# Lab Guide
# 360 Vision

## Content Description

The following document describes a image stitching example in both MATLAB and Python software environments utilizing the virtual QCar.


Content Description 1
Lab Description 1
MATLAB 2
  Running the example 2
  Details 3
Python 3
  Running the example 3
  Details 3


Prior to starting the example please go to the **Cityscape Lite** workspace and run the **qlabs_setup_applications.py** python script to configure the virtual world.

## Lab Description

In this example, we will capture images from the four CSI cameras at the same resolution and frame rate. These will be stitched together, and passed to a display module.

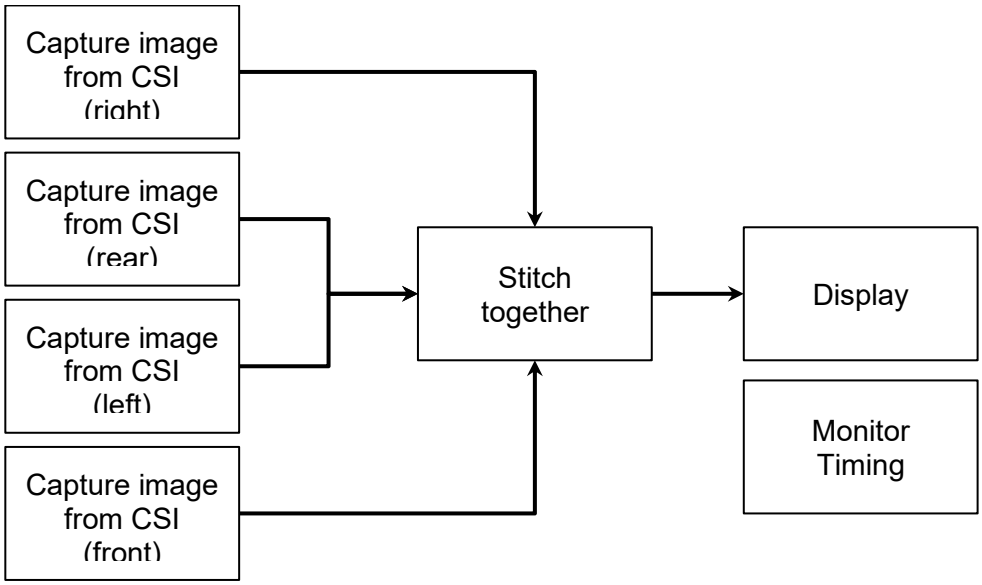



Figure 1. Component diagram

## MATLAB

The Simulink implementation is displayed in Figure 2 below.

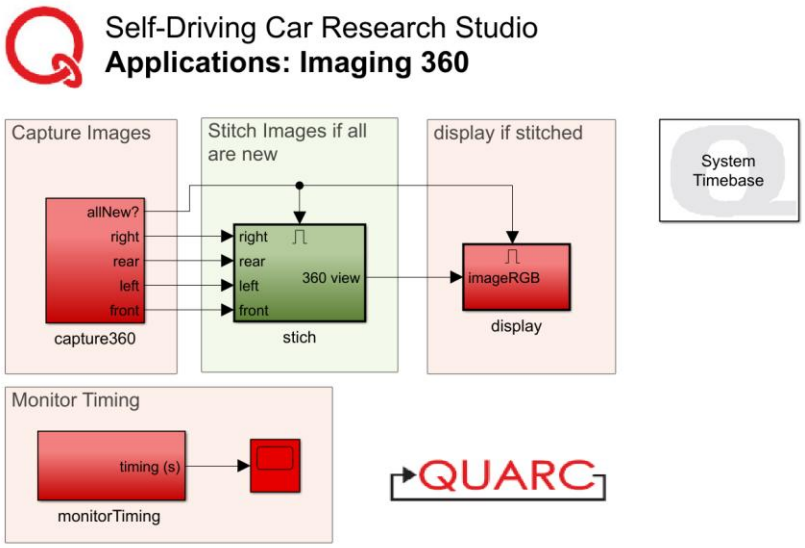
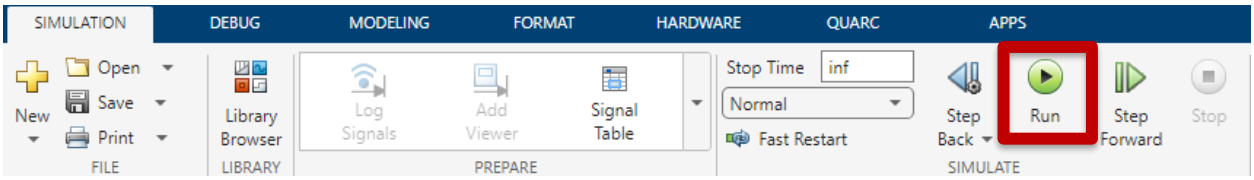



Figure 2. Simulink implementation of 360 Vision

### Running the example

To run examples for virtual QCar please go to the **SIMULATION** tab in the ribbon interface and click on the Run icon.




The output in the **Video Display** block should look like Figure 3. (The order of stitching is **Right, Rear, Left,** and **Front)**

Figure 3. 360 view output showing images stitched together

### Details

The implementation has a few important elements that help boost performance. The model is deployed at 30 Hz with each camera streaming a 640 x 480 RGB (3 channels) unsigned integer image (8 bits or 1 byte per data sample). The data rate can be estimated as,

$$30\ \frac{steps}{s} \times (640 \times 480 \times 3)\ \frac{samples}{image} \times 1\ \frac{byte}{sample} \times 4\ \frac{images}{step}$$

In the implementation provided, the images are stitched together only when all the images are new. The **Video Capture** block in the **capture360** module also outputs a **new** signal that is high (**1**) when the image is new. The four **new** signals are passed into a logical AND, which is then used to stitch images and display them, improving performance.

Note that a simple matrix concatenate method is used here to stitch the images together side by side. Alternatively, you may consider lens distortion correction on the raw images first, and using feature matching to remove translational and rotational discrepancies in the placements of the 4 cameras.

## Python

### Running the example

Check User Manual – Software Python for details on deploying python applications. Run the **imaging_360.py** example on your local machine. The output in the **cv2.imshow()** function should look like Figure 2. Just like before the order of stitching the images is still **Right, Rear, Left,** and **Front.**


Figure 4. 360 view output showing images stitched together

### Details

The implementation maintains the same structure as the simulink example by combining all 4 images into a single displayed image.

```
# Creating a combined 360 view
imageBuffer360 = np.concatenate((cameras.csiRight.imageData,
                                 cameras.csiBack.imageData,
```

```
                                cameras.csiLeft.imageData,
                                cameras.csiFront.imageData),
                                axis = 1)
```

Note that a simple matrix concatenate method is used here to stitch the images together side by side. The script runs in a **while** loop at 30Hz The **cv2.waitKey(msSleepTime)** will pause the loop for the time difference between the **sample time** and **computation time.**