Using Machine Learning to Identify and Generate 3D Objects from 2D Drawings made by the User through a low-cost Haptic Glove

Smart gloves function as wearable interfaces that enable the detection of movements and gestures, as well as provide kinesthetic and even tactile feedback. These functions are critical for creating immersive applications that heighten the realism of the experience. The paper being presented introduces a work-in-progress application of an affordable smart glove. The proposed hardware involves using augmented reality to determine spatial positioning, as well as an arduino with servo motors to provide haptic feedback to the user. With the haptic glove, the user draws a 2D image on a canvas that is then identified by a machine learning algorithm to generate a 3D object within a game environment. The goal is to utilize this glove as an interactive tool in a gaming environment, where users can solve puzzles by creating 3D objects via 2D drawings.

CCS Concepts: • **Human-centered computing → Haptic devices**; • **Applied computing → Interactive learning environments**; • **Computing methodologies → Machine learning**.

Additional Key Words and Phrases: haptics, machine learning, internet of things, augmented reality

ACM Reference Format:
. 2023. Using Machine Learning to Identify and Generate 3D Objects from 2D Drawings made by the User through a low-cost Haptic Glove. 1, 1 (May 2023), 8 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

At Connect 2021, Facebook CEO Mark Zuckerberg introduced Meta, which brought together their apps and technologies under one new company brand. Their focus was to bring metaverse to life. The research focus went towards Augmented and Virtual Reality to enhance the immersive experience. This experience included social connection, entertainment, gaming, fitness, work, education and commerce. However the important component for providing this experience was the Meta Quest VR devices which has been hugely popular ever since its release. Meta Quest is a capable device which can provide augmented reality and mixed reality experience and has a very robust hand tracking system. Even though it is one of the cheaper VR devices, not everyone is able to afford one and would like to have an alternative device for interacting within a virtual world.

This was the basis for our research where the aim was to provide hand tracking without the need of a VR device. At the same time, we wanted to provide the user with an added haptic feedback so we can give the sensation of touch as well through a haptic glove. We wanted to combine all this functionality to a meaningful application. As a result, we created a puzzle based application where the user drew 2D shapes on a virtual canvas using a virtual marker, that is held by the haptic glove. The machine learning algorithm identifies the drawn shape and produced a 3D object. Hence, in order to solve a puzzle, the user draws a 2D approximation of an object and the 3D model of the object is materialized in the environment. The user can then interact with that object to solve the puzzle. This helps in creating a low cost alternative to a VR device and also adds a haptic dimension to interactivity.

Author's address:

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2023 Association for Computing Machinery.
XXXX-XXXX/2023/5-ART $15.00
https://doi.org/10.1145/1122445.1122456

2 STATE OF RESEARCH

There were many parts of the research that we had to come across when working with the initial idea of our project. In order to create haptic feedback, we had to work with Arduino and its various sensors and motors. We had to work with several hand gesture recognition software to settle with an option that can read the finger movements better. We had to come up with a digital hand that worked like our actual hand, mimicking its movement. We had to train an ML algorithm to identify the objects drawn on canvas. We also needed to do socket programming to transfer data between python and Unity 3D as our ML algorithm was Python based while the game engine runs C#. All in all, it was a process that brought multiple technologies together into one project and it was an achievement to have them work with the least amount of bugs and errors. We will now describe this process in the coming sections.

2.1 Creating a Haptic Glove

A haptic glove, also known as a tactile glove or a force-feedback glove, is a wearable device that incorporates sensors, actuators, and haptic technology in general to provide users with a tactile and immersive experience. This glove enables users to feel and interact with virtual or remote environments by receiving feedback in the form of vibrations, pressure, or force applied to their hands. By simulating the sense of touch, haptic gloves have diverse applications ranging from virtual reality and gaming to medical training and rehabilitation. They offer a unique and intuitive way to enhance human-computer interaction, enabling users to engage with digital content in a more immersive and realistic manner.

The haptic sensors built within the glove let you feel what you are touching or holding in your hand in a VR environment while in reality there is nothing in your hand. We have implemented haptic feedback by utilizing the functionality of servo motors which we use to control the movement of the haptic modules. The haptic modules are equipped with a spring mechanism that extends and retracts, and the servo motors are responsible for allowing this mechanism to operate on a case-by-case basis. When the servos are activated, they lock the haptic modules in place, preventing the strings connected to the user's fingers from unraveling any further. This restriction allows the user to maintain a specific hand posture by limiting the free movement. when the haptic feedback is turned off, the user

regains the freedom to naturally curl their fingers into a fist or adopt any other desired hand position. The haptic glove connects to the user's personal computer through a USB cable and the users are able to see their actions on screen. Making a haptic glove requires knowledge of electronics, programming, and 3D printing. It involves designing and building a glove with micro controller and actuators, then programming the glove to provide tactile feedback.

In order to create our haptic glove we had to work with the following hardware components:

(1) **Hardware Components**;
  (a) b10k ohm Potentiometer (5x per hand)
  (b) 9g servo motors for haptics (5x per hand)
  (c) ESP32 Dev Board or Arduino or Teensy 3.2 board (1x per hand)
  (d) Retractable badge reels (for the rotary springs, 5x per hand)
  (e) Glove material (preferably cycling or golf gloves)
  (f) Velcro strap for the rigid mount
  (g) Wiring/Connectors/Breadboards (multiple sets)
  (h) Lubricant for the potentiometers (Deoxit, Triflow, machine oil etc.)
  (i) Power bank (with two outlets) or batteries (Ideally 5v 2+ amps)
(2) **3D Printed Parts**;
  (a) Guide nodes for knuckles (15x for each hand)
  (b) Tensioners (5x for each hand)
  (c) Spools (5x for each hand)
  (d) Covers (5x for each hand)
  (e) Holders (5x for each hand)
  (f) Rigid mount
  (g) End caps for fingers (5x for each hand)

The total cost for all these components came close to 16,690 Pakistani rupees which is around 58 American dollars which is quite modest for the amount of functionality it provides.

2.1.1 Assembling. First step in building the haptic glove is assembling the 'haptic module' (5 per hand) which uses the aforementioned 3D printed parts; the tensioners, spools, covers and holders along with the potentiometers (responsible for calculating finger flexion), and a servo motor. The badge reels will be responsible for the spools, the rotation of potentiometers. We start the assembling process by taking a potentiometer and fitting it through a tensioner 3D part and screwing it in with one of the nuts to set it in place. We need to make sure that the potentiometer has already been lubricated and any excess material wiped off. For haptics, we relied on strings for feedback, as they provide a simple yet effective approach to such mechanism implementation. Previous works have made use of attaching strings to the user's hands, along with using motor-controlled strings [Dworman and Harrison 2020] & [Ishii and Sato 1994] where the latter connected user's fingers to four corners of a large cubic frame. Another work made use of haptic hand grip upon a planar surface. [Yang and Zhang 2009]

We use rotary spring in the badge reels and coil it around the inside of the tensioner, tightening the spring as needed then finally putting the other end of the spring in the slit of the potentiometer that we have previously screwed in, this gives us a spring loaded potentiometer which should rewind back all the way (if it's lubricated well) when its head is turned. We then take the spool 3D part and pass the string (from the badge reel) from the inside out through a hole on the spool's side. This spool can now be fitted on to the tensioner part from earlier keeping in mind that the slit on the spool aligns with the slit on the potentiometer. This should give us a spring-loaded spool; all we need to do now is pass the same string through the hole in the cover 3D part before clamping it on the entire thing.

Finally, to complete the haptic module we need to wind the string round the spool. To do this we hold the base of the potentiometer in our left hand and rotate the head cover clockwise which should wind the string up on the inside. There's just one more thing we have to do which is to screw in one of the smaller screws. We screw this one into the small hole on top of the spool. All that's left to do now is mount this on the 3D printed holder, and we have our haptic module ready. We made 5 of them, one for each finger and this completes the haptic part of the glove.

After we are done with the 'haptic module', next we work on the base mount glove (nylon material) by gluing on the 'end caps' and the 'guide nodes' (around the knuckles) for the string to be passed through.We then add the 'rigid mount' around the wrist which essentially holds the entire haptic glove together in place. As soon as we're done with mounting the haptic modules onto the rigid mount, some modules may be needed to mount backwards, depending on the size of your hand, in order to get the string to come out at a better angle. Once done we can now move on to the final step in assembling the glove, which is wiring.

Depending on the board we choose, whether it is Arduino or ESP32, we would have to use a breadboard or connectors to connect the potentiometers as the microcontroller boards have limited pins for power and additional components are required to connect the five potentiometers (one per finger) and five servo motors. The board we have used is the Arduino Nano, the specifications for which suggest we use the 5V pins for the power to the potentiometer, since we have to extend this same pin to 5 potentiometers, we decided to use a breadboard to extend the connection to all 5 of them, and similarly to connect the potentiometers to the ground of the microcontroller.

To read the signal from the potentiometer on the Arduino Nano, we need to connect the signal pin from each potentiometer to the pins A0 through A4, from pinky finger to thumb respectively, on the Nano board. With the wiring for potentiometers now complete, we have finger tracking enabled. Moving on to the wiring for servo motors we again use the breadboard to extend both power and ground but this time the power and ground we use instead are from the power bank with a 2-2.5A output. The signal pin will be connected to the pins D2 to D6, from pinky finger to thumb respectively on the Nano. Finally, we will need to connect the batteries or the power bank to the breadboard to power the servo motors which completes the haptic glove.

With haptic glove now complete, we need to download the VCP drivers such that a virtual UART port can be accessed at the device end of the USB connection for the board, and also download the firmware for Arduino IDE which will allow us to program, calibrate

and configure our board to finally be able to start using it. Thus we provide the schematic for the circuitry below;

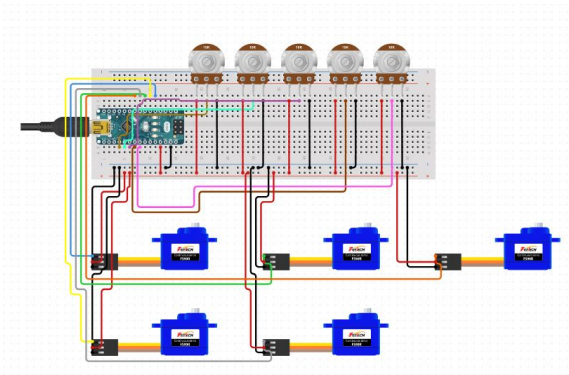

Fig. 1. Arduino Circuit Schematic

Once the users have connected their glove to their PC, they can run our Unity 3D application to experience VR in a whole new way. The glove conveys information about the bending of fingers which are then translated to a 3D hand in the application. As soon as the 3D hand makes contact with a surface, the strings pull on the fingers to limit the motion, thus giving haptic feedback.

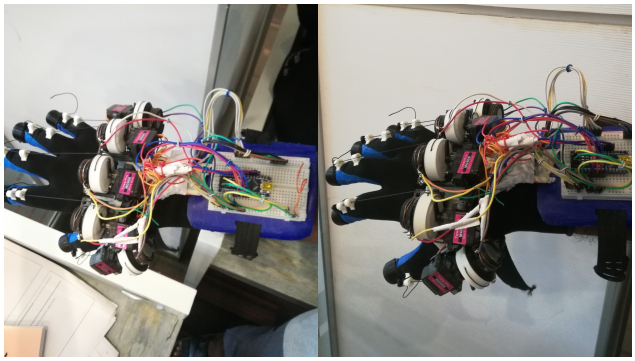

Fig. 2. Completed Haptic Glove with attached Breadboard

2.2 Handtracking

One of the challenges of creating a haptic glove is giving the glove some sort of tracking mechanism so that it can be used to get information about the user's hand movement and that information can then be utilized to create a myriad of applications.

The initial solution to this dilemma was to use a vive tracker. The tracker not only provides us with highly accurate motion tracking but it also has a low latency which is a must in such interactive systems[Caserman et al. 2019]. But, due to the price of the tracker that ranges in hundreds of dollars, the idea was scrapped. Simply because the idea for this glove was to build something that can be easily accessed by almost anyone around the world.

This leads to the creation of the second idea that required the use of a physical IMU device attached to the glove's Arduino. The IMU sensor that was experimented on in this project is the MPU9250. This approach required the use of the three common sensors Accelerometer, Gyroscope and Magnetometer. The plan was to utilize the Gyroscope for orientation tracking by simply calculating roll, pitch and yaw on the sensor. Since the sensor would be attached to the glove slightly above the wrist joint, the calculated roll, pitch and yaw would entail the orientation of the hand as well. For the tracking part the accelerometer and magnetometer were the focus. Since the acceleration of an object is the rate of change of its velocity and the velocity is just the rate of change of the object's displacement, "in theory" it is possible to calculate the distance traveled by the sensor in the three axes from a base point. The emphasis on "in theory" came into play here as the sensors are not free of noise while the margin of error for the calculations to be made needs to be low. The values given by the sensor would need to be numerically integrated twice before the desired value is obtained. This means that an error of a few decimal places can convert to tens of hundreds of units. Fortunately there was a solution to this dilemma in the form of the Kalman filter.

The Kalman filter is quite simple yet accomplishes seemingly complex tasks. To put it simply, the Kalman filter estimates the state of system by relying on external means. This means that the system doesn't need to be monitored to assume its state. The plan was to combine the outputs of the Accelerometer and calibrate them with the outputs of the Magnetometer to make sure that the noise is kept to bare minimum. There was also another option of combining the former two with a GPS sensor. The GPS sensor might not provide as precise calculations as we want but it can be used to predict the state of the hand and negate the noise to compute better results.

Unfortunately, even this approach had a flaw as it was not enough to keep the values from drifting over time. The Kalman filter works great for instantaneous results but what we wanted was continuous results and monitoring of a system which was bound to have drift over time. This drift lead to an error margin of a couple of meters which was in no way ideal for a system as closely linked to responsiveness and interaction as this glove. Even though the IMU has been successfully used for inertial navigation [Yan et al. 2018] we realized that such works make use of navigation data that was formulated using repetitive movements whereas our drawing application did not expect the user to carry out such similarities in their efforts of drawing with their hand.

This led to a new idea which revolved around computer vision and camera based tracking. Though the camera based tracking doesn't provide as much freedom since there is a requirement to be in the camera frame, it still provides a reliable and robust system for tracking. The hand tracking module from Google's mediapipe library was considered for the tracking but did not work since it is made for bare hands [Manresa-Yee et al. 2010] while this project is focused on wearing a glove. Not to mention the many 3D printed parts, motors and wires that are on top of the glove. The solution proposed to this was to create a new tracking model that tracks gloves with the same characteristics as the one used in this project for prediction. This meant that a dataset was to be prepared which would then be used to train a machine learning model that tracks objects in real time. The dataset was created by labeling the wrist of a hand wearing the glove in close to a hundred images. It was suggested

that a pertained model be used for this purpose and be retrained after locking all the layers and adding new convolutional and deep neural network layers on top. These new layers would be trained on the prepared dataset. The results were however egregious as not even a 50% accuracy was achieved and every single percentage increase came at the cost of adding tens if not hundreds of new images to the dataset.

This idea was still being worked on when we found the approach that we are using right now, that is, Marker Based Tracking. The previous approach was scrapped as this new approach was everything that the previous approach was, but better. Marker Based Tracking is quite simple. You put a marker in the camera frame that has been pre-trained upon and then track the marker in the camera frame. Since these markers have been created for the sole purpose of being tracked by their respective libraries, they can achieve an uncanny level of accuracy and robustness. The markers that were chosen for this project were "Aruco Markers". The specific marker that was picked is marker number 6 from the DICT.4X4.50 batch of aruco markers as shown below.

Fig. 3. Aruco Marker used for Tracking

Originally developed in Spain [Garrido-Jurado et al. 2014], aruco is a module in the OpenCV library of Python that has a vast dictionary of fiducial markers to be chosen. The markers come in various shapes sizes and IDs. The aruco model also provides methods that track these markers using your camera. This module has already facilitated real world use cases such as to measure the displacement of industry structures [Tocci et al. 2021]. The only change to the overall glove design was the addition of a marker being glued to the bottom end of the glove's palm facing the direction of the camera. This allowed the tracking of the user's hand. But the movement tracked was just in 2D. This was resolved through the following equation;

$$D = \frac{W \cdot F}{p} \tag{1}$$

which finds the distance D of the marker from the camera given the known width W of the marker and the focal length F of the camera being used.

The formula for focal length is given by;

$$F = \frac{D \cdot p}{W} \tag{2}$$

to calculate the focal length an aruco marker with a known width W is placed at a known distance D from the camera. The code then calculates the focal length using (2).

The perceived marker width P can be easily calculated by calculating the Euclidean distance between two of the marker corners (marker is symmetric). The corners are received by the "detectMarkers" method in the aruco library. This method returns the four corners of all the markers in the current camera frame. To compute the Euclidean distance between two corners, **linalg.norm()** function from the **numpy** library of Python is used. The linalg.norm() function calculates the Euclidean norm, which is equivalent to the Euclidean distance in this case. This approach worked more than adequately as the x, y and z values of the markers we computed and transferred to Unity via socket programming for further use. There was still a flaw to this method. Once a while the image from the camera gets distorted or the marker maybe oriented at a certain angle while moving the hand around but this distortion caused the z value to flicker momentarily. The momentary flicker caused the virtual hand to not be able to grab objects as they get thrown away by the instantaneous momentum of the hand. Upon further research it was noticed that the aruco module provides its own method of getting the x, y and z transformation matrix of the marker rather than just x and y values of the marker. The estimatePoseSingleMarkers method is utilized for this purpose. This method returns there separate lists and one of them contains the translation vector of the marker from the camera. Though there is a requirement to use this method as it relies on basis that the intrinsic parameters of the camera and its distortion coefficients are known. To find the two, the calibrateCamera method in the OpenCV library is utilized. This method requires capturing multiple images from multiple locations and multiple angles of a chess board with known number of corners and known size of each box. The chess board that was used for calibration is this project had 10x7 corners and 25mm size of each chessboard square as shown;

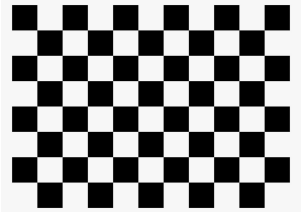

Fig. 4. Chess board used to calibrate camera

These images and data are then used to extract all the necessary parameters for the calibrateCamera method which then returns a bunch of objects. Two of these are the camera intrinsic parameter matrix and the distortion coefficient matrix. These two can then be stored in a file and used in the real time detection script with the estimatePoseSingleMarkers function to get the x, y and z values of the marker. This was the conclusive portion for the completion of the hand detection module for this project.

Fig. 5. Hand tracking with a Webcam

2.3 Fingertracking

After hand tracking was achieved, finger tracking became a much easier task. The idea was to create a hand 3D model at the 3D point that is received from the hand tracking module and use the voltage value from potentiometers to open and close the fingers of the 3D hand model. To achieve this, a serial port connection between Unity and the Arduino was established using the "Ardity" library from the unity store. This library provides the functionality of bi-directional communication between Unity and Arduino. Bi-directional communication was required due to the addition of haptics that will be discuss in a bit. Now that the connection between Unity and Arduino was established, data could be sent from Arduino to Unity. This data was obviously the values of the five potentiometers on each finger of the user's hand.

The minimum value of the potentiometer is 0 while the maximum value is 1023. Since some of the potentiometers were placed inverted and some were placed normally, the inverted potentiometers got 1023 subtracted from their read value as that would place all the potentiometers at 0. Now, when the fingers of the hand close into a fist, the value of the potentiometer increases from 0 and when the fingers open up the value decreases back to 0. These values were then mapped to a range of 0-100. The mapped values were then used to rotate the finger joints in the 3d hand model. To facilitate rotation around joints, Euler angles were used.

The fingers of the 3d hand model were created in a hierarchical structure where the first joint that is attached to the palm has two children. One, a part of the finger and the second was another joint. This new joint also had two children in the exact same manner as its parent. This last joint however, only had one child that was the remaining part of the finger. The finger rotation was handled in the following manner. Receive a value from 0-100 from the Arduino and rotate the first joint. It's transform's Euler angle is changed to the value received in the appropriate axis. What the appropriate axis means is that if the finger is to be bent forward or backwards then Y axis is changed in the transform's Euler angle and similarly for the other two axes. Now, rotate the second joint's transform's Euler angle by the received value plus the Euler angle of the first joint. This makes sure that the second joint bends faster than the first joint. Similarly for the third joint, rotate its Euler angle by the received value plus the Euler angle of the second joint. This helped in achieving a perfect way for the finger to open and close while still following the position of the hand.

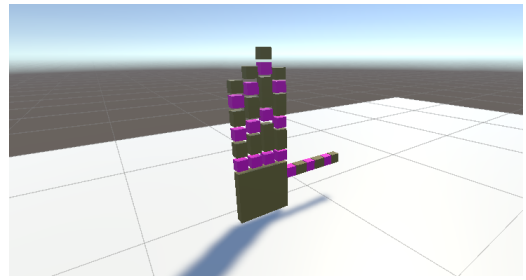

Fig. 6. The Virtual Hand

2.4 Haptic Feedback

Now that the hand tracking and finger tracking parts are taken care of, it's time to interact with objects in the Unity environment and receive haptic feedback. To deliver the feedback we attached five servo motors on each finger. The goal of these motors was to rotate their handle and block the spool from rotating further. This ensures that the string can't go any further, which directly means that your finger can not bend anymore as the string is attached to the end cap that you will be wearing on your glove's fingers.

The spool has a nut/knob sticking out of it which rotates with the spool. This knob is then intercepted by the motors handle which translates to stopping the spool from moving. The challenge is to know exactly the degree with which the motor's hand will end up intercepting the knob. To facilitate this, when the hand was attached to the motor, it was made sure that all the motors were set to angle 0 degree and when the motors were placed on the glove it was made sure that the motor's hand is resting right above the knob and did not allow for the spool to move forward. When the potentiometers are all at 0 volts the motors are all placed at 0 degree angle. Now, as the value of the potentiometer increases, the angle can also increase. First we set all the motors to 180 degree to allow the spool to move again. Then, we used the following equation to set the motor's angle whenever a "haptic event" occurs;

$$\frac{(potValues[i])}{1023.0f} \cdot 180.0f \tag{3}$$

Where potValues[i] represents the value read from potentiometer number 'i'. This equation makes sure that the handle of motor 'i' will be placed right above the knob of the spool that contains potentiometer number 'i'. This logic of the equation is quite simple as we have just mapped the value of the potentiometer from 0 to 180.

As mentioned in the finger tracking section, there is a bi-directional serial port communication setup between Unity and Arduino. In Unity, there is a script that always checks if a finger has collided with any other object in the environment (besides the 3D hand model). If a collision is detected, an array of is modified based on which finger was part of the collision. This array consists of five 0s and the modification turns the 0 at the index pertaining to the

finger/thumb into a 1. This five length array is constantly being sent to the Arduino through the communication channel and the Arduino then checks if any of the values in the array are 1. In case a 1 is received, Arduino sets the corresponding servo motor number to the resultant angle. If however a 0 was received then, the motor is set back to 180 degree so the finger can move forward again as the collision is no longer active. This completes the haptic feedback cycle and makes sure that the collisions that occur in the Unity world with the object in its environments are immediately felt by the users through haptic feedback.

2.5 Drawing Module

Our drawing module consists of basic elements that would facilitate the user to produce the required sketches for prediction such as whiteboard, eraser, and a marker [Wang and Jing 2021] the tip of which can change color.

2.5.1 Whiteboard. The virtual whiteboard serves as a canvas for users to write and draw. To create the whiteboard, a 3D object plane is created in Unity. Custom shaders are used to determine whether a specific point on the whiteboard has been painted or not. Collision detection in Unity is employed to detect the contact between the pen tip and the whiteboard, allowing the painting of specific points.

The "Whiteboard" script initializes the whiteboard by creating a texture and assigning it to the material of the whiteboard object. The texture size is specified, and collision detection is used to detect contact between the pen tip and the whiteboard, enabling painting on the whiteboard.

2.5.2 Marker. The whiteboard marker is responsible for drawing on the virtual whiteboard. It simulates the action of holding a physical pen and provides haptic feedback through the haptic glove. The marker consists of a body, a tip, and a virtual grab point to determine the pen's position. Raycasting is used to check if the pen tip touches the whiteboard. If contact is detected, the marker utilizes the material on the tip to draw on the specific part of the whiteboard's texture.

2.5.3 Eraser. The whiteboard eraser allows users to remove drawings from the virtual whiteboard. It mimics the action of rubbing an eraser on a physical whiteboard and utilizes raycasting to detect contact between the eraser's bottom and the whiteboard. Upon contact, the eraser restores the original texture and color of the whiteboard at the specific location.

2.5.4 Camera Setup. To enhance the user experience and capture the drawn images, a camera setup is implemented. The camera is positioned to capture the whiteboard area and is set to render a specific layer containing the whiteboard and marker objects. This allows for a focused view of the virtual whiteboard, enabling users to see their drawings clearly. The captured images can be saved or processed for further analysis if desired.

2.6 Object Predictor for 3D Instantiation

One of the core components of the project was the feature of being able to draw objects in two dimensions and have an machine learning model predict them into their three dimensional counterparts. The model makes use of Google's Quick! Draw dataset. The github repository for its explanation and listing of projects using it is found at [Google 2023], however on an overview it is generated using results of an online game titled "Quick! Draw" where players need to sketch out a figure relevant to the category asked for. This dataset perfectly comprehends our choice for choosing free-hand sketching as the method for object identification process. These simple sketches were chosen for their simplistic yet meaningful nature for conveyance of ideas relevant to everyday entities, as opposed to using complex and detailed images, our application being just one of the many ideas possible due to availability of its usage. A great inspiration for this is found in a work by [Xu et al. 2022]

The sketches are stored as stroke data arrays and provided as (**.npy**) files among other formats. The model itself is provided as a base tutorial guide to make use of the data, for which the colab notebook along with a good explanation is provided in a Medium article by [Alyafeai 2018]. Our own addition to the model comprises of changing relevant parameters that suit our object repository along with the addition of a preprocessing procedure for prediction suffixed with a Python socket client.

The functioning for this is shared by the model, Unity, and socket communication between Python and C# instances. The user draws a shape and rings a bell within the game to signify that the drawing is done. A bell is heard and the image is sent via socket communication to a Python server through another script. for image identification.

The python server provides the model with the required image data. Once the model has the data, it performs preprocessing on it to make it suitable for prediction which includes grayscaling, applying a thresholded color binary, resizing, conversion to numpy array, and rearranging the numpy matrix dimensions to suit the model's **(PixelDatainXaxis, PixelDatainYaxis, CategoryLabel)** structure, which here is (28, 28, 1) which suggests that the given picture dimension is 28 by 28. Once received, the model instance writes the data, preprocesses it, and then predicts the image to be one of the object categories it has been trained upon. The prediction then once again is sent as a byte string to Unity which has a socket server running in C#.

Since Unity is a single threaded application, and in order to prevent the server from stopping all of its normal procedures, the server is started in a new thread. The received byte string is used by Unity to decide which object to instantiate from a set of already designed prefabs in its asset repository. This generation from two dimensional sketches is carried out by first finding the best viewpoint or simply stated a view of the three dimensional model which then is selected as the two dimensional sketch input [Wang et al. 2015] & [Ji and Chun 2020].

2.7 The Model Itself

The numpy files containing the objects belonging to each category are read and loaded separately into variables for drawings and their labels. They are put through equal permutations so that no drawing loses its index and has randomness induced in its placement as well. This is followed by creation of training and testing variables, each of which is then normalized through a division of 255 after which class

vectors are converted to class matrices that is, addition of column headings in other words.

The model architecture follows the well-known methodology of having multiple convolutional layers followed by dense layers [Krizhevsky et al. 2012] followed by previous developments as well [Chatfield et al. 2014] & [Simonyan and Zisserman 2015]. Interleaved **MaxPooling2D** and **Convolution2D** layers all of which make use of **Rectified Linear Unit(relu)** activation have their outputs flattened before being passed to two dense layers the first of which is relu activated and the next one using softmax.

The dissimilarities between predicted and labeled classes is conveyed using **categorical crossentropy** loss function, optimized through the **Adaptive Moment Estimation(Adam)** algorithm relying on the **top K categorical accuracy** evaluation method.

3 APPLICATIONS

The glove employs functionalities that can translate into real world use cases. The most prominent of which are availability of the drawing mechanism, creating 3D counterparts to 2D sketches, movement within an area instead of a fixed hand position, and haptic feedback response.

A good area for the glove's usage is that of education. We only recently recovered from a global pandemic that forced everyone to switch to a virtual space wherever possible. One specific section that faced the brunt of this change was that of education itself. Students and teachers had to resort to online modes for their regular classes, and to experience the absence of engaging learning activities for the most part. The glove can resolve such cases where classes can be made much more engaging both for students and teachers where instead of using the conventional tools of physical makers and boards they can now make use of virtual ones. This is specially useful for elementary sections of schooling where young students can already start practicing their drawing skills, and witness their sketches being converted to 3D objects. It would greatly improve learning due to the improved interactive nature of the entire experience.

Another branch of application for the glove is that of training environments in virtual reality. Our technology now allows prospective practitioners in numerous fields to go through life-like training simulations in which they face problems related to their work and resolve them without having to worry about facing real world consequences. The haptic feedback here proves crucial to creating that very environment. We can take the example of completing basic tasks in an operating room where the user is working as a surgeon. This requires constantly dealing with the required tools available and thus the action of grabbing and holding them provides a more immersive experience.

There can also be scenarios such as driving simulations, aircraft cockpit simulation, mechanic simulation, as the gloves haptic response provides the same degree of reality to the experience.

The biggest platform for the glove's functionality is the Metaverse. Having a haptic glove provides a more natural experience than VR controllers as the users can now feel the physical response when interacting with the virtual world.

4 FUTURE WORK

The glove like any other project did begin with abstract ideals which have then morphed into an actual working constructions with an applicable output.

The hand tracking portion served to be the most challenging portion of the development effort. There were many methods available however, we had to leave out many due to certain obstacles. The hand tracking has been implemented but does not provide the total freedom of direction. The fingers do move up and down, the hand is precise for the x and y axes, although a little hazy for the z axis. It however does not rotate clockwise or counter clockwise and is required to face the camera at all times. Using multiple cameras may provide more freedom and accuracy, but it will come at a cost of being more expensive and dependency on more hardware. This is a future area of research that will be looked into in future.

The drawing tools have been created but there is some polish required as during the process of drawing, the marker tends to push into the board object if moved in quickly enough, otherwise our implementation for the board and marker collision detection works fine with some bumpy behaviour.

We set for the goal of good physics simulation for the application where the hand object when controlled by the user is used to interact with virtual objects just like in real world. This goal was achieved with various success rate. For grabbing and holding large objects this method worked fine however, there was a lack of precision and delicate control of other in-game resources was not very accurate. The haptic feedback works although not being extremely efficient. It does tend to get affected by loose wiring, and slipping connections if such issues are present. This is mainly because this is a very rough prototype and it can be improved by using industrial level construction processes.

Thus coming to our future prospects for the development, it includes switching to a much more minimalist design. This can comprise of, for example plastic cog like constructs, akin to used in zip ties, for using into our haptic feedback instead of strings. Other materials may be looked into and a lot of experimentation is required before we come up with a robust and cheap way of doing the same.

Another upcoming goal would be the inclusion of a large repository of 3D objects, as we have tested with a small one for now. Those objects would comprise of interesting and complex entities along with simple constructs the like of which we are currently working with.

5 CONCLUSION

Since the beginning of this decade, there have been many events that have pushed the direction of research towards virtual communication and interaction. With Covid-19, social distancing became necessary and the dependency on online communication has increased. During the same time VR devices have improved significantly in performance and the announcement of Metaverse has brought them into the mainstream. However, the cost associated with these devices makes it hard for everyone to acquire it.

With that concept in mind, we came up with the idea of a low cost haptic glove which can provide virtual interaction with objects

and improve on it by providing realistic feedback only by using a webcam. As a proof of concept, we created an educational application which can be used for teaching and also uses machine learning to decipher what a person has drawn on canvas and instantiates a 3D object in the game world.

Our research proves that in the coming world, with the technologies like VR, AR, IoT and computer vision, we can create low cost alternatives of the general VR devices that are quite costly. This allows everyone to take part in the Metaverse with only a partly investment as opposed to the VR devices in general use. With a glove, we have a more natural interaction with the environment and haptic feedback gives a better illusion of interaction than VR controllers.

REFERENCES

Zaid Alyafeai. 2018. Train a model in tf.keras with Colab, and run it in the browser with TensorFlow.js. https://medium.com/tensorflow/train-on-google-colab-and-run-on-the-browser-a-case-study-8a45f9b1474e

Polona Caserman, Augusto Garcia-Agundez, Robert Konrad, Stefan Göbel, and Ralf Steinmetz. 2019. Real-time body tracking in virtual reality using a Vive tracker. *Virtual Reality* 23 (2019), 155–168.

Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. 2014. Return of the devil in the details: Delving deep into convolutional nets. *arXiv preprint arXiv:1405.3531* (2014).

Cathy Fang Yang Zhang Matthew Dworman and Chris Harrison. 2020. Wireality: Enabling Complex Tangible Geometries in Virtual Reality with Worn Multi-String Haptics. (2020).

Sergio Garrido-Jurado, Rafael Muñoz-Salinas, Francisco José Madrid-Cuevas, and Manuel Jesús Marín-Jiménez. 2014. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition* 47, 6 (2014), 2280–2292.

Google. 2023. The Quick, Draw! Dataset. https://github.com/googlecreativelab/quickdraw-dataset.

Masahiro Ishii and Makoto Sato. 1994. A 3D spatial interface device using tensed strings. *Presence: Teleoperators & Virtual Environments* 3, 1 (1994), 81–86.

Myunggeun Ji and Junchul Chun. 2020. A Sketch-based 3D Object Retrieval Approach for Augmented Reality Models Using Deep Learning. *Journal of Internet Computing and Services* 21, 1 (2020), 33–43.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger (Eds.), Vol. 25. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

Cristina Manresa-Yee, Javier Varona, Ramon Mas, and Francisco J Perales. 2010. Hand tracking and gesture recognition for human-computer interaction. In *Progress in Computer Vision and Image Analysis*. World Scientific, 401–412.

PimDeWitte. 2022. GitHub - PimDeWitte/UnityMainThreadDispatcher: A simple, thread-safe way of executing actions (Such as UI manipulations) on the Unity Main Thread — github.com. https://github.com/PimDeWitte/UnityMainThreadDispatcher.

Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556 [cs.CV]

Tommaso Tocci, Lorenzo Capponi, and Gianluca Rossi. 2021. ArUco marker-based displacement measurement technique: uncertainty analysis. *Engineering Research Express* 3, 3 (2021), 035032.

Fang Wang, Le Kang, and Yi Li. 2015. Sketch-based 3d shape retrieval using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1875–1883.

Jiangkun Wang and Lei Jing. 2021. Operation Efficiency Study on a New Cooperative VR Whiteboard System. In *HCI International 2021-Posters: 23rd HCI International Conference, HCII 2021, Virtual Event, July 24–29, 2021, Proceedings, Part III 23*. Springer, 629–636.

Peng Xu, Timothy M Hospedales, Qiyue Yin, Yi-Zhe Song, Tao Xiang, and Liang Wang. 2022. Deep learning for free-hand sketch: A survey. *IEEE transactions on pattern analysis and machine intelligence* 45, 1 (2022), 285–312.

Hang Yan, Qi Shan, and Yasutaka Furukawa. 2018. RIDI: Robust IMU double integration. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 621–636.

Yi Yang and Yuru Zhang. 2009. A new cable-driven haptic device for integrating kinesthetic and cutaneous display. In *2009 ASME/IFToMM International Conference on Reconfigurable Mechanisms and Robots*. IEEE, 386–391.