific location.

B. Tests for False Positive Rate

Our goal for the false positive rate is to have there be less than one false positive touch every five minutes. In order to test this, we plan to allow the frame to remain attached to the screen and untouched for one hour with the script running, and we will then count the number of touches that are detected during that time period. Given our goal for this test, we expect to see less than 12 touches in order to consider the test as passing. Ideally, the false positive rate will be much lower, but given how long this test takes to run, we consider this metric to be reasonable. This test is important because it will ensure that the screen does not seem to randomly register touches even though the user is not touching the screen. If the false positive rate is high, the product would be extremely frustrating to use and could not satisfy any MVP.

C. Tests for False Negative Rate

To test the false negative rate, we plan to repeatedly touch the screen 100 times and count how many touches are registered by the system. Since our goal for this requirement is a false negative rate of less than 5 percent, we expect to see at least 95 touches registered with this test. Similar to the false positive rate, a high false negative rate would make the product frustrating to use since a large number of touches would not receive any response. However, we feel that the acceptable threshold for the false negative rate is much higher than that of the false positive rate. No response is not that big of a deal since the user can simply touch the screen again, however a random, unwanted response will leave the user confused.

D. Tests for Response Time

The goal for the response time requirement is for the screen control function to occur within 150 milliseconds of the touch. This requirement was determined based on the response time of other touch screen devices, and since our product has significantly less processing, we believe that we should be able to achieve similar results. Our test for this requirement is performed by recording a video of the touch and screen response, and then performing a frame-by-frame analysis to determine the exact time difference between the touch and screen response. A successful pass of this test will ensure that the user experience using Touch TrackIR will be similar to the experience that consumers expect from their touch screen devices.

E. Tests for Refresh Rate

In order to test the refresh rate, we plan to analyze a slow motion video to determine the cursor update time. To do this, we will need to temporarily disable the smoothing algorithm. Our goal for this test is to achieve a refresh rate that is greater than 15 Hz. A high refresh rate is important so that there is a higher likelihood of a touch actually being registered since it will be checked for more frequently.

F. Tests for Frame Weight

The test for the frame weight is very simple since we can measure the weight of the frame on a scale. We consider this test passing if the frame weighs less than half a pound, which is a value that we determined by seeing how much weight could be placed on the top of the laptop screen without it opening or closing. This test is important so that the product does not affect the screen's ability to stay stable.

G. Tests for Stationary Frame

A stationary frame is necessary to achieve the use-case requirements since it will allow for the touch precision, false positive rate, and false negative rate to remain constant with movement of the laptop. The test for this is shaking the frame for 15 seconds and then reperforming the tests for the three use-case requirements mentioned in the last sentence. We consider a passing test for the stationary frame as one where there is less than 0.1% average change in the result of the three tests. The idea here is that if the frame is stationary, then the other tests should not be affected.

H. Tests for Power Source

Similar to the frame weight, the power source test is also somewhat trivial. Given that we will be using an Arduino Mega, we can use the 5V output pin, so the quantitative goal for this test is 5V. The test can be conducted by verifying the voltage with a multimeter. The power source is essential for the circuit to exist, so it is necessary for all of the use-case requirements.

I. Tests for Software Processing

While there is no quantitative metric to measure the software processing, it is necessary in order to fulfill the use-case requirements. Specifically, the software processing will be considered passing its test if the response time and refresh rate are passing their respective tests.

VIII. Project Management

A. Schedule

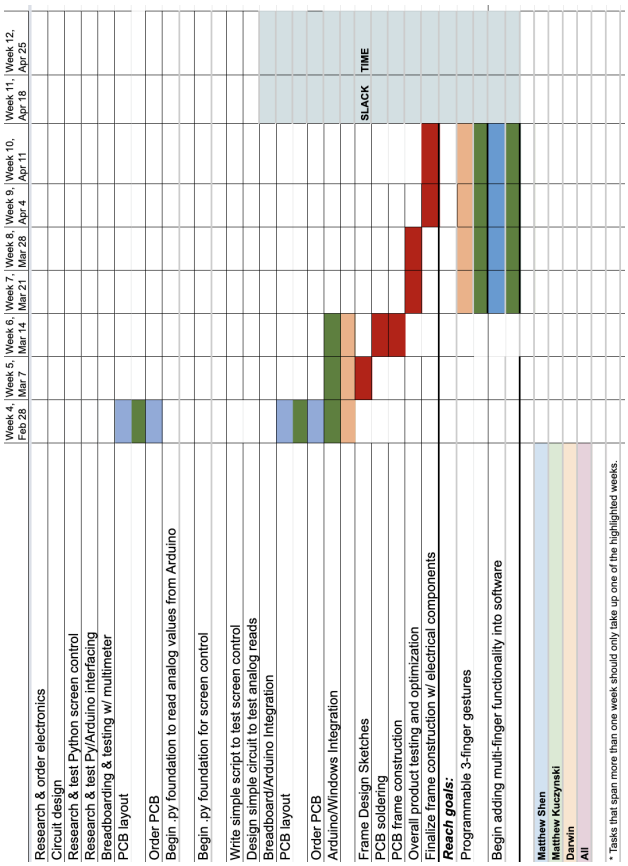


Fig. 2. Gantt chart for week 4 to completion.

B. Team Member Responsibilities

Each team member has a different subsection of the block diagram for which they are primarily responsible, however all of the components work together, so there is significant overlap. Matthew S.'s main focus is the design of the hardware side of the project, which involves determining which components to use, how they will be connected, and also performing the majority of the PCB design. Matthew K. is primarily responsible for the software interface with the Arduino, which involves writing the Arduino code for performing data collection, controlling the select logic, and sending data over the serial interface to Python. This also involves using the data sent to the Python sampling code to determine the finger position(s) and quantity. Darwin is primarily responsible for the Python screen control backend, which involves interpreting the finger position as a command that is sent to the OS, as well as allowing for programmable multi-finger functionalities.

The secondary responsibilities for all of the team members are the same, and they include frame design and construction, soldering, and integration of the components that were worked on individually by each team member. More specific information about the secondary responsibilities can be found in the Gantt chart.

C. Bill of Materials and Budget

See page 9 for Bill of Materials.

D. Risk Mitigation Plans

For each of our tests, we have developed a risk mitigation plan in case the assessment fails. If touch precision is not high enough, we can try reading multiple photodiodes for each of the LEDs to extract more information about the finger location. For the false positive test, we can use software to filter out the touches that do not span across multiple photodiodes since we know that a normal finger will cover multiple. If our false negative rate is too high, response time is too long, or refresh rate is too low, we may improve the software quality by switching from Python to C++. Our current plan for the frame is to construct it out of wood, but we feel that a plastic frame will be a lighter option, although more difficult to construct. We also plan for the frame to be able to slide onto the laptop screen, however if the stationary frame test fails, then we will add a mechanism to tighten the frame to the screen. Since we are dependent on the Arduino and a 5V power source based on our design, our only possible risk mitigation plan for the power source test failing is to have a backup Arduino Mega. As mentioned above, issues with the software processing tests can be fixed by moving from Python to a faster language.

IX. Related Work

A similar project that provided inspiration during the early stages of the project was an approach to providing touch-screen compatibility using light-triangulation. This project also used an array of LEDs, but instead utilized a CIS scanner from a printer instead of a self-built array of photodiodes. While his idea uses far fewer LEDs than our design, it is compensated by the insanely high resolution and accuracy of a CIS scanner. This high resolution and accuracy enables him to triangulate light from the LEDs to calculate the position of a finger. However, the low resolution of LEDs meant that touches made close to the LEDs experienced drop-offs in accuracy. For our approach, we wanted equal effectiveness across the board, so we opted for a dense array of diodes. Additionally, we decided to design our own sensor array since a CIS sensor would be far to bulky to put on a laptop frame.

Another similar product is Airbar, which somehow uses a single bar along the base of the screen. Initial research for our product aimed at a similar approach. Unfortunately, we couldn't find distance sensors that were small enough to achieve this. After hours of research, the only option that remained would be to buy an Airbar to reverse engineer. We decided this would not be an effective use of our budget.

X. Summary

Touch TrackIR is an attachable frame intended to transform the screens of non-touch-compatible laptops into touchscreens. This is done via an array of Infrared LEDs and photodiodes that detect the presence of a finger on the screen through sweeps of LED/photodiode pair activations. To make sure the user has the best experience, we aim to have precise contact detection, with errors less than 0.3 inches, low response times of under 150ms, and a refresh rate of at least 15Hz, providing smooth screen updates and an imperceivable delay. In addition, we want to ensure that every touch counts, with responsive sensors that provide near-zero false-positive and false-negative rates, removing any hiccups that may occur due to unexpected behavior. By providing a sturdy and light frame, we will also make the system as non-invasive as possible, allowing for more freedom in handling and transporting the device without having to worry about the hinges being under too much stress or changes in sensor accuracy due to movement.

As we work to implement our design, our main concerns deal with accuracy and speed, as these are the biggest factors that affect user satisfaction. Accuracy is not a complicated issue as it mainly depends on how tight we can fit our LEDs. A higher density of LEDs gives us a higher resolution, and thus better accuracy. Speed, however, is a more complicated issue as all parts of our system affect it. From our hardware, to the Arduino, to the Python backend. The faster we are able to transfer data along all subsystems and convert it into touch commands, the smoother the user experience will be. Initial tests had revealed some problems in sweeping speed and data collection, but as we spent more time analyzing the issues, we were able to come up with better approaches. Individually, the speeds we measure seem to be sufficient, but we will be able to have a better idea of where we stand once we are able to conduct more thorough tests after integrating our subsystems.

Glossary of Acronyms

API - Application Programming Interface
CIS - Contact Image Sensor
GPIO - General Purpose Input/Output
IR - Infrared
LED - Light Emitting Diode
OS - Operating System
PCB - Printed Circuit Board
PDN - Pull-Down Network
USB - Universal Serial Bus

References

[1] Perardel, Jean. “Magic Frame : Turn Everything into a Touch Area.” *Hackaday.io*, 5 Sept. 2017, https://hackaday.io/project/27155-magic-frame-turn-everything-into-a-touch-area.

[2] “Photodiode Basics.” *Wavelength Electronics*, Wavelength Electronics, 11 Feb. 2020, https://www.teamwavelength.com/photodiode-basics/.

[3] Nasir, Syed. “Introduction to LM317.” *The Engineering Projects*, 2 July 2020, https://www.theengineeringprojects.com/2017/06/introduction-to-lm317.html.

[4] Shawn. “Types of Distance Sensors and How to Select One?” *Latest Open Tech From Seeed*, 29 June 2021, https://www.seeedstudio.com/blog/2019/12/23/distance-sensors-types-and-selection-guide/.

Item	Quantitiy	Price Per ($)	Total Price ($)	# used in design	Part #
Arduino Mega	2	18	36	1	
IR LED	100	0.3272	32.72	87	SFH 4555
IR Photodiodes	100	0.1729	17.29	87	INL-5ANPD80
210k Resistors	100	0.0184	1.84	87	ERJ-3EKF2103V
Muxes	15	1.051	15.765	13	CD74HCT151M
Decoders	5	0.41	2.05	4	74HC138D
PCB	1	80	80	1	
MOSFETs	34	0.298	10.132	30	PJA3430_R1_00001
Low Ohmic Resistors	5	0.1	0.5		?
Solder Mask & Paste	1	80	80		
Header Pins	10	0.1	1		
		Total:	277.297		

TABLE I. BILL OF MATERIALS