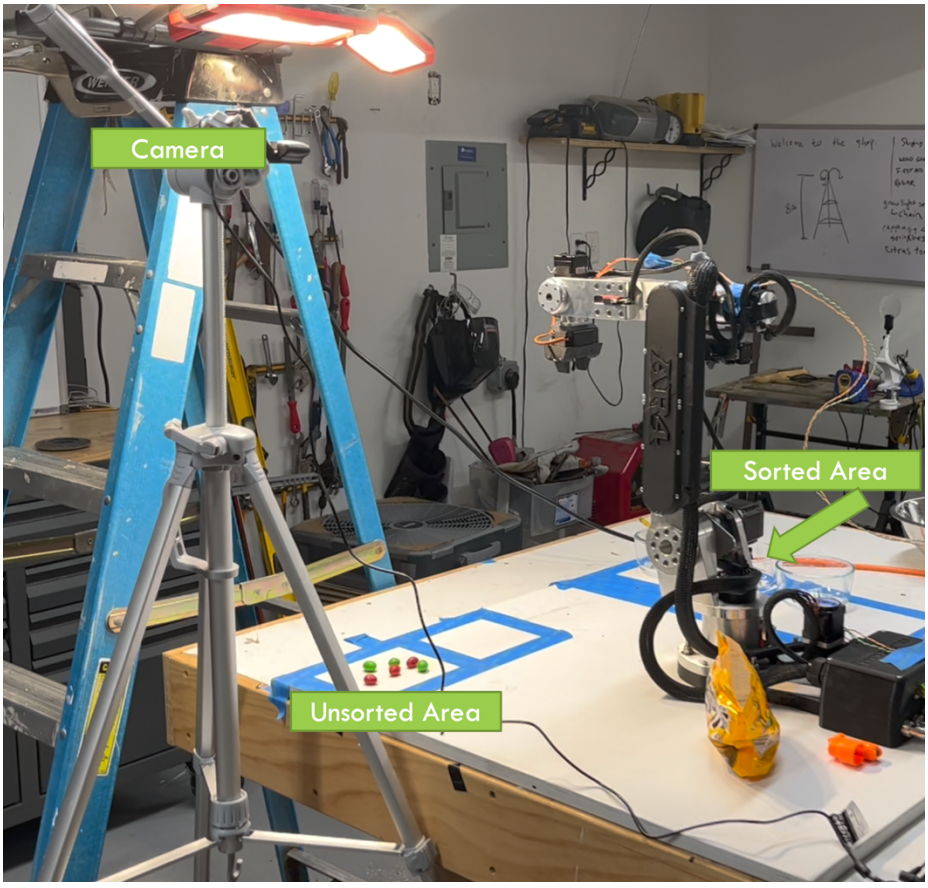
Robert Murrer
EML6805 – Project 3
December 1, 2022

AR4 Robotic Arm M&M Sorting

Description

This is the final installment of the Annin Robotics AR4 Arm project this semester. In the previous reports the assembly of the robot and basic pick and place procedures were discussed. This paper will present a sorting application with the arm using a vision system. The platform was able to successfully sort 2 colors of peanut M&M candies.

Vision System



Hardware

- Logitech C920S HD 1080p Webcam
- Tripod & Ladder for Rigging + Clamps
- LED Light for consistency

Software

- Python 3.11 setup with Virtual Environment
- Python Packages:
    - OpenCV 4.6
    - Numpy 1.23.5
    - Jupyter Notebook 6.5.2
    - Matplotlib 3.62

Object Detection

Several methods were tried to extract specific objects from the image frame taken from the webcam. Blob detection with OpenCV was considered but the method of sorting one color first and then the other was problematic. The final method used was creating masks for each color needed be sorted and calculating contours within an area range.

To do color detection Hue Saturation Value (HSV) color space is used. Masks are generated by applying a threshold of the lower and upper limits of the color. This mask is then made bigger with a 5x5 dilation to ensure the entire object is enclosed in the mask object. Then a bitwise and is made to remove all objects from the image that are not visible in the mask. This is then passed to a contour finding algorithm that finds arcs and shapes of the colored objects in image.

```
red_lower = np.array([0, 200, 150], np.uint8)
red_upper = np.array([10, 255, 255], np.uint8)
green_lower = np.array([25, 52, 72], np.uint8)
green_upper = np.array([102, 255, 255], np.uint8)
```

These values were originally calculated at night and then on trying the next day the vision system was having trouble detecting. A light was added to make vision more consistent regardless of the environment.

Camera setup was important also. Autofocus was disabled along with automatic white balance feature. Having the camera as close as possible to the table to create high-enough resolution of the unsorted area and having the camera lens as parallel as possible to the table was critical for calibration. The cropping of the image was also important to reduce false positives from the environment that the robot could not reach.

I originally had the camera setup which would cause the camera pixels to coordinate with negative robot coordinates. The HMI software failed to do this math correctly, so it is advised to place the camera to the left of the robot as in the picture above.

```
def get_image():
    _, imageFrame = cap.read()
    imageFrame = imageFrame[0:205,210:400] #adjust crop  as needed
    return imageFrame
```

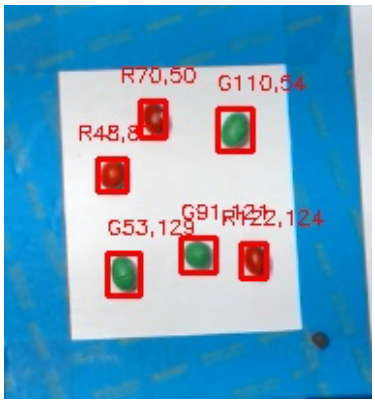
```
def get_shapes(imageFrame, mask):
    mask = cv2.dilate(mask, kernel)
    res_color = cv2.bitwise_and(imageFrame, imageFrame, mask = mask)
    contours, hierarchy = cv2.findContours(mask,
                                           cv2.RETR_TREE,
                                           cv2.CHAIN_APPROX_SIMPLE)
    return contours
```

```
def get_sorted_contours(imageFrame):
    hsvFrame = cv2.cvtColor(imageFrame, cv2.COLOR_BGR2HSV)

    red_mask = cv2.inRange(hsvFrame, red_lower, red_upper)
    green_mask = cv2.inRange(hsvFrame, green_lower, green_upper)
    orange_mask = cv2.inRange(hsvFrame, orange_lower, orange_upper)

    contours_red = get_shapes(imageFrame, red_mask)
    contours_green = get_shapes(imageFrame, green_mask)
    contours_orange = get_shapes(imageFrame, orange_mask)

    return contours_red, contours_green
```

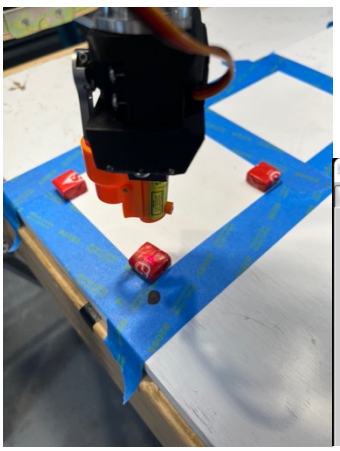


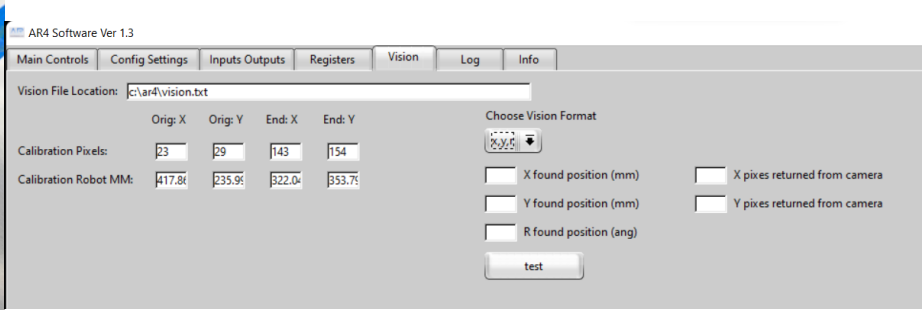
Calibration

With the objects being detected in the camera image, it is now time to associate the pixel images from the image to AR4 robot coordinates.

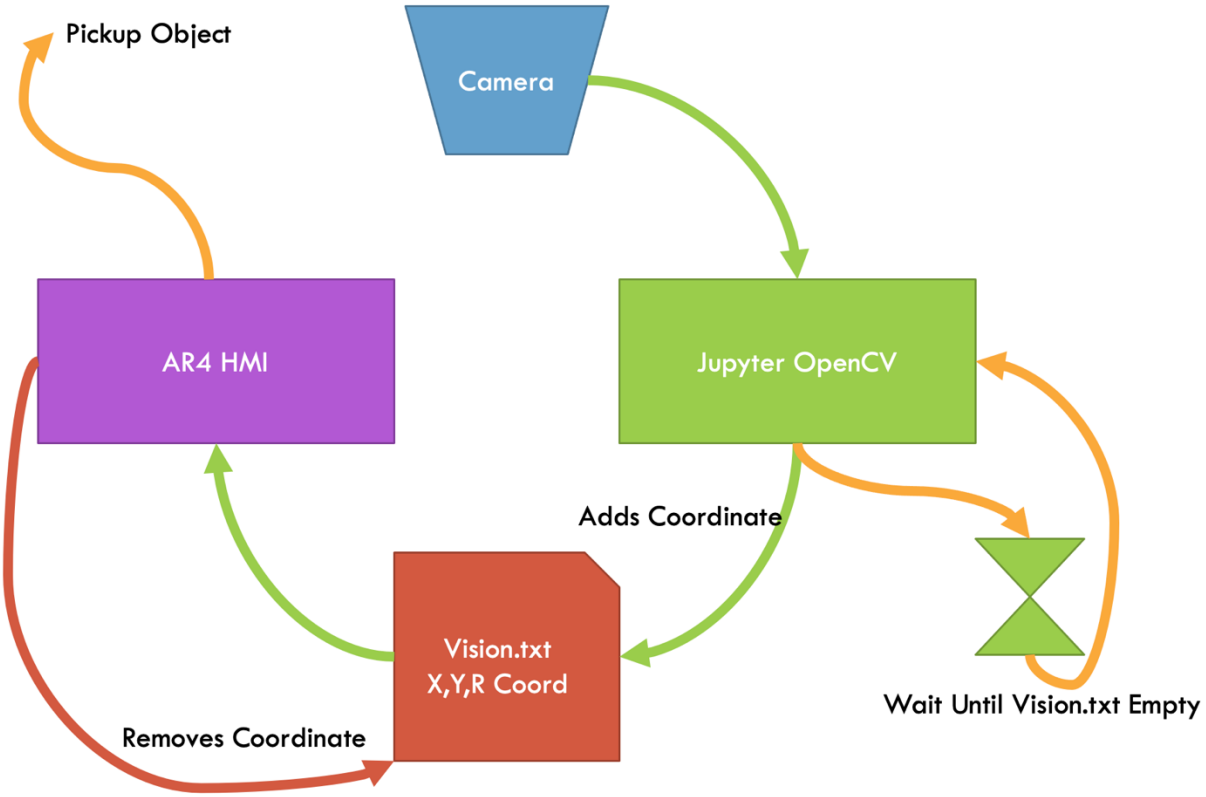
Procedure

1.) Orient Camera so that X & Y coordinates correspond to robot coordinates. In the AR4, with the robot at rest position the forward and backward motion is in the X direction, side to side is Y
2.) A rectangle is used to calibrate pixel coordinates to the robots. An object is placed at the origin as well as the opposite corner which will act as the largest X,Y
3.) Object Detection is done, and the coordinates are noted in the AR4 Human Machine Interface (HMI) GUI's vision tab
4.) The robot is then jogged to each of these points and its current X,Y coordinates are associated with pixel coordinates. A laser pointer can be used enclosed in jaws of gripper to help with this procedure
5.) Finally, the Z position is recorded above and below the object pickup point to give the robot a solid target to pick up





Vision Integration and Synchronization



There was a challenge of how to coordinate which color was being picked up. Originally it was thought that the vision register could be used to pass information from the OpenCV program to the HMI. This was not the case.

To avoid modifying the HMI software, which was possible and will be done in future, to support this feature the count of the objects to be sorted must be fixed and of equal number.

The vision.txt was the coordinate method which is a simple text file that contains the pixel coordinates of the object to be picked up. When the HMI runs its GetVision command the file is emptied. The OpenCV program then notices this change and applies adds the next coordinate to the file. This process repeats until no objects are left.

Jupyter OpenCV Notebook

```
    def spin_until_vision_file_empty():
        while True:
            fileObject = open(vision_file, 'r')
            data = fileObject.read()
            fileObject.close()
            if (data == ""): return

    def write_coords(contour):
         x, y, w, h = cv2.boundingRect(contour)
         with open(vision_file, 'w') as f:
            f.write(str(x) + "," + str(y) + ",0")

[ ]  for pic, contour in enumerate(contours_red + contours_green):
         area = cv2.contourArea(contour)
         if(area > min_area and area < max_area):
             spin_until_vision_file_empty()
             write_coords(contour)
```

Because multiple contours could belong to the same object, and minimum and maximum area is defined to filter out all but the peanut M&Ms.

Normally it is not a great option to synchronize via the file system two programs. This is because the system calls to read and write files are not guaranteed to be atomic or ordered. In this case it worked out but in the future the plan is to modify the HMI interface to communicate via socket.

Full Notebook can be found here: https://drive.google.com/file/d/13RNPV10qxd83St-mk2giDydSn8yB2fpS/view?usp=sharing

AR4 Human Machine Interface (HMI) Program

Version 1.3 was used for the duration of this project this semester. Although a new version of the HMI was released, none of the features changed mattered to this particular project.

##BEGINNING OF PROGRAM##	
Tab Number 1	
Move J [*] X 322.984 Y 0.000 Z 474.880 Rz 0.035 Ry 89.936 Rx 0.035 Tr 0.0 Sp 25 Ac 10 Dc 10 Rm 50	Home Position
Position Register 1 Element 4 = -20	
Position Register 1 Element 5 = 80	Set Hand Position Registers
Position Register 1 Element 6 = 32	
Position Register 2 Element 1 = 0	Clear Position Registers for Offsetting
Position Register 2 Element 1 = 0	
Position Register 2 Element 2 = 0	
Position Register 2 Element 3 = 0	
Position Register 2 Element 4 = 0	
Position Register 2 Element 5 = 0	
Position Register 2 Element 6 = 0	
Register 1 = 0	Zero Out Pickup Count Register
Tab Number 2	
Move J [*] X 322.984 Y 0.000 Z 474.880 Rz 0.035 Ry 89.936 Rx 0.035 Tr 0.0 Sp 75 Ac 10 Dc 10 Rm 50	Home Position
Servo number 0 to position: 55	Open Gripper
Position Register 2 Element 3 = 15	Set Z Offset Position 15mm
Get Vision	Get Vision Coordinates
Position Register 1 Element 3 = 110	Set Z Table Position (get reset by vision)
OFF PR [PR: 1] offs [*PR: 2] [*] Tr 0.0 Sp 100 Ac 10 Dc 10 Rm 50 $ N	Move Arm Right Above Object
Position Register 2 Element 3 = 0	Move Arm To Table
OFF PR [PR: 1] offs [*PR: 2] [*] Tr 0.0 Sp 25 Ac 10 Dc 10 Rm 50 $ N	Close Gripper
Servo number 0 to position: 0	
Position Register 2 Element 3 = 15	
OFF PR [PR: 1] offs [*PR: 2] [*] Tr 0.0 Sp 25 Ac 10 Dc 10 Rm 50 $ N	Move Arm Right Above Object (2 Moves)
Position Register 2 Element 3 = 100	
OFF PR [PR: 1] offs [*PR: 2] [*] Tr 0.0 Sp 100 Ac 10 Dc 10 Rm 50 $ N	
Move J [*] X 322.984 Y 0.000 Z 474.880 Rz 0.035 Ry 89.936 Rx 0.035 Tr 0.0 Sp 100 Ac 10 Dc 10 Rm 5(	Home Position
Register 1 = ++1	Increment Pickup Count Register
If Register 1 = 4 Jump to Tab 4	
Tab Number 3	
Move L [*] X 408.455 Y -362.396 Z 184.266 Rz 37.603 Ry 98.853 Rx -29.272 Tr 0.0 Sp 75 Ac 10 Dc 10	Red Dropoff
Servo number 0 to position: 55	
Jump Tab-2	
Tab Number 4	
Move L [*] X 261.729 Y -360.877 Z 189.624 Rz 37.280 Ry 97.123 Rx -29.622 Tr 0.0 Sp 75 Ac 10 Dc 10	Green Dropoff
Servo number 0 to position: 55	
Register 1 = 3	Hack To Force Green Next…
Jump Tab-2	

To increase the number of items to be sorted modify the If Register 1= 4 to be the number being sorted. Also the setting of the Register 1 = 3 in green drop off will also need to modified.

Results & Analysis

The choice to try and pickup peanut M&Ms had some consequences that in the end made the task of sorting a bit more difficult. I was able to sort successfully when the robot could pick up the objects. Original Starburst candy was used but was abandoned because the rotation data was not being read correctly from the vision.txt file.

The speed at which the robot moved was changed to 75% of its maximum speed to try and reduce time in sorting. This caused some table vibrations, and the ramp and acceleration

features of movement could have been tuned to minimize the disturbances. The setting of the speed to 100% caused motor errors for some of the movements.

The choice to only have two colors was because of the false positive rate I was receiving. The mask tuning became tedious and the lighting of the room as well as shadows cast were consistently causing a failure to identify the objects correctly. To improve this in the future I think would do a proper blob detection first and do a color averaging algorithm or perhaps train a support vector machine to properly identify the M&Ms.