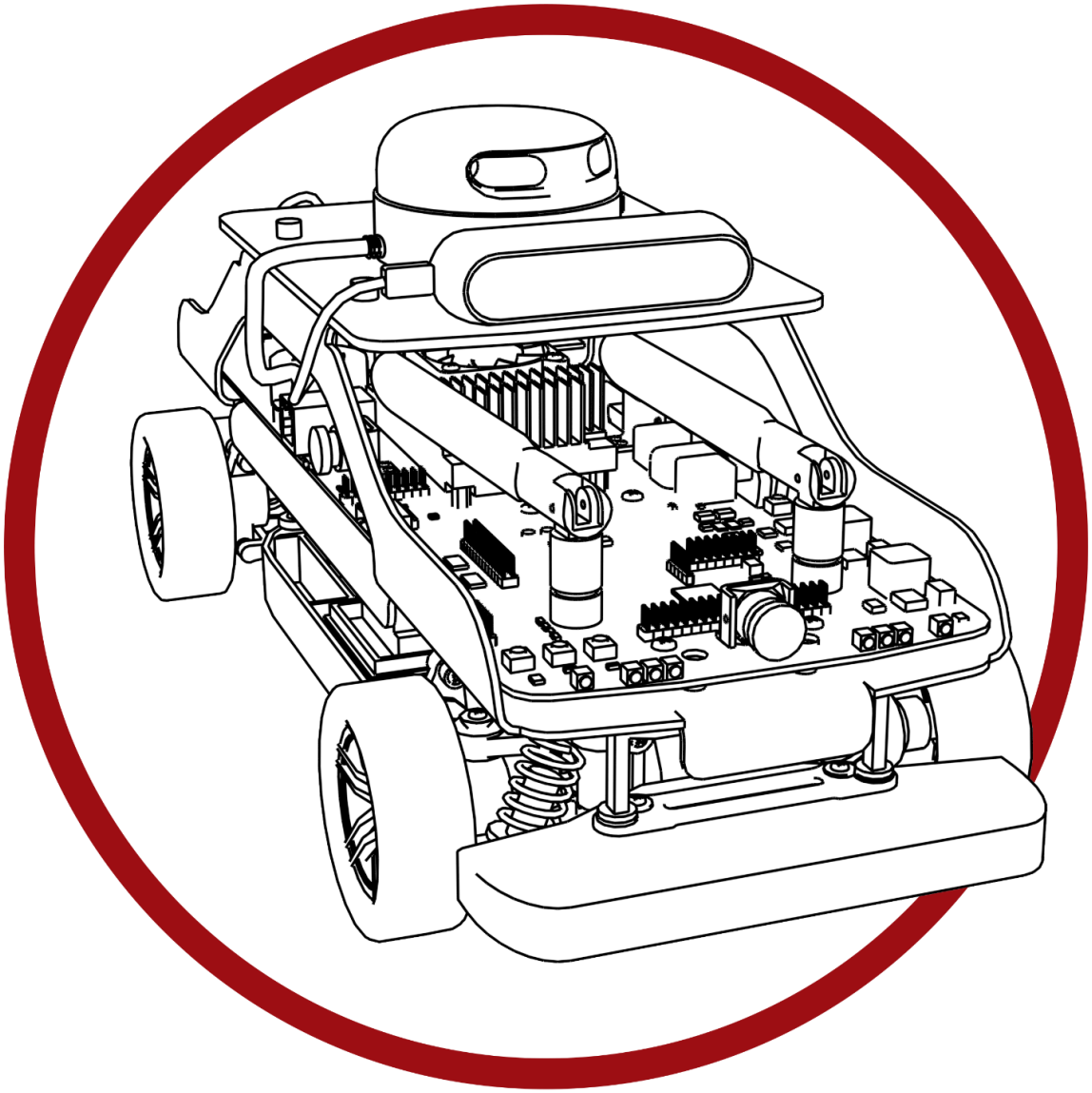
QUANSER
INNOVATE·EDUCATE

Self-Driving Car Research Studio



Autonomous Driving Car Example 2 - Simulink

V 1.2 (April 2021)

Table of Contents

I. System Description 3
II. Running the example 4
III. Details 5

I. System Description

In this example, we look at the application of autonomous lane following and obstacle detection using the QCar. The process is shown in Figure 1.

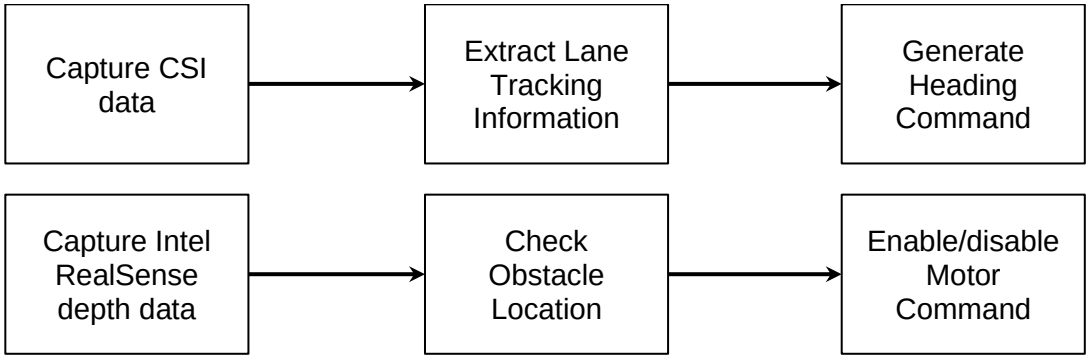


Figure 1. Component diagram

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

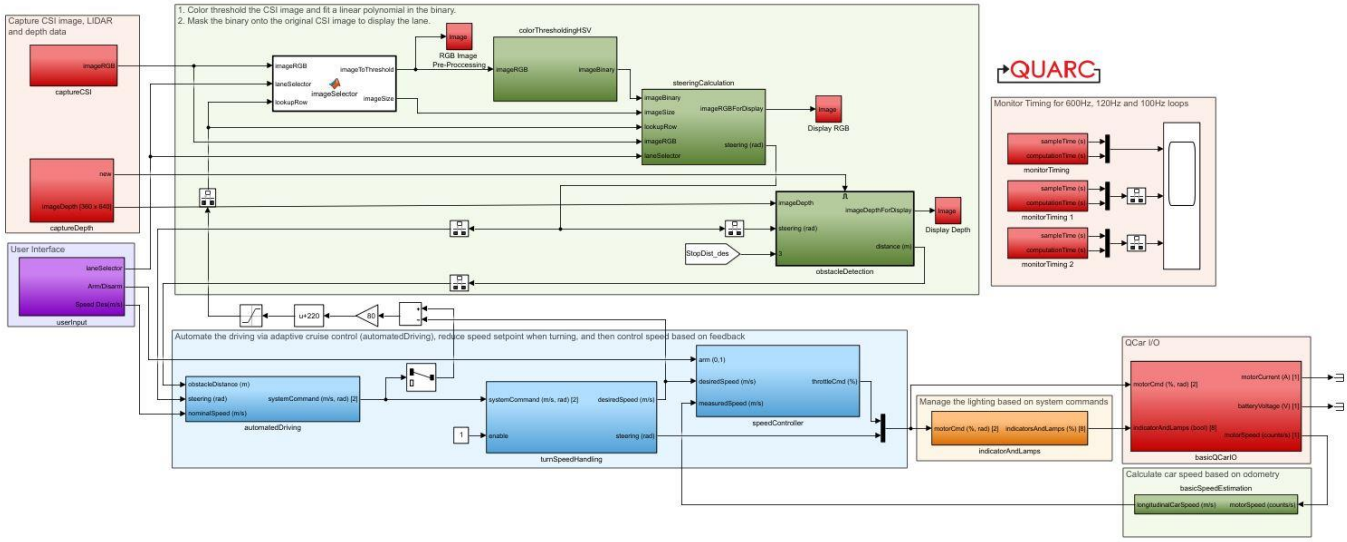


Figure 2. Simulink implementation of lane following with obstacle detection.

II. Running the example

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

The following example can be run by configuring a continuous dual lane loop using the roadmap layouts as part of the SDRS which are an optional add-on of the SDRS ground station peripherals.

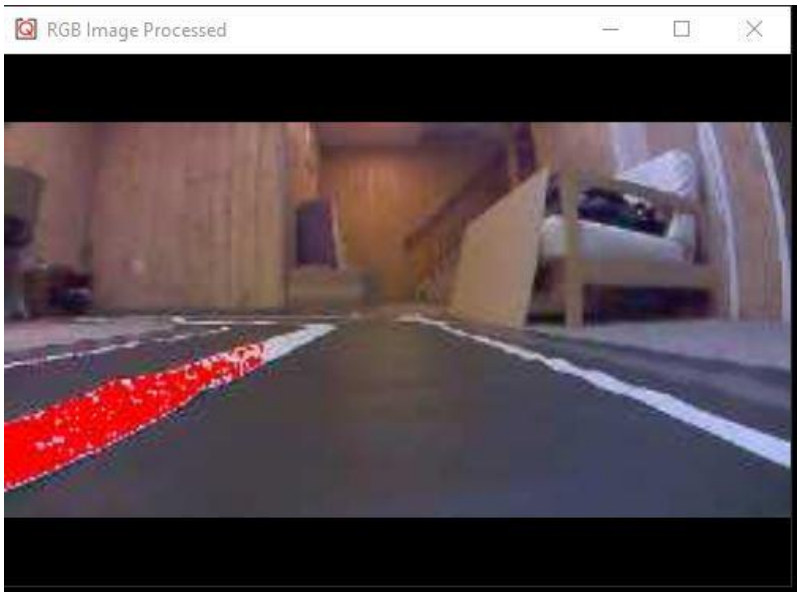


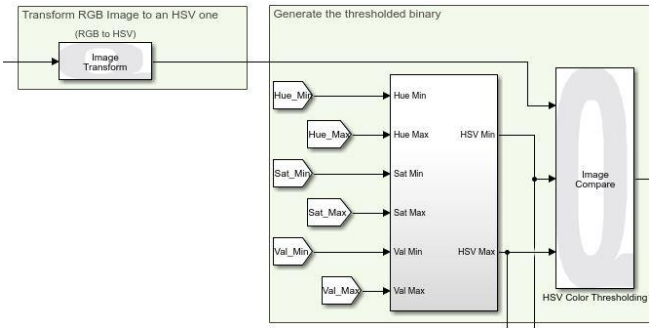
Figure 3. Processed lane extraction Image.

III. Details

1. imageSelector

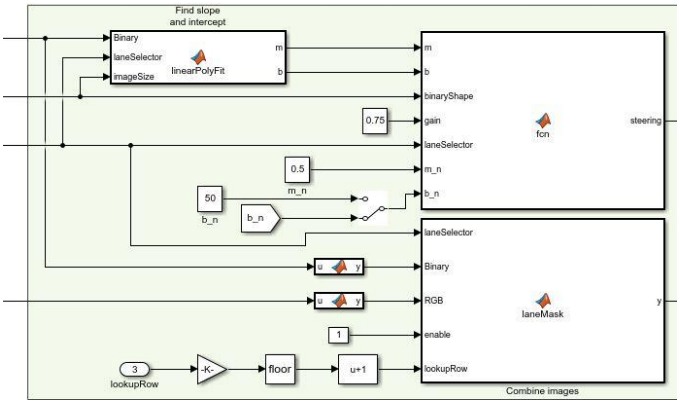
Based on the **laneSelector** option the following function will select the portion of the image used for the colour thresholding algorithm. A laneSelector value of **1** will select the region of the image where the **right lane** is most likely to be present. A laneSelector value of **0** will select the region of the image where the **left lane** is most likely to be present.

2. colorThresholdingHSV



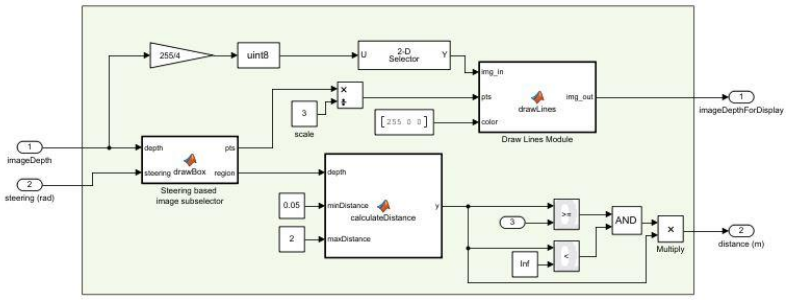
Color thresholding is done in two components. Component one converts the **imageRGB** input from the RGB to the HSV image plane. For in-depth information for the HSV image plane you can look at the **Image Color Spaces** document in the **3. Supporting Documentation directory**. Prior to identifying the regions of the image where a specific HSV values are present a subsystem generates the **HSVMin** and **HSVMax** values used to set the range for the specific color we want to select. Using the **ImageCompare** block we can generate a binary image which contains the portions of the image for which the selected color is valid.

3. steeringCalculation



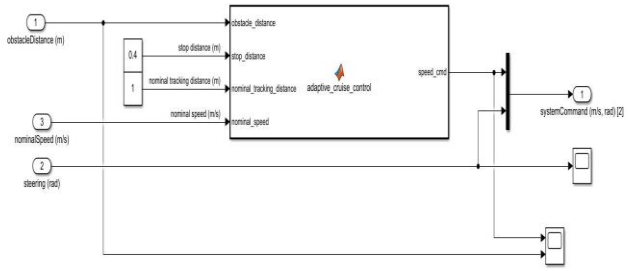
The first step for calculating the steering angle is to approximate two parameters which define the lane being tracked. The **linearPolyFit** function analyses the lane properties based on the **laneSelector** input. Using a linear approximation a **slope [m]** and **y-intercept [b]** is passed onto a second MATLAB function. We **compare** the **nominal_x** and the **desired_x** components of the slopes to identify how much our steering angle needs to be adjusted. The last MATLAB function **laneMask** combines the RGB image with the binary image from the **colorThersholdingHSV** to show the regular RGB image with red pixels over the lane which is currently being tracked.

4. obstacleDetection



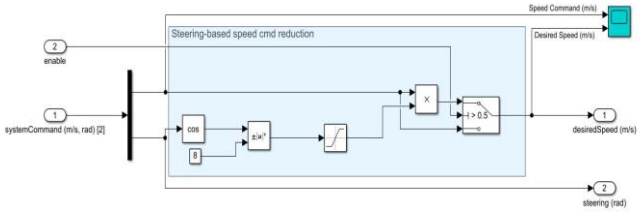
ThedrawBox function uses an input depth image of size 640x480 and the desired steering angle to extract a region of interest and sequence of points for the border of this region of interest. The depth information for the selected region is passed to the **calculateDistance** function. We find the region of points in the following interval **0.05m < depth < 2m** and calculate what the average depth is in the selected region. The input **stopDistance** lets us compare whether or not the average depth which we calculated is >= the stopDistance. The **drawLines** function uses the pixels from the **drawBox** function to define the lines that draw a red box for visualizing the region in the image of where the depth information is being computed.

5. automatedDriving



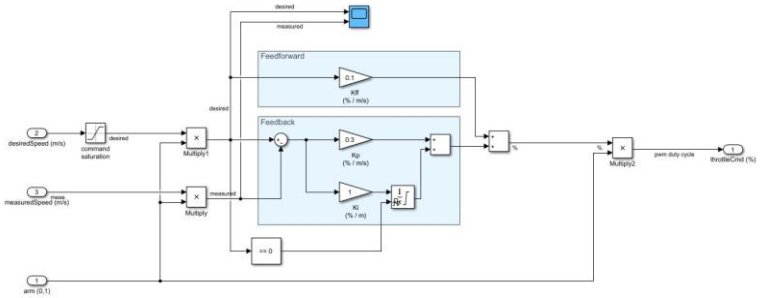
This controller subsystem will adjust the **systemCommand** desired speed based on a commanded **nominalDistance(m/s)** and **obstacleDistance(m/s).** Using a fixed stop_distance and nominal_tracking_distance the linear speed command is modulated such that the QCar slows down until the **stop_distance** is greater than **obstacle_distance**.

6. turnSpeedHandling



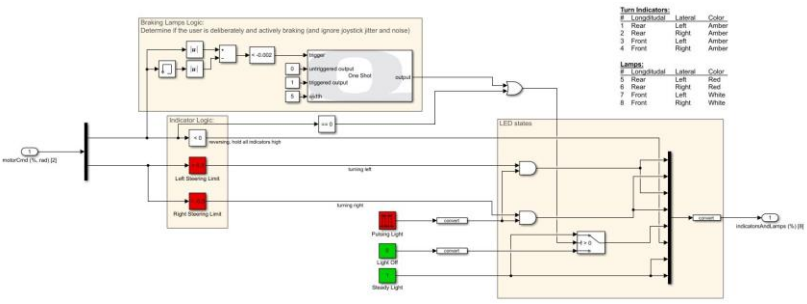
An enable constant is used to configure what the **desiredSpeed(m/s)** should be. We can pass the linear velocity command directly or evaluate the cosine of the steering to the power of 8. This secondary method will slow down the QCar closer to a turn and speed up during straight sections of road.

7. speedController



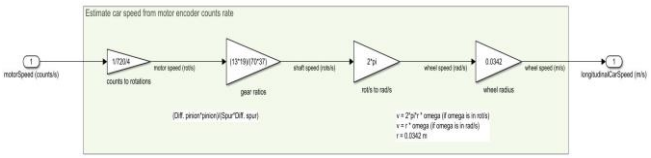
A **feedforward PI** controller is used to generate the desired **throttleCmd(%)** signal sent to the ESC of the QCar. The measuredSpeed of the QCar is compared to the desiredSpeed where the error term is converted from m/s to % via a proportional gain and m to % via an integral gain. To avoid integrator windup due to error accumulation over time the integral is reset using the arm signal which is also in charge of enabling the motor command. Lastly the error term is adjusted by a feed forward gain which converts the desired speed from m/s to %. By using a feedforward gain the controller command is no longer centered about zero but the desired setpoint defined by the feedforward gain.

8. indicatorAndLamps



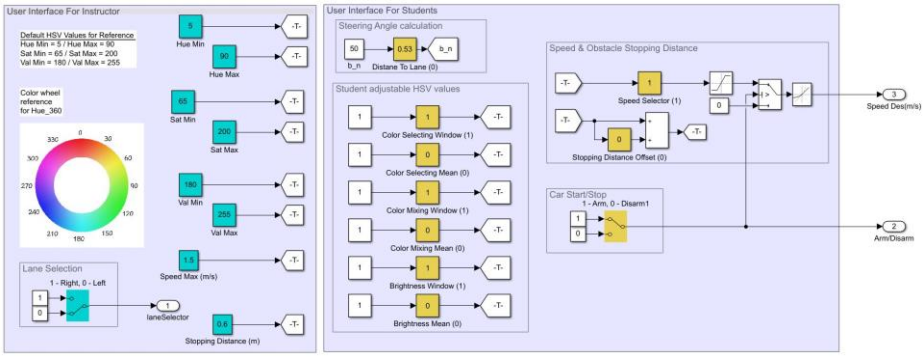
The logic inside this subsystem enables the LEDs on the QCar to act as a direction indicator. For the amber LEDs located at the front and the back of the QCar, these act as steering indicators. The **steering** is either **greater** than **0.3** for a **left** direction or **less** than **0.3** for a **right** steering indication. The rear left and right lamps are set to red when the QCar has a negative linear velocity while they are off during regular operation.

9. basicSpeedEstimation

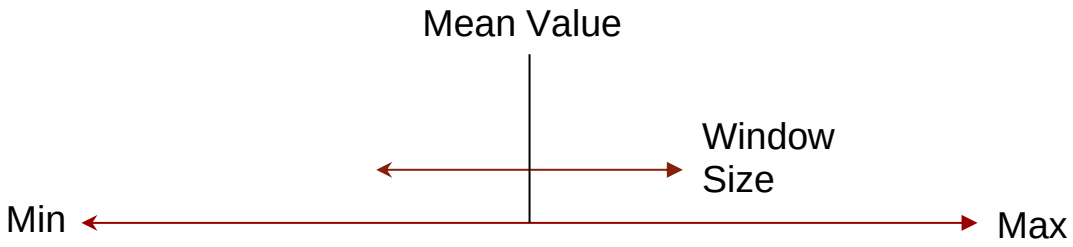


The **motor encoder** on the QCar can give us **counts/s** which is passed through four scaling terms. The **first scaling term converts** from **counts/s to rotations/s**, a **second scaling term** passes the motor rotations through a gear ratio which **gives the wheel shaft rotational speed.** The third scaling term converts the shaft speed from rotation/s to rad/s and lastly the angular speed is multiplied by the wheel radius to get an estimate of the **logitudinalCarSpeed** (m/s).

10. userInputs



This subsystem is divided into two sections. The group with the label QCar Parameter Definition allows us to define the max/min values used for the HSV thresholding, the lane which we want to follow, the stopping distance to an obstacle and the maximum speed of the QCar. For modifying the speed of the QCar we have a slider gain called **Speed Selector** which amplifies the desired speed of the QCar between **0%** to **100%** of the maximum speed defined in section 1 of this subsystem. To control the stopping distance, we added an offset term called **Stopping Distance Offset**. By default, the minimum stopping distance is defined in section one of this subsystem. By **default,** the minimum stopping distance is set to be **0.6(m)**, the stopping distance offset adds an additional percentage of the minimum stopping distance. If the offset varies from **0%** which means the QCar stops at **0.6(m)** from an obstacle to **100%** offset which stops the QCar at **1.2(m)** away from the obstacle. To control how closely the QCar tracks the desired lane can be modified using the **Distance To Lane** slider constant which amplifies the desired lane slope from 0% to following the line directly and 100% which will set the QCar close to the center of the lane. Lastly, we have the sliders for the HSV parameters. They work using the following properties:



Every HSV parameter has a mean value and window size. The fine-tuning aspect of this model works as follows: The window size lets you decide how much of the interval **Max-Min** you want to use. A window size of 100% uses the complete range of values between Max and Min. The mean value allows you to modulate where in the color line the average value for your window will be.