EC544 FINAL PROJECT: COMPUTER VISION DRIVEN ROBOT ARM

Bobby Garces and Chelsea Lau

Contents

Table of Figures

Initial Project Proposal

Objective

Manual categorization of materials has a number of drawbacks, the most prominent being that it exposes the process to human error. When the effectiveness of a system depends on something as unreliable as human subjectivity, the system could potentially be rendered incapable of achieving its original purpose. An example exists at Boston University, where under the Zero Waste Plan, waste management in areas such as the George Sherman Union Food Hall require patrons to sort their waste into “trash,” “food waste,” or “mixed recycling,” as can be seen in Figure 1.

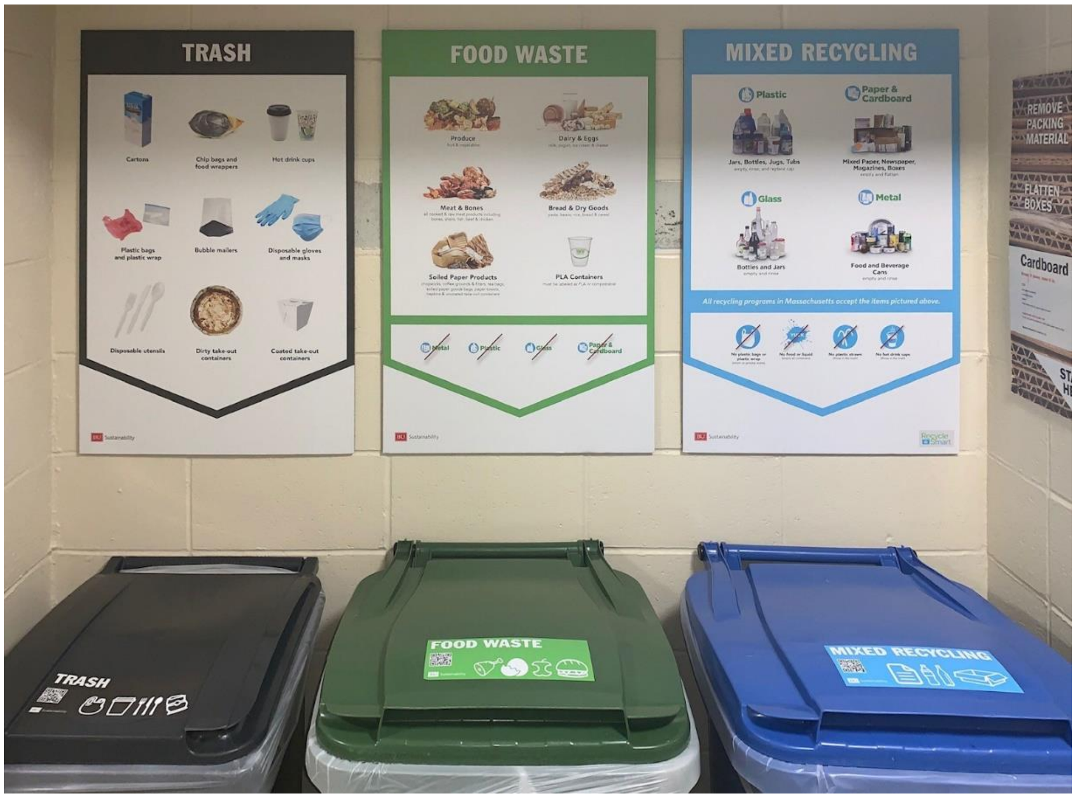


Figure 1: GSU Waste Disposal

Patrons are frequently confused about how to categorize their waste and either must ask employees for guidance or guess on their own. Most do not take the time to make their decision based on close inspection of the provided diagrams, leading to mis categorizations that could prevent the entire batch of waste from being processed properly.

Our proposed solution to this issue is automating waste management in these public areas using a robotic gripper aided with computer vision. By developing a system that will visually distinguish between and categorize items algorithmically, we will be able to eliminate the uncertainty introduced by human error and subjectivity and improve the success of processes such as those involved in BU’s Zero Waste initiative.

Project Execution Plan

The main steps for completing this project are as follows:

1. Order and receive components.
2. Install and test OpenCV and ROS on Raspberry Pi.
3. Ensure proper functioning of COTS parts such as robotic gripper by executing simple tutorials.
4. Begin learning the fundamentals of working with both ROS for 2-finger grippers and OpenCV.
5. Learn the kinematics of the robot arm and plan its appropriate paths.
6. Implement paths and end effector articulation via ROS.
7. Interface the camera's video signal to the Raspberry Pi.
8. Use OpenCV to identify objects based on color and shape, as well as determine the objects' position and orientation.
9. Using the information from OpenCV, program the robot using ROS to extend it's arm to the desired object and grab it.
10. Once an object is grabbed, program the robot to move the object to a predetermined location, then return to a 'home' position before next action.
11. Testing of various setups and objects to determine flaws of the project.
12. Update/fix issues.

After the initial stage of procurement, which will begin as soon as the project proposal is approved, the technical approach for this project will be executed in two modules: computer vision and robotic control. The modules will be completed in parallel, with Bobby Garces working on the former and Chelsea Lau working on the latter, and final integration will be a joint effort between all team members.

Computer Vision

The first step to working with OpenCV will be installation onto the Raspberry Pi, followed by interfacing the camera's video signal to the board. Next, we will complete simple tutorials to familiarize ourselves with the software library and its capabilities. Concepts that are common in image processing are: resizing / rescaling, contour detection, color channels, shape detection, color detection, masking, position detection, and edge detection. Using a static image, these concepts will be applied to a few test programs to gain a basic understanding of reading and manipulating image data. Afterwards, these core concepts will be applied to real time video from the camera. A simple test will be to place a singular object (such as a ball) in front of the camera, verify that the object is detected, and then move the object within the camera frame to ensure the object is tracked correctly. This learning period is planned to last about one week. Project-specific development will start promptly after, and begin with identifying the functions most relevant to distinguishing targets based on color and shape. Upon preliminary research, some of the functions that will be relevant to the project will be:

- img() returns pixel RGB information
- Img.shape returns image dimensions and number of color channels

- Cv.split and cv.merge for splitting and merging color channels of an image
- cv2.Canny() for edge detection
- cv2.findContours() for contour detection
- cv2.inRange(), cv2.bitwise_not(), and cv2.bitwise_or() for creating masks

These functions will be applied to a single object at first, and if successful, the next step will be to add objects of varying shapes and colors within the frame. If the program can handle multiple different objects within the frame at once and identify them correctly, this information can be fed to the robot arm for command. If there is sufficient time, position detection can be added to the program to allow for dynamic robot movement.

Robotic Control

Similarly to OpenCV, the first step in development will be software installation. To increase ease of use, Ubuntu Mate will be installed onto the Raspberry Pi as the operating system of choice. Ubuntu Mate was chosen since Ubuntu is the main supported operating system for ROS. Between the different versions of ROS, including ROS Melodic, Kinetic, and Noetic, we have chosen to use Noetic as it is the most recent distributed ROS1 version. Compared to the last version released, Melodic, Noetic supports Python3 and is compatible with Ubuntu 20.04.

Upon receiving the procured hardware, the robotic arm will be tested for proper hardware functionality using simple tutorials and the included joystick control. Then, a learning period of about one week will be used to gain a fundamental, general understanding of ROS before work on controlling the gripper is initiated. Some topics that will be covered during this familiarization process include creating a Catkin workspace, creating ROS packages, and ROS nodes.

Following the learning period will come the bulk of the system development to achieve the desired goal: controlling the robotic arm based on the decisions made through OpenCV. Initial research has shown the MoveIt! ROS package to be the most promising and relevant for our applications. It can be used for path and motion planning by generating and executing trajectories for the arm to bring the end effector to a desired position. To start, we will first try to plan the simplest path for the arm using related tools such as the ROS Visualizer, and have the arm successfully reach its destination and do a pick operation. Then, we will increase the complexity of the arm's path by having it move to a bin and drop before returning to a neutral position. Finally, we will plan and execute the full intended path of the arm: starting from the neutral position, it will move to the location of one object, do a pick operation, and drop in a bin. Then it will return to the neutral position before going to retrieve the next object within its vision, and so on. This development phase is planned to last around 3 weeks, from Week 2 until Week 4.

Milestones

Week 1	• Order parts • Install OpenCV and ROS to Raspberry Pi • Learn ROS fundamentals
Week 2	• Use OpenCV to rotate, scale, and filter a static image • Use OpenCV to find a single object based on color • Use Moveit! ROS to plan a simple path for robotic arm and do pick operation
Week 3	• Identify and track multiple objects at once • Use Moveit! ROS to pick item and bring to bin before returning to neutral
Week 4	• Finalize multiple object tracking from week 3 • Set up a test environment with the robot arm, mounted camera, containers, and objects • Execute complete trajectory of robotic arm
Week 5	• Command a simple straight line movement of the robot arm based on information from the video (i.e interface OpenCV with ROS)
Week 6	• Have robot arm execute full trajectory based on selected color
Week 7	• Test limits of object detection and arm movement
Week 8	• Report and documentation

Anticipated Challenges

- Lighting and Shadows: The main criteria for grouping objects will be color and shape. For color detection, this consists of pulling the red-green-blue components (0 - 255 for each color) from the camera. The brightness and contrast of the environment will change the exact readings. In a simple case where two distinct colors are used this won't pose an issue, however in more complicated scenarios with multiple objects and varying lighting this may cause misidentification of an object's color. Shadows can also cause issues if a shadow is identified as an object by the robot, even though it is not. To minimize any lighting related issues a possible solution is to mount a light next to the camera in a top-down view of the table. This will allow for a consistent background and minimize shadows.
- Zero prior experience with OpenCV: Given that the length of the project is only 8 weeks long, learning how to properly use OpenCV will need to be done in a week or two. The OpenCV library supports Python. Luckily, last semester in EC602 (Software by Design) we were taught Python programming and how to read documentation for various libraries. Those skills will be heavily leveraged in applying OpenCV to the project. This risk will be mitigated by limiting the scope of the project to distinguishing between two

similarly-shaped objects that only differ in color. Prior experience with other image processing tools will be beneficial to learning OpenCV in a timely manner as well.

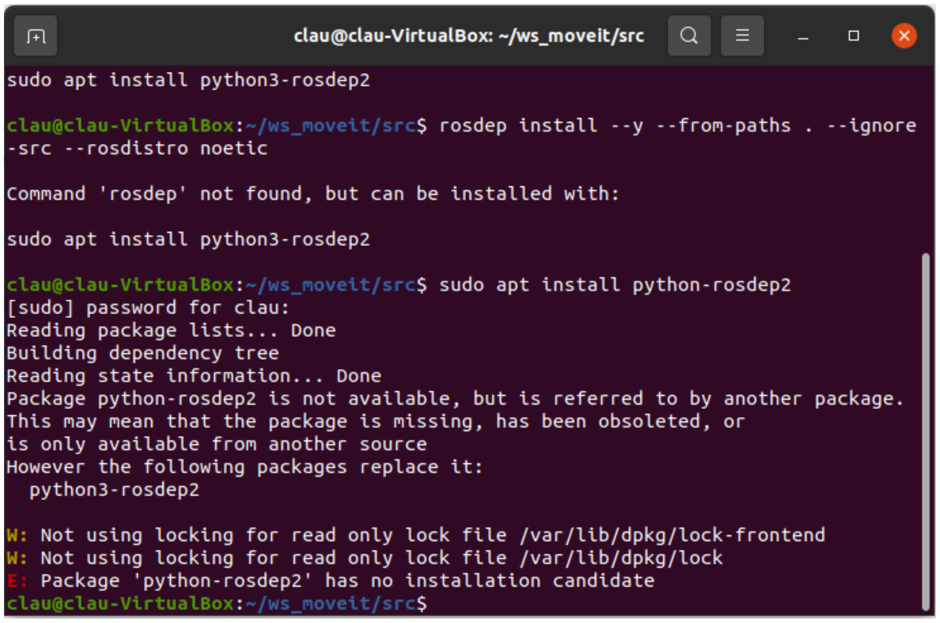
- Zero prior experience with ROS: Despite having worked alongside ROS development, no team members have direct experience developing in ROS. Similar to OpenCV, becoming familiar with the software libraries and how they interface with the robotic gripper will have to be accelerated. Proper utilization of ROS documentation, including tutorials, demos, and discussion forums, will be essential. Limiting the number of paths for the robotic manipulator by placing target items in set positions will reduce problem complexity as well. Furthermore, previous experience with robotic hardware will be beneficial and mitigate risk associated with final integration between software and hardware.
- Interfacing ROS and OpenCV: Along with the learning curve that comes with working with both ROS and OpenCV individually, there will also be complexities involved with getting the two integrated with one another. Getting to integration as early as possible in the project timeline and allocating ample time for this step will be crucial in preventing this from becoming an issue.
- Procurement of all materials: Normally none of the materials required for this project would have significant lead times, but with the global supply chain recovering from the COVID-19 pandemic certain parts can take weeks or months to arrive. If any parts are unattainable within a suitable time frame the project will need to be changed to accommodate an alternative part.

Project Execution

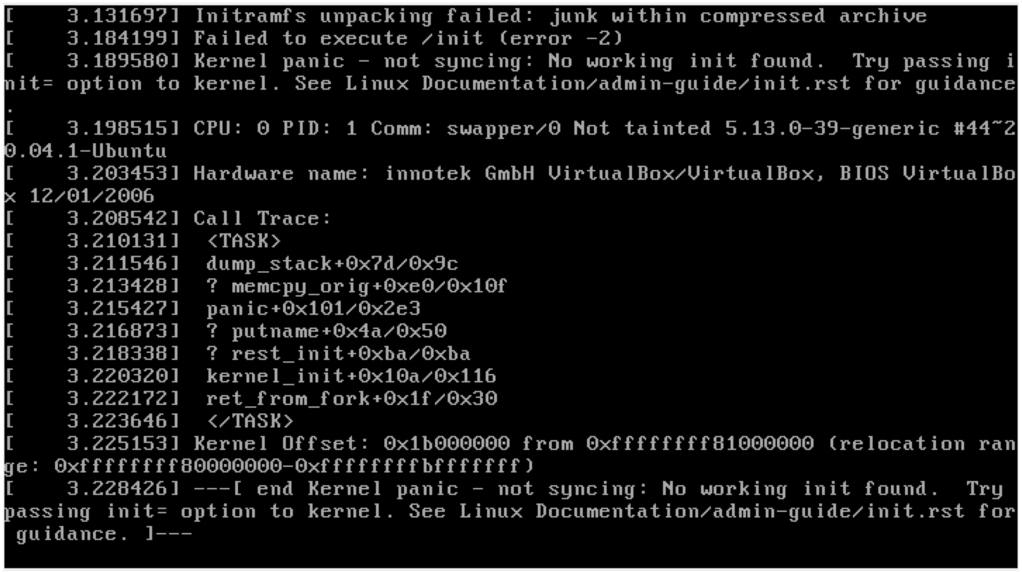
Challenges We Encountered

- Procurement: Due to the COVID-19 pandemic and the shipping delays that have resulted from it globally, our robotic arm arrived significantly later than expected, delaying the initiation of the project by 1-2 weeks.
- ROS and OS Installation: As anticipated, our inexperience with the Robot Operating System framework proved to be the biggest hurdle we faced, and ultimately became the most significant failure. The original plan involving ROS had been to download Ubuntu and ROS onto the Raspberry Pi itself and utilize the MoveIt! Motion Planning Framework to execute path planning and motion control of the robotic arm. The first hurdle we encountered was installing Ubuntu and ROS onto the Raspberry Pi, eventually discovering a bug that made Ubuntu 20.04 incompatible with the Raspberry Pi Model 3B+. As a result, Ubuntu 18.04 was installed onto the board instead, and ROS Noetic, the thirteenth ROS distribution but not the latest, was chosen. At this point a meeting with Professor Babak Kia resulted in the decision to work primarily from our PCs instead of the Pi, so Ubuntu 18.04 and ROS Noetic was also installed onto a new VM in order to maintain uniformity between the two systems, since it is uncertain whether or not different distributions of ROS can communicate with one another.
- MoveIt Package Installation: Installing the MoveIt package after ROS was successfully installed also proved to be a challenge. Due to what seemed to be package dependency

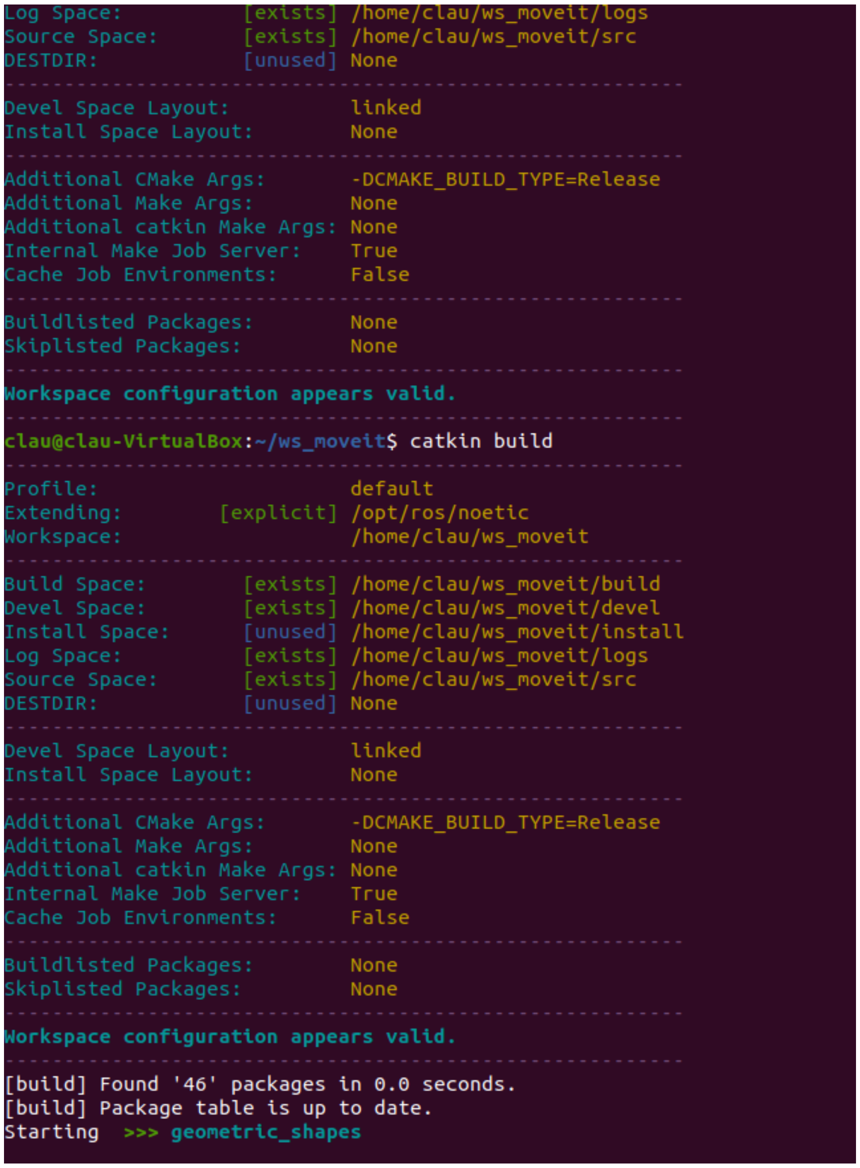
issues, we struggled to make and build a catkin workspace, as can be seen in Figure 2 below.


```
sudo apt install python3-rosdep2

clau@clau-VirtualBox:~/ws_moveit/src$ rosdep install --y --from-paths . --ignore
-src --rosdistro noetic

Command 'rosdep' not found, but can be installed with:

sudo apt install python3-rosdep2

clau@clau-VirtualBox:~/ws_moveit/src$ sudo apt install python-rosdep2
[sudo] password for clau:
Reading package lists... Done
Building dependency tree
Reading state information... Done
Package python-rosdep2 is not available, but is referred to by another package.
This may mean that the package is missing, has been obsoleted, or
is only available from another source
However the following packages replace it:
  python3-rosdep2

W: Not using locking for read only lock file /var/lib/dpkg/lock-frontend
W: Not using locking for read only lock file /var/lib/dpkg/lock
E: Package 'python-rosdep2' has no installation candidate
clau@clau-VirtualBox:~/ws_moveit/src$
```
Figure 2: Package Dependency Error

After attempting to amend the issues with additional package installations and rebooting the VM, we encountered a kernel panic and was no longer able to boot the system past this point.

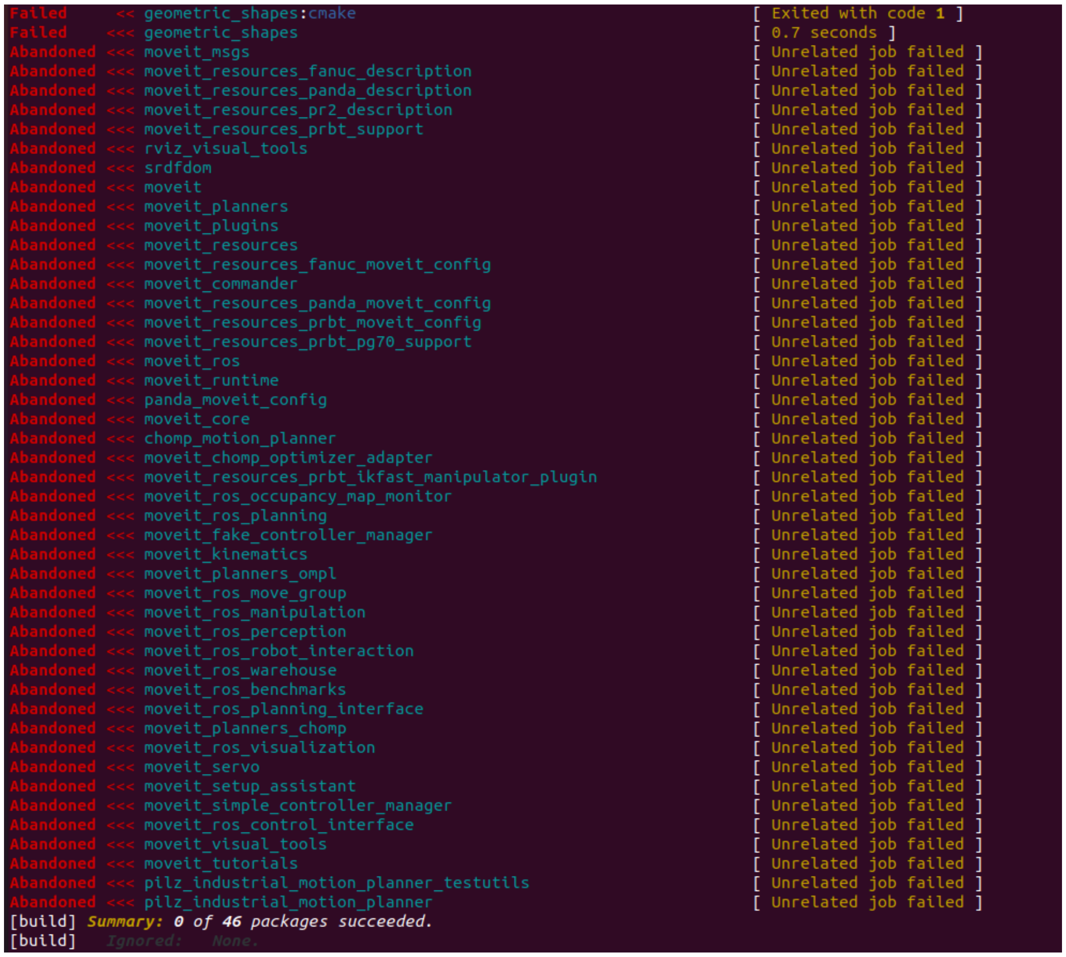

```
[    3.131697] Initramfs unpacking failed: junk within compressed archive
[    3.184199] Failed to execute /init (error -2)
[    3.189580] Kernel panic - not syncing: No working init found.  Try passing i
nit= option to kernel. See Linux Documentation/admin-guide/init.rst for guidance
.
[    3.198515] CPU: 0 PID: 1 Comm: swapper/0 Not tainted 5.13.0-39-generic #44~2
0.04.1-Ubuntu
[    3.203453] Hardware name: innotek GmbH VirtualBox/VirtualBox, BIOS VirtualBo
x 12/01/2006
[    3.208542] Call Trace:
[    3.210131]  <TASK>
[    3.211546]  dump_stack+0x7d/0x9c
[    3.213428]  ? memcpy_orig+0xe0/0x10f
[    3.215427]  panic+0x101/0x2e3
[    3.216873]  ? putname+0x4a/0x50
[    3.218338]  ? rest_init+0xba/0xba
[    3.220320]  kernel_init+0x10a/0x116
[    3.222172]  ret_from_fork+0x1f/0x30
[    3.223646]  </TASK>
[    3.225153] Kernel Offset: 0x1b000000 from 0xffffffff81000000 (relocation ran
ge: 0xffffffff80000000-0xffffffffbfffffff)
[    3.228426] ---[ end Kernel panic - not syncing: No working init found.  Try
passing init= option to kernel. See Linux Documentation/admin-guide/init.rst for
 guidance. ]---
```
Figure 3: Kernel Panic

This was eventually resolved by entering the GRUB menu upon booting, choosing a different configuration to boot, and running fsck manually to fix. However, even upon moving past this issue and making the catkin workspace successfully, the catkin workspace was unable to be built without errors.



```
Log Space:              [exists] /home/clau/ws_moveit/logs
Source Space:           [exists] /home/clau/ws_moveit/src
DESTDIR:                [unused] None
----------------------------------------------------------
Devel Space Layout:              linked
Install Space Layout:            None
----------------------------------------------------------
Additional CMake Args:           -DCMAKE_BUILD_TYPE=Release
Additional Make Args:            None
Additional catkin Make Args:     None
Internal Make Job Server:        True
Cache Job Environments:          False
----------------------------------------------------------
Buildlisted Packages:            None
Skiplisted Packages:             None
----------------------------------------------------------
Workspace configuration appears valid.
----------------------------------------------------------
clau@clau-VirtualBox:~/ws_moveit$ catkin build
----------------------------------------------------------
Profile:                         default
Extending:            [explicit] /opt/ros/noetic
Workspace:                       /home/clau/ws_moveit
----------------------------------------------------------
Build Space:            [exists] /home/clau/ws_moveit/build
Devel Space:            [exists] /home/clau/ws_moveit/devel
Install Space:          [unused] /home/clau/ws_moveit/install
Log Space:              [exists] /home/clau/ws_moveit/logs
Source Space:           [exists] /home/clau/ws_moveit/src
DESTDIR:                [unused] None
----------------------------------------------------------
Devel Space Layout:              linked
Install Space Layout:            None
----------------------------------------------------------
Additional CMake Args:           -DCMAKE_BUILD_TYPE=Release
Additional Make Args:            None
Additional catkin Make Args:     None
Internal Make Job Server:        True
Cache Job Environments:          False
----------------------------------------------------------
Buildlisted Packages:            None
Skiplisted Packages:             None
----------------------------------------------------------
Workspace configuration appears valid.
----------------------------------------------------------
[build] Found '46' packages in 0.0 seconds.
[build] Package table is up to date.
Starting  >>> geometric_shapes
```

Figure 4: Catkin Workspace Configuration



```
Failed      << geometric_shapes:cmake                                       [ Exited with code 1 ]
Failed     <<< geometric_shapes                                             [ 0.7 seconds ]
Abandoned <<< moveit_msgs                                                   [ Unrelated job failed ]
Abandoned <<< moveit_resources_fanuc_description                            [ Unrelated job failed ]
Abandoned <<< moveit_resources_panda_description                            [ Unrelated job failed ]
Abandoned <<< moveit_resources_pr2_description                              [ Unrelated job failed ]
Abandoned <<< moveit_resources_prbt_support                                 [ Unrelated job failed ]
Abandoned <<< rviz_visual_tools                                             [ Unrelated job failed ]
Abandoned <<< srdfdom                                                       [ Unrelated job failed ]
Abandoned <<< moveit                                                        [ Unrelated job failed ]
Abandoned <<< moveit_planners                                               [ Unrelated job failed ]
Abandoned <<< moveit_plugins                                                [ Unrelated job failed ]
Abandoned <<< moveit_resources                                              [ Unrelated job failed ]
Abandoned <<< moveit_resources_fanuc_moveit_config                          [ Unrelated job failed ]
Abandoned <<< moveit_commander                                              [ Unrelated job failed ]
Abandoned <<< moveit_resources_panda_moveit_config                          [ Unrelated job failed ]
Abandoned <<< moveit_resources_prbt_moveit_config                           [ Unrelated job failed ]
Abandoned <<< moveit_resources_prbt_pg70_support                            [ Unrelated job failed ]
Abandoned <<< moveit_ros                                                    [ Unrelated job failed ]
Abandoned <<< moveit_runtime                                                [ Unrelated job failed ]
Abandoned <<< panda_moveit_config                                           [ Unrelated job failed ]
Abandoned <<< moveit_core                                                   [ Unrelated job failed ]
Abandoned <<< chomp_motion_planner                                          [ Unrelated job failed ]
Abandoned <<< moveit_chomp_optimizer_adapter                                [ Unrelated job failed ]
Abandoned <<< moveit_resources_prbt_ikfast_manipulator_plugin               [ Unrelated job failed ]
Abandoned <<< moveit_ros_occupancy_map_monitor                              [ Unrelated job failed ]
Abandoned <<< moveit_ros_planning                                           [ Unrelated job failed ]
Abandoned <<< moveit_fake_controller_manager                                [ Unrelated job failed ]
Abandoned <<< moveit_kinematics                                             [ Unrelated job failed ]
Abandoned <<< moveit_planners_ompl                                          [ Unrelated job failed ]
Abandoned <<< moveit_ros_move_group                                         [ Unrelated job failed ]
Abandoned <<< moveit_ros_manipulation                                       [ Unrelated job failed ]
Abandoned <<< moveit_ros_perception                                         [ Unrelated job failed ]
Abandoned <<< moveit_ros_robot_interaction                                  [ Unrelated job failed ]
Abandoned <<< moveit_ros_warehouse                                          [ Unrelated job failed ]
Abandoned <<< moveit_ros_benchmarks                                         [ Unrelated job failed ]
Abandoned <<< moveit_ros_planning_interface                                 [ Unrelated job failed ]
Abandoned <<< moveit_planners_chomp                                         [ Unrelated job failed ]
Abandoned <<< moveit_ros_visualization                                      [ Unrelated job failed ]
Abandoned <<< moveit_servo                                                  [ Unrelated job failed ]
Abandoned <<< moveit_setup_assistant                                        [ Unrelated job failed ]
Abandoned <<< moveit_simple_controller_manager                              [ Unrelated job failed ]
Abandoned <<< moveit_ros_control_interface                                  [ Unrelated job failed ]
Abandoned <<< moveit_visual_tools                                           [ Unrelated job failed ]
Abandoned <<< moveit_tutorials                                              [ Unrelated job failed ]
Abandoned <<< pilz_industrial_motion_planner_testutils                      [ Unrelated job failed ]
Abandoned <<< pilz_industrial_motion_planner                                [ Unrelated job failed ]
[build] Summary: 0 of 46 packages succeeded.
[build]   Ignored:   None.
```

Figure 5: Catkin Workspace Configuration Errors

At this point the decision was made to start the process all the way from scratch, by creating a new VM, and reinstalling Ubuntu, ROS, and the MoveIt packages, following the installation instructions extremely meticulously. The catkin workspace configuration and build finally completed without issue once this was done. However, we were unable to proceed with using this motion planning and control software because we realized it required the generation of a URDF (Unified Robot Description Format) file for the robot arm. This file consists of a human-readable textual description of a robot's geometry and its parts, which is needed by the software to create the robot model and run simulations. However, writing it manually is quite complex and demanded more time than we had left. The alternative would have been to model the arm in a CAD software such as SolidWorks and generate the URDF, but we were restricted in our access to such software as ECE students and purchasing would have been extremely costly. For these reasons we made the decision to deviate from the plan of using ROS and use other robot control software modules instead.

Technical Achievements

Introductory Image Processing with OpenCV

To begin familiarizing ourselves with the OpenCV library, we first utilized it to accomplish several simple tasks: reading and modifying image data, splitting and merging color channels, and connecting to a webcam to take photos and videos. To read and modify image data, sample images were read into an integer array img[x,y], and data from the image file was extracted using methods such as img.item(x,y,z), img.shape, img.size, and img.dtype. The image array was then split using cv.split(img) into the three color channels (red, green, blue). Finally, we were able to interface the library with our webcam to capture images and videos through the cap.read() command.

Color and Object Detection

After laying the foundation for equipping the robotic arm with computer vision capabilities through those first basic achievements, we worked on developing the object tracking algorithm that would be used by the robot arm to identify and locate objects. For this project we selected color detection as the parameter for object tracking, though other parameters can be used. Although the most common color space is RGB (red-green-blue), to combat the issue of sensitivity to light and shadow variation we decided to use the HSV (hue-saturation-value) color space instead. HSV allows for the separation of image intensity from the color information, which mitigates the effects of lighting and shadow variation. Hue consists of the color portion (0-360 degrees), saturation describes the amount of gray present (0-100%), and value describes the brightness and intensity of a color (0-100%). Hue has a range of 0 to 180 while saturation and value have a range of 0 to 255 in OpenCV. In order to employ the HSV color space instead of the default RGB color space, the frame data was converted using the cv.cvtColor() command.

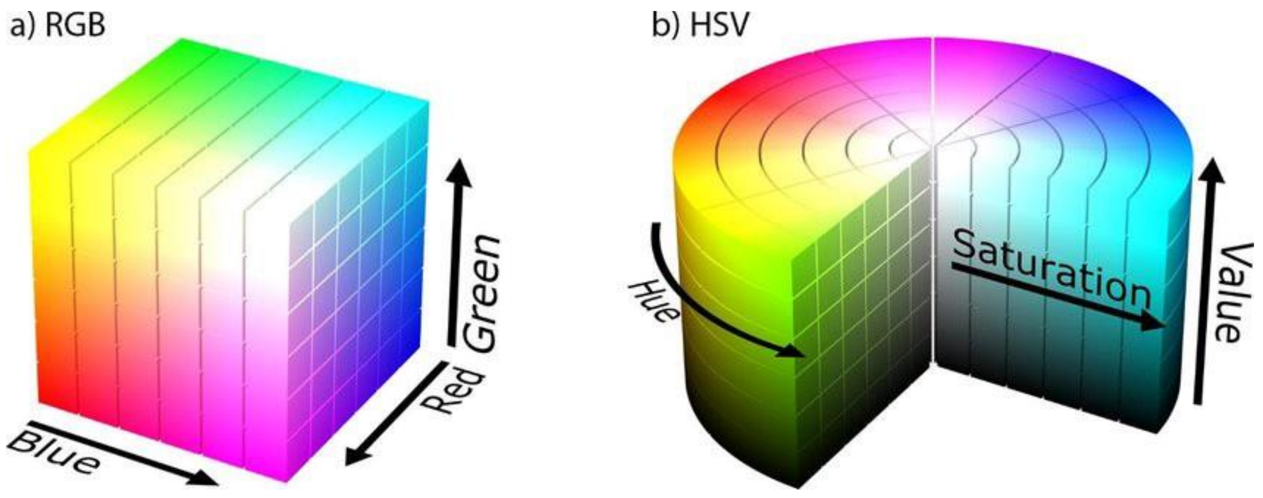
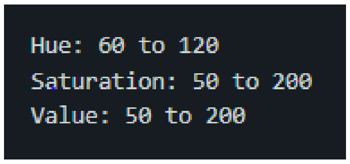


Figure 6: RGB vs HSV Color Space

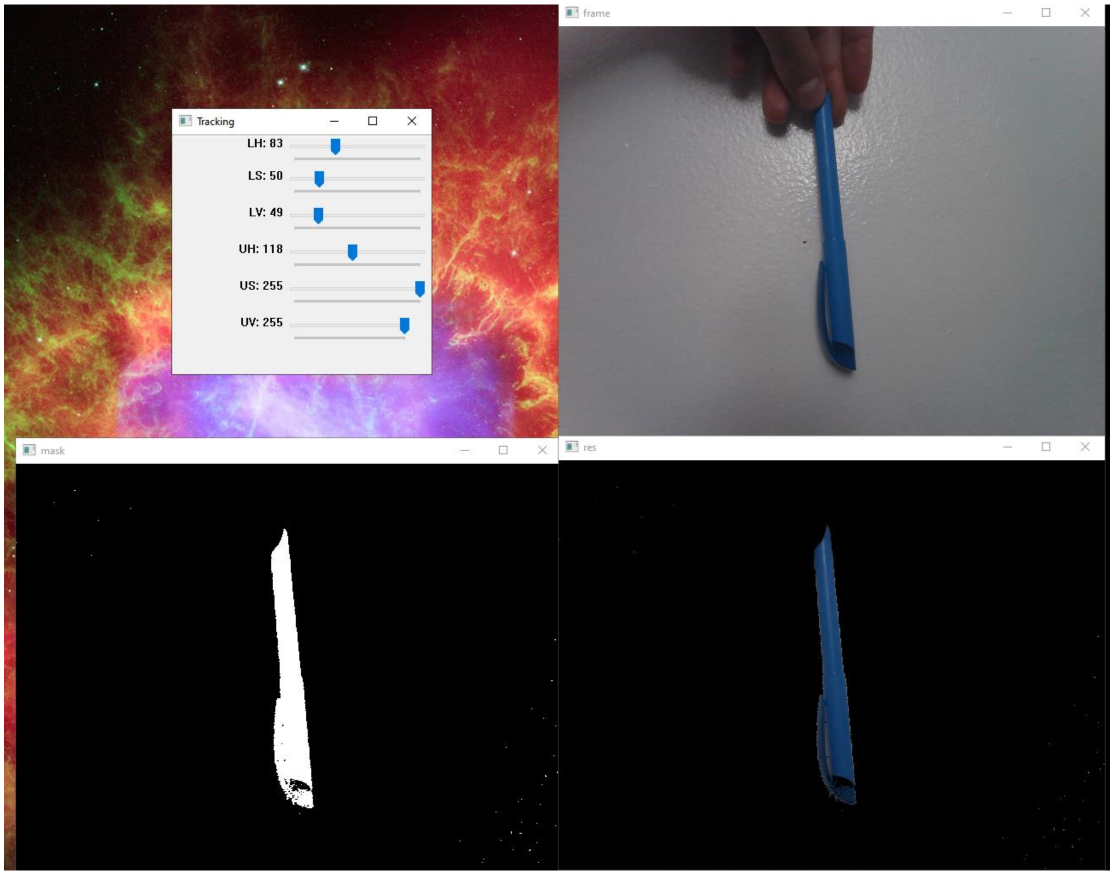
To utilize the HSV color space to isolate the object of instance in the frame, the upper and lower bounds had to be defined for each value. For instance, in order to identify and isolate objects of the color ‘blue,’ pixels with HSV values falling into the following ranges could be considered:



```
Hue: 60 to 120
Saturation: 50 to 200
Value: 50 to 200
```

Figure 7: HSV Sample Ranges

Once the thresholds were determined, a mask was created to distinguish the pixels that fell within the range by coloring them white from the pixels that fell outside of the range by coloring them black. This new black and white image then underwent a bitwise AND operation with the original frame to create an image with the isolated object of interest.



```
LH: 83
LS: 50
LV: 49
UH: 118
US: 255
UV: 255
```

Figure 8: Mask Generation Example

Next, this masking capability was expanded to detect two different-colored objects simultaneously. This was accomplished by generating two separate masks and combining them into one before undergoing the bitwise AND operation with the original frame.

To locate the objects within the frame, the concept of contours was used. A contour in this context is defined as a curve connecting continuous points along the boundary of a region of a single color. Contours smaller than a threshold were ignored to eliminate noise in detection. Since the objects chosen for this project were each only one color, each object only consisted of one contour, and the center of a contour was the same as the center of the object. Having the coordinates (x,y) of this center, we would now be able to direct the robot arm to the object's location for grasping.

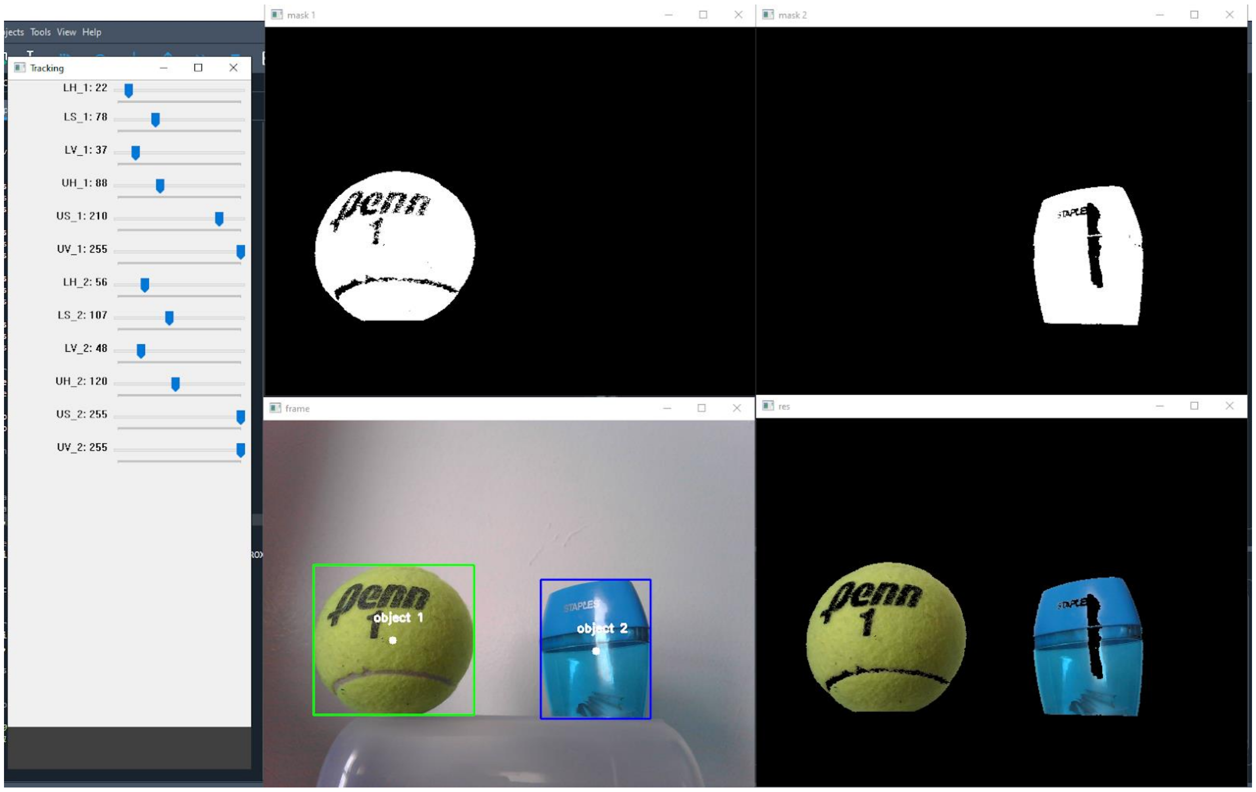

LH_1: 22
LS_1: 78
LV_1: 37
UH_1: 88
US_1: 210
UV_1: 255
LH_2: 56
LS_2: 107
LV_2: 48
UH_2: 120
US_2: 255
UV_2: 255

Figure 9: Multi-Object Mask Generation and Location

Interfacing OpenCV with Robot Arm

At this point we were ready to combine our project developments in image processing with controlling the robotic arm. Since we could not utilize MoveIt for path planning, we had to utilize existing code that would be able to calculate and execute the necessary displacements of each member of the arm in order to get the end effector to our desired coordinates. The robot arm has different operational modes, but for this project the simplest to use was coordinate mode. In this mode the user inputs x, y, and z coordinates and the robot navigates to that location. The definition of the x, y, z coordinate plane is shown below. Note that the units for the coordinates are in millimeters from the origin.

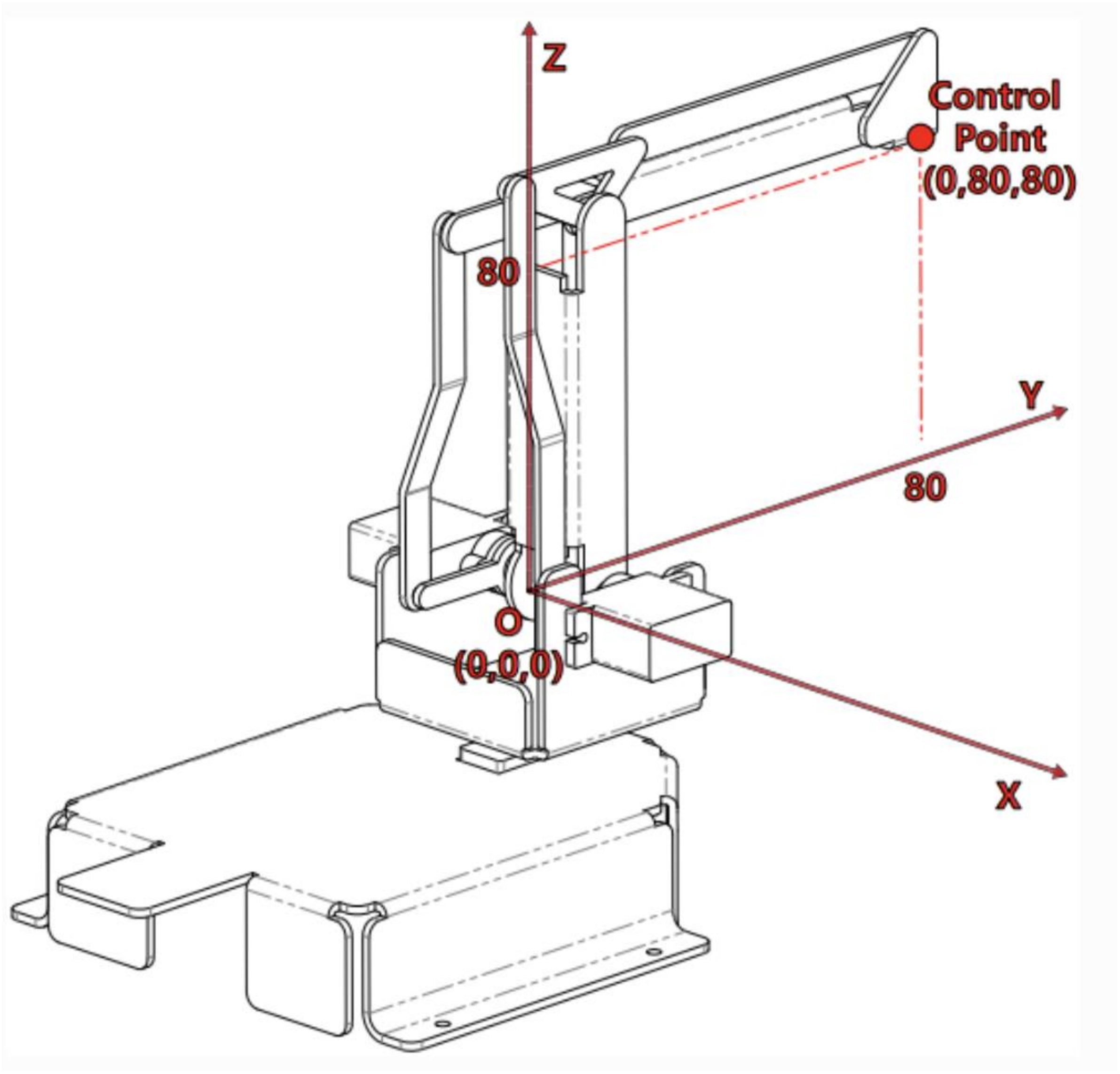


Figure 10: Robot Arm Coordinate System

An object's location within the frame data is calculated relative to the top left corner of the screen (that is OpenCVs origin point). Since the robot arm is placed at a different location than openCVs origin the calculated location is converted to be relative to the origin of the robot arm. Once we implemented and tested this code on the arm, verifying its functionality, we fed the object coordinates detected by OpenCV into motion control code, and got the arm to successfully pick up a target object based on color detection and place it in a new location. Please refer to the project's Github for a demo video on the operation of the arm.

Stack Breakdown

System Overview

The main components for this project are the webcam, raspberry pi, robot arm, and trackable objects. The webcam is mounted above the robot arm and movable objects as shown in the below figure:

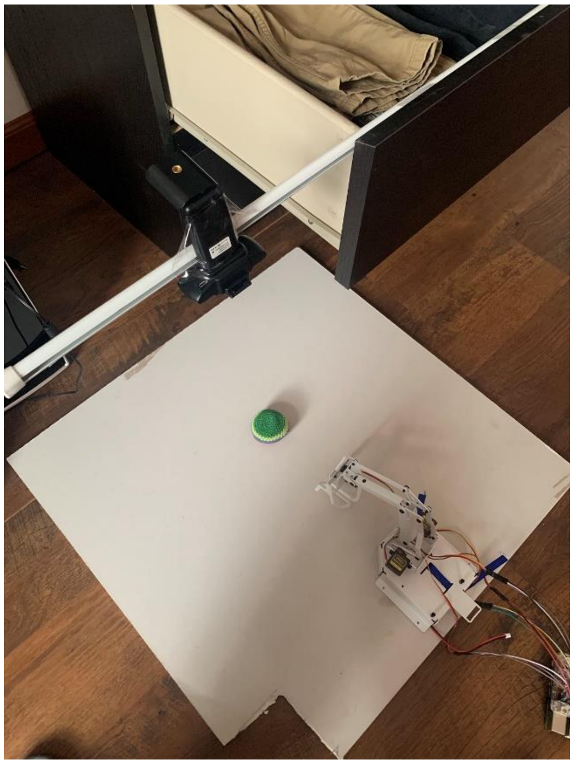


Figure 11: Demo Setup

The webcam is connected to the Raspberry Pi which runs our code. Using the aforementioned algorithms, the objects are tracked and return the location of the object in units of pixels by pixels. The location of the object is passed to the robot arm, which then navigates to the chosen object to pick it up and move it to the designated location. A block diagram of the connected system is shown below.

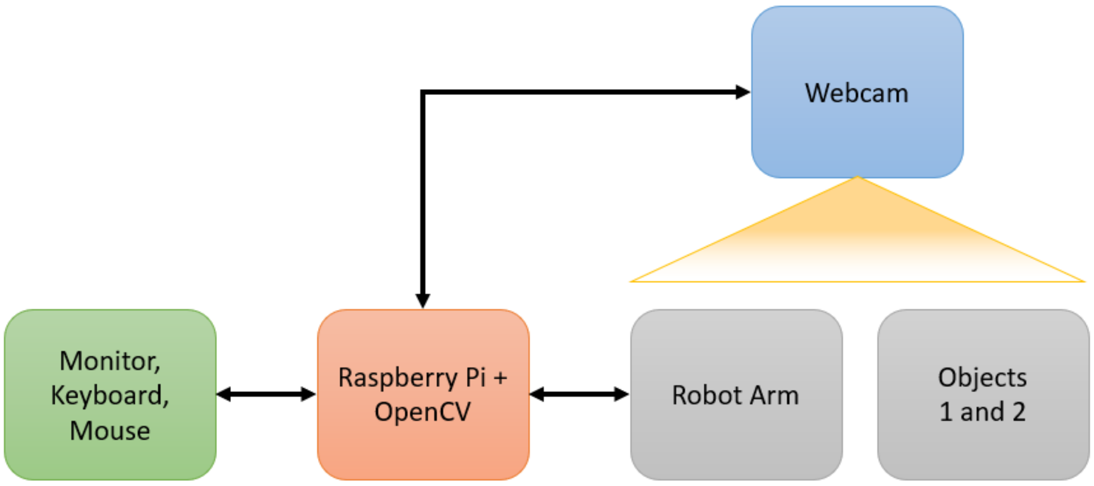


Figure 12: Tech Stack Diagram

Code Overview

- Purpose of Code: Identify two separate objects, then have the robot arm navigate to and pick up the chosen object.
- Important libraries: OpenCV, Numpy, Threading
- Mask creation
    - Using OpenCV the video stream from the webcam is captured frame by frame. The frame information is formatted as a multidimensional array that can be manipulated further.
    - The frame data is converted from the RGB color space to the HSV color space.
    - The upper and lower thresholds for defining the creation of the mask in each object is defined by adjustable trackbars that appear when running the code. Allowing for dynamic calibration of the masks.
- Position Calculation
    - With the masks tuned to the desired position, the contours functions is used to find concentrations of pixels in the mask. There will potentially be dozens or hundreds of groupings, however most will be noise. To eliminate the noise we only consider contour groupings that are greater than a certain size. Since we only one one object per mask, there should be only one contour.
    - A rectangular box is drawn around this one contour and displayed on the webcam feedback screen.
    - The position of the object within the mask is calculated in x,y position within the frame. The units of this coordinate is in pixels. The resolution of the image is 640x480.
    - The location is converted from pixels to millimeters. This conversion factor is determined by measuring the field of view of the webcam with a tape measure. With this specific setup the conversion factor to get from pixels to millimeters

    - was to divide by 1.5. As an example, a X coordinate of 320 pixels equates to 213.3 millimeters.
    - The origin of the video is the top left corner of the frame, but the commands are relative to the robot's origin. The coordinate position is converted to be relative to the robot.
    - The same process is done to calculate the position of both objects at the same time.
- Threading
    - Threads are used to perform multiple actions at the same time.
    - This was necessary to allow the video stream from the webcam to continue as the robot arm was commanded. Without threading commanding the robot arm would freeze the camera feedback until the robot arm had finished all commands.
- Picking and grabbing an object
    - The robot arm uses the x,y coordinates of the object to move directly above the chosen object, then lowers itself, closes the hand, lifts it into the air, moves over to the final location, then drops it.
    - The robot arm will not move to pick up any object until the user inputs one of the following commands:
        - If key pressed is ‘a’ object 1 is picked up
        - If key pressed is ‘d’ object 2 is picked up
    - Since the arm is continuously calculating the current position of both objects, the robot arm can accurately navigate to the object of the user's choosing and pick it up.
    - If key pressed is ‘q’ program is exited

Skills Acquired

Though we made the decision to not proceed with using ROS, the research conducted throughout the process still allowed us to gain a better understanding of the framework, and how it relates to the networking principles taught in this course. We learned that the ROS framework operates similarly to a cloud computing platform such as AWS, which we explored for Lab 2. It consists of packages and nodes that form a distributed system and talk to one another to allow the user to accomplish the desired task for their robot. Communication is executed through topics, which allow two nodes to continuously share data streams, services, which allow for synchronous client/server communication, and actions, which allow for asynchronous client/server communication and for clients to make time-consuming requests.

Furthermore, we learned how to utilize threads to run multiple processes in parallel, when the webcam image capture had to continue through the movements of the robotic arm. Although in this case the two threads were not utilizing the same resources, it was still critical to take this into consideration when considering whether the threads had to communicate with one another. We also gained experience in programming on hardware with more limited resources, such as the Raspberry Pi, and considering the consequences and possible optimizations that could be made to mitigate those consequences. Finally, we gained a foundational understanding of how to use OpenCV for image processing and pairing that capability with robotic control, as can be seen in the Technical Achievements section of the report.

Resources

https://github.com/bobbyg31425/EC544_Final_Project

https://roboticsbackend.com/what-is-ros/

https://discourse.ros.org/t/ros-based-opencv-image-detection-grasping-with-arduino-braccio-arm/15635

https://ubuntu.com/download/raspberry-pi

https://roboticsbackend.com/using-ros-on-raspberry-pi-best-practices/

https://automaticaddison.com/working-with-ros-and-opencv-in-ros-noetic/

https://ros-planning.github.io/moveit_tutorials/doc/getting_started/getting_started.html#build-your-catkin-workspace

http://wiki.ros.org/TestingRepository

https://ros-planning.github.io/moveit_tutorials/doc/quickstart_in_rviz/quickstart_in_rviz_tutorial.html

https://opencv.org/

https://www.realvnc.com/en/connect/download/viewer/

https://docs.python.org/3/library/threading.html

https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html

https://piarm.readthedocs.io/en/latest/