Touch TrackIR - Final Presentation

Team A6 - **Matthew Kuczynski**, Matthew Shen, Darwin Torres

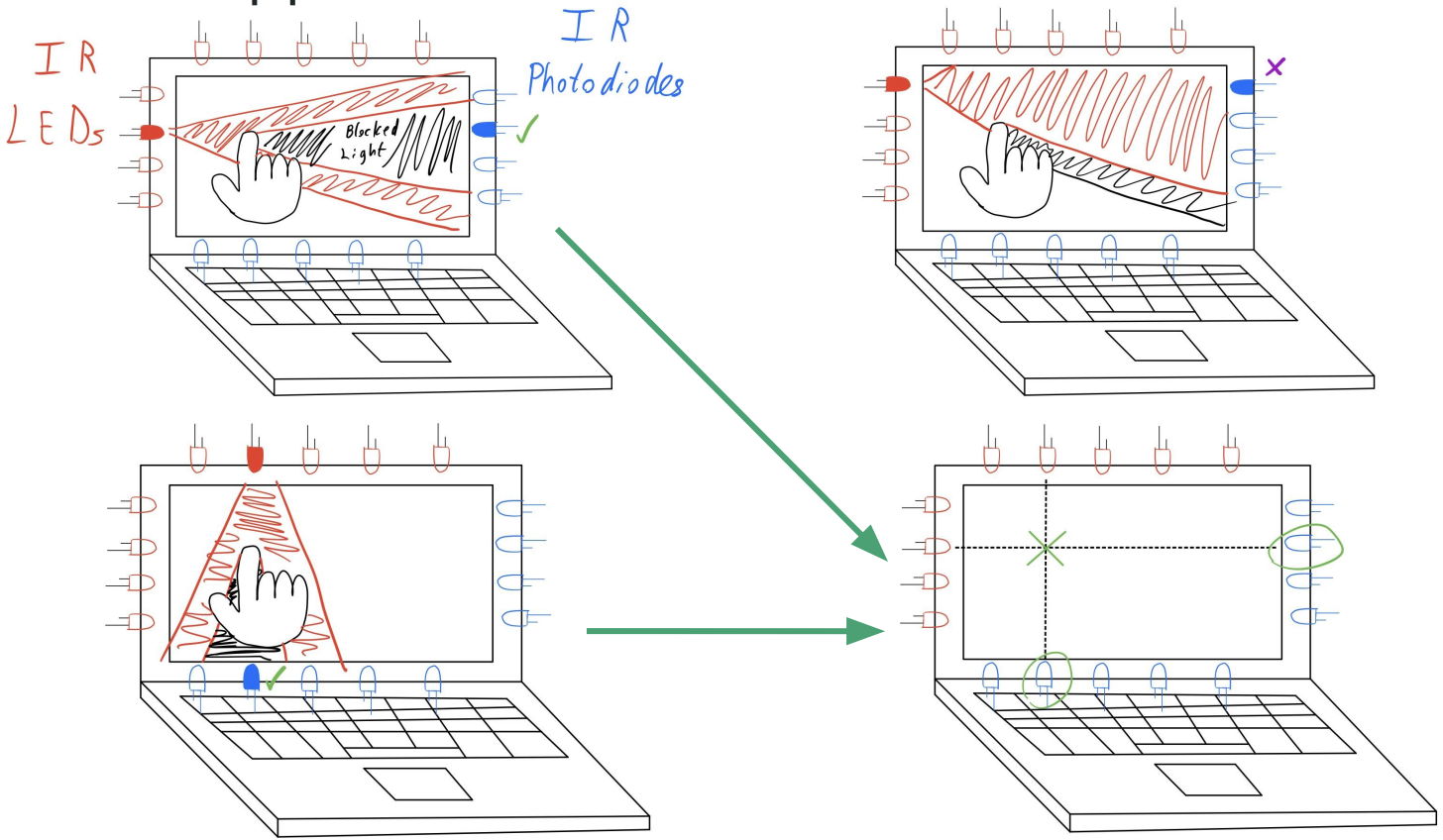
Use Case & Use-Case Requirements

- Touch TrackIR is designed to convert non-touch-compatible laptops of a specific size into touchscreen laptops.
- Overview of use-case requirements:
  - **Touch Precision**: < 0.3 inch
  - **False Positive Rate** (Screen not touched, but touch detected): 1 per 5 minutes
  - **False Negative Rate** (Screen touched, but touch not detected): 5%
  - **Response Time**: < 150 ms
  - **Refresh Rate**: > 15 Hz.
  - **Weight of frame**: 1/2 lb.

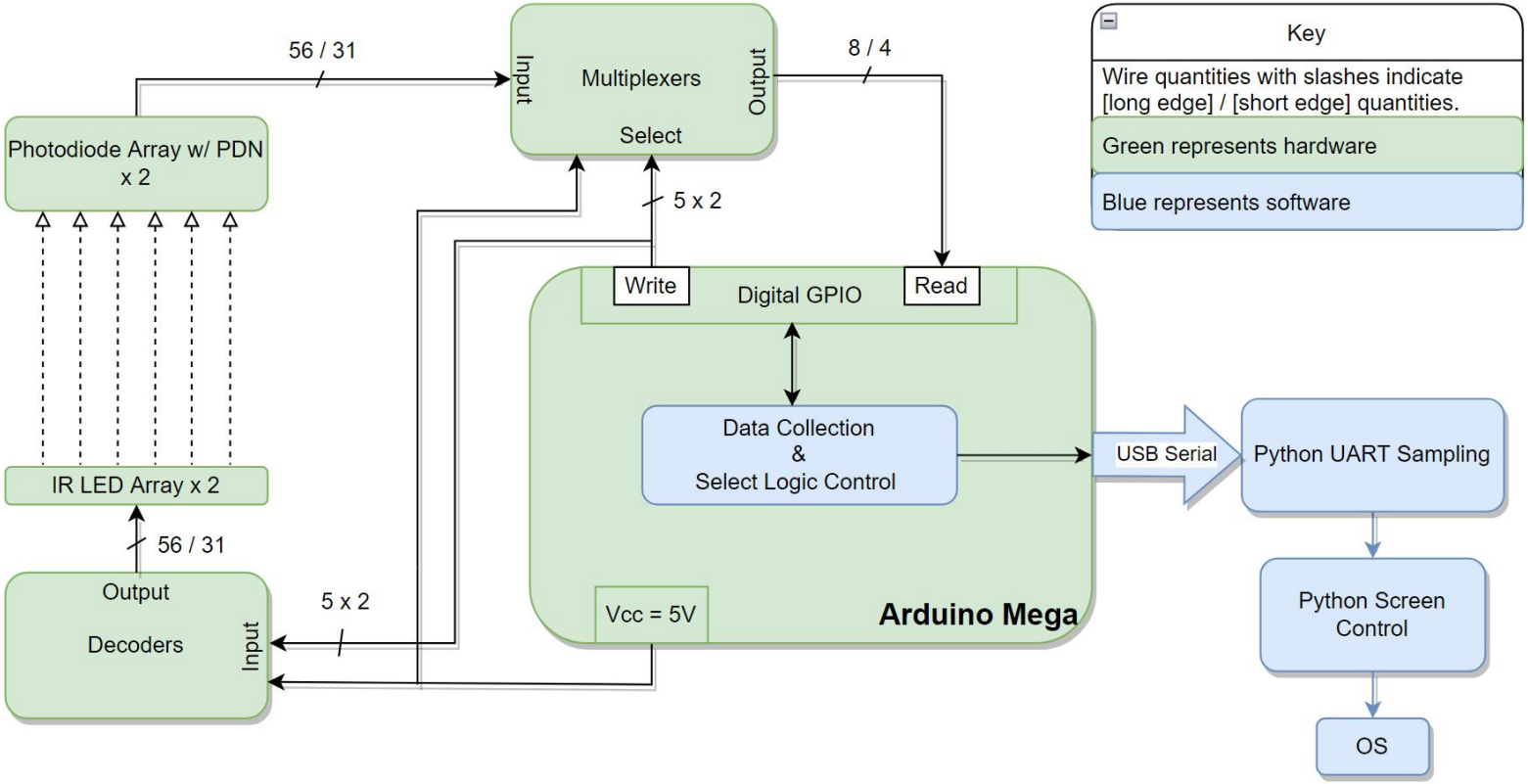
Solution Approach - Overview

- Hardware
  - Use a 2D array of IR LEDs and Photodiodes to determine finger position, powered by Arduino Mega.
  - The circuit is designed to have the centers of the LEDs 5.5mm apart, giving us a precision of about 2.75mm, which is far less than our 0.3 inches (7.62 mm) requirement.
  - Hardware secured by PCB in a frame around all 4 edges.
  - 31 x 56 LED/Photodiode Arrays along short and long edges of frame.
- Software Processing
  - Uploaded Arduino code toggles LED-photodiode pairs.
  - Finger positioning and screen control via Python scripts.
  - First, photodiode values are read from digital GPIO pins on Arduino Mega, then data is passed via serial USB to Python backend.

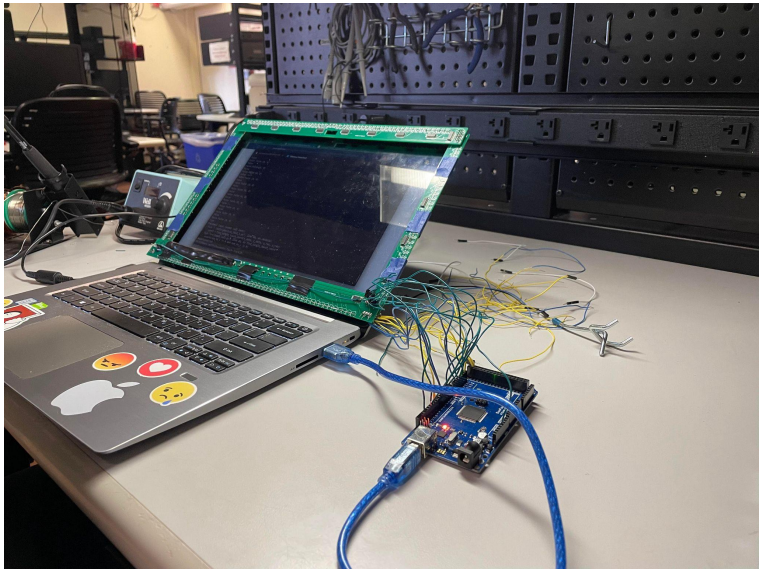
Solution Approach

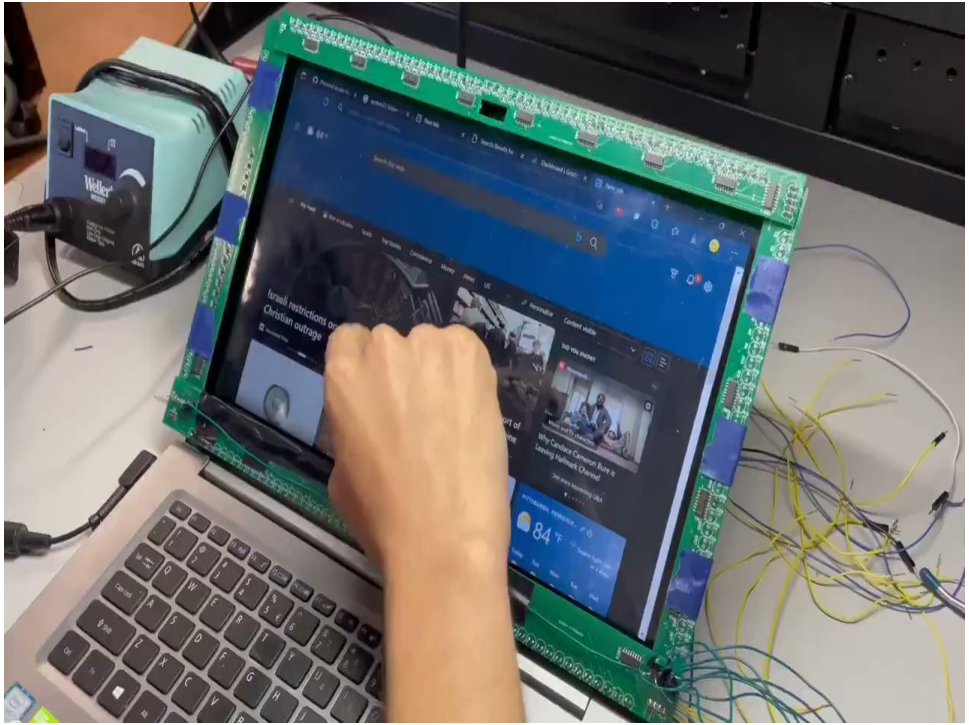


Block Diagram



Complete Solution





Testing & Verification - Use-Case Requirements

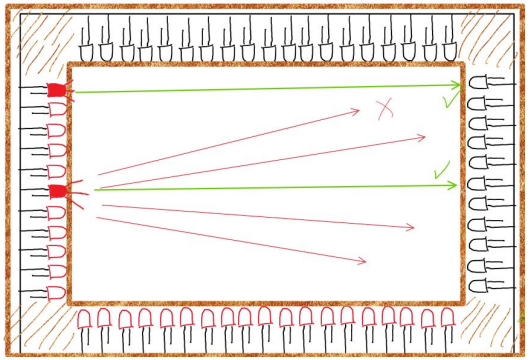
Requirement	Testing Method	Quantitative Goal	Measured Results
Touch Precision	Measure distance between center of finger and the actual click.	< .3 in	0.13 in
False Positive Rate	Let frame remain attached with screen untouched for one hour with script running, and count touches detected.	< 1 per 5 mins	0%
False Negative Rate	Repeatedly touch screen 100 times, and count how many requests were sent.	< 5%	0%
Response Time	Calculate time using frame-by-frame video analysis of touch and screen.	< 150 ms	120 ms
Refresh Rate	Analyze slow-mo video to determine cursor update time (temporarily disabling smoothing algorithm).	> 15 Hz	66 Hz
Weight	Measure on scale, and ensure screen stays open with frame attached.	< ½ lb	0.4 lbs

Testing & Verification - Design Requirements

Requirement	Testing Method	Quantitative Goal	Measured Results
Stationary Frame	Shake frame for 15 seconds and recalculate false positive and false negative rate.	< 0.1% error change	0% Error Change
Power Source	Verify with multimeter.	5V	4.73V
Software Processing	Measure response time and refresh rate as explained before.	Meet goals for response time and refresh rate	Requirements met

Testing & Verification - Design Tradeoffs (Hardware)

- Layout area vs. hardware complexity:
  - Due to narrow PCBs responsible for routing signals from 87 photodiodes, multiplexers are necessary to condense data flow back to the Arduino. We balance the number of traces going to the Arduino and the complexity of our select logic to reduce this number.
- Layout area vs. # of LEDs illuminated in parallel vs. sensor sensitivity:
  - LEDs are illuminated by switching a power FET. To reduce the number of FETs, we illuminate more than one per NMOS switch.
  - This is a tradeoff regarding parallel LED interference. If an LED is blocked, we wanted to ensure that its respective photodiode was tuned to be insensitive to a parallel LED's beam. To decrease the sensitivity, we would decrease the resistance tied to the photodiode.



Testing & Verification - Design Tradeoffs (Software)

- Programming Language vs. Speed
  - Using an interpreted language like Python on the OS side results in slower performance, however we found that it has intuitive, easy-to-use libraries for reading serial data
  - Despite the loss in performance, we found that our current speed exceeds our minimum requirements by a large margin
- # of Serial Messages vs. Python Processing Speed
  - We found that serial communication is slow and must be minimized, so we use as few messages as possible, which requires more Python processing rather than Arduino code processing.
- Refresh Rate vs. Sensitivity
  - When updating too quickly, it is difficult to distinguish between a touch and small drag.
  - Instead, it is better average small samples and send fewer updates.
  - This makes it easy to register taps, however quick drag motions might not appear as smooth

Schedule

	24-Apr	1-May
Improve single finger clicks		
Improve drag smoothness		
Implement multi-finger functionality		
Hardware Beautification		
Matt K		
Matt S		
Darwin		

Remaining Tasks

- Improve Single-Finger Tap Performance
  - One finger scroll works well, but taps can be mistaken for scroll/drag, often due to shaky fingers or slow taps.
- Multi-Finger Functionality
  - Two-Finger Taps + Zoom
- Drag smoothing
  - Smoother interpolation of position updates during finger drags
  - Requires post-processing of input signals
- Frame Construction
  - Current method works, but not visually appealing or guaranteed to be secure.