

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,

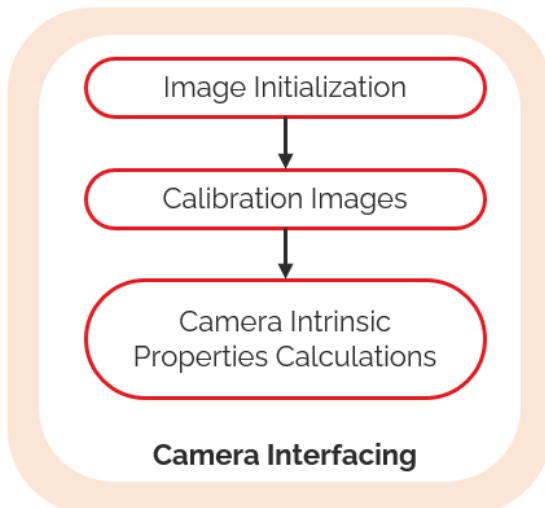


Figure 1: Camera Interfacing Pipeline

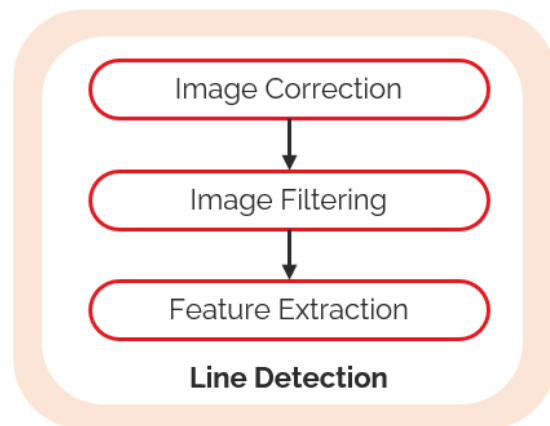


Figure 2: Line Detection Pipeline

- **Camera Interfacing** is designed to 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 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. Note that currently a QCar 1 will be created, the spawn model for QCar 2 will be release in a future update.

Parameters to consider for the virtual chess board:

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

If configured correctly, a chess board will be spawned as shown in Figure 1.

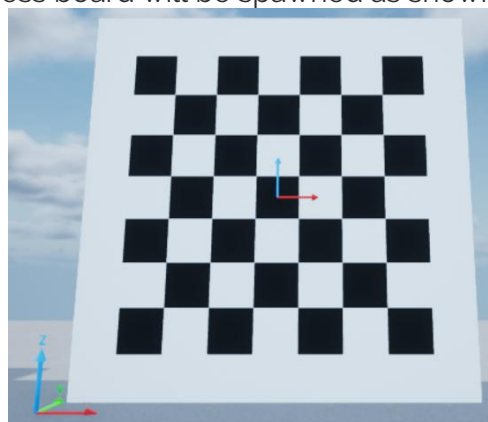


Figure 1: Virtual (7,7) chess board spawned with 1.25 cell size

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 chess board dimensions which will be the number of columns and rows in the board. In Figure 1, 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. To move the camera around in space please specify a desired location as (x, y, z, roll, pitch, yaw) values. Distances should be in meters; angles should be in radian.

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 **gridDim** parameter according to the value you used in Step 1. Note that **gridDim** represents the number of grids used within the chess board, not number of cells. The values of **gridDim** are always 1 less than the dimensions of the chess board. The (7,7) square chess board in Figure 1 would have grid dimension of 6x6. Lastly, set **camMode** to "Calibrate" for the camera calibration sequence.

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

Section 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)
```

Section 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)
```

Section 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.

Physical setup -

Review the [User Manual - Connectivity](#) for establishing a network connection to the QCar. If you are using a QCar2, please connect the QCar2 to a monitor via HDMI, as streaming of CSI camera feed remotely is currently not supported.

Open a terminal window in the folder where the image_interpretation.py file is located:

If using a QCar 1, sudo authority is needed, run the following command:

```
sudo PYTHONPATH=$PYTHONPATH python3 image_interpretation.py
```

If using a QCar 2, run the following command:

```
python image_interpretation.py
```

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 -

You will need 2 terminal sessions for this skills activity. In terminal session 1 use the following command to run the virtual camera calibration tool

```
python virtual_camera_calibration.py
```

In terminal session 2 use the following command to run the skills activity

```
python image_interpretation.py
```

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 finished.

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"],
                                             gridDims = (6,6),
                                             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"
```

Section 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 Section 4 match the resolution specified in the inputs to the **imageInterpretation** class in Section 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])
```

Section 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 Section 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
```

Section C1 – Image Correction

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

Section C2 – Image Filtering

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

Section 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.

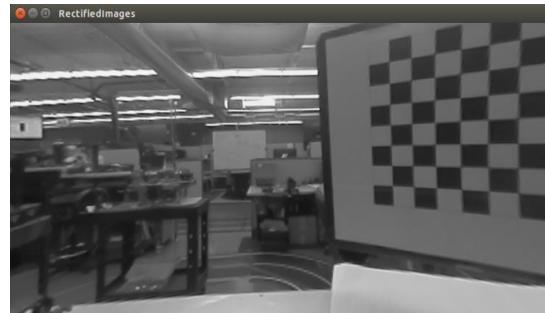


Figure 5: Sample CSI image with distortion corrected.

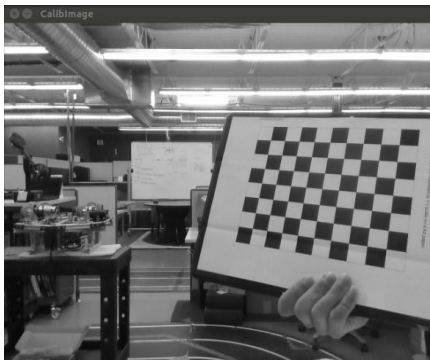


Figure 6: Sample Intel RealSense D435 raw image.

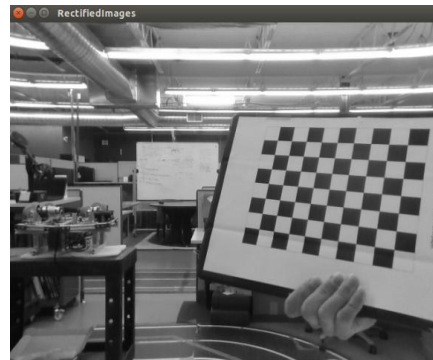


Figure 7: Sample Intel RealSense D435 with distortion correction.

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.

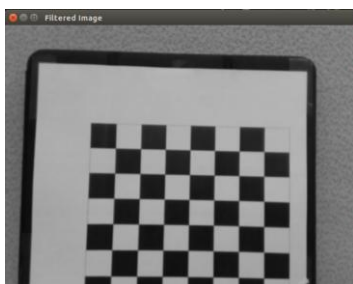


Figure 8. Gaussian blur

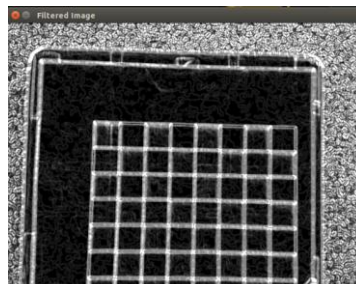


Figure 9: Sobel filter

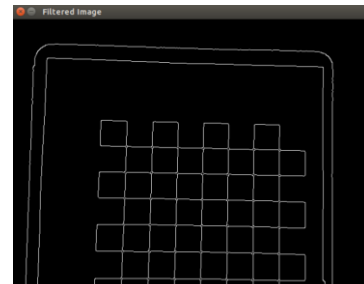


Figure 10: Canny edge detect

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.

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.

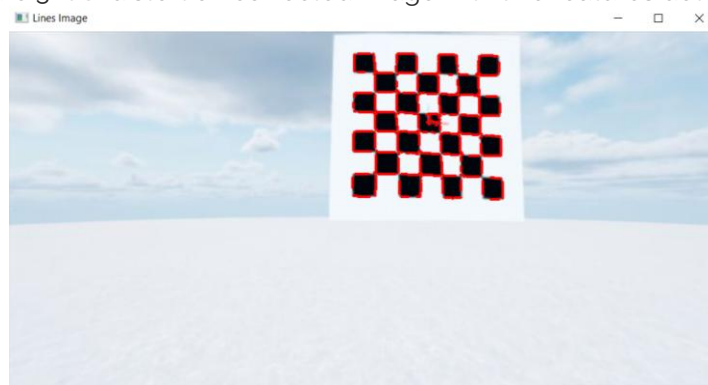


Figure 4: Virtual CSI distortion corrected image with line features detected

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.