April 2019

Object Manipulation and Control with Robotic Hand

Apiwat Ditthapron
Worcester Polytechnic Institute

Bhon Bunnag
Worcester Polytechnic Institute

Colin Matthew Buckley
Worcester Polytechnic Institute

Rebecca Eileen Miles
Worcester Polytechnic Institute

Object Manipulation and Control with Robotic Hand

Major Qualifying Project Report

Submitted by:
Colin Buckley (ECE)
Bhon Bunnag (RBE/CS)
Apiwat Ditthapron (CS)
Rebecca Miles (RBE/ECE)

Project Advisor:
Prof. Zhi (Jane) Li
Prof. Stephen Bitar
Prof. Loris Fichera

Submission Date:
April 20, 2019

Abstract

From industrial robots to nursing robots, object manipulation has become a growing area of robotics research. This Major Qualifying Project explores methods of teleoperation through the use of a wireless data glove able to detect multiple degrees of freedom. This project also explored methods for autonomous control. A computer vision model was developed by integrating two state-of-the-art Mask Region Convolutional Neural Networks (Mask-RCNN) models to create a final model for determining both object location and grasp angle. This modeling allows the Baxter Robot to autonomously detect and reach towards objects, as well as complete complex tasks. Using learning by demonstration, the robot learned how to perform these tasks without any additional hard-coding.

1. Introduction

The Industrial Revolution laid the foundations of a new manufacturing process: delegating micro-tasks to multiple entities for the sake of mass production of goods. However, these tasks were often not only repetitive but also dangerous. The 20th century introduced the new invention of robotics. This breakthrough was perhaps initiated by the advent of the first programmable robot invented by George Devol and Joe Engleberger in 1954, and used effectively on assembly lines[1]. These robots were able to mimic human's ability to manipulate objects. In the present day, industrial robotics is led by what is known as the 'Big Four', encompassing 'ABB' of Switzerland, 'Kuka' of Germany, and 'Fanuc' and 'Yaskawa' of Japan[2].

However, robotics is expanding from the industrial realm into other fields, including the medical and nursing field. An example of this is Dinsow (ดินสอ), a robot designed for aiding the elderly in multiple areas of their lives[3]. This robot has found much commercial success in Japan, whose rapidly aging population requires greater assistance.

Object manipulation is a crucial component for modern robotics applications. From industrial robots to prosthetic limbs, the desire for robots to be able to successfully and reliably grasp and utilize a variety of objects is growing. There exist many different robotic manipulators in use today, each with their own advantages and disadvantages.

1.1 Objectives

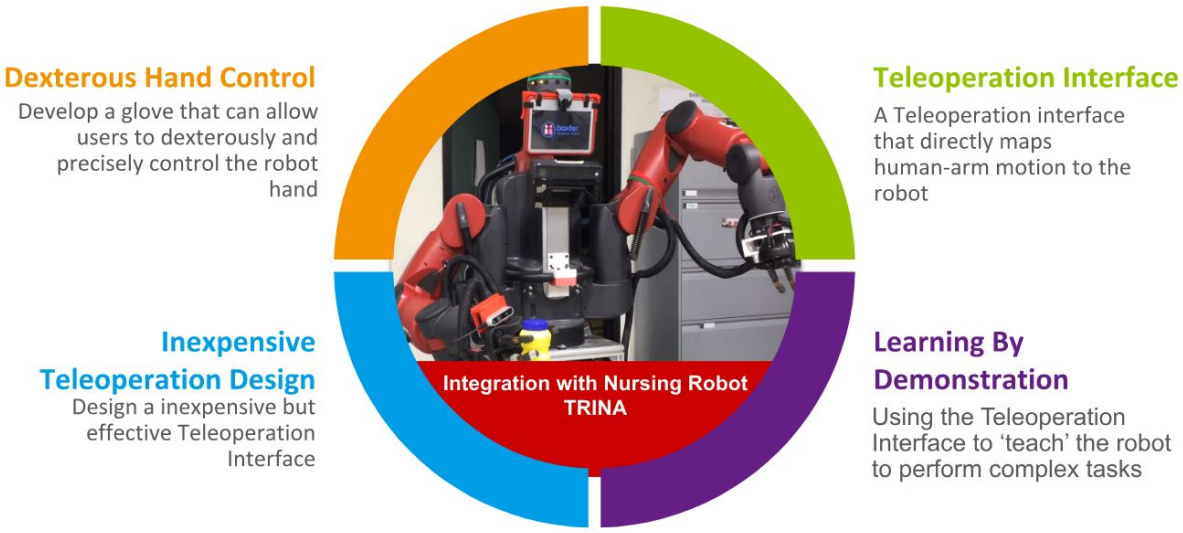

Figure 0:Outline of the Motivations and Objectives of the MQP

One of the objectives of this MQP is to perform grasping and object manipulation autonomously using Learning by Demonstration with the TRINA nursing robot. We propose a novel system using Probabilistic Motion Primitives (ProMP), and paired it with a light weight teleoperation interface for

demonstration data. This data was gathered and recorded in order to "teach" TRINA how to grab and drop objects.

Our other objective is the Data Glove, a novel invention used to control a robot hand up to five degrees of freedom. The Data Glove also played a crucial role in the aforementioned teleoperation interface. The Data Glove was designed to be lightweight, inexpensive and wireless. Along with these requirements, the glove needed to fulfill two primary tasks. Its first task consisted of providing an intuitive means of teleoperation for the TRINA robot. Its other task involved using the skills learned in the first task, to demonstrate the pertinent motions to TRINA and teach it to be able to perform simple tasks with the aid of computer vision and learning by demonstration.

1.2 Background

The Tele-Robotic Intelligent Nursing Assistant (TRINA) was developed by Zhi Li, Kris Hauser et al. in order to operate in potentially hazardous locations involving infectious diseases. TRINA's primary purpose is to perform simple repeatable tasks often done by nurses such as removing trays and retrieving and removing blankets from patients[4]. Each arm of the TRINA robot has seven degrees of freedom and requires inputs either from a terminal or game controller in order to follow commands.

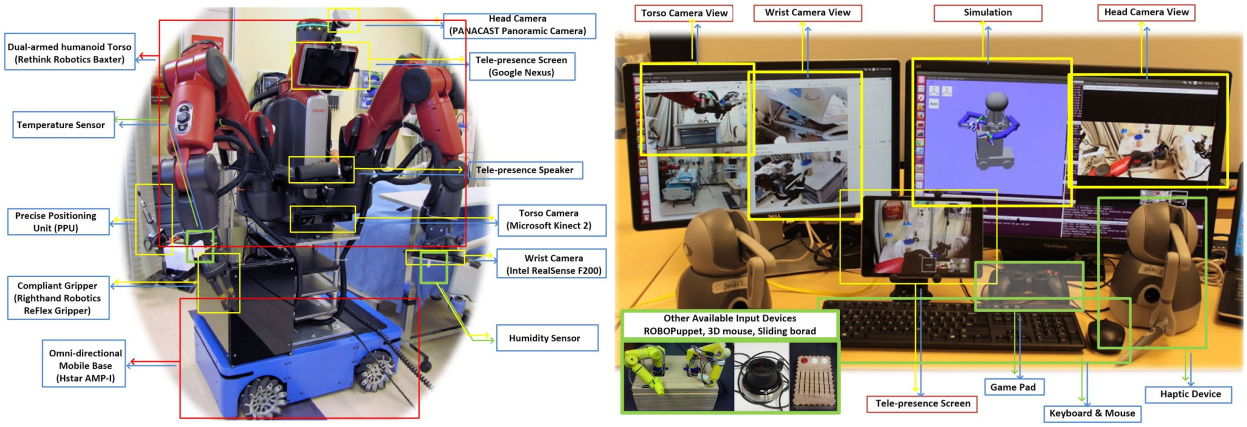

Figure 1:Tele-Robotic Intelligent Nursing Assistant (TRINA)

The TRINA system is the main robot that this MQP revolves around. "It consists of a mobile manipulator robot, a human operator console, and operator assistance algorithms"[5].

1.2.1 Yale OpenHand Project

The Yale OpenHand project was developed as a platform for 'rapid prototyping techniques in order to encourage more variation and innovation in mechanical hardware[6]. This objective is done through providing open-source hand designs, which could be downloaded from the internet where the modular components could be 3D-printed locally. Dr. Lael U. Odhner, a previous team member of the OpenHand project, went on to found RightHand Robotics, a startup that uses the OpenHand as the foundation for design.

1.2.2 ReflexSF Hand

Reflex SF hand is a research product from RightHand Labs. The hand comprises three sensor-free fingers to perform grasping tasks. Each finger has one degree of freedom from its blending and another one degree of freedom from coupled rotation of two fingers, resulting in a total of four degrees of freedom. However, last year MQP group, named YaleHand, modified mechanism of the hand by adding an additional Dynamixel motor to rotate a thumb, providing a half degree of freedom.

1.2.3 Previous MQP

This project is a continuation of work done by a previous MQP team which designed a sensor glove to teleoperate the Reflex SF hand. Through the use of flex sensors located on the thumb, index, and middle fingers the users finger movements could be detected. These movements were then transmitted to a graphical user interface (GUI) which controlled the robotic hand to mirror the user's movements. The robotic hand was controlled using messages sent through the Robotics Operating System (ROS), which transmits messages to different aspects of a complex system through a network of publishers and subscribers. This process meant that the data collected from the fingers of the data glove could be transmitted directly to the Reflex SF hand. As mentioned earlier, the previous MQP also modified the Reflex SF hand to add another degree of freedom on the thumb.

2. Manual Control of Robot Hand

In order to accomplish manipulation of the TRINA robot and to instruct it for Learning by Demonstration, we decided we would need some means of controlling and the movements of the robot. We identified several different means of controlling the robot and improved the graphical user interface to improve response time and readability for the user. In order to test the movement and feasibility we wanted a means to manually control the robot for testing purposes.

2.1 User Interface

The Reflex SF hand can be controlled out of the box via terminal commands. This is inconvenient for extended use and does not enable one to save a series of commands to be executed again in the future. The previous MQP team developed a graphical user interface (GUI) to control the Reflex hand which can be seen in Figure 2.

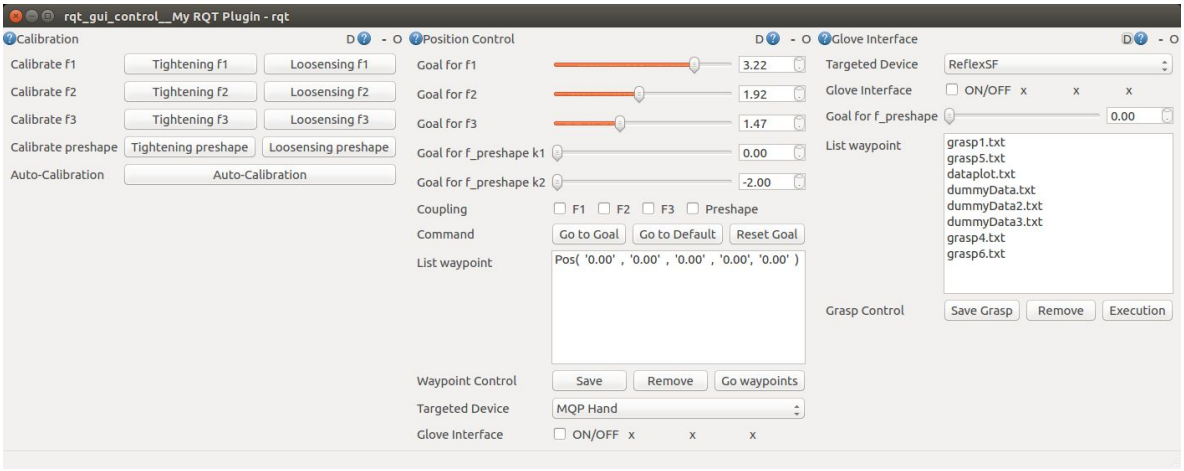

Figure 2: Original GUI

When we first started the project and were attempting to use this GUI for basic control, we found the interface to be counterintuitive and inconsistent. On top of this, the layout of the GUI was difficult to use. As can be seen, there were two checkboxes which supposedly enabled the user to control the Reflex SF hand with a wearable sensor glove that could track the users hand movements. Only one of these boxes appeared to work, however and even that was sporadic and unreliable. For these reasons, we concluded that the GUI needed to be completely redesigned and reimplemented before any real work could be done with the Reflex SF hand.

The final redesign can be seen in the following Figure 3. The reason for the previous GUI being unreliable turned out to be primarily due to unorganized and hectic code. Therefore, after a full rewrite of the GUI code, we were able to get comprehensive and reliable control of the robotic hand.

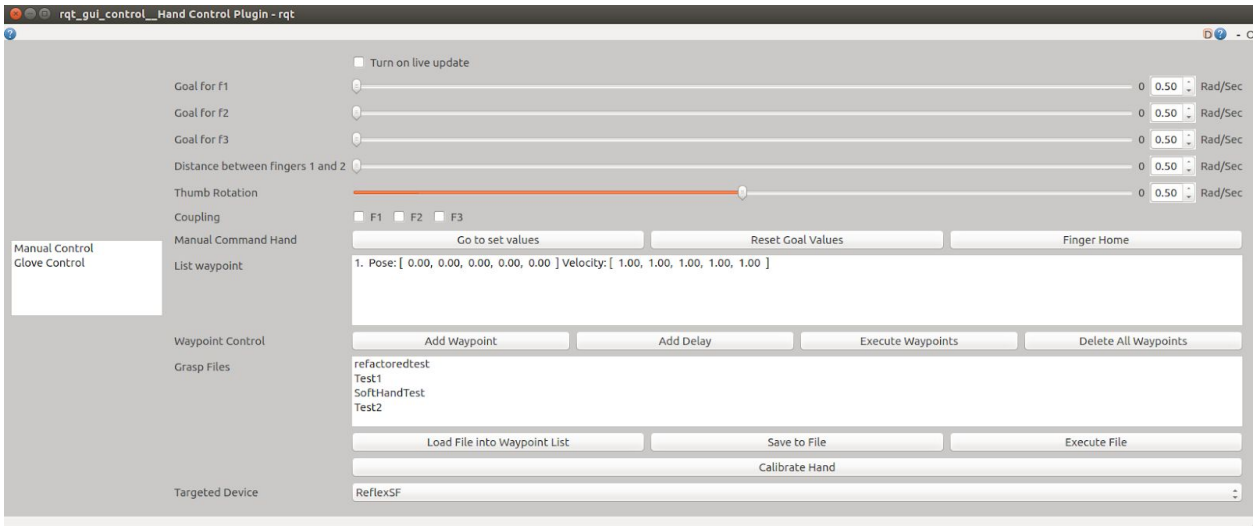

Figure 3a: Manual Control Interface

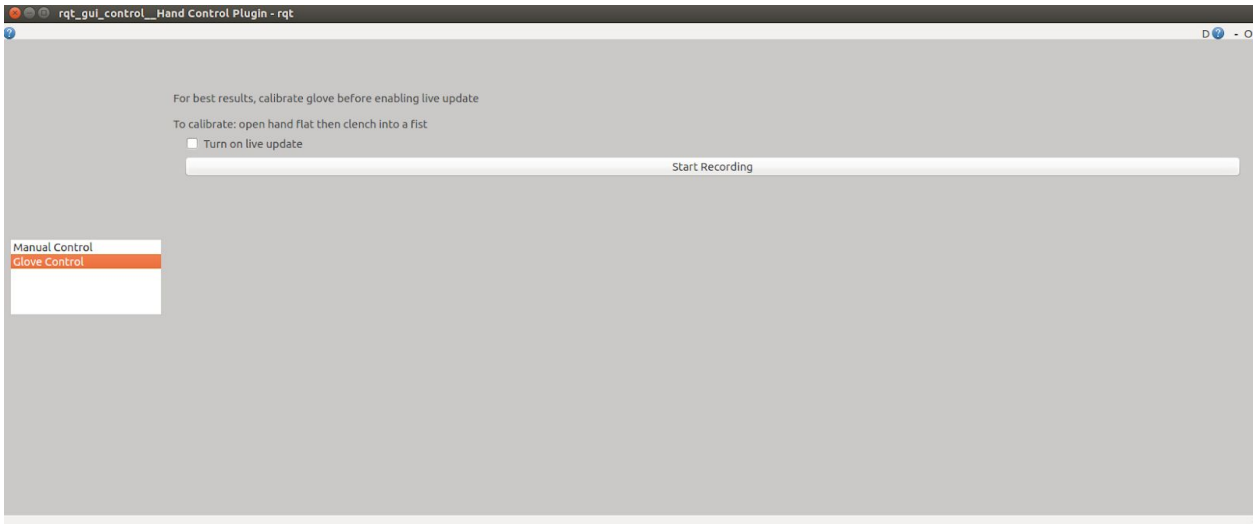

Figure 3b: Data Glove Control Interface

3. Data Glove

In order to effectively teach the robotic hand to grasp different objects, it was necessary to develop a teleoperation method which could map the position of the hand. The team wanted something that was intuitive and responsive. Rather than teaching the robot commands through a controller or terminal inputs, something more responsive and easier to practice. Due to the complex nature of the hand and the task of grasping objects, the best solution to teach the robot was through the use of a physical glove measuring the movements of the wearer's hand.

3.1 Problem Statement

The hand is a versatile appendage which can form a wide variety of shapes and move in numerous different ways. Therefore, in order to get an accurate model of how the hand moves to grasp different objects, it is important to be able to detect as many degrees of freedom of the hand as possible. There are primarily two types of data gloves on the market currently: ones which use computer vision to track finger movements, and ones which utilize sensors mounted directly inside the glove. Each of these options have their own drawbacks. For instance, cameras are easily thwarted when the object being grasped blocks its view of all of the fingers. It very quickly becomes a costly and limiting method of tracking the hands movement as the user must also perform all actions within the camera's field of view. Of the gloves which utilize sensors directly, nearly all use exclusively flex sensors located along the length of the fingers. This means that they are only able to measure finger flexion as depicted in Figure 5 and cannot get an accurate depiction of the entire state of the hand including how spread the fingers are.

For our purposes, we needed a glove which could measure the complete positions of the fingers regardless of obstructions. We first investigated the glove designed by the previous MQP team (Figure 7). It used an Arduino Nano connected directly to the computer for both power and data. The nano was then connected to three flex sensors placed inside a glove.These flex sensors did not fully extend the length of the fingers and they were connected with inflexible wire to a soldered protoboard. We tested this glove following the ReadMe file provided by the previous team and found the fingers to behave erratically when connected to the glove. Additionally, the glove controlled merely three of the five degrees of freedom available to the robotic hand. The glove controlled the flexing of the fingers, but the GUI was needed to control the rotation of the thumb and the rotation of the two synchronized fingers. After reviewing the previous MQP's results and testing their own glove we concluded it was too unstable and inaccurate for our mapping and controlling purposes.

Further research into existing data gloves similarly came up lacking. Since none of the gloves on the market were able to suit all of our needs and typically costed around $250, we decided to design our own data glove. The fundamental principle of a data glove is simple, detect the hand shape by taking direct measurements of the users hand. As mentioned earlier, there are data gloves currently on the market that are intended for Virtual Reality gaming. In fact, the original data glove that last years MQP used was

purchased though it was ultimately redesigned to increase the data rate and make it compatible with Linux and the Robotics Operating System (ROS).

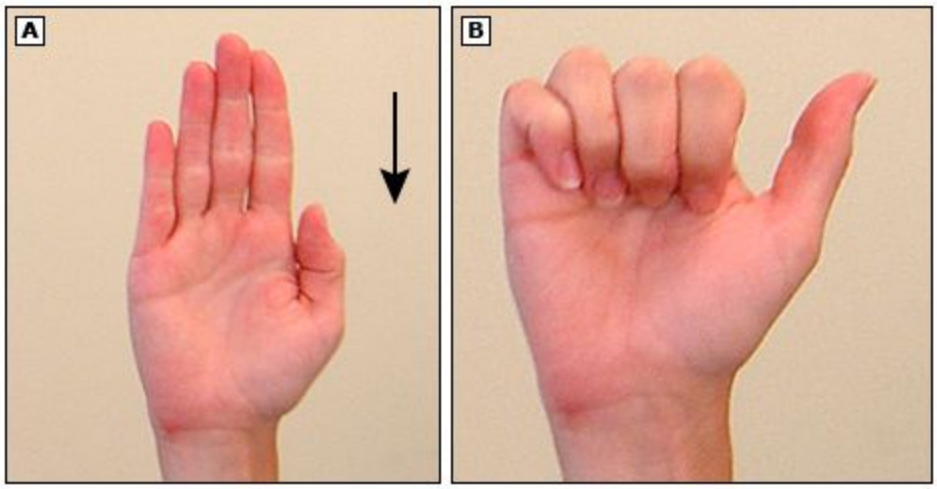

Figure 4: Finger Flexion

When grasping an object, it is essential to be able to measure the distance between a users fingers along with the fingers flexion. This movement is referred to as finger adduction and abduction as illustrated in the following Figure 5.

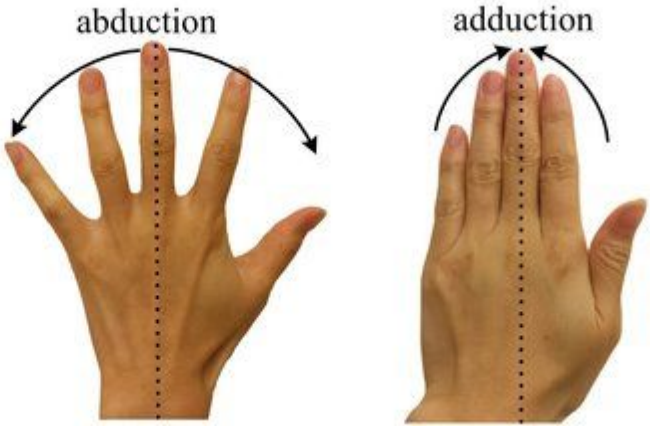

Figure 5: Finger Adduction and Abduction

3.2 Design: Previous Work

Virtual reality video games offer the highest demand for research into gloves designed to track the movement of the human hand. The CaptoGlove, depicted in the following Figure 6, was one such glove and was purchased by the previous MQP team to serve as a basis for the autonomous teleoperation of the Reflex SF hand.

Figure 6: CaptoGlove

They found that since the glove was intended for gaming, it was designed to work with Microsoft Windows and there were no alternatives that supported Linux. As it was necessary for the data collected from the hand to be sent to the Reflex SF hand via ROS messages, it was most convenient to have the data glove be compatible with Linux. For this reason along with the overall difficulties that the previous MQP team had getting the CaptoGlove to work with the Reflex SF hand, the team decided it would ultimately be easier to make their own data glove instead.

The design of the previous MQP's data glove utilized the exterior of the original CaptoGlove to house 3 flex sensors located along the thumb, index and middle fingers. Figure 7 illustrates the sensors laid out along their respective locations in the glove.

Figure 7: Redesigned CaptoGlove

The flexion of the fingers was measured with the illustrated flex sensors connected to an Arduino Nano which transmitted the data via ROS messages through a MiniUSB connection.

3.3 Our Design

The primary issue with the previous years glove was that it could only measure finger flexion. As it is crucial for our purposes to be able to measure finger adduction and abduction, the original glove was not of much use to us. It should be noted that the previous MQP also needed to send the desired distance between the fingers to the Reflex SF hand, yet as their glove was not designed to measure this metric, they had to manually input a value using the GUI described earlier. When using the glove as a method of teleoperation, there is an assumption that it will be able to control the entirety of the hand in all of the desired dimensions (control finger flexion, adduction and abduction). Requiring the user to manually input a value for finger adduction and abduction defeats the purpose of using the glove: if the user has to input one parameter they could just as easily input all parameters.

Measuring the distance between fingers is a fundamentally difficult problem. There are no, or very few, sensors currently on the market which are designed for this purpose. We investigated several different types of premade sensors to measure the finger adduction and abduction but our parameters required the sensor to be able to fit inside the finger of the glove. We investigated the use of Hall Effect sensors but could not find one suitable for our project. Such sensors were either too large or too small to effectively integrate. If the range of sensing was large enough the sensor itself would be too cumbersome to fit along a finger, but if it was sufficiently small the data range would not be significant enough to measure to a reasonable degree. Additionally, placing multiple sensors in the glove would cause them to interfere with each other's magnets and generate inaccurate readings. Therefore, we determined that in order to accurately collect data on the users hand-shape, it was necessary to design custom sensors.

Ultimately, the design which we settled on utilized a 555 timer and a custom capacitor. A 555 timer is an integrated circuit which produces a square wave with a frequency dependant on a capacitive input. The capacitive input was 2 copper plates attached with velcro such that the copper surrounded the finger like a ring and could be adjusted to fit different hand sizes. Such a setup created a capacitance approximately varying between 0.05nF and 0.07nF depending on the position of the fingers. Having the copper plates be rings enabled us to maximize the surface area of the sensor and have it continue working no matter the position of the fingers. A total of three copper rings were installed in the glove on the index, middle and ring fingers of the hand with the middle finger ring being a ground plate.

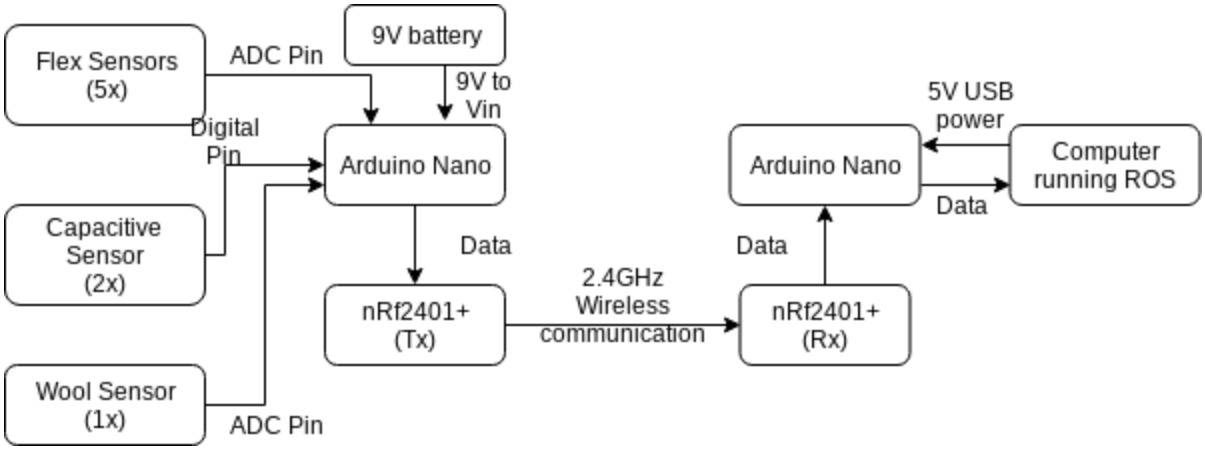

Figure 8: Glove Block Diagram

The block diagram in Figure 8 illustrates the main components within the glove. The whole system consists of two Arduino Nanos and two nRF2401+ communication devices. The arduinos act as microprocessors for the information that is gathered and communicated and work in between the sensors, the nRFs and the computer. The transmitting arduino and its associated components are all located within the physical glove and the data is sent to the receiving arduino.The arduino on the transmitter side takes inputs into its ADC and digital pins from the different sensors, including flex sensors, capacitive sensors and the custom wool sensor. The data is mapped and processed by the arduino and sent in an array via the nRF2401+. It is powered by a 9V battery connected to the Vin pin of the Arduino Nano. The receiving nRF decodes and processes the data and relays that information to the connected computer

running ROS via a USB cable. That computer then interfaces with the TRINA robot which mimics the glove wearer's movement

Figure 9 below demonstrates the capacitor circuit using the variable capacitance of the copper plates to alter the output frequency.

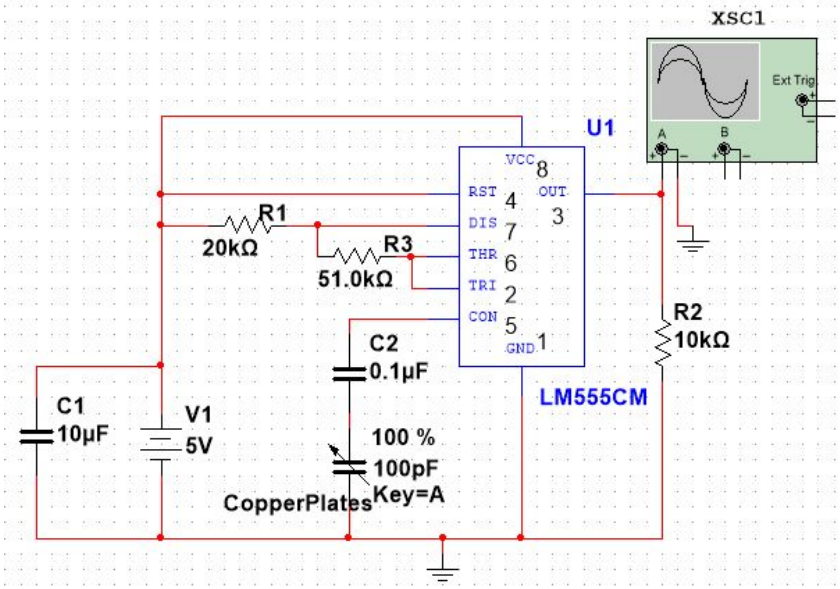

Figure 9: 555 Capacitor Circuit Diagram

The Index-Middle Capacitor data in Figure 10 measures the distance between the index and middle fingers using the capacitor sensor in between them. The mapped output value on the y axis of all following graphs refers to the final scaled value that gets sent to the robotic hand to control it. The capacitor sensors are scaled from 0-50 while the other sensors are scaled between 0-100. The blue sinusoidal movement shows the mapped values of the sensor as the hand alternates between a closed fist and an open hand. The minimum and maximum values for the sensor were taken separately. To eliminate impulse noise, a 10 point median filter was used. As can be seen, there is a clear distinction between when the fingers are spread out or "Open", close together ("Closed") and when they are moving apart and together to create the sinusoidal signal observed. The resolution of the data is relatively low but this can be attributed to the method in which the frequency was measured: through a pre-built library that was specially designed to measure frequency on a specific pin (D5) on the arduino nano. To get better resolution, a frequency to voltage converter could have been used and inputted into an analog pin on the arduino which likely would have resulted in more accurate data overall.

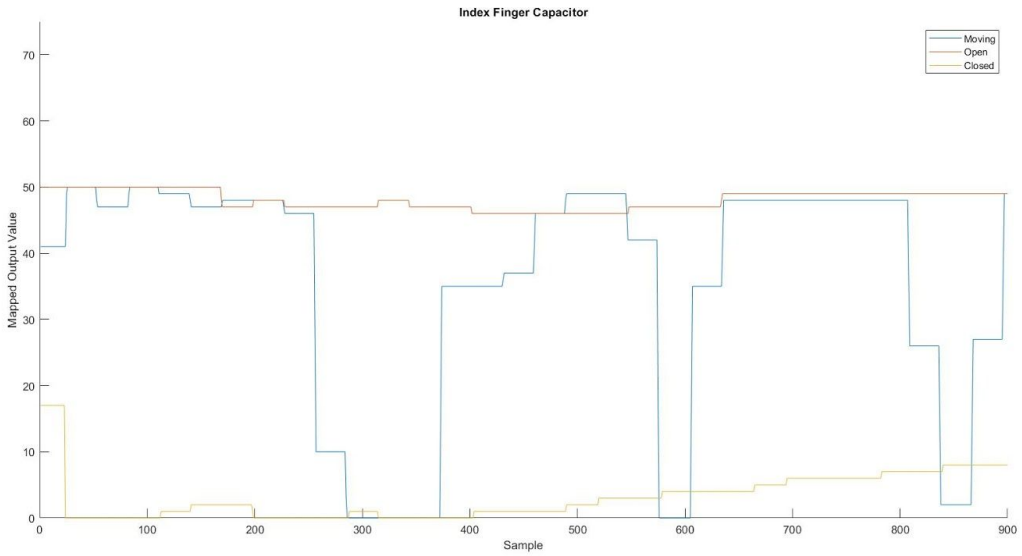

Figure 10: Capacitive Sensor Data

As with other data gloves, we determined that the easiest and most reliable way of measuring finger flexion was to use flex sensors located along the backs of the fingers. In order to get a complete depiction of the hand we installed a 4.5 inch long flex sensor along each finger of the hand. Figure 10 demonstrates the moving flex sensor data for each finger. In that experiment, the gloved hand alternated between an open palm and a closed fist. This repeated gesture generated the sinusoidal pattern seen in the graph below. The x-axis represents the number of samples taken. Samples were taken at a rate of 2 samples per second. The y-axis represents the mapped value between 30 and 100 that the ADC registered. The slight inconsistencies of the thumb value likely arise from the differing position the thumb reseted in, either over the fingers, or held inside the fist.

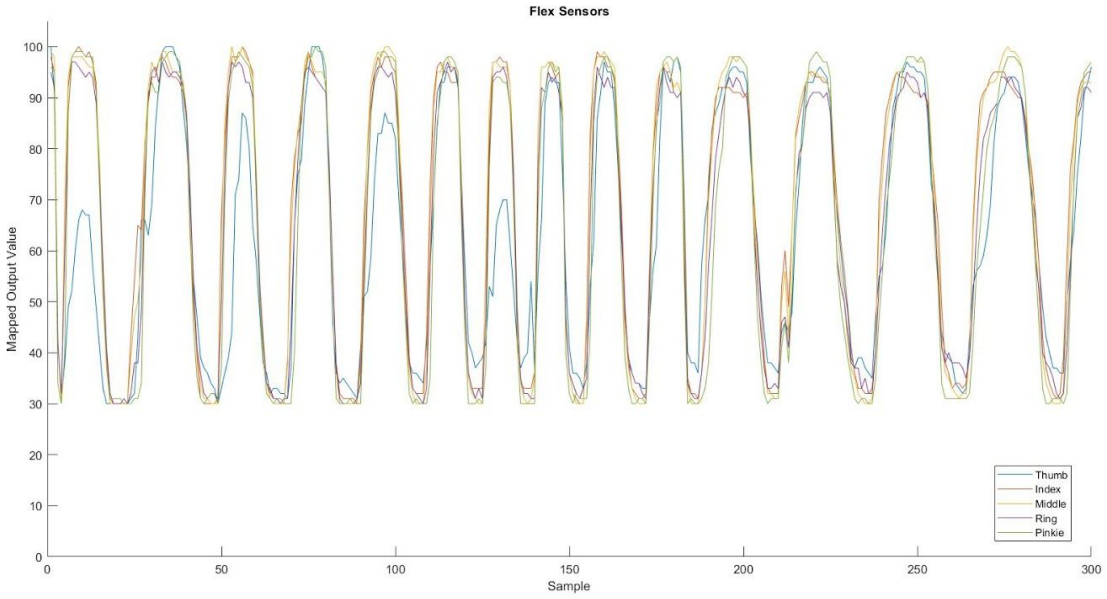

Figure 11: Flex Sensor Data

Measurement of the adduction and abduction of the thumb was the next issue addressed. While the fingers consisting of three bones are only able to move approximately 1 cm from side to side, the thumb is capable of moving several inches about a much more versatile field than the rest of the fingers. As a result, the capacitive sensor which was used for the other fingers would be ineffective for the thumb simply because the air gap which served as the dielectric in the capacitor would be too large and variable. Therefore, we developed a resistive sensor that is constructed from a blend of wool and stainless steel fibers.

The construction of the sensor is relatively straightforward. Wires are first soldered onto copper pads which are placed atop a bundle of stainless steel fibers with the loose ends pushed to one side as shown in Figure 12a the wool surrounding each copper pad is then needle felted into a firm felt leaving the midsection relatively loose. The stainless steel fibers are then arranged such that there is not a direct connection between the two copper pads, rather overlapping fingers which could easily touch if the sensor is bent as seen in Figure 12b. Finally, the wool and interior stainless steel fingers are needle felted into a small compact package shown in Figure 12c.

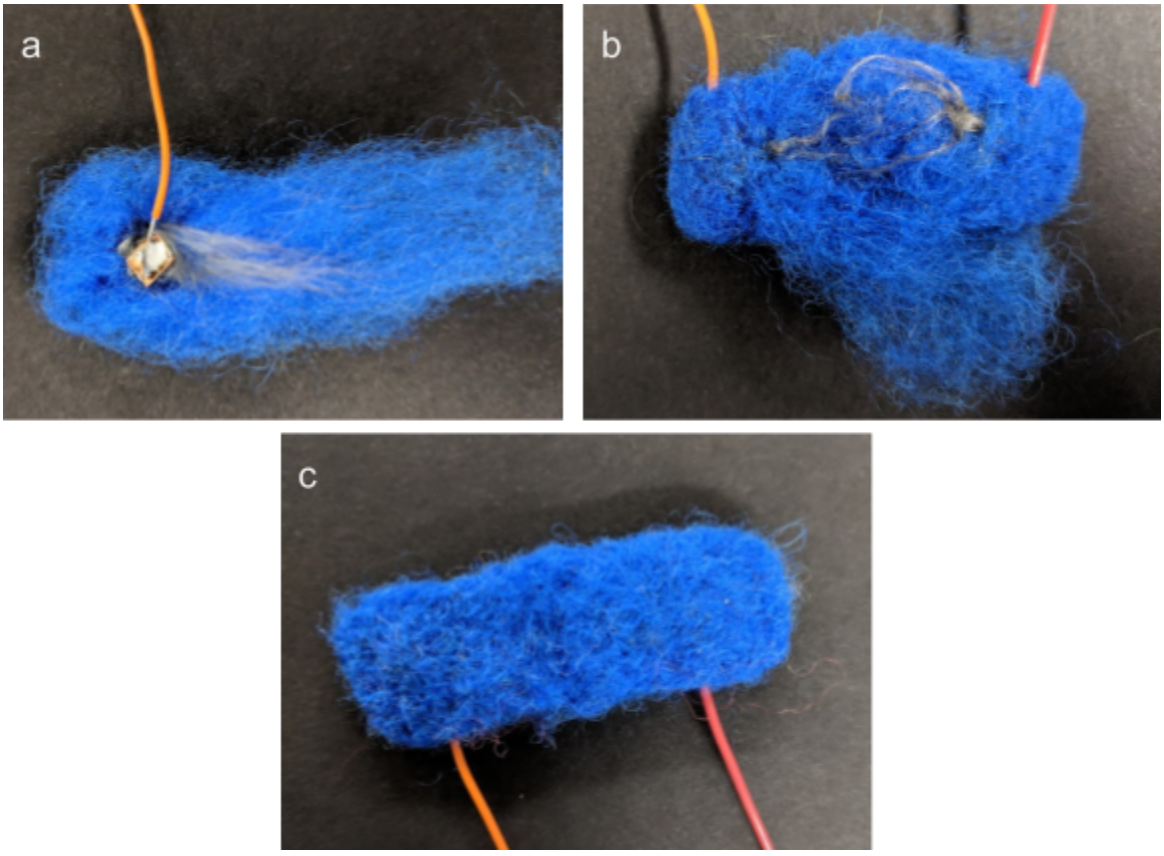

Figure 12: Construction of Resistive Wool Sensor

The wool sensor data in Figure 13 shows a clear demarcation between the minimum and maximum values generated by that sensor. This sensor measures the lateral movement of the thumb towards the other fingers. The maximum values were taken when the hand was held open and the minimum values were taken when the hand was held closed in a fist. For this graph, a 15 point median filter was used to eliminate impulse noise however as the data still appears to be rather noisy, a low-pass filter would be able to clean it up further.

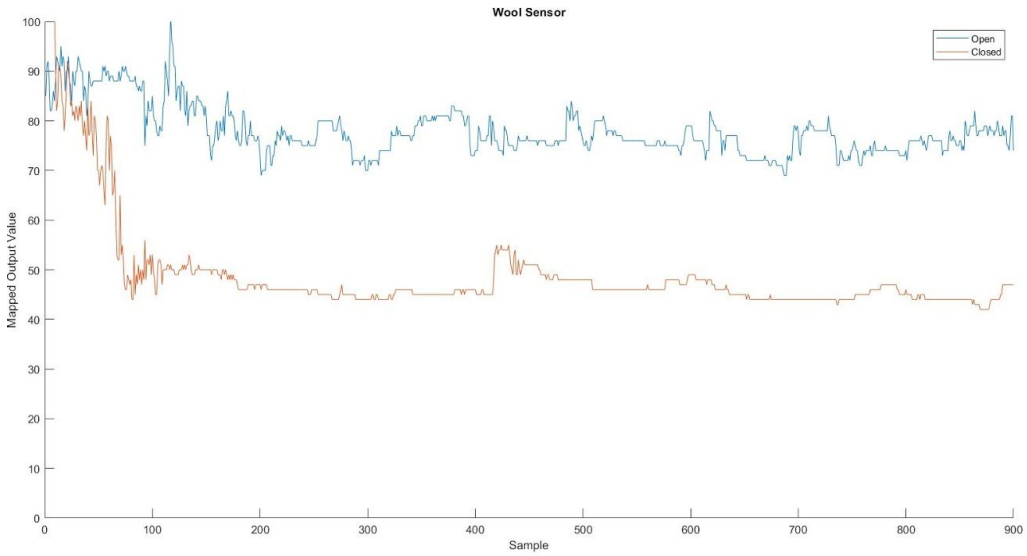

Figure 13: Wool Sensor Data

3.4 Data Transmission

Once data had been collected from the hand, it was wirelessly transmitted to a receiver. We originally wanted to use bluetooth to transmit data from the hand to the computer but we eventually abandoned this idea for several reasons. First, our glove was intended to eventually be capable of dual hand manipulation which would require more than one glove to be connected to the computer at a time. While it is not impossible to do this with bluetooth, it became more complicated than seemed reasonable given that our focus was on getting reliable sensor data and not having a fancy way of transmitting the data wirelessly. Secondly, the primary appeal to using bluetooth was the ability to avoid needing to plug in a USB receiver into the computer yet the desktop in the lab which is used to control the TRINA robot did not have bluetooth built in and therefore we would have had to purchase a USB bluetooth dongle anyways. Another option which we discarded was to use UDP for wireless communication. As we were using an arduino nano, it would technically have been possible to use UDP, however, it would have required an ethernet shield and, in addition, would have required us to connect the arduino to the WPI wifi which would have been an ordeal to say the least. We therefore discarded the idea and opted for the simple, well tested NRF transmitter/receiver module shown in Figure 14. These low power modules transmit at 2.4GHz and seem to be the standard for arduino hobby projects using wireless communication between multiple devices. This means there are significant resources available for their use.

Figure 14: NRF Transmitter/Receiver

3.5 Schematic

Figure 15 below represents the integrated components of the glove. We chose to use an Arduino Nano microprocessor as the processor for our glove due to its compact size and shape as well as its low cost and ease of use. The glove is designed to be wireless and untethered, so the glove will be powered through a 9 volt battery and distributed through the microprocessor. Analog to Digital converters are used to identify the different degrees of strain for the three different flex sensors connected to the thumb, index finger, and middle finger respectively, as well as the prototype wool sensor. Each of these sensors are connected to a voltage divider to stabilize the values and yield the greatest range for the data.

The two identical circuits below the Arduino are the 555 timer circuits described previously. Each of the circuits are connected to capacitive rings around different fingers which yields a variable capacitor value. The change in capacitance alters the frequency of the square wave generated by the 555 timer. This difference in frequency is measured by the arduino and mapped to appropriate values.

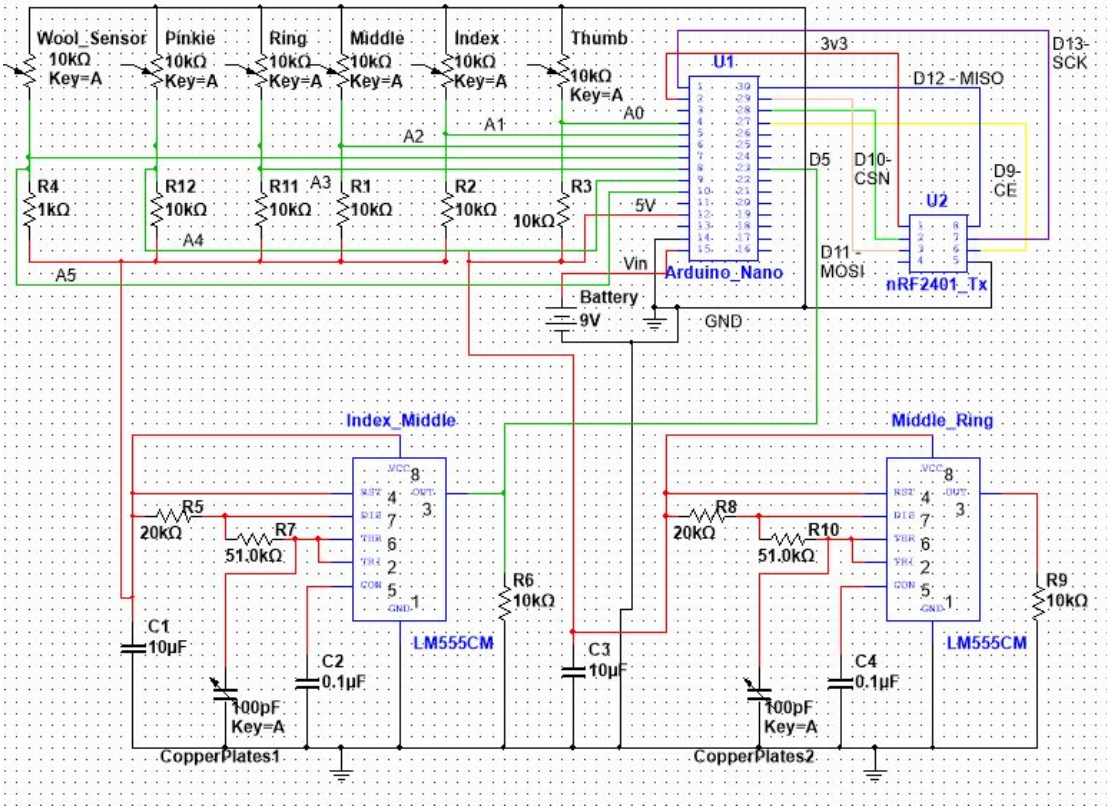

Figure 15: Glove Schematic

Connected to the Arduino's 3.3 volt rail is an nRF2401 transceiver. This component sends the gathered data as an array of integers to another nRF2401 which acts as a receiver. It transmits every 500 milliseconds across a frequency of 2.4 GHz. The receiver nRF is connected to another Arduino Nano hooked up to a laptop running ROS, which then relays the data to the robotic hand.

3.6 Power Consumption

Because the glove is now unfettered by the power and information cables, it needs to be able to power itself for extended periods of time. The table below contains the power concerns for the wireless glove.

	Part	Quantity	Power Consumption (A)	Specifications	
Arduino Nano	Processor: ATMEGA 328P	1	0.00036	active mode pins max specs: current=40mA, voltage=5V	
	Voltage Regulator				
	Board	1	0.019		
Bluetooth	Processor: CC2541	1	0.009		
Capacitive Sensors	555 timer	2	0.0005	*no load Max i, V=5	
Flex Sensors	flex sensor in voltage divider	3	0.0001363636364	*110k ohm	
Transmitter	nRF2401	1	0.009		
				Bluetooth Total:	0.02863636364
				nRF Total	0.02863636364

Table 1: Power Consumption Calculations

Equation 1: $(0.02863636364A\ /1000\ )/580mAh\ =\ 20.25\ hours$

Both Bluetooth and nRF are listed above because the team wanted to test the power consumption in a side by side comparison. While the power consumption came out approximately the same, the Bluetooth was ultimately dropped due to compatibility issues and difficulty in integrating it with the arduino.

In addition to battery capacity, size and weight also needed to be considered. The power source could not significantly weigh down the hand or occupy too much precious space on the glove. Based on the above data and calculations, a 9 Volt battery with a life of 580 mAh could power the glove for approximately 20 hours at a time. Given the nature of the project, 20 hours for a single battery seemed like a reasonable expectation for a glove that would not be used for extended periods of time.

3.7 Finished Product: Data Glove

Once all of the components had been designed, they were attached to a spandex glove via velcro. The arduino nano and PCB were placed into a 3D printed box shown in Figure 16a and b where 16a shows the CAD for the box and 16b shows the PCB mounted inside the finished product. This was done both for protection and so that velcro could be attached to the bottom to mount the box onto the rest of the glove. Originally, it was intended for the PCB and battery to be mounted on the back of the hand, however despite making the box as small as possible, it was just too big and clunky to reasonably place on the back of the hand. As a result, we decided to move the box back to a cuff on the wrist. In order to prevent the user from drawing the wires too taught and potentially breaking the solder connections to the sensor or the PCB, we decided to physically sew the cuff onto the end of the glove to limit how taught it was possible to make the wires. In order to make the wiring as neat as possible, wires for the sensors were passed through a hole in the bottom of the PCB case and into a hole on the top of the wrist cuff. The wires then threaded through the cuff and out the bottom as seen in Figure 17. This allowed the velcro attaching the box to the cuff to be unobstructed by the wires.

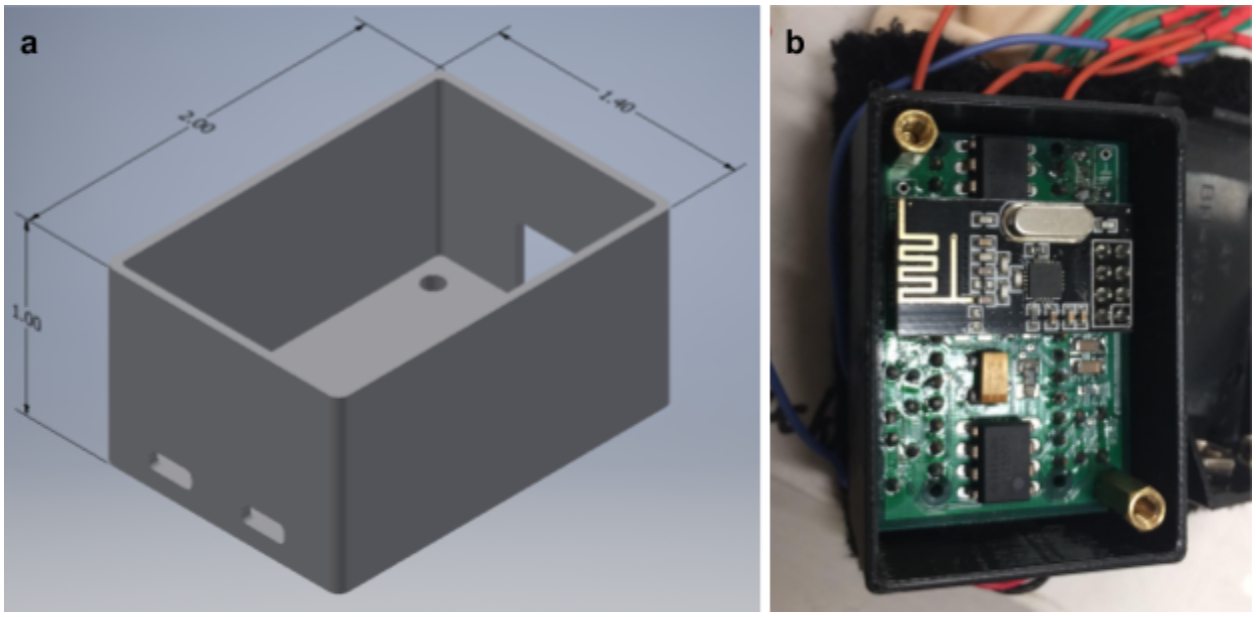

Figure 16: PCB Box

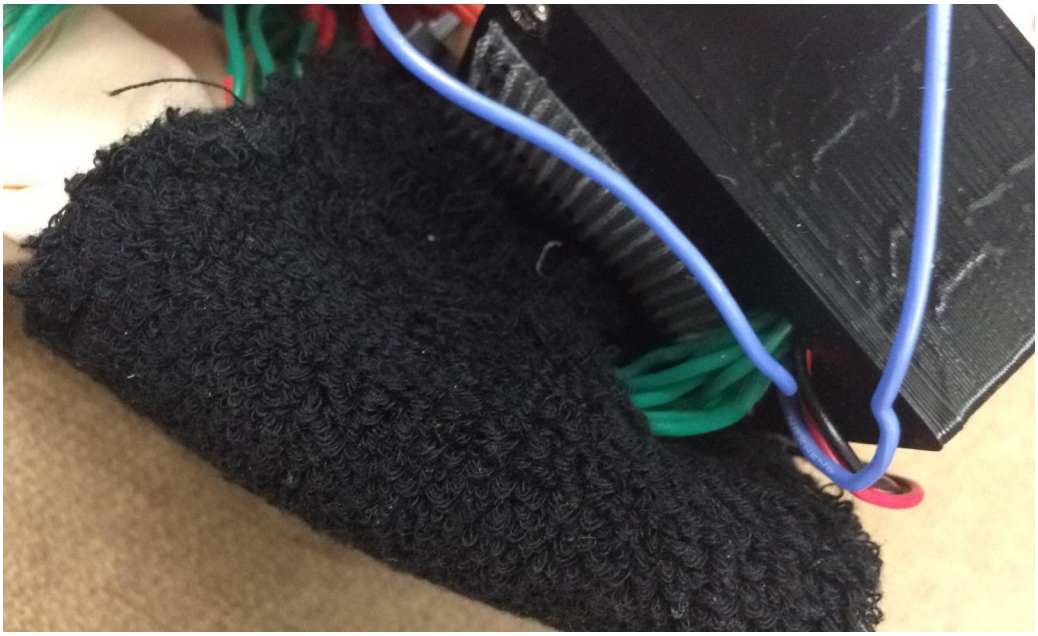

Figure 17: Wiring of the PCB to the Glove

We chose to mount the sensors on a spandex glove as it would fit snugly on any size hand, enabling the sensors to get the most accurate measurements of the fingers as there would be limited amounts of slack. Once the sensors were mounted, a thin cotton glove was placed on top to protect the sensors and prevent the copper rings from shorting. The final construction can be seen in Figure 18 which depicts the finished glove with and without the cotton glove.

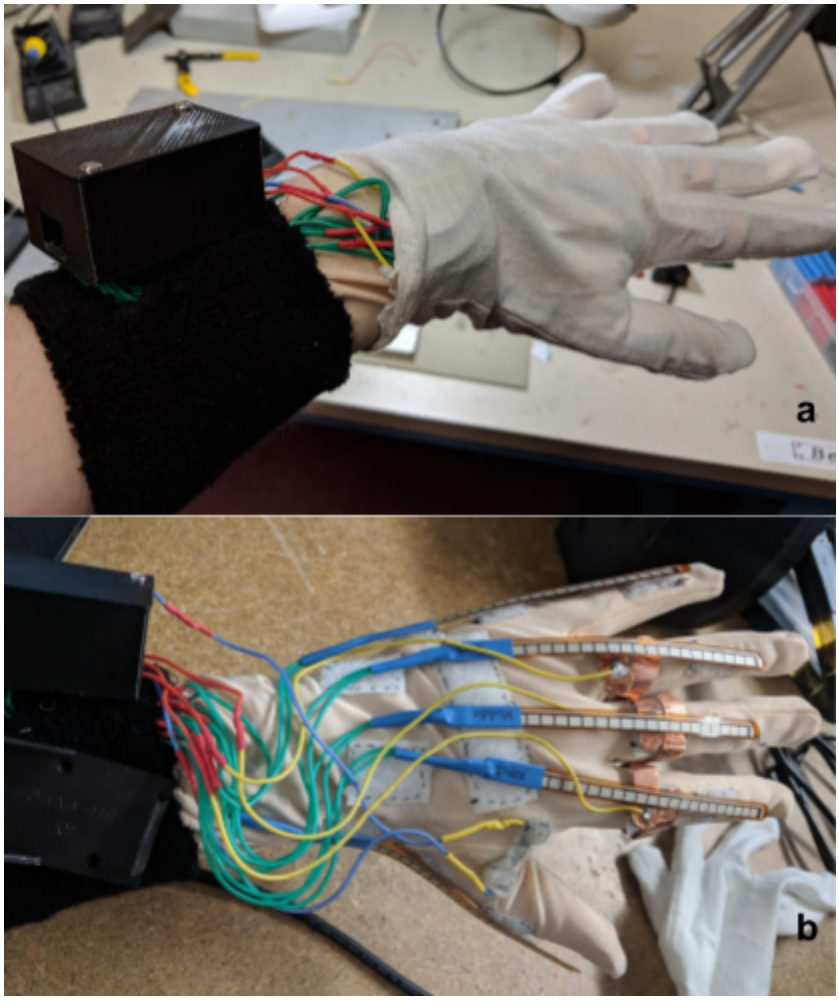

Figure 18: Finished Data Glove

4. Computer Vision

The second portion of our project is object detection via Computer Vision. We require a means of detecting object in order for the robot to reach to it autonomously. We have two main objects: objection detection/localising and orientation determination.

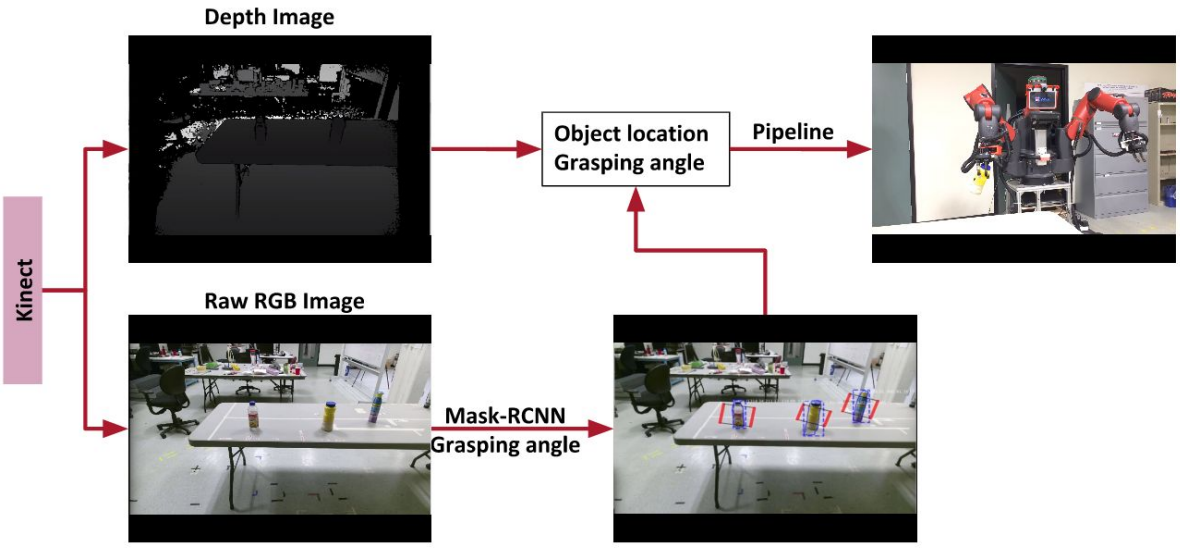

Figure 19: Workflow for Computer Vision

4.1 Object Detection

The first step required to accomplish this goal is to locate objects within an image. We examine two state of the art models that can achieve this task: You Only Look Once (YOLO[7]) and the Region Convolutional Neural Network (RCNN[8]). We found that although YOLO has a higher speed, RCNN was able to detect objects with a greater accuracy.

4.1.1 You Only Look Once (YOLO)

YOLO is currently a state-of-the-art real time object detection and localization system. Unlike other state-of-the-art neural network object detection algorithms that runs on every proposed region of interests (ROIs), the YOLO proposed a method to compute class probability for all equally divided grids (5x5 grids in [7]) in a single neural network, separately from the ROIs proposing method. Specifically, class-probabilities are predicted from the regression neural network, composed of 24 convolution layers and 2 fully-connected layers. Simultaneously, ROIs are proposed separately and later classified by class probabilities.

In this work, we used the YOLOv3 model, the current version of YOLO which improves small-object detection by exploiting residual CNN blocks. To adapt the network to our grasping detection, we adopted

the Imagenet pre-trained model and re-trained it with affordance dataset [9]. Only last three CNN blocks were re-trained to match classes in the affordance dataset. The affordance dataset contains 105 RGB-D images, each containing one object labeled as grasp, cut, scoop, contain, pound, support, or wrap-grasp.

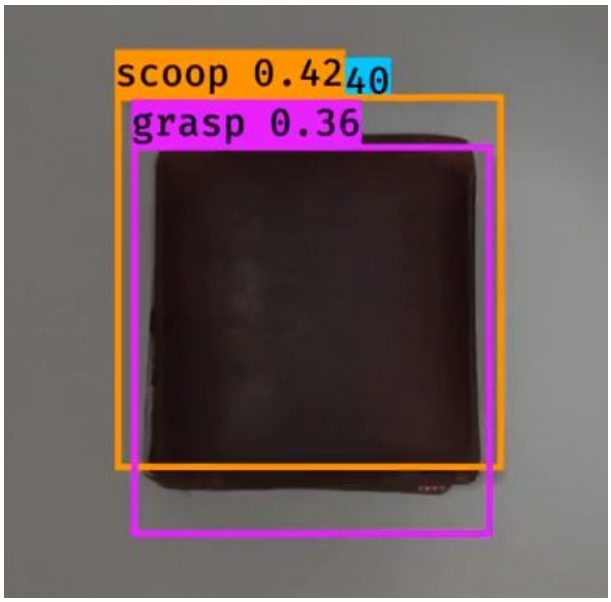

Figure 20a: Demonstration of YOLO

4.1.2 Mask Region-Based Convolutional Neural Network (Mask R-CNN)

Mask R-CNN[9] is another state-of-the-art object detection, localization, and segmentation model. The model was intuitively extended from Faster R-CNN[11] by adding capability of pixel-wise segmentation which provide more information than only the bounding box implemented in Faster R-CNN. The Mask R-CNN model comprises of two neural networks: feature extraction network and bounding box recognition network. For feature extraction networks, various pre-trained state-of-the-art image classification models were adapted by connecting bounding box recognition network prior to classification layer of the feature extraction model.

For evaluation, we used a pre-trained RESNET-50 model as a feature extraction network instead of RESNET-101 model in [10] because of limited available training data. The affordance dataset is used to train the model.

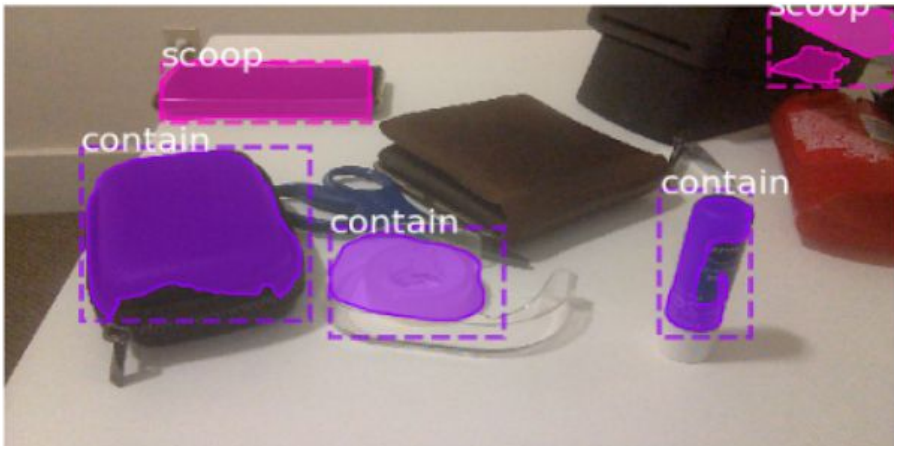

Figure 20b: Demonstration of Mask-RCNN

4.1.3 Evaluation

We validated the models using 10% of the dataset that was not used in the training process and conducted bootstrapping tests with a sample size of ten. To calculate mean and standard error of each metric, ten bootstrapping tests were performed. We tested the models by three measurements: the error of classification based on bounding box region, bounding box error in pixel and time used in a single inference.

	YOLO	Mask-RCNN
Error Comparison on classification error (± standard error)	34.2 ± 0.83 %	**18.7 ± 0.23 %**
Error Comparison on bounding box (± standard error) pixels	12.2 ± 0.176	**11.5 ± 0.112**
Runtime Comparison	**0.051s/frame**	0.42s/frame

Table 2: Comparison between YOLO and RCNN. Better performances are highlighted

From provisional experimentation, we have determined the relative effectiveness of the two models. As seen in Table 2, the lower error rate yielded by the Mask-RCNN suggests that Mask-RCNN is the more powerful classifier. However, Mask-RCNN is much slower than YOLO, by over 10x the speed.

After testing the Mask-RCNN trained with affordance dataset in the scene that had unclear background, the model detected additional object in the background and failed to distinguish the objects. To solve it, we trained the model with both affordance dataset and coco dataset, which has objects labeled out of the background.

4.2 Grasp-Angle

Using the multi-object multi-grasp model [12], we were able to integrate grasp-angle into the system. This allows us to determine the orientation required of the robot hand to pick up objects. The model is trained on a convolution neural network.

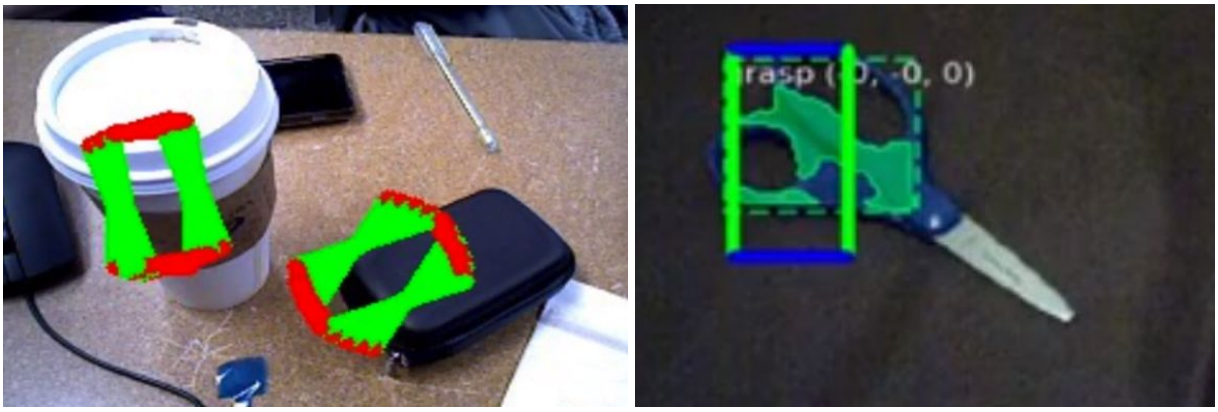

Figure 21: Multi-Grasp Detection (left) vs. Single-Grasp Detection (right)

The grasping angle is determined by four variables. The width and height of the rectangle, the angle of grasp, and the center of the rectangle. The model can be modified to detect an arbitrary number of grasp-angles. We found that sticking with smaller numbers of grasp angles reduced the noise and error.

4.3 Calculating Location

After an object is detected in the camera by aforementioned algorithms, its location in reality could be approximated relatively to the RGB-D camera using similar triangles. Given (*x,y*) is an object coordinate detected in the image with (w,h) size, *z* is a depth at a certain coordinate, and *fov* is a constant camera field-of-view.

$$F = \frac{h}{2\ tan(\frac{fov}{2})}$$

$$Z = \frac{zF}{\sqrt{x^2+y^2+F^2}}$$

$$X = Z \times \frac{x}{F}$$

$$Y = Z \times \frac{y}{F}$$

Thus, the position of object is (X,Y,Z) relatively to the camera with the same unit to z.

4.4 Cameras

We first developed object positioning algorithm using the ASUS Xtion PRO LIVE camera, which has an integrated depth field alongside the regular RGB camera. We used the OpenNI library to integrate real-time camera detection capabilities to our system.

Figure 22: The ASUS Xtion PRO LIVE camera

However, the ASUS Xtion PRO LIVE is difficult to mount on the TRINA Robot. Hence, we explored two suitable cameras: Kinect V2 and Realsense d435. From the experiment, Readsense d435 has a noisy in depth channel, comparing to Kinect V2 and ASUS Xtion PRO LIVE. Kinect was chosen as an input source to Deep Learning models. Also, Kinect has a Skeleton Joint detection feature which is useful in teleoperation.

Figure 23: The Kinect camera.

Feature	Kinect V2
Color camera	1920x1080, 30 fps
Depth camera	512x424
Depth distance(min,max)	(0.5m,4.5m)
Skeleton Joints	26

Table 3: Kinect V2 camera and its specification

5. Autonomous Grasping

We now explore the implementation of autonomous grasping. The Kinect V2 is mounted on Trina, as shown in Figure 21. It then proceeds to send RGB-D frames to the computer vision model. The Computer Vision model allows us to detect objects and calculate their position.

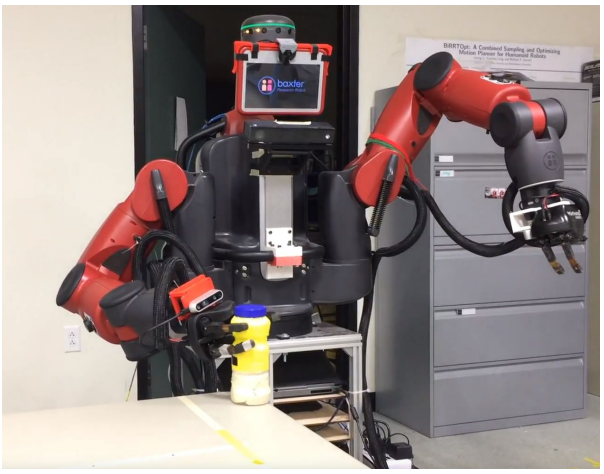

Figure 24: The Kinect is mounted on the Baxter's upper 'chest'.

5.1 Inverse Kinematics

The calculated position in cartesian space allows us to use Inverse Kinematics to calculate the joint angles of the arms necessary to move the arm out to reach the object using Baxter API to avoid any collision. With grasping angle from the deep learning model, the Trina Robot is able to successfully perform grasping and pick the object up in different reachable position and object orientation.

6. Teleoperation

In order to successfully perform learning from demonstration, we utilized the data glove developed earlier in this report to train the robot to perform reach to grasp motions. The successful grasping of objects relies on Trina being able to perform grasping naturally, in the same way as a human. To accomplish this, we mapped the skeleton joints of the human arm, returned from the Kinect, to the Trina arm which has seven joints. However, Kinect is unable to detect any joints pertaining to the wrist, and hence the grasp angle. Hence, we integrated our developed data glove to detect fingers movement for grasping an object in addition to the Kinect API. By combining the data glove with the Kinect, we were able to accurately map the movements of the operators arm and hand movements.

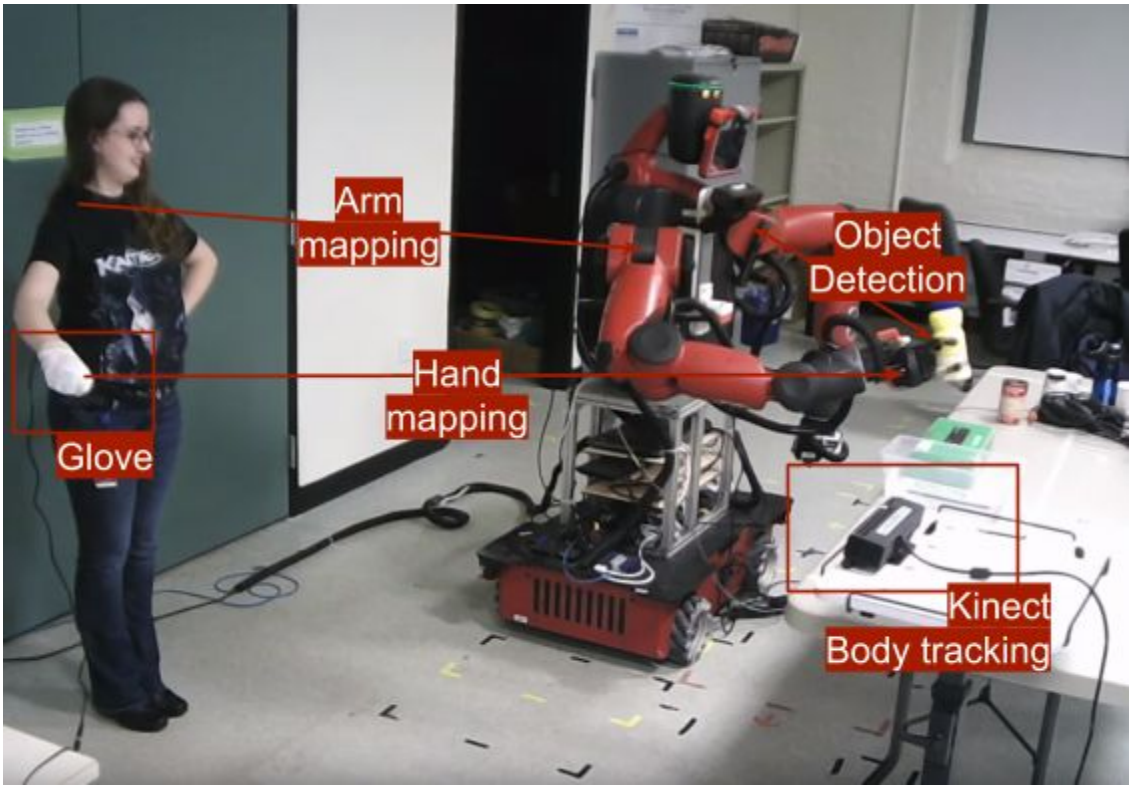

Figure 25: Teleoperation using Kinect and Glove.

In comparison with other teleoperation methods, such as the full-body mapping with Vicon, our system has a significantly lower cost. The Kinect+Glove system is a mere $300 total, where as Vicon can cost up to $10,000. Furthermore, this system is light-weight; it does not require a substantial setup area to operate. However, the main drawback of this system is from the Kinect device itself which is a low precision in the depth field, resulting in degrading of grasping accuracy.

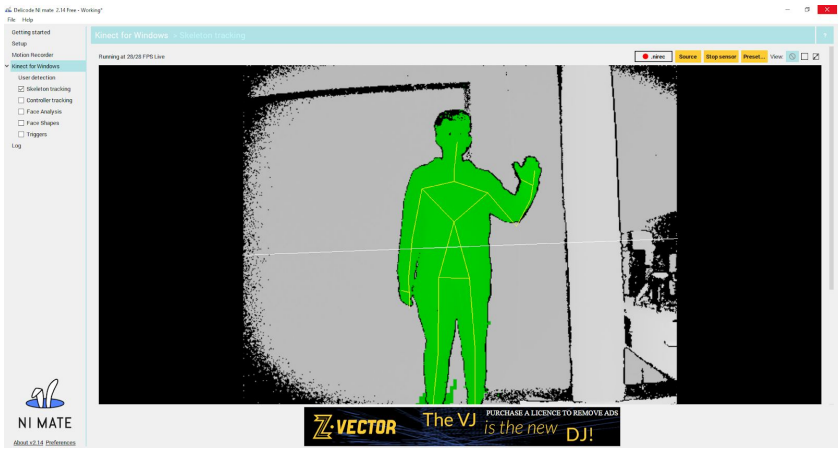

Figure 26: Kinect Body Tracking.

The demonstrations we performed using a combination of Kinect and the glove have 10 degrees of freedom in total: 7 from single Trina joints and 3 from thumb index and middle fingers. The objective of the training is the object position in which we obtain with the same method in autonomous grasping. We adopted a deterministic learning method, called probabilistic movement primitives(ProMP) [13], to mimic the movement of the human arm.

7. Learning by Demonstration

One of the goal for this project is to develop a 'teaching interface', in which we are able to teach TRINA to perform complex task autonomously without coding required. The standard approach is to pursue a reinforcement learning algorithm to achieve this. However, reinforcement learning requires a large amount of data and time to train, which for many demonstrations is not feasible. Instead, we use a different technique known as 'Learning By Demonstration', another machine learning algorithm that can teach the robot to do complex tasks with reduced data requirements.

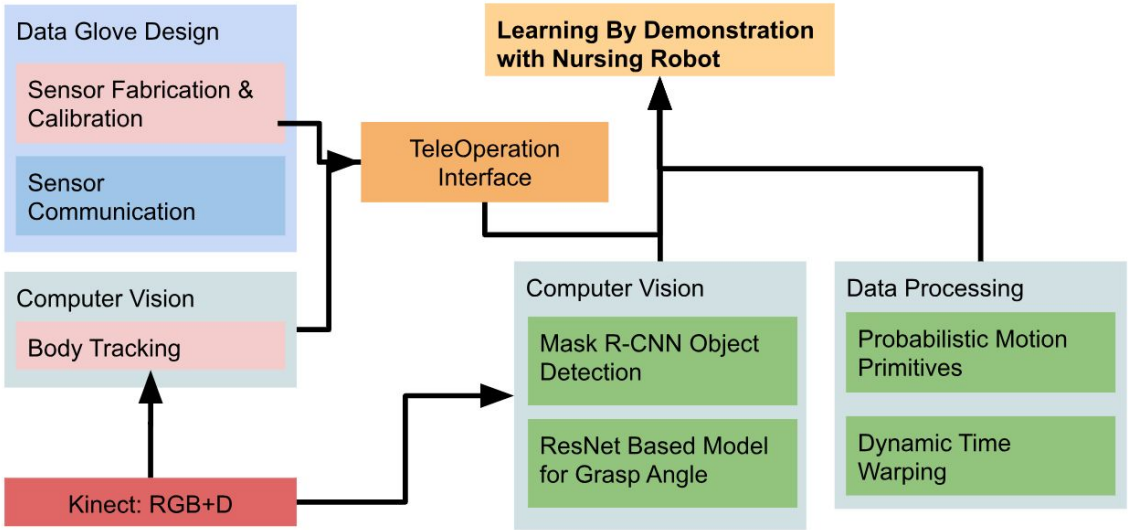

Figure 27: Framework for the different aspects of the project that add up to the Learning By Demonstration

Each individual component described above serves as the foundation for a Learning By Demonstration interface. Figure 27 summarizes the workflow of this MQP. Using the Teleoperation Interface introduced previously, we can record the movement of the human-controlled robot. This data can be used to train the robot to 'mimic' complex tasks.

7.1 Experiment

As a proof of concept, we conduct the simple task of moving a bottle into a box. We show that the robot is able to detect the object in a novel position, and execute this task with only a few demonstrations. This is achieved without the need of any hard-coding.

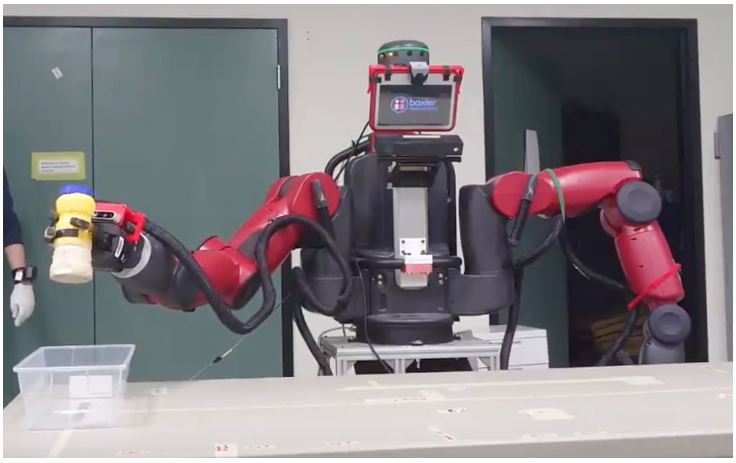

Figure 28: Teaching robot to put bottle in the bin

7.2 Probabilistic Motion Primitives (ProMP)

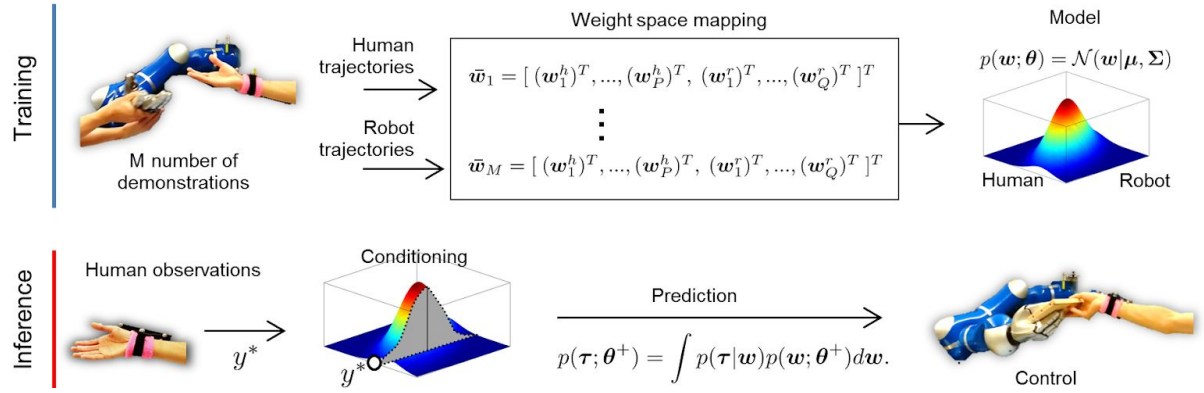

Figure 29: Illustration of the ProMP framework [14]

ProMP is a Gaussian probabilistic learning framework based that learns the mean and standard deviation from a set of training time series data and makes inferences by generalizing this data based on the number of standard deviations below or above of the given objectives. This algorithm can be used to teach the robot to perform complex tasks

7.3 Dynamic Time Warping

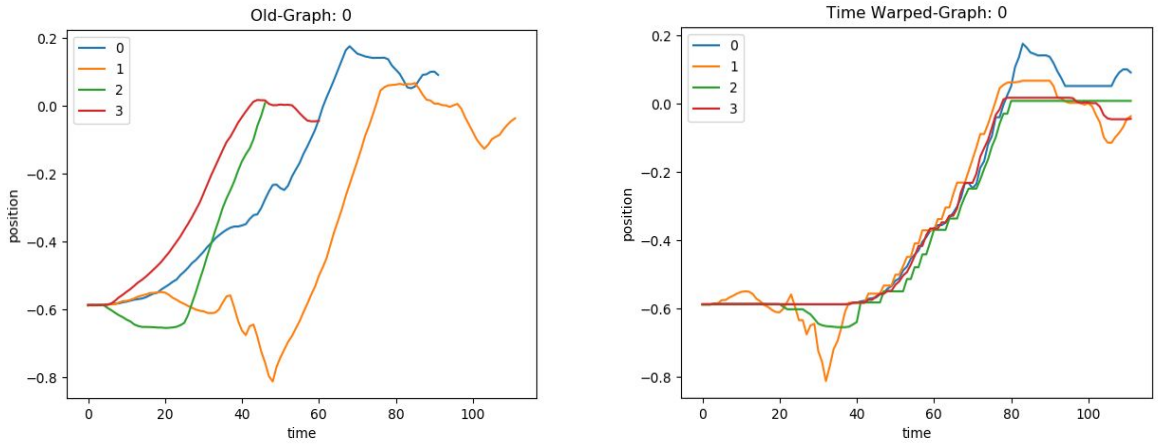

Figure 30: Dynamic time warping over Trina Joints. Left- Before time warping. Right- After time warping

A challenge we faced with the dataset is the inconsistency between actions throughout demonstrations. For example, the time interval between grabbing the bottle and putting it into a box differs in both absolute and relative time intervals. This causes unwanted variation in the joint-space data, and has led to inaccuracies when testing the ProMP model. To solve this, we introduce Dynamic Time Warping (DTW), an algorithm, an algorithm used to measure and plot the similarities between two time series. This allows us to align the data of the motion of the robot joints.

8. Future Work

While this project made many improvements to the prior MQP there were areas where we encountered different challenges. There were tasks that we either did not have time to fully address or noted that they were beyond the scope of our MQP. We have collected some suggestions for improvements on these particular tasks as well as areas that we recognized could become problematic in the future.

8.1 Further Modifying the Data Glove

Future work which would benefit this project includes utilizing a frequency counter or frequency to voltage converter to measure the readings of the capacitive sensor in the data glove more accurately. This addition would also enable the ring finger capacitor to be more accurately read as well since the method which we used to measure the frequency only worked on a single pin on the arduino and therefore could not be used to measure both capacitors at once.

Additionally, adding an IMU to the glove to enable tracking of the hand in 3D space would be a major improvement to the data glove as a kinect camera was still required to track the teleoperators arm movements. Work could also be done on the data glove to use a different wireless transmitter, such as bluetooth to give added security and more reliable data transmission.

8.2 Further Optimization in Computer Vision

In this work, two computer vision models were used in order: Mask-RCNN to detect and propose the region of interest, which is sent to the Multi-Grasp detector model for angle detection. It is possible to combine these two models into a single model with some GPU optimization, decreasing the inference time.

Object localization depends on the precision of the depth values, which are interpolated(or extrapolated) from point cloud by Kinect, leading to an artifact.

8.3 Performing Complex Tasks using Learning by Demonstration

We have designed a foundation for Learning by Demonstration using the ProMP model, trained by data recorded by the teleoperation interface. There is potential to teach the robot to perform a various number of complex tasks, and this system a systematic analysis of this system should be conducted to determine its strengths and limitations.

9. References

[1] Robotic Industries Association, "About Joseph Engelberger - Father of Robotics." https://www.robotics.org/joseph-engelberger/about.cfm. Accessed 9 Oct. 2018

[2] Eugene Demaitre, "Meet the 2017 RBR50: Top 50 Robotics Companies." https://www.roboticsbusinessreview.com/wp-content/uploads/2017/03/RBR50_Whitepaper_vFF.pdf. Accessed 11 Oct. 2018.

[3] T. Maneewarn, "Survey of social robots in Thailand," 2014 International Electrical Engineering Congress (iEECON), Chonburi, 2014, pp. 1-4. doi: 10.1109/iEECON.2014.7088527

[4] Zhi Li, Peter Moran, et al., “Development of a Tele-Nursing Mobile Manipulator for Remote Care-giving in Quarantine Areas” May 2017 http://motion.pratt.duke.edu/papers/ICRA2017-Li-TeleNursing.pdf . Accessed 5 Apr. 2019

[5] Jane Li, "Human-Inspired Robotics Lab - WPI." 28 Jun. 2017, http://users.wpi.edu/~zli11/mqp_PPU.html. Accessed 5 Apr. 2019.

[6] Walter G. Bircher, "Yale OpenHand Project - Yale University." https://www.eng.yale.edu/grablab/openhand/. Accessed 9 Oct. 2018.

[7] Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[8] Girshick, Ross, et al. "Rich feature hierarchies for accurate object detection and semantic segmentation." Proceedings of the IEEE conference on computer vision and pattern recognition. 2014.

[9] Myers, Austin, et al. "Affordance detection of tool parts from geometric features." 2015 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2015.

[10] He, Kaiming, et al. "Mask R-CNN." Proceedings of the IEEE international conference on computer vision. 2017.

[11] Girshick, Ross. "Fast r-cnn." Proceedings of the IEEE international conference on computer vision. 2015.

[12] Chu, Fu-Jen, Ruinian Xu, and Patricio A. Vela. "Real-world multiobject, multigrasp detection." IEEE Robotics and Automation Letters 3.4 (2018): 3355-3362.

[13] Paraschos, Alexandros, et al. "Probabilistic movement primitives." Advances in neural information processing systems. 2013.

[14] Maeda, Guilherme J., et al. "Probabilistic movement primitives for coordination of multiple human–robot collaborative tasks." *Autonomous Robots* 41.3 (2017): 593-612.

10. Appendix

A. Demonstrations

Demonstration videos of various tasks can be found here:
https://www.youtube.com/playlist?list=PLduj4asdOQD7emX9YTNQ8XCj_Zjehdbr0

B. Code Base for Computer Vision and Teleoperation

The Computer Vision and Teleoperation Interface code can be found at https://github.com/aditthapron/Mask-RCNN_Grasp-detector_models. Please note that you may need permission to access this code repository.

C. Code Base for Robotic Hand Teleoperation (GUI)

The code for and instructions to operate the Reflex SF robotic hand used in this MQP can be found at https://github.com/Remiles45/golden_claw_mqp . This is a GUI which we designed in order to make interfacing with and becoming familiar with the robotic hand easier.

D. Code Base for Data Glove Sensor and Communication

Integrated_Receiving_Module.ino

```
/*The below program is designed to work with the Whole_Glove program
and relay messages
 * received from the transmitting Arduino Nano to the robotic hand
via ros packages.
 *
 */

#include <SPI.h>                          //nRF communication
#include <nRF24L01.h>                     //nRF communication
#include <RF24.h>                         //nRF communication
#include <ros.h>                          //ros library
#include <std_msgs/Int16MultiArray.h>     //ros library
#include <std_msgs/MultiArrayLayout.h>
#include<std_msgs/MultiArrayDimension.h>
```

```
RF24 radio(9, 10);                // CE, CSN     Identifying which
pins to communicate with
const byte address[6] = "00001";  //address to receive the data from

//Ros Stuff
ros::NodeHandle nh;
std_msgs::Int16MultiArray glove_status_msg;
ros::Publisher glove_data("glove_data", &glove_status_msg);

bool newData = false;
int ReceivedData[5];  //Received Data

int NumSensors = 5; // Number of Sensors

void setup() {

  //Set up baud rate and open up reading pipe to receive message
  Serial.begin(9600);
  radio.begin();
  radio.openReadingPipe(0, address);
  radio.setPALevel(RF24_PA_MIN);
  radio.startListening();


//***Ok, so I'm not entirely sure what's going on here ***



  nh.initNode();
  nh.advertise(glove_data);
  glove_status_msg.layout.dim = (std_msgs::MultiArrayDimension *)
  malloc(sizeof(std_msgs::MultiArrayDimension)*2);
  glove_status_msg.layout.dim[0].label = "Glove Data";
  glove_status_msg.layout.dim[0].size = NumSensors;
  glove_status_msg.layout.dim[0].stride = 1;
  glove_status_msg.layout.data_offset = 0;
  glove_status_msg.data = (int *)malloc(sizeof(int)*8);
  glove_status_msg.data_length = NumSensors;

  glove_data.publish(&glove_status_msg );  //Attempt to setup
publisher in setup not waiting for loop.
       nh.spinOnce();                      //^^
```

```
}

void loop() {
  getData();
  showData();
  //delay(500);
}

//Below is a function that checks if there is data avaiable from the
recently opened pipe
//and if there is data, it saves the data to int_arr_msg and changes
the newData boolean to true.
void getData() {
    if ( radio.available() ) {
        radio.read(&ReceivedData, sizeof(ReceivedData));
        newData = true;
    }
}

//If the newData is true then the function showData runs, displaying
each part of the received data
//array.
void showData() {
    if (newData == true) {

       //Publish the Received Data and send ROS package to computer

       glove_status_msg.data[0] = ReceivedData[0]; //Thumb
       glove_status_msg.data[1] = ReceivedData[1]; //Index
       glove_status_msg.data[2] = ReceivedData[2]; //Middle
       glove_status_msg.data[3] = ReceivedData[3]; //Index-Middle
Capacitor Sensor
       glove_status_msg.data[4] = ReceivedData[4]; //Wool Sensor

       glove_data.publish(&glove_status_msg );
       nh.spinOnce();
       delay(200);


       // Comments below display the received data on the Serial
Monitor for testinga and
       // observational purposes.
```

```
        Serial.print("Index: ");
        Serial.println(ReceivedData[0]);
        Serial.print("Middle ");
        Serial.println(ReceivedData[1]);
        Serial.print("Thumb ");
        Serial.println(ReceivedData[2]);
        Serial.print("Cap Sensor");
        Serial.println(ReceivedData[3]);
        Serial.print("Wool Sensor ");
        Serial.println(ReceivedData[4]);


        newData = false;

    }
    //If there is no new data then the following message is displayed
on the serial monitor
    //Change to display error message in ROS too?

    Serial.println("No message");

 // The below code may publish to ros that no message was received.
Needed testing. For now, commented.
   // std_msgs::String msg;
   // std::stringstream ss;
   // ss << "No message" << count;
   // msg.data = ss.str();
}
```

Whole_Glove.ino

```
/*The below program is designed to work with the Whole_Glove program
and relay messages
 * received from the transmitting Arduino Nano to the robotic hand
via ros packages.
 *
 */

#include <SPI.h>                        //nRF communication
#include <nRF24L01.h>                   //nRF communication
#include <RF24.h>                       //nRF communication
#include <ros.h>                        //ros library
```

```
#include <std_msgs/Int16MultiArray.h>   //ros library
#include <std_msgs/MultiArrayLayout.h>
#include<std_msgs/MultiArrayDimension.h>


RF24 radio(9, 10);                // CE, CSN     Identifying which
pins to communicate with
const byte address[6] = "00001";  //address to receive the data from

//Ros Stuff
ros::NodeHandle nh;
std_msgs::Int16MultiArray glove_status_msg;
ros::Publisher glove_data("glove_data", &glove_status_msg);

bool newData = false;
int ReceivedData[5];  //Received Data

int NumSensors = 5; // Number of Sensors

void setup() {

  //Set up baud rate and open up reading pipe to receive message
  Serial.begin(9600);
  radio.begin();
  radio.openReadingPipe(0, address);
  radio.setPALevel(RF24_PA_MIN);
  radio.startListening();


//***Ok, so I'm not entirely sure what's going on here ***



  nh.initNode();
  nh.advertise(glove_data);
  glove_status_msg.layout.dim = (std_msgs::MultiArrayDimension *)
  malloc(sizeof(std_msgs::MultiArrayDimension)*2);
  glove_status_msg.layout.dim[0].label = "Glove Data";
  glove_status_msg.layout.dim[0].size = NumSensors;
  glove_status_msg.layout.dim[0].stride = 1;
  glove_status_msg.layout.data_offset = 0;
  glove_status_msg.data = (int *)malloc(sizeof(int)*8);
  glove_status_msg.data_length = NumSensors;
```

```
  glove_data.publish(&glove_status_msg );  //Attempt to setup
publisher in setup not waiting for loop.
       nh.spinOnce();                      //^^

}

void loop() {
  getData();
  showData();
  //delay(500);
}

//Below is a function that checks if there is data avaiable from the
recently opened pipe
//and if there is data, it saves the data to int_arr_msg and changes
the newData boolean to true.
void getData() {
    if ( radio.available() ) {
        radio.read(&ReceivedData, sizeof(ReceivedData));
        newData = true;
    }
}

//If the newData is true then the function showData runs, displaying
each part of the received data
//array.
void showData() {
    if (newData == true) {

       //Publish the Received Data and send ROS package to computer

       glove_status_msg.data[0] = ReceivedData[0]; //Thumb
       glove_status_msg.data[1] = ReceivedData[1]; //Index
       glove_status_msg.data[2] = ReceivedData[2]; //Middle
       glove_status_msg.data[3] = ReceivedData[3]; //Index-Middle
Capacitor Sensor
       glove_status_msg.data[4] = ReceivedData[4]; //Wool Sensor

       glove_data.publish(&glove_status_msg );
       nh.spinOnce();
       delay(200);
```

```
       // Comments below display the received data on the Serial
Monitor for testinga and
       // observational purposes.

        Serial.print("Index: ");
        Serial.println(ReceivedData[0]);
        Serial.print("Middle ");
        Serial.println(ReceivedData[1]);
        Serial.print("Thumb ");
        Serial.println(ReceivedData[2]);
        Serial.print("Cap Sensor");
        Serial.println(ReceivedData[3]);
        Serial.print("Wool Sensor ");
        Serial.println(ReceivedData[4]);


        newData = false;

    }
    //If there is no new data then the following message is displayed
on the serial monitor
    //Change to display error message in ROS too?

    Serial.println("No message");

 // The below code may publish to ros that no message was received.
Needed testing. For now, commented.
   // std_msgs::String msg;
   // std::stringstream ss;
   // ss << "No message" << count;
   // msg.data = ss.str();
}
```