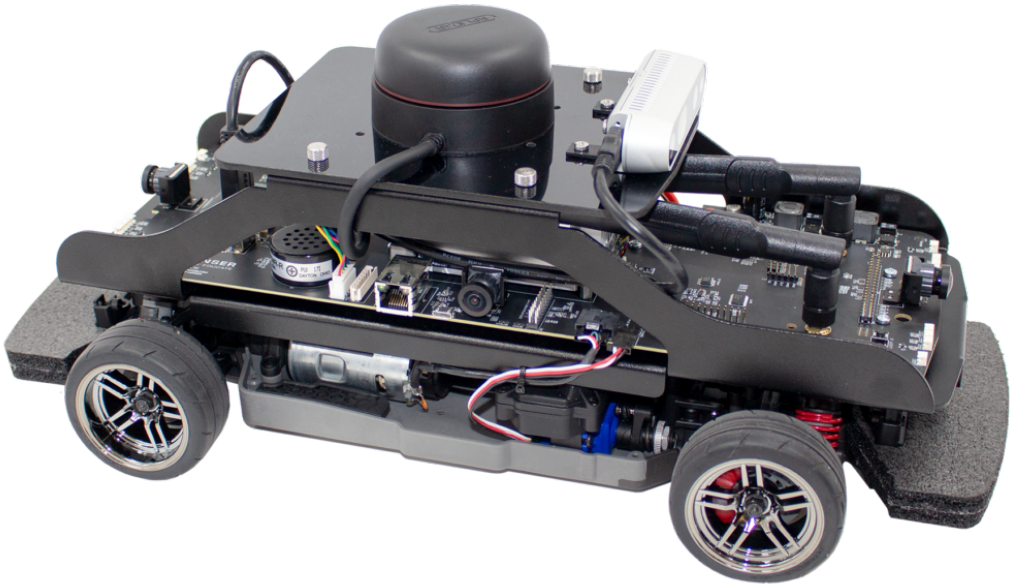
TP QCar Lab

Mayank-Shekhar JHA,
Hugues GARNIER

January 16, 2023

IA2R SIA



Acknowledgements

Authors would like to thanks QUANSER for providing documents on basic details of QCAR, Mr. Pierre JACQUOT for having contributed majorly to the contents presented in this Lab and Mr. Antonin Guyot for adapting and organising MATLAB/Simulink files from Peter Corke's book on Mobile robotics for this lab.

Contents

1 Introduction

Autonomous vehicles are smart vehicles that have the capability to have automatic motions and navigate itself depending on its environments and scheduled tasks. Autonomous vehicle systems may differ depending on the environment it is operating on. Flying and aerial vehicles are autonomous vehicles that operate above ground in higher altitude which usually known as unmanned aerial vehicle (UAV). This lab work focuses on the autonomous vehicles that are operating on the ground which also known as autonomous ground vehicle (AGV).

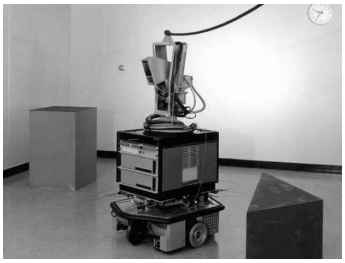


Figure 1: Shavey : Mobile Robotics Pioneer

Figure 2: Curiosity Mars Rover

Autonomous driving has been a real subject of study for years now. Autonomous highway system test were already conducted back in 1950 in America. In 1977, pioneering Japanese driverless car from Tsubuka Laboratory achieved tracking white street lines at 30 kilometers per hour.

Figure 3: Tsukuba Mechanical Engineering Lab, Japan, 1977.

Figure 4: Tesla self-driving car

In developing a successful autonomous ground vehicles, few typical typical challenges need to be addressed by answering the following issues:

- Where am I?
- Where do I want to go?
- How do I get there without hurting myself?

The approach can be summarised in a general manner using the "See-Think-Act" cycle as depicted in Fig. 5 This figure also demonstrates the interaction between various functional modules that are involved in successful functioning of an autonomous vehicle.

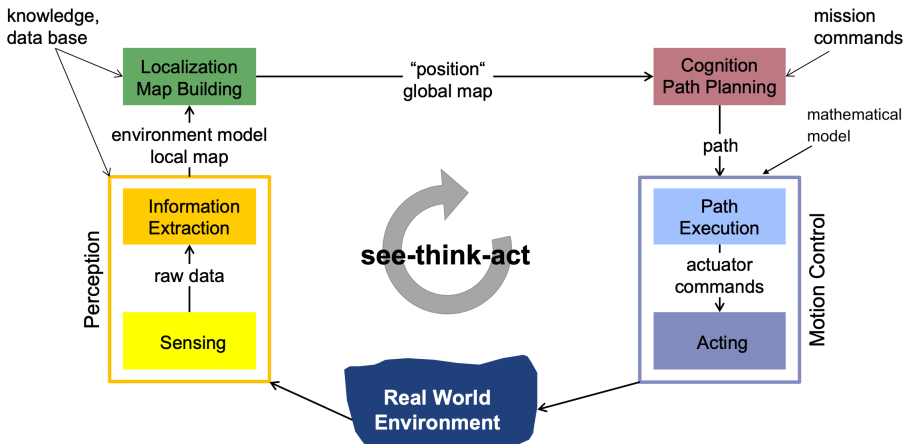


Figure 5: See-Think-Act Cycle, See here for more details: `https://asl.ethz.ch/education/lectures/autonomous_mobile_robots/spring-2021.html`

The various tasks involved can be broken down as shown in Fig.6.
There are three basic stages and modules in the system which are:

- Sensing and Perception – To provide real time data to let the system know the real time location and environment around the vehicle and prepare the raw data in format that is feasible for the system to process.
- Planning – To use the data provided by Sensing Perception to dictate the safe and feasible path for the vehicle to follow.
- Control – Contain control strategies to move vehicle to the desired path. This include actuator control of each sub-system

In this lab, the main focus will on:

- **Trajectory tracking** control in the control phase of the autonomous vehicle system and vehicle model. To that end, the autonomous vehicle will be required to follow a specific path and trajectory given by an on-board planner.
  - **Steering Control** using Pure Pursuit: Path following or trajectory tracking controller is usually developed to ensure the vehicle to follow a predefined path and trajectory by determining and calculating the desired actuating input for the vehicle to follow. This can be a correctional steering input to adjust the vehicle's position in lateral direction or correctional braking or throttle setting to adjust vehicle's motion in longitudinal direction.
  - **Speed Control** : The vehicle is supposed to accomplish the steering using desired speed. The speed control will be effectuated using Proportional-Integral (PI) based controllers.**Perception** based tasks that will include Lane following and obstacle sensing using LIDARs.

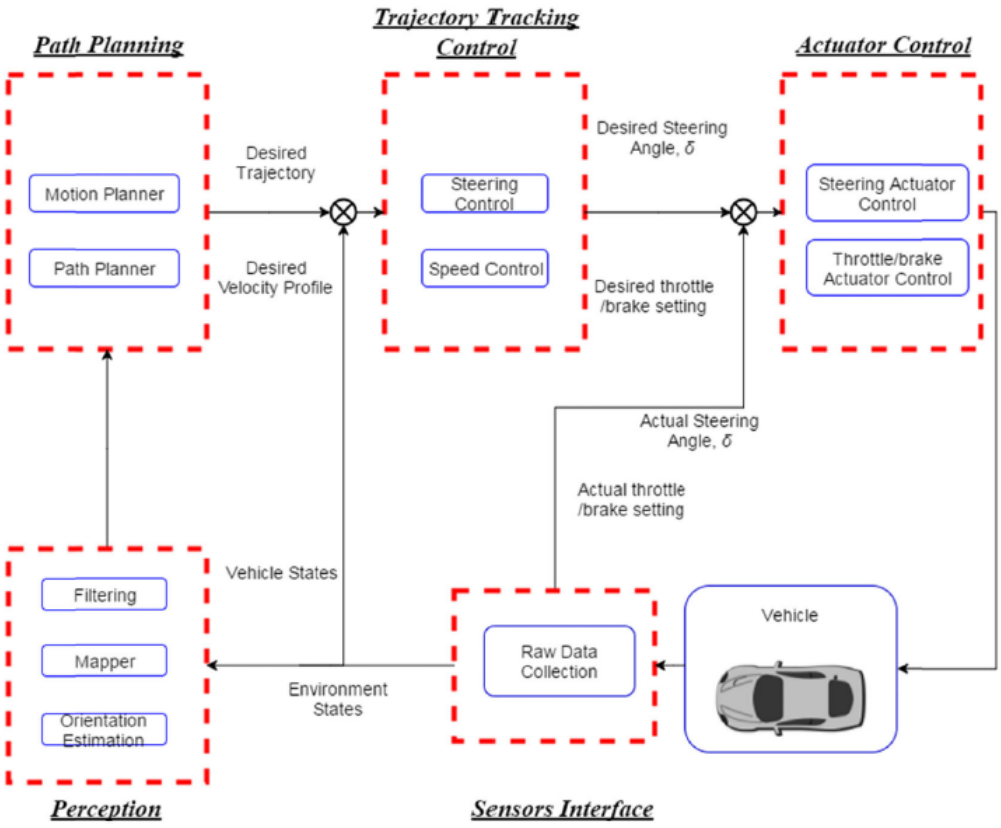


Figure 6: Overview of Interacting Functioning Modules in an Autonomous Vehicle System

- **Perception** based tasks that will include Lane following using cameras and obstacle sensing using LIDARs.

Here is a sneak peek of what the QCar can do : `https://www.youtube.com/watch?v=47cUdBbczUI`

2 The Quanser QCAR mobile platform

The Qcar is a scaled model vehicle equipped with a wide range of sensors, dimensions of a real car on 1/10 scale and weighs 2.7kg .

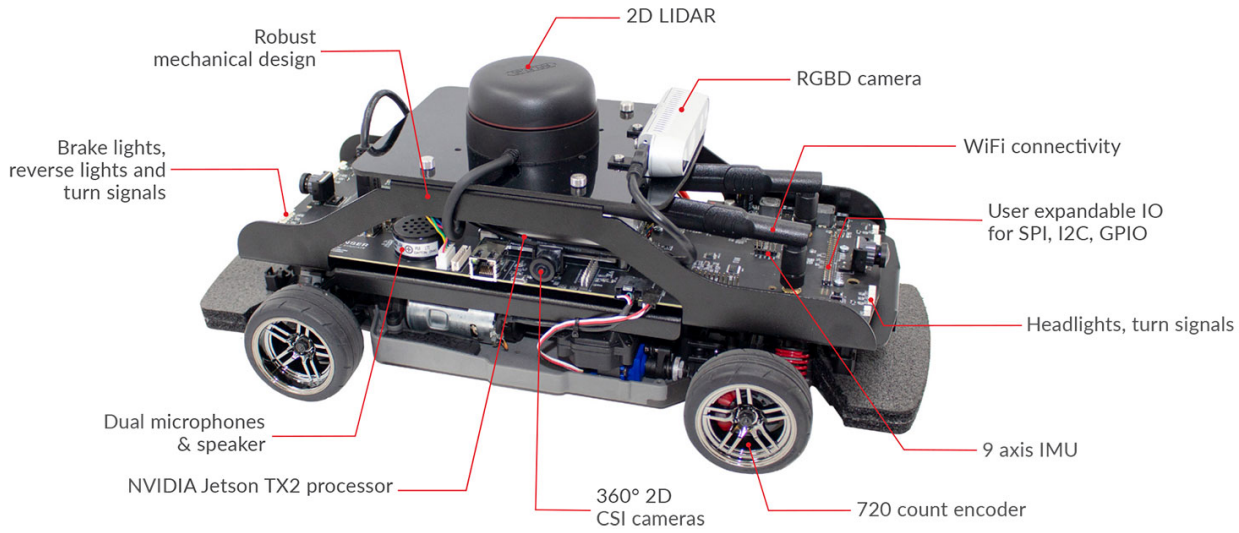


Figure 7: Quanser Qcar

Item	Value
Length	0.425 m
Width	0.192 m
Height	0.182 m
Wheelbase (1)	0.256 m
Front and rear Track (2) (3)	0.170 m
Maximum steering angle	±30° (0.5236 radians)
Tire diameter	0.066 m

Figure 8: Qcar Dimensions

The Self-Driving Car Research Studio comes with a high performance Computer and a router for Wifi conncetion. It is pre-configured to use both 2.4GHz and 5GHz bands for multiple PCs and autonomous vehicles.

Figure 9: Communication between a computer with QUARC installed and the QCAR

Connectivity Test :

- Make sure the yellow ethernet cable is linked to the computer. Do not use the yellow port.
- Turn on the router. After a few minutes, the lights on the front of the router should star flashing with a white light which indicates to the user that the particular ports are active
- Power on the QCar.

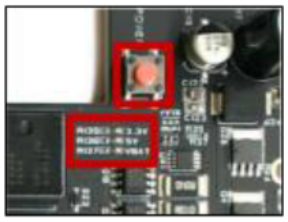
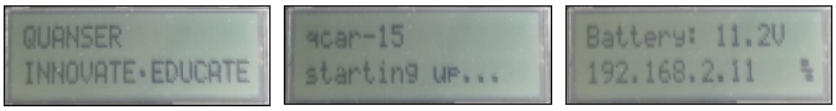


Figure 10: Red Power Buttons and Green power LEDs on the Qcar

Note : Refer to Power.pdf in User Guide to know everything you have to know about connecting or charging the battery

- Use the ping test to confirm connectivity between the QCar and the Computer. Type "ping 192.168.2.XX"" in the Command Prompt using the IPv4 address of the QCar which can be found on the LCD display



a. First bootup message b. Second bootup message c. Default display message

Figure 11: LCD statrtup messages

With the QCar comes QUARC, MATLAB Simulink Libraries which provides blocks that will be used to interact with QCar sensors and actuators.
Note the Qcar battery has a 2 hours autonomy. Make sure the battery is not empty before running programs and do not let the QCar ON if you do not use it. Refer to (II-Power.pdf)

Instructions :

- Do all your tests first with the Qcar on its socle not to damage the Qcar if command issues occurs. Or check if the battery is not empty or connection not lost.
- Refer to the Troubleshooting part at the end of this document when you come through errors.
- Take care of the QCar when you handle it.
- To avoid unfortunate damages, do not input too high speeds to the Qcar.

3 Getting started

This section aims to make you browse and understand the different inputs and outputs of the QCar and the operation in relation to the simulink programs.

3.1 MATLAB 2021a

a. Open MATLAB 2021a and NOT any other version.

b. From **Data Disk D** » go to the Folder `"TP_QCAR"`.
All the files required for this lab are stored in the folder named "TPfiles".

c. Go in the folder named `TP_21_22`. Create a new folder named: `NOM1_NOM2_NOM3`

d. Copy ALL the files from folder "TPfiles" and paste them in the just created folder `NOM1_NOM2_NOM3`

e. Add the folder (and subfolders) `NOM1_NOM2_NOM3` to the path of MATLAB in MATLAB workspace.

f. In MATLAB workspace, set you working directory to: `NOM1_NOM2_NOM3`
From now on, in the rest of the document this will be your "working-directory".

3.2 Configuration : How to generate, deploy and run your Simulink model on QCAR

To understand various steps for code generation, deployment of executable code on QCAR and running QCAR using Simulink, Open the file *BasicIO.slx* in the Folder *I.Explore*

All the tasks asked using the BasicIO.slx has to be done with the Qcar on its provided base.

Here are the instructions to follow EVERY time you need to run a model. Come back to it as soon as you need.
First, make sure both Qcar and Router are on.
In the **Modeling** section of your simulink program, click on Model Settings and then write down the IP written on the **back of your QCar** as shown below :

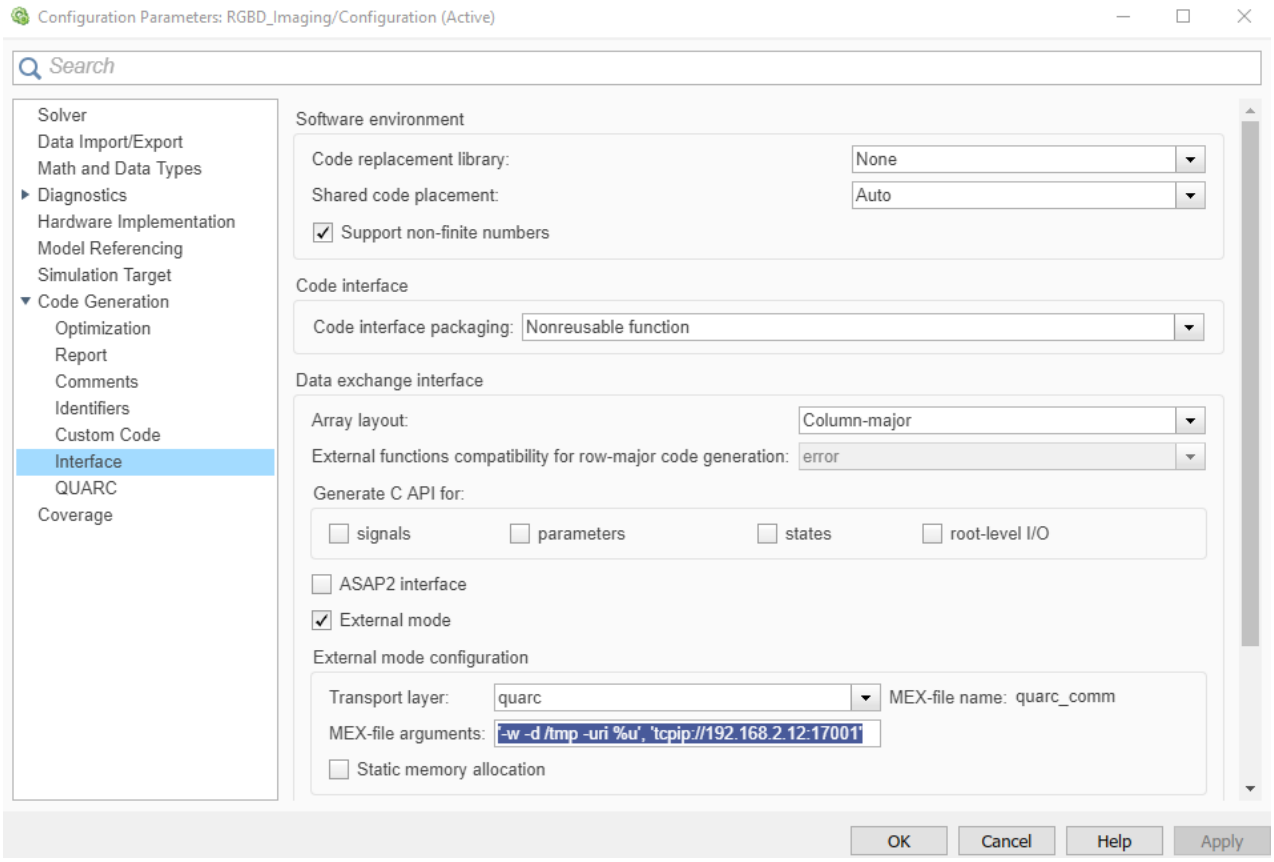


Figure 12: Configuration Parameters

The MEX-file arguments field should be filled with this :

'-w -d /tmp -uri %u', 'tcpip://192.168.2.XX:17001'

The XX number should be 11, 12 or 13. You can see it on the back of your QCar when it is turned ON (See LED display of QCAR).

To build code, deploy and run the Simulink program on your QCar, follow the instructions:

a. To build, download, connect, and start the model for real-time external operation, click the Monitor Tune button on the HARDWARE tab (see Fig.13)

b. If the model is successfully built and downloaded to the target. Simulink will automatically connect and start the real-time code on the target and the Monitor Tune button will automatically change to a Stop button (See Fig. 14)

See complete details (Step by Step procedure) here: `https://docs.quanser.com/quarc/documentation/quarc_procedures.html#external_mode_operation`

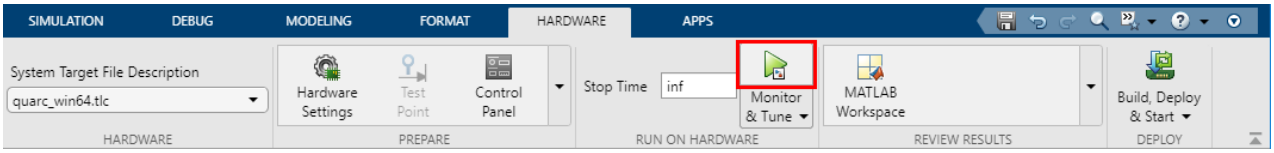


Figure 13: One step process for code generation, code deploy and Run Simulink (QUARC+QCAR) Model in External Mode

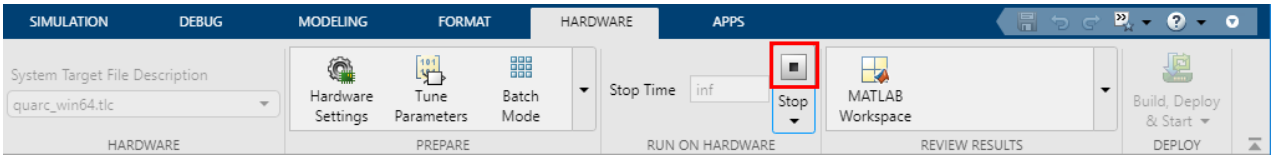


Figure 14: Running Simulink (QUARC+QCAR) Model in External Mode

If an error occurs, please refer to the Troubleshooting Section at the bottom of the document.

Refer to these steps to FOR EVERY NEW MODEL YOU RUN

3.3 Basic Input Output Tests

Open the file BasicIO.slx in the Folder I.Explore

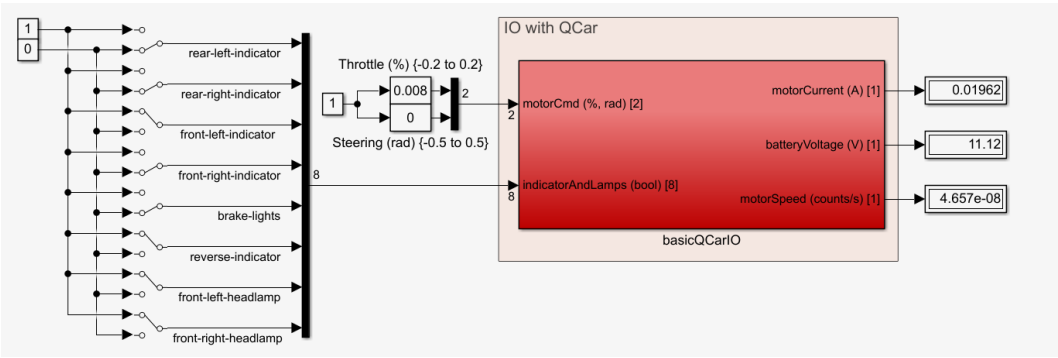


Figure 15: Basic I/O simulink file

The model used in this Lab will use two blocks to read or write information to the Qcar. The blocks are named HIL-Read and HIL-Write. You can see them when opening the red basicQcarIO block.

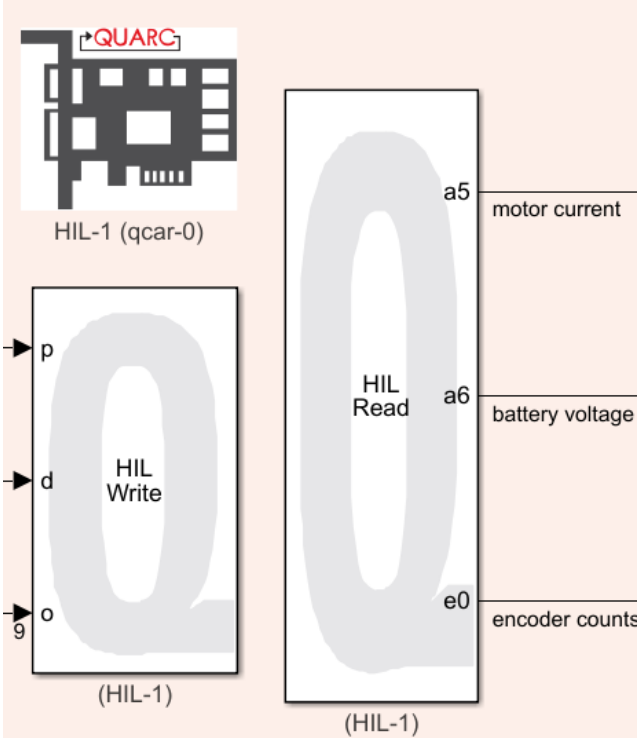


Figure 16: HIL-Read and HIL-Write Blocks

The HIL Read/Write blocks read/write the specified channels every time the block is executed. These blocks can read/write more than one type of channel at the same time.
The channels correspond to the types of sensors or actuators on which to act or read.

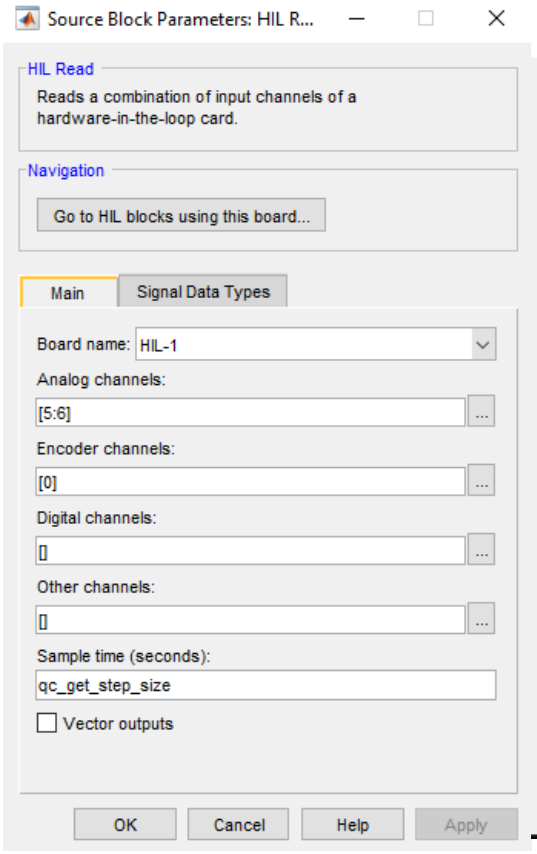


Figure 17: HIL-Read

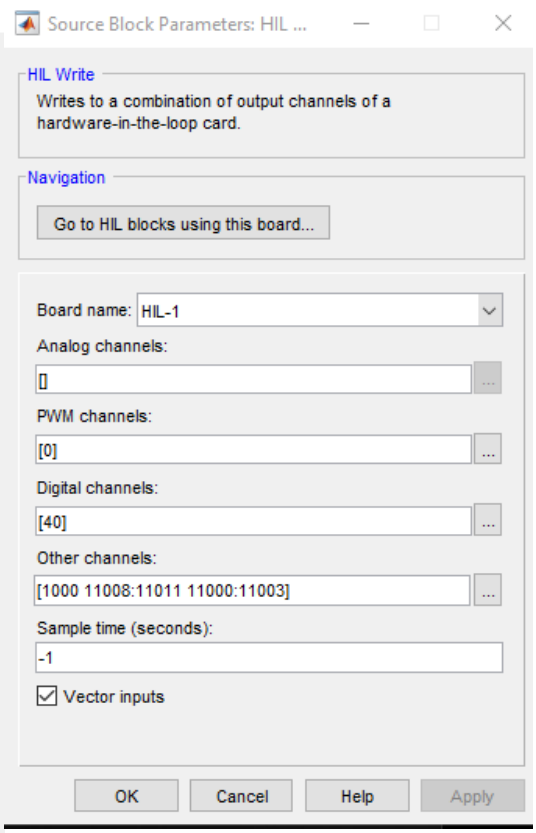


Figure 18: HIL-Write

Here, on the HIL-Read block, the channel 5,6 in the Analog channels section corresponds to the motor current and battery voltage sensors while the **Encoder channels filled with a 0 correspond to the encoder count sensor**, which is the main output of this Lab. The main inputs and outputs that will be used in this model and in this lab are :

- **The PWM input :** PWM channel 0 drives the motor. Channels 1 through 7 are user PWM channels available. The PWM value is a value in % of duty cycle.
- **The steering input:** The channel 1000 from the section Other channels is used to act on the steering servo with a command in radians.
- **The lights :** Other channels from 11000 to 11011 act on all the lights port available on the QCar as depicted below.

Additional Information!
(PWM) for motor control: As its name suggests, pulse width modulation speed control works by driving the motor with a series of "ON-OFF" pulses and varying the duty cycle, the fraction of time that the output voltage is "ON" compared to when it is "OFF", of the pulses while keeping the frequency constant.
The power applied to the motor can be controlled by varying the width of these applied pulses and thereby varying the average DC voltage applied to the motors terminals. By changing or modulating the timing of these pulses the speed of the motor can be controlled, ie, the longer the pulse is "ON", the faster the motor will rotate and likewise, the shorter the pulse is "ON" the slower the motor will rotate.
In other words, the wider the pulse width, the more average voltage applied to the motor terminals, the stronger the magnetic flux inside the armature windings and the faster the motor will rotate. See more details here if required: `https://www.electronics-tutorials.ws/blog/pulse-width-modulation.html`

The figure below show all the channels of the HIL-Read Write block that will be used are in the "other channel" type of block, except the PWM of the motor which is the 0 channel of the PWM channels.

Other Output Channel	*Measurement Description*	*Units*
1000	Steering angle	(rad)
11000	Brake lights LED intensity	(%)
11001	Reverse lights LED intensity	(%)
11002	Left headlight LED intensity	(%)
11003	Right headlight LED intensity	(%)
11004, 11005, 11006, 11007	User LED0-LED3 intensities	(%)
11008	Left rear signal LED intensity	(%)
11009	Right rear signal LED intensity	(%)
11010	Left front signal LED intensity	(%)
11011	Right front signal LED intensity	(%)

Figure 19: "Other" Channels for the HIL-Write block

More details available here : `https://docs.quanser.com/quarc/documentation/qcar.html`
Task 1: Keep the car on its base. Run the file and click on the Throttle and Steering blocks to send value to PWM 0 channel (See top of Fig. 20 and generate a steering angle command (See bottom Fig. 20)

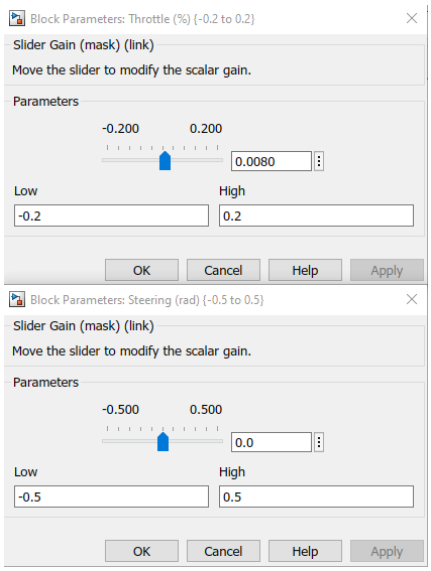


Figure 20: Basic IO throttle and steering command windows

Task 2: Explain all the outputs. Verify the battery voltage as indicated in the Simulink model and the LCD display of QCAR.

3.4 Keyboard Manual Drive

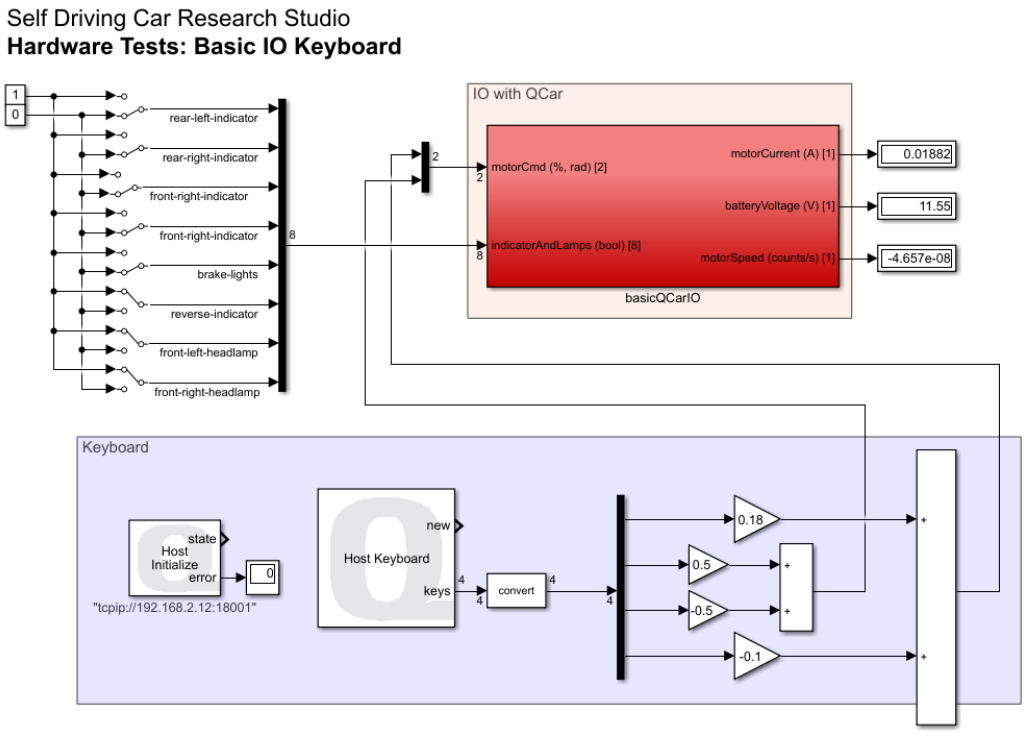


Figure 21: Keyboard simulink program

Task 1: : Launch the program and drive the QCar using your Keyboard.

3.5 Explore the sensors : LIDAR

Lidar is an acronym of "light detection and ranging" or "laser imaging, detection, and ranging". Lidar sometimes is called 3-D laser scanning, a special combination of 3-D scanning and laser scanning.
Lidar is commonly used to make high-resolution maps, with applications in atmospheric physics,laser guidance, airborne laser swath mapping (ALSM), and laser altimetry. The technology is also used extensively in control and navigation for autonomous cars and for the helicopter Ingenuity on its record-setting flights over the terrain of Mars.
Open the file *Lidar_Point_Cloud.slx* This example will capture LIDAR data and send these data to a polar plot to generate a point cloud map as described below.

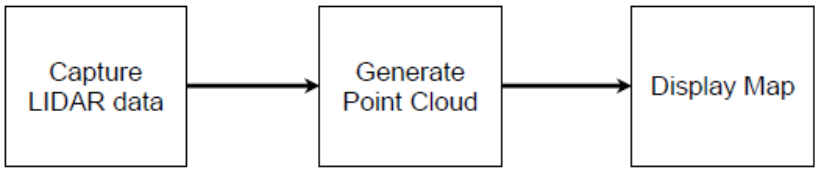


Figure 22: LIDAR Point Cloub diagram

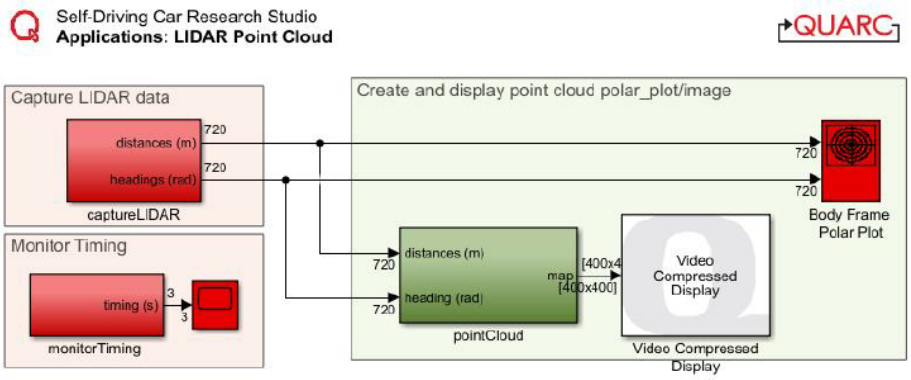


Figure 23: LIDAR Point Cloud Simulink implementation

Run the model.
To run a simulink model, refer to the subsection "How to run your model" above.
You should see two windows displaying the LIDAR data in a different way. The Body Frame Polar Plot block directly plot the points from the distance information captured by the LIDAR.

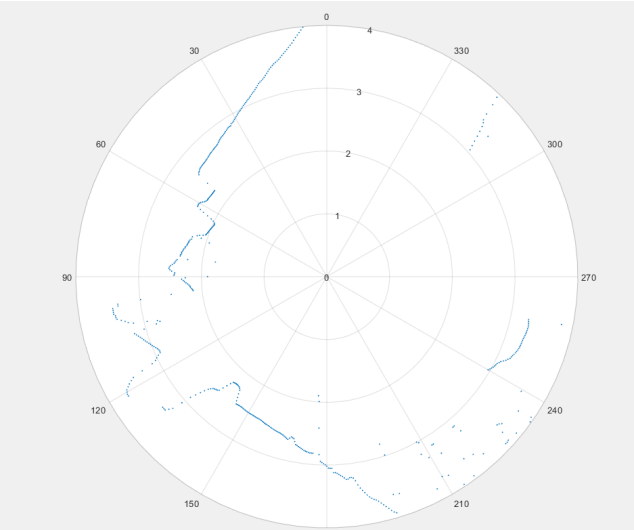


Figure 24: LIDAR Polar Plot

Then a pointCloud block as been build to generate a map using the distance and the angle given by the LIDAR and to display it using a Video Compressed Display block provided by Quanser.

Task : Check the range of the LIDAR and its precision using a piece of paper or your hand. Bring a large object (Carton box) in front of the QCAr and observe the changes in the LIDAR plot. Can you see yourself in the LIDAR plot?

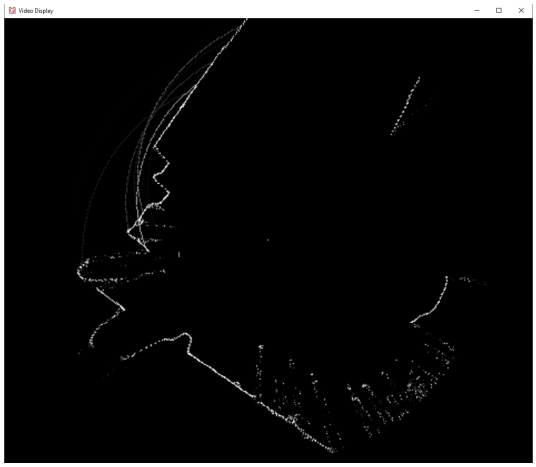


Figure 25: LIDAR Point Cloud Video Display

3.6 Explore Sensors : CSI Cameras for 360 Vision

This example captures the images from the four CSI cameras at the same resolution and frame rate and stitch them together.

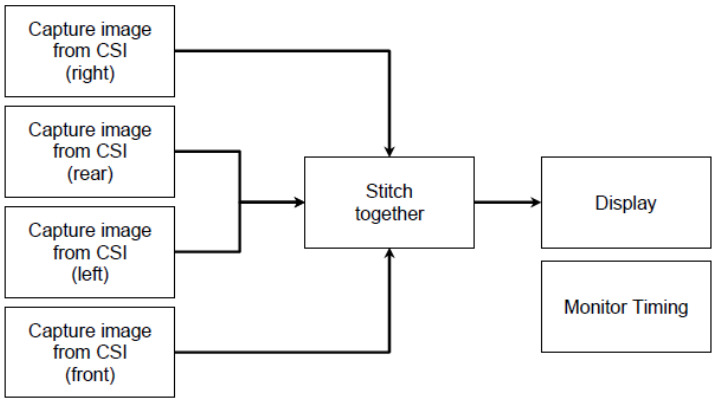


Figure 26: 360Vision diagram

To do so, four Video Capture block from the QUARC Library are used and connected to each CSI Camera.

capture360 Module:

This subsystem captures all four CSI images and also outputs a combined all new signal.

Outputs:

allNew?	(bool)	:	True if all the frames are new, false otherwise
right		:	480 x 640 x 3 image
rear		:	480 x 640 x 3 image
left		:	480 x 640 x 3 image
front		:	480 x 640 x 3 image

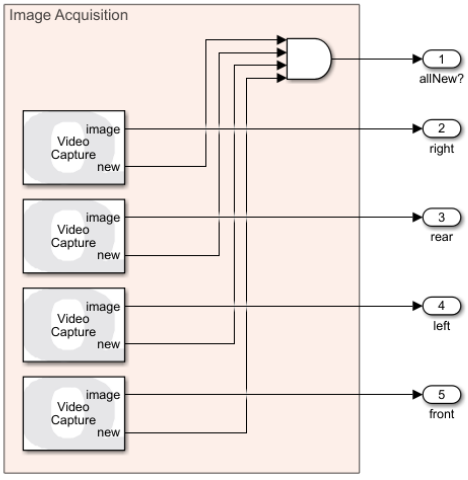


Figure 27: Capture of Each camera

Task: Open the file *Imaging_360.slx* and run it.
Here is what you should get as a result.

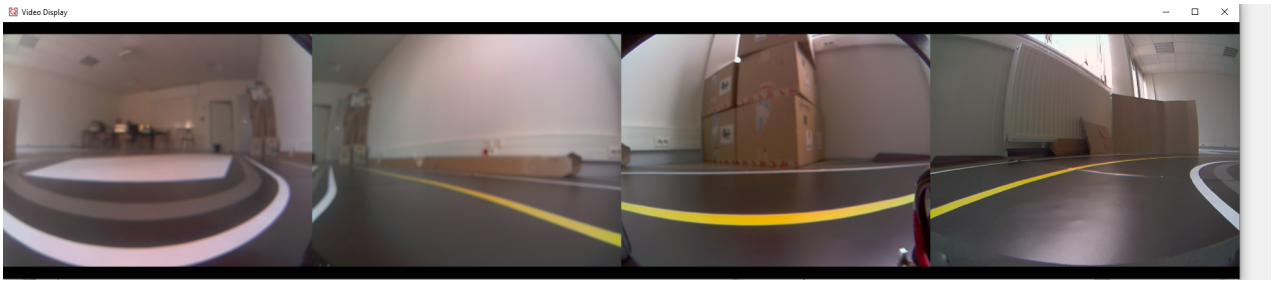


Figure 28: Right — Rear — Left — Front View

3.7 Explore Sensors : RGBD Camera

Open the file *Intel_Realsense.slx*.

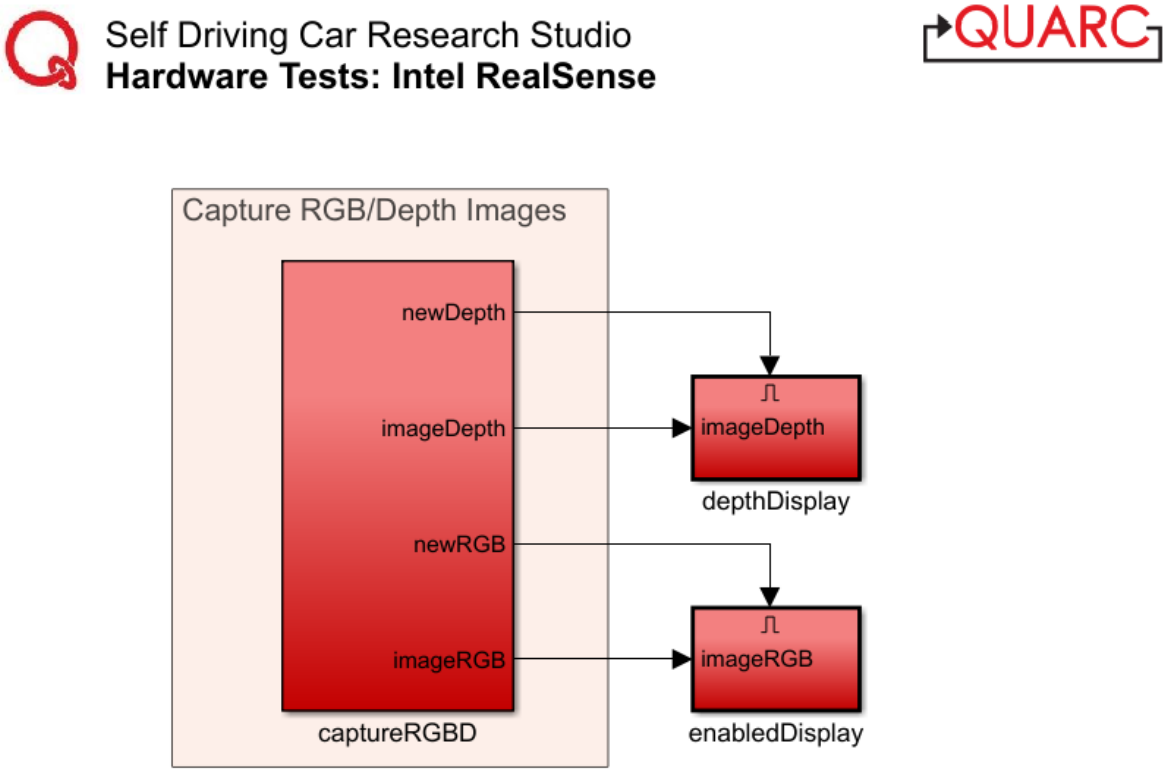


Figure 29: Intel Realsense Sensor Test

Task : Run the file and explain what each block does. You must see two windows, a RGB one and another with depth information.
Find out if darker pixels in the Depth Image means shorter or longer distance?

4 Mobile Robot Vehicle

This chapter is an extract from the well known book on Mobile Robotics called "Robotics, Vision and Control Fundamental algorithms in MATLAB: Second Edition" by Peter Corke (see here: `https://petercorke.com/rvc/home/`) You are advised to read the Chapter extract and perform each Example (exercises) as shown in the book using By-cycle model in Simulation. Namely, you are advised to understand and perform: Lane change, Moving to a point,following a line and Following a trajectory.
The Simulink files can be downloaded from course webpage here: `http://w3.cran.univ-lorraine.fr/mayank-shekhar.jha/?q=content/teaching-activities-enseignements`.

The authors are thankful to Mr. Antonin Guyot (5th year IA2R, 2023) whos contribution has been immense in organising, structuring these code and files).
Note: In the later sections, these files can be used to perform similar exercise in real time with QCAR

5 Path Following (Reference trajectory follower)

A good challenge for autonomous vehicle is to make them follow a desired trajectory, computed in real time in line with the current changes of the road.
The Qcar will be asked to follow arbitrary predefined trajectories by developing controllers that can drive the Qcar along a path.
The two parameters that must be controlled are the speed of the Qcar and the angle of the front wheels (steering angle).
No perception/vision sensors (cameras) will be used to follow the path desired.

5.1 Information

5.1.1 About Bicycle Model

To follow predefined paths or trajectories accurately, the Qcar needs to know with precision where it is. To do so, different car-like vehicle model does exists, with more or less complexity.

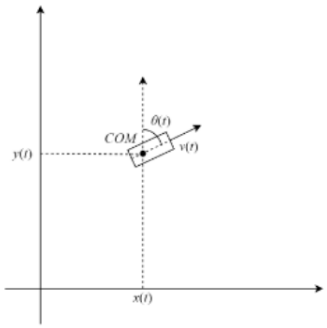


Figure 30: Unicycle Model

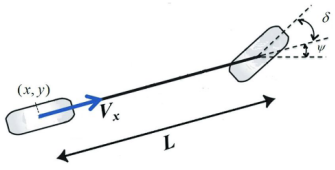


Figure 31: Bicycle Model

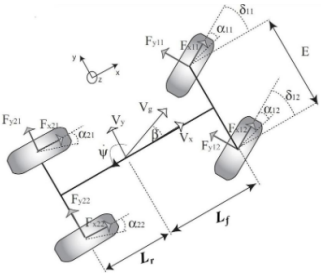


Figure 32: 4 wheels model

The bicycle model is a reduced kinematic model of vehicle, in which the two front and rear wheels are replaced by one front and one rear wheel respectively.
The vehicle is described with 2 sets of Cartesian coordinates, local (x-y) and global (X-Y). Local coordinates are set to be fixed on the vehicle body. Global coordinates, on the other hand is set to be fixed on the earth coordinates. Local coordinates are always regarded as moving coordinates with respect to the global coordinates. Fig. 33 shows both the frameworks and nomenclature adopted here. The vehicle moves on the plane and its coordinates are described by the vector (x,y,Ψ), where x, y is the position of the centre of gravity and ψ is the orientation (heading) of the vehicle.
Steering angle of the front wheels is denoted by δ. It is noted that $v_x = V_{car}$ i.e. velocity of the car in local coordinates is denoted by v_x .
A basic assumption of the bicycle model is that the inner slip, outer slip and steer angles are equal.
Task: Consult the Fig.33. Given that $lr = lf = L/2$ with L being the length of the wheelbase, precise which is local framework? which is the global framework? Which is the steering angle? Which is the heading angle? What is the difference between heading and steering angle? If r represents the change in yaw i.e. r is the yaw rate, express r in terms of Ψ .

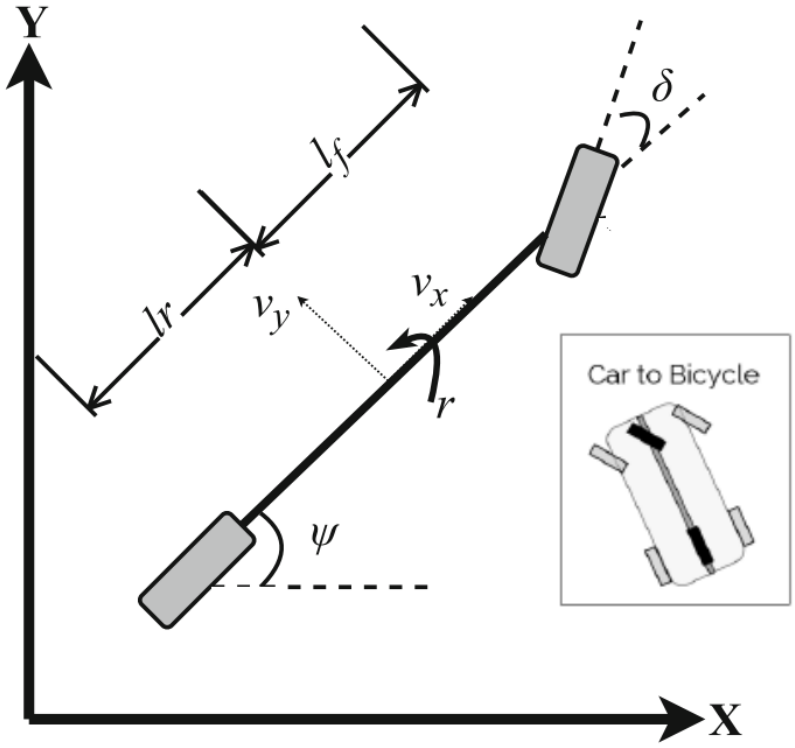


Figure 33: Car to bycycle Model with global and local references depicted with $v_x = V_{car}$ and $lr = lf = L/2$

Using such a model aims to know with precision the Pose of the vehicle using known input parameters.

Pose means Position and Orientation.

i.e. the Position in world frame coordinates (X,Y) where X and Y are the axis of the horizontal plane. The angular Orientation of the vehicle Ψ which is the rotation around the Z axis.

In world frame coordinates, the kinematics relation lead to equations of motion as :

$$\dot{X} = V_{Qcar} \cdot \cos\Psi - (L/2) \cdot \dot{\Psi} \cdot \sin\Psi$$
$$\dot{Y} = V_{Qcar} \cdot \sin\Psi + (L/2)\dot{\Psi} \cdot \cos\Psi$$
$$\dot{\Psi} = \frac{V_{Qcar} \cdot \delta}{L}$$

where L is the distance between the front and the rear wheel. X,Y and Ψ are the coordinates in global framework, V_{car} is the velocity of the car in local coordinates (velocity of

the centre of gravity) and $\dot{\Psi}$ is the yaw rate.
In this model, the position (X,Y) is the position of the middle of the bicycle, that is why there is a L/2 argument.

This is called a kinematic model since it describes the velocities and not the forces or torques that causes the velocity.
The orientation Ψ can be known with sensors like gyroscopes or can be deduced from the steering angle.
The following simulink block implements the kinematics equations of a bicycle model to simulate the motion of the vehicle.

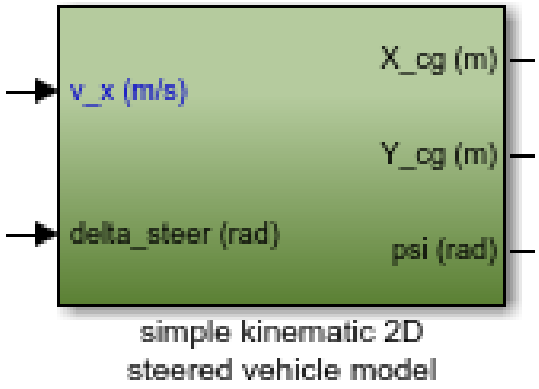


Figure 34: Bicycle Model Kinematics simulink implementation

Knowing the linear velocity of the Qcar and the steering angle of the wheels, with the approximation of a bicycle model, we are able to know the Pose (X,Y,Ψ) of the Qcar.

5.1.2 Trajectory Control Strategies

Trajectory tracking control in autonomous vehicles is mainly aimed to provide sufficient steering input as well as throttle and braking input to control the direction and speed of the vehicle to guide the controlled vehicle along a path. The focus in this lab is over the control strategy used in order to control the steering angle, δ and vehicle speed along the pre-defined path.
To that end, we consider the simplest and the most popular type of controller used in autonomous steering control that are based on geometries of the vehicle model.

5.1.3 Follow the carrot

This is the most basic geometric controller, based on the analogy of riding on a donkey and guiding the donkey using a carrot directed to the intended direction. In this approach, the nearest point on path at one look-ahead distance, L_{carrot} is chosen to be the instantaneous goal point, known as the "carrot point" (see Fig. 35) Orientation error with respect to the global coordinates, Ψ_e, is determined by the difference between the vehicle orientation, Ψ

and the orientation of carrot point, Ψ_e as shown in Fig. 8. The overall steering command can be expressed as $\Psi_e = \Psi_e - \Psi$. The steering angle required to correct the vehicle heading and orientation can be determined using proportional controller as $\delta = K_carrot * \Psi_e$.

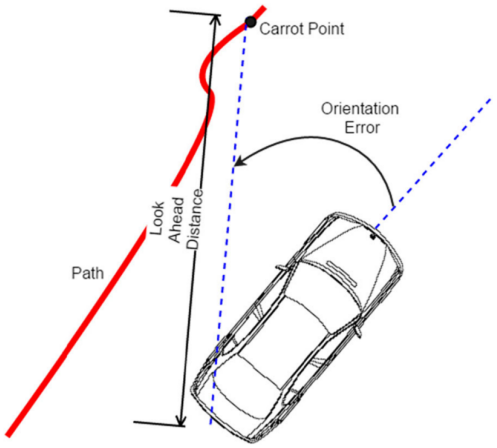


Figure 35: Representation of follow the carrot method

5.1.4 Pure Pursuit

Pure pursuit is the most popular geometric controller among the existing methods.
The vehicle is understood to be chasing a moving point throughout the duration. Therefore, it is constantly "in pursuit" towards the moving point (carrot point). In understanding pure pursuit method, it is the extension of the earlier follow-the-carrot method where a carrot point is chosen based on the nearest point on path within one look ahead distance from the vehicle.

Pure pursuit method extends this approach at this stage by fitting a smooth circular curve from vehicle to the carrot point for the vehicle to follow. This circular curve is defined such as its line passes through two points; the vehicle position and the carrot points. This curvature will ensure smooth steering for the vehicle and reduce the oscillatory nature of the basic follow-the-carrot method.

Given that the circular arc goes through the carrot point and control point on the vehicle with centre at O; a local coordinate axes with origin on the control point; and intended steering angle δ as shown in Fig. **??** Note that the triangle produced is an isosceles triangle with common length R.

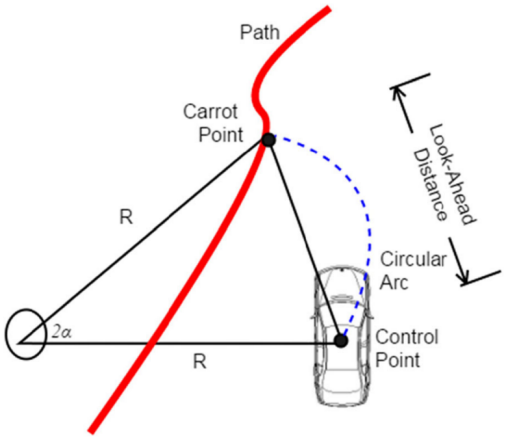


Figure 36: Representation of follow the carrot method

Using the rule of sine involving look-ahead length, l_d and the angle α between the vehicle's heading vector and look ahead vector, the steering angle can be derived as:

$$\delta = \tan^{-1}\left(\frac{L}{R}\right) = \tan^{-1}\left(\frac{2L\sin\alpha}{l_d}\right)$$

where L is the wheelbase.
The property "LookAheadDistance" decides how far the look-ahead point is placed.
The "LookAheadDistance" is the main tuning property for the controller. It computes a look-ahead point on the path, which is an instantaneous local goal for the vehicle. The angular command is computed based on this point.
The figure below shows the robot and the look-ahead point. As displayed in this image, note that the actual path does not match the direct line between waypoints.

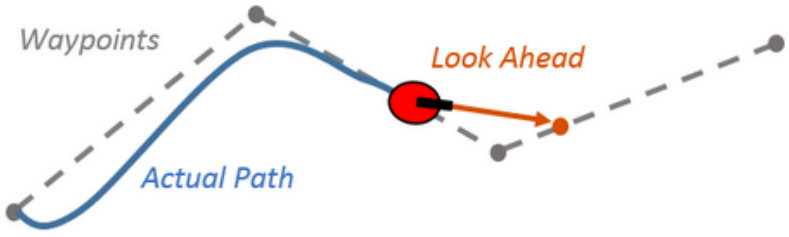
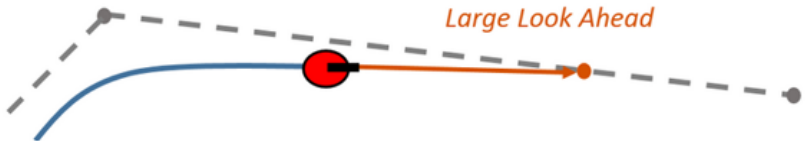


Figure 37: Pure Pursuit

Changing Lookahead distance has a significant impact on the performance of the algorithm. A higher look-ahead distance results in a smoother trajectory for the vehicle, but can cause the vehicle to cut corners along the path.



Too low of a look-ahead distance can result in oscillations in tracking the path, causing unstable behavior.

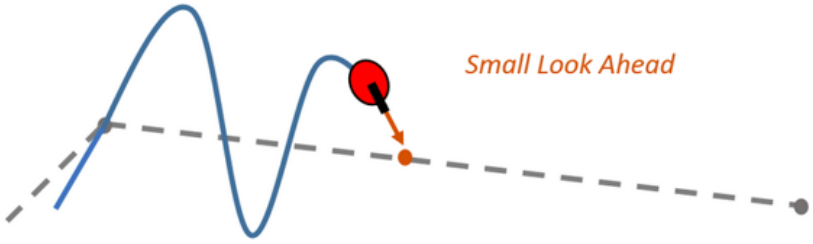


Figure 38: Large Look-Ahead Distance

LookAheadDistance property has to be tuned for your system. Different linear and angular velocities will affect the Pure Pursuit response as well.
Here is how the Pure Pursuit block looks like in Simulink.

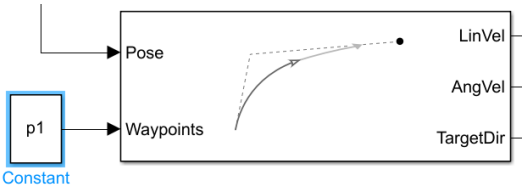


Figure 39: Small Look-Ahead Distance

It needs as inputs the current Pose (position and orientation) of the Car and the waypoint towards the traget position (aim).
The input waypoints are [x y] coordinates, which are used to compute the robot velocity commands. The robot's pose is input as a pose and orientation (psi) list of points as [x y psi]. The positive x and y directions are in the right and up directions respectively (blue in figure). The theta value is measured counterclockwise in radians from the x-axis (robot currently at 0 radians).

The outputs are :

LinVel : The Linear Velocity of the vehicle (constant to fix in the parameters of the block)

Note that the speed of the car has a considerable influence on its behavior and thus, on the efficiency of the Pure Pursuit.

AngVel : The Angular Velocity command for the vehicle (this will not be used in this lab)

TargetDir or α : The Angular Direction of the Target Point computed by the Pure Pursuit Algorithm

As a limitation, this pure pursuit does not stop the robot at a point. A distance threshold with respect to a goal location should be applied to stop the robot near the desired goal.

5.1.5 About Simulink Model

This simulink model called PathFollowing.slx uses a bicycle model and the Pure Pursuit method to make the QCar follow given waypoints.
It uses HIL-Read block as entry to collect the encoder information, and the HIL-Write block as output to command the QCar in speed and steering.

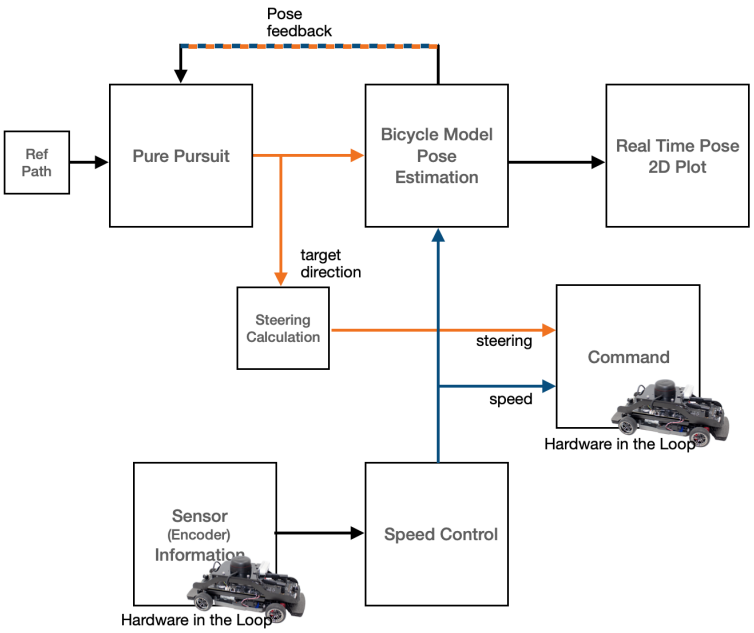


Figure 40: Path Following Model

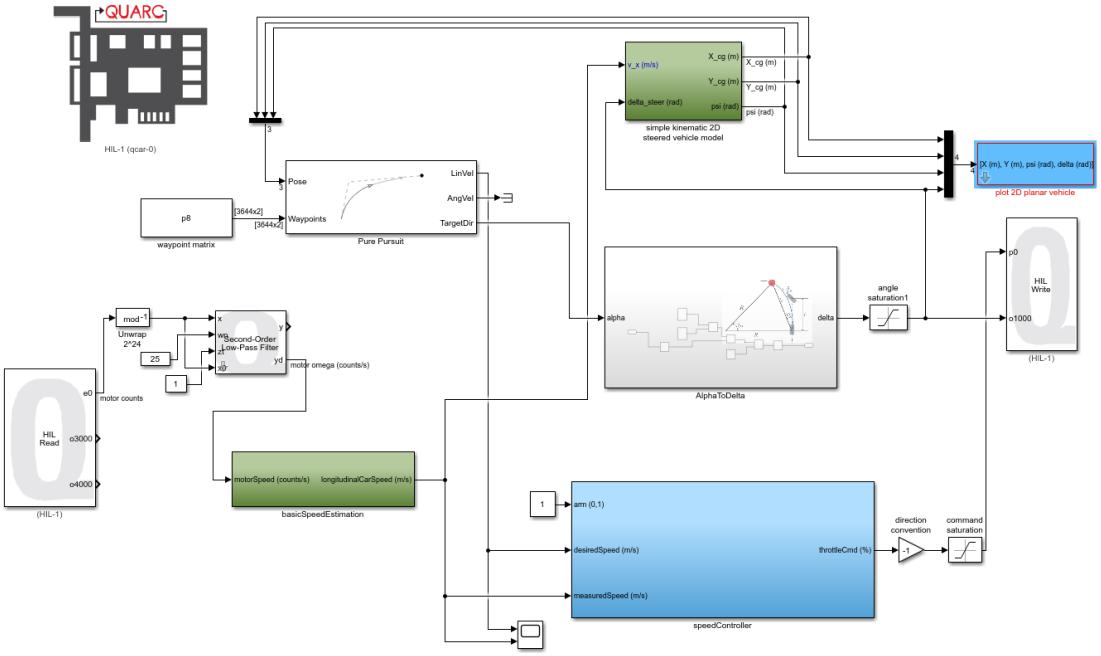


Figure 41: Path Following Simulink Model

About HIL (Hardware in the Loop) : HIL is a technique used in the development of embedded systems. It is the hardware being linked to a computer reproducing the environment. It allows the hardware to be tested in different and numerous simulated environments.

The TargetDir (α) angle given by the Pure Pursuit block is used to compute the steering command (δ) of the front wheels. Alpha is the angle of the direction to follow while delta is the angle the wheel must steer to do so.

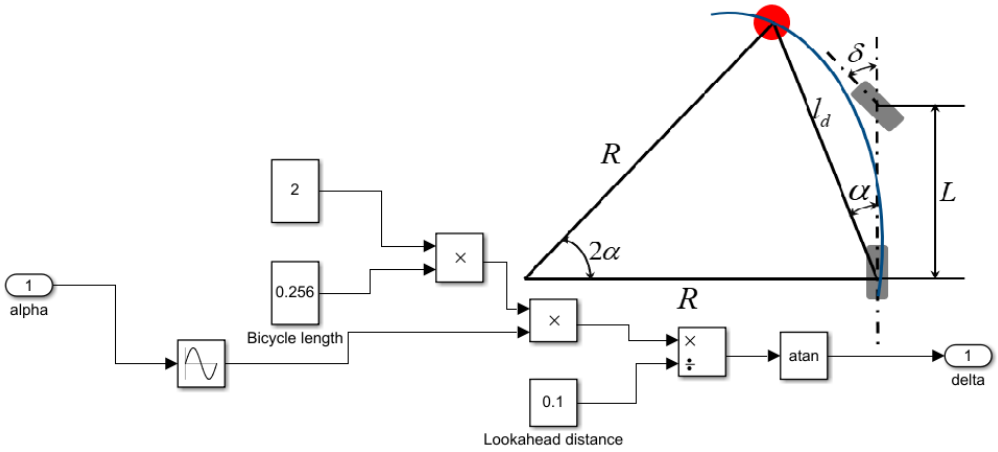


Figure 42: Steering (δ) calculation from TargetDir (α)

The LinVel output of the pure pursuit is used as a desiredSpeed value for the speedController. The Speed Controller is a feed-forward PI Controller used to generate the desired throttle command.

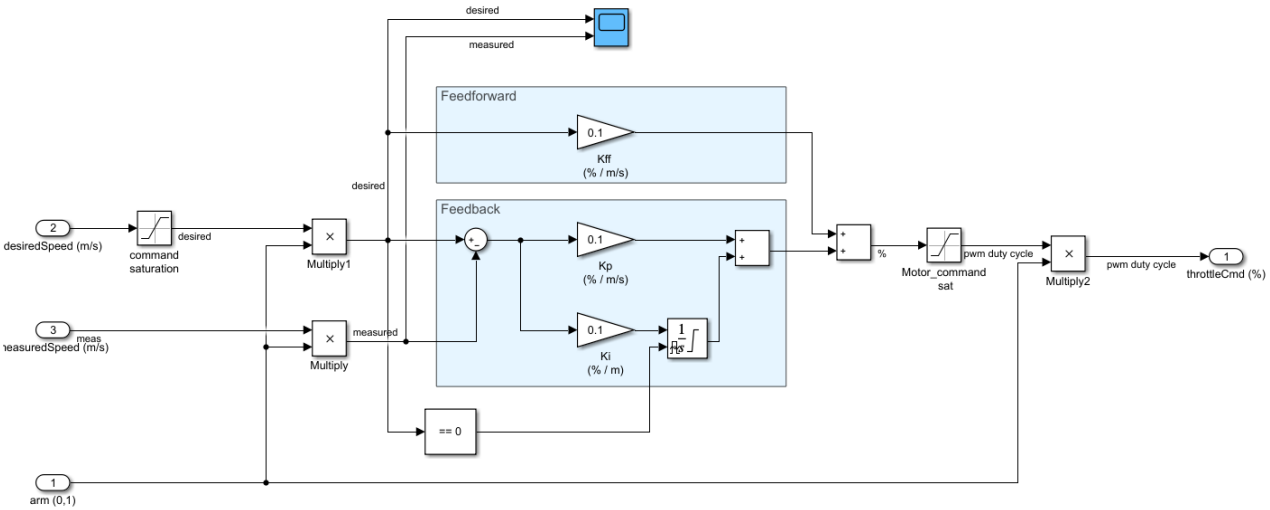


Figure 43: speed Controller

Both steering δ and throttle command are send to the **HIL-Write** block, thus to the QCar. On the other side, the HIL-Read block is linked in Real Time to the sensors of the QCar, with the e0 output being the count of the motor encoder.

5.2 Experiment

Every file needed in this section are placed in the Folder II.PathTracking

5.2.1 Generate a Reference Path

The first objective will be to create a path for the Qcar to follow. As said before, the Pure Pursuit box needs waypoints which are the targets to follow. These waypoints are x,y coordinates.

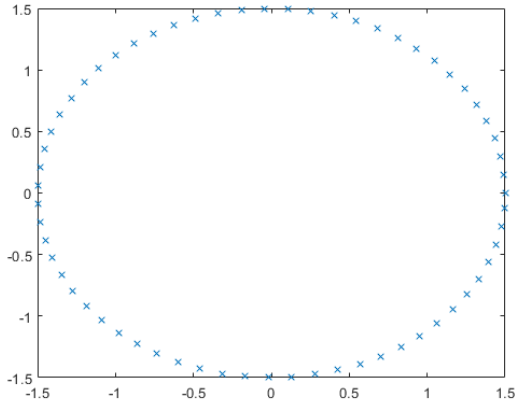
> The QCar initial position is **(0,0)** and is turned toward the **negative y axis**.

In the PathGeneration.mlx file, do these different task :

Task 1: Generate and **plot** a Matrix of 4 waypoints to make the Qcar follow a straight line of 3 meters. The matrix should be 2x4 and start in (0,0) position. Name it p1.

```
LigneDroite = 4×2
     0     0
     0    -1
     0    -2
     0    -3
```

Task 2: Generate and **plot** waypoints for a 1.5m radius circle and a step of 0.1 radians. You must get a 64x2 matrix. Name it p2

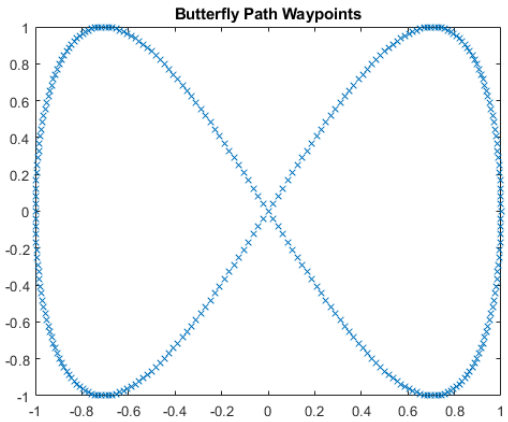


Task 3: Generate and **plot** waypoints for a butterfly of 4 meters high and 4 meters wide. Name it p3

Example of butterfly path generation :

```
freq = 2*pi/30;
Ts = 0.033; % Sampling time
t = 0:Ts:30; % Simulation time
xRef = 0 +wingWidth*sin(freq*t);
yRef = 0 - wingHeight*sin(2*freq*t);
plot(xRef,yRef);
title('Butterfly');
```

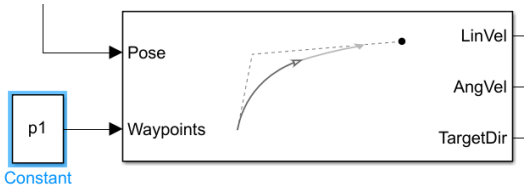
```
p3=[0  xRef;0  yRef]'
```



Generate the same butterfly but 2 meter long and wide so it can suit to the room size. Name it p4

5.2.2 Run the Simulink Model

Open the simulink file called PathFollowing.slx.



Place the Qcar on the center of the Room. Make sure there is enough space available to follow the path you just created and check that model configuration contains the IP of the QCar. Then do the following tasks :

Task 0: Run the file setup.m

Task 1: Run the simulink program and observe the behavior of both the simulated animation and the real Qcar on the floor for p1 as waypoints. What happens ? Is the path followed precisely ?
Advice : The Qcar will not stop after reaching the final waypoint. You will have to click on the stop button in the simulink toolstrip.

Task 2: Repaeat the same excercise with p2, p3 and p4 as way points separately.

Task 3: Keep working with p4. Tune the Lookahead distance parameter inside the Pure Pursuit block to get a better trajectory.
Do not forget to change the Lookahead distance on both steering calculation and Pure Pursuit blocks.
You can test the QCar on floor if the simulated results seems satisfying.

Task 4: Propose a better Look-ahead distance and explain why the behavior is better?

5.2.3 PI Controller

The speed regulation is done with a PI Controller. This speed Controller has 3 Parameters, Kp, Ki, the proportional and integer gains of the PI controller, and a Kff gain for the feed-forward loop. This last gain will remain untouched.

Task1: Study the controller structure. What kind of controller is it?
Task2: Use the scope to observe the performance of the controller.

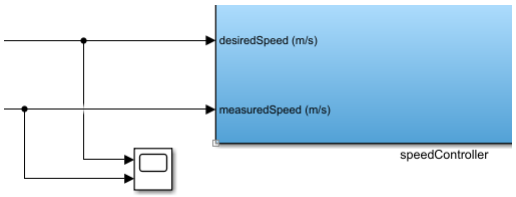


Figure 44: Scope to compare speed input and output

Adjust both Kp and Ki gains to get what is - according to you - the best speed control and explain how and why you selected these.

Task 6: Having seen the above examples of HIL, and "lane-change" "point-to-point", "following a line" from the book extract (by Peter Cork), adapt the above 3 tasks in real time on Qcar. This will require mixing HIL concepts as well as bycycle model used above.

6 Vision Based Strategies

6.1 Image Processing for Stop Sign detection

Open the file *RGBD_Imaging.slx*. This example uses captured images from the Intel RealSense's RGBD camera to detect the red color of a stop sign. After a stop sign is detected, the Depth (D) value from the RGBD Camera is used to compute the distance between the Qcar and the Stop Sign.

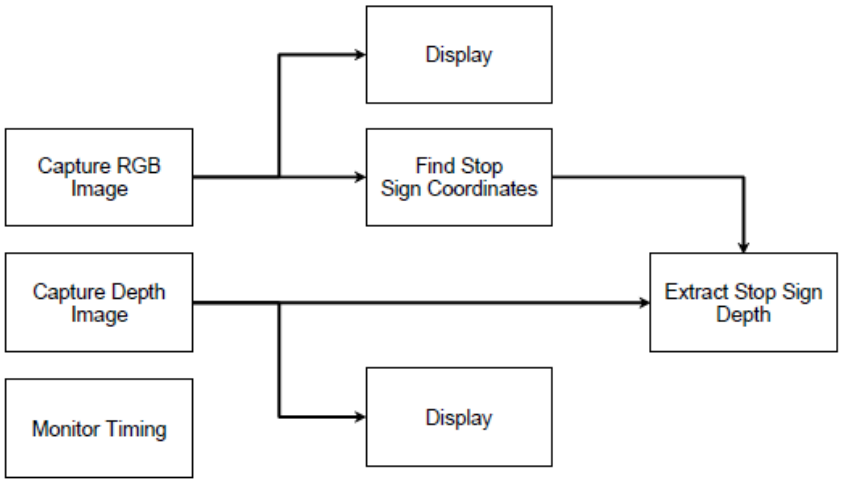


Figure 45: RGBD Imaging diagram

This Simulink program founds the stop sign location thanks to the RGB image and then founds the depth in the area where the sign has been detected thanks to the depth sensors. To properly find the Stop sign, we first have to know that the red color from a stop sign has a Hue value of 0 as depicted on this Hue Wheel

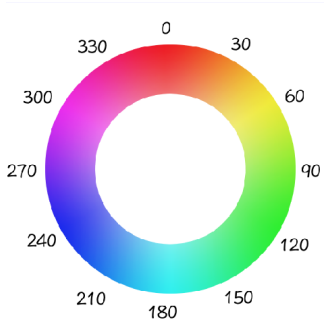


Figure 46: Color wheel reference

Then, the image is turned to a binary image with a thresholding with reference Hue a red color and saturation and value between 100 and 255.

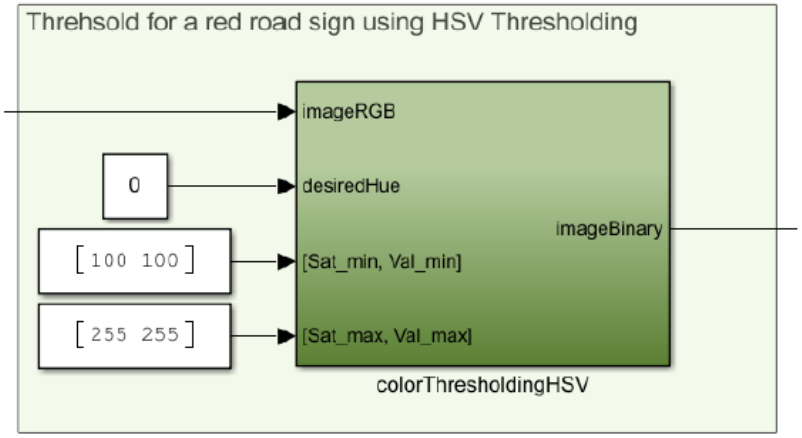


Figure 47: RGB to Binary HSV thresholding

Then, the block **Image Find Objects** from the QUARC Target simulink library founds the center of the object detected.
This location of the center of the sign will be used after to know the distance of the stop sign thanks to the depth sensor of the RGBD Camera.

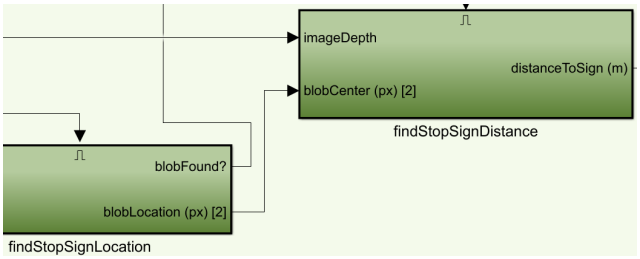


Figure 48: Find the distance of the Found Stop Sign

You can see how the thresholding worked and
Your Task : Browse the different blocks shown above to understand the simulink model. Then place the stop sign about 50 centimeters in front of the Qcar and run it (refer to the subsection : How to run your model).
Here is what you must see :

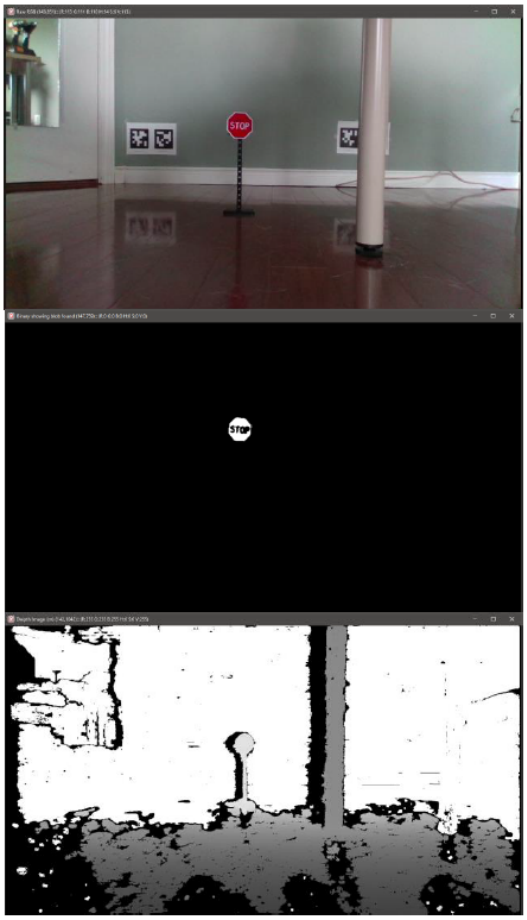


Figure 49: RGB image (top) — Thresholded image (middle) — Depth image (bottom)

You can see if the Thresholding did well, and how the Depth information is displayed, with darker colors depending on distance.

6.2 Stop the Car in front of a Stop Sign

In this section, we will make the QCar stop in front of a Stop Sign using the RGBDImaging simulink model seen before as a basis.
Open the File Stop.slx

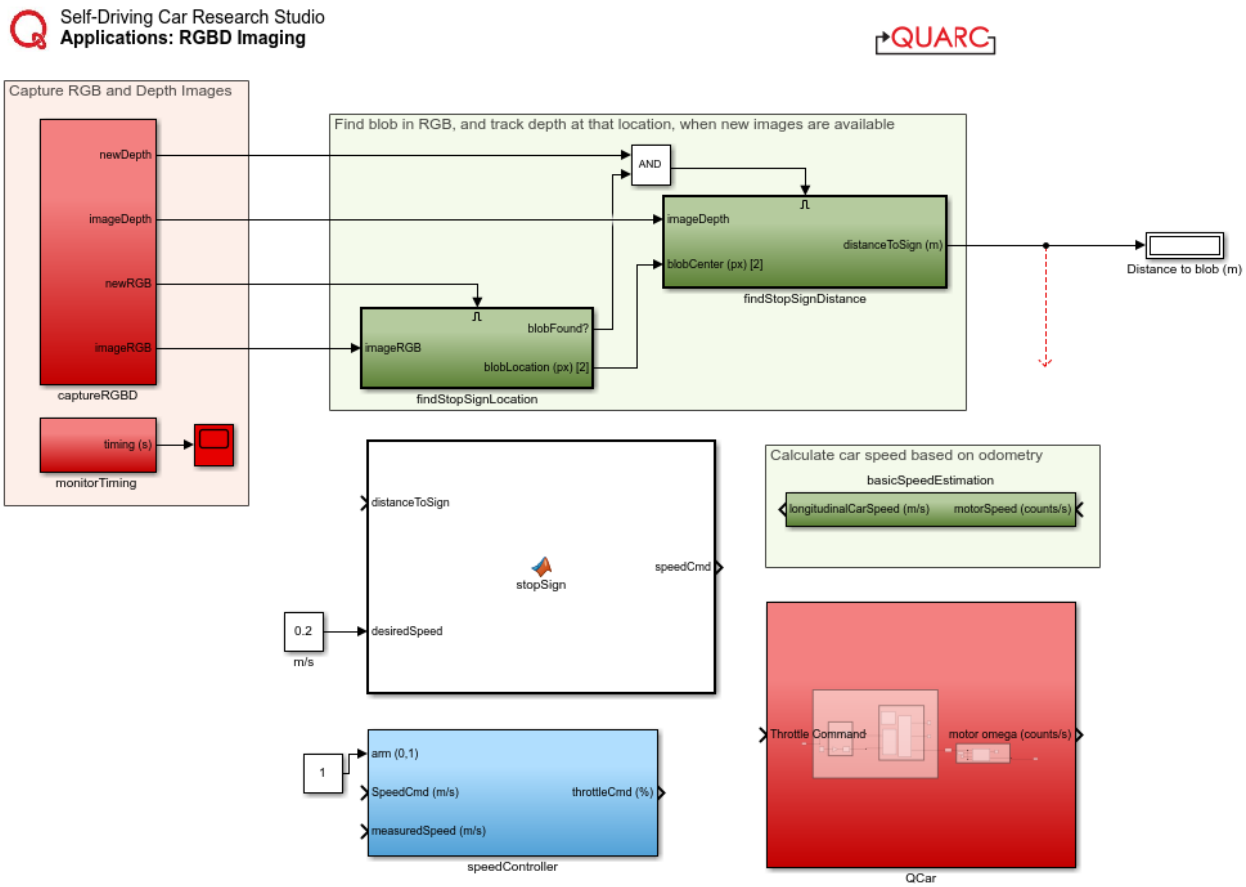


Figure 50: Stop Sign Simulink Model to Connect

Task 1 :
Open the MATLAB Function block called stopSign. Fill it to make the QCar stop if it is between 5 and 40cm. And make the speed output equal to the desiredSpeed input if the Stop Sign is not in that range.

Note : we are not looking for a stop sign between 0 and 40cm but 5 and 40cm because the sensor can give a distance of 0 when it does not find the sign.

```
function speedCmd = stopSign(distanceToSign,desiredSpeed)

if  %if the stopSign is between 0.4 and 0.1 m distance
    %stop the Car

else
    %go to desired speed


end
```

Figure 51: Stop Sign Matlab Function (to Fill)

Task 2 : Connect all the blocks together.
Knowing :

- speedController is a PI controller that turn the speed into a throttle command in %
- basicSpeedEstimation block transform motorSpeed from the sensors from count/s to longitudinal car speed in m/s.
- Qcar block is using ThrottleCommand input to command the QCar in the HIL-Write block. Motor omega output is the count/s of the motor encoder read on HIL-Read block.

Task 3 : Place the QCar at least 1 meter ahead of a stop sign and test your program.

Task 4 : Suggest improvements

6.3 Vision Based Lane Following

Lane Tracking or Lane Following is one of the main aspects for self-driving cars. Vehicles have to both use perception and control aspects to perfectly detect the Lane to follow and to control the car behavior consequently. The input are image driven information while the command are still steering and speed.

In this section we will see how the QCar can detect and track a line using its RGBD and CSI cameras.

6.3.1 How it works

The file you will work on now is Autonomous_Driving.slx. It is so called because it allows both lane following and obstacle detection using the Qcar RGBD and CSI cameras. The process is shown below.

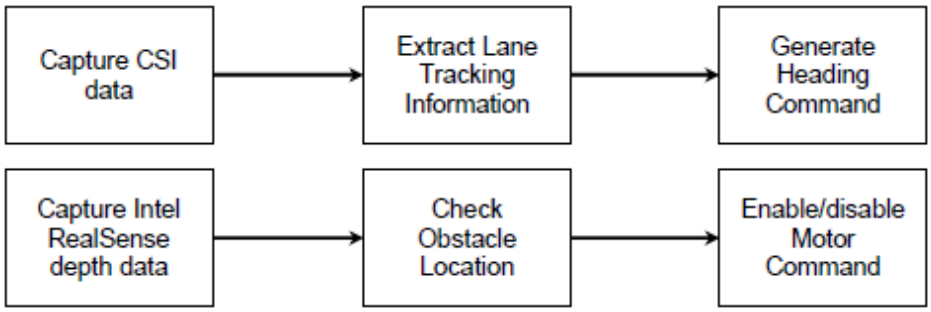


Figure 52: Autonomous Driving Diagram

The simulink implementation is displayed in the Figure below.

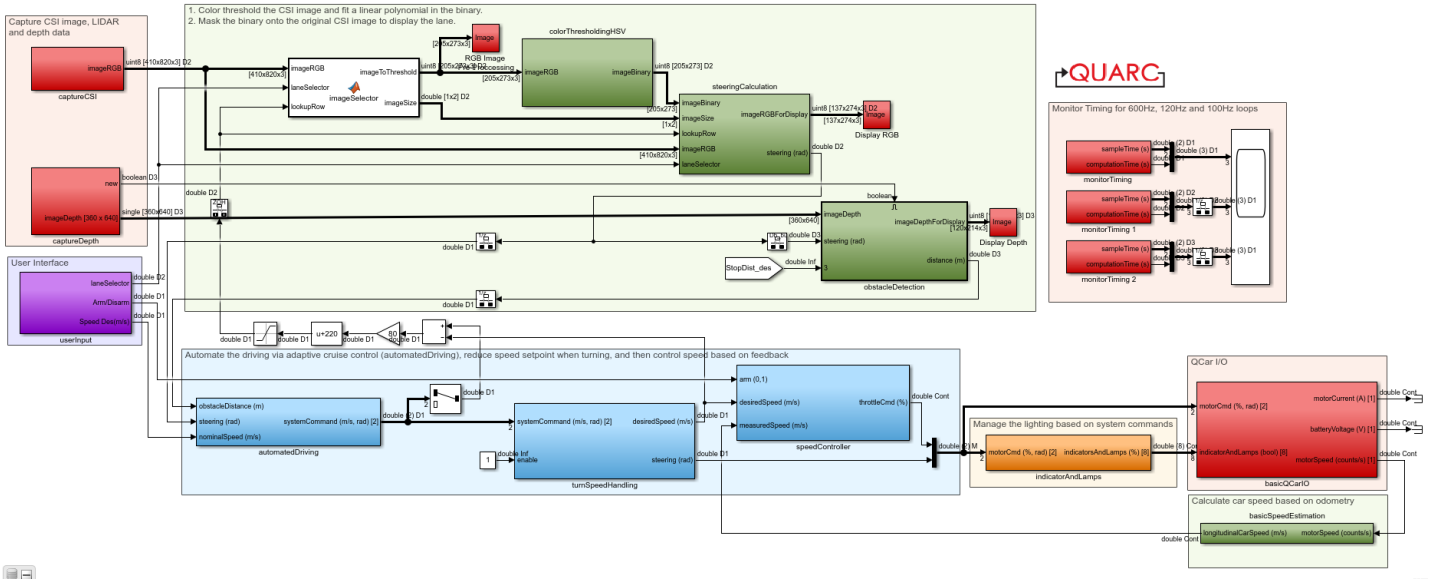


Figure 53: Autonomous Driving Simulink

- Green part is Lane Information Detection and Obstacle detection.
- Blue part automates the driving via 3 main blocks, **automatedDriving** which is an adaptative cruise control which reduces the speed command based on the distance to an obstacle, **turnSpeedHandling** which reduces speed setpoint when turning, and then **speedController** which controls speed based on feedback with a PI controller and a feedforward gain.
- Red parts are linked to the QCar as inputs or outputs.

6.3.2 Perception — HSV Tuning

The goal here will be to make the QCar detect the line so it can follow it and travel around the track
HSV is the abbreviation for Hue Saturation Value, which is a system of color management in computer science.
Hue : The hue is coded following the angle which correspond to it on the Color Circle seen in Figure 46

- 0° or 360° = red
- 60° = yellow
  ...
- 240° = blue

Saturation : The saturation is the intensity of the color, it varies between 0 and 100 percent. The lower the saturation is the more the image will be grayed out and will look bland.
Value : The value is the brightness of the color, it varies between 0 and 100 percent. The lower the saturation is the darker the image will be.

Here is an example of different HSV tunings for blue and yellow color :

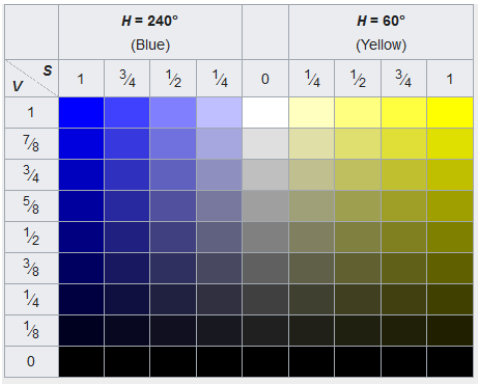


Figure 54: HSV Example Grid

In this section, the simulink block you will act on is the purple block called userInput.

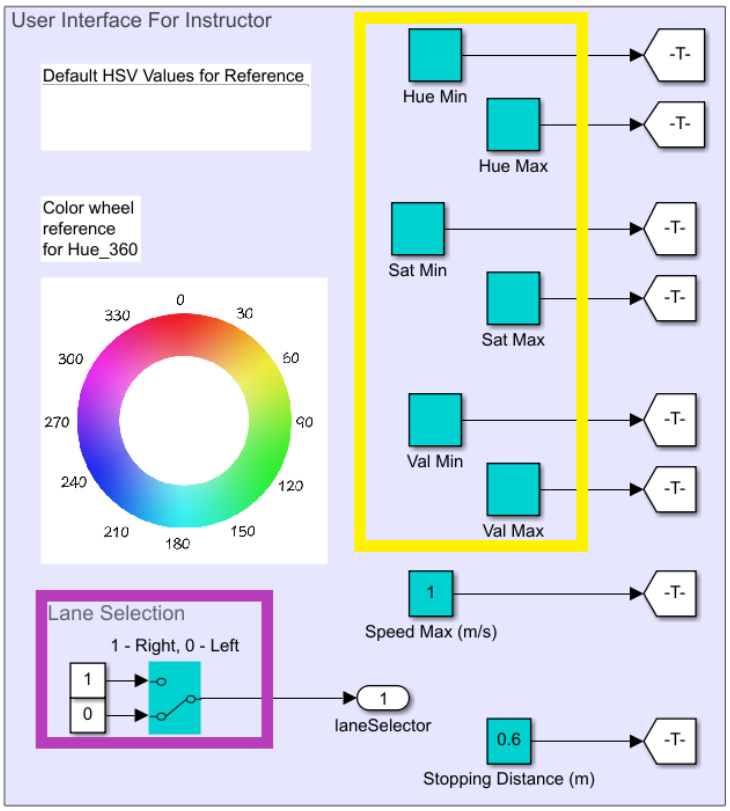


Figure 55: user interface

The purple box is the Lane Selection. This is where you can choose if the Qcar has to follow the line on its right or on its left. The yellow box is about Tuning the HSV value to get a proper HSV Thresholding to correctly detect the line

Task : In the yellow box, tune the parameters to find the middle yellow lane. The Saturation and Value Range are between 0 and 255, with 0 being 0% and 255 being 100% saturation or value.
To do so, place the Qcar on the track as shown below.

Figure 56: Qcar on the Track

Make sure the Stop/Start switch on the User Interface is on 0 position and run the simulink model.

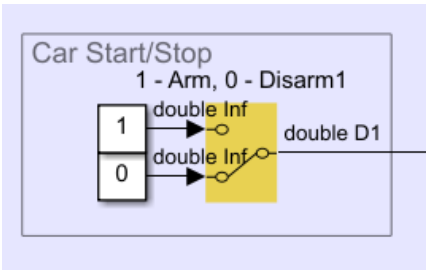


Figure 57: Start/Stop Button

Then tune the HSV parameters ranges until the yellow line is well detected by the RGBD Camera. If the middle yellow line is well detected with your HSV tuning, the line must be colored with red pixels on the RGBD Image Processed displayed window

Figure 58: Yellow Line Detected

Only the yellow line must be detected, no other objects of the room.

> Advice : The brightness could be different in different segments of the Track and the yellow line could be lost. To avoid this issue as much as possible, place the QCar at different spots on the track to find the perfect tuning.

When you found the appropriate HSV Tuning, click on the Car Start/Stop switch to let the QCar start his lap.

7 Obstacle Detection and avoidance

The Trajectory Following aspects seen before were not adaptive to the environment. In this section, we will see two ways to deal with the environment surrounding the QCar. First, we will see how to avoid obstacles using a first sensor, the LIDAR. Then we will see how to detect stop signs using RGBD Camera and make the car stop in front of them.

7.1 LIDAR Obstacle Detection

In this section, we will use the LIDAR to make the QCar avoid obstacles.

Every file needed in this section are placed in the Folder IV.LIDAR

The two programs covered by this section are as follows :

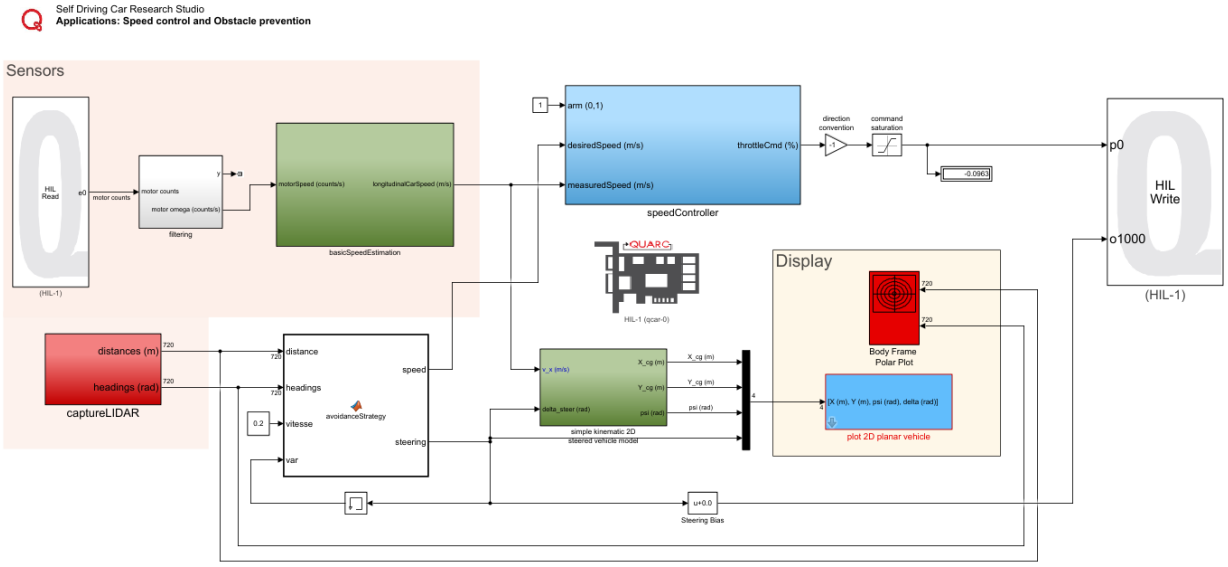


Figure 59: Simulink Obstacle Prevention

Both LIDAR_Stop.slx and LIDAR_Basic.slx are made the same way. The difference Lies in these MATLAB Function blocks :
In the first simulink model, the function stops the Qcar in front of an obstacle.
Both functions use as inputs the distance and headings data from the LIDAR and a desired speed input constant.
The first one acts on the speed only. Setting the speed to 0 when the Qcar is near an obstacle.
The second one acts on the steering only. It sets the steering angle up to 0.5 or -0.5 radians depending on the position of the detected obstacle. The speed remain untouched.
Task 1 : Open the File LIDAR_Stop.slx.

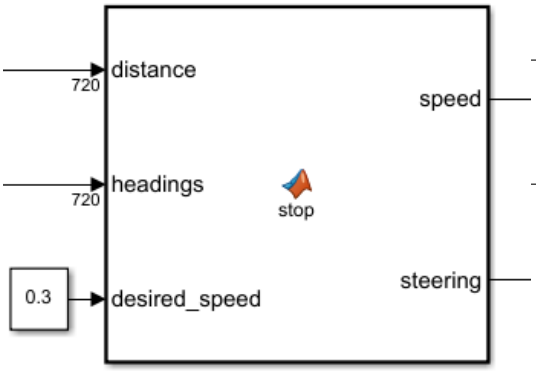


Figure 60: Stop strategy

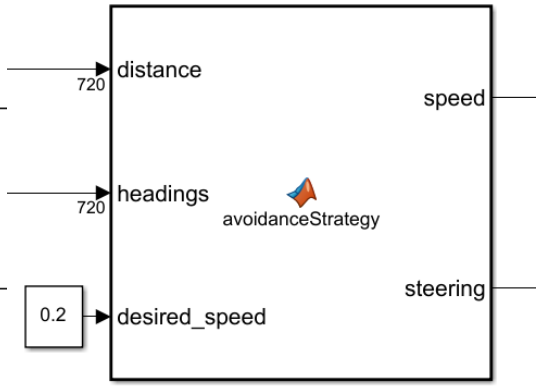


Figure 61: Steering strategy

Task 2 : Place the Qcar as displayed on the image below. Run it and observe the behavior of the Qcar.

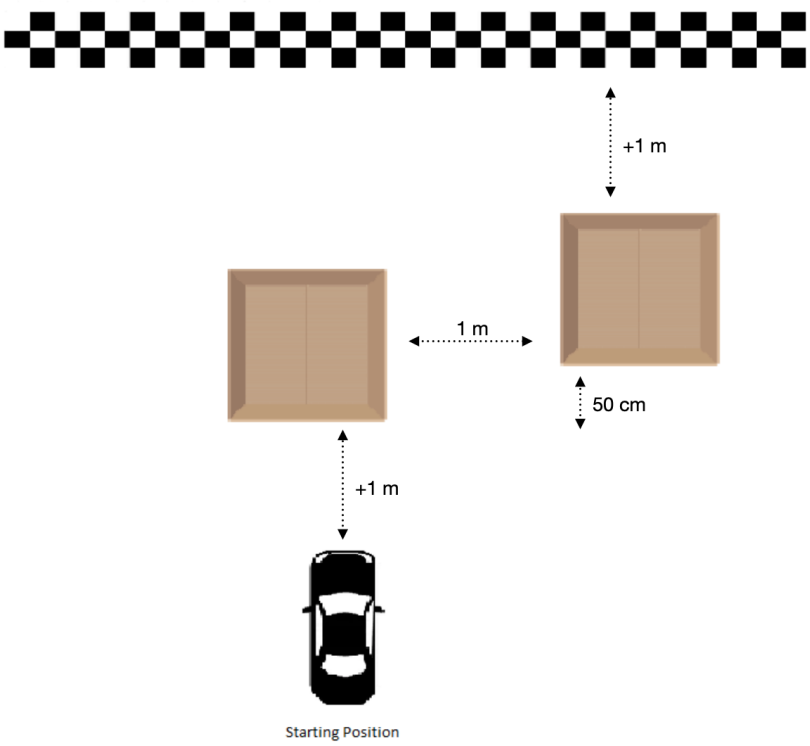


Figure 62:

Task 3: Do the same with the second file : LIDAR_Base.slx

Task 4 — Open Problem : Combine Speed and Steering strategies in the same simulink model. The Qcar should avoid the obstacles and if it is about to make a contact with an obstacle.

To avoid configuration issues. Copy the LIDAR_Base.slx program to work on it.

8 Troubleshooting

How to fix the issues that may appears :

- If the following error message appears : It was not possible to connect to the specified URI. Refer to "How to run your model" section. Your car may be switched off or you have entered the wrong IP address
- If the following error message appears : "The file could not be found" here is the method to follow : Make sure no special symbols or spaces are found in your repertory. Delete the NAMEOFTHEFILE Quarc Linux Nvidia executable and the Simulink Cache of that same NAMEOFTHEFILE
- If a Message Box pops up, saying the file is not added to the path : Click Add To Path
- If the following error message appears : "Error occurred while executing External Mode MEX-file 'quarc_comm': One of the arguments is invalid." : Make sure the Qcar ping with the computer and is not connected to another router.
- If the following error message appears : Error reported by S-function 'hil_initialize_block' in 'Stop/QCar/HIL Initialize': The "Active during normal simulation" feature is only licensed for QUARC Home use. Due to safety and liability concerns, this feature is not enabled, and will not be enabled, for full QUARC installations.
  Try again.

9 Appendix

Specifications :

Dimensions	39 x 21 x 21 cm
Weight (with batteries)	2.7 kg
Power	3S 11.1 V LiPo (3300 mAh) with XT60 connector
Operation time (approximate)	2 hr 11 m (stationary, with sensors feedback) 35 m (driving, with sensor feedback)
Onboard computer	NVIDIA® Jetson™ TX2 CPU: 2 GHz quad-core ARM Cortex-A57 64-bit + 2 GHz Dual-Core NVIDIA D GPU: 256-core NVIDIA Pascal™ GPU architecture , 1.3 TFLOPS (FP Memory: 8GB 128-bit LPDDR4 @ 1866 MHz, 59.7 GB/s
LIDAR	LIDAR with 2k-8k resolution, 10-15Hz scan rate, 12m range
Cameras	Intel D435 RGBD Camera 360° 2D CSI Cameras using 4x 160° FOV wide angle lenses, 21fps to 12
Encoders	720 count motor encoder pre-gearing with hardware digital tachomete
IMU	9 axis IMU sensor (gyro, accelerometer, magnetometer)
Safety features	Hardware “safe” shutdown button Auto-power off to protect batteries
Expandable IO	2x SPI 4x I2C 40x GPIO (digital) 4x USB 3.0 ports 1x USB 2.0 OTG port 3x Serial 4x Additional encoders with hardware digital tachometer 4x Unipolar analog input, 12 bit, 3.3V 2x CAN Bus 8x PWM (shared with GPIO)
Connectivity	WiFi 802.11a/b/g/n/ac 867Mbps with dual antennas 2x HDMI ports for dual monitor support 1x 10/100/1000 BASE-T Ethernet
Additional QCar features	Headlights, brake lights, turn signals, and reverse lights (with intensity co Dual microphones Speaker LCD diagnostic monitoring, battery voltage, and custom text suppor

Table 1: Specifications of the QCar

Fore more information:
`https://www.quanser.com/products/qcar/#overview`

References

[1] https://medium.com/dish/75-years-of-innovation-shakey-the-robot-385af2311ec8

[2] https://fr.mathworks.com/help/robotics/ug/pure-pursuit-controller.html

[3] https://docs.quanser.com/quarc/documentation/qcar.html

[4] https://www.quanser.com/products/self-driving-car-research-studio/