

Lab Procedure for Python Vision: Image Processing

Setup

1. It is recommended that you review [Lab 3 - Application Guide](#) before starting this lab.
2. Hardware Preparation:
 - a. Ensure that the QArm Mini is securely attached to the base.
 - b. Verify that the manipulator is in the rest position.
 - c. Confirm that the QArm Mini is connected to the PC and turn it ON (the light in the switch should be red).
 - d. Check and update the latency setting as shown in Figure 1:
 - i. Navigate to Device Manager > Ports
 - ii. Select the appropriate device - USB Serial Port (COMx) Make a note of the COM port Number.
 - iii. Go to Port Settings > Advanced > Latency
 - iv. Set the latency to 2 ms

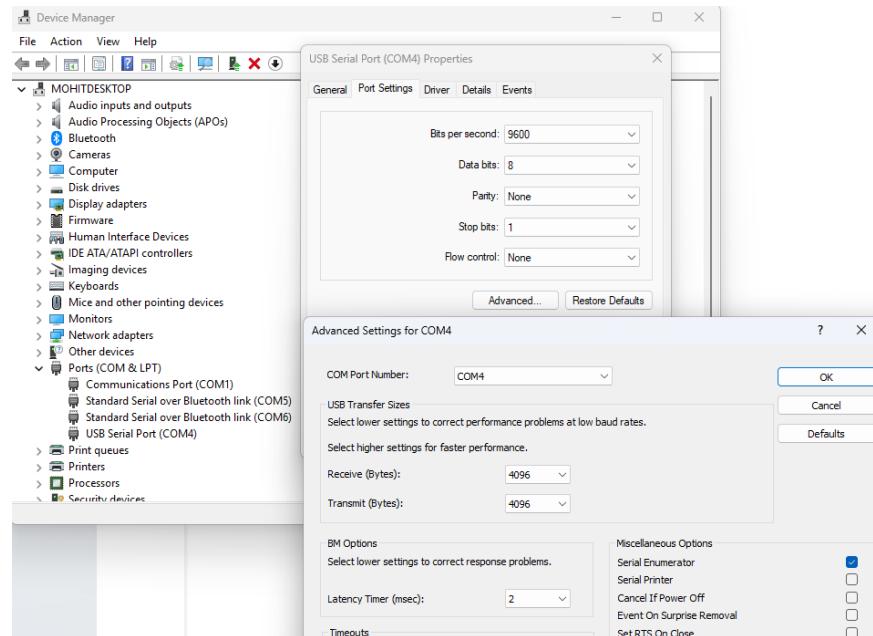


Figure 1. Latency Settings

Image Processing

1. Launch **Visual Studio Code** and browse to the lab directory via the **File > Open Folder** menu. Once in the correct directory, open **image_processing.py**.
 - a. Update the **id** parameter in line 15 to match the COM port you noted during setup.
2. Find any solid-colored object (e.g., a blue phone, a green marker, a purple notebook etc). You will use it later in this lab.
3. In the script, under Section C – Image Processing, lines 29 to 35 are currently commented out. In this lab, you'll uncomment these lines one at a time to observe how each filter affects the image captured by the camera.
4. Ensure the space around the QArm Mini is clear of any objects.
5. Run the script using the **▶** button in the top-right corner.
6. The manipulator will move to the home position. You should see the live camera feed in the OpenCV window titled 'Image Data 1'. Once confirmed, terminate the script by pressing CTRL-C until you see the message: **Received user terminate command**.
7. Uncomment lines 29, 38, and 41 to apply a Gaussian filter to the original video feed. Run the script again. Two OpenCV windows should now appear: 'Image Data 1' and 'Image Data 2'. Take note of the differences between the original and filtered images. When you're done, stop the script using CTRL-C.
8. Uncomment line 30 to convert the image from RGB to HSV color space. Update the script on lines 37 and 38 so that **imgdata_1 = blur** and **imgdata_2 = hsv**. This will display a side-by-side comparison of the Gaussian-filtered image and its HSV-converted version as shown in Figure 2. Run the script again. What do you observe? Stop the script tapping CTRL-C.

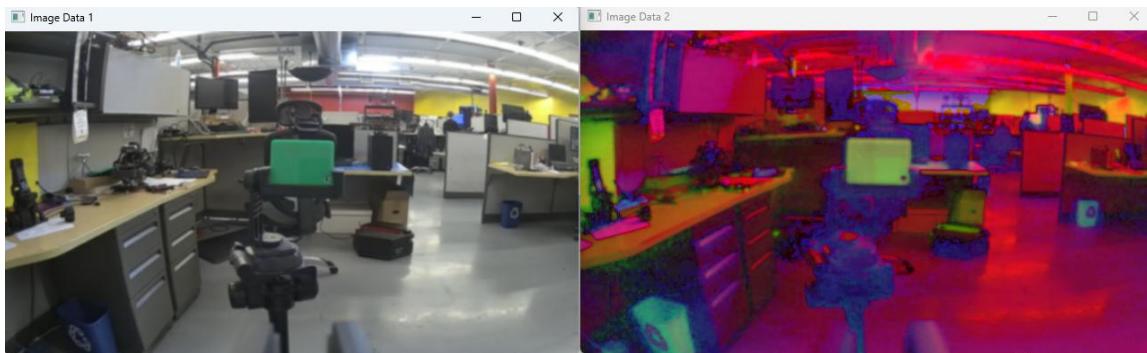


Figure 2. RGB to HSV

9. Uncomment line 32. In this step, the script performs color thresholding on the HSV image to isolate the object of interest. The **threshold_color()** function is used to filter the image based on a specified range of HSV values. You can adjust the upper and lower bounds (for example (80, 180, 150) and (50, 40, 70)) to improve detection. Try changing these values and observe how it affects the result.

Update the script on lines 37 and 38 so that `imgdata_1 = hsv` and `imgdata_2 = colorThresh`. Try to isolate your object as shown in Figure 3. Some noise is expected—this will be addressed in the next step. Stop the script tapping CTRL-C.

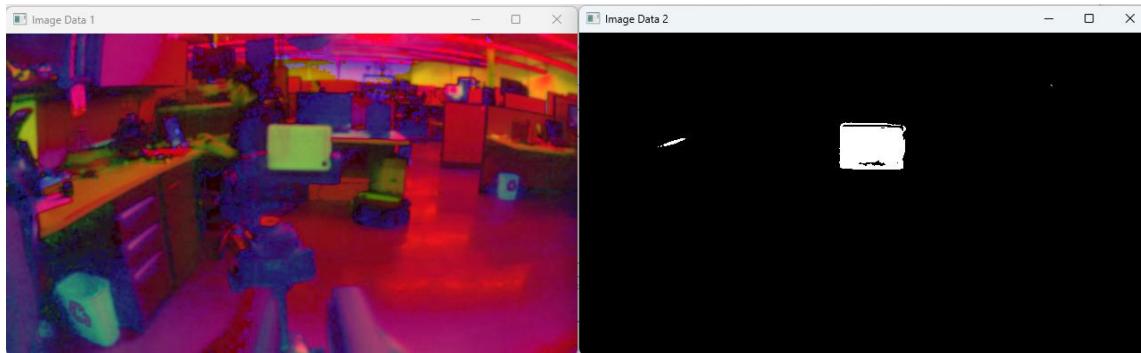


Figure 3. Identify Object of Interest

10. Uncomment line 32, this applies erosion (minimum filter) to the colorThreshold image from the previous step. Update lines 37 and 38 to `imgdata_1 = colorThresh` and `imgdata_2 = mask_opened`. Run the script again. Note down your observations. Uncomment line 32 to apply both erosion and dilation (maximum filter). Update lines 37 and 38 to `imgdata_1 = mask_opened` and `imgdata_2 = mask_closed`. Rerun the script, what do you observe? Stop the script tapping CTRL-C.

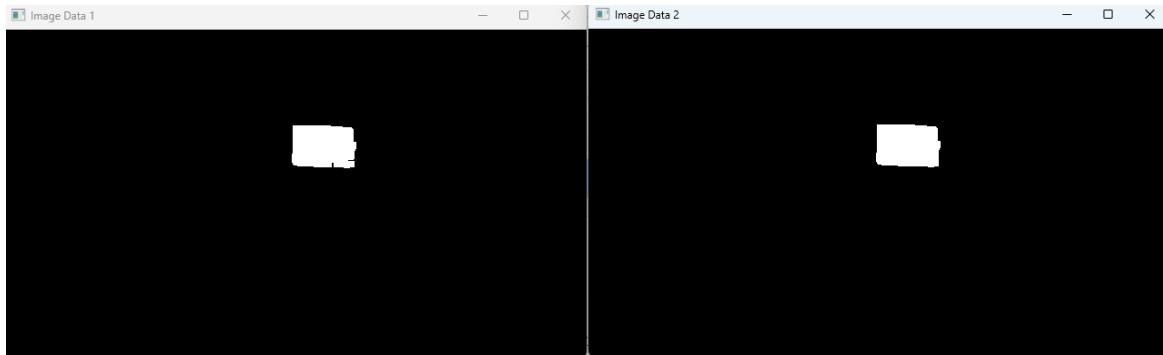


Figure 4. Image Filtering

11. Uncomment lines 34 and 35. In this step, the script highlights the object of interest in the video feed. First, the image is converted to grayscale, which simplifies the background. Then, the `highlight()` function is used to visually separate the object from its surroundings.

The function works by using the threshold mask. It grays out all areas of the image that are not part of the detected object. It **preserves the color** of the object where the mask is active as shown in Figure 4. Update lines 37 and 38 to `imgdata_1 = grayscale` and `imgdata_2 = highlight`. Rerun the script. When you're done, stop the script using CTRL-C.

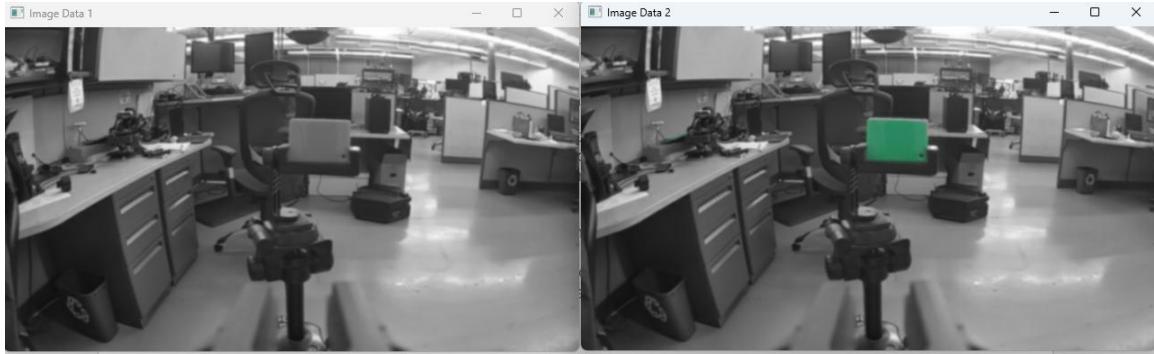


Figure 4. Highlight Selected Color

12. Comment lines from 37 – 41. Uncomment lines from 43 – 48. Rerun the script. In this step, the script identifies the object of interest and draws a red box around it. The function `find_objects()` looks for connected regions in the binary mask (`mask_closed`) that fall within a specific size range. It returns the centroid coordinates and the area of the largest valid region. If a valid object is found, `draw_box_on_detection()` creates a bounding box centered on the detected object using the centroid and area. Finally, the boxed image is displayed in a new OpenCV window titled "Boxed Image" as shown in Figure 5.



Figure 5. Bounding Box around Highlighted Object

13. Terminate the script by pressing CTRL-C until you see the message: **Received user terminate command.**
14. Turn off the arm, gently move it back to its resting position.