Lab Guide

Image Interpretation

Background

Image interpretation guides students through the process of camera calibration and line detection. Prior to starting this lab, please read through the following concept documents:

- Concept Review – Camera Model
- Concept Review - Image Filters
- Concept Review - Calibration Images

Concept Review - Camera Model outlines the exact step in the image creation process where lens distortion affects the digital image. Concept Review – Calibration Images outlines a few types of images that can be used to characterize the camera and geometrically calibrate the raw images provided by the camera sensor. Concept Review – Image Filters outlines various filters that can be used to extract information from raw images.

This lab will ask you to complete 2 pipelines as shown below,

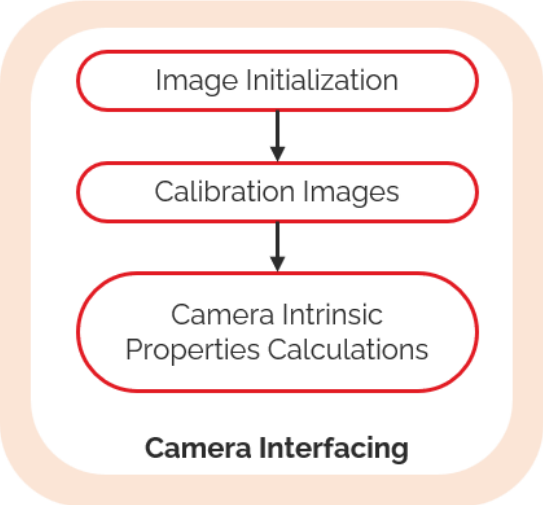

Figure 1: Camera Interfacing Pipeline

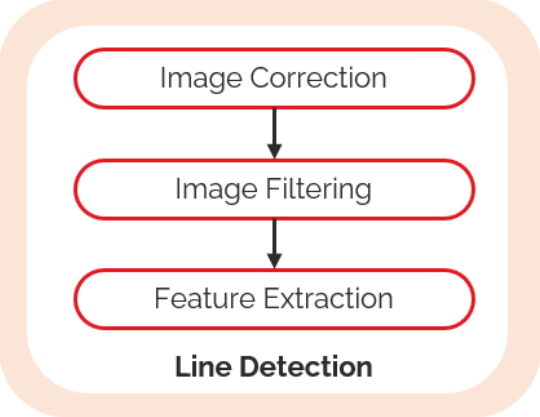

Figure 2: Line Detection Pipeline

- **Camera Interfacing** is designed give students access to raw camera information, calibrate the camera, and then calculate its intrinsic parameters.
- **Line Detection** guides students on using the camera intrinsic parameters to remove lens distortion. Students then learn how image filters can be used to highlight desired parameters in the distortion-free image. Feature extraction takes the post processed and filtered image and uses a feature extraction pipeline to detect lines present in an image.

Skills Activity 1 - Camera Interfacing

The objective of this section is to understand how intrinsic parameters of a camera can change depending on the requested image resolution. When requesting images from a camera two settings are required - image resolution and image framerate. In this lab, the distortion coefficients of two different cameras will be compared – the QCar's CSI wide angle camera, as well as its Intel RealSense D435 camera.

Note: if the skill activity will be performed on the virtual QCar, please review the Setup Guide for students on how to configure the directory structure to use **Quanser Interactive Labs**.

In this lab, you will carry out the following steps,

1. Use a calibration image (chess board pattern)
2. Capture a sequence of images and run a calibration tool

Step 1 – Use a calibration image (chess board pattern)

Physical setup -

If you are using the physical car, you will need a physical chess board pattern. You can create yourself a chess board grid for calibration by using a rigid piece of cardboard and gluing on a pattern from a sample version - link.

Virtual setup -

- **virtual_camera_calibration.py** to define and spawn a virtual chess board and spawn the virtual QCar .

Parameters to consider for the virtual chess board

- Size of each cell in the chess board grid
- Number of rows and columns for the chess board

Figure 1 shows a sample (7,7) chess board with a grid size of 1.

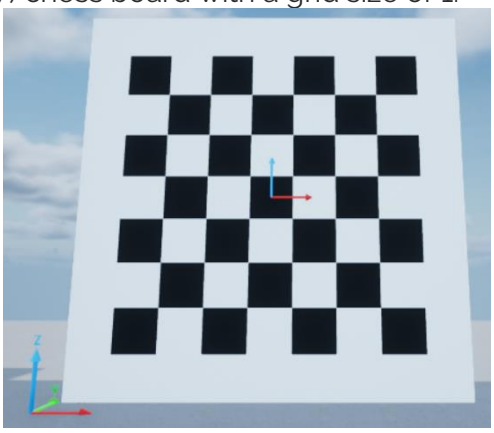

Figure 1: Virtual (7,7) chess board spawned with 1.25 grid size

In a new terminal, run the script **virtual_camera_calibration.py**. For simplicity, this spawns a square chess board. It will ask you to specify the grid size which will be both the number of

columns and rows in the grid. In the chess board grid in Figure 4, a value of 7 was used. The script will also ask you for the size of each cell, which would be in meters. The value used in Figure 1 was 1.25 meters.

The camera calibration process will also require you to move and rotate the board around in space. Follow the prompts from the script **virtual_camera_calibration.py**. Prior to starting the camera interfacing skill activity become familiar with the potential rotations which can be applied to a virtual body. If more than one rotation is applied, they are carried out in the following sequence:

1. Rotation about x-axis
2. Rotation about y-axis
3. Rotation about z-axis

Rotations and translations are applied about the center of the virtual chess board. Include an offset in the z-direction to avoid spawning a chess board below the virtual ground plane.

Step 2 – Capture a sequence of images and run a calibration tool:

Open the script **image_interpretation.py**. Browse to `SECTION A - Student Inputs for Image Interpretation` in the **main()** loop. Specify the desired image size being read by the front CSI camera and the Intel RealSense D435 by modifying variable **imageSize.** For example,

```
imageSize    = [[820,410],[640,480]],
```

will set the resolution for the front CSI camera as (820,410), and the RealSense D435 to (640,480), which are recommended. Also set the **chessDim** parameter according to the value you used in Step 1. Note that **chessDim** represents the number of cells used, not the grid size. The (7,7) square chess board in Figure 1 would have a cell size 6. Lastly, set **camMode** to "Calibrate" for the camera calibration sequence when running this script later.

```
# ============  SECTION A - Student Inputs for Image Interpretation ===============
cameraInterfacingLab = ImageInterpretation( imageSize    = [[820,410],[640,480]],
                                            frameRate    = np.array([30, 30]),
                                            streamInfo   = [3, "RGB"],
                                            chessDims    = 6,
                                            boxSize      = 1)

''' Students decide the activity they would like to do in the
ImageInterpretation Lab

List of current activities:
- Calibrate   (interfacing skill activity)
- Line Detect (line detection skill activity)
'''
camMode = "Calibrate"
```

Codeblock A – Student Inputs for Image Interpretation for camera calibration

Next, focus on `SECTION B1 – CSI Camera Parameter Estimation` and `SECTION B2 – D435 Camera Parameter Estimation` to implement your own camera calibration routine that extracts the camera intrinsic matrix and distortion coefficients and stores them into the following variables for the CSI and D435 cameras.

self.CSICamIntrinsics **self.CSIDistParam** **self.d435CamIntrinsics** **self.d435DistParam**

```
# ==============  SECTION B1 -  CSI Camera Parameter Estimation ====================
print("Camera calibration for front csi")
self.CSICamIntrinsics = np.eye(3,3,dtype= np.float32 )
self.CSIDistParam = np.ones((1,5), dtype= np.float32)
```

Codeblock B1 – CSI Camera Parameter Estimation

```
# ==============  SECTION B2 -  D435 Camera Parameter Estimation ====================
print("Camera calibration for  D435 RGB camera")
self.d435CamIntrinsics = np.eye(3,3,dtype= np.float32 )
self.d435DistParam = np.ones((1,5), dtype= np.float32)
```

Codeblock B2 – D435 Camera Parameter Estimation

You can use OpenCV's camera calibration tools to extract the chess board properties from the saved images and estimate the camera intrinsic and lens distortion coefficients. Quanser's solution is also provided in commented format.

Physical setup -

Run **image_interpration.py** in a terminal on the QCar directly to start the camera calibration skills activity. At the start images from the front CSI camera will be displayed, followed by the Intel RealSense D435. Move your physical chess board around and ensure that it is completely visible in the camera's view. Use the "q" keyboard key to save 15 images for each.

Virtual setup -

Recall that you already have 1 terminal open running **virtual_camera_calibration.py**. In a secondary terminal, run **image_interpretation.py** to start the skills activity code. In the terminal running **virtual_camera_calibration.py**, specify a new location and orientation for the chess board prior to saving the image using the keyboard's 'q' key. Ensure that you have the "Camera Feed" window active. Repeat this 15 times for the CSI camera images, and then 15 more for the d435.

image_interpration.py will stop once the camera calibration sequence has been finished. Quanser's solution also includes an undistort method based on OpenCV. You can find this under the **ImageProcessing** library imported from **hal.utilities.image_processing**.

Calibration Results:

A successful calibration for the QCar should yield distortion and intrinsic parameters like the following:

Front CSI camera

Intrinsic Matrix

$$\begin{bmatrix} 318.86 & 0 & 401.34 \\ 0 & 312.14 & 201.50 \\ 0 & 0 & 1 \end{bmatrix}$$

Distortion Coefficients

$$[-0.9033 \quad 1.5314 \quad -0.0173 \quad 0.0080 \quad -1.1659]$$

Intel RealSense D435

Intrinsic Matrix

$$\begin{bmatrix} 455.20 & 0 & 308.53 \\ 0 & 459.49 & 213.55 \\ 0 & 0 & 1 \end{bmatrix}$$

Distortion Coefficients

$$[-0.5114e-01 \quad 5.4549 \quad -0.0226 \quad -0.0062 \quad -20.190]$$

Note: Use these numbers as reference, mechanical differences across cameras can cause the intrinsic or distortion parameters to change slightly.

Skills Activity 2 - Line Detection

Identifying camera parameters gives you the ability to correct an image for distortion caused by a camera lens. Image filters let you amplify or remove aspects of an image based on the application being developed. The goal of this module is to use the results from Skill Activity 1 – Camera Interfacing and apply image manipulation techniques to create a line detection pipeline.

Note: if the skill activity will be performed on the virtual QCar, please review the Setup Guide for students on how to configure the directory structure to use **Quanser Interactive Labs**.

In this lab, you will carry out the following steps,

1. Initialize the Image Interpretation class for Line Detection
2. Correct the raw image using image calibration results
3. Filter the image to enhance key features such as lines
4. Extract information about the key features
5. Extending to self-driving

Step 1 – Initialize the Image Interpretation class for Line Detection

Specify the desired image size being read by the front CSI camera and the Intel RealSense D435 by modifying variable **imageSize** in `SECTION A - Student Inputs for Image Interpretation` Change **camMode** to "Line Detect" to configure the **imageInterpretation** class to use the line detection skills activity.

```
# ============  SECTION A - Student Inputs for Image Interpretation ===============
cameraInterfacingLab = ImageInterpretation( imageSize    = [[820,410],[640,480]],
                                            frameRate    = np.array([30, 30]),
                                            streamInfo   = [3, "RGB"],
                                            chessDims    = 6,
                                            boxSize      = 1)

''' Students decide the activity they would like to do in the
ImageInterpretation Lab

List of current activities:
- Calibrate   (interfacing skill activity)
- Line Detect (line detection skill activity)
'''
camMode = "Line Detect"
```

Codeblock A – Student Inputs for Image Interpretation for line detection

Use the variables **cameraMatrix**, and **distortionCoefficients** to specify the camera intrinsic matrix and distortion coefficients for the camera being used based on Skills Activity 1 – Camera Interfacing. Ensure that the intrinsic values and distortion coefficients you specify in CodeBlock 4 match the resolution specified in the inputs to the **imageInterpretation** class in CodeBlock 3.

```
# ============  SECTION D - Camera Intrinsics and Distortion Coeffs. =============
cameraMatrix  = np.array([[495.84,   0.00, 408.03],
                          [   0.00, 454.60, 181.21],
                          [   0.00,   0.00,   1.00]])

distortionCoefficients = np.array([-0.57513,
                                    0.37175,
                                   -0.00489,
                                   -0.00277,
                                   -0.11136])
```

Codeblock D – Camera intrinsics and Distortion coefficients.

In the next three steps, you will create an image processing pipeline to

- Remove distortion from an image (Step 2 below, Section 1 in Code)
- Filter an image (Step 3 below, Section 2 in Code)
- Detect lines (Step 4 below, Section 3 in Code)

The individual sections for the image processing pipeline are shown in CodeBlock 5. You can set the variable **imageDisplayed** to any image in the next three steps to check your code as you develop.

```
# ==============  SECTION C1 -  Image Correction ====================
print("Implement image correction for raw camera image... ")
undistortedImage = image
```

Codeblock C1 – Image Correction

```
# ==============  SECTION C2 -  Image Filtering ====================
print("Implement image filter on distortion corrected image... ")
filteredImage = image
```

Codeblock C2 – Image Filtering

```
# =============  SECTION C3 - Feature Extraction ==================
print("Extract line information from filtered image... ")
linesImage, lines = image, []
```

Codeblock C3 – Feature Extraction

Step 2 – Correct the raw image using image calibration results:

Depending on the camera you would like to use (CSI or D435) manipulate the variable image to create an undistorted image with relatively straight lines. The effect of image distortion correction is better displayed using the CSI camera. For guidance, figures 4 and 5 demonstrate distortion correction on an image from the CSI camera while Figures 6 and 7 demonstrate distortion correction on an image from the D435 camera.

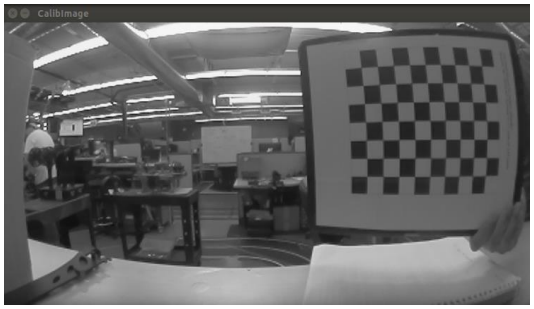

Figure 4: Example CSI image with barrel distortion.

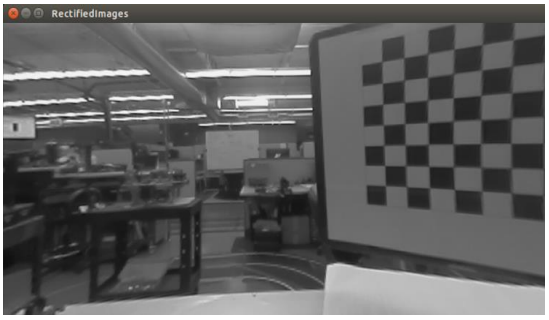

Figure 5: Sample CSI image with distortion corrected.

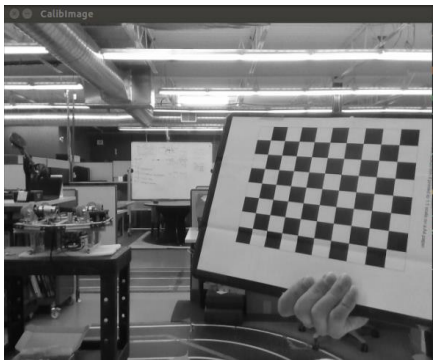

Figure 6: Sample Intel RealSense D435 raw image.

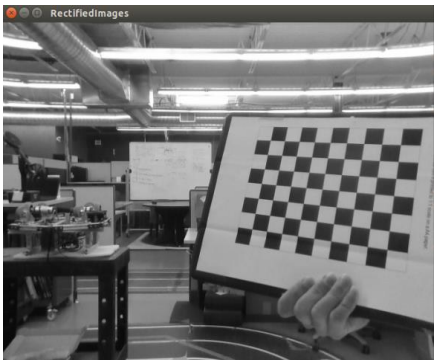

Figure 7: Sample Intel RealSense D435 with distortion correction.

Quanser's solution is commented out and it makes use of the undistort_img() method found in the **hal.utilities.image_processing** class.

Step 3 – Filter the image to enhance key features such as lines:

Depending on the application developed different image filters can output a desired bitonal image. For guidance figures 8-10 show the affect of common filters on a chess board pattern image that highlights information related to lines. Implement a filter that produces similar results.

Quanser's solution is commented out and it makes use of the do_canny() method found in the **hal.utilities.image_processing** class.

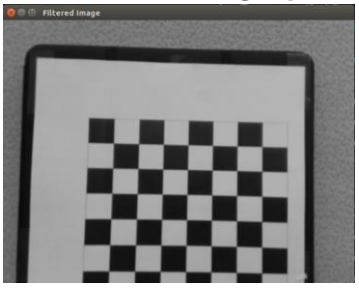

Figure 8. Gaussian blur

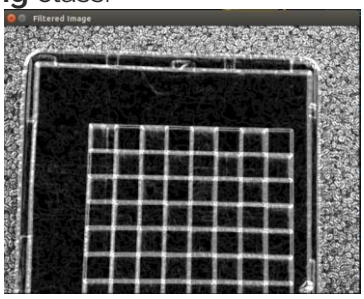

Figure 9: Sobel filter

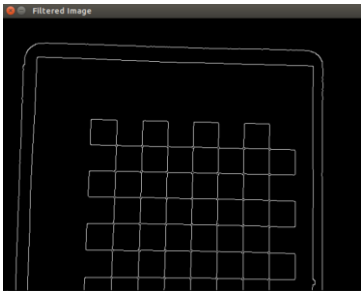

Figure 10: Canny edge detect

Step 4 – Extract information about the key features:

The intent is to use a feature detector to identify lines in an image. These lines will eventually build up to an image-based lane detector. The goal is to manipulate the filtered image to output only the visible lines.

Quanser's solution is commented out and it makes use of the extract_lines() method found in the **hal.utilities.image_processing** class. This method will return two values, the lines present in an image and a reference image with lines overlayed on top.

Use the variable **imageDisplayed** to visualize the final image with the highlighted lines.Sample result of image with highlighted features

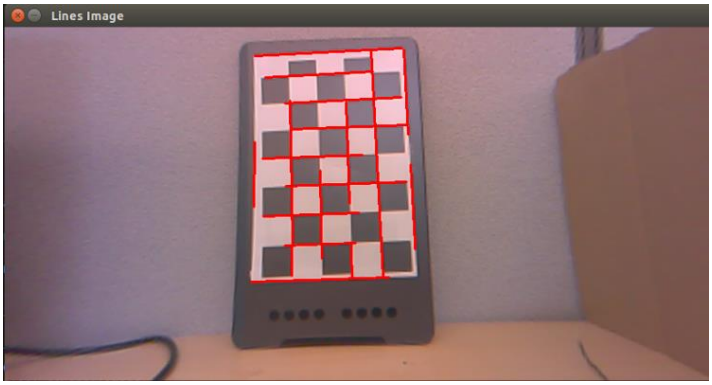

Figure 3: CSI distortion corrected image with line features detected.

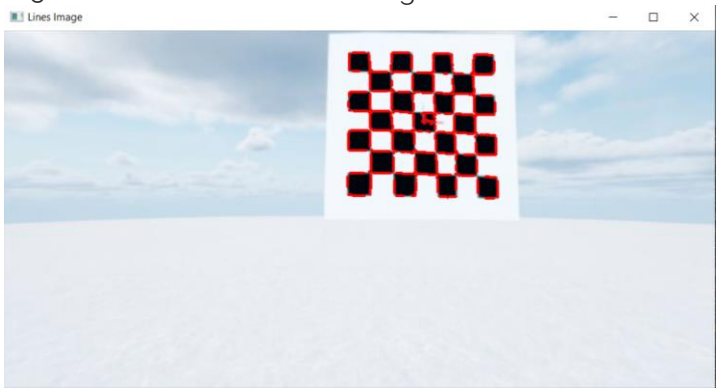

Figure 4: Virtual CSI distortion corrected image with line features detected

Physical setup -

Run **image_interpration.py** in a terminal on the QCar directly to start the line detection skills activity. When prompted, specify if the image feed used will be **D435** or **csi**. Move an object with lines in front of the camera (It's recommended to use the same chess board as for the camera calibration). Verify that lines have been successfully detected in the input image.

Virtual setup -

Recall that you already have 1 terminal open and running **virtual_camera_calibration.py**. In a second terminal, run **image_interpretation.py**. When prompted, specify if the image feed used will be **D435** or **csi**. In the terminal running **virtual_camera_calibration.py**, specify a new location and orientation for the chess board such that the chess board is in the field of view

of the selected image feed. Verify that lines have been successfully detected in the input image.

Step 5 – Extending to self-driving:

Comment on the key line features detected in the following self-driving environments. How would you algorithmically sub-select edges related to lane markings while ignoring buildings, pedestrians, signs etc.?

Physical setup –

Place your QCar on the roadway of the physical Studio Flooring on a lane. Run your Line Detection lab again.

Virtual setup –

Repeat the Line Detection lab with the QCar Cityscape or QCar Cityscape lite environments instead of the QCar Plane. After launching the desired environment, run **qlabs_setup.py** in a separate terminal first, and select "Y" at the prompt, which directly spawns a QCar in the middle of a lane.