GENG5508 Project Report
Pick and Place UR5 Robot Implementation

Luke Carpenter - 22110274

Harrison Grout - 23077163

Sean Stepehenson - 22239862

Talin Taparia - 22298002

Introduction

This project seeks to implement a pick-and-place program using the URX Python library[1]. The project will include both a computer vision component and pick-and-place component. The computer vision component will be used to take visual information from the current location of the UR5 robot and extract useful information about object coordinates. The pick-and-place component will be used to take coordinate information and map it to the real world.

The software libraries, and their uses, that are used in this project are:

- urx: allows for interaction with the robot.
- OpenCV: provides image processing functions.
- math: provides mathematical operations.
- matplotlib: for displaying images.
- numpy: provides various mathematical operations and data manipulation.
- Pillow: provides image processing functions.
- io: for dealing with various input/output (I/O) operations.

The hardware used in this project are:

- The Universal Robot arm (UR5e)
- RobotIQ Gripper attachment
- UR5e camera
- Two laptops
- A wireless router
- Environment objects

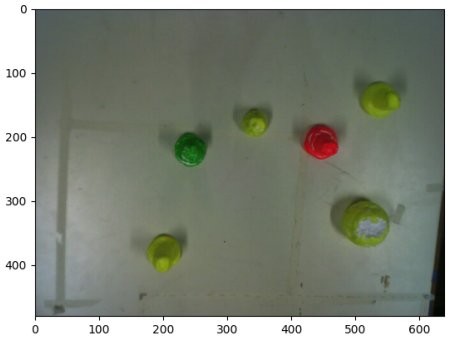
The RobotIQ 2F-85 gripper[2] is an attachment for the universal robot capable of picking up and placing objects. It has a grip force of anywhere from 20 to 235 N and has a maximum payload of about 5kg. It has two members which are controllable with the URX library. These members are pushed inward to grip an object and are loosened to release the object. The limitations of the 2F-85 series adaptive gripper are that it cannot easily grip slippery surfaces (payload is tested on steel) and can only apply force in a 2D plane.

[1] The library's source code can be found at: https://github.com/SintefManufacturing/python-urx.
[2] The gripper's user manual is available at: https://assets.robotiq.com/website-assets/support_documents/document/2F-85_2F-140_Instruction_Manual_e-Series_PDF_20190206.pdf.

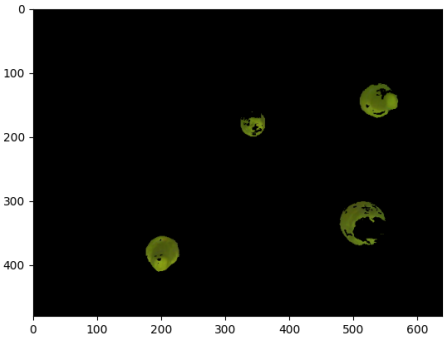
Computer Vision

The computer vision component of the project was used to take the image from the URE5 camera and process it such that the location of the desired object was returned. This involved several smaller computer vision operations including binarization, thresholding, corrosion, dilation and contouring. The pipeline of these operations is explained in sequence below.

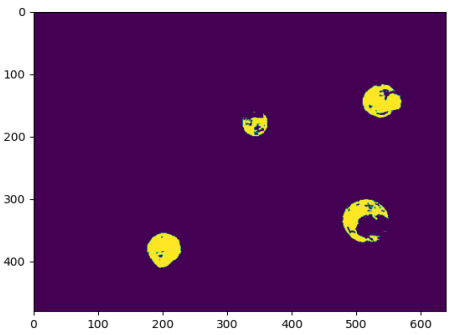
The following images detail the entire computer vision pipeline. Where image 1 is the raw image from the camera, image 2 is the thresholded image to only include yellow pixels, image 3 is the binarization step, image 4 is the dilation to remove spots, image 5 is the corrosion to prevent center point skewing and image 6 is the final axes flip and grid allocation.



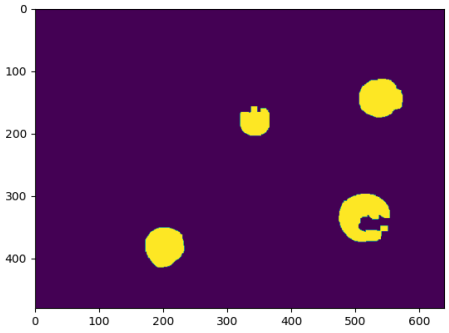
(1)



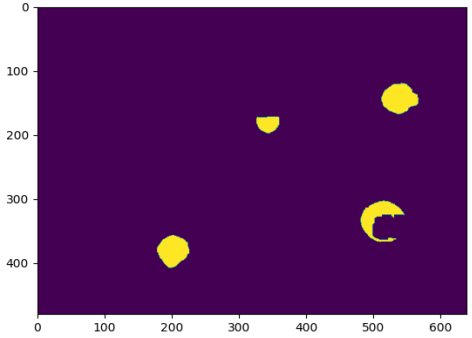
(2)



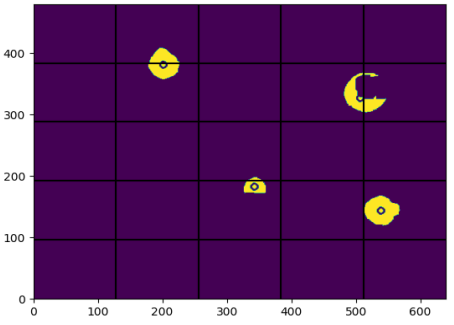
(3)



(4)



(5)



(6)

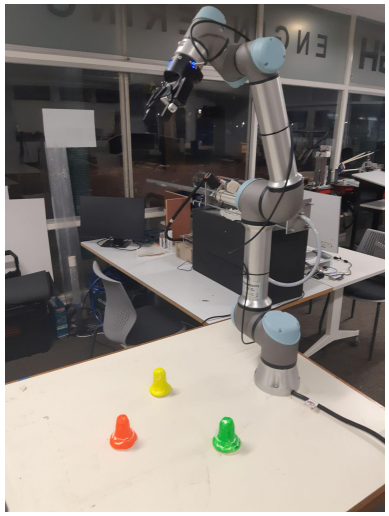
A full list of the purpose of these steps can be found in the program flow section. The output of this pipeline should be twofold: the exact coordinates of an object to pick up and the corresponding grid the object resides which will determine the offset applied to the pickup operation. This offset will be further explained in the pick and place section

Pick and Place

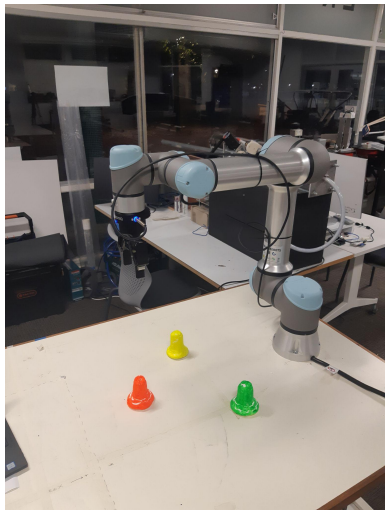
The pick and place component receives a single [x, y] coordinate set and must find, pick up, carry and place the object into the bucket corresponding to the color of the object. The selection of images shown below demonstrate the following eight main steps involved in this process:

1. The robot is in the “camera pose” and undergoes the computer vision step.
2. The robot returns to the “default pose” with the desired coordinates still stored in the system.
3. The arm translates to the pose directly above the desired object.
4. The tooltip is lowered a fixed distance to be around the object.
5. The tooltip members push together and grip the object.
6. The robot moves to the relevant bucket.
7. The robot drops the object into the bucket.
8. The robot returns to the “default” pose ready to repeat the process.

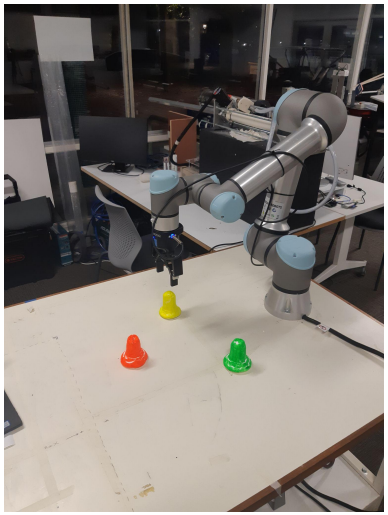
An important consideration for the pick and place operation is the small offset required based on each object's horizontal distance to the camera. Specifically, since the camera is not located at a perfect birds-eye view to each object, the calculated center points will be subject to some error depending on the angle between the line perpendicular to the camera and the object. Image 6 in the computer vision sections shows the process in which each grid has a manually calculated offset applied to it which is able to consistently handle this error.



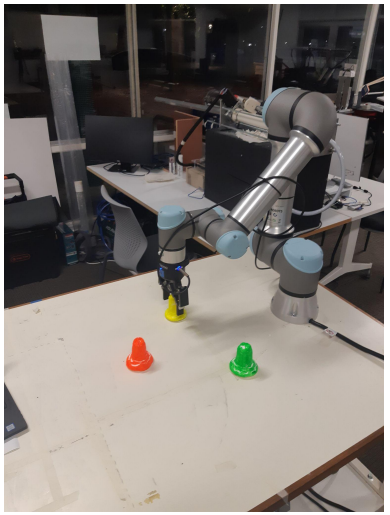
(1)



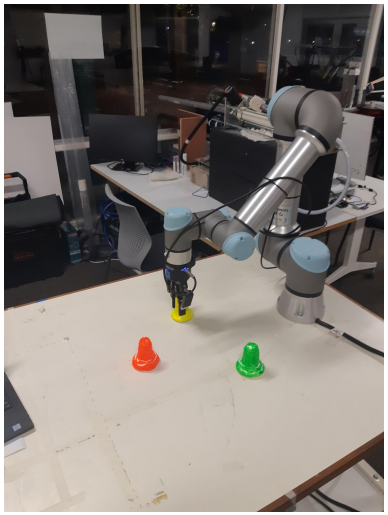
(2)



(3)

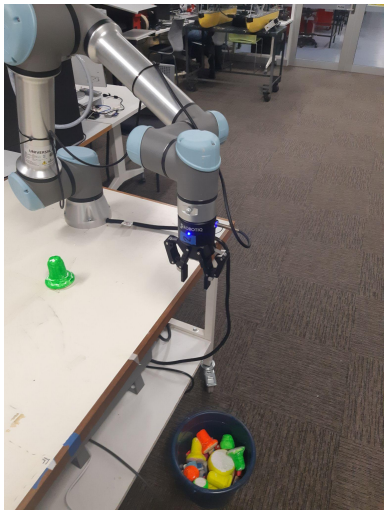


(4)

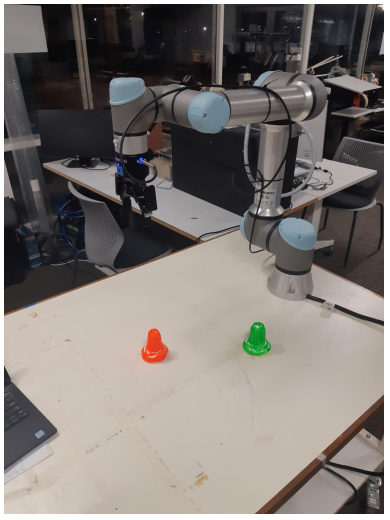


(5)

(6)



(7)



(8)

Program Flow

The overarching flow of the program will continue to scan and pick up objects while objects are still in the environment to do so. This is encapsulated with the following pseudocode:

```
# GENG5508 PseudoCode
# Overall program flow
while objects_in_environment > 0:
    default_pose() # Center of table relative to base
    camera_pose() # Highest easily reachable point for camera
    get_image_centers() # Computer vision component
    pick_and_place() # Find, pickup and place object in bucket

```

The default pose functionality of the program seeks to ensure that the forward kinematics of the robot do not bump into buckets or other hazards in the environment. There are several edge cases discovered during the testing of this project which would cause the kinematics calculations of the URE5 robot to path strangely if:

1. Members were close to their maximum distance from the base.
2. Joints were a full 360° rotation from the original position.
3. The robot was not aware of an intermediate obstruction.

These cases caused the group to seek a reliability solution in the form of a default pose which would be in the center of the environment. This was useful because it allows the group to utilize the fact that any movement the robot makes in the environment can be reversed without any of the described edge cases occurring.

The camera pose functionality is intended to give the URE5 camera the largest area of which to see the environment. The exact position was intentionally chosen to align with the table edges so that the robot doesn't accidentally take in obstructive information from the floor around the table. From the default pose, the robot should simply increase the height of the tooltip and align the camera to be perpendicular with the table.

The functionality responsible for getting the coordinates of the center point of a desired image uses computer vision to analyse the image and extract the desired information. The steps involved are described with the pseudo code below:

```
# GENG5508 PseudoCode
# Computer Vision Component

def get_image_centers():
    > Take picture from URE5 camera
    > Threshold image using green, yellow and orange
    > Dilate, then corrode the object
    > Identify remaining objects above a certain size
    > Find contours and the corresponding center point
    return center_point of a single object
```

The pick and place functionality uses the manually measured area of the environment along with the center point to determine where in the real world the object is. The exact process for doing this is detailed in the pseudocode below:

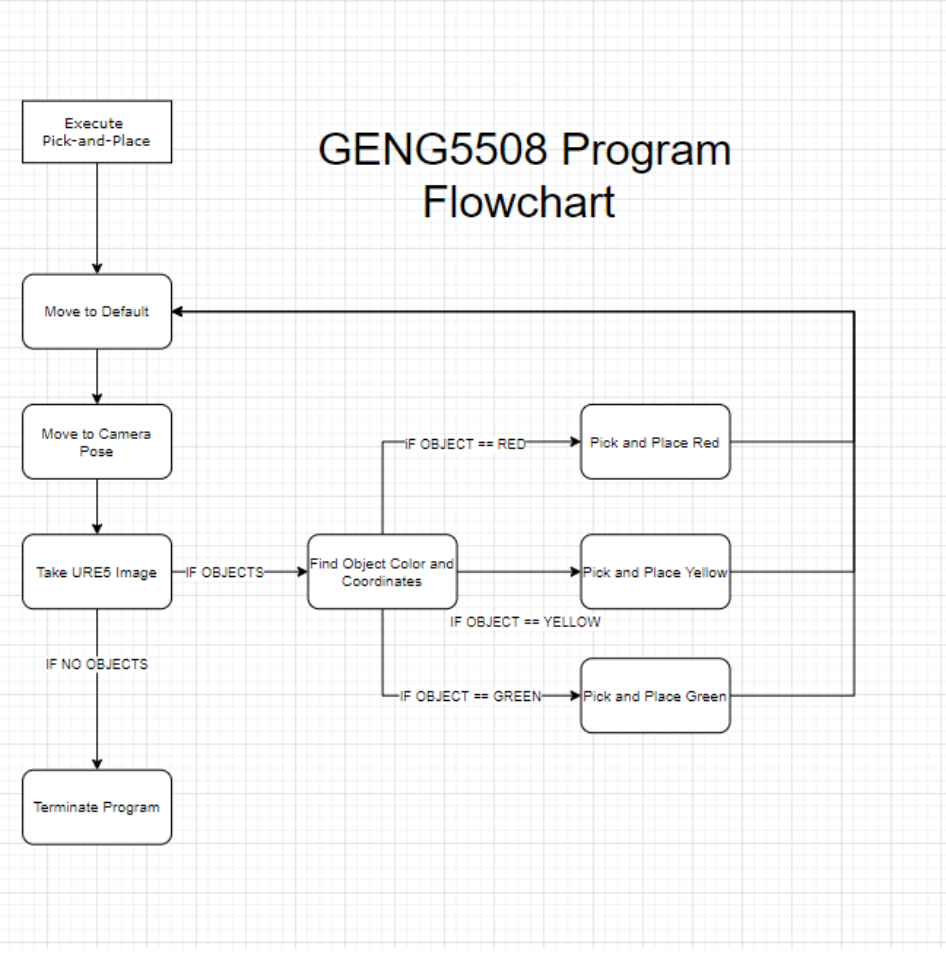
```
# GENG5508 PseudoCode
# Pick and Place Component

# Manually measured environment size values
env_width = 100
env_height = 100

# x and y coords of buckets, predefined
bucket_red = 1,1
bucket_green = 2,2
bucket_yellow = 3,3

def pick_and_place(object_center, color):
    real_world_x = object_center[x]/env_width # Get object x location in cm
    real_world_y = object_center[y]/env_height # Same for y
    robot_move(real_world_x, real_world_y)
    pickup()
    # Depending on color of object
    robot_move(bucket_red)
    drop()
```

The following table gives more nuanced reasoning behind each function listed in the above pseudocode diagrams and the reasons they were implemented.

Function	Explanation
URE5 Picture	The image must be accessed through a web browser and formatted into a format which the openCV library will accept for further processing.
Image Thresholding	Scanning the image for pixels which has a pixel RGB value corresponding to the desired color. This was also

	implemented using HSV format but later removed. The output is an image only containing the pixels in the specified color range.
Binarization	After color is established, the RGB format only seeks to complexify the problem and the image can be converted to only contain 0 (not specified color) and 1 (specified) color.
Dilation	Since the styrofoam objects are not completely colored, the binarized images will often be spotty. Dilating the image applies a mask which adds pixels to all boundaries of the image. This fills in empty spots and helps the contour calculation later.
Corrosion	Since the dilation process fills in blank spaces and widens the boundaries of the image, corrosion is performed to stop the image becoming too large and extending off screen.
Contour identification	A contour is a continuous string of points used to form the boundary of a given object. Contours can be combined to decide where a likely center point will be.

The previously given pseudo can be combined into a high level flowchart of the program flow which is given below.



Conclusion

The group was able to use a combination of computer vision and the urx library to implement a functional pick and place program which can remove all objects colored orange, yellow and green from the environment and place them into their corresponding buckets. The computer vision component of the project utilized thresholding, binarization, dilation, corrosion and contouring to return the center points of the desired objects. The pick and place component used the urx library in combination with several manually calculated offsets to convert image coordinates to real world coordinates.

Work Breakdown

The following is a tabulated summary of individual group members' contributions to the project.

Group Member	Contributions
Sean	● Attended most project sessions ● Implemented object-detection of all objects using HSV image format ● Implemented circle detection and centre-point finding (proof-of-concept) ● Contributed to user manual ● Edited design report and user manual
Talin	● Computer vision algorithm- colour detection, masking, contours and object centre ● Wrote user manual ● Contributed and edited project report
Luke	● Wrote design report ● Wrote user manual ● Wrote computer vision and pick-and-place code ● Attended most project sessions
Harrison	● Wrote pick and place code ● Wrote computer vision pipeline ● Wrote user manual ● Contributed to design report ● Attended every project session