Zach Knee, Alan Zhong, Jacob Rock
Professor Adamczyk
ME/ECE 439
12 December 2024

# ME 439: Final Project - Color Sorting Robotic Arm

Our project is focused on integrating computer vision technology to sort objects based on color. This robot arm is a 6 axis robot with a gripper attached to the end, allowing it to pick up objects. Our robot will be given a lightweight object that is either red,or blue in a predetermined location. The robot arm will pick up the object and hold it in front of a color sensor. The camera will process the color and send it back to the robot, which will pick it up and place it into the bucket corresponding to the object's specific color. We chose this project because the exciting challenge of integrating computer vision with the kinematics code and the robotic arm would require a combination of the entire team's skill sets. The scope of this project is to achieve successful detection of red vs blue, while using predefined bucket locations and objects (red and blue whiteboard erasers) and autonomous sorting to ensure we have a minimum viable product in our time frame of around two weeks. The color sorting robot arm can serve as a basis for several real-world projects: it could be used in the farming industry to separate bad crops from the healthy ones or mixed crops (red vs white potatoes). Additionally, it can be used in factories where you need to separate out a specific number of different color pieces such as lego bricks (where there needs to be a specific ratio of each color brick).

The most important novel component of the robot is utilizing computer vision with a logitech camera. To achieve color detection we took advantage of the OpenCV library. The OpenCV library allows us to take colors from a live camera feed and return the color detected. We decided to convert the video to HSV color-model instead of RGB (or BGR in OpenCV) because it is better for detecting color. This is because RGB systems tend to have issues detecting color with inconsistent lighting. The HSV system stands for hue, saturation and value. The saturation and value can help us account for lighting disparities, and we can easily create a filter for colors by specifying a different range of hue values. For example, the range of blue hue for OpenCV is [100-140] and red is [0-10] and [170-180]. Notice, red "wraps around" on the hue color wheel which is why red has two ranges. Another concept required is masking, which returns a representation of 255 (white) when the color is present and 0 (black) if not. In order to reliably detect color from an entire picture, we can simply perform a mask on the center of the frame. If there are more pixels in the red mask above a tunable threshold, then we select red,

and vice-versa for blue. If there are not enough pixels of either color, we output None. This fixed some of the errors regarding the color sensing, although it is still not perfect and would need to be adjusted to generalize to more colors such as green. We have found that lighting still has some effect but it has been mostly mitigated by working in the constant lighting/background of Wendt Commons. The color the camera detected was published to ROS2 as a node, which was subscribed to by the main program node. This had some issues at first, as the robot would say it detected a color and should move to position, yet it would not move. We managed to solve this problem in ROS2 by using topic echo and RQT graphs to figure out which nodes were publishing and subscribing properly.

Once we got the camera color working, we made a testing file (named “final_move_keyboard.py”), which instead contained our mappings of positions the arm needed to get to in xyz coordinates. Using this file, the user can manually input the following commands: ‘g’ for close gripper, ‘r’ for release gripper, position names (‘block’, ‘bucket1’, ‘bucket2’), manual endpoint coordinates (x,y,z), and exit. To get these positions, we used an endpoint slider program to get the exact coordinates in the frames the robot is calibrated to. We used this program earlier in the class and it is more accurate than measuring them physically due to the previous calibration done with the robot. For code structure, we used the xarmrob package provided to us and developed the following files: xarm_keyboard.launch.py (for starting kinematics files), color_detect.py, final_move_keyboard.py, and color_move.py.

This main program node -color_move.py- is the one subscribed to the color detection node -color_detect.py-, although there were some initial issues getting the actions to occur. To function, it uses the same code as final_move_keyboard with an added behavior once it has subscribed to color. When the user types in ‘color’, the robot grabs the block and moves to the right bucket for red and left bucket for blue, drops the block in the bucket, and returns to the starting location. After it grabs the block, there is a debug message which displays what color is perceived and the user simply presses ‘q’ to move on with the above steps. While it succeeded in detecting red or blue and moving to the respective bucket, there were issues with its speed and gripping. The speed failure was an issue as the arm moved far too fast and almost knocked over the robot and hit the bucket due to the speed. This was fixed using smoother movement and lower speeds.

The gripper failure was a bit more interesting and a tougher fix as it took 3 combined solutions to solve. The gripper would only work on its own and not for the color function. We partially solved this problem by rewriting the section that it was involved in, and discovered it was an issue with misplaced ‘if else’ and ‘break’ statements. The other issue was mechanical as

the servo operating the gripper was closing way too much and was getting stuck due to the torque loads on the components of the linkage connected to the device. This was solved by setting the gripper to leave a gap that was slightly smaller than the block. The gap stops the servo from over gripping, which not only protects the servo but stops it from freezing up later once it over grips.

The following issue was the arm was releasing the block at the same time it was moving to the bucket, and resulting in the block getting thrown across the table. The issue was resolved by using the ‘time sleep’ command, as it stops the code from running anything for a short period of time. This allows the robot to get into position before the gripper is released. Once this was functional, there was a minor issue with the bucket and the arm. Depending on where the block was grabbed, it may tilt in a manner that makes it connect with the bucket while crossing over the top edge of it. This causes the bucket to move out of the way and result in the block to the bucket as it is no longer in the correct position.

We corrected this by creating another location that served as a midpoint at a higher position. It is between the block and bucket position but at a higher position to allow the block to clear the bucket. It creates a smoother movement as well, which makes the block less likely to fly out of the gripper. To make sure that it was enough of a fix, we also changed the positions of the buckets to be a bit higher to add higher clearance. This helped a bit with the smoothness, but it was not fully resolved by doing this.

To fully address this issue, we created steps between the joint angle and the target joint angles. Instead of being linear in its progression, we gradually transition to the middle point to its maximum speed then slowing back down as it reaches the target. This is controlled by changing the number of steps and the delay between each step using the ‘time sleep’ function.

While this all works at this stage, we wanted to make it so that there is only a need to run the program instead of giving it several commands to reset the robot for the next block. The commands that are needed to reset the robot are setting it back to the start position. If the program is run again immediately, it knocks the block over while also gripping midair. This was solved by adding another two commands to the cycle. We set it to go back to the same midpoint as earlier then go to the start position with the same smoothing functions, allowing it to easily return to the initial spot.

This step made our robot fully functional, meaning that it can perform the following steps: identify a color of a block, reach down to the block, grab the block, move it to the correct bucket smoothly, release the block, and return back to the start position. It can do all of this smoothly and with minimal error. Re-examining our goals, we were able to successfully sort based on two

colors with predetermined locations, but the process is not fully autonomous. To initiate the sorting, the user enters a prompt and then a confirmation based on the color received. With this being said, there are some future improvements that can be looked at. We have learned a lot about color detection, openCV, and inverse kinematics in the project and received satisfaction from the troubleshooting process.

As for future steps, there are a lot of options for us to choose from. The program could be a bit more autonomous, not requiring a user to place the block in a predetermined location. This would have it so the camera determines the position of the block and the arm autonomously reaches for it. This could be done through shape and edge detection, where the camera identifies lines in the object it can see and then uses algorithms to determine which object is the required one. The same could be done with the buckets as well. There could also be more colors added to the program but we do not have enough time to add more colors. Luckily, it would be an easy addition as our code is parameterized and modular. There have been several examples online that have inspired some future ideas, such as having the blocks move on a conveyor belt and the arm can catch and sort them [1]. This is a possible way it can be integrated into society in a helpful manner that could increase the efficiency of factories. There is an ethical dilemma that follows this, as it could potentially take jobs away from those who currently man these conveyor belts. Under a utilitarian framework, this would be acceptable as it would produce the most good for the whole of society. On a more fun project, we found other projects where the arm could be used to stack blocks and create fun towers [2]. This could be a great demonstration at STEM events to inspire the next generation to join the field and is one of the things that personally inspired us to get into robotics.

| | Zach | Alan | Jacob |
|---|---|---|---|
| Concept/Planning/Research | 34% | 33% | 33% |
| Design - Hardware | 20% | 20% | 60%. |
| Design - Logic | 33% | 34% | 33% |
| Programming | 40% | 40% | 20% |
| Presentation | 30% | 30% | 40% |
| Report | 20% | 20% | 60% |

References

[1] Mr. Goddard's STEM Tutorials. DOBOT Magician Color Sorter and Conveyor Belt, Mr. Goddard's STEM Tutorials, May 25, 2022, https://www.youtube.com/watch?v=hd3iq6ghkHY&ab_channel=Mr.Goddard%27sSTEMTutorials

[2] Freelance Robotics. Robotic Arm Sorting by Colour. Freelance Robotics, Jan 31, 2018, https://www.youtube.com/watch?v=4sL5Ljerb7w&ab_channel=FreelanceRobotics