LAB 5

Camera Sensing and Integration into the World Frame for a Pick and Place Task

5.1 Important

Read the entire lab before starting and especially the "Grading" section so you are aware of all due dates and requirements associated with the lab.

5.2 Objectives

This is the capstone lab of the semester and will integrate your work done in previous labs with python, ROS, and forward and inverse kinematics. It will also introduce you to some basic OpenCV techniques. In this lab you will:

- Use OpenCV functions to find and mark the centroid of each colored block.
- Develop transformation equations that relate pixels in the image to coordinates in the world frame
- Report the world frame coordinates (x_w, y_w) of the centroid of each block in the camera's view.
- Move the block to a predefined position in the robot's work area.

5.3 References

- Appendix B which explains how to derive the intrinsic and extrinsic equations for the camera.
- HSV Color Space:
  - `https://en.wikipedia.org/wiki/HSL_and_HSV`
  - https://stackoverflow.com/questions/10948589/
  - Converting RGB to HSV
- Simple Blob detector:
  - `www.learnopencv.com/blob-detection-using-opencv-python-c/`
  - https://stackoverflow.com/questions/8076889/
  - `https://www.programcreek.com/python/example/89350/cv2.SimpleBlobDetector`
  - `https://www.programcreek.com/python/example/71388/cv2.SimpleBlobDetector_Params`

5.4 Tasks

5.4.1 Color Thresholding and Object Centroids

In this lab, we will be using some built in functions in the **OpenCV** library to locate blocks based on their color and find their centroids. **OpenCV** is a powerful set of open source tools that are useful for computer vision tasks. They will allow us to efficiently complete tasks without having to develop the algorithms ourselves.
Thresholding is the way we will isolate the objects that we are interested in by color. We will be using the HSV color space to accomplish this. HSV stands for Hue, Saturation and Value. Using **OpenCV**, we will mask out any objects that don't fall within the desired HSV range that we set. This allows us to focus only on the green and yellow blocks that we wish to locate.
Side Note: What is the H,S,V color space? Please see the reference link to the Wikipedia page discussing color spaces. You are probably most familiar with the R,G,B (red,green,blue) color space. Here each pixel of an image has a R,G,B value. With H,S,V colors are placed on a color wheel and you indicate which color you are interested in by selecting an angle range. In **OpenCV** the range of the angle is from 0 degrees to 180 degrees. (You will find other implementations that use 0-360). For example a blueish color would be in the angle range of 110 degrees to 130 degrees. The S value is a number between 0 and 255 with 0 being very white and washed out and 255 the full, clear color. The V value is also a number between 0 and 255 with 0 being very dark and shaded and 255 the full clear color. The main reason we use the H,S,V color space in this lab

is that it makes finding a color with different shading and lighting conditions much easier.
Once we have thresholded the image, we are left with ‘blobs’. Blobs consist of the groups of pixels that were not masked out by thresholding. No matter how well we perform thresholding, there will always be noise in the image. We only really care about the blobs that correspond to the blocks we wish to find and the **OpenCV** function **SimpleBlobDectector** can help us do that. It allows us to filter out blobs based on features like size, roundness, eccentricity, etc. In addition, once we have narrowed down to the blobs we care about, it can help us find the centroid of these blobs.

5.4.2 Camera Calibration and Transformation

The problem of camera setup is that of relating (row,column) coordinates in an image to the corresponding coordinates in the world frame (x_w, y_w, z_w). A camera transformation equation will be developed to allow us to perform this conversion. Several parameters must be specified in order to implement the equations. Specifically, we are interested in θ the rotation between the world frame and the camera frame and β the scaling constant between distances in the world frame and distances in the image and T_x and T_y the relationship between the origin of the camera frame and the world frame.
Appendix B will guide you through the development of the camera transformation equations and explain the camera calibration process.

5.4.3 Pick and Place

Your program should:

- Move the Robot out of the way so that the camera can view the entire work area in order to find the blocks randomly placed in that area.
- Use **OpenCV** functions to find the centroid (x_w, y_w) of each block.
- Display a center point for each discovered block in the video display window.
- Command the robot to pick up each block one at a time and place it in the appropriate goal position.
- When finished with all blocks exit from the program.

5.5 Procedure

5.5.1 Lab Set Up

Start as you normally do by downloading the Lab 5 starter files from the lab website, unzip them and place them in your **src** directory along with the rest of the lab packages.

If you look inside **lab5pkg_py/scripts**, you will see a number of Python files, but we will only be editing a few during this lab. A brief description of each file is provided here:

- **lab5_exec.py** - The main executable. It contains all the code to move the arm. You will edit this to complete the pick and place task.
- **lab5_blob_search.py** - This contains all the code related to image thresholding and finding blobs and centroids, as well as converting pixel coordinates to world coordinates. You will edit this to locate the desired objects and return their positions in world coordinates.
- **lab5_func.py** - This contains the forward and inverse kinematics code. You should edit to add your solutions from Lab 3 and 4.
- **lab5_header.py** - This is a standard header file with imports and constant definitions. There is no need to edit this.

5.5.2 Running the Program

To run the code, open a terminal window and in your catkin working directory:

```
$ source devel/setup.bash
$ roslaunch ur3_driver vision_driver.launch
```

Then open a second terminal window and in your catkin working directory:

```
$ source devel/setup.bash
$ rosrun lab5pkg_py lab5_exec.py
```

When you run the program, you will see two image windows pop-up. One is the color or raw image. The other is a black and while masked image that will change based on your threshold values. Note that sometimes the two windows will appear on top of each other making it appear that one is not present. Simply move the top window to see both.

5.5.3 Threshold camera image to distinguish the colors of choice

In **lab5_blob_search.py**, in the function **blob_search(...)**, you will edit the following to locate the green and yellow blocks. They are currently set to threshold blue colored objects.

```
lower = (110, 50, 50)   # Blue lower
upper = (130, 255, 255) # Blue upper
```

Selecting the correct values for thresholding can be rather challenging and requires some trial and error. There are several methods that you can use to find these values. One is to review how the HSV color space works and select an appropriate Hue value. Once that is selected, you should adjust Saturation and Value until a clean image is created. If necessary, you may need to widen or

narrow your Hue range. Another method is to convert the RGB value given in the image window to an HSV value. Note the RGB values displayed in the lower left hand corner of Figure 5.2. They represent the RGB value of what ever pixel the mouse pointer is hovering over. Using these values in a website like the one listed in the references will allow you to get a starting point for your HSV ranges. Pay attention to how the HSV value is represented on the website compared to the way it is represented in **OpenCV**. Note that you will have to create a different set of ranges for for each color and select the appropriate range depending on what color you are looking for.
When successfully completed, you will see a black and white image with white shaped blobs that have little 'noise' inside their boundaries. It is okay to have more than one blob in the image as long the block we desire is seen as this process works in conjunction with blob detection to isolate the desired block. See Figure 5.1 for reference. You will only see one threshold image at a time. To test your masks for the two colors, change which thresholds are used to create the mask image in the code.

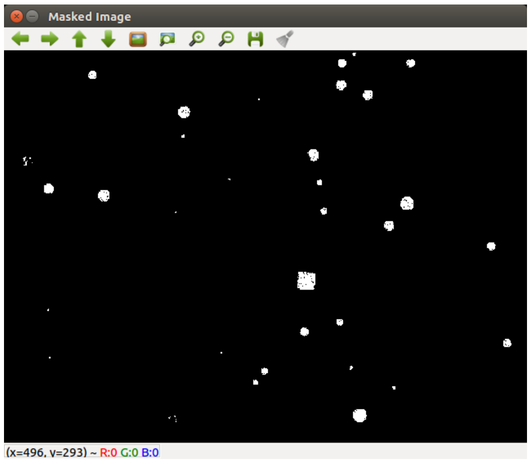

Figure 5.1: A well thresholded image. Note you can see the block clearly.

5.5.4 Blob Detection

In **lab5_blob_search.py**, in the function **blob_search(...)**, you will edit the following to identify blobs within the thresholded image.

```
# Setup SimpleBlobDetector parameters.
params = cv2.SimpleBlobDetector_Params()

# Filter by Color
params.filterByColor = False

# Filter by Area.
params.filterByArea = False
```

```
# Filter by Circularity
params.filterByCircularity = False

# Filter by Inertia
params.filterByInertia = False

# Filter by Convexity
params.filterByConvexity = False
```

Currently all the 'filterBy...' parameters are set to False. You need to decide which filter parameters you want to use to limit the detected blobs to the block you care about. This is done by setting it to **True** and adding the necessary parameters to the code. You should consult the **OpenCV** references for help in understanding what these parameters mean and how to properly set them. It is not necessary or desirable to use all the parameters, so you only need to select the ones that you think will help. When set correctly, the only keypoints detected will be those of the desired block and their centroids will be drawn accurately on the block as seen in Figure 5.2. Note that these parameters will have no effect on the mask view, only the keypoints found. In order to mark the keypoints on the block, complete the following line of code and uncomment it:

```
# Draw the keypoints on the detected block
# im_with_keypoints = cv2.drawKeypoints(im_with_keypoints, keypoints, ???)
```

Also uncomment the viewer for the keypoint image once you have implmented the keypoint drawing.

```
# cv2.namedWindow("Keypoint View")
# cv2.imshow("Keypoint View", im_with_keypoints)
```

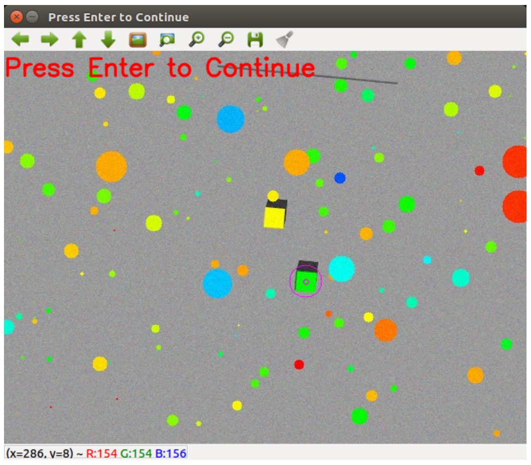

Figure 5.2: A properly drawn centroid on the block.

Camera Calibration

Camera calibration is the process of figuring out the parameters used in the camera transformation. Follow the steps in Appendix B to determine the values of β, θ, T_x, T_y. Use the calibration stick combined with your blob detection and keypoint code to find the centroids of the orange circles. You will need to create a new color mask for the orange color. Note: Keep anything orange away from the area under the camera when doing calibration, including your hands.

Camera Transformation

The camera tranformation will be completed in the function **IMG2W(x,y)** in **blob_search.py**. You should work through the process of finding the camera transformation equations in Appendix B. The goal of this function is to take the received pixel values and convert them to world frame coordinates in meters.

5.5.5 Pick and Place

To complete the pick and place portion of this lab, you will need to edit **lab5_exec.py**. There are a number of functions in this section that should be familiar to you from Lab 2. In the **main()** function, you will have to complete the logic of the pick and place task.
After the **ImageConverter** class is instantiated, the **image_callback()** function will make calls to **blob_search()** for each color every time the camera data is updated. The returned world coordinates will be saved in global variables. Using these coordinates, write code to pick each block and place it in the appropriate location. Make use of the helper function **move_block()** to simplify the repetitive task. Note that there is a 5 second delay programmed in between instantiating the class and proceeding to the pick-and-place logic. This is to ensure there is enough time for the camera to send the image information, for the blob detection to locate the block centroids, and for the camera transformation to convert pixels into world coordinates. You should save a copy of the world coordinates of your blocks before doing any arm moves, as the arm may obscure some blocks during it's motions.
To execute an individual pick-and-place action, you need to complete the **move_block()** function. While the concept is similar to the work you did in Lab 2, we now no longer have hard-coded joint angle values. Instead, you will need to use your inverse kinematics functions to find these angles. **start_xw_yw_zw** should be the world coordinates of the block to pick up, and **target_xw_yw_zw** should be the world coordinates of the goal location. Additionally, your **move_block()** function should detect missing blocks as you did in Lab 2. If a block is missing, the program should print an error to the terminal, but continue picking-and-placing the remaining blocks. If multiple blocks are missing, an error should be printed for each one.
When complete, your code should be able to detect four blocks of two different colors, and pick up and place the blocks in the goal locations.

5.5.6 Debugging

This lab can be a bit tricky to work through as several parts require trial and error. Here are some suggestions:

- Approach the lab sequentially - obtain a good threshold first, then do blob search and only when you have one blob deal with the image transformation.
- Deal with the colors one at a time.
- Comment out parts that you are not currently using (Don't forget the uncomment them though...)

5.6 Report

This lab does not require a formal lab report - only the submission of your code. You should a submit a document to GradeScope containing all of the files that you edited to complete the lab. This will be submitted to GradeScope as in the past. You should include

- blob_search.py
- lab5_exec.py
- lab5_func.py

5.7 Demo

Your TA will require you to run your program twice, each time with different initial block positions and verify that your program can locate the blocks and place them in the correct location. You will also demonstrate that suction feedback is working.

5.8 Grading

- 95 points, successful demonstration and submission of the code. (No code = 0 points)
- 5 points, attendance

Appendix B

Notes on Camera Calibration and Transformations

B.1 Simplified Perspective Transform

To extract spatial information from camera images, we need to derive equations to compute an object's coordinates in the world frame $(x, y, z)_w$ based on row and column pixel coordinates in the image $(r, c)_c$. Instead of giving you these equations, this section will walk through how the transform works to allow you to derive them yourself.

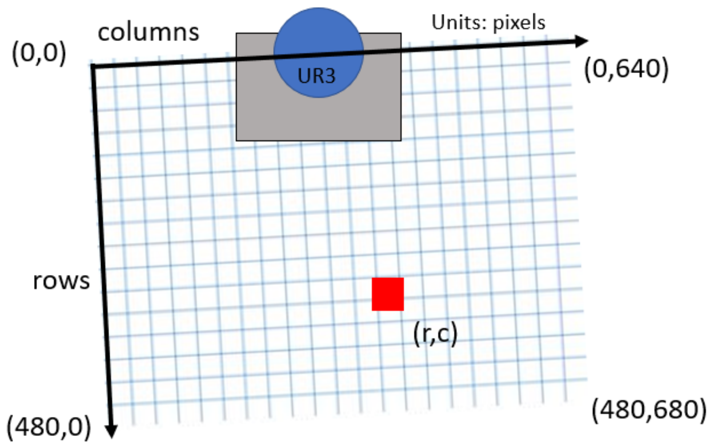

Figure B.1: The view from the camera.

The image starts as seen in Figure B.1. We can see that all data is in pixels, so the location of the object would be described by the rows and columns of the pixel at the centroid of the red block. We can further see that the image is not square to the table, but instead slightly rotated. We will ignore that for the time being.

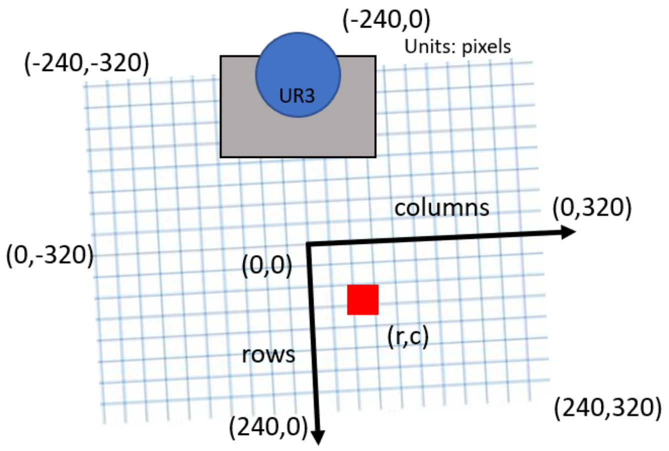

Figure B.2: The view from the camera with the origin shifted to the principal point.

The first step is to translate the frame from the top left corner, to the center of the image also known as the principal point as seen in Figure B.2. As seen in Table B.1, the location of the principal point is in our original frame are called (O_r, O_c). Based on this image, what are O_r and O_c

$$\begin{aligned} O_r &= \\ O_c &= \end{aligned}$$

Side note: It is possible to develop the camera transformation without this translations, but shifting to the principle point is the standard method. It is recommended that you consult a more exhaustive text on computer vision to better understand why.
The next step is to move from pixel measurements to something we can measure in the real world such as meters. To do this, we need a relationship between pixel distance and physical distance on the table top. This value depends both on the physical properties of the camera and the distance between the camera and table top. The physical properties of the camera are encapsulated by focal length (f_x and f_y as seen in Table B.1). Because we do not know the focal length of the camera, we instead create a simple value β which encapsulates both pieces of information:

$$\beta = \frac{f_x}{z_c} = \frac{f_y}{z_c}.$$

This is a simplification of this relationship. It assumes that pixels are square and that the lens does not distort the image at all. Because of this, you might find it is less accurate at the edges of the image, but it should work fine for our purposes. The units of β are pixels per meter.
We now have enough information to complete our first equation. This equation gives us the location of the block in meters, but with respect to the camera frame and not the world frame. These equations are often called the **intrinsic** equations because they rely on the intrinsic properties of the camera such as O_r, O_c, f_x and f_y (i.e. properties that don't change). These equations are essentially a transition from the camera sensor to the tabletop as seen in Figure B.3. Please complete the intrinsic equations using only β, O_r, and O_c

$$\begin{aligned} x_c(r) &= \\ y_c(c) &= \end{aligned}$$

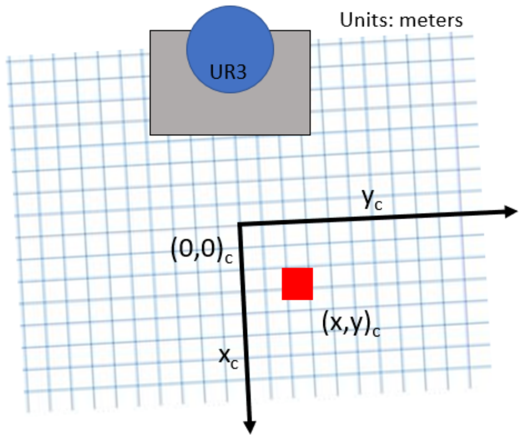

Figure B.3: The camera frame projected onto the table top. This is the output of the intrinsic equations.

Now that we have the intrinsic equations and have transitioned to the table top, we can try to align this table top camera frame with the world frame at the corner of the aluminum plate. We start this by shifting the frame from its current position which sits where the principal point lines up on the table top as shown in Figure B.4. As seen in Table B.1, the relationship between these two frames is O_{cw} which is the origin of the world frame in terms of the camera frame. The key values here are T_x and T_y as $T_z = 0$.
Now that we have aligned the origins, it is time to straighten up the frame as shown in Figure B.5. We do this naturally by a rotation about the z-axis by the angle θ. We can see in Table B.1 that θ is defined for R_{cw}.
Now that we have translated and rotated our frame to align with the world frame, we can write the **extrinsic** equations. These are known as extrinsic

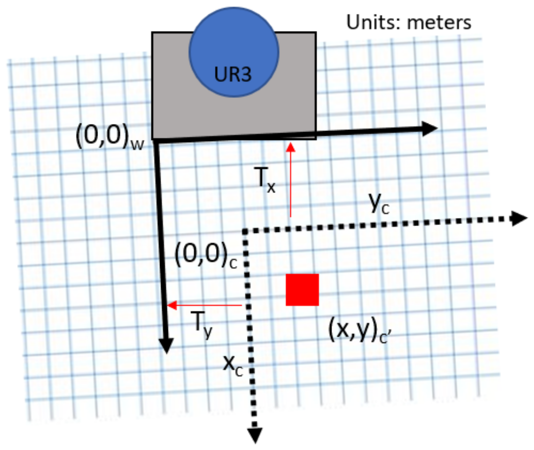

Figure B.4: The table top camera frame must be translated to align with the world frame.

Name	Description
(r, c)	(row,column) coordinates of image pixel
f_x, f_y	ratio of focal length to physical width and length of a pixel
(O_r, O_c)	(row,column) coordinates of *principal point* of image
	(principal point of our image is center pixel)
(x_c, y_c)	(meters) coordinates of point in camera frame
O_{cw}	$= [T_x T_y T_z]^T$ origin of world frame expressed in camera frame
R_{cw}	rotation expressing world frame w.r.t. camera frame.

Table B.1: Definitions of variables necessary for camera calibration.

equations because they are based on values outside of the camera and change depending on placement and orientation. Let's write the extrinsic equations:

$$\begin{bmatrix} x_w \\ y_w \end{bmatrix} =$$

Pay special attention to the signs of T_x and T_y!
We now have both the intrinsic and extrinsic equations and we can put them both together:

$$\begin{aligned} x_w(r, c) &= \\ y_w(r, c) &= \end{aligned}$$

These are the equations that you will use in your code to perform the camera transformation. What about z_w? Well, since we are working on the flat table top, z_w is just the height of any object we wish to pick up.

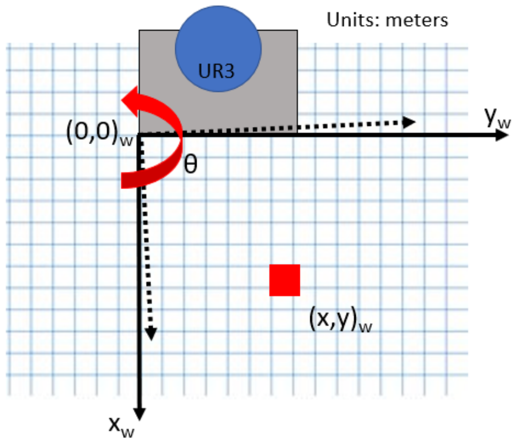

Figure B.5: The table top camera frame must be rotated to align with the world frame.

B.2 Camera Calibration

After formulating the camera transformation it is natural to ask how the key values such as β, θ, T_x and T_y are found. This is through the process of camera calibration. This section will give an overview of how this process works.

- β - This value is found using a calibration stick. The stick has two target set 10cm apart (center to center). When the centroids of these targets are found, their distance in pixels can be determined. It is then just a matter of relating the pixels to the fixed 10cm distance.
- θ - The same calibration stick is used again. It is placed on the table top such that it carefully aligns with the y-axis of the world frame. When the centroids of the targets are found, a right triangle can be formed with the two points and the angle formed will be θ.
- T_x and T_y - This time a single target is placed on the table top with a known position in the world frame. With this known positions, the location of the centroid (r,c) and β and θ, the only unknown in the camera transformation equations are T_x and T_y, which can easily be solved for.