Image Recognition based Master-Slave Manipulator Mechanism

Rhitvik Kumawat – rk3494

Kalpan Mehta – ksm469

Ankit Kumar – ak7311

Link to Video: https://youtu.be/p_vQnpDnV3w

Table of Contents

Description

The project demoed by us is a Master-Slave Manipulator Mechanism that reads the gestures in real-time with the code we will be implemented. The user / Master points his/her hand towards the camera which in turn will process, what we want the machine to do. This data post-processing controls the actuator assembly, a robot prosthetic limb with 2 degrees of freedom. The data generated by the real-time gesture recognition algorithm we implemented controlled the motor-driver controller that included its direction to turn the hand right and left. Also, the hand supported a claw that opened and closed to grip any object that fit its size. The functionalities can include a variable gain in the motion control that can amplify the effects of the gestures perceived by the system. The mechanical actuation was realized using an over clocked Raspberry Pi 4 and the custom designed circuit that supported fast motor switching capabilities by employing fast switching transient suppression snubber circuits. Two independent power systems were used and used that delivered more than 50 Watts of power combined. The end system realized was able to actuate a mechanical arm that mirrored the gestures made by the user in front of the IR camera.

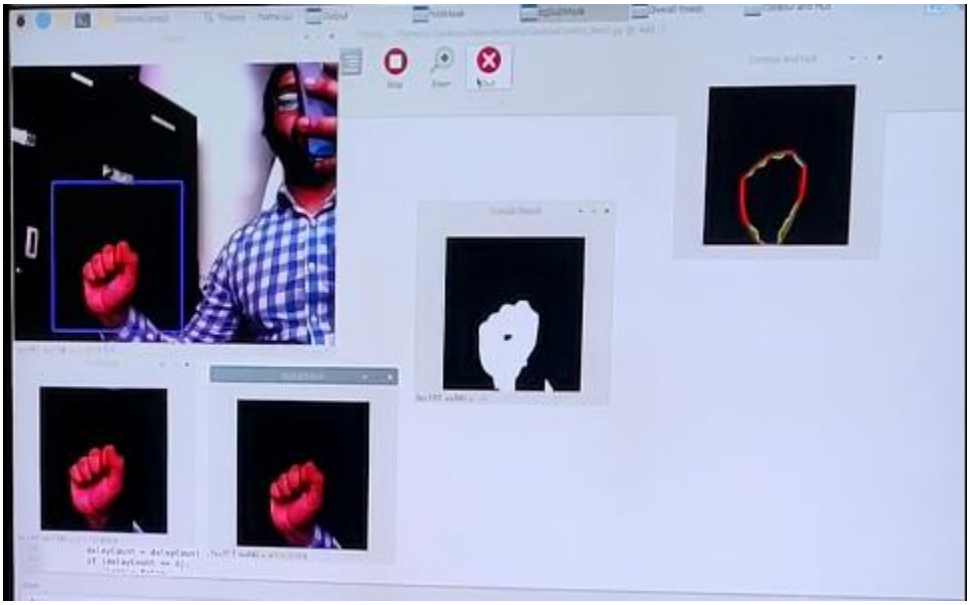

Fig 1: Windows showing different masking operations

Now let's understand the components used and the systems implemented in detail

Decoding the Idea behind the Code

[1]The Code was developed in python that utilizes open CV2 based libraries that run on Linux based Operating System. The gestures shown in front of the IR camera read the data

The approach we used is close to HSV (Hue, Saturation, Value) segmentation. the idea is to segment the hand based on the color. At first, we will sample the color of the hand. Then, we detect. Usually, a pixel in a frame or an image is represented as RGB (Red, Green, Blue). The reason we use HSV rather than RGB because RGB contains the information on the brightness of the color. Therefore, when we sample the color of the hand, we sample the brightness as well. This is an issue when we detect the hand because the hand must be under the same brightness in order to be detected. The brightness of a color is encoded in the Value (V) in the HSV. Hence, when we sample the color of the hand, we sample only the Hue (H) and Saturation (S) and leave out (V). [1]

Based on this technique we first place our hand near the green square to sample our hand color. This forms a histogram of the frequency of each color appearing in the sample. By normalizing this, we can predict the certainty of each color being a part of the hand.

Now, after we come up with a mask, for detecting the fingers there are two ways to do it: 1. Finding convexity defects 2. CNN. For CNN, due to GPU constraints we were finding it hard to classify a large data set and use it real-time for our purpose. Nevertheless, we will be including our CNN results as well in the report.

Coming to the method we applied here, i.e., Finding Convexity defects. We start by finding the contour of the largest which is assumed to be hand in this case. After finding the largest contour, we will find its convex hull. The convex hull is simply a curve covering the contour. From the convex hull, we can find the convexity defects. Convexity defects are the place where the curve is bulged inside. These are assumed to be the spaces between the fingers. We will use this to determine the number of fingers.

Signal Processing System

Connections to the camera were established and enabled using the Raspberry pi configuration window. Infra-red camera was preferred over the standard camera because performance in foreground and background separation further improved with the added benefit of being able to use it in a comparatively dimmer and multichromatic environment i.e., now we won't need a single colored background for achieving desirable results post image processing.

This system comprises of the Raspberry Pi 4 and the IR camera module that directly gets mounted on to the raspberry Pi's camera connector. The Operating system installed runs the python script (find code with this report as a separate ".py" file) that will transcode the gestures into commands for the MSM.

Create Hand histogram function creates the histogram which stores the Frequency of color occurrences. Then, the function histogram masking is called what subtracts the background from the image using the values obtained from hand histogram.

The data then is stored as an integer that we feed to our system as an input for the state machine that in turns generates the signals to send to the motor controller motor controller and actuation functions as an input. With these inputs certain control signals are generated that goes to the motor controller which in turns regulates the amount of power and duration for which the power must be sent to each motor.

The way this works is that the GPIO pins generates individually are configured in the output mode that gives these pins a considerably strengthened driving capacity. Now, as per the state, the pins are set to the desired logic levels. These signals are sent to the motor driver IC that turns on/off and determines the direction of rotation of the motors.

The optimize the control functionality of the GPIO peripherals, we interfaced NAND GATE as an inverted pair logic supply. This reduced the number of pins required to control the motor. Another consideration taken by us was opting for the current limiting resistor that protects the pins of Raspberry Pi from short circuits in case there is any.

Actuation Control System

The motor driver employed was L298n, a double half bridge motor controller that can source up to 3 Amps at 48 volts. The operating range of our project was 12V at 3Amps. The driver was also attached to an active heat sink that operated at 5 volts and kept the motor driver cool while the system was on. The big challenge with using the L298n motor driver IC is that there is no freewheeling protection circuit. (a freewheeling circuit directs the spikes and switching surges to either the ground or the input power, which ensures that the transistors inside the drivers are not damaged). So, to protect the driver from the effects, we implemented our own snubber circuit, employing 2 reverse biased Schottky Diodes to that will act as our freewheeler circuit. Also led were added for an added visual feedback mechanism that ensured if the motors were fed power or not. The following is the schematic of the motor driver configuration we used.

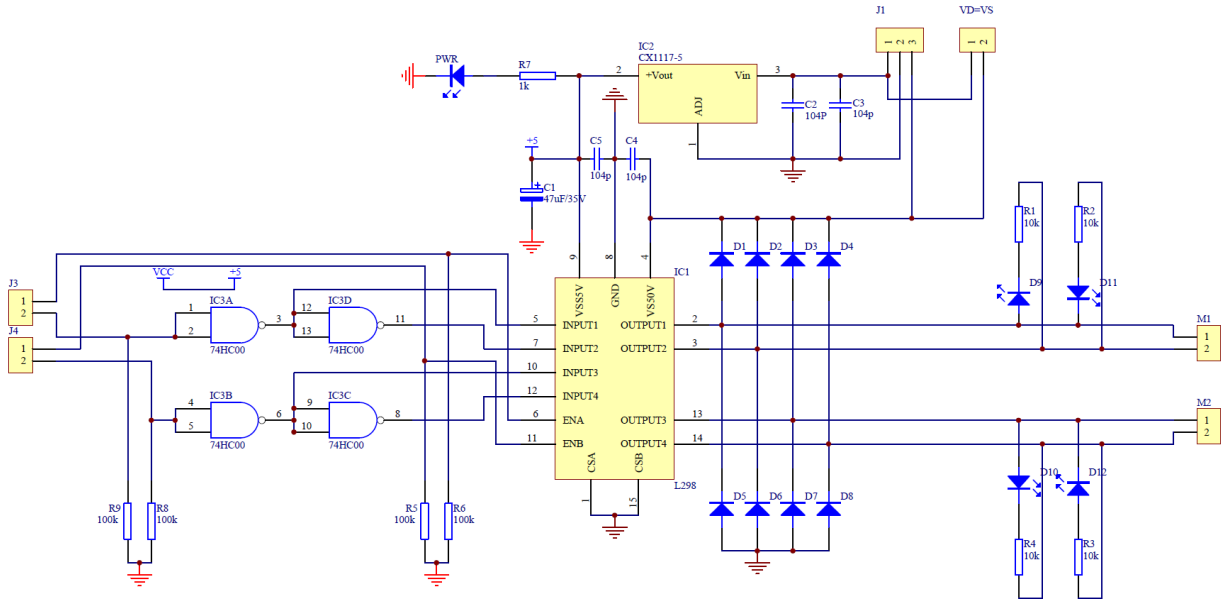

Fig 2: Schematic implemented for the input logic minimization and switching voltage protection circuit.

This driver is also capable of driving a single stepper motor. Please note that the NAND GATE Pair won't be compatible with a stepper motor.

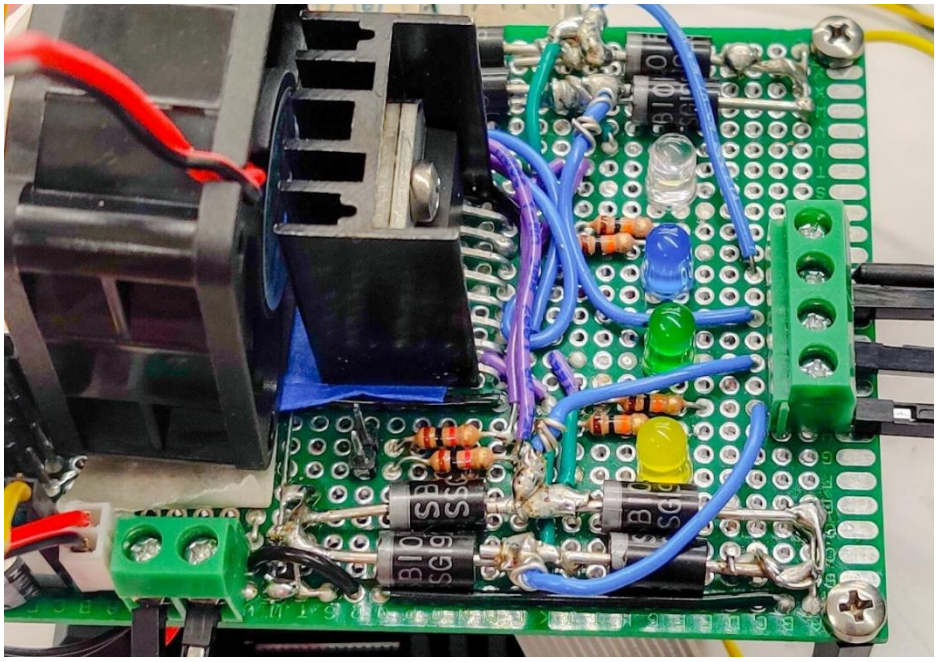

Fig 3. The motor driver assembly of the MSM (We can observe the 4 pairs of 2 diodes forming the freewheeling circuit).

Power system

The power supply employed in the system is a pair of SMPS (switched mode power supply) 5V at 4 amp for powering and running the raspberry pi, motor driver logic supply, cooling fan of raspberry pi and motor driver IC.

The 12 V power supply is connected to the motor driver that feeds the power to the motors. It also forms the part of freewheeling snubber circuit.

The main reason that we don't use the same power supply in the system is because the initial power drop of voltage to start the actuation assembly might trigger the Brown-out reset in the circuit. This will result in forced restarting of the processing unit (raspberry pi).

Actuator assembly

Actuator assembly compromises of a modified prosthetic limb that is fitted with two motors that functions independently for the purposes of automating the actuation. (the base design did not have any motors that can be interfaced with any automation system). The gripper mechanism has a 10 RPM motor and the wrist mechanism comprises of a 30 RPM. Both motors are connected to the L298n IC output Port 1 and 2 respectively and are controlled by the motor driver that in turn is working as per the signals given by the GPIO pins of the Raspberry Pi.

Fig 5: Actuator Assembly of the MSM.

Final Images

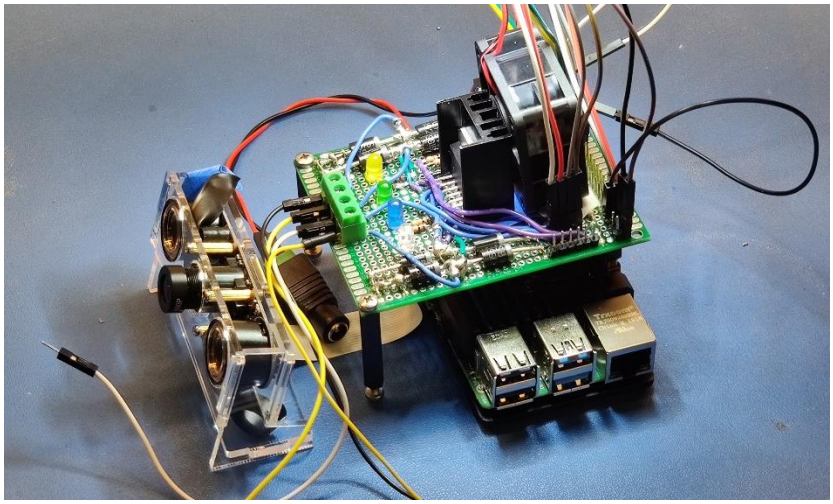

Fig 6: Motor Controller Shield mounted on a Raspberry Pi with an external Infra-Red Camera.

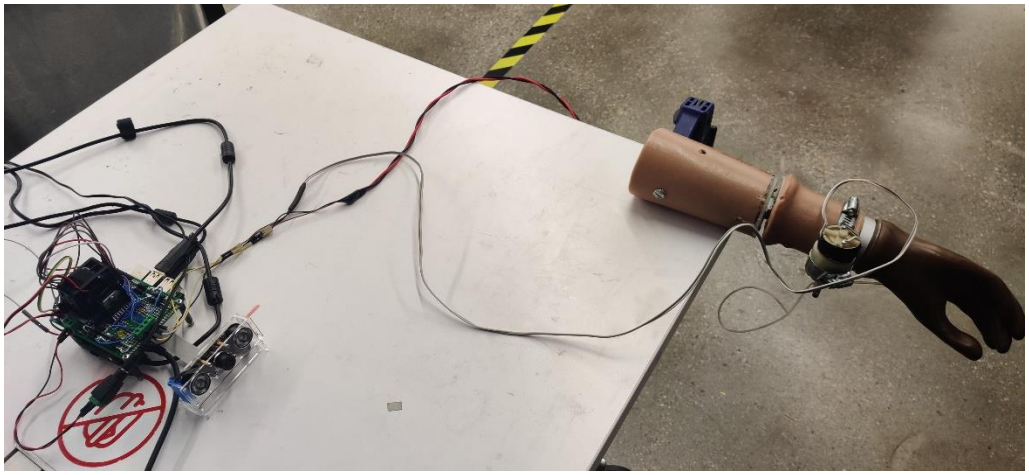

Fig 7: Final Assembled Prototype.

References

1. https://realpython.com/python-opencv-color-spaces/