EECS 442 Final Project Report: CV-Based Robotic End-Effector Controller

Anik Mumssen
University of Michigan
mumssen@umich.org

Owen Park
University of Michigan
owenpark@umich.org

1. Introduction

Robotic end-effector (EE) control is currently an extremely prominent and important area of robotics research. Although there are many ways of controlling an end-effector, like a robotic arm, through mechanical processes like a remote control or manual programs, a new state-of-the-art method for such a task is utilizing machine learning, specifically computer vision models. A crucial benefit of using computer vision techniques for EE control is that it can act as a catalyst for human-inspired robotic control. We present a computer vision-based EE controller that tracks human arm and hand motions to control a robotic arm EE. Our method motivates a large area of research in robotics and computer vision due to the multitude of important applications it provides such as in assistive robotics for handicapped users, surgical robotics given its intuitive human motion mirroring, and robotic control for inaccessible and extreme environments.

2. Related work

Prior approaches to robotic EE control include wearable motion-capture devices and vision-based servoing methods, each offering distinct benefits and limitations. Wearable systems such as gloves, inertial trackers, and exoskeletons provide low-latency motion capture and precise joint measurements, but they introduce significant drawbacks, including user discomfort, cumbersome electro-mechanical infrastructure, and high equipment costs. Although effective for direct teleoperation, these systems can hinder natural hand motion and reduce intuitiveness, limiting their practicality for long-term or everyday use [1, 2]. In contrast, image-based visual servoing (IBVS) controls the robot directly in the image plane by minimizing 2D feature error, enabling visually guided manipulation without full camera calibration [3, 4]. IBVS is robust to certain modeling errors and avoids the need for 3D reconstruction, but tracking only 2D image-space displacement produces motion that is less human-like and less suitable for applications requiring naturalistic arm trajectories. Moreover, the lack of explicit 3D reasoning can limit precision in complex environments. Our approach bridges these gaps by using calibrated multi-camera computer vision to recover true 3D hand motion, thereby retaining the convenience and low hardware burden of vision-based systems while enabling more intuitive, human-inspired robotic behavior.

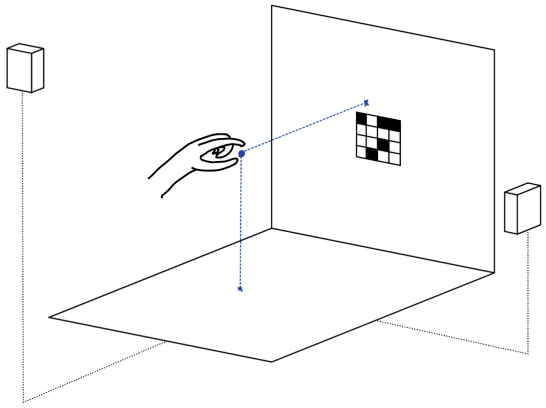

Figure 1. Operator setup with two cameras and an ArUco tag.

3. Method

3.1. Calibration

Our calibration procedure consists of estimating both camera intrinsics and extrinsics prior to triangulation. To obtain intrinsics, we captured multiple views of a ChArUco board while moving each camera, allowing OpenCV's `cv::aruco::calibrateCameraCharuco` function to solve for the intrinsic matrix $\boldsymbol{K}$ and distortion coefficients $\text{dist} = [k_1,\ k_2,\ p_1,\ p_2,\ k_3]^\top$ by minimizing reprojection error of detected corners. This step recovers focal lengths, optical center, and lens distortion needed for undistortion and pixel-to-ray normalization [5]. Extrinsics were then computed by detecting an ArUco tag, which defines the origin of the world frame, enabling estimation of each camera's rotation $\boldsymbol{R}$ and translation t relative to the tag (Fig 1).

3.2. Finger Detection

We use Google's MediaPipe Hand Landmarker to detect 2D finger joints in each camera's image frame [6]. MediaPipe provides twenty-one landmark coordinates per frame, giving precise (x, y) pixel locations. These detections form the inputs to our stereo triangulation pipeline, allowing recovery of corresponding 3D fingertip positions.

3.3. Stereo Triangulation

We triangulate 3D fingertip positions using OpenCV's `triangulatePoints`, which intersects normalized rays from both calibrated cameras in a shared world frame. This triangulation happens in real time during the experiment. We use full intrinsic parameters and each camera's extrinsic transform $(\boldsymbol{R}, t)$, enabling accurate 3D reconstruction from arbitrary viewpoints which is more robust than assuming parallel, rectified geometry.

3.4. ROS 2 Integration

The end-effector controller is broken up into four nodes that run in parallel (launched by `cv_hand_controller.launch.py`). They are launched with parameters such as camera FPS, event loop frequency, and camera intrinsics/extrinsics. The nodes are as follows:

1. $2\times$ CameraNode (`camera_node.py`)
2. TriangulationNode (`triangulation_node.py`)
3. IKBridgeNode (`ik_bridge_node.py`)

The two CameraNode's are each responsible for capturing an image and running MediaPipe's Hand Landmarker. It publishes this result to be used by the TriangulationNode. Its behavior is described by Algorithm 1.

Once the TriangulationNode receives the finger coordinates from both CameraNode's, it will run stereo triangulation and calculate a final (x, y, z) in the ArUco tag world frame. It sends this result to IKBridgeNode which will transform the ArUco tag world frame coordinate to a coordinate in the robotic arm's frame. This transformation is purely translational, scalar, and permutational. We define the origin of the ArUco tag to map to the arm EE position when it is fully centered and extended. We then send this final arm coordinate to control the robotic arm EE.

4. Experiment

We implemented our controller in real life using an NVIDIA Jetson Orin Nano Super and two USB cameras (innomaker U20CAM-1080P). We used a ROS 2 simulator built by the Michigan Mars Rover Project Team in order to simulate their robotic arm (Fig. 2). Our camera was configured to run at 640x480 30 fps. We were able to get one full control loop (from camera capture to commanding final coordinate) to run at 10 fps. Our initial experimentation was successful and we were able to visually confirm responsive and accurate control of the robotic arm EE in the simulator. A video of this demonstration can be found on YouTube (`https://youtu.be/gbK6bgUwYUo`).

Algorithm 1 CameraNode

$params \leftarrow ros2.get\_parameter()$
$cap \leftarrow cv2.VideoCapture(params[camera])$
$landmarker \leftarrow mediapipe.HandLandmarker()$
loop
  $img \leftarrow cap.read()$
  $result \leftarrow landmarker.detect\_async(img)$
  $await(result)$
  $fingerCoords \leftarrow result.hand\_landmarks$
  $publish(fingerCoords)$
end loop

Algorithm 2 TriangulationNode and IKBridgeNode

$fingerCoords[0] \leftarrow subscribe(CameraNode[0])$
$fingerCoords[1] \leftarrow subscribe(CameraNode[1])$
loop
  $await(fingerCoords)$
  $indexWCoord \leftarrow triangulate(fingerCoords)$
  $thumbWCoord \leftarrow triangulate(fingerCoords)$
  $centerWCoord \leftarrow \frac{indexWCoord + thumbWCoord}{2}$
  $finalCoord \leftarrow ikTransform(centerWCoord)$
  $publish(finalCoord)$
end loop

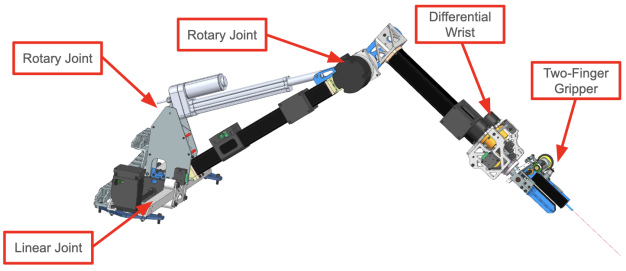

Figure 2. MRover's 5-DOF robotic arm.

5. Conclusions

This work shows that classical computer vision, especially calibrated multi-camera systems, remains highly effective for robotic EE control. Our calibration pipeline enables accurate 3D hand-to-robot mapping and can be improved through things like multi-view expansion and enhanced wrist tracking. Moreover, the system naturally produces high-quality 3D demonstrations, offering a valuable foundation for future imitation-learning research in robotic manipulation.

Our code can be found on GitHub. `https://github.com/owenpark8/cv_hand_controller`.

References

[1] Yingying Wang and Michael Neff. Data-driven glove calibration for hand motion capture. *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation.*, 2013.

[2] Chunxiao Lu et al. Teleoperated grasping using data gloves based on fuzzy logic controller. *Biomimetics*, 2024.

[3] Seth Hutchinson Hager, Gregory and Peter Corke. Tutorial tt3: A tutorial on visual servo control. *ICRA*, 1996.

[4] Danica Kragic and Henrik I. Christensen. A survey on visual servoing for manipulation. *Computational Vision and Active Perception Laboratory, Fiskartorpsv*, 2002.

[5] Nicolas Florent. A comprehensive guide to camera calibration using charuco boards and opencv for perspective correction. https://medium.com/@nflorent7/a-comprehensive-guide-to-camera-calibration-using-charuco-boards-and-opencv-for-perspective-9a0fa71ada5f. Date Published: 2023-12-18.

[6] C. Lugaresi et al. Mediapipe: A framework for building perception pipelines. *arXiv preprint arXiv:1906.08172*, 2019.