Hand Gesture Robot Control

Joshua Cox
Mechanical Science and Engineering
University of Illinois Urbana-Champaign
Champaign, United States
jgcox2@illinois.edu

Asher Mai
Electrical and Computer Engineering
University of Illinois at Urbana-Champaign
Champaign, United States
hanlinm2@illinnois.edu

Ian Xu
Electrical and Computer Engineering
University of Illinois at Urbana-Champaign
Champaign, United States
yiyangx6@illinois.edu

***Abstract*—Real time robotic arm control has evolved tremendously in the past decade. Existing control methods either require users to purchase additional hardware, or they do not offer a natural experience. We introduce a new way to control robotic arms using gestures. The implementation is fully vision based, requiring only a desktop or laptop webcam, without needing depth sensing. We demonstrate drawing and pick-and-place tasks as a proof of concept. Our system runs in real time on a CPU at about 10 frames per second. A video demo is available at:**
https://hanlinmai.web.illinois.edu/images/HRI.mp4

Keywords*—gesture control, robotic arm, computer vision, gesture detection, gesture recognition, teleoperation, telepresence, pick and place, drawing task, real time control

I. Introduction

A. Motivation

Keyboards and hand-held controllers traditionally used for gaming have become a popular tool to control robotic arms. The buttons can control the end-effector's actions, such as gripper opening and closing while the joysticks on these controllers can be used to move and rotate the robotic arm. However, this is not as natural or fluid as using gestures. Existing gesture controls involve using special hardware such as gloves with inertial measurement units (IMU) and VR controllers. The need for these hardware limits the scalability of these systems, requiring the users to purchase addition equipment.

Teleoperation and telepresence have also been a popular area of research. They allow someone be "present" in the form of a robot without physically being there. It has applications in many domains, including search and rescue, remote surgical work, and remote assistance. However, these systems also suffer from the same problem of requiring additional specialized equipment to control the robots.

Vision-based approach to robotic arm control allows the system to be easy to use and scalable without needing special hardware. However, it is challenging due to latency, variable lighting conditions, and lack of depth information. We solve these challenges by using a small neural network architecture for gesture detection that has fast inference and generalizability to a wide range of lighting conditions. Gestures allow us to control the robotic arm in the z-direction, so it eliminates the need for depth information.

B. Overview of our system

We introduce a fully vision based and natural way of controlling robotic arms. The only additional hardware required is a common webcam that comes with most laptops or a separate inexpensive desktop webcam that can be easily

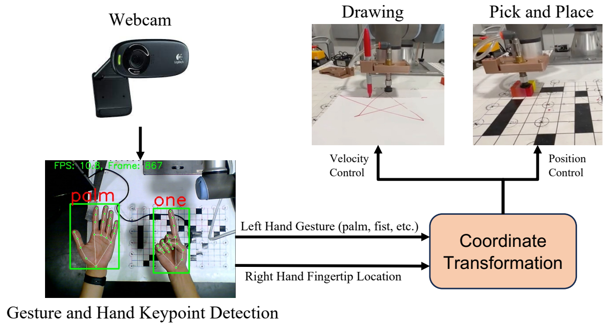
Webcam
Drawing
Pick and Place
Velocity Control
Position Control
FPS: 10.8, Frame: 867
palm
one
Left Hand Gesture (palm, fist, etc.)
Right Hand Fingertip Location
Coordinate Transformation
Gesture and Hand Keypoint Detection

Figure 1. Diagram showing our pipeline. We start by capturing a video with a webcam, then run each frame through our gesture and keypoint detection pipeline. These information then gets translated into robot workspace positions and robot actions that will be used

purchased. There is no need for specialized depth sensing cameras as we only use 2D image pixels for gesture recognition and detection. Our vision system runs at about 10 FPS on a CPU without the need for a GPU.

The user controls the robot with two hands. The left hand performs different gestures to control the end-effector's actions (turning on and off the vacuum gripper, etc.) and movement in the z-direction, while the right hand's index fingertip moves the robot in the x- and y-direction.

We demonstrate two tasks accomplished by this method of control: drawing and pick-and-place tasks. In drawing tasks, the left hand's fist and palm gesture controls the robot's up and down movement respectively. In pick-and-place task, the left hand uses three gestures: palm, fist, and scissors, to control both the robot's vacuum gripper and the up and down motion. In both tasks, the right hand's index fingertip controls x- and y-direction movements.

II. Literature Review

Our team reviewed similar teleoperation solutions completed from around the world. These include a project from MIT demonstrating teleoperation of a robot using an Oculus Rift virtual reality headset and hand controllers.

Figure 2. MIT Lab Robot Arm Teleoperation Demo

Solutions such as these demonstrate a unique human-robot control scheme that is natural to humans: hand motions. However, they usually require additional hardware or technology, such as the headset and controllers pictured above. Our solution aims to achieve this type of control with minimal hardware. We demonstrate teleoperated control using a computer and a USB camera.

III. Methodology

A. *Gesture Recognition*

We use a pre-trained SSDLiteMobileNetV3Large model from *HaGRID - HAnd Gesture Recognition Image Dataset* [1] for gesture detection. The model provides a bounding box location of the gesture as well as the classification among 20 classes of gestures. We experimented with different detection models, including SSDLiteMobileNetV3Large, SSDLiteMobileNetV3Small, and FRCNNMobilenetV3LargeFPN. We found that the FRCNN based model provides the most accurate classification and bounding box, but it's very slow, running at less than 3 frames per second, which is unfit for the real time tasks that we want to perform, such as drawing and pick-and-place tasks. We also found that the Large version of SSDLite model is more accurate than the Small version, with little to no performance difference, both running at around 10 frames per second. So we decided to use the SSDLiteMobileNetV3Large model.

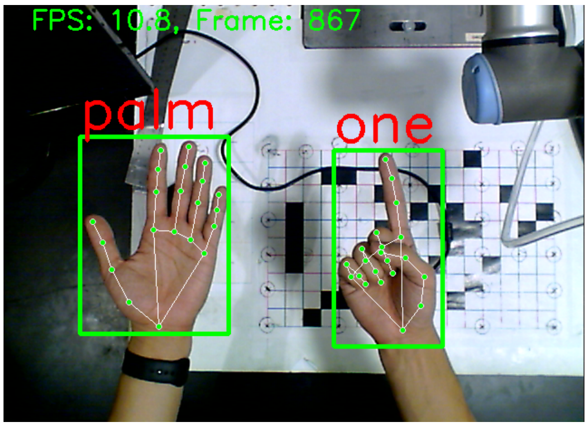

Figure3. The visualization of landmark detection and gesture detection. Bounding box and red text shows the location of gestures and classification of the gesture respectively. Green dots connected by green line segments show the keypoints detected in each hand.

We use hand keypoint and landmark detection from Mediapipe [2], which provides the locations of 21 hand joints for each hand. We extract the location of location of the right hand's index fingertip in pixel coordinates.

Neigher [1] nor [2] provides information about whether the hand is a left hand or a right hand. When there is only one hand present, we set that hand as the right hand so that the robot's x- and y-direction movement can be controlled without holding up both hands the whole time, preventing fatigue for the user. In the case of two hands being present on camera, we use locations to determine whether it's a left or right hand. In hand gesture detection with [1], the hand with leftmost top-left corner of the bounding box is determined as the left hand. In hand landmark detection with [2], the hand with the rightmost index fingertip location is the right hand.

We performed ablation experiment to see how it could improve our system's frames per second frequency. We find that our system runs at about 10 frames per second with both gesture detection and keypoint detection enabled, and it runs at about 15 frames per second if one of them is disabled.

Gesture Detection	Keypoint Detection	Average FPS
×	×	~30
✓	×	~15
×	✓	~15
✓	✓	~10

Table1. Ablation experiment to see if we could improve frame rate if we eliminate some detection systems.

B. *Coordinate Transformation*

The locations provided by gesture detection and keypoint detection systems are in the units of image pixels. We need to translate these 2D points from the image space to the physical workspace of the robot in units of centimeters. To solve this problem, we added a coordinate transformation layer. First, we define the top left corner and bottom right corner in image space to be $(x_1, y_1) = (0, 0)$ and $(x_2, y_2) = (w, h)$ where w and h are the width and height of the image. Then we define the top left corner and bottom right corner of the robot workspace as (X_1, Y_1) and (X_2, Y_2) respectively in units of centimeters. These two coordinates are user defined and can be customized to adjust the rectangular space that the user wants the robot to move within. Then for every point (x_i, y_i) of the index fingertip defined in the image space, we translate it into a point in robot workspace (x_r, y_r) with the following equations:

$$x_r = X_1 + \frac{X_2 - X_1}{x_2 - x_1} * (x_i - x_1) \tag{1}$$

$$y_r = Y_1 + \frac{Y_2 - Y_1}{y_2 - y_1} * (y_i - y_1) \tag{2}$$

C. Gesture to Action Mapping

For drawing task, it is straight forward to map one gesture for going up, and one gesture for going down where the pen contacts the paper. We mapped "palm" gesture to "going up" and "fist" gesture to "going down".

For pick and place task, it is not so straight forward because the action of going up and going down need to be coupled with the action of turning on and off the gripper. We first listed the steps it requires to do pick and place tasks:

1. Going down with vacuum gripper turned on
2. Going up with vacuum gripper turned on after picking up object
3. Moving to destination with the gripper turned on and in an up position
4. Going down with vacuum gripper turned on
5. Going up with vacuum gripped turned off

We notice that steps 1 and 4 can be grouped into one gesture, steps 2 and 3 can be grouped into another gesture. So we decided that we only need three gestures:

- Fist: Going down and turn on vacuum gripper
- Scissors: Going up and with the vacuum gripper still on
- Palm: Going up with the vacuum gripper turned off

D. Control

We opt for the velocity controller over the position controller when selecting between the two types of robot controllers for the drawing task.

Position control directly regulates the position of the robot component, the end-effector in our case. It's widely used in tasks which require high-precision or any robotic system where exact positioning is crucial. However, position control might not be the best choice for applications requiring constant movement because it tends to make the movement of the system discontinuous between each waypoint as it will come to a complete stop at each target position before proceeding to the next. In other words, there are necessary transitions between waypoints. Even though many methods could increase the smoothness of the transition, such as S-Curve Motion Profiles, Predictive Control, or mechanisms that ensure soft start and stop, the motion inherently involves moving from one point to another, which introduces some level of discontinuity. For the drawing task, while precision is important, the continuity of the motion is much more crucial as people usually do not appreciate broken lines in artworks.

Velocity control consists of two main components: speed regulation and direction control. It regulates the speed of the robot's moving part, the end-effector in our case; it also controls the direction of movement, ensuring the robot moves along the desired path. Under this system, the robot moves continuously, maintaining its direction and speed until the new command is received.

We used functions from the 'ur_rtde' library, a software interface designed for real-time data exchange with Universal Robots (UR) robotic arms, to get the information of the current position and velocity of the TCP, as well as controlling the robot.

For velocity controller, we use 'speedL' function from the 'ur_rtde' library, it enables us to command the robot to move its end-effector with a specified linear velocity, in other words, the end-effector will move in a straight line at a constant speed. The function parameters include the velocity vector, acceleration, and the duration that the motion should be executed. The velocity vector is a 3D vector representing the desired linear velocity in Cartesian space, and the acceleration is the rate that the end-effector will reach the desired velocity.

We use PD control to generate the control input for tracking the target path accurately:

$$u(t) = K_p \cdot e(t) + K_d \cdot \frac{e(t) - e(t-1)}{\Delta t} \tag{3}$$

where $u(t) \in \mathbb{R}^3$ denotes the control input, K_p and K_d are control gains, $e(t) \in \mathbb{R}^3$ denotes the position error, and we approximate the derivative of position error using Δt, the time interval between the current and previous error measurements.

To fine-tune the responsiveness of the system, we add a scalar multiplier to the original control signal $u(t)$:

$$u_s(t) = \alpha \cdot u(t) \tag{4}$$

where $u_s(t) \in \mathbb{R}^3$ denotes the scaled control signal, and α is the scalar value used to scale the control signal.

We update and send the 'speedL' command to the system every 5ms. We have observed that using a higher control frequency can overwhelm the system, causing the frame rate to drop below one frame per second.

E. Utensil Holder

Our initial goal for this project was to use gesture recognition in order to make a robot arm draw using some form of writing utensil. This required a utensil holder that would fit onto the existing UR3 robot arms available to us. Later, we switched to the default vacuum UR3 end effector in order to control the arm doing pick-and-place tasks with blocks.

The utensil holder featured a mounting system for a writing device such as the red sharpie we used for this project. A nut was inserted into the print in order to lock the sharpie into place with a bolt. The holder needed to be printed in two pieces, so they were bolted together after being mounted. It was important the locking mechanism for the pen be adjustable in order to lower the sharpie such that it did not interfere with the default end effector.

The utensil holder was designed in Creo (a CAD software) and subsequently 3D printed. Below are several pictures of this design and the intermediate pieces created in order to reiterate and improve on the fit.

The initial design in Figure 4 was a bit unrefined – the fit with the robot arm was bad resulting in the entire mount wobbling. This would not work well for a task requiring a static pen. The holder was redesigned as shown in Figure 5. Pictures of the bolt nut slot and the printed and tested holders are also shown in Figures 6-8.

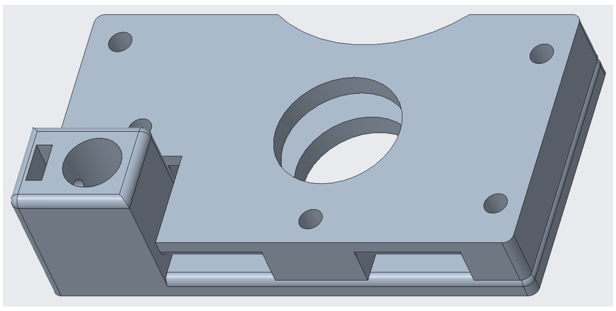
Figure 4. Initial Utensil Holder CAD

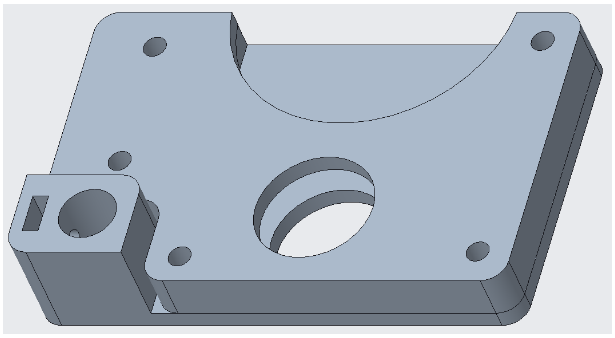
Figure 5. Final Utensil Holder CAD

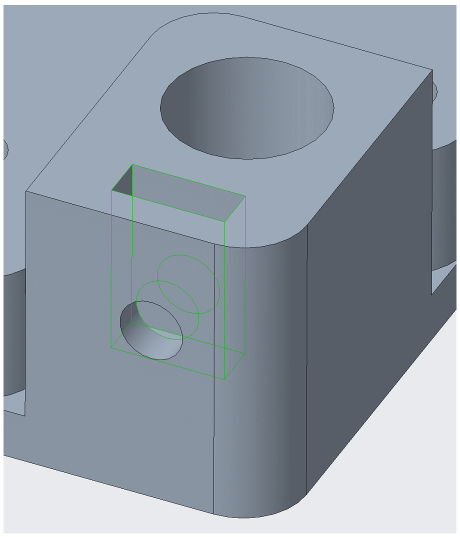
Figure 6. Bolt Nut Slot

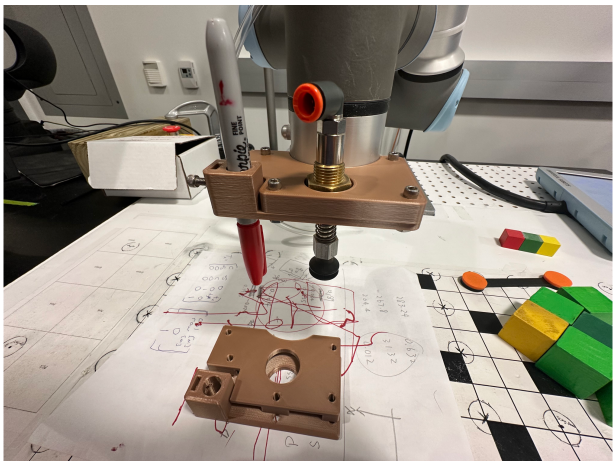
Figure 7. Final Utensil Holder Mounted (Initial Prototype Below)

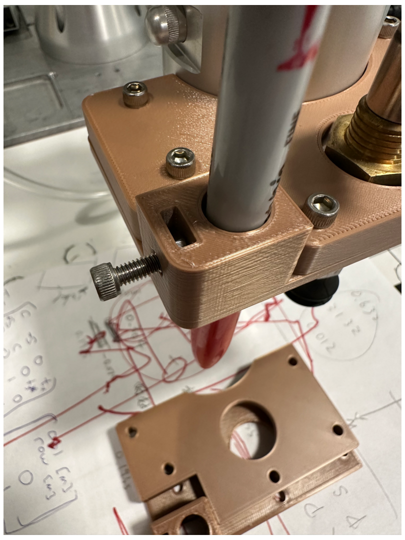
Figure 8. Pen Locking Mechanism

A design change that ended up helping was to CAD the UR3 end effector to begin with. This allowed for more accurate measurements and a better fit to be made.

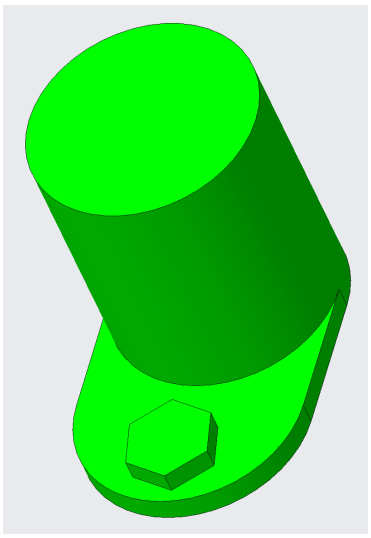
Figure 9. UR3 End Effector CAD Model

IV. Results

A. Gesture Recognition

In our experiments, we are able to see that the gesture recognition and keypoint detection works well enough for our purposes, with very occasion misdetection or misclassification that don't critically damage the user experience. Our 10-frame-per-second system allows real time feedback of the user's intended gesture.

We do find, however, that having two gestures at the same time limits the range of the right hand. Because the left hand has to be present in the image, the right hand cannot move too much to the left so as to overlap with the left hand. Thus, it limits the pick and place task to a rectangular region no more left than the left hand.

B. Control

We conducted tests on the controller by performing the following experiments: the robot's trajectory was hardcoded to follow (1) straight lines, (2) rectangles, and (3) circles. We then evaluated whether the movement was continuous and smooth, and assessed if the end-effector consistently reached the designated target positions during the movement, such as the corners of the rectangles. After performing multiple trials, the results showed that the controller was able to achieve continuous and smooth movement, and the end-effector accurately reached target positions in every trial.

We conducted similar experiments for testing the performance of the real-time control capabilities, specifically its responsiveness to hand movements and its precision. Instead of predefined trajectories, we passed the coordinates of the hand to the system and facilitated teleoperation. While the robot could still move smoothly in human-intended trajectories, we noticed that there was always a delay between hand movements and the corresponding robot movements. We believe the latency is due to the limitation of computing power of the lab computer.

C. Utensil Holder

The utensil holder worked well to demonstrate the initial idea of the project. However, simply locking the pen in place is not the best solution due to uneven surfaces. As shown in Figure 10, not all lines were an even thickness. This happens because the pen tip "misses" the surface as it dips. A way to avoid this would be a spring-loaded mechanism to apply roughly constant pressure on the paper.

A drawback of how the pen holder was set up is that it is not realistic to how humans draw or write. Humans angle writing devices which vary line thickness and writing style greatly. More degrees of freedom of the end joint and force control would be required to create a human-accurate drawing robot. The design we created is simple enough for transferring gestures to paper, however.

Other applications of this kind of gesture recognition would require different end effectors. Creating 3D models and subsequently printing, iterating, and redesigning end effector accessories allows for quick prototyping, an important skill in the engineering world.

D. Drawing Task

As previously mentioned, we utilized left-hand gestures to signal the robot to either raise or lower its arm, enabling the sharpie to make contact with the canvas, and we used the landmark of right hand's index finger to teleoperate the end-effector. Experiments were conducted with the aim of having the robot draw various shapes on the canvas.

The system was capable of drawing shapes such as rectangles, straight or curved lines, and circles. However, the results were not perfect as we initially expected. For instance, while we were able to draw a star (see Fig. 10), there were discontinuities which might be caused by delays or inconsistent application of pressure from the sharpie to the canvas.

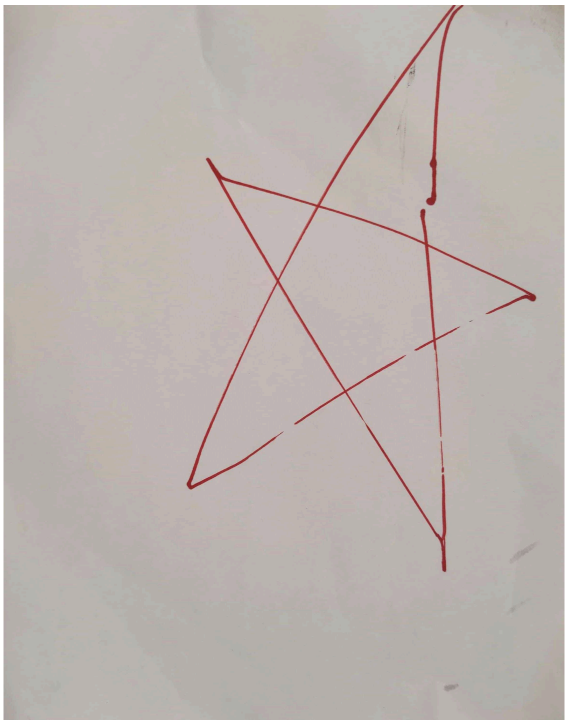
Figure 10. Five-Point Star Drawn via Teleoperation

E. Pick-and-place Task

We implemented three distinct gestures for left hand--fist, peace, and palm--to turn on the suction cup and lower the arm, raise the arm, and turn off the suction cup and raise the arm, respectively. Considering that the pick-and-place task does not require continuous and smooth movement, we implemented the position controller to facilitate a comparison with the velocity controller utilized in the drawing task (see Fig. 10). As expected, the movement of the end-effector was discontinuous, however, the system was capable of executing all the fundamental functions: the robot was able to relocate the blocks in response to the commands issued by both hands.

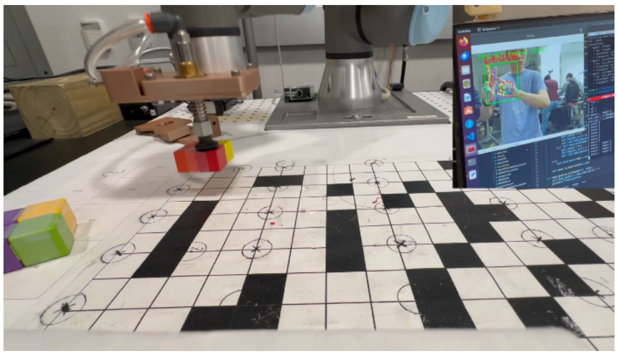

Figure 11. Pick and place task. The image shows the user using “scissors” gesture to pick up the block.

V. Conclusion

Our robotic arm control demonstrated the possibility of vision-based real-time gesture control. We were able to successfully draw with the robot and complete simple pick-and-place tasks. This implementation required no wearable hardware, only the recognition of proper hand gestures was needed to control the robot.

We learned that this implementation is limited by the computational ability of whatever computer it is running on. The tasks were completed slowly with a real-time FPS of 10, and we experienced noticeable delay between the hand movement and the robot movement. This could be a possible trade off with other methods that require additional hardware such as motions sensors or virtual reality devices.

For future directions, upgrading the hardware to support higher control frequencies and more powerful processing capabilities is essential. Such upgrades are expected to reduce the latency and improve responsiveness. Furthermore, we can expand the system’s capabilities to other applications such as manufacturing processes, interactive education tools, etc. Finally, we can develop a more intuitive user interface, and possibly incorporating with AR/VR technologies to make the system more accessible.

References

[1] Kapitanov, Alexander, Andrew Makhlyarchuk, and Karina Kvanchiani. "Hagrid-hand gesture recognition image dataset." (2022).

[2] Mediapipe hands. https://developers.google.com/mediapipe/solutions/vision/hand_landmarker, 2019.