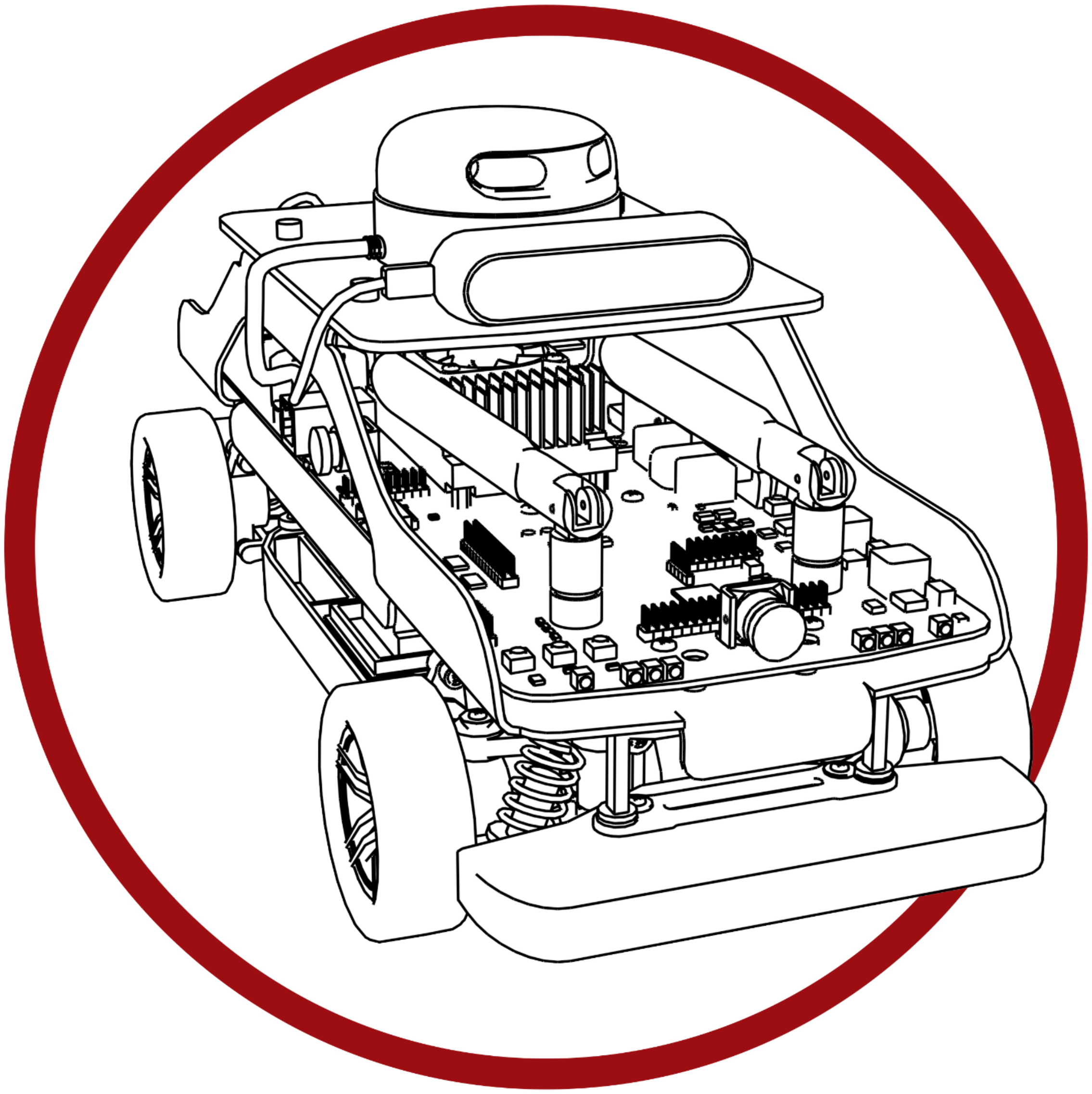
Self-Driving Car Research Studio



RGBD Imaging - Simulink

V 1.1 (November 2020)

Table of Contents

I. System Description

In this example, we will capture images from the Intel RealSense's RGB and Depth cameras. After thresholding the RGB image for a red stop sign, and extracting the sign's coordinates, the distance to the sign will be extracted from the corresponding coordinates in the depth image.

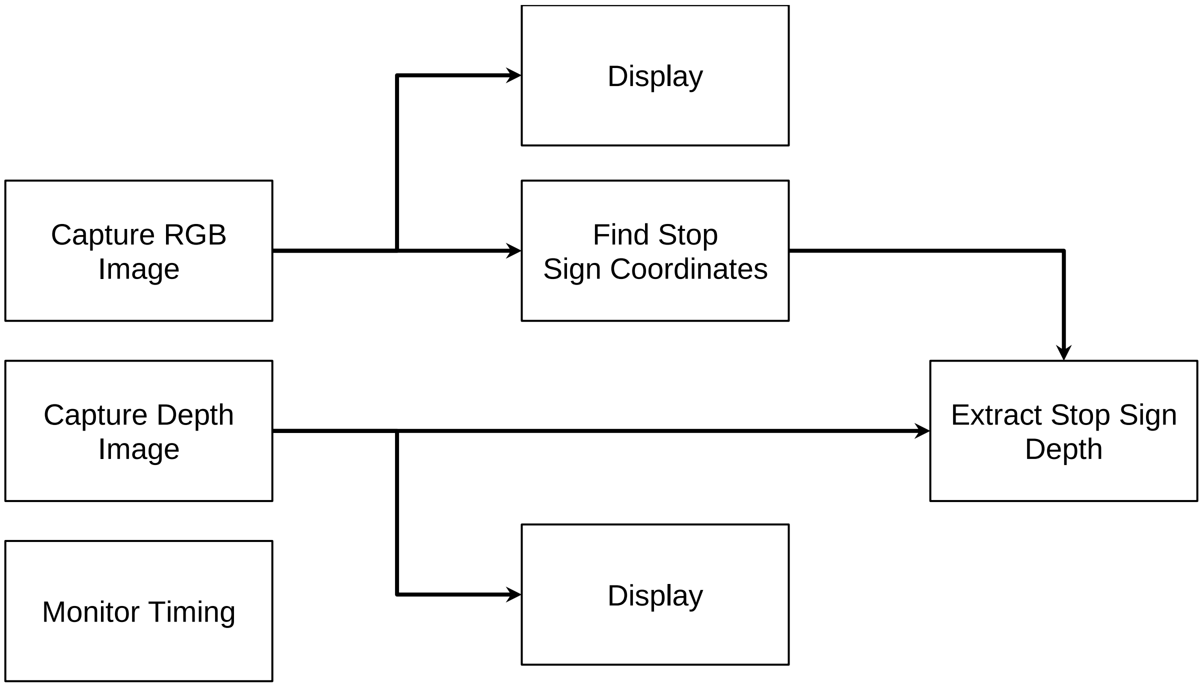


Figure 1. Component diagram

In addition, a timing module will be monitoring the entire application's performance. The Simulink implementation is displayed in Figure 2 below.

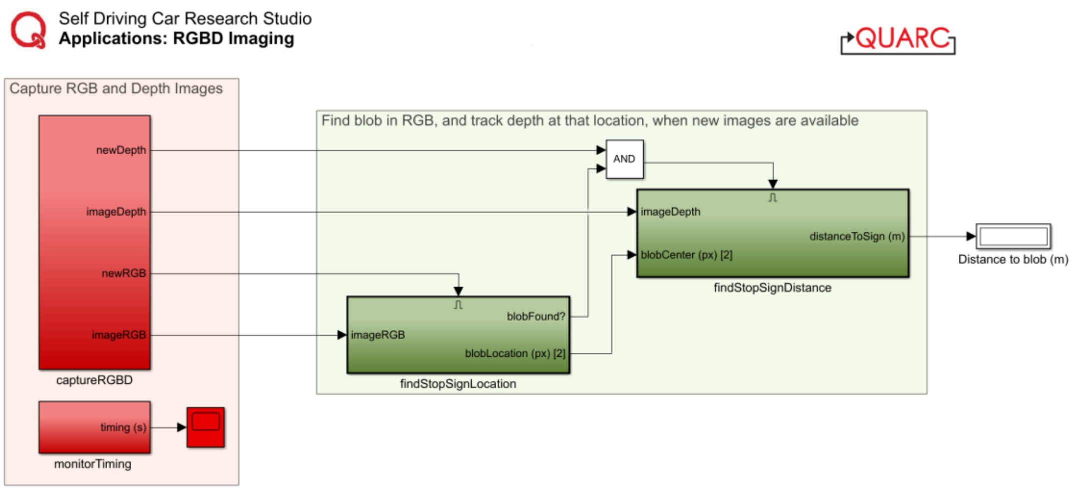


Figure 2. Simulink implementation of RGBD Imaging

II. Running the example

Check the user guide **IV - Software - Simulink** for details on deploying Simulink models to the QCar as applications.

As lighting conditions and other objects in the scene may vary, you may need to adjust/tune the thresholding parameters inside the **findStopSignLocation** module. See the support documentation on **Image Color Spaces** and **Image Thresholding** for more information on this. Tune the saturation and value parameters until the binary image only displays the stop sign. The output in the 3 **Video Display** blocks should look those in Figure 3, which shows the raw RGB output, a binary output after thresholding, and the depth output.

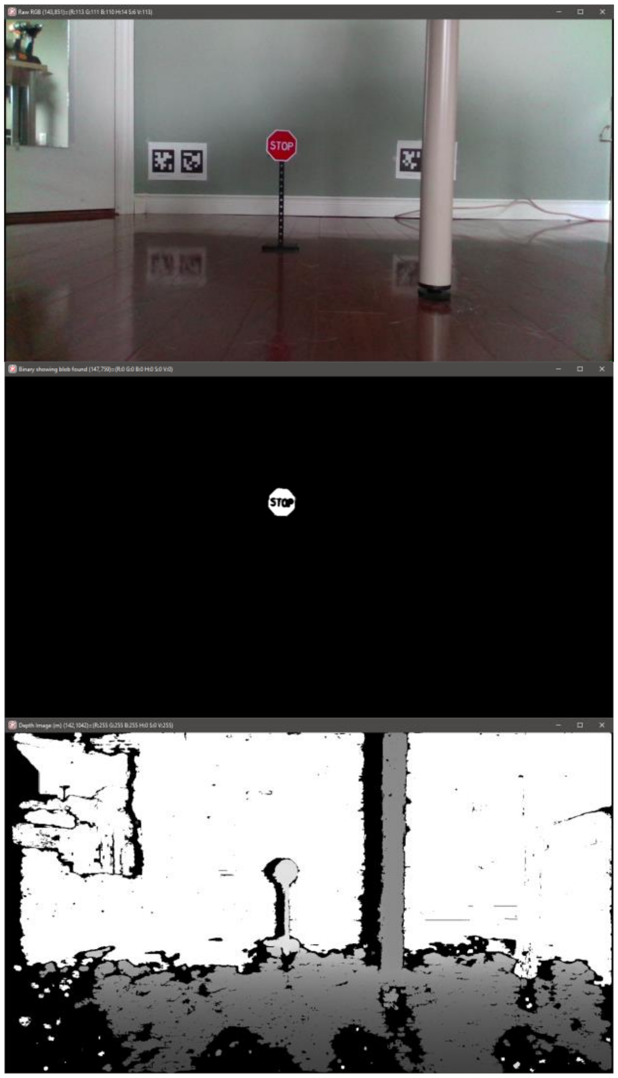
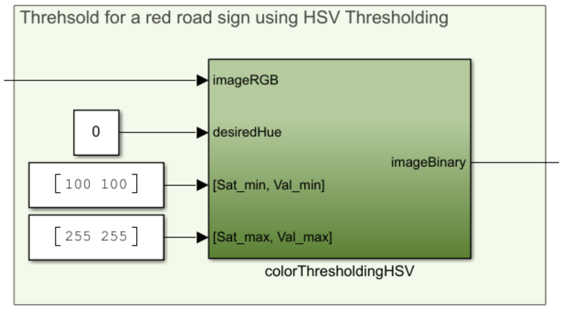

Raw RGB (143,851)=(R:113 G:111 B:110 H:14 S:6 V:113)
STOP
Binary showing blob found (147,739)=(R:0 G:0 B:0 H:0 S:0 V:0)
STOP
Depth Image (m) (142,1042)=(R:255 G:255 B:255 H:0 S:0 V:255)

Figure 3. RGBD outputs showing RGB image (top), bitonal thresholding image (middle) and depth image (bottom)

III. Details

1. Capturing nothing but the Stop Sign

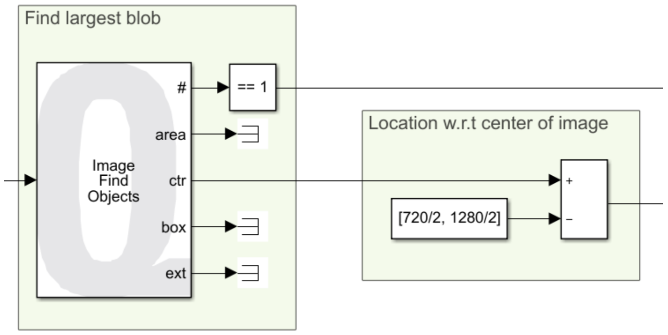
   First, we pass the RGB image as **imageRGB** to the **colorThresholdingHSV** module inside the **findStopSignLocation** subsystem. This converts it to the HSV color space, decoupling the color itself from its intensity and lightness/darkness. subsystem using an **Image Transform** block.



   The stop sign is red, which corresponds to a hue of **0**. Once we threshold the HSV image with suitable saturation and value parameters, the **imageBinary** output shows all the pixels that fall within our color search region.

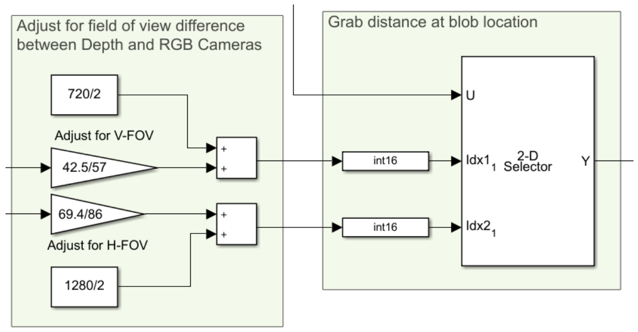
2. Finding the location of this blob in image coordinates

   Within the **findStopSignLocation** subsystem, we use the **Image Find Objects** block to find the (row, col) image coordinates of the largest blob in the **imageBinary** input. We are only interested in the center of this blob, available at the **ctr** output. Note that this is w.r.t a coordinate frame attached at pixel location (1, 1). We need the location with respect to the center of the image, which is handled by a simple subtraction.



3. Estimating distance to the blob from the depth image

   Although the RGB and Depth images are both captured at a 1280 x 720 resolution, the depth camera has a higher field of view, and hence the coordinates from step 2 must be adjusted before extracting the depth. We account for the field of view differences and then use a selector to extract the distance in the depth image at the adjusted coordinates.


Adjust for field of view difference between Depth and RGB Cameras
720/2
Adjust for V-FOV
42.5/57
69.4/86
Adjust for H-FOV
1280/2
Grab distance at blob location
int16
int16
U
Idx1
Idx2
2-D Selector
Y

4. Performance considerations

   To improve performance, we only execute the **findStopSignLocation** subsystem when **imageRGB** is new, through the means of an enabled subsystem. Therefore, the **blobFound?** output is high (**1**) if and only if a blob is found AND a new RGB image was available. Next, the **findStopSignDistance** subsystem is executed only if the **imageDepth** input is new AND **blobFound?** is true.

   Overall, the distance to the stop sign is only calculated if three conditions (new RGB image, new Depth image as well as a blob actually found) are met simultaneously, improving performance. Also note that the **Video Display** blocks are placed within the enabled subsystems.