Directing Shutter with Subject Gestures

Justin Chang[*]
Department of Computer Science
Building Interactive Machines
Yale University
New Haven, CT, USA
justin.chang@yale.edu

Connor Lee[*]
Department of Computer Science
Building Interactive Machines
Yale University
New Haven, CT, USA
connor.lee@yale.edu

Anna Zhang[*]
Department of Computer Science
Building Interactive Machines
Yale University
New Haven, CT, USA
anna.zhang@yale.edu

ABSTRACT

Taking a picture of yourself is a special experience: the subject and the photographer are one and the same. Our project aims to empower users to assume control of a robot photographer to capture their ideal portrait. Shutter has the ability to help take photos of the user, but there are some key flaws in its abilities. If the user is too tall or too far to one side, Shutter will take the photo regardless. Ideally, we want to control the angle of Shutter's photo. Therefore, our project works to direct Shutter to move as a photographer by recognizing the user's hand gestures.

CCS CONCEPTS

• **Human-centered Computing** → **Gestural input**; • **Computing Methodologies** → *Machine Learning*; • **Computer systems organization** → Robotics.

KEYWORDS

gesture recognition, neural networks, interactive machines

ACM Reference Format:
Justin Chang, Connor Lee, and Anna Zhang. 2022. Directing Shutter with Subject Gestures. In *Proceedings of Building Interative Machines (BIM '22).* ACM, New York, NY, USA, 5 pages.

1 INTRODUCTION

Across varying levels of expertise, individuals have a range of preferences for the framing of photos. Each person has a different concept of what they deem a "good" photo. Thus, we want to give the user control of that framing aspect. We hope to allow the user to direct Shutter on how to reframe the camera to their liking as they take a photo. Shutter will therefore avoid taking a photo that is completely out of frame or cuts people out. (Alternatively, the user may wish to crop out part of someone as a practical joke.)

Professional photographers frequently use gestures in directing subjects, from encouraging rotations ("tilt your head to the left!" accompanied by rotating their hand) to subtle translations ("move your hand a little down!" accompanied by a patting motion). This form of user interaction is a quintessential part of photography. These kinds of human-computer interactions can help to create an engaging, human-like experience for whoever may be using Shutter to take photos.

The primary goal of our project is to have Shutter recognize gestures that can be used to reframe the photo. For instance, if the user points up, Shutter would move the frame of the camera upwards. The same could be said for left, right, and down, giving the user more control over the outcome of their picture. While Shutter is readjusting the frame, Shutter's camera viewpoint can be shown on a display for the user to see. Finally, when the user is satisfied with the displayed camera angle and is ready to take their photo, they can give a thumbs-up gesture, which Shutter will recognize as a command to snap the photo. The user can also use their open hand to have Shutter look towards their body using the Kinect data body tracking. Overall, we believe our project will help to improve the user experience for people who interact with the Shutter robot and use it to take photos.

2 RELATED WORK

We were inspired by the work of Adamson et al., which used simulated humor to improve photographer-subject relationships with their robotic photographer [2]. We aim to build upon previous efforts to advance the robot photographer Shutter and design a programmatic pipeline for directing Shutter with subject gestures: the hand motions of the individual(s) being viewed by the robot.

Robot photographers hold tremendous promise in enhancing the quality, precision, and artistic appeal of images [6], but many current approaches take minimal user input. In examples from Oleary and Zhang, robotics have been used to help assist videographers in their artistic process, but in an isolated environment, the robot cannot be asked to take photographs for a user. Existing models either choose from a list of fixed poses[9], maximize an aesthetic metric learned from a limited data set [3], or balance facial orientation with perceived emotions [5]. Therefore, the applicability of robot photographers to interactive subject-directed photography has yet to be extensively explored.

Other interesting and relevant work looks at ways we can make human-robot initial interactions more user-friendly and natural, like in Reddy et al. [7]. These researchers explore the ways that humans naturally interact with the robots and machines in front of them. In our research, this information is helpful to our understanding of how users tend to initially approach Shutter.

In our project, we combine these facets of prior work to incorporate smooth, seamless user interaction with the assistive nature of a robotic photographer who can use natural hand gestures to support the user's intentions.

[*]All authors contributed equally to this research.

Table 1: Gesture Commands

Gesture	Action
Hand Open	Turn Shutter toward user
Thumbs-Up	Take snapshot
Point Up	Move Shutter up
Point Down	Move Shutter down
Point Left	Move Shutter left
Point Right	Move Shutter right

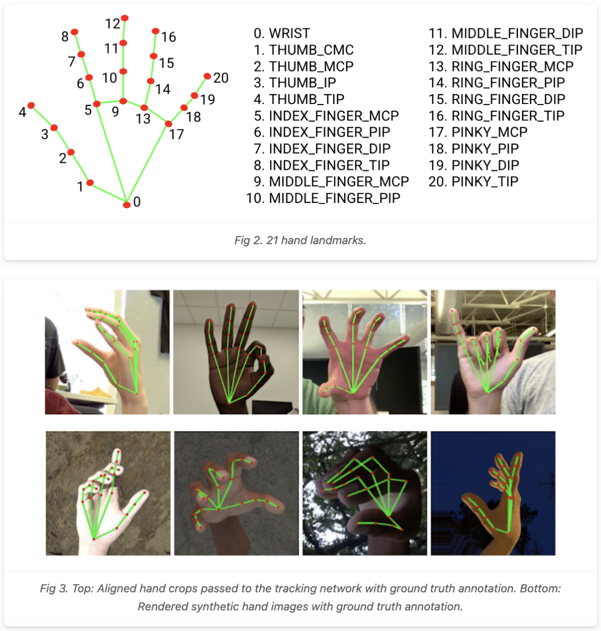


Fig 2. 21 hand landmarks.

Fig 3. Top: Aligned hand crops passed to the tracking network with ground truth annotation. Bottom: Rendered synthetic hand images with ground truth annotation.

Figure 1: MediaPipe Hands API

3 METHOD AND TECHNICAL APPROACH

To achieve our goal of having an interactive photography experience with Shutter, we utilized the hand gesture commands listed in Table 1. To ensure a comprehensive view of the subject, we take in input from the following sources: Shutter (including joint-state and robot model information), the Azure Kinect (including color, depth, and body tracking information), and the D435i RealSense camera (including color and depth information). The corresponding Shutter topics are:

- Shutter: /joint_states
- RealSense: /camera/color/image_raw
- RealSense: /camera/depth/image_rect_raw
- Kinect: /rgb/image_raw
- Kinect: /depth/image_raw
- Kinect: /body_tracking_data

In order to perform hand tracking, we utilized the MediaPipe library, which has an API for hand tracking [1]. It primarily utilizes the Kinect camera image input to isolate hand landmark data. This data is an array of coordinates where each index is associated with a different part of the hand, like the wrist or index fingertip. In Figure 1, you can see all of these hand landmark index references as well as a few examples of how that hand tracking looks in the images. This library works in real time, which helps us create a more interactive and responsive user experience. As shown in Figure 1, the data points are all quite accurate and representative of the overall user hand position in the image [8].

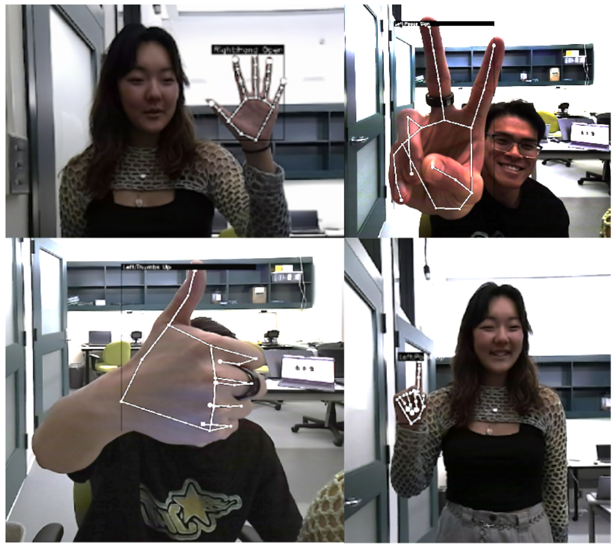


Figure 2: Sample Training Data Images

The hand landmark data from MediaPipe is then pushed through a Multilayer Perceptron model that classifies the hand gesture inspired by the GitHub repository from @kinivi [4]. We have included some examples of the training data that we provided in Figure 2 with various hand landmark data and user gestures.

The classified hand gestures are subsequently mapped to corresponding actions for Shutter. We integrate these data inputs and output a set of new joint positions for Shutter based on the mapped gesture. This implies publishing to the /joint_group_controller/command topic, which yields downstream movement through the shutter_bringup package.

Since each sensor updates at different rates, the integration of these inputs posed substantial challenges. Our solution entailed creating a HandTrackingNode that subscribes to each of the input topics. For each topic, a callback function assigns new messages to a corresponding variable member of the HandTrackingNode.

Initially, we had each callback function call a common update function which would take the node's variable members as input and publish joint movements as output. Since our sensors published new messages far faster than the updates could be completed, a backlog of update function calls accrued. This causes Shutter to lag further and further behind, taking minutes to respond after only a few seconds of run time. To limit updates to a certain frequency, we revised the callbacks to request updates through an intermediary fixed_rate_update function. This function only calls downstream updates after a threshold period of time, resolving the issue.

To prevent multiple threads/processes from performing concurrently, which yields segmentation faults in MediaPipe, we place a fairly simple lock on the updating process. The *updating* Boolean

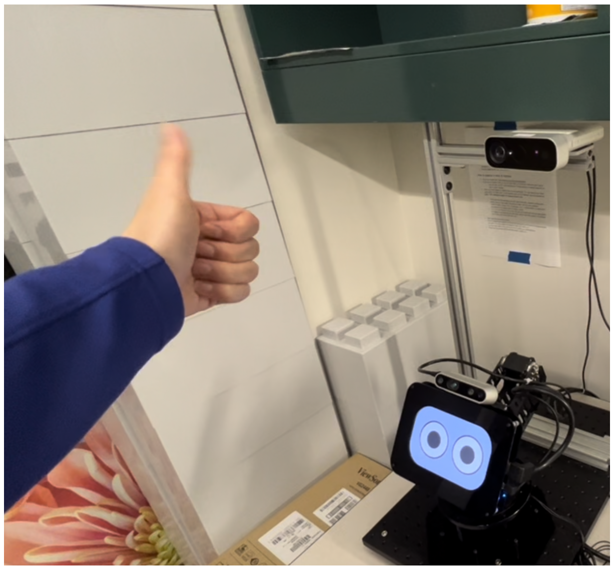


Figure 3: Thumbs Up Hand Gesture

is set to true at the beginning of the 'update' function and false at the end of the 'update' function. We then revised fixed_rate_update to avoid updating if *updating* is set to true. These design choices yielded stable Shutter performance — a necessary prerequisite for any movement.

Our next challenge was ensuring safe Shutter movements and implementing a safe idle position. We began by retrieving Shutter's joint limits from URDF and restricting any published joint angles to the provided values. However, Shutter appeared to be capable of touching the Azure Kinect apparatus even under these restrictions. Therefore, we tested various empirical values and further restricted Shutter's range of joint motion. We also created a safe position for Shutter to reside when there are no gestures detected for more than 5 seconds. Shutter moves to the resting position to decrease the risk of falling while running.

We further refined the safety features of the program by restricting the joint limits. If the robot remains inactive for a long period of time, it is moved to a safe position with the coordinates (a, b, c, d), which our joint clamper converts to (a', b', c', d'). Additionally, we have ensured that no raw joint positions are published through the use of self.move_joints_raw; all joint positions are now moved through self.move_joints. When the program shuts down, cleanup now includes moving the robot to a safe position.

Additionally, we achieve the launch of our system in a single command with the use of the terminal multiplexer *tmux*. The RealSense camera may experience issues, which we find can be resolved by disconnecting the camera, waiting 10 seconds, closing all open terminals, opening a fresh terminal, and trying again. The controllers are turned on through a system command in Shutter. The launch file also passes the current path of the hand-gesture-mediapipe-main folder to the program, enabling hand_tracking.py to use *os.chdir* to set its path appropriately and subsequently open the model.

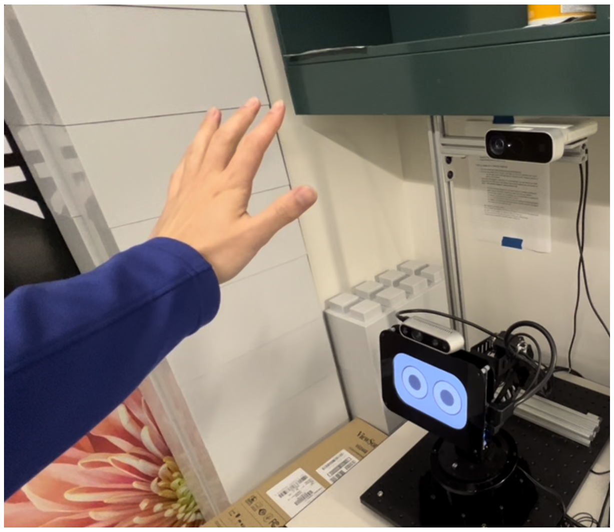


Figure 4: Open Hand Hand Gesture

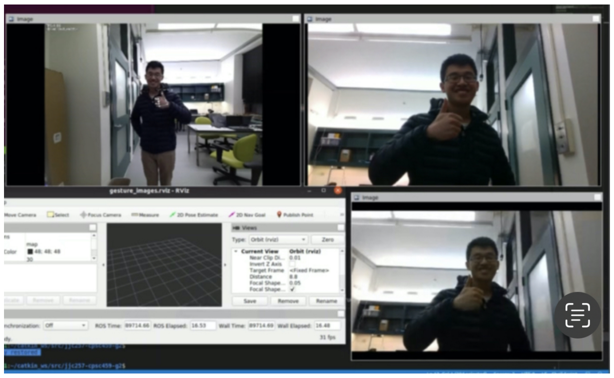


Figure 5: Screenshot with the RViz Displays After Project Launch

4 TRAINING AND TESTING

In our first attempt with the gesture classification model, we trained around 10,000 image data inputs from a laptop camera. The model performed quite well, but from our demo and these brief tests, a few concerns arose in Shutter's behavior. Shutter had difficulty with the following:

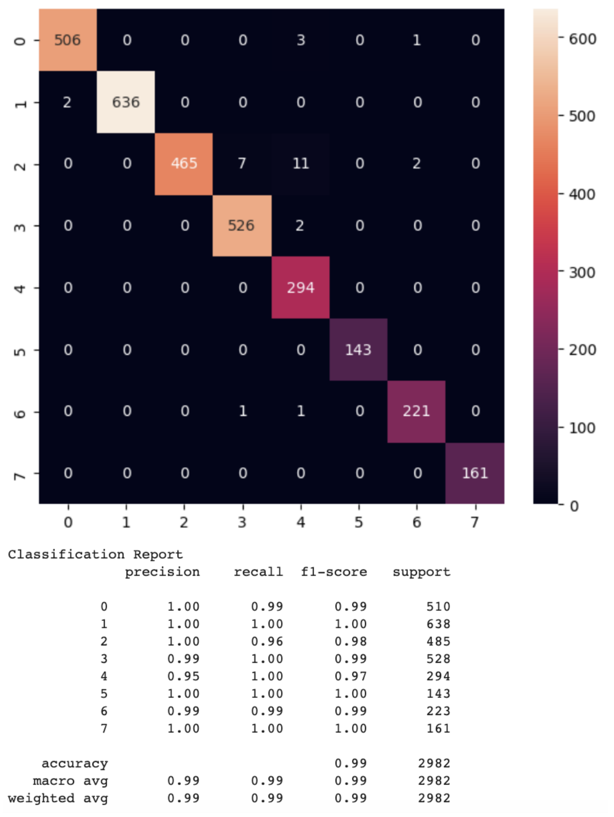
- Identifying/classifying a hand gesture when there were multiple people or hands in the frame
- Distinguishing the user's hand when they were wearing a lighter-colored shirt or top.
- Differentiating thumbs up from a pointing hand that has the thumb out
- Recognizing a left-hand pointing left or a right-hand pointing right.

Noting some of the flaws in our initial gesture classification, we retrained the robot's gesture classification model with the Kinect image data and the specific parameters outlined in Table 2 to make the model more robust.

Table 2: New Training Set Parameters

Parameter	Variations
Distance from Kinect	1 meter 2 meters 2+ meters
Handedness	Left hand Right hand
User	Chang Lee Zhang
Gesture	Open hand Thumbs up Point up Point down Point left Point right

With each training condition set, we utilized 15 seconds of training time in which the trainer moved their hand gesture around the camera frame and changed the angle of their hand to accommodate for different potential user hand positions. In total, our training data consists of 32, 253 images from three individuals. Figure 7 shows the new model's confusion matrix and classification report, which we can compare to the previous model in Figure 6. The confusion matrix depicts excellent predictions of true positives and true negatives with relatively low false positives and false negatives. The classification report indicates a 96% overall score for the model's precision, recall, and f1-score. While the old model had higher scores in the classification report, it contained less data which made it less effective than our new model. The first and third inputs of the new confusion matrix and classification report were omitted because they were unused gestures that had no training data.



```
Classification Report
              precision    recall  f1-score   support

           0       1.00      0.99      0.99       510
           1       1.00      1.00      1.00       638
           2       1.00      0.96      0.98       485
           3       0.99      1.00      0.99       528
           4       0.95      1.00      0.97       294
           5       1.00      1.00      1.00       143
           6       0.99      0.99      0.99       223
           7       1.00      1.00      1.00       161

    accuracy                           0.99      2982
   macro avg       0.99      0.99      0.99      2982
weighted avg       0.99      0.99      0.99      2982
```

Figure 6: Confusion Matrix from MLP Model Before Retraining

5 EXPERIMENTS AND RESULTS

We are successful in identifying gestural inputs from a single person to make the robot perform basic movements and take pictures. A live demonstration of our project can be found at:

https://github.com/justinchang1124/cpsc459-g2.

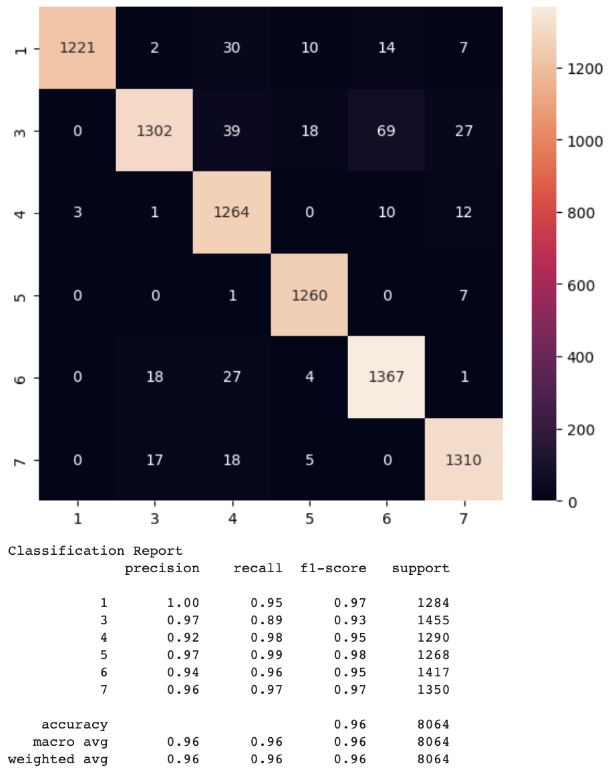
In Figure 5, we have a sample screenshot of how the project looks when you initially launch the hand tracking launch file. The result is that a couple of RViz displays appear on the screen that provides helpful information to the user. The top left image is from the Kinect, which is where our program performs hand tracking with the MediaPipe library. This image also has debugging content displayed on top to be able to see the locations of all of the hand landmarks and the classified gesture. The image in the bottom right corner is the RealSense camera image. This is necessary for the user to see the angle of the photograph that Shutter will take. Finally, once the user is satisfied with the image in the RViz display, they can use the thumbs-up gesture to signal to Shutter to capture their photograph. Shutter will then take a picture from the RealSense camera and display it in the frame on the top right corner of the screen as seen in Figure 5.

To evaluate our project's effectiveness, we asked several participants to test our project. Participants gave Shutter various gestures to test the accuracy of Shutter's recognition model behavior. In these experiments, we evaluated Shutter's response varying certain independent variables such as user distances, gender, and interferences (i.e. background people).

We tested Shutter's hand recognition at three different distances from the Kinect Azure camera: 1 meter, 2 meters, or 2+ meters away. At 1 meter, Shutter recognized user hands about 97% of the time. At 2 meters, Shutter recognized user hands about 40% of the time. Anything above 2 meters was not picked up by MediaPipe.

During our tests, we also timed Shutter's responses to gestures. We noticed that Shutter would typically respond in less than a second. However, Shutter would occasionally require longer to decide which gesture was being shown, resulting in an average response time of 2 seconds.

This latency arose from requiring Shutter to see, for instance, 20 updates where the user was pointing left in a row before moving to the left. However, occasional glitches in the MediaPipe hand tracking detection or in our classification model would delay Shutter's reaction. Previously, the robot would not move unless the last 20



	1	3	4	5	6	7
1	1221	2	30	10	14	7
3	0	1302	39	18	69	27
4	3	1	1264	0	10	12
5	0	0	1	1260	0	7
6	0	18	27	4	1367	1
7	0	17	18	5	0	1310

```
Classification Report
              precision    recall  f1-score   support

           1       1.00      0.95      0.97      1284
           3       0.97      0.89      0.93      1455
           4       0.92      0.98      0.95      1290
           5       0.97      0.99      0.98      1268
           6       0.94      0.96      0.95      1417
           7       0.96      0.97      0.97      1350

    accuracy                           0.96      8064
   macro avg       0.96      0.96      0.96      8064
weighted avg       0.96      0.96      0.96      8064
```

Figure 7: Confusion Matrix from MLP Model After Retraining

gestures led to a unanimous outcome (so the gesture recognition system would have to agree for 20 consecutive iterations). We refined the system so that Shutter instead stores the last 20 gesture IDs and requires a supermajority of 80% of a single gesture before proceeding with the corresponding action. The number of IDs sampled and the fraction required to achieve a successful result are adjustable as parameters.

Previously, our program would fail to function if the robot began in an emergency-braked position. Now, it will wait for the controller service to activate before beginning. However, if the emergency brake is activated after the program begins running, the program will not resume until freshly run again.

When a participant tried to use two hands to control Shutter, Shutter arbitrarily chose one of the hands to track. Once a hand was recognized, our system remained consistent in tracking that hand until it went out of frame or got too far from the Kinect.

6 CONCLUSION AND FUTURE WORK

Future work for this project would improve the flow of the interactive system. An additional feature could be to publish the RealSense camera live stream to Shutter's display, enabling the user to view the camera image straight on.

One of the biggest issues that Shutter currently faces is not being able to operate smoothly when there are too many people in the camera frame. In the future, this could be improved by connecting the Kinect body tracking data with the MediaPipe hand tracking data. Connecting the hand with a user's body achieves several distinct goals. First, you can isolate one user to give Shutter instructions for photography movement. Also, Shutter currently looks toward a user when they present an open hand. However, if we were to more discretely connect the Kinect body tracking data, we could ensure that Shutter looks at the user who has their hand open instead of just the first body in the Kinect body tracking data.

Another potential feature would be to implement a snapshot initiation sequence that starts a countdown when the user presents a thumbs-up. This countdown will let the user and other people in the frame prepare and pose for an upcoming picture. Additionally, the system's robustness in identifying hands could be improved by enhancing the input camera data or integrating other human landmark data.

One potential problem with our project is that people could covertly teleoperate Shutter to take photos of people without their consent. However, this issue can be mitigated by only allowing Shutter to take pictures in photo booths or public areas.

We hope our added functionality to Shutter makes it easier for people to engage with Shutter. We also hope that our project inspires others to advance Shutter's interactivity.

REFERENCES

[1] 2022. Google MediaPipe Documentation: Hands API Sample Solutions. (Nov 2022). Retrieved Nov 8, 2022 from https://google.github.io/mediapipe/solutions/hands

[2] Timothy Adamson. 2020. Designing social interactions with a humorous robot photographer. (2020). Retrieved Nov 8, 2022 from https://dl.acm.org/doi/10.1145/3319502.3374809

[3] Hadi AlZayer et al. 2021. AutoPhoto: Aesthetic Photo Capture Using Reinforcement Learning. (Sept 2021). Retrieved Nov 8, 2022 from http://arxiv.org/abs/2109.09923

[4] @kinivi on Github. 2021. Github Repository with Sample for Hand Gesture Recognition Using the MediaPipe API. (Jan 2021). Retrieved Nov 8, 2022 from https://github.com/kinivi/hand-gesture-recognition-mediapipe

[5] Rhys Newbury. 2020. Learning to Take Good Pictures of People with a Robot Photographer. (2020). Retrieved Nov 8, 2022 from https://doi.org/10.1109/IROS45743.2020.9341086

[6] Mike O'Leary. 2019. Photographer Uses Robots to Shoot Commercial Videos. (2019). Retrieved Sep 22, 2022 from https://fstoppers.com/bts/photographer-uses-robots-shoot-commercial-videos-383421

[7] Anca D. Dragan Siddharth Reddy, Sergey Levine. 2022. First Contact: Unsupervised Human-Machine Co-Adaptation via Mutual Information Maximization. (2022). Retrieved Dec 13, 2022 from https://arxiv.org/abs/2205.12381

[8] Research Infinite Solutions. 2021. Hand Gesture Recognition with Python. (Jan 2021). Retrieved Nov 8, 2022 from https://ris-ai.com/hand-gesture-recognition

[9] Michael Zhang. 2019. Photographer Uses Robots to Shoot Commercial Videos. (2019). Retrieved Sep 22, 2022 from https://fstoppers.com/bts/photographer-uses-robots-shoot-commercial-videos-383421