Final project report

EECS 452
Fall 2019

Title of Project: Hand it Over

Team members: Christopher Herran, Deepen Solanki, Liam McNeil, Saruf Alam

Executive Summary

Hand it Over is a platform for Real-Time Human Hand tracking and robotic biomimicry. We use a computer vision-based approach to both detect and track a human hand in real time to drive a 3D-printed robotic hand to move based on detected keypoints. Nowadays, low-cost robots--while armed with autonomous mobility and state-of-the-art computation systems--in general maintain relatively simple functionality, capable only of simple tasks such as pushing, pulling, raising, and grabbing, limited by few axes of motion. Our project aims to provide human hand-level dexterity to a robotic hand, allowing for intricate motion viable for more complex applications, with an easily accessible and transferable remote environment.

The use-case scenarios for such a system are abundant. Imagine a bomb-defusal event or an emergency surgery needed to be performed but the doctor is out of town. We could have the robot present physically at the scene sending live feedback to a base and the doctor/engineer could virtually perform the surgery/defusal. This technology would enable a safer standard for dangerous high-skilled labor, and greatly increase the accessibility to high-skilled work in industries worldwide.

Our original goal was to design for complete reproduction of a reduced hand model, comprised of: single-axis finger movement, 120 degree wrist rotation, and pitching of the palm on a naked human hand. For our final implementation, we had to scope down the system to movement of all fingers – individually controllable - and 90 degrees rotation of the wrist. The demonstrated system has these capabilities with a few physical constraints in wrist rotation. Additionally, the subjects must wear a custom-made glove, while keeping their hand in a constructed “stage” where the camera is setup.

Our project can broadly be divided into two sequential sections: human hand detection and robotic hand control. Raspberry Pi 4 was used for human hand detection related signal processing and computer vision functions, while Arduino Uno(s) and servo motors were used to control the 3D printed robotic hand. The major signal processing elements for hand detection included color thresholding and detection, region of interest identification using contours and convex hulls, Kalman filters for tracking (this idea was implemented but discarded later) and distance mapping. While on the robotic hand control side, we used SPI for data transmission between the Raspberry Pi and Arduino and Pulse Width Modulation to drive the six servo motors.

Project Description

Motivation: Robotics has always caught the attention of the public due to its potential to change human life. However, given recent technological advancements, only now has robotics become a pragmatic and affordable solution to the various problems faced by society today. With costs being driven down, robots are becoming competitively priced and can truly be envisioned as the best approach to achieve certain tasks. The robotics market was valued at USD 31.78 billion (Mordor Intelligence) in 2018, and is projected to grow 24% in the next five years (Mordor Intelligence). A subset of this market, medical robotics, has shown the most promise when it comes to integration of machinery with conventional human-powered tasks. Companies such as Auris Health and Mako Surgical, developers of medical robots, are being bought for multiple billion dollars in an effort to bring such machines to the mass markets. With the increasing significance of robotics, it is important now more than ever to understand the fundamentals required to develop such appliances.

Vision: An important and advantageous quality of robots is their ability to be controlled remotely and precisely. This can allow us to work in environments unfit for humans or reach areas we would not think possible. Motivated by this strength in robots, we wanted to make our own robotic hand that can be controlled remotely in real time by the user.

We plan on building a system, in which our hand and finger movements are tracked in real time, and control the hand and finger movements of a custom printed robotic hand.

Overview: While our vision is quite simple and palatable, realizing all aspects of it requires meticulous planning and detail. We will now cover all the abstract processing steps required to achieve our end product.

Image Processing Steps:

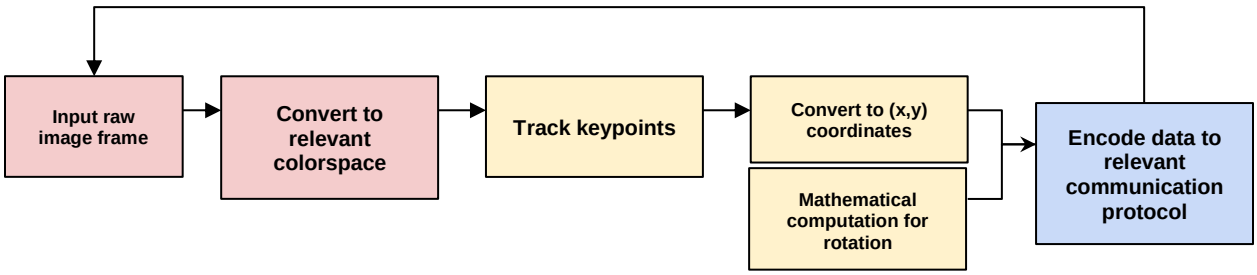


Fig 1: Abstract Block Diagram for Image Processing Structure

Our image processing section focuses on the high level capture and formatting of image data. Because we want the user's hand to be the controls for our robotic hand, we need to perform real-time processing to track what the user is doing. The outline of this can be seen above in the block diagram, but will be talked about in further detail below.

This first step to the image processing is to capture a frame and then convert it to the relevant colorspace needed for all the filtering. Choosing what colorspace to use for our project was a non-trivial task as there is a lot of research on which colorspace conveys the most useful data for human skin.

The second step in our image processing involves multiple stages. Initially, we have to filter for our keypoints such as our fingertips and the center of the palm. This filtering also involves some hysteresis based method to make sure we have a prediction of where a keypoint is even if it is occluded. Next, we have to perform some sort of mathematical operations to convert the location of these keypoints into (x,y) coordinates + rotation. This stage has to be done through trial and error to see type of conversion is most compatible with our hardware.

Lastly, we take our processed data and encode it into packets that we can send to our hardware. Once the information has been sent out, we go back to stage one and start again for a new frame. This cyclical property is essential as it is single handedly what makes our processing a real-time operation.

Digital Control:

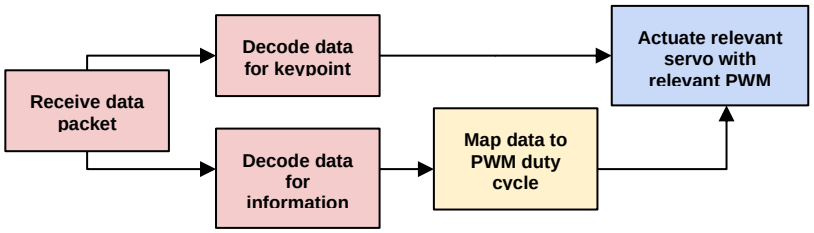


Fig 2: Abstract Block Diagram for Digital Control

Our digital control section focuses on receiving the processed data and converting it into the relevant signal that is fed into the hardware itself. This stage is carried out on a more hardware capable processor and thus it is run on secondary computing system while the image processing occurs on the primary computer.

We firstly have to receive the data packets sent by the image processor, and then decode it to find out which keypoint is being referred to, along with what data is being sent about that keypoint. We then have to convert our data into a relevant duty cycle. This is because our robotic hand, as will be specified later, is powered by servos. Finally, we actually actuate the relevant servo with the duty cycle we deduced, in order to actually move the hand.

Breakdown of Project:

Now that we have covered the abstract progression of our project from start to end, we can start covering the specifics. In this section, we will go over each stage mentioned earlier and then describe the final approach we settled on. The approaches we settled on (This has been covered here instead of the appendix to justify our conclusions better and ensure a smooth flow of events in the report)

Colorspace Conversion and Keypoint Tracking:

To begin, we first looked at how to implement our tracking mechanism. Our initial idea was to allow users to move their bare hand, and we would analytically extract keypoints from it. We started by exploring the YUV colorspace as it had good background with tracking involving human skin, explored it thoroughly and found that it had certain limitations that could not be overcome for our implementation. Our findings are discussed in detail in *Appendix, Section 1 – YUV Colorspace and bare hand tracking.*

HSV with Glove Approach

Due to YUV's weaknesses, we decided on a more straightforward approach to filtering and tracking by using the HSV colorspace. We hoped taking this approach would make our code faster and more accurate. An overview of our algorithm can be seen in the block diagram below:

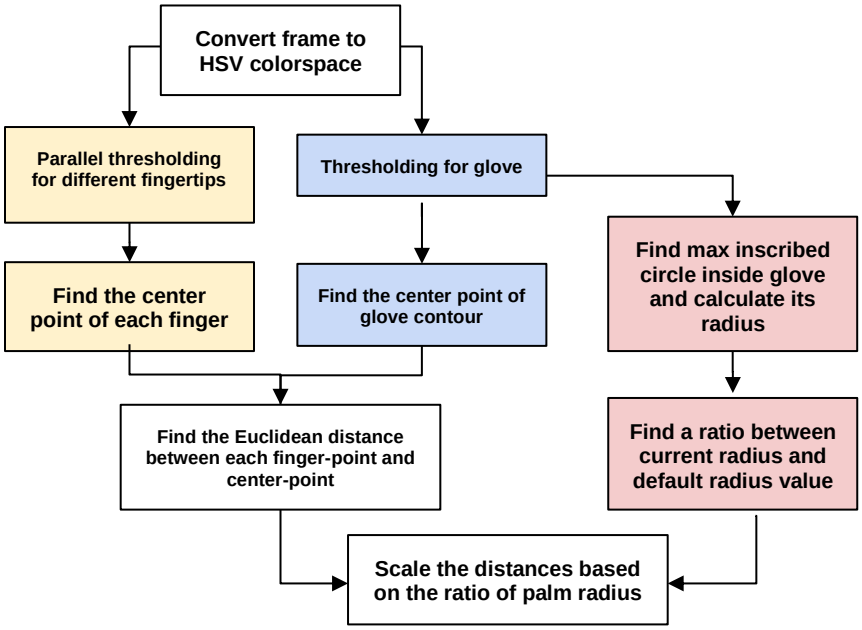
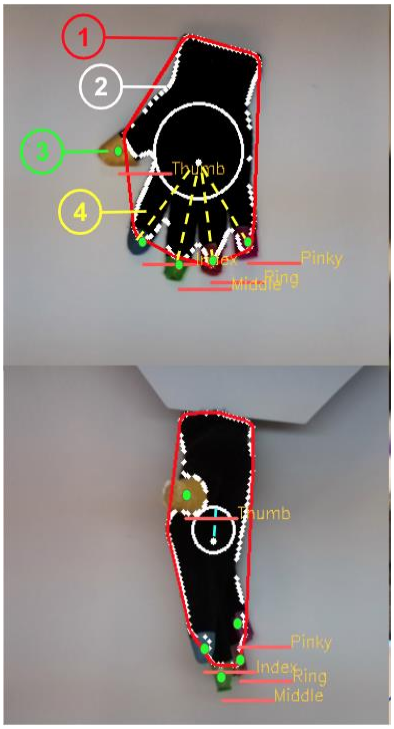


Fig 3: Block Diagram of Algorithm for HSV based Finger Tracking

In the HSV colorspace based algorithm, we set a physical constraint of a black glove with fingertips of different colors. So, instead of being able to use their bare hand, the user would now have to wear a custom-made glove. To track the fingers, we set custom thresholds that filter everything out but the relevant tips. We also run another threshold to filter everything out that isn't the glove. Next, we find the (x,y) of the centroid, for each of these contours. This results in us having the (x,y) coordinates of every fingertip and the center of the glove. We then calculate the Euclidean distance between each fingertip coordinate and the center of the glove. This distance corresponds to how much the fingers are bent.



1) Convex hull of the glove
   a) Find the center(x,y) of the hand

2) Filtered hand contour of the glove, used later for the max inscribed circle

3) Each of the fingertips had different colors and were filtered for in order to find their (x,y)

4) The Euclidean distance between each fingertip and the center (x,y)

Fig 4: Walking through some of the features of the algorithm as seen on a live camera feed of the glove

However, different hands and different distances from the camera can change the meaning of the raw distance. The same number for a distance can either mean the finger is bent, or that the whole hand is just a lot farther away from the camera. To account for this, we added distance scaling to the algorithm. Parallel to the aforementioned steps, we find the maximum inscribed circle inside the contour of the glove. We find the distance between fingertips (for open and close) and the center of our hand contour when the hand is at its farthest point and closest point and note down the palm circle radius for both. We also did a few values in between and approximated how much we need to scale the values of distance based on the palm radius. We then scale every calculated distance with this ratio. This adjusts for the varying hand sizes and camera distances that inevitably occur in real world scenarios.

Fig 5: Live code printing the palm radius of max inscribed circle. As you can see, lifting the arm increases the radius, allowing for us to use it as a reference for all our distances.

The HSV code was quite successful. The filtering and distance calculation only took 0.035 and 0.030 seconds respectively. That was much faster than YUV. Furthermore, as can be seen in the diagrams above, the tracking was being done quite accurately. The fingers weren't being confused for one another and the code wouldn't simply break down if there was any sort of overlap between the hand's extremities. Due to this increase in accuracy and speed, the HSV approach was the preferred choice.

Our original idea was to make Kalman filtering the backbone of the project. While rotating the hand more than 45 degrees, or coming up with different gestures, there were situations where we thought our fingertips would be occluded. In order to make sure we still had a relevant location for that tip, we tried implementing this filter which could predict the location of the fingertip based on its past locations, even though the tip was hidden from the camera's sight. The detailed implementation and theory behind this approach is discussed in *Appendix, Section 2 – Kalman Filter*

However, the results of this approach were quite poor. Firstly, Kalman Filtering slowed down our whole pipeline to 2-3 frames per second. This is much too slow for real-time processing. Secondly, it wasn't very accurate. Within a couple frames of occlusion, the predicted location of a fingertip would become

inaccurate. Furthermore, this inaccurate prediction would persist even though the fingertip was able to be located again (we didn't have an override, we only used the predicted as our final location). The occurrence of this phenomenon can be seen in the figure below.

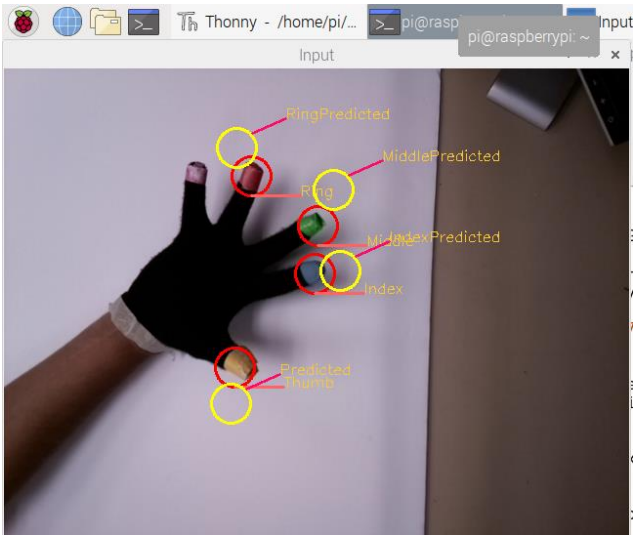


Fig 6: The Persistence of Inaccurate Prediction by Kalman Filtering

Because Kalman Filtering was detrimental to our effectiveness, we decided to exclude it from our final architecture. And so, in the case of tracking the location of the finger, we had successfully created an algorithm that used HSV filtering to locate distinct fingertips and found the distance between their coordinates and the center of the hand. We had to add a physical constraint of making a user wear a glove and not occlude any of the fingertips, but we thought it was a worthy constraint to add and have a fast and responsive hand for majority of the cases rather than have a slow responding one that can take care of all possible cases. This was one of the major trade-off decisions that we had to take.

The next part deals with our implementation of the rotation of the wrist. We approached this using two methods – inscribed circle and the rectangular strip. Due to the limitations discovered in the inscribed circle method, we decided to opt for the rectangular strip method. Complete details of the inscribed circle method are discussed in *Appendix, Section 3 – Rotation using inscribed circle*

Rectangular Strip on Glove

Inspired by how effective our finger tracking algorithm was, we took a similar approach to track rotation of the wrist. The diagram below covers the algorithm:

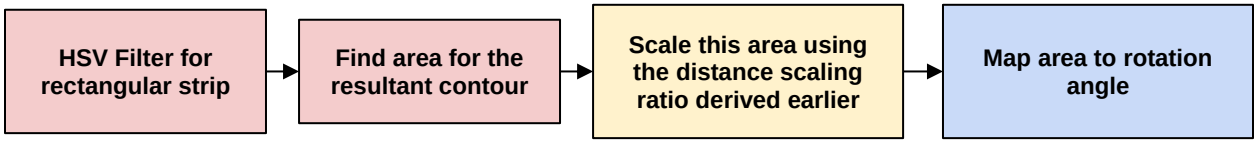


Fig 7: Block Diagram for Rotation Tracking Algorithm

We added a rectangular strip at the base of our glove with a color different from the colors of the fingertip. We carried out HSV thresholding for this strip and found the contour for it. We then found the area of this contour and scaled it based on the distance scaling we came up with before. Using the change in this area, we were able to map it to a change in the rotation angle of the wrist.

This algorithm was quite effective in achieving accurate palm rotation. Unlike the first approach, there was a drastic change in the raw area data as the palm was rotated. This allowed us to be precise with our mapping. Furthermore, because this data was independent to the palm radius, we were able to use our distance scaling on it. This ensured it was robust to different users, which wasn't the case in the previous algorithm.

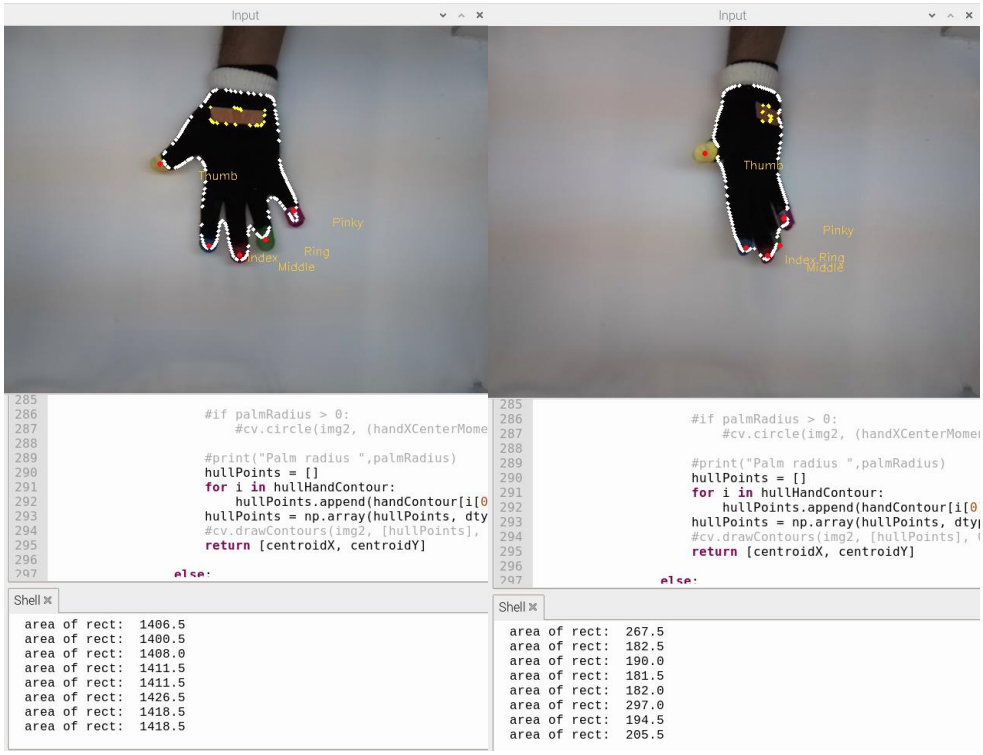


Fig 8: As can be seen, a small amount of rotation causes a huge change in the area of rectangular strip. This makes this approach less error prone than the initial approach we had

Conclusion:

Overall, we took advantage of adding a physical constraint to our problem, which was the glove, and created code that was quite fast and precise. Both our keypoint and rotation tracking involved using HSV filtering to find a specific contour. Then, for the fingertips, we located their x,y coordinates and found the distances between them and the center of the hand. For the rotation, we tracked the area of a rectangular strip strategically placed at the bottom of the palm. We then scaled all these measurements using our distance scaling algorithm and then mapped them to the relevant angles.

Control Mechanisms

With our image processing being finalized, we now looked at how we conveyed this information to the hardware. We needed to implement a system that could convey the information with as little delay as possible, and with as much precision as possible. For transferring data to Arduino, we first tried out USB communication. This was because USB is fast and does not need tedious wiring and pinouts to get started. It would need only one cable. However, when we implemented the USB communication, we saw that the data was intermittent, and the system was unreliable. Our next choice was SPI. It needs only four wires and is fast and robust. The resources available for implementing SPI efficiently on Raspberry Pi and Arduino are abundant too. More detail regarding this implementation has been covered in the *System* Architecture section. With this protocol, we explored two methods of digital control.

We had distances based on finger bending on the Raspberry Pi side and on the Arduino side we had 90 degrees of rotation of servo motors available. Our first approach was a one to one mapping of our finger open/close distances to the available servo angles for open/close. The approach and its limitations have been discussed in *Appendix, Section 4 – Near Analog Control of Robotic Hand*

Discretize Digital Control:

To tackle the problems we faced in the first scheme, we then decided to prioritize speed over precision. We took the following approach:

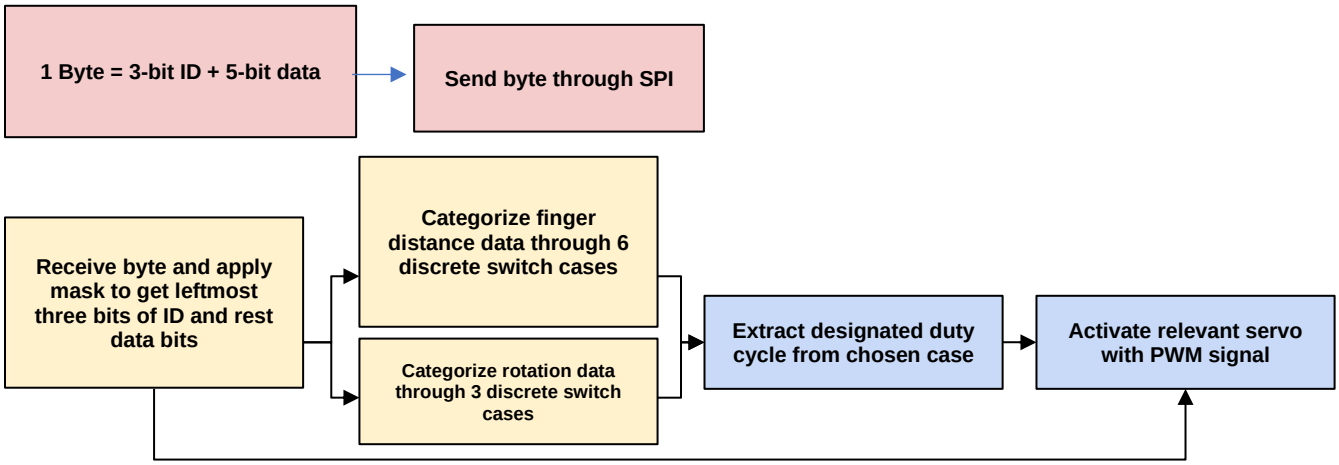


Fig 9: Block Diagram for Discretized Digital Control Algorithm

In this approach we package all the relevant data into one byte. The left-most three bits are assigned the ID of the extremity, and the right-most 5 bits are used to fit a linearly interpolated value of the measurement for that extremity. This byte is sent through SPI.

On the hardware side, we take the data and filter it based on what the ID is. We created 6 bins of hand positions that the robotic arm could achieve for the fingers and 3 bins of wrist positions. If the data is about a finger, we take the data and see which of the 6 thresholds it lies within. Based on that threshold we assign a duty cycle to the servo associated with the ID. We do the same thing for the wrist, however, instead of there being 6 thresholds there are only 3.

This approach worked much more smoothly than the previous one. While we gave up our precision, we gained much more stability and ultimately more accuracy than before. This is because our hand was actually able to keep up with the user's movements, even though it was not trying to reach the exact position the user's hand was in. SPI was no longer the bottleneck and we were able to reach around 12 frames per second for the whole pipeline. By discretizing the data, we were able to virtually smooth the data, allowing for the hardware to keep up with the real-time situation. In addition, we were able to make our system more robust because it was less sensitive to any sudden changes.

Conclusion:

By discretizing our data and prioritizing speed, we were able to actually get better digital control of our hardware. We were able to get a relatively fast and robust pipeline by keeping our control mechanism less precise.

System Architecture

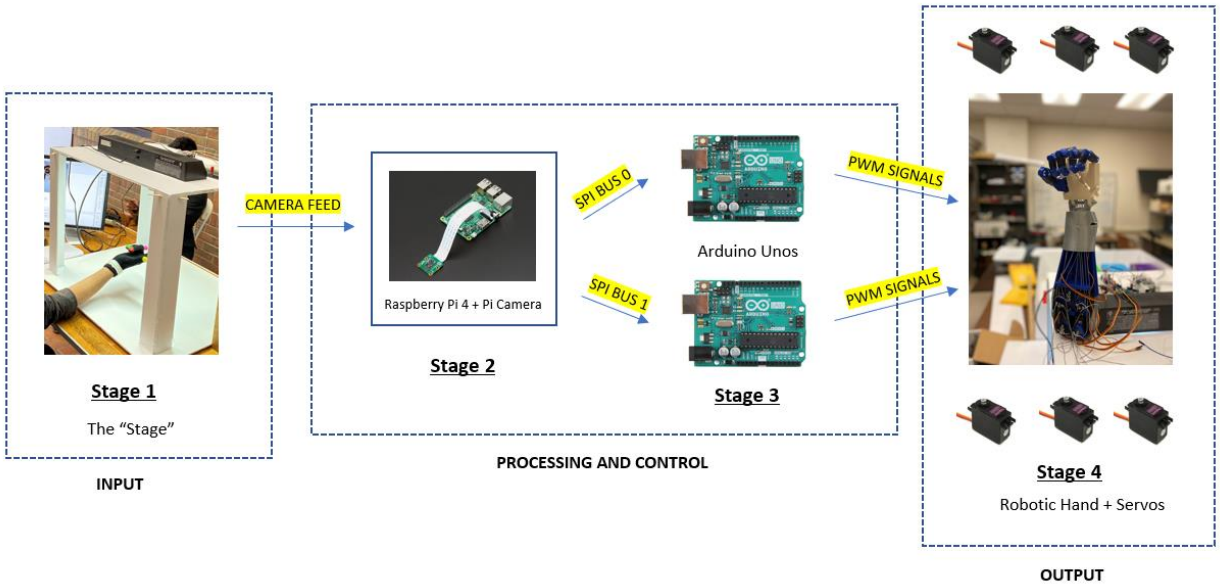


Fig 10: System Architecture

The system architecture can be divided into 4 sequential stages.

Stage 1

The user wears a glove and puts their hand inside a "stage". The stage is an enclosure made from foam board at the roof of which sit the Raspberry Pi and Pi Camera. The system is triggered only when there is an object inside the stage area.

Stage 2 and Stage 3

The Raspberry Pi Camera is fixed at the roof of the stage continuously recording video. The recorded frames are read into the Raspberry Pi for all Signal Processing and Computer Vision related processing. The Raspberry Pi and Pi Camera communicate via the *Camera Serial Interface* which is a specialized interface allowing the Pi to talk to Pi Camera. The video is read into the Raspberry Pi as continuous frames using OpenCV functions. The outputs of the Raspberry Pi's processing are six numbers between 0-90. Each of these numbers represent the angle that the servo motor of each finger and wrist on the robotic hand need to move to reproduce the position of the fingers of the subject's hand. The bulk of the processing involved in this stage has been discussed in the previous section.

Once the servo angles are generated by the Raspberry Pi, they are communicated to the Arduino via *Serial Peripheral Interface* or SPI as it is more popularly known. SPI is a synchronous, master-slave, serial

protocol that allows data transfer between two devices. For SPI transmission, four wires need to be connected between two devices :

- MOSI (Master Out Slave In) - This line carries data from the Master, in this case the Raspberry Pi, to the slave device.
- MISO (Master In Slave Out) - This line sends information from the Slave device, in this case Arduino Uno, to the Master device. For the purpose of this project, we do not need a MISO line connection as no data is sent back from the Arduino to the Raspberry Pi.
- SCLK (Serial Clock) - SPI is a synchronous communication protocol. A clock line is required to ensure synchronization at all times. This clock line is generated by the Master.
- CS (Chip Select) - Since SPI is a bus protocol, a Master is allowed to talk to more than one device. However, only one Master is allowed per bus. The number of slaves allowed depends on the number of address bits used and also the fan out characteristics of the hardware.

For our implementation, we chose to go with two Arduino Unos connected on two different SPI buses. The Raspberry Pi has support for two SPI buses, SPI0 having two CS lines and SPI1 having three CS lines. In theory, five SPI devices can be connected on these two buses but additional devices may be introduced with some library tweaking which would make some of the GPIO pins CS pins.

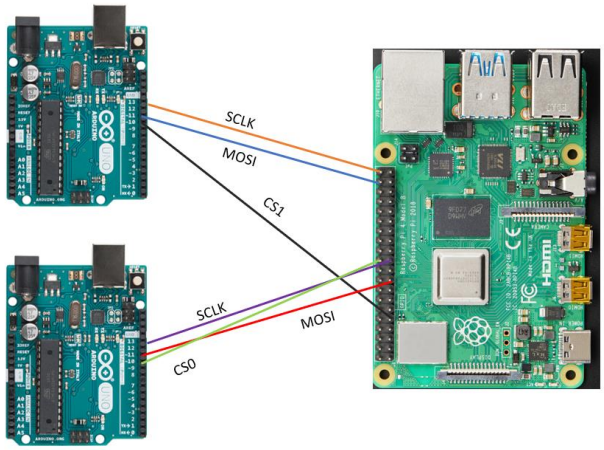


Fig 11: Connections between Arduino Unos and Raspberry Pi

One Arduino is on SPI0 and the other one is on SPI1. This was chosen to avoid congestion on the same line and to mitigate the problem of erroneous and jumbled up values. The Arduino was configured to receive values using an interrupt mechanism. Each arriving value is a number between 0 and 255 (1 byte). Once the appropriate operations are applied, the arrived byte yields two values: ID - a number between 0 and 5 determining which finger it is or if its the wrist, and a number between 0 and 31, determining the angle the servo motor needs to be moved. The encoding and decoding for data transmission has been discussed in detail in the previous section.

For sending data to the servo motors, we use Pulse Width Modulated (PWM) signals. The duty cycle of the PWM signal varies giving different amounts of rotation angle. The servo motors we are using are of make and model MG946R and MG996R. According to the datasheet, the period is 20ms long. Additionally, a duty cycle of between 5% and 10% of the period is expected. The pulse duration therefore is between 1ms and 2ms. The servo rotates between 0 and 180°. The way our hand is designed, fingers close maximally for a servo angle of 90 degrees. Therefore a period of 1.5ms or 7.5% gives 90 degrees rotation. All of this underlying information has been compiled in a ready to use Arduino Servo library. For each servo motor, a separate object needs to be created. The object has member functions called 'attach' to identify the Arduino pin and 'write' which allows us to directly write an angle value to the motor.

----------x----------x----------x----------x----------x----------x----------x----------x----------

```
Servo RingFinger; //Create object
RingFinger.attach(pinNumber); // Assign motor a pin on Arduino
RingFinger.write(angleValue); // Write angle value
```

----------x----------x----------x----------x----------x----------x----------x----------x----------

Code snippet for controlling servo motor on Arduino

Each servo motor connection has 3 wires. Power, Signal and Ground. The servos require 5V at the supply pin and the signal pin needs to be connected to any PWM compatible pin on the Arduino (pins marked with a ~ -> {3, 5, 6, 9, 10, 11}).

The running current draw for each servo motor is 500mA for the maximum rated torque according to the datasheet. An Arduino cannot provide more than 40mA current output per digital pin. The obvious solution was to use a motor driver but before we did that we decided to experiment without it. The InMoov[7] model too does not use any motor driver because the assembly of the fingers and braided fishing line does not require a lot of torque to move. We tested rapid movements of open and close to make sure the robot responded appropriately to the commands and it did. However, a single Arduino powered over USB by the laptop could not provide sufficient power for 3 motors to move all at once. We connected external AC-DC adapters. Our adapters can provide an output of 5V, 4A. For 6 motors we would have a usage of roughly 3-3.5A.

Stage 4

The servo motors are assembled on the Robotic Hand in a servo bed. A rotational pulley is fixed on the shaft of the motor. A braided fishing line runs from the fingertips all through the palm and ends at the rotational pulley for each finger. The pulley mechanism and strings tug on each finger to open or close based on the angle written to the motor.

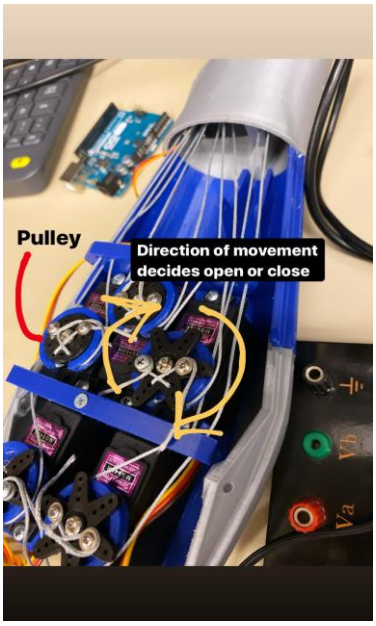


Fig 12: Arrangement and explanation of servos and pulley mechanism on servo bed

For rotation of the wrist, a gear mesh mechanism is used as shown in the image below.

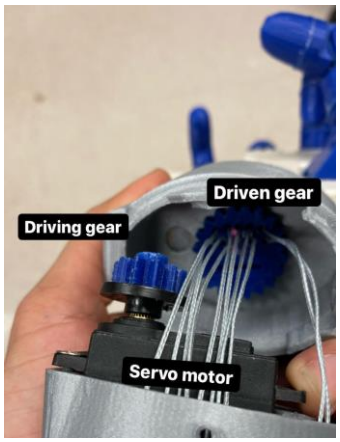


Fig 13: Closeup image showing wrist rotation mechanism

Constraints

Our final goal was to have a fast, responsive hand. Because this was a heavily intertwined electro-mechanical system, there were a few constraints that we had to introduce to best present our design. Adding the glove with the painted fingertips was one of the biggest and earliest constraints we added. We added this because of the drawbacks of YUV filtering a bare hand as discussed earlier.

Another major constrain that we had to impose was to physically disallow occlusion. While the Kalman Filter was successfully implemented, it introduced an unnatural amount of system lag which was detrimental to the overall demonstration. The response time of the hand was between 1.5 to 2s with the Kalman Filter along with the uncertainty as discussed before, so we thought it was best that we impose a movement constraint to not allow occlusion.

Another noteworthy item we added was a white ring of cloth to the bottom of the glove. We had to make sure our algorithm identified the correct region of interest. The contours detected by the camera would get merged from the glove right on to the user's piece of clothing. To avoid this, we added a small white ring to the base of the glove. The white would act as a thresholding barrier and separate out the glove from the other contour. From there, it was just about picking the northern most contour which would always be our hand region.

The major physical constraint that we had was with the rotation of the wrist. The movement of the fingers is string and pully based and highly dependent on the tension in the system. That's what keeps the fingers accurately in their positions. The wrist, however, moves using a gear meshing mechanism (Fig 13). Moving the wrist involved actuating the joint through which the fishing lines connected to the palm and subsequently the fingers. This tugged on the fingers unnaturally and resulted in loss in tension of the strings which resulted in fingers losing their positions. A fix for this was to add a tensioner part which contained springs that would compress and stretch when the wrist rotated. We fit the tensioner inside with springs but did not have enough time to refine it. The springs with the right stiffness were not with us at that moment. The two milestones were separately achieved but could not be integrated. We would have the wrist move but the strings for the fingers would lose tension after two to three minutes of operation and we would have to re-tighten them.

The last physical constraint was to have the user's hand in the "stage" to help with lighting so the HSV thresholding was more effective.

Parts List

In its entirety, our project and its respective presentation is comprised of three main sections: 1) the robotic arm, 2) the processing setup, and 3) the stage-glove setup (the controlled environment for performing our computer vision.

1. **Robot Arm**
   - 1.1. inMoov Right Hand (3D printed): http://inmoov.fr/inmoov-stl-parts-viewer/?bodyparts=Right-Hand
   - 1.2. inMoov Wrist (3D printed): http://inmoov.fr/inmoov-stl-parts-viewer/?bodyparts=Rotation-Wrist
   - 1.3. inMoov Forearm, Servo Bed, and Tensioner: http://inmoov.fr/inmoov-stl-parts-viewer/?bodyparts=Forearm-and-Servo-Bed
   - 1.4. 5x MG946R Servo Motors: http://www.towerpro.com.tw/product/mg946r/
   - 1.5. 1x MG996R Servo Motor: http://www.towerpro.com.tw/product/mg996r/
   - 1.6. .8mm Braided Fishing Line: https://www.amazon.com/Dorisea-Extreme-Braided-109Yards-2187Yards-109Yards/dp/B076J8RRN7/ref=sr_1_25?crid=35W4HU6FZYWRF&dchild=1&keywords=200%2Blb%2Bfishing%2Bline&qid=1576367178&s=sporting-goods&sprefix=200%2Blb%2Bfis%2Csporting%2C168&sr=1-25&th=1&psc=1
2. **Processing**
   - 2.1. 2 x Arduino Uno
   - 2.2. 1 x Raspberry Pi
   - 2.3. 1 x Pi Camera
   - 2.4. 2 x 5V Switched Mode Power Supplies (to power the Arduinos)
   - 2.5. Wires
3. **Environment**
   - 3.1. Black and White Knit Gloves
   - 3.2. White Cloth Paint and Assorted Acrylic Paint
   - 3.3. Posterboard
   - 3.4. 120V Aquarium Lamp

Milestones

Original goals

- Track bare human hand in real time using anatomical keypoints (knuckles, fingertips)
- Use two cameras to always get continuous, unobstructed profile of hand
- Neural Networks based system
- Robotic hand movement including finger movement, wrist rotation and palm pitching.
- Make system robust to occlusion, environmental changes
- Overall negligible system lag in subject movement and robot's reproduction
- Closed-loop control for finger movements

Our original proposal described an almost ideal system putting too much faith in the hardware performance and software capabilities. We intended to track a bare human hand without a defined region of interest area. Two cameras would make sure that all parts of the hand were always in some frame. Closed-loop control for the robot hand movement would ensure accurate positioning of fingers every time. A trained neural network would ensure a high level of performance of the system.

As the complexities in implementation manifested, we had to alter the original goals. The idea of using a neural network was the first to go. The neural network would be a black box which would have minimal contribution to the learning aspect as well as present implementation difficulties. Another key change was switching from closed-loop control to open-loop control. The robotic hand we are using is part of an open-source robotic design of the complete human body from InMoov. The model makes no provisions for attaching sensors to measure current positions of fingers. This led us to make the decision of switching to open-loop control instead first and to analyse the results before trying to incorporate the sensor ourselves. Another major change was bringing in the glove. With the neural network idea out we had to find another way to track all the fingers so we switched to color based finger identification. Palm pitching too had to be forgone because our robotic assembly did not provision it. We scoped down to finger movement and wrist rotation.

Modified goals after Milestone 1

- Use a glove with colored fingertips as key points.
- Kalman Filtering for key point tracking
- Single camera system, Kalman Filter takes care of occlusion
- Separate "stage" for subject's hand to narrow down the region of interest.
- Open-loop control system

Milestone 2 and Design Expo deliverable

- Glove with colored fingertips.
- Keypoint tracking without Kalman Filter
- Single camera system, occlusion now a physical constraint (tradeoff, system lag)

The goals following Milestone 1 and Milestone 2 were focused on realizing a practically achievable system. We proposed and successfully implemented a Kalman Filter for keypoint tracking. Our Kalman objects were each of the colored fingertips. While the Kalman filter did a good job of tracking our fingertips and being resilient to occlusion, it posed some problems as discussed earlier.

Another major setback was with the HSV thresholding for color detection. The robustness of HSV in changing light conditions was really put to the test when we did not use a controlled stage. To solve this issue, we built a completely white, mini enclosure fitted with a light source at the top. This gave us a deterministic setting and prevented mismatches in color detection. This was an adjustment in our original goal of tracking the hand in a natural environment.

Project Demonstration

For the Design Exposition we were set up in the following manner. We had one table on which we had the following items

- Foam board “stage”
- Wooden stand for robotic hand
- Display screen showing users the processing stages and outputs

The users were given a chance to wear our glove and watch the robotic hand mimic their actions.

Fig 14: Our setup at Design Exposition 2019 (EECS Building, University of Michigan)

EECS ELECTRICAL ENGINEERING AND COMPUTER SCIENCE UNIVERSITY OF MICHIGAN

EECS 452: DSP Design Lab
Professor Wakefield
Fall 2019

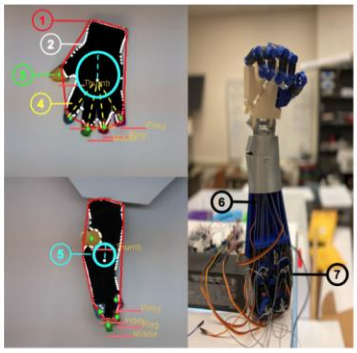
Hand It Over: Vision-Based Hand Control

Christopher Herran, Saruf Alam, Deepen Solanki, Liam McNeil

Abstract

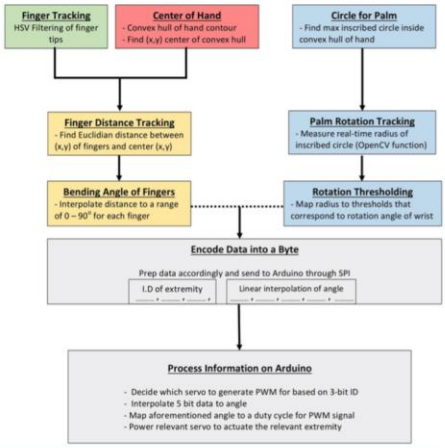
Hand it over allows users to control a robotic hand in real time through using computer vision concepts. We explore digital signal processing while tackling robotics and prosthetics. Our project methodology has potential for widespread applications such as bomb defusal, surgeries and extreme weather operations. We track facets of a hand and then transform relevant data into command signals to be executed by arduinos--powering our own 3D-printed and assembled prosthetic hand.

System Overview



1) Convex hull of the whole glove used to find inscribed circle (blue)
2) Hand contour of the whole glove used to find center of hand
3) HSV filtering to find fingertips, each uniquely colored
4) The distance between each finger and center of hand, used to find how much the finger is bent
5) Max inscribed circle for convex hull and its radius, used to find rotation of hand
6) Open source 3-D Printed robotic hand
7) Servos connected to arduino that actuate the hand extremities

System Architecture



Challenges & Solutions

Challenges	Solutions
- When tracking a bare hand, we had to implement a Kalman filter to track different fingers. This was too slow (2- 3 fps).	- Instead of the Kalman filter, we added a glove with different colors for each fingertip and set different HSV thresholds per finger
- HSV filtering was very fragile and broke in different lighting conditions or if alike colors were present.	- We built a covered stage that had its own light source and a plain white background.
- The open-source hand has limited ranges for the fingers (0 - 90°) and wrist (0 - 30°)	- We request users to only place their hand palm up and have set hard thresholds for the PWM duty cycles
- SPI transfer was fragile with one arduino. If it was too fast, data became corrupted, but at the speed it worked, our tracking was jittery and lagging.	- Instead of one arduino, we used two, with each other taking 3 out of 6 controllable extremities, and we set SPI speed to its default state

Results

- Successful real-time detection of user's hand and extrapolation of relevant information at approximately 15-16 frames per second.
- Accurate actuation of robotic hand based on extrapolated information with a lag rate of ~ 0.105 s.

Algorithmic Step	Time (s)	Percentage
HSV thresholding	0.035	33.3%
Figuring out Position	0.030	28.6%
SPI Transfer	0.010	9.5%
PWM Generation	0.030	28.6%

Note:
- PWM Generation timing was inconsistent. Sometimes it would jump up to 150 ms.

Conclusion

We were able to successfully create a product that mimicked a user's hand gestures using computer vision systems. Our implementation required a lot of physical vs algorithmic considerations. We had to find physical work arounds to software limitations. For example, instead of using multi-processing to speed up data transfer, we used two arduinos to control all our servos. If we were to continue this project, our next step would be to remove such physical constraints and work on making a more free system. Notably we would retry implementing a hysteresis based tracking of the hand, allowing for glove-free tracking.

Acknowledgements

We thank Raytheon for project funding so Hand it over was possible.

Also would like to thank Prof. Gregory H. Wakefield, Dr. Kurt Metzger, and Ashish Nichanametla for their guidance on this project.

Fig 15: Poster at Design Exposition 2019

Contributions of each member of the team

Liam McNeil – 25%

- Assembly and testing of the Robotic Hand
- Design Expo Poster
- Relevant sections on the report
- Wooden box and stage setup construction for hand demonstration

Christopher Herran – 25%

- Assembly and testing of the Robotic Hand
- Milestone 1, Milestone 2 presentations
- Relevant sections on the report
- Wooden box and stage setup construction for hand demonstration

Saruf Alam – 25%

- YUV Exploration
- Raspberry Pi related processing
- Final presentation
- Relevant sections on the report

Deepen Solanki – 25%

- Kalman Filter exploration
- Raspberry Pi processing
- Arduino related implementations and explorations
- Relevant sections on the report

While the Raspberry Pi related work was split between Saruf and Deepen, everybody was responsible for pitching in ideas and identifying drawbacks. Ideas were programmed and fine tuned by Saruf and Deepen.

Notable ideas

Chris → Preventing hand contour merging using white band separation + northernmost contour
Saruf → Palm scaling, palm rotation based on inscribed circle
Liam → Area of rectangular strip for rotation
Deepen → Discretization of hand movements to smoothen response

References and citations

Wrist rotating video: https://youtu.be/q14GAOF5HYA

[1] **"***Skin segmentation using YUV and RGB color spaces"* - Zaher Hamid Al-Tairi**,** Rahmita Wirza**,** M Iqbal Saripan**,** Puteri Suhaiza Sulaiman
https://www.researchgate.net/publication/273105163_Skin_segmentation_using_YUV_and_RGB_color_spaces

[2] "A Hybrid Color Space for Skin Detection Using Genetic Algorithm Heuristic Search and Principal Component Analysis Technique" - Mahdi Maktabdar Oghaz , Mohd Aizaini Maarof , Anazida Zainal ,Mohd Foad Rohani , S. Hadi Yaghoubyan
https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0134828

[3] "*Human Skin Detection Using RGB, HSV and YCbCr Color Models*" - S. Kolkur , D. Kalbande , P. Shimpi , C. Bapa , and J. Jatakia - https://arxiv.org/pdf/1708.02694.pdf

[4] https://iitmcvg.github.io/summer_school/Session4/ [Online]

[5] *"Moving Target Tracking based on CamShift Approach and Kalman Filter"* - Shao-Fan Lien , Kuo-Hsien Hsia and Juhng-Perng Su - http://www.naturalspublishing.com/files/published/u956oro5157k5y.pdf

[6] https://www.myzhar.com/blog/tutorials/tutorial-opencv-ball-tracker-using-kalman-filter/ [Online]

[7] http://inmoov.fr/hand-and-forarm/ [Online]

Appendix

Section 1 – YUV colorspace and bare hand tracking

Our desire for bare hand tracking lead us to looking at frame conversion to the YUV space. YUV is a lesser known color space that tries to closely mimic human perception of the world. The Y component focuses on brightness and "tint" of the colorspace, meanwhile the U and V components focus on the red and blue components of the color space. While YUV is not common, there is a lot of research that compared it to other spaces and came upon the fact that it is very friendly to detecting human skin. For example, in the Journal of Information Processing Systems, Al-Tairi and all ran tests and came upon the conclusion that thresholding for human skin in the YUV space is "more robust in terms of dealing with the complex background and light conditions than others".[1] Other research papers asserted similar strengths (Oghaz et al., 2015[2] and Kolkur et al., 2017[3]). So, we decided to use YUV and came up with an algorithm to track each finger. The block diagram is shown below:

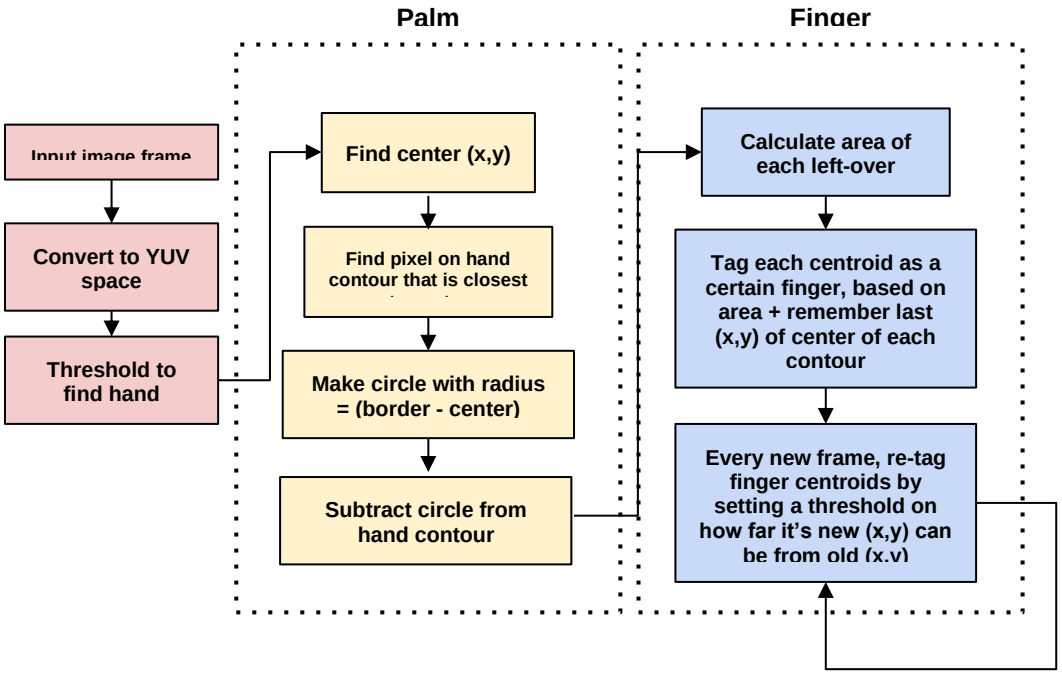


Fig 16: Block Diagram of Keypoint Tracking Using YUV colorspace

Our YUV approach involves three main stages. The first stage is to actually get the contour of the hand from an image frame. Since YUV isn't well known, we had to come up with our own algorithm to convert RGB to YUV. The equation we used came from Al-Tairi:

$$Y = \frac{R + 2G + B}{4}$$

$$U = R - G$$

$$V = B - G$$

After converting each pixel into YUV components, we then used a relevant threshold to make our hand contour.

The second stage, once we had our hand contour, was removing the palm from this contour, so that we can track the location and size of each finger. This was carried out by finding the center of the whole hand contour, and then finding the border pixel of this contour, that was closest to the center. This pixel would almost always be located at the edge of the palm. We would then make a circle with a radius that was the distance between the center and this bordering pixel. We would then subtract this circle from the overall hand contour, resulting in a contour of the five fingers.

The third stage then, was to categorize these fingers consistently and track them. Initially, we categorized each finger based on its area. We know the contour with the biggest area is realistically the index and so forth, so this approach was a good first approach to tag each finger and distinguish the contours from each other. However, once this initial categorization was done, we needed a way to follow each finger. The way we did this was, for frame 0, given a certain finger had its centroid at (x,y), the next new frame, we wouldn't let the category jump to another centroid that had a very different (x,y) centroid. This way, we made sure the categorizations didn't shift too much, allowing for a sense of consistency in our approach to distinguishing which contour is which finger.

YUV Results:

Our YUV approach was quite successful in filtering a hand. Just as the research said, it was quite accurate and robust as a colorspace when it came to distinguishing human skin. It was able to work reliably in different lighting conditions and with different backgrounds.

Fig 17 : YUV tracking the fingers quite easily even though the background is quite similar in hue to human skin

However, our approach ultimately failed due to the numerous other aspects of this approach. Firstly, the conversion was much too slow. The conversion code itself slowed down our frame rate to 5 - 6 fps. On top of that, all the post-processing such as the palm subtraction, tracking and categorizing further slowed our code down. Lastly, implementing a hysteresis based tracking algorithm was very difficult. Our approach to categorizing fingers worked with poor accuracy and was also quite fragile. As can be seen in the image below, it would wrongly categorize fingers easily and would completely break down if there was any occlusion.

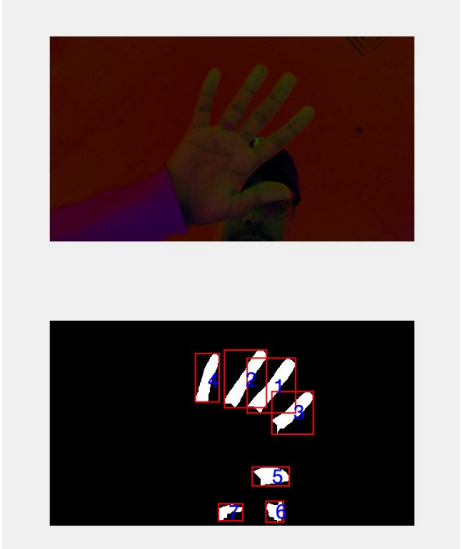


Fig 18: Categorization code was imprecise. Apart from the thumb (5) and pinky (4), it mis-categorized everything else.

Section 2 – Kalman Filter

Kalman Filter[4-6] is an optimal state estimation algorithm. It allows measurement of a quantity indirectly when direct measurement isn't possible. For object tracking, two methods are available.

- Position Only Measurement System
- Position and Velocity Measurement System.

The key components of a Kalman Filter are its *state vector, initial state covariance, measurement state covariance, transition state covariance* and *observation covariance.* For a Position Only Measurement System, the Kalman Filter state vector would consist only of the space coordinates of the target object. For a Position and Velocity Measurement System, the state vector would include velocities too (which we used)

The Kalman filter works in two steps, *Update* and *Correct*. The Update step allows to predict the position of the object knowing its history, the speed of its movements and knowing the equations that identify its movements. The equations identifying the movements are contained in a *transition matrix*. Every time a measure of the state of the object is available, the prediction is refined. In our case, every time the fingertip is in the frame and detected, the Kalman Filter prediction would be updated. The correction step would then change the filters internal variables. Mathematically,

$$System\ state:\ s(t) = \Phi(t-1)s(t-1) + w(t)$$

$$Measurement:\ C(t) = H(t)s(t) + v(t).$$

Where Φ represents the state transition matrix, H is the measurement matrix, w(t) is the process noise and v(t) is the observation noise.

Update step :

$$\hat{s}(t) = \Phi(t-1)\tilde{s}(t-1)$$

$$\hat{P}(t) = \Phi(t-1)\tilde{P}(t-1)\Phi^{T}(t-1) + Q(t-1)$$

Correction step :

$$K(t) = \hat{P}(t)H^{T}(t)[H(t)\hat{P}(t)H^{T}(t) + R(t)]^{-1}$$

$$\tilde{s}(t) = \hat{s}(t) + K(t)[C(t) - H(t)\hat{s}(t)]$$

$$\tilde{P}(t) = [I - K(t)H(t)]\hat{P}(t)$$

where sˆ and s˜ are prior estimate and posterior estimate respectively. Pˆ and P˜ are prior estimate error covariance and posterior estimate error covariance respectively. K is the Kalman gain and R and Q are the measurement error covariance and process noise covariance.

Our Kalman Filter implementation used the OpenCV Kalman module. We had a two-dimensional filter initialized for Position and Velocity based measurement. The Kalman objects were the thresholded and detected fingertips.

Section 3 – Rotation using inscribed circle method

Our first method involved using the radius of the palm yet again, to track the rotation of the hand. If we placed the camera at a certain height, we would know the smallest possible value the radius of the inscribed circle could take. This would be when the hand is laying at rest on the surface the camera stand is placed on. Now, if we rotated this hand, this radius would get even smaller and we could track that change to find the angle of rotation. This is can be visually verified through the diagram below:

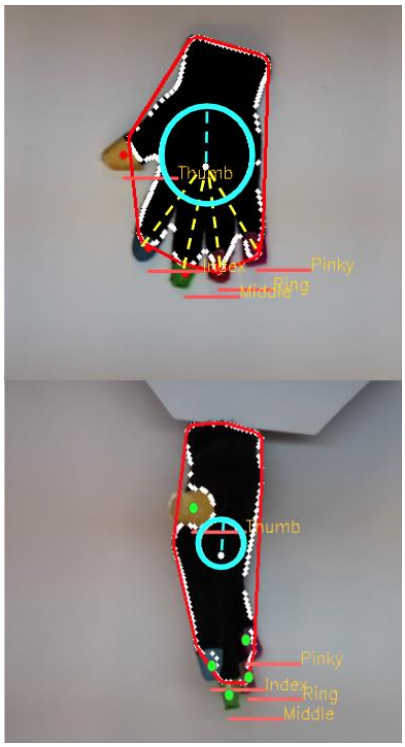


Fig 19: When the hand is laying on the surface the camera stand is placed on, rotating the hand will shrink the radius of the inscribed circle

This algorithm was quickly scrapped due to numerous issues. The first issue was the fact that the radius didn't change as much as we wanted it to. Empirical data showed that, the radius for the rest position was 40, however only shrank to 35 when the hand was rotated a full 90 degrees. For every 10 degrees, we only saw a change in radius of around 0.55. However, the resolution itself was quite poor and without moving the hand we already saw a deviation in data of ± 0.3. That means, the ground truth could be the ± 0.3 units for any radius value we were getting. This made the data quite useless in this scenario because we almost never knew what angle of rotation we were at.

Secondly, the algorithm would break down if the person lifted their hands above the surface. There was simply no work-around this issue because we had already used this piece of data in order to calculate distance scaling, and there was no way to overlap it to also carry out rotation. In addition to it breaking if the hand was lifted, it would also break if any of the fingers were bent. Bending any finger made the contour smaller and shrunk the radius. This shrink simply added too much noise. There was already a need for very precise measurement, but this phenomenon was masked the real data even more.

Note that these were not issues faced by the distance scaling algorithm primarily because the radius changed a lot if the hand was closer/farther from the camera, so we did not have to be as precise with our mapping.

Given the numerical and systematic issues with this approach, we decided to change our algorithm. Since we already had a glove, we realized that adding another facet to it doesn't change our constraint, and, if it adds important functionality, it is a worthy option.

Section 4 – Near-Analog Control of Robotic Hand

Our first idea was to prioritize precision over speed when sending data and mapping the data to digital signals that controlled the hardware. The diagram below covers this approach:

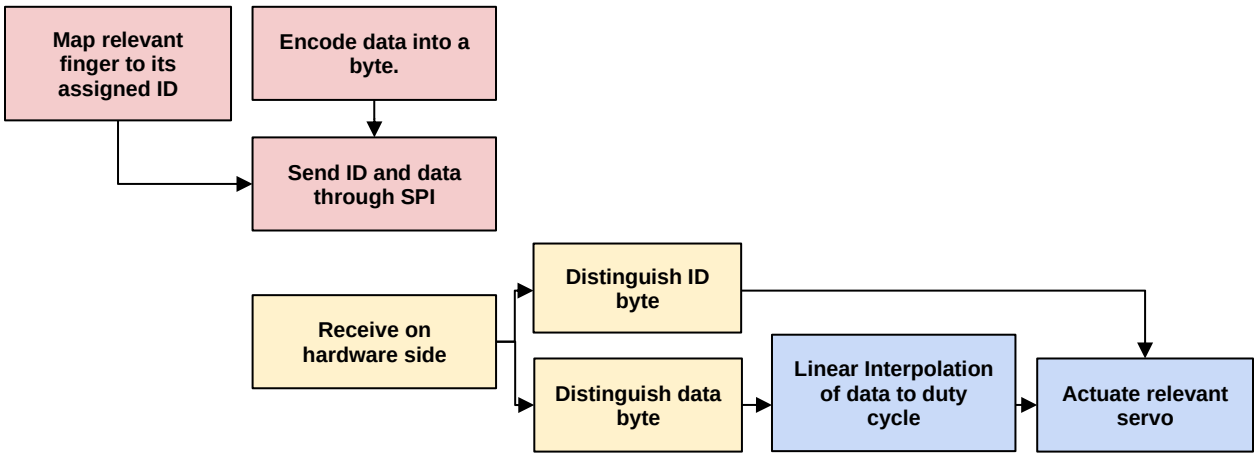


Fig 20: Block Diagram for Near-Analog Style Algorithm

We first took the data we had for an extremity and scaled it so it could fit within a byte. Then, through SPI, we sent 2 bytes of information, the first being the ID of the extremity, and the second being the data itself. The ID was a number between 0-5 (five fingers and a wrist). On the hardware side, we then distinguished which one is which using global variables and keeping track of previous values. We checked if the SPI buffer had a value less than equal to 5 and equated that to a global variable called ID. If the SPI buffer had a value greater than 5 then we knew it corresponded to the rotation angle for that ID. Since the ID was being sent first and then the rotation angle, the global variable would always contain the ID of upcoming finger data. While actuation, we would check the current value of the global carriable ID and then run the appropriate servo.

This control scheme was problematic. Firstly, for unknown reasons the SPI would mix up the ID and data it was sending. For instance, instead of sending the thumb ID followed by the thumb data, it would send the thumb ID, the index ID and then the thumb data and then the index data. Or, it would send the index ID with the thumb data. This jumbling inevitably caused inaccurate mapping of our hand poses.

Secondly, the SPI mechanism was too noisy. Other than jumbling up data, it would, at times send pure noise. This was fixed by slowing down our SPI command from 1,000,000 to 500,000 clock cycles. However, this made the code too slow. Our overall refresh rate for any hand pose, was around 5 - 6 frames per second, with SPI being the bottleneck.

Thirdly, the data was simply too precise for the hardware. There was a constant stream of data incoming that forced minute variations too to change the servo PWM signal. While a servo adjusted to one PWM signal, another one with another duty cycle would come in. This didn't allow the servo enough time to move itself to the desired position, before it is asked to move to another position. So, our outcome using this scheme was a very jittery and unresponsive hand.