Gesture Controlled Robot

By
Arvind Vijaykumar
Qinlun Luan
Bofan Yang

Final Report for ECE445, Senior Design, Spring 2019
TA: David Hanley

2nd May 2019
Project Number: 30

Abstract:

This project essentially serves as a proof of concept for the implementation of a hand gesture controlled robotic vehicle. Hand gesture recognition is achieved using modern machine learning algorithms and techniques in Python using a CNN classifier. Data transmission from the software application to the vehicle is facilitated via Bluetooth and the vehicle utilizes Arduino and a dual H-bridge motor driver for motion control. Seven gestures translate to seven different directions of motion including moving forward, left, right, backwards, reverse left, reverse right, and stopping. Our main goals as engineers in the context of this project was to explore a modular design based methodology, specifically one involving software-hardware interactions, as well as develop a novel robotics control mechanism such that the field can be more accessible towards older people and people with disabilities.

Contents:

1. Introduction:

1.1 Background and Purpose

Wireless communication systems form the backbone of modern-day human-robot interaction. Namely, wireless remote control methodologies including IR, RF, and network-based technologies such as WiFi and Bluetooth facilitate communication between a client and a robot such that the robot can successfully actualize its desired functionality. The main advantage they possess over wired control is that they provide a much broader range for the robot to interact with its environment. External peripherals (game controllers, smartphones, etc.) are usually required in order to wirelessly transmit data to the robot, however, these wireless physical peripherals still withhold constraints such as the difficulty of using for people with physical disabilities and easy to wear issue for delicate electronic parts. In our project we want to free the user of these physical peripherals by introducing a touchless UI technology based on artificial intelligence, in particular, the use of machine learning methodologies like CNN (Convoluted Neural Networks) for gesture recognition combined with wireless technology such as Bluetooth to form a gesture-based control scheme, could form the backbone of a potential solution to these problems, as our solution only requires one hand to operate and requires no hardware control device, this means people with only one arm can also operate the robot control by our gesture control system, and users no longer need to worry about control device running out of battery or getting less sensitive and accurate due to extensive using. Moreover, our primary objective in this course is to combine these technologies to explore a new and innovative method of robotic control.
of motion for the robot as well as a gesture to cease movement. We intend to utilize a convolutional neural network (CNN) based architecture for our hand gesture recognition system, the details of which will be elucidated upon in the following sections.

1.2 Block Diagram

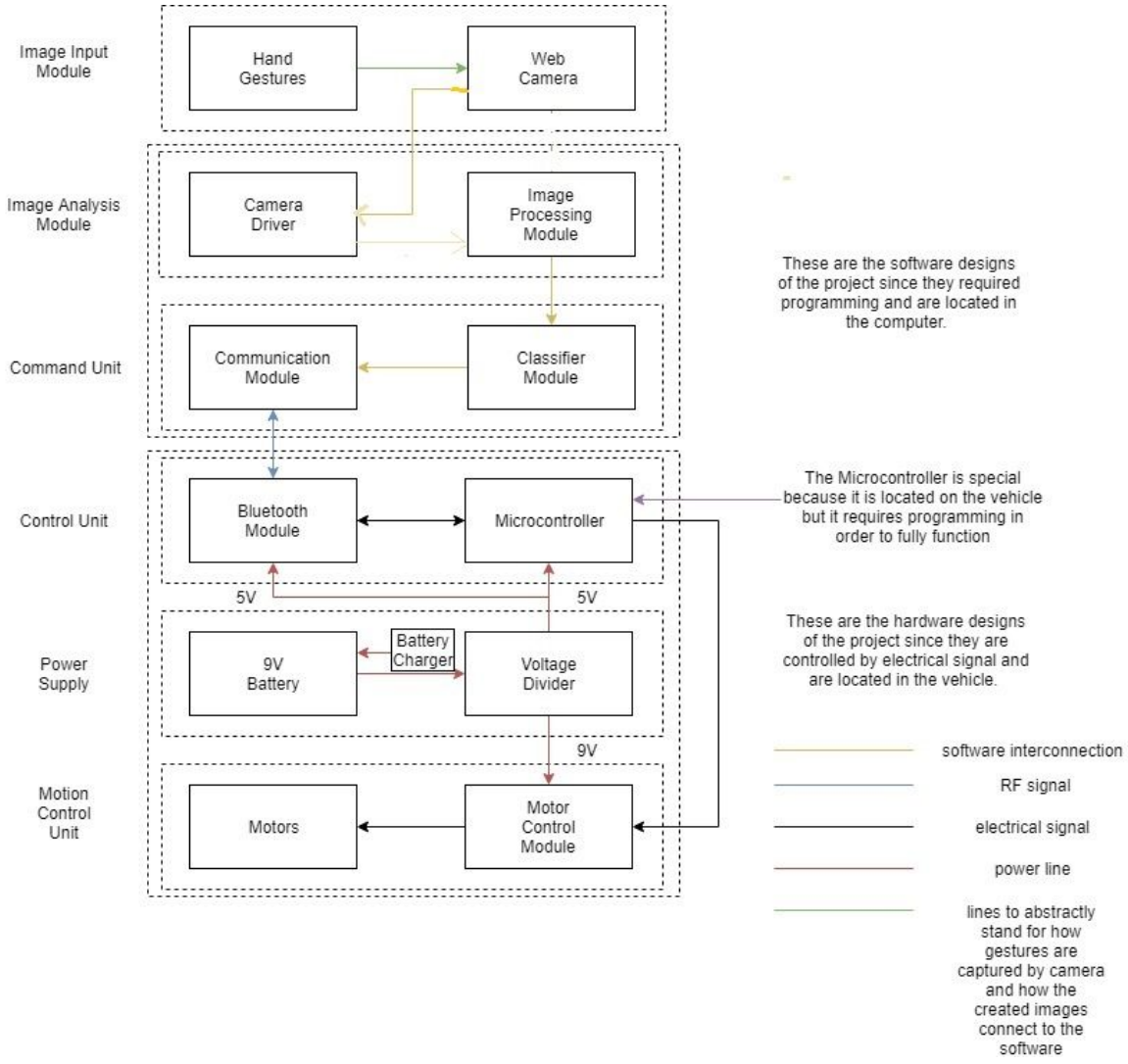

Figure 1: Gesture Controlled Robot Block Diagram

1.3 Gesture Design

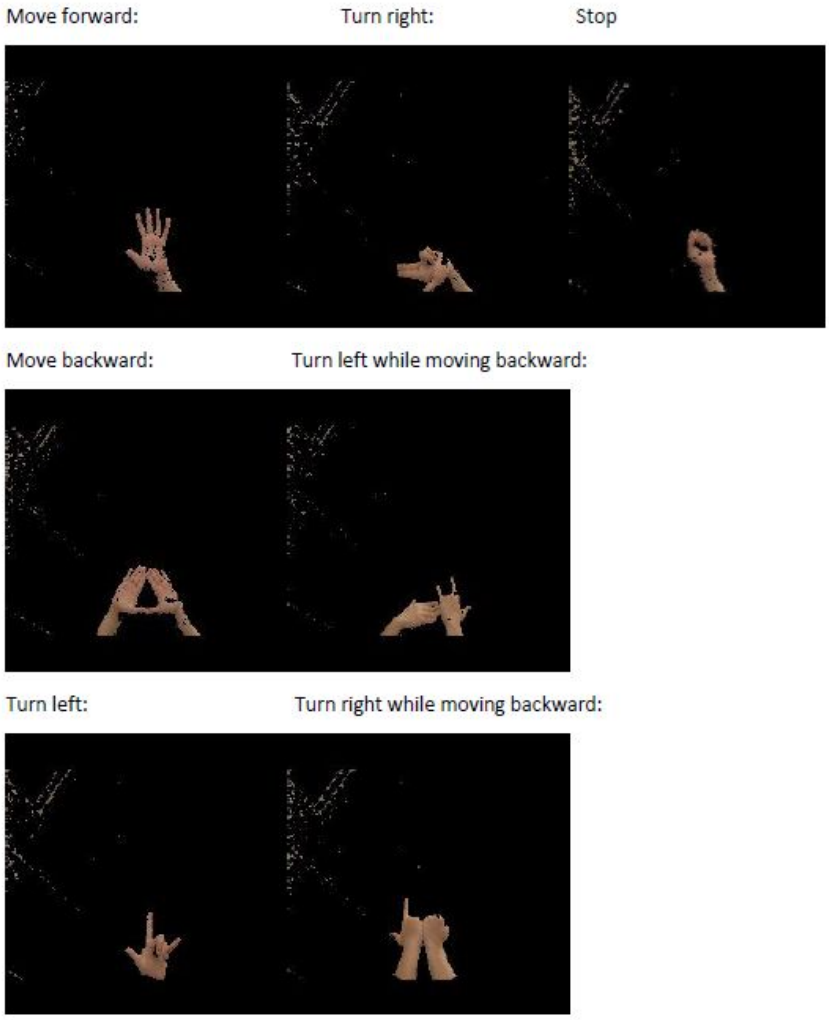

Figure 2: Gesture to Corresponding Direction

2. Design:

2.1 Design Procedures

Hand Gestures:

At first, we only considered using one hand to demonstrate gestures, because gestures made by one hand were simpler. However, later we switched to using

two hands because the 7 gestures need a minimum level of distinction from one another. We cannot design so many notably distinct gestures with just one hand. Another important reason is that the gestures work well with the CNN classifier because the two-handed gestures are larger in size and size is an important variable in classifying different images.Figure 3 and 4 in the Appendix demonstrate the ideal and real time cases.

Web Camera and Camera Driver:

We decided to include the above two components together because they represent the hardware camera and the software to adjust the images taken by it. We use the built-in camera of Qinlun's personal computer and used the OpenCV Python 3 library to manipulate the camera. The advantage is that implementation is simpler since we already have a camera and we can directly use functions from the OpenCV library to achieve our goal. One issue is that the camera is low quality and it does not have great performance under all light conditions. We will discuss the importance of light conditions in the image preprocessing module and the conclusion was that the low-quality camera is heavily influenced by the light conditions. For example, under strong light conditions, the camera can only recognize a small part of skin while it fails to recognize anything under weaker light. The following two images show the ideal cases and real-time cases.

Image Preprocessing Module:

The functions of this module are to process the real-time images such that only aspects of an image with a similar complexion to the user's skin can be picked out and the background is removed. At first, we planned to further process the images to grayscale images because we can reduce noise and eliminate the influence of the light conditions to some extent. However, the CNN classifier is better with three channels because it would have more information to predict the class of an image. Therefore, we decided to use color images instead. This option actually further makes our algorithm vulnerable to extreme light conditions.

Classifier Module:

We have two choices in implementing this classifier. The first option was to use a 2D image filter as the convexity detection to find out the convexities and defects and then predict the gestures based on these inputs. This option is easy to implement but it might not be able to classify all the 7 gestures. The second option is to build a CNN as a classifier with Kera and Tensorflow libraries. This choice is more accurate and could be used to other applications if we want to further work on the project.

Power Supply:

At the beginning of our design, we planned to use an Arduino board to power our Bluetooth chip, and a 9V external battery to power our PCB. However, due to our failure to design a PCB for our project, we switched to using a separate 9V battery to power the L298N motor controller, and another 9V source to power the Arduino board. The Arduino Mini required the use a power bank to power the Arduino board in the final demo due to time constraints.

Communication Module:

Pyserial and Pybluez are the two libraries we used for communicating with the HC-06 chip from the user's laptop. We had to use two libraries because we were testing the HC-06 chip on both Mac and Windows platform, and due to the differences between both systems, we were not able to get Pybluez to work on Windows correctly, resulting in us opting for Pyserial instead.

Pybluez is a library design specifically for Bluetooth programming on Python, however, the library has not received any major update since 2015, so there are a lot of compatibility issues with installation. As shown in Figure 5, Pybluez connects with a Bluetooth chip through an RFCOMM channel. The RFCOMM channel, regardless of the availability of the Bluetooth device, stays in an unconnected state if the program does not attempt to send any data through it. If the program intends to send data through the RFCOMM channel to Bluetooth chip, the MAC address of the chip is required to locate the chip and establish a connection, the port number is chosen by the programmer to listen on.

Another thing worth mentioning about pybluez is that it did not work with Windows. We weren't able to find solid evidence on the reason why, but the inbuilt function provides clear evidence that the problem is the port number windows has provided is invalid.

This could be caused by the Windows 10 security update, forbidding an unauthorized application to use an RFCOMM port.

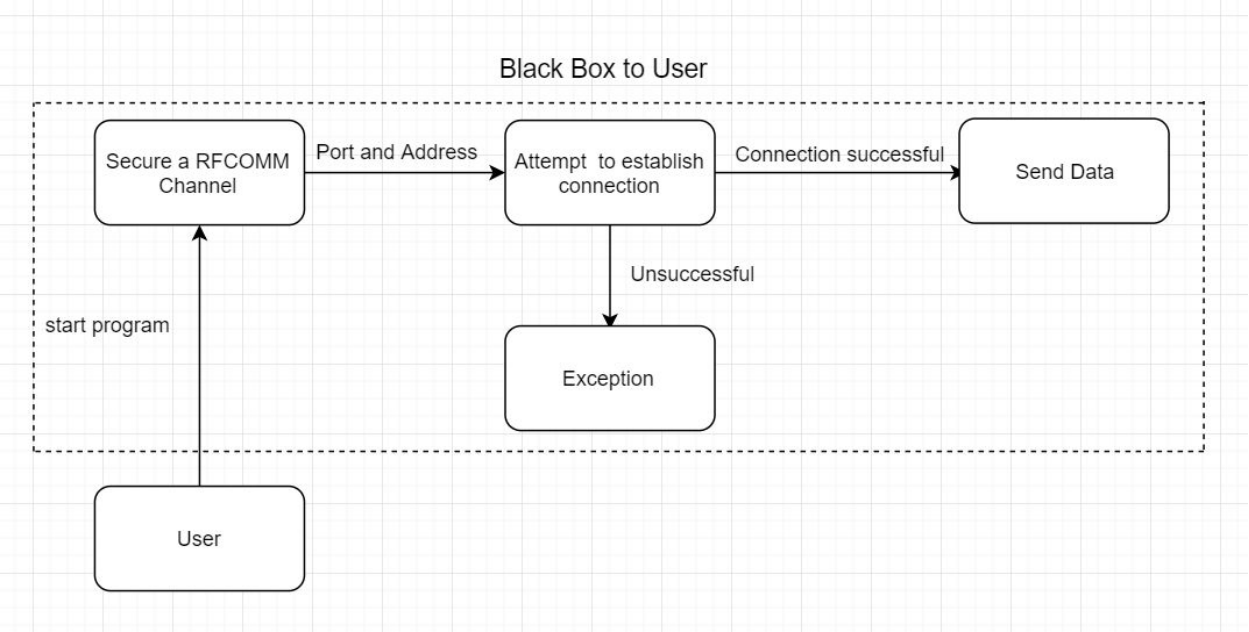

Figure 5: PyBluez Flowchart

Pyserial is more straightforward to connect and is much easier to install, as the user can simply install it through pip command. However Pyserial is not designed specifically for Bluetooth connectivity, in that the user would have to manually pair his laptop with the HC-06 chip, as Pyserial transfers data through a serial port. On both Windows and Mac this would require the operating system to first assign a port for the connected device. Figure 6 demonstrates how Pyserial works in our project. As shown in the graph what Pyserial does is simply push data to the assigned serial port, and nothing more.

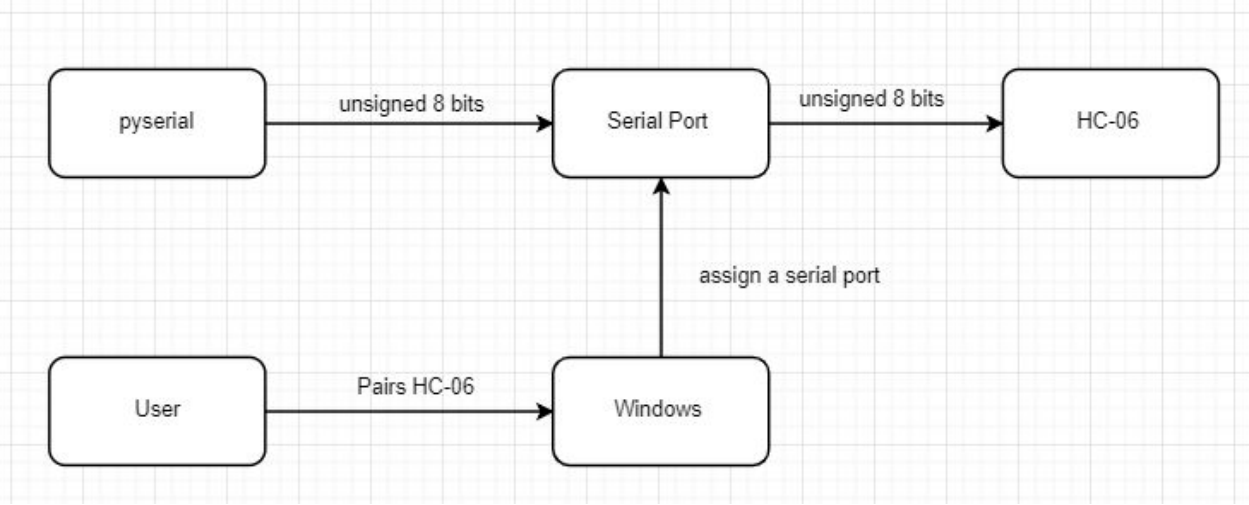

Figure 6: PySerial Flowchart

Bluetooth Module and Voltage Divider:

The HC-06 chip can provide a transmission range of 9 meters, equipped with a 2.4GHz digital wireless receiver and can operate at a low voltage of 3.3V to 5V. All these specs exactly meet the requirement of the communication module for our project. The Tx and Rx ports take in 3.3V whereas the power supply for this chip takes in 5V, so a voltage divider was required for conversion.

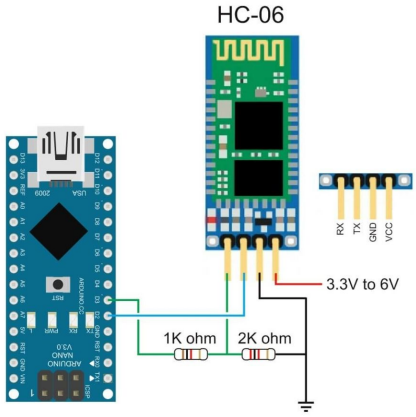

Figure 7: HC-06 to MCU Connection

Microcontroller:

For our initial design, we intended to use a 40 pin PIC16F877A. The microcontroller consists of 5 port registers, Tx and Rx ports for serial communication purposes, and 2 CCP ports used for speed control via PWM. An external crystal oscillator is required for the microcontroller to function. We believed that it would be the best option for this project since it was readily available in the lab and possessed all the necessary components for interfacing with the HC-06 and the motor control module. In addition, we possessed prior programming experience with PIC microcontrollers. The required circuit environment was to be transferred to a PCB; however, due to unforeseen circumstances that will be elaborated upon in the following sections, we made a last minute switch to the Arduino Mini. For reference, below is a pinout diagram of the PIC16F877A:

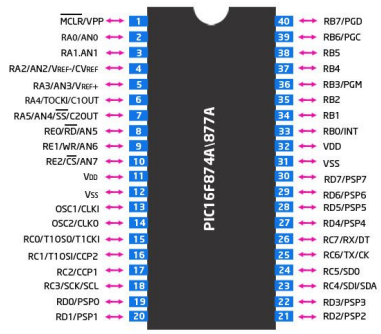

Figure 8: PIC16F877A Pinout

Motor Control Module:

We decided the use of an special IC, namely the L298N. The L298N consists of two internal H-bridges that can control two separate motors respectively. An alternative option would have been manually designing two H-bridge PCBs or perfboard based circuits to interact with the motors, but as the main focus initially was to facilitate the interactions between the software and the microcontroller, we believed that relying on the use of a pre-made motor control module could save us time and effort since the full project required all three major components to successfully operate in tandem (software, interface, and hardware). Had we succeeded much earlier in our endeavors, we could have had time to further increase the hardware complexity. Below is a labeled diagram of this IC:

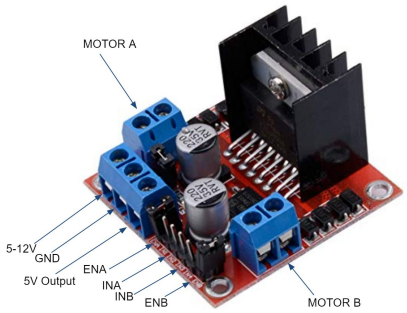

Figure 9: L298N

Motors:

Two ROB-13302 SparkFun brushless motors were used for the purposes of this project as they were readily available to us and possessed an acceptable DC motor voltage of 4.5V. These specific DC motors were available to us from the beginning as they were a part of a SparkFun hobby kit that we used in ECE110. Incidentally, the hobby kit cart also served as the chassis for our robot, in order to mitigate the potential difficulties that could result from needing to design our own.

Figure 10: Motors

2.2 Design Details

Hand Gestures, Web Camera and Camera Driver:

We discuss these three modules together because they do not involve a major hardware or software design. The hand gesture and web camera design do not require anything. The camera driver is used to open and close the camera and set the size and frame rate of the images. These functions can be achieved by built-in functions of OpenCV. The size of the images would be discussed in the classifier module.

Image Preprocessing Module:

Due to the decisions made previously, the most important part of this module is to detect the skin color under harsh light conditions. We use an RGB color threshold to mask the image and only keep the skin-color part while removing the background. Based on a large number of tests, we find that the key to detecting skin color is to set the blue color threshold relatively low and keep the other two colors high because human skins barely contain blue color. The final color threshold is R: 0-255, G: 50-240, B:100-150. The final result works in a wide range of skin colors and in real time test, the hand gestures of all of our teammates and our TA’s hand could be detected.

Classifier Module:

The CNN we use 8-node soft-max layer and use sparse categorical as the loss function. These designs are used because we have 8 classes and all the classes are exclusive, or in another word, one image can only be classified to be in one class. Since the light conditions have great influences on the performance, we collect the images of all hand gestures under different light conditions. There are totally 19,180 images in the training dataset. As shown in the figure below, there are some samples from the “stop” class, and the shape and contour of hand gestures can very much under different light conditions. We include the gestures under many different cases such that the CNN could make correct predictions based on the input images.

In addition, the CNN use rotated images or symmetrical images from the training dataset as aggregated dataset to train itself, so the classifier can recognize the hand gestures made by either left or right hand. However, two of the classes use both hands but the CNN can also recognize symmetrical gestures.

Figure 11: Stop Gesture Examples

Microcontroller:

As mentioned in the previous section, in order to interface with the Bluetooth module and the motor control module, the PIC requires one of the ports to be set as an output (Port B, in this case) in order to set voltages on the L298N H-bridge inputs (RB0-RB4, Pins 33 to 36), Tx and Rx connections between the HC-06 and the corresponding PIC pins (25 and 26) and CCP register outputs (16 and 17 to set the PWM enable pins on L298N. Additionally, the MCLR pin requires a pull-up resistor to set it high during operation in order to disable external resetting. Moreover, the PIC16F877A required an external crystal oscillator in order to operate. 16 MHz crystals were available in the lab, that possessed a rated load capacitance of ~18 pF. The parallel resonance condition must be met in order for the crystal to actually run at the intended operating frequency, which requires the use of load capacitors at each crystal terminal connected to ground such that the oscillation is stabilized. The value for the load capacitances are

$$C_L = \frac{C_1 C_2}{C_1 + C_2} + C_s \qquad \text{(Eq. 1)}$$

where C_1 and C_2 refer to the load capacitances, C_s refers to the stray capacitance (usually around 2-5 pF), and C_L refers to the crystal's rated load capacitance. We decided to use 22 pF capacitors, since in the context of the formula above, they result a ~16 pF rated load, which while not exactly 18 pF, isn't quite close and these capacitors were readily available for us in the lab. Our initial proposed schematic can be found in the Appendix, labeled as Figure 12. This decision was made to save space on the report.

The general program flow for this design is relatively simple. Regardless of the microcontroller used, it involves first initializing the necessary ports as inputs/outputs and set up the UART-based serial communication protocol with the Bluetooth module. Then, if the Bluetooth connection is established, set the port B outputs and the PWM registers to the necessary values for each input code corresponding to each gesture.

Motor Control Module and Motors:

Below is the block diagram for the H-bridge circuit in the L298N:

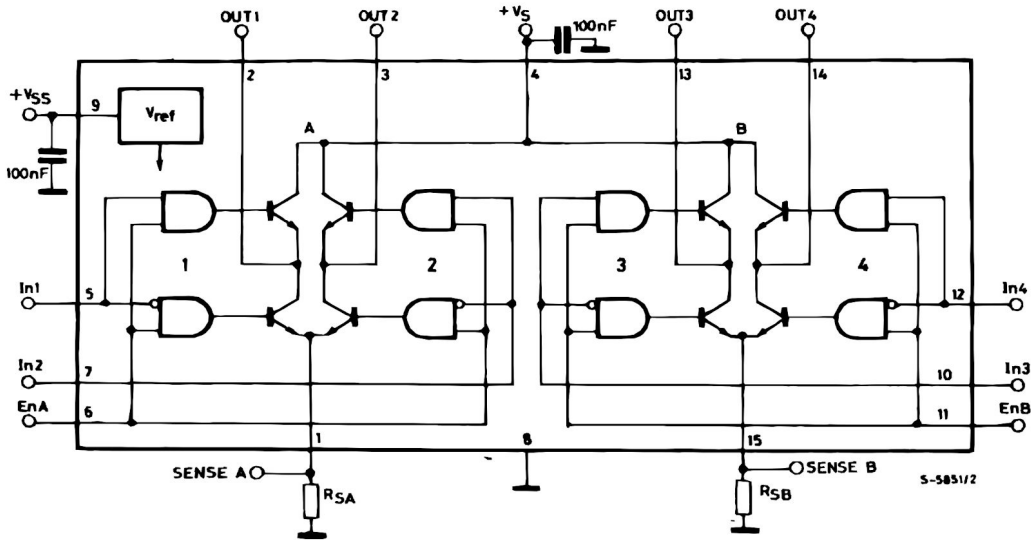

Figure 13: L298N Block Diagram

IN1 and IN2, as well as ENA control one motor and IN3, IN4, and ENB control the other. From the schematic, we can see that each adjacent input controlling each motor must be set to opposite values in order to facilitate forward and backward motion and both must be set to 0 for the motors to stop. ENA and B adjust motor speed since they are ANDed with the input signals to lower the average gate voltage based on the duty cycle. The L298N also consists of a voltage reference that provides a constant 5V output. We initially were planning on using this 5V output to power the PIC; however, the switch to Arduino proved that to be unnecessary. The L298N also requires a 5-12V source voltage in order to operate. We determined that at full speed (i.e. 100% duty cycle) there was a consistent voltage drop of ~4V between the input voltage and the motor voltage. Thus, we decided that a 9V input would be optimal for our design since the rated DC voltage of our motors was 4.5V.

2.3 Design Verification:

Machine Learning

Explanation:

Since the image input module, image analysis module and the classifier work together to make predictions, so if the accuracy of the classifier is high enough (over 70%), then it would prove that each submodule is fully functioning because the whole machine learning program could not work if any of the submodules breaks down.

Accuracy in the training process:

We use the software to select some samples from the training dataset and test the samples with the CNN to get the accuracy in the training steps. The picture below shows a small number of test results and the diagram of the accuracy and loss show how the tendency changes during the training. The final accuracy after all the training processes is 97.67%.

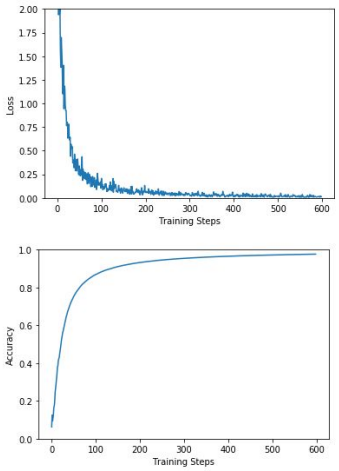

Figure 14 Accuracy in training

Accuracy in real time:

We use the same method proposed in the design document to verify the results of the classifier. One of our group members would hold his hand and pose one of the seven designed hand gestures before the camera for 5s to get 150 test images. We require at least 105 of them should be classified correctly, i.e. the predicted results should match the gesture shown. The test is repeated for 6 times for each classes. Then we get the real-time results as shown below.

	1	2	3	4	5	6	Average accurate rate
F	148	149	146	148	147	140	97.56%
B	150	150	150	149	150	150	99.89%
L	150	138	140	150	149	148	97.22%
R	150	150	150	150	149	146	99.44%
LB	144	132	132	148	146	145	94.11%
RB	150	127	144	149	147	110	91.89%
S	150	148	149	150	149	150	99.56%

Figure 15: Real Time Results

As shown in the table above, the average accuracy of the real-time tests is close to the accuracy in training processes. Two of the classes are very special. The RB and LB stand for turning right while moving backward and turning left while moving backward. The two gestures are made by two hands and they make the classifier confused sometime in the real-time tests so some of the test results fall out of range like the 110 in the RB line. Despite the two cases, the average accuracy matches the accuracy in the training so the overall accuracy is more than 90%. The worst case data still has 73.3% accuracy which satisfies the high-level requirements. In general, we successfully implement machine learning software with OpenCV, Keras, and Tensorflow libraries.

Hardware

PIC Issues:

Prior to to the week before the demo, we were able to verify that the PIC was outputting the correct values from the Port B and CCP registers, although we had trouble using them to control the L298N. However, during the week before the demo, the PIC was not even able to be recognized. For reference, we were using the PICKIT3 for our programmer/debugger, as they were readily available in the lab and were compatible with the PIC16F877A. While earlier the outputs were verified to be correct on the multimeter, we were suddenly receiving the following error:

```
Currently loaded firmware on PICkit 3
Firmware Suite Version.....01.48.18
Firmware type.....PIC18F

Target voltage detected
Target Device ID (0x0) is an Invalid Device ID. Please check your connections to the Target Device.
```

Figure 16: MPLABX IDE Error

As the programmer was unable to detect the device anymore, we went through all the necessary debugging steps, including ensuring that the configuration bits were set to the correct values, making sure the wire connections between the PICKIT3 and the MCU were functional, replacing our microcontrollers, sometimes even with other PIC models, replacing other components like the crystal oscillator and load capacitors, using other PICKIT3s, and verifying that the signals being sent from the PICKIT3 were correct. Specifically, we followed the steps associated with the Microchip Developer documentation, namely the section titled “Troubleshooting Invalid Device ID Errors.” The documentation mentioned that the common causes were a bad connection between the debugger and the MCU, the presence of external components on the programming pins, and missing connections between the pins. We verified that there was no problem with the PICKIT3 by probing the clock, data, and MCLR pins and ensuring that they were set to the proper voltages once the PICKIT3 firmware was initialized in MPLABX, in addition to repeatedly verifying that all the connections were correct, especially since we were using the same connections as when it was working before. We even tried seeking help from other TAs, but we came to a point where we decided that it wasn’t worth trying to make further progress and that it was best to downgrade our microcontroller to the Arduino Mini such that we can have a working project by the demo date, greatly reducing our hardware complexity.

Had we been given more time to work on this project, we would have attempted to further debug the PIC issues by using a PIC development board. We were initially reluctant about buying a PIC development board because they were somewhat expensive and we wanted to have a physical circuit for reference when designing the MCU PCB. In this case, however, a PIC development board would have mitigated the effort in continually replacing components on the breadboard, since most of the external components including the crystal oscillator and the power sources are integrated onto the board. In addition, the PICKIT3 can directly be plugged into the board without the need to wire each of its pins to the corresponding pin on the microcontroller. This would save us a lot of time with debugging and once the PIC is sufficiently programmed, we could then recreate the original circuit with the program still uploaded and continue from there.

Arduino

The Arduino setup was similar to the PIC setup, albeit much easier, with pins 7, 6, 5, and 4 functioning as IN1 to IN4 and pins 10 and 11 acting as digital PWM outputs to ENA and ENB. Pins 12 and 13 were connected to the Tx and Rx of the HC-06. The program flow was the same as we had mentioned in the

microcontroller design details. When testing the complete circuit, we found that the left wheel was more susceptible to friction effects than the right wheel, causing the vehicle to turn left when going forward. In order to program the PWM outputs, the analogWrite function was used in order to set the duty cycle of the motors accordingly, in which the acceptable range of values was between 0 (0%) and 256 (100%). Below are the PWM values used for each direction to compensate for friction while ensuring that the vehicle runs at the desired speed of 10 cm/s. For no motion, the IN pins were set to 0.

Direction	Left PWM	Right PWM
Forward/Reverse	160	192
Left/Reverse Left	120	255
Right/Reverse Right	255	120

Figure 17: PWM Values

The final schematic is also added to the Appendix.

3. Costs

The cost calculations are added to the appendix. For now, our cost calculation will only consist of the hardware component we've used. We did not encounter situations where we have to pay for the software libraries in the development of our project. However, this may not be the case for commercial uses. The labor cost will be calculated based on the average worker salary of the United States and the time to assemble the robot.

	Manufacturer	Quantity	Part Price
Arduino Board	Sparkfun(Arduino Uno)	1	$16
Bluetooth Chip	DST-TECH(HC-06)	1	$8
Power Source	9V Battery	2	$0.7 each
Motor Control	Qunqi(L298N)	1	$7 or below
Motors	Hobby Gearmotor (ROB-13302)	2	$4.95 per pair
Robot Frame	Sparkfun(Shadow Chassis)	1	$12.95

Total Hardware cost			$50.3

Labor Cost(per hour)	24.57
Time	60 Hours per person
People worked on the project	3
Total Labor Cost	$24.57*3*60 = $4422.6
Total Cost	$4422.6+$50.3=$4452.9

4.Conclusion

4.1. Accomplishments

Generally speaking, our project did work in the sense that it met all the High level requirements. The operating speed is satisfactory after modifications on the power supply for L298N and PWM setting in the arduino code. The response of the robot is prompt, in most cases it response to a change in gesture input within a second, transmission range is met as we have tested operating the robot in the 445 lab and have the laptop transmitting signals in another room. We've also cleared ways for running our application on both Mac and Windows with a few lines of code change.

4.2. Setbacks

The major setback for our project would be our failure to program the PIC, this directly let to our failure to design a PCB board. This setback
Forced us to use a perfboard and leaving most of the wiring exposed outside, making our cart extremely vulnerable during operation, when debugging we encountered many time the case where two wires accidently touch each other and cause a short and shut off the HC-06 chip. The lack of PCB also drastically decreased the complexity of our project, leaving the complexity of the hardware portion unfitting to a senior design. In general, our main issue was a lack of efficacious parallelization of modules.

4.3. Ethics

During the design of our project we carefully abide the IEEE ethical codes. We made sure that everything we used in our project that is from an external source

are clearly cities, following the IEEE code of ethic 9(9). To ensure the safety of our project, following IEEE code of ethic 1(9), we carefully followed the specs of every hardware component used in our project, making sure that no voltage exceeding component recommendation is provided. However due to our lack of PCB, many wires are exposed for our robot, this could be a safety issue since we've observed shorting effects on our cart.Another possible safety issue lies with the voltage divider During the demo we also noticed a high power consumption for the voltage divider we've built, which is most likely cause by the 2k and 1k ohms resistors we used to built the voltage divider. This has caused the wire used for powering the HC-06 chip to get quite hot. If we had time, we would have used a level shifter or a voltage regulator.

4.4 Future development

If given more time to develop on our project, we would like to fix the issues with the PIC, make a PCB to replace the exposed wiring we currently have, and to replace the Arduino board and L298N with PIC, two on board H-Bridge and a voltage divider. This will greatly reduce the cost of production of our robot and clear away the safety issues we have right now.

5. Citations

[1] "Touchless User Interface Utilizing Several Types of Sensing Technology," *PDF*. [Online]. Available: http://docplayer.net/70705266-Touchless-user-interface-utilizing-several-types-of-sensing-technology.html. [Accessed: 28-Feb-2019].

[2] S. Chen, H. Ma, C. Yang, and M. Fu, "Hand Gesture Based Robot Control System Using Leap Motion," *SpringerLink*, 24-Aug-2015. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-22879-2_53. [Accessed: 28-Feb-2019].

[3] N. T. Thinh, N. T. V. Tuyen, and D. T. Son, "Gait of Quadruped Robot and Interaction Based on Gesture Recognition," *Journal of Automation and Control Engineering*, vol. 3, no. 6, pp. 53–58, 2015.

[4] M. Wang, W.-Y. Chen, and X. D. Li, "Hand gesture recognition using valley circle feature and Hu's moments technique for robot movement control," *Measurement*, vol. 94, pp. 734–744, 2016.

[5] B. Iscimen, H. Atasoy, Y. Kutlu, S. Yildirim, and E. Yildirim, "Smart Robot Arm Motion Using Computer Vision," *Elektronika ir Elektrotechnika*, vol. 21, no. 6, 2015.

[6]"3.2. Communicating with RFCOMM," *Communicating with RFCOMM*. [Online]. Available: https://people.csail.mit.edu/albert/bluez-intro/x232.html. [Accessed: 01-May-2019].

[7] S. User, “Home,” *OSEPP*. [Online]. Available: https://www.osepp.com/electronic-modules/breakout-boards/91-bluetooth-module#. [Accessed: 01-May-2019].

[8] “Developer Help,” *Troubleshooting Invalid Device ID Errors - Developer Help*. [Online]. Available: http://microchipdeveloper.com/dtda:invalid-device-id. [Accessed: 02-May-2019].

[9] “ROB-13302 by SparkFun Electronics | Brushless DC Motors,” *Arrow.com*. [Online]. Available:https://www.arrow.com/en/products/rob-13302/sparkfun-electronics?gclid=Cj0KCQjwh6XmBRDRARIsAKNInDHyy2w4QpElFIyWmZGB5NM9ShnVJJmUYxFLRzuaZt141QHO3do2IvAaAlSMEALw_wcB. [Accessed: 02-May-2019].

Appendix

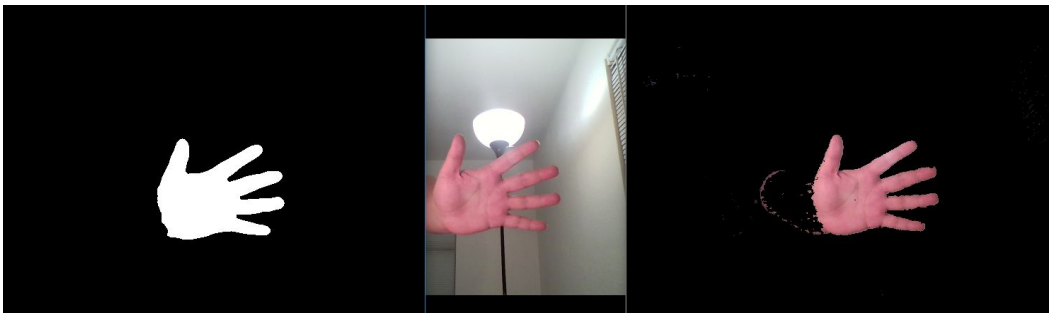

Figure 3: Ideal Cases

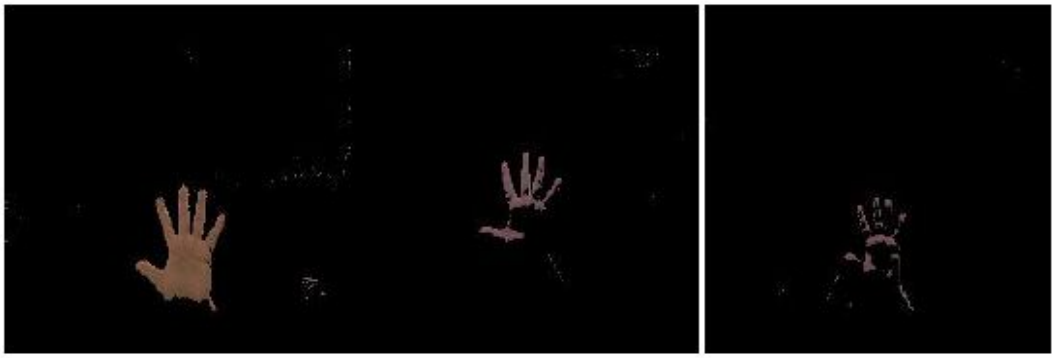

Figure 4: Real-Time cases

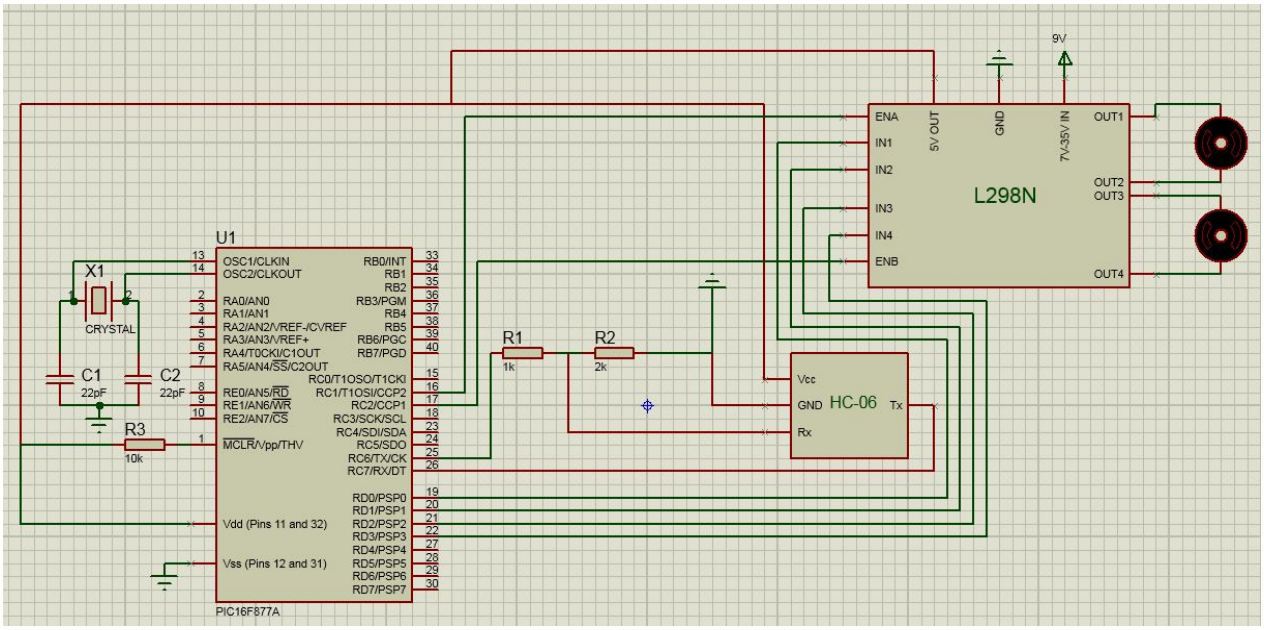

Figure 12: Proposed Hardware Schematic

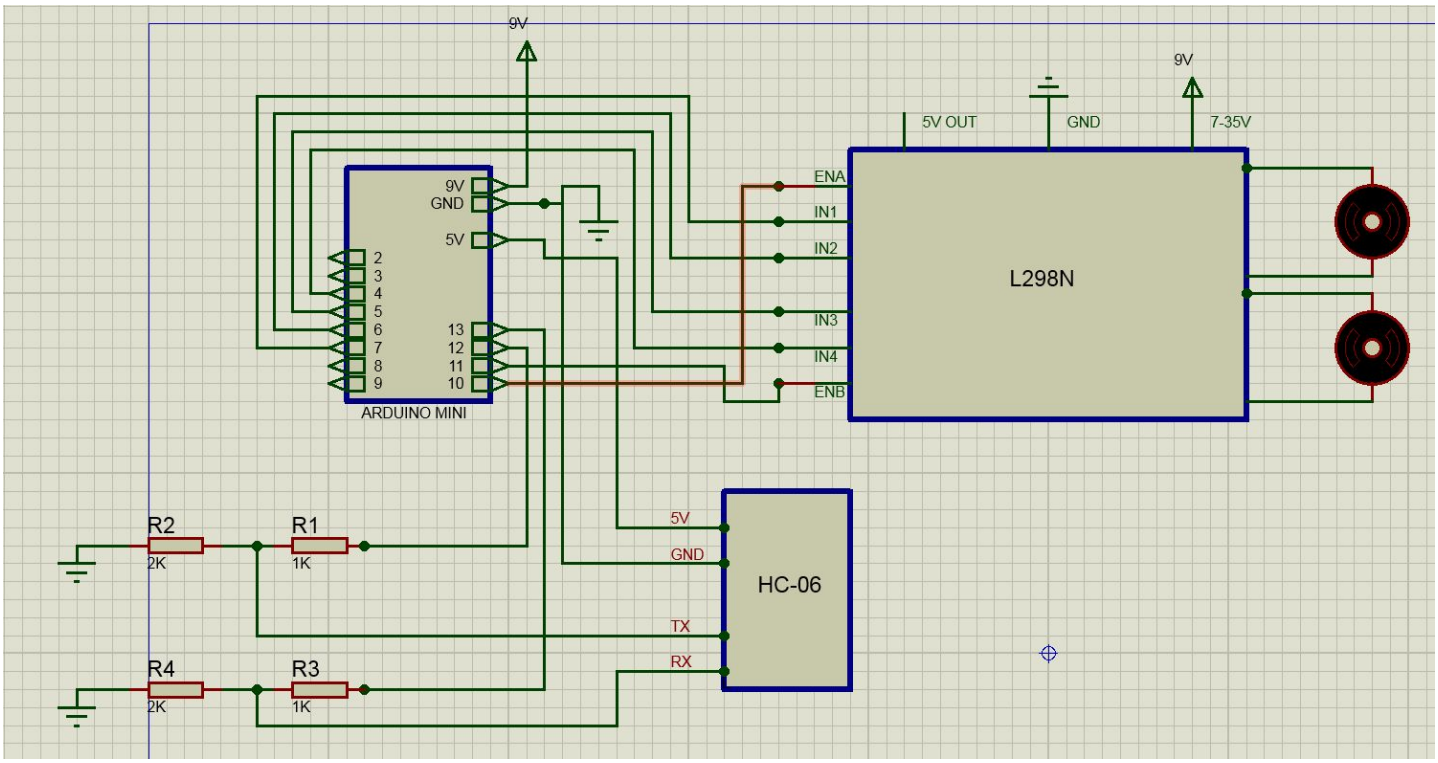

Figure 19: Final Hardware Schematic