Multimodal Interaction Project Report

Touch Typing Tutor:

A multimodal investigation of breaking bad typing habits

Prof. Maria De Marsico
Mattia Stellacci

1 Abstract

The following work presents a system aimed at correcting users' bad typing habits. By monitoring their use of a computer's keyboard, determining what fingers are being used to press which keys, the system allows the user to understand where their typing habits differ from canonical typing techniques. The problem was addressed using both a machine-vision based systems and a hardware based implemenation using a micro-controller and custom sensors. Notes on the implementation, challenges and insights are presented.

2 Introduction

While touchscreens and (more recently) voice-based techniques have gained great relevance for mobile computing, the keyboard remains the keystone of user interaction in desktop computing. Ever since the very beginning of computing devices, keyboards represent the most basic form of interaction that the user can have with the system: Entering text, unambiguously, through he use of a unchanging, physical device infront of them.

As technology and computers have become a central part of our (professional) lives, people spending many hours a day on them, using this interface in the most ergonomic and thus efficient way is paramount. Not only for productivity, but also for the user's health.

Though some details of this interface, such as the arrangement and layout are fields of scientific research, others such as the tacitlity and resistance of the keys of an ideal keyboard are subject to personal preference. Despite controversy surrounding these latter aspects, it is generally accepted that the most efficient way of using a keyboard involves using all 10 fingers, minimizing the hands overall movement and taking advantage of muscle memory to obtain optimal ergonomics and higher speeds.

The act of typing, as in using a typewriter to write, predates it's modern use an input for computers. While it was not uncommon in the past for people to have their first contact with typing/typewriters as part of typing classes teaching technique, nowadays many people learn and internalize self-taught techniques before ever even being confronted with the theory of "correct" typing.

In the following work I will describe a system I explored to aid users in their effort to change their typing habits by monitoring their activity on the computer. I was motivated my own experience of wanting to unlearn past bad typing habits, the difficulties I encountered in the process and the possibilities that a multimodal approach to user interaction open up.

As various approaches were assessed to determine their feasibility, no approach was taken past an initial *proof of concept*, as this would have exceeded the scope of this project and it was preferred to comparatively implent diverse techniques to obtain a similar goal.

3 *Touch Typing* in theory

> "Do you not find," he said, "that with your short sight it is a little tiring to do so much typewriting?" "I did at first," she answered, "but now I know where the letters are without looking."
>
> Arthur Conan Doyle, A Case of Identity (1891)

Before having a closer looks at the system devised, it is import to define what is meant by *Touch Typing*.

Touch Typing(sometimes also referred to as *Home Row Typing*) is a style of typing where the hands have a fixed resting position and every finger has specific regions/keys of the keyboard assigned to them. After every keystroke/when not being used, the hands should return to their *home* position. The rationale being, that this technique allows the typist to type without having to look at the keys. Ostensibly this is advantageous both because it reduces neck strain by not forcing the typist to look down, but also as practice will commit the locations of the keys to muscle memory and typing becomes an automatic process which does not require the typist's conscious attention. (As an aside: It is worth noting that most keyboards have little locating features on the `f`- and `j`-key to facilitate locating them through the sense of touch.)

The invention of touch typing is generally attributed to Frank Edward McGurrin, a court stenographer from Salt Lake City, USA, reportedly invented the technique in 1888, well before keyboards were used for text input into computers.

In the context of typing efficiency it is also important to consider the keyboard layout, as many alternative to the standard `QWERTY` (or country specific variants thereof) exist. While this is certainly an interesting field in itself, it is beyond the scope of this work. Furthermore, the character that a given key is assigned can be seperated from the physical technique employed while pressing said key. While it can be beneficial to learn a new layout altogether while attempting to break out of sub-optimal typing patterns, these are two seperate aspects of machine typing and this work focusses solely on the former.

4 The Design Process

Many considerations went into the design process of the system and led to the multiple approaches implemented. The initial idea for the system was to create a keyboard that would only register a keystroke, if this keystroke had been performed with the correct finger. The system was therefor intended as a restrictive input device, that forces it's users to type with proper form, even if this occurs at the loss of speed. The idea was to create a practical obstacle to incorrect typing and force the user to gradually learn touch typing during every day use of the keyboard. Some of the noteworthy considerations will be listed hereafter.

4.1 Latency

As the system deals with keyboard inputs, which experienced user produce with great speed, there can't be any perceptible latency. Latency(also *input lag*) is easily perceived and can be very frustrating to the user. Vision based approaches were therefor seen with a certain scepticism, as 100ms processing time(an arbitrary threshold set during experimentation) are easily exceeded by

Figure 1: A screenshot taken in the popular touch-typing app *KTouch*. The user types out the text displayed at the top. Lessons need to be concluded within a specified time, while maintaining a certain accuracy. The lower portion of the screen displays the reccomended finger-to-key mapping which is expressed using a colour-code

image processing pipelines. Later scrutiny actually revealed that this aspect would turn out to be less critical than initially anticipated.

4.2 Bridging the Gap between Theory and Practice

While comitting a couple of minutes a day to some typing exercises using software specifically designed to teach *Touch Typing* can go a long way towards gradually changing one's typing habits, these tools focus heavily on typing out coherent natural text. The reality of keyboard usage in a desktop environment however is often defined by short bursts of typing followed by lengthy keyboard inactivity(e.g. when typing a search in the browser). Most learning software puts little emphasis on symbols and modifier keys, which, for instance when dealing with source code, make up a large percentage of the characters typed. In my personal experience, focussed work on programming problems or entering *the flow* of writing documents using LaTeX also tends to loosen new typing habits and make past poor form re-emerge. As such committing regular hours to typing exercises may be a useful way of initially setting one's technique up, but still leaves a long way to full internalization in everyday use.

Similarly to the consideration of latency, it is paramount for the system itself not to get in the way of the users' typing. The users are conditioned to type in a certain way. Thus it should feel just like typing normally, as the chances of permanently changing one's typing habits are greatest, when there is no perceptible differnece between when the system is active and when it isn't.

4.3 Cost and Convenience

The system should ideally be convenient to incorporate into everyday computer use and as such should not require complicated setting up or tuning. The initial cost should be contained, in order to make the system as accessible as possible.

4.4 Customizability

While "canoncical"/standard assignmenents exist regarding what fingers keys ought to be pressed by, this aspect of the systems policy should be how configurable by the user. This could be useful either to be more lenient initially and make the system more strict over time, or could serve the learnner to focus on certain aspects at a time.

4.5 Feasibility of the Implementation

As the initial idea for the system was to intercept the user's input, process it to assert it's conformity with predetermined finger-to-key mappings and pass it on conditionally, low level interaction with the operating system's keyboard input was required by a daemonized application. Implementing this to work across platforms is not trivial and even when focussing on a single OS requires advanced interaction with low-level OS processes. From a practical point of view, it was therefor not feasible to implement such a system and the focus was therefor rather to determine mechanisms of deriving the neccessary information within a focussed application, without intercepting the input.

While more contrived to all appearences and less conforming to the "Cost and Convenience" consideration, a hardware-based system connected to the keyboard and conditionally relaying the information to the computer seemed more realistic from this point of view.

5 The Approaches

The various approaches scrutizined generally fall into two categories. Vision-based and sensor-based ones. While two different approaches were experimented with for the vision based application, the sensor based approach was built around a conventional (albeit old) PS/2-keyboard, pressure sensors attached to the typists hands and a microcontroller combining the signals.

This section will adress these approaches, preliminary considerations, the implementation and some insights derived in the process.

5.1 The Sensor-based approach

As outlined in the "feasibility" design principle(4.5), there was a certain appeal to building a hardware-based system that gathered and processed all relevant data before the keyboard signal enters the computer. Some preliminary research revealed that an inexpensive version of the common *Arduino* developer board exists called the *Aruduino Leonardo* which on top the common analog and digital GPIO pins, comes equipped with a USB-controller that allows it to natively emulate input devices such as mice or keyboards. Perfect for this application .

As the availibity of this chip covered the logic requirements, the next question was how to obtain the neccessary readings to augment the the data coming from the keyboard, such that the finger employed for every keypress could be derived. I considered many options including, piezo-electric vibration sensors, tiny microphones equipped with low-pass filters to pick up only the low frequency thuds of key-presses. A challenge that every solution had in common however, was that it would have to work for at least 8 fingers and therefor would have to be inexpensive/simple enough to be replicatable across all fingers. To avoid excessive hardware debugging, I was also trying to avoid placing the sensor (or parts of the sensing circuit) on the keys of the keyboard, as this would require the sensing to work consistenly across 40+ relevant keys. Throughout all these considerations and factoring in the expense, I was left with no viable commercially available sensors, so i decided to fabricate gloves with pressure sensitive fingertips myself.

Though I was under no illusion that this process would pose a challenge, I thought that the fact that I only needed relative readings over a small window of time would make matters easier. The idea being, that all sensors continuosly read the pressure from the fingertips and only when a keypress occurs, the values are compared to a set of preceding samples and the finger with the largest relative change is considered the finger employed. To make matters even more tricky(this would later turn out to be the downfall of this approach) most "smart" fabrics sensitive to pressure, also respond to curvature/bending deformation. Considering the intricate(and diverse!) shape of finger-tips, their extreme tactile sensitivity relevant to the preliminary design considerations, the renouned difficulty of sewing good gloves(even amongst experienced tailors) and the fact that hands tend to sweat over time and thus also humidity had to be accounted for should probably have been enought to persuade me out of further pursuing this route. In a reluctance to let these challenges deter me, I decide to pursue the build anyway, as I had a simple enough design in mind, I hoped would solve the task adequately.

The Sensor Design

The custom design for the pressure sensitive fingertips, was based on a fabric I found online called *Velostat*(sometimes also referred to as Linqstat). *Velostat*, developed and manufactured by 3M, is a thin polymer impregnated with conductive carbon black which is predominantly used to make packaging for electronics sensitive to Electrostatic discharge. It is electrically conductive, and it's impedance varies, as it is deformed or pressure is applied. This property allowed me to produce the circuit I intended and replicate it 8 times(once for every fingertip). *Velostat* would be *sandwiched* between a two soft contacts(copper mesh). I would then measure the variation in electrical resistance using a simple voltage divider.

Figure 2: A piece of *Velostat* conductive fabric.

While rather simple from an electrical point of view, this plan proved to be quite tricky for a series of reasons, in actual implementation:

1. Contact between the *Velostat* fabric and the contacts needed to be uniform and realiable in spite of physical deformation. This was further complicated by the fact that most adhesives are either rigid or electrically insulating, thereby degrading the performance of the system.

2. Sewing through the *Velostat*, though possible, could lead to short circuits, when the thread got caugth on the copper mesh and pulled a thread through the hole created

3. Soft, stretchy fabrics generally suited to make fingertips for gloves, tend to absorb/cushion most of the pressure applied, thereby falsifying the readings from the sensors.

4. Iron-on patches with heat-activated glue proved a good material to connect the contacts to fabric and conduct the pressure well, as they are made of a dense mesh of non-stretch fabric, but heating the *Velostat* under an iron proved to destroy it's electrical properties. Bonding the fabrics using heat-activated glue was not an option therefor

5. The entire assembly needed to wrap around the fingertip as different parts of the finger are used for different keys on the keyboard.

6. Touching the copper with bare skin in the fabrication process favoured corrosion, changing the mesh's surface conductivity. Gloves were required for some steps.

7. The braided copper mesh used to distribute the voltage accross a large area needs to be reliably connected to leads taking the signal to the microprocessor. Copper is an excellent conductor not only of electricity, but also of heat, this means that if soldering is to be employed, these connections need to be made before the copper braid is bonded to the *Velostat* as the heat would otherwise destroy the fabric's special properties and short out the circuit.

Most of these insights were obtained in the process of prototyping various pressure-sensitive finger tip assemblies which resulted in a large collection of prototypes from variations on the assembly and fabrication process.(See Fig 3)

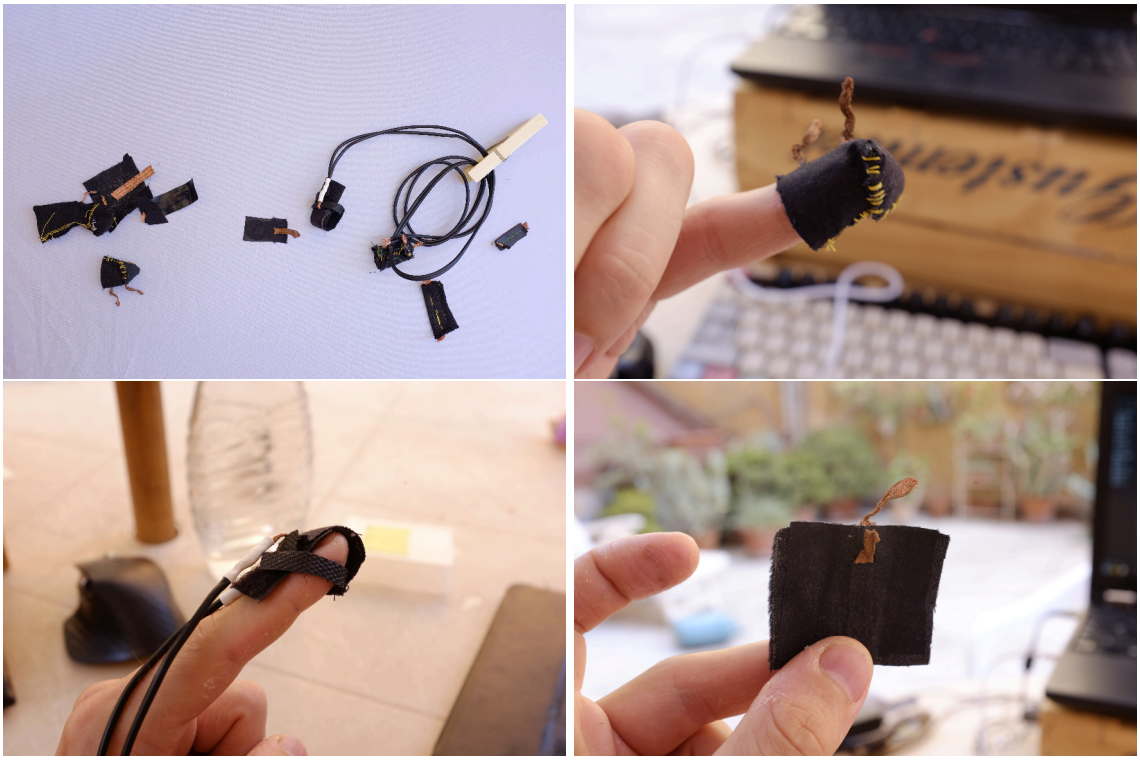

Figure 3: A selection of material/assembly experiments conducted to determine a functional and repeatable assembly for the pressure sensitive gloves.

The Assembly

Though, in the light of adversity faced, the prototyping phase dragged out for quite a while I eventually decided to move on to making two full fledged gloves(actually just sensitve fingertips). I

intended to see whether I would be able to make the system work and wanted to see this approach through, though I was starting to doubt the feasibility of the hardware-based stytem at this point.

Please see 7 for annotated images documenting the build if you are interested in the steps involved.

After successfully fabricating 2 leads with 8 pressure sensors, soldering all the electronics and connecting a PS/2 keyboard to the microcontroller, I succeeded In getting readings in real-time from all sensors. I further managed to relay the signal coming from the PS/2 Keyboard to the computer, though I had to jump through some proverbial hoops in order to make modifier keys work.

Sadly, the process of setting up the *Arduino Leonardo* was delayed by a frustrating 3-4 days troubleshooting the device itself. Being my first Arduino project I did not have the neccessary experience to immediately realize, that the unit I purchased from `arduino.cc`'s webshop was in fact faulty. It was only after having tried to make it work under 2 different operating systems, buying a new micro-usb cable and an ASP-programmer to flash the bootloader that I concluded that it was not user error, but rather a faulty device. The subsequent purchase of a second Arduino Leonardo confirmed this.

Conclusions

Though all requirements to implement the logic initially envisioned where met now, it was at this point that I decided to abandon this approach. The pressure sensitve tips work, however readings are in vastly different orders of magnitude, e.g. some fingers have a resting/base resistance of $\sim 15k\Omega$ while others are around $\sim 180k\Omega$. This in itself is not a problem and was actually expected, however it requires the system to be calibrated every time it is employed. The effort required simply didn't seem worth it, as the pressure sensitive tips, despite my best effort to make them as unintrusive as possible, are so bulky that I can't see anyone(including myself) actually typing with them for any length of time.

If I were to go back, I would probably change the approach to physical sensing altogether, as I have come to realize that tactility is paramount. I would possibly experiment with the smallest microphones available, use low-pass filters and a tactile mechanical keyboard, to change the approach to focus on vibration/sound. Hoping that one would succeed in sensing the vibration/motion only in the finger pressing the key. While this approach would probably face problems of it's own, it is clear that any layer of fabric on the fingertips is simply unacceptable.

5.2 Vision-based Approaches

The idea behind vision based approaches is quite simple. A camera is pointed at the keyboard, evaluating the hands position continuosly and determining the digit used for every key-press.

As mentioned in 4.1, latency was an important consideration going into the design of the vision based system. The fear, that processing latency would cause input lag, was based on the assumption, that high resolution frames would have to be continuosly processed in order for the system to function correctly. It was also feared, that high typing speeds could possibly exceed the system's real-time capability.

These considerations led to two seperate approaches, both of which were expected to mitigate performance issues, by doing without a complex image processing pipeline. The more traditional of the two uses simple colour images and coloured fiducials on the typists hands to discern fingers, while the other uses a low latency finger tracking solution based on machine learning. Large parts of the complexity of the latters implementation could be avoided by using a pre-trained model.

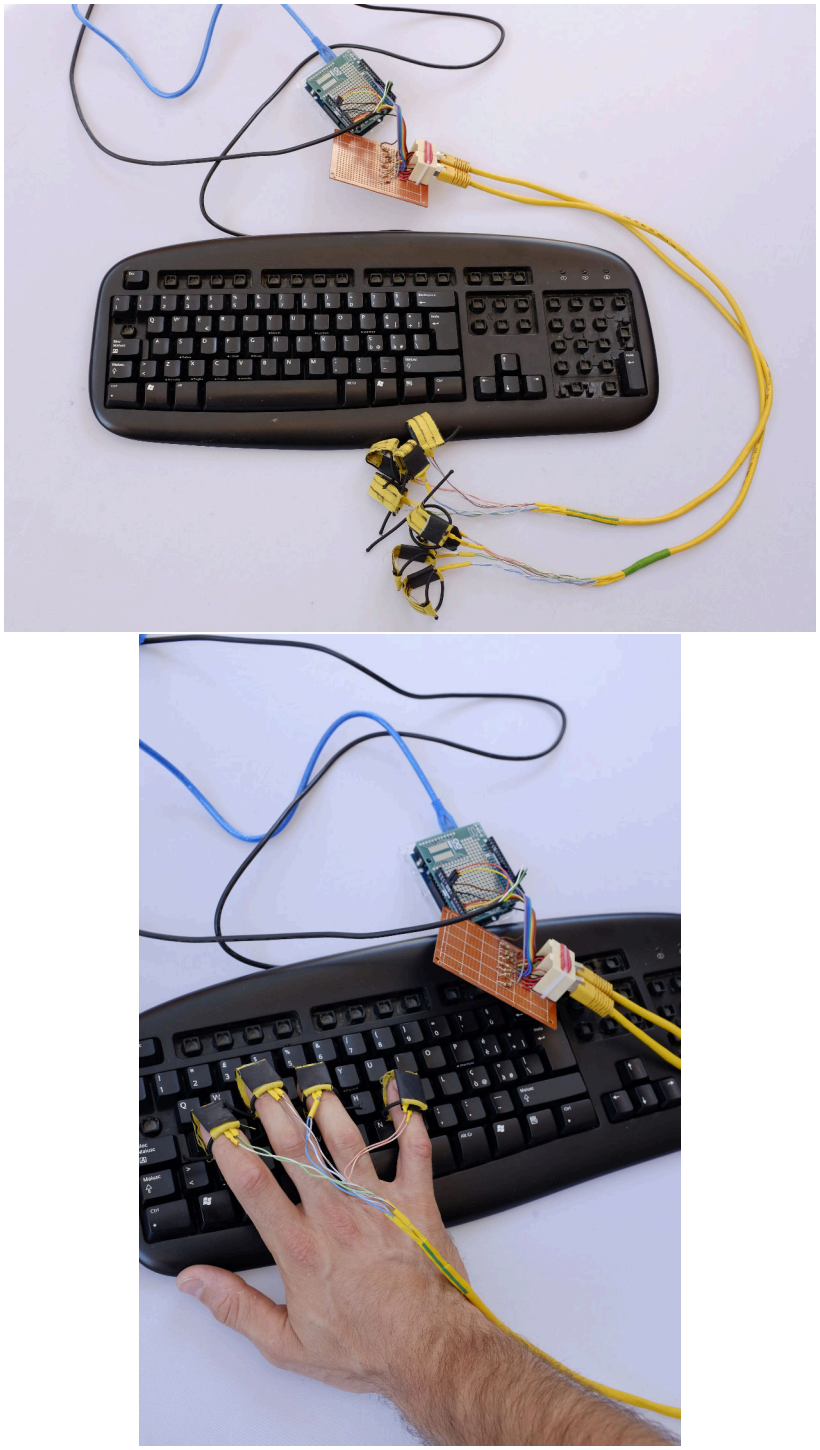

Figure 4: Two pictures of the final assembly.

The Camera

While webcams, inexpensive colour imagers, are built into most computers nowadays, both systems described here, require a vantage point from above the keyboard to precisely determine the fingers position. An ideal system would have a camera pointing orthogonally at the keyboard. The camera

should be equipped with a relatively long focal lens to mitigate excessvie perspective effects and should be calibrated to eliminate any distortion in the image. The orthogonal angle to the keyboard is favourable as both systems are required not only to identify the hands' digits in the *image space*, but also to create a correspondence between the image coordinates and the *keyboard space* to determine what key a finger resides on. In an attempt to determine this correspondence with relative accuracy despite low-resolution images coming form a webcam, little perspective disortion is favourable. While it is conceivable that the aspiring touch-typist would set up such a system at their desktop working environment, the accessibility of such a system would be reduced by such a requirement and it's lacking portability.

This is particularily true because many people(like myself) use laptop computers nowadays, which are moved/stowed away between uses. As I didn't have access to a external capture device myself, I fabricated an inexpensive mirror on a bracket out of brass wire and a $3cm \cdot 3cm$ mirror salvaged from make-up packaging. This mirror redirects the laptops webcam to the keyboard, thereby framing the relevant part of the laptop's built-in keyboard (see 5).

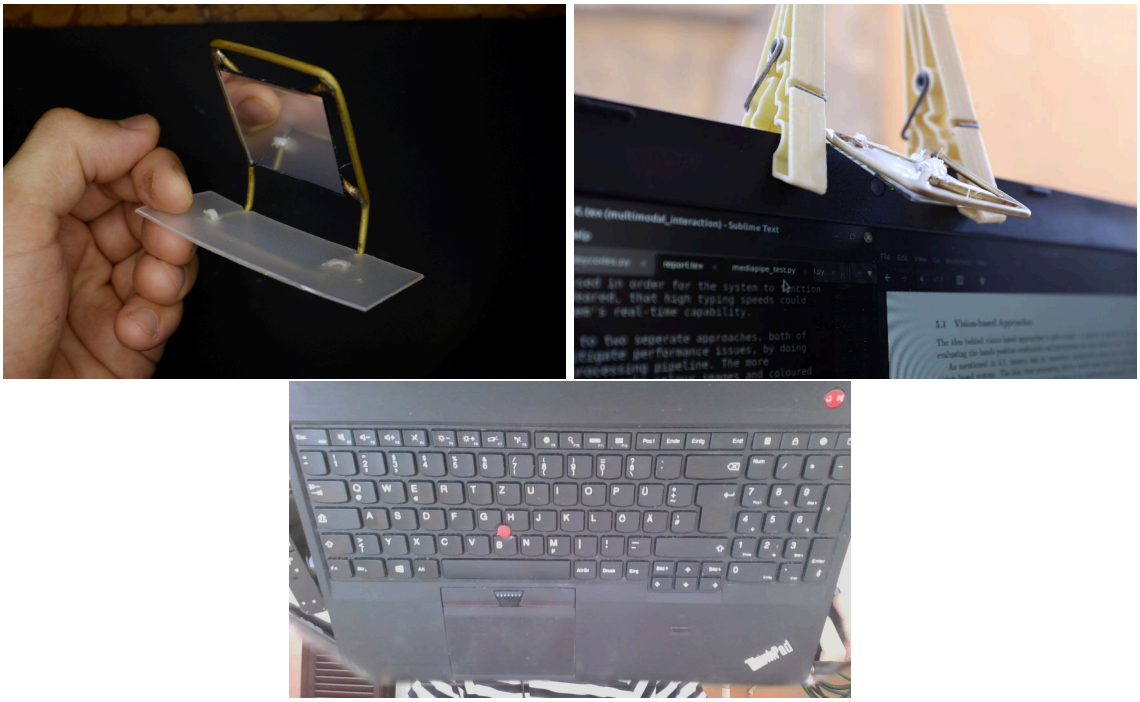

Figure 5: A little mirror redirects the user-facing built-in webcam towards the keyboard resulting in an (almost-) bird's eye view of the keys.

Keyboard space* vs. *Camera Space

Before taking a look at the two approaches envisioned to tackle the vision-based system, it is worth briefly adressing the steps required in tracking users' typing. The steps involved in determining the conformity of a given keypress are the following:

1. Capture an image once a key is pressed.
2. Determine the position of hands in frame(and the *handedness*, though crossarmed typing is unlikely, this aspect is relevant if one considers the case of only one hand being visible or even

the letter`b` for instance. A key which may just as well be pressed by by either index finger but is arbitrarily assigned to the left hand.)

3. Determine the position of the keyboards keys in frame.
4. Calculate the pairwise distance of all fingertips to the key that was pressed.
5. Consult the finger-to-key mapping policy whether the keypress was legal.
6. **Optional:**log/store stats.

While most of the complexity of the implementation of this high-level outline of the steps involved is buried in reliably determining the position of the hands, determining the key location is actually not trivial in itself. Assuming a static relative position of the keyboard and camera the use of a *a priori* calibration frame of the keyboard is conceivable. This could either be performed automatically using a combination of image segmentation and OCR(using the keys' legends) or the user could also be prompted to select every single key in the image and interactively label it by pressing the corresponding key on the keyboard. Both of these approaches are unlikely to be very robust. The first one on account of the notorious difficulty of reliable image segmentation, unusual keyboards, worn legends and many more. The latter, because of user error and the prohibitive nuisance of having to go throught approx. 40 keys of calibration every time the system is to be employed.

From a theoretical point of view it is possible to use 4 points on a plane(e.g. the 4 corners of a flat keyboard) in the image recorded by the camera(assuming this image doesn't suffer from excessive abberations) to determine a homography linearly projecting objects to their locations within a predetermined map of said plane. The projection to this space(here referred to as *keyboard space*) given the edges, locations and labels of all keys in this space. It is then possible to determine the position of the fingers, by reprojecting their location in *camera space* onto the plane of the keyboard(i.e. *keyboard space*)

There is an inevitable reprojection error inherent to this approach, however curiosly this error is conveniently minimized, when the objects fall onto the plane determined by the 4 calibration points. This means that our predictions are most accurate for the fingers are actually being used to type. Furthermore, the relative position of camera and hands tend to project the error away from the keys being pressed, which further aids the systems accuracy.

Based on these observations of the geometry involved and some past experience with photogrammetery, I instead decided, to tailor the prototype only towards the built-in keyboard of the Thinkpad L560 it was developed on. To this end a planar, distortion-free two-dimensional "map" of the keyboard was required. While a carefully calibrated camera setup or a 3d scanner could have been used to this end, I took advantage of the flatness of the keyboard and opted for a more pragmatic approach: The keyboard was simply placed on the tray of a flatbed document scanner and scanned. The resulting distance-preserving, high-resolution image was then manually divided into easily seperable tiles representing the individual keys. Every time the prototype application is executed, the user is prompted to click on the 4 corners of the keyboard in the webcam's image and the projection is determined on the fly. As the calibration of the system is split into two steps, it is sufficient for the location of the keys to be known in *keyboard space* and for the homography to be determined using 4 user-selected keypoints.

A Colour-based Implementation

As performance bottlenecks were anticipated for the vision-based approach, the most simple, yet reliable way of determining the fingers' position was needed. Using coloured markers(fiducial markers)

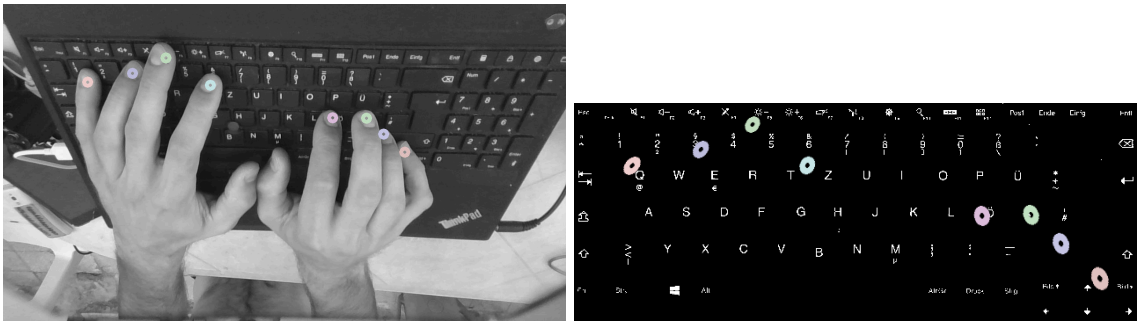

Figure 6: A screenshot from the app's view of the hands and the resulting projection into *keyboard space* are displayed. The eliptical shape of the fingers position demonstrates how the image is sheared. For both geometric-(the object is closer to the camera's principal point) and optical(the wide image suffers from some barrel distortion) reasons the accuracy on the pinky fingers is not very high. In spite of this, the left is clearly the closest finger to the 1 being pressed when the image was taken. The angle on the camera was also exagerated, to visualize the perspective shearing and in normal typing arrangements, the reprojection error is not as drastic

Figure 7: The laptops built-in keyboard is *mapped* by creating a distance-preserving scan using a flatbed scanner. The image on the right represents the result(transparent pixels are displayed yellow)

on the finger tips, limiting the region of interest(ROI) to the surroundings of the key pressed(using previous calibration) and then comparing the intensity of the colours present in that ROI seemed like a straight-forward approach to solving this problem.

Using the tools in the "traditional" computer vision toolkit, this would have been a reasonable way of approaching such a problem with the correct lighting could probably give good results while keeping down hardware requirements. Experimentation however revealed, that the use of pre-trained hand-tracking software was efficient enough, reliable and did away with the need for markers/gloves at the same time.

For practical reasons, this option was therefor not pursued any further. It's accuracy could probably have rivaled or even surpassed the ML-based approach with judicious implementation. It is a drop-in replacement for the "locating" step in the processing pipeline(step 2 in 5.2), albeit requiring specific gloves, extra calibration and more susceptible to changes in lighting.

It is worth noting, that though limited to simple cotton gloves, even this requirement already

Figure 8: The coloured markers on a glove. Finger positions are determined based on the dominant colour on a key/in a key's surroundings at the time of the keystroke.

conflicts with the design consideration 4.2 as it faintly influences the tactile experience of typing. Possibly this could have been improved using different markers that don't cover the sensitive fingertips.

The Machine Learning-based Implementation

In implementing a version of the system requiring no markers added to the users hands, while not sacrificing speed, Machine Learning based pose estimation was the obvious choice of technology, as recent progress in this field (especially in it's performance constrained application for mobile use) has led to blazing fast and highly accurate libraries. Google's `mediapipe` was chosen, as it features a `Hands`(`https://google.github.io/mediapipe/solutions/hands.html`) module which does just that. It takes an image, finds one or more hands, annotates them with points for every joint and guesses whether they are right or left hands. Everything the system requires and more. As this library is fairly well documented and it's use in conjunction with `OpenCV` seems to be widespread, it was easy to incorporate into a working prototype.

Though performance of this library was incredibly good, with little to no configuration required, some adjustments had to be made to the camera setup, as the pre-trained networks ability to recognize hands seems to be adversely affected if parts(think of the wrists, if the perspective is fully orthongonal and only contains the keys) are cropped out of the frame. Unfortunately, shifting the camera, away from the ideal, centered position(above the keyboard), the radial re-projection error caused by lifted(i.e. not located on the keyboard plane) fingers is amplified. This problem could have been adressed by using an external/freely movable camera which sadly wasn't available to me for experimentation.

Interesingly, most of the problems that usually plague single-view computer vision setups are alleviated either by the nature of the problem, the geometry of the setup or the specificity of the

area being evaluated at every keypress, i.e. the location of the key is fixed, known and the ROI fairly small.

Overall I consider the performance of the ML-based prototype to be absolutely satisfactory(well beyond my initial expectation) The only caveat, is that the application requires some pretty advanced libraries and thus cannot easily be run on a simple microcomputer serving as an intermediary(such as a Rasberry Pi).

6 Conclusion

It was very interesting to approach the problem of multimodally augmenting the keyboard interaciton from different angles and has been a learning experience challenging and enriching in many ways.

The simplicity(factoring out the pre-trained model running in `TensorFlow` under the hood obviously) and functional adequacy of the vision-based approach was a pleasant surprise and it would have been intersting to use all the funds spent on fabric, electronics and microcontrollers towards something like an Nvidia Jetson Nano(cost approx 100€, though reusable for future projects): An embedded microcontroller with enough computational power to handle running the `OpenCV`, `mediapipe.Hands` and the necessary GPIO pins to handle the Keyboard input and relay it to the computer. It would further be interesting to experiment with more camera angles. I initally considered the fact that I could only use a mirror mounted to the built-in webcam a tedious limitation, it later proved to be a valuable step towards making the system as accessible as possible though. Automating the calibration step of identifying the keyboards location in the frame is another interesting aspect to explore. Possibly through the use of fiducial markers in the corners or similar. Automating this side of the processing chain(possibly for keyboards of known shape and layout) could lead to a fully autonomous pipline.

The hardware-based approach had a steep learning curve, substantial monetary expenses(probably almost 100 euros, considering I had to buy 2 arduinos as one was defective) and incredible expense of time spent prototyping, planning, experimenting and troubleshooting. I would estimate that about 40h of work went just into the development of the wearable pressure sensors.

It's eventual abandonment could be considered a failure, though I don't see it as such. Simply a testimony to the intricacies and challenges of solving problems in hardware, as the computational alternatives get more and more powerful.

7 Appendix 1: The build process of the Hardware-based approach

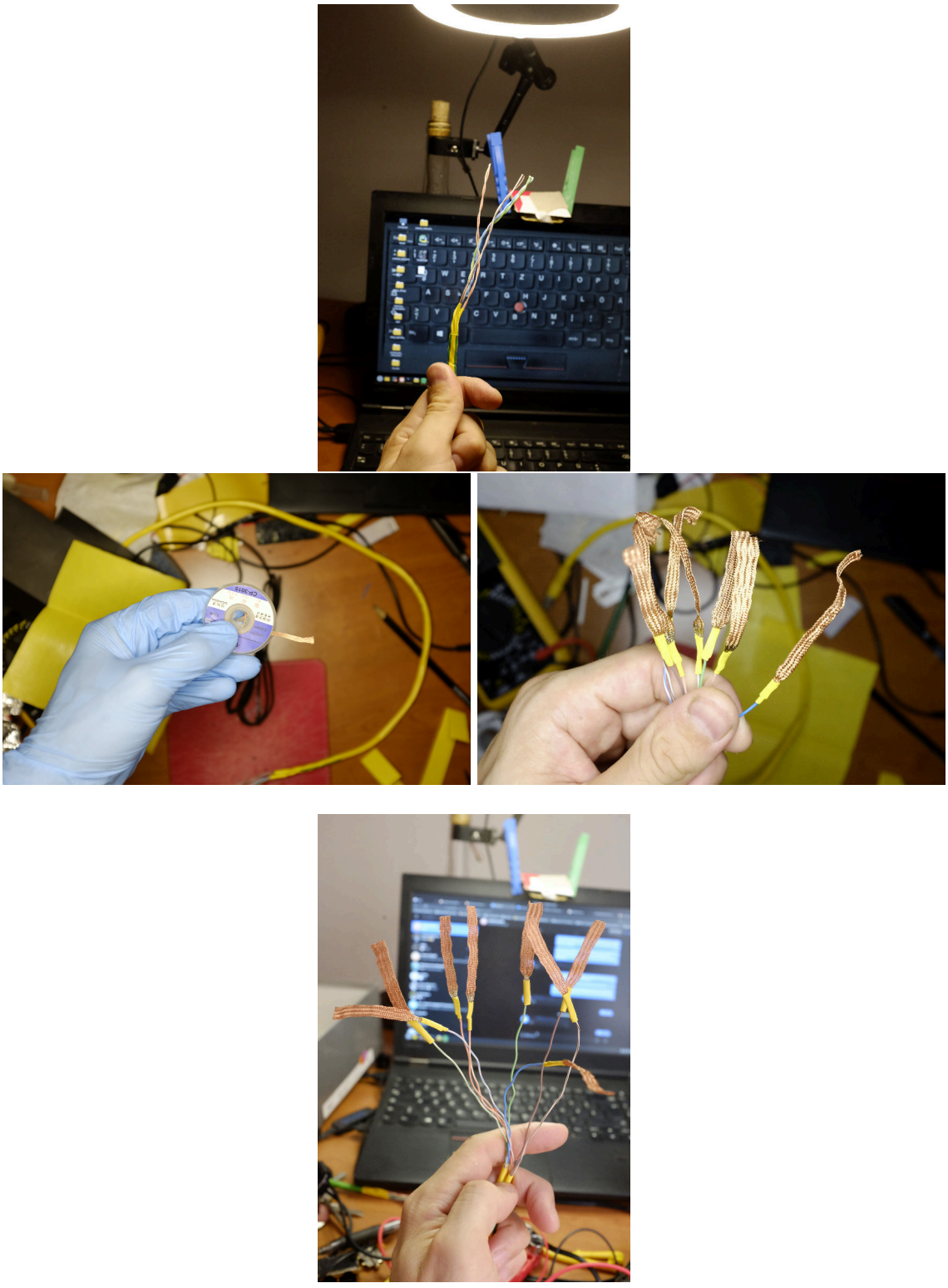

Figure 9: A CAT-5(ethernet) cable was stripped and all 8 leads were tinned. They were then soldered to strips of copper braid(desoldering wick repurposed for this application) and insulated using heat-shrink tubing
. When applying a little friction along the length of the desoldering wick, the braid can be made to "bunch up" whereby the width is increased and the strand becomes thinner. This technique was employed to maximize the surface area.

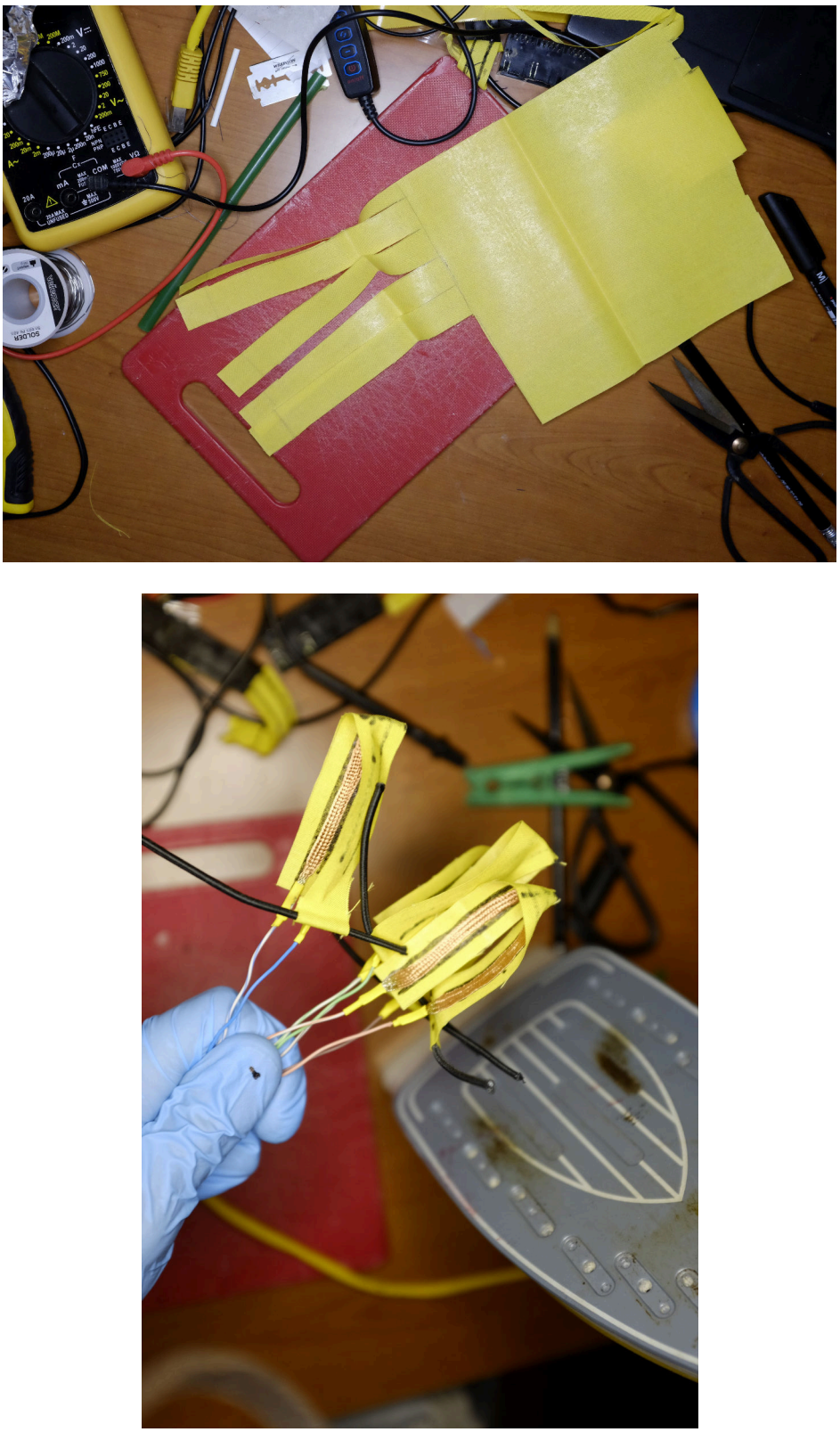

Figure 10: Strips of iron-on patch fabric are cut to length. The exposed copper braid is bonded to two opposing ends of the strip using the thermal glue and a hot iron.

Figure 11: A layer of *Velostat* is sandwiched between the copper braids and a sewing machine is used to sew along the outside perimeter of the fabrics strip, avoiding sewing directly throught the copper braid as this might cause a short-circuit.

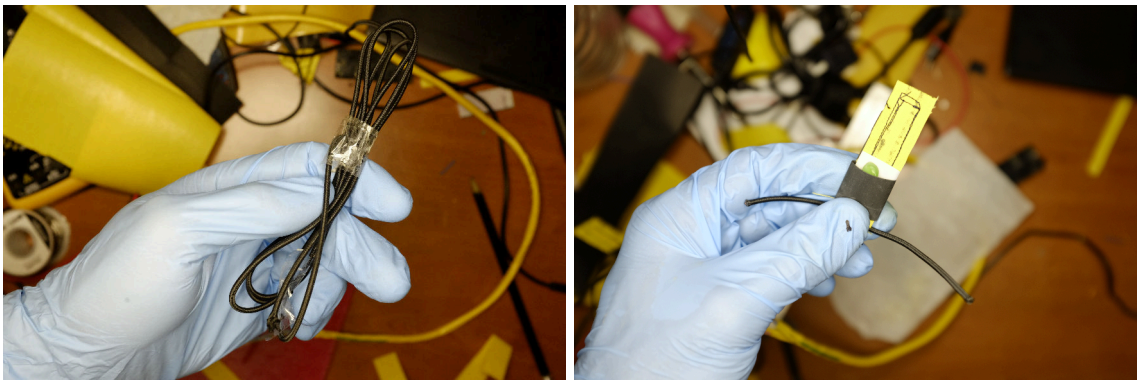

Figure 12: An elastic string that was previously attached to the yellow fabric will later provide adjustability for fingers of different sizes. A "jacket" of heat shrink with a reinforcing piece of plastic doubles as strain-relief for the wire's point of contact and provides rigidity on fingers rear side(above the nails) to keep the assembly from laterally "rolling" off the finger

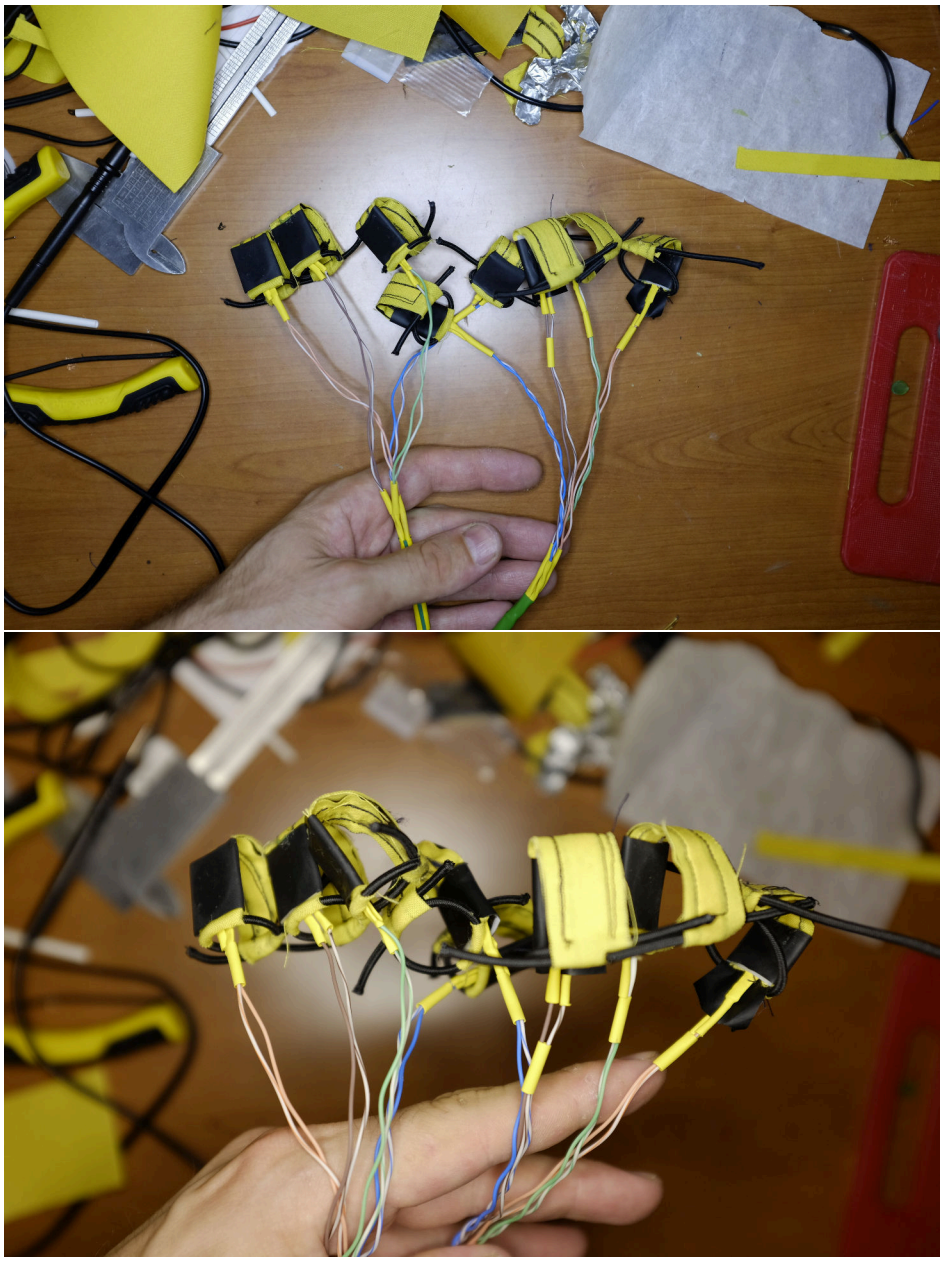

Figure 13: The completed sensitive tips. It later turned out that the sewing on 1-2 tips needed to be refined, as there was excessive resistance on one and a short circuit on another.

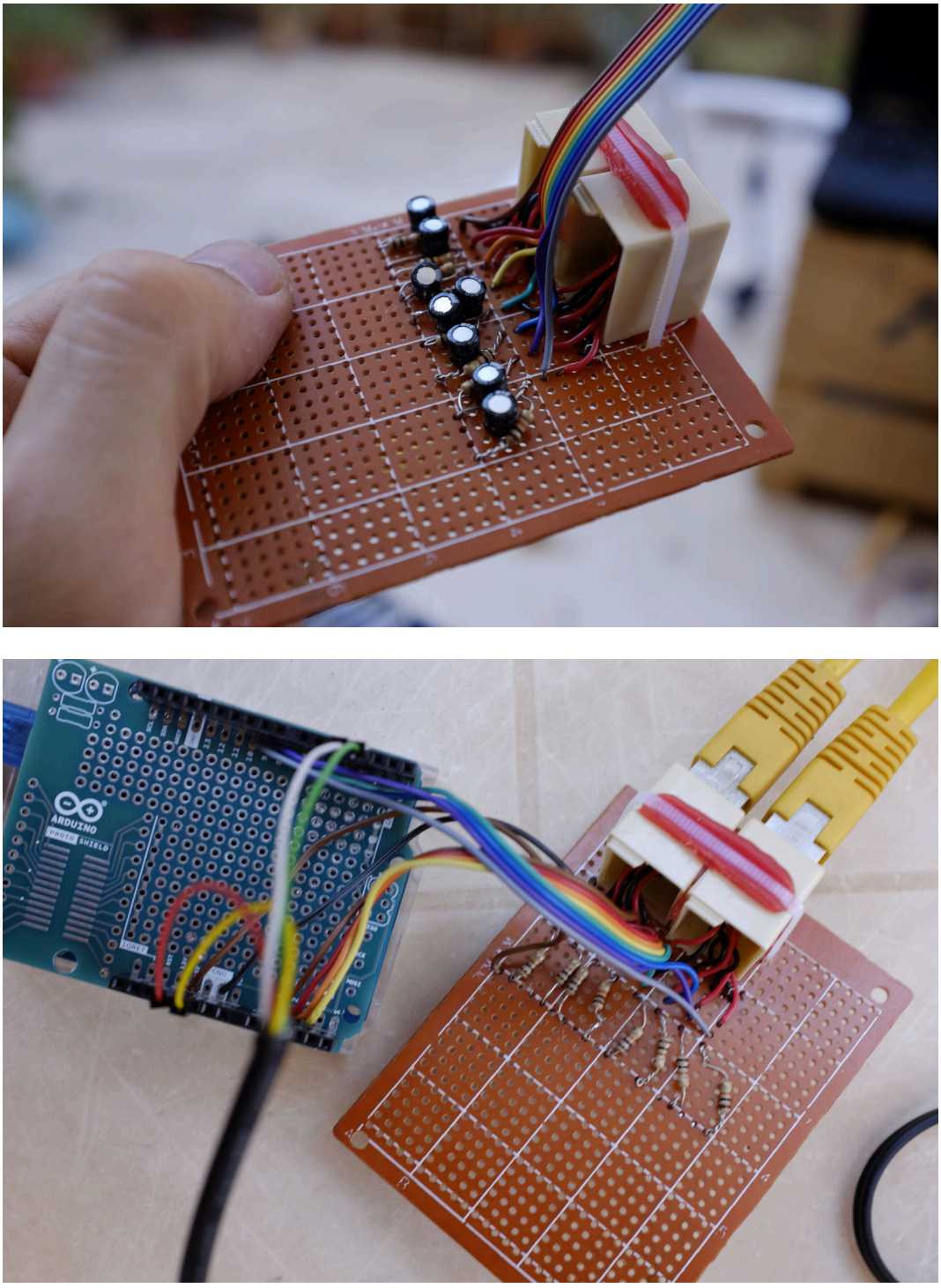

Figure 14: A detail shot of the electronics assembled on a small bread-board. The RJ45 jacks were salvaged from a extension coupling and crudely attached to the breadboard. The leads coming from the breadboard are attached to a *proto shield*. A PCB that connects to the Arduino using header pins and reliably connects all relevant pins. The capacitors visible in the first image were actually removed, as their capcitance was chosen to high and therefor their effect wasn't limited to removing jitter as hoped, but rather erroneously introduced delay in the sensor's response to pressure.