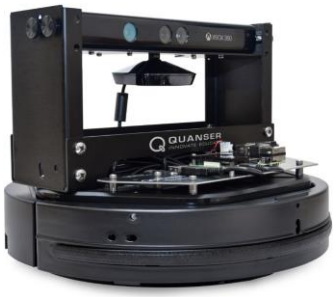
POLYTECH NANCY

Lab n°2

Vision guided control for a differential drive mobile robot



Hugues Garnier
Mayank Jha

January 2022

Contents

Acknowledgements

The contents of this lab has been adapted from an initial version provided by Quanser. This is fully acknowledged.

Special thanks go to Sarah HELLY for having provided some nice contributions to the contents.

Lab 1

Vision guided control for a differential drive mobile robot

Figure 1.1 depicts an overview scheme that summarizes the **see-think-act** cycle in mobile robot systems[1].

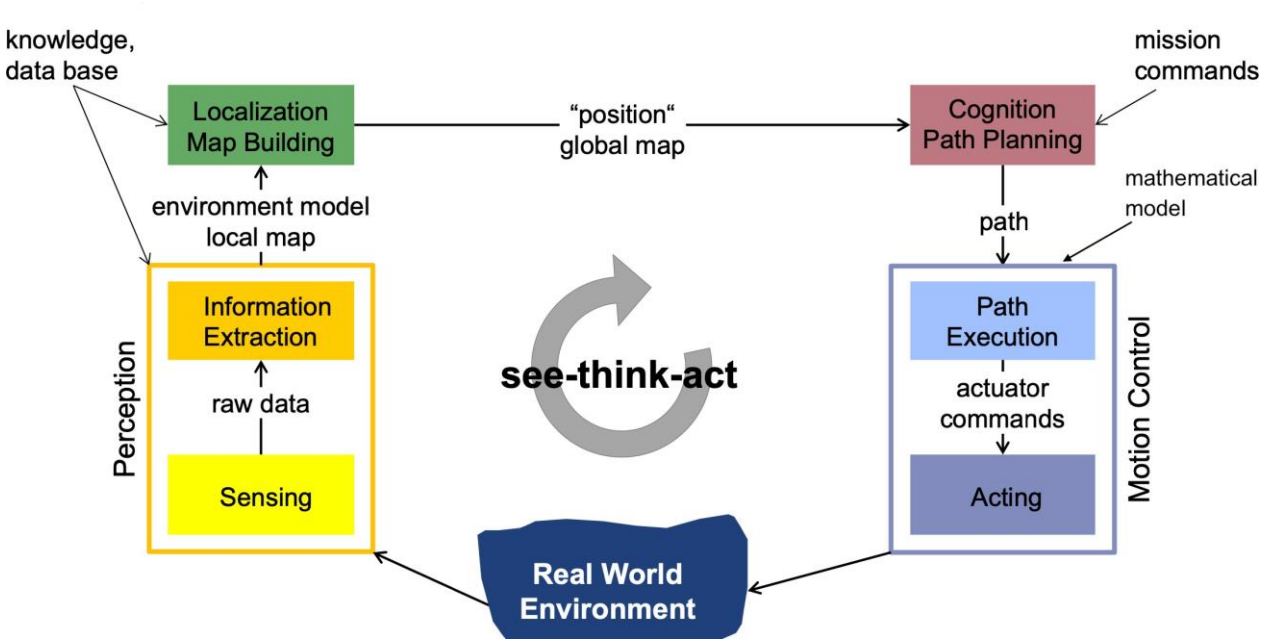


Figure 1.1: The **see-think-act** cycle in mobile robot systems (adapted from Siegwart et al., Autonomous Mobile Robots, ETH 2021[1])

This final goal of this Lab is to study and develop a vision guided control strategy for the Quanser two wheel differential drive QBot mobile robot.

[1] asl.ethz.ch/education/lectures/autonomous_mobile_robots/spring-2021.html

1.1 The Quanser QBot mobile platform

The Quanser QBot[2] is an open-architecture autonomous ground robot for indoor environment, built on a 2-wheel mobile platform.

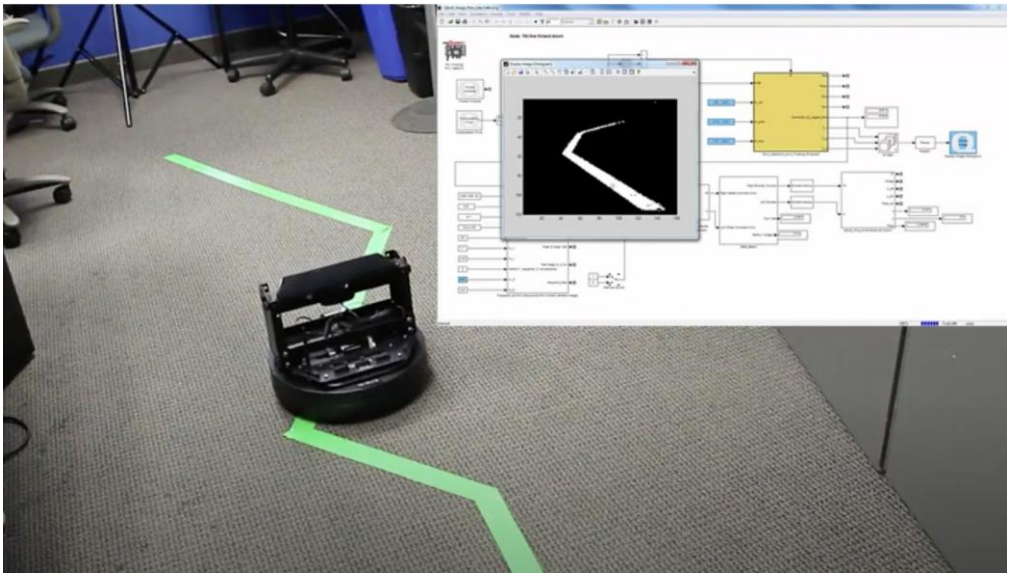


Figure 1.2: Vision guided control for the QBot mobile robot

Equipped with built-in sensors and a vision system, the QBot is ideally suited for studying standard and advanced control for mobile robots.

The purpose of this introductory section is to familiarize you with the basic concepts related to the QBot platform including the hardware (actuators and sensors) and the software components.

To start with, watch the first 10 mn of the video by two Quanser engineers presenting some of the capabilities of the QBot mobile robot:
www.youtube.com/watch?v=QwKh-45yQ9Y

Further information about the QBot platform can also be obtained from:
www.quanser.com/products/QBot-2e/

[2]We are actually using the Quanser **QBot 2e** but we will use its shorter name **QBot** in this Lab.

1.1.1 Main Hardware Components

As shown in Figure 1.3, the QBot robot consists of two left and right drive wheels mounted on a common axis. This drive configuration is known as differential drive. Two castor wheels at the front and back of the robot stabilize the platform without compromising movement. The two drive wheels are independently driven forward and backward in order to actuate the robot.

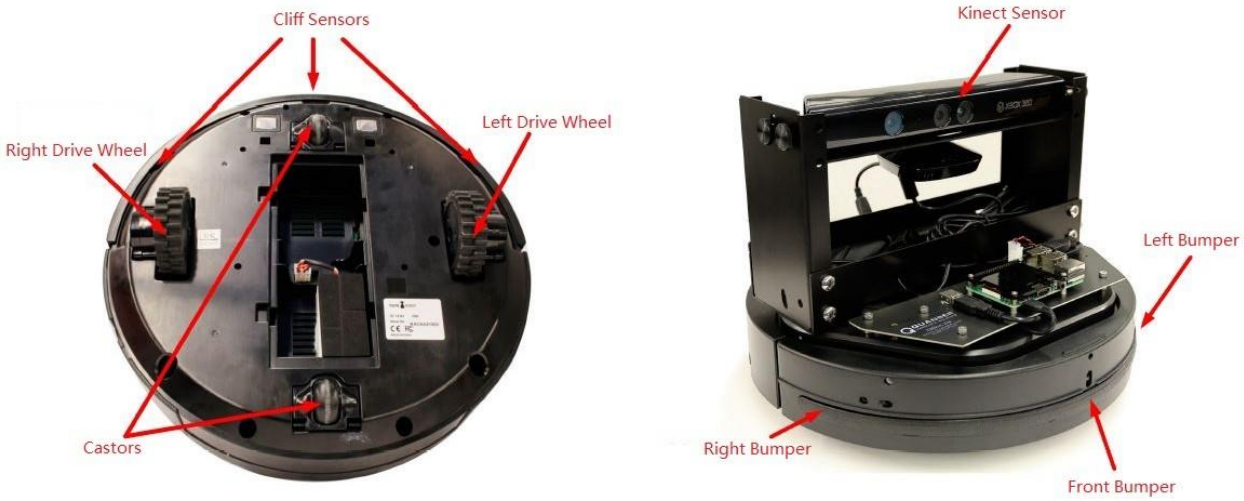


Figure 1.3: Bottom and top views of the QBot mobile robot

The motion of each wheel is measured using encoders, and the robot orientation, or yaw angle, are estimated using the integrated gyro. More information on the Kinematics, and how you can generate wheel commands to achieve specific motion trajectories will be given in the upcoming sections. You will also learn how the measured sensory information is used for odometric localization.

In addition to the encoders and gyro, the QBot comes with a Microsoft Kinect (fr.wikipedia.org/wiki/Kinect) for computer vision, that outputs color image frames (RGB) as well as depth information. You can process RGB and depth data for various purposes including visual inspection, 2D and 3D occupancy grid mapping, visual odometry, etc. More information on these concepts will be given in the Vision Guided Control Section.

As shown in Figure 1.3, the QBot robot also comes with integrated bump sensors (left, right and central), and cliff sensors (left, right, and central). These sensors can be used in a control algorithm to avoid obstacles or prevent damage to the robot. The battery fits underneath the QBot, and can last continuously for about 3 hours after a full charge. The robot can charge with a battery charger connected to the plug as shown in Figure 1.4.

Figure 1.4: Button on/off and plug for the battery charger

The QBot uses a Raspberry Pi 3Model B+ as a small-scale embedded computer with integrated WiFi that runs any compiled programs. The Raspberry Pi 3 Model B+ interfaces with the Kobuki mobile chassis as well as the Kinect sensor.

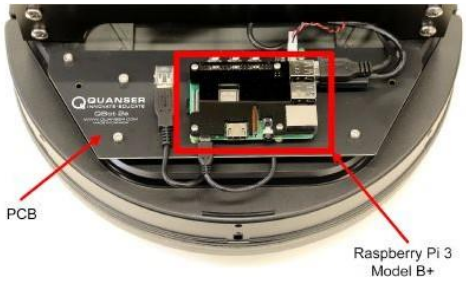


Figure 1.5: Raspberry Pi 3 Model B+ with integrated WiFi embedded on the QBot robot

1.1.2 Software and Communication

During this Lab, we will make use of QUAnser Rapid Control (**QUARC**) Prototyping software which seamlessly integrates with MATLAB/Simulink to provide real-time communication and interfacing to hardware platforms.
QUARC is a very efficient way to design, develop, deploy and validate real-time applications on a variety of hardware target systems using Simulink. QUARC will generate real-time code directly from Simulink and runs it in real-time on the QBot robot, all without digital signal processing or without writing a single line of code.

Further information and tutorial video introduction to QUARC can also be obtained from:
www.quanser.com/support/tutorials/?fwp_tutorial_categories=quarc

Controllers will be developed in Simulink with QUARC on the host computer, and these Simulink models will be cross-compiled and downloaded to the target (Raspberry Pi 3 Model B+) seamlessly by using the wireless router. A diagram of this configuration is shown in Figure 1.6.

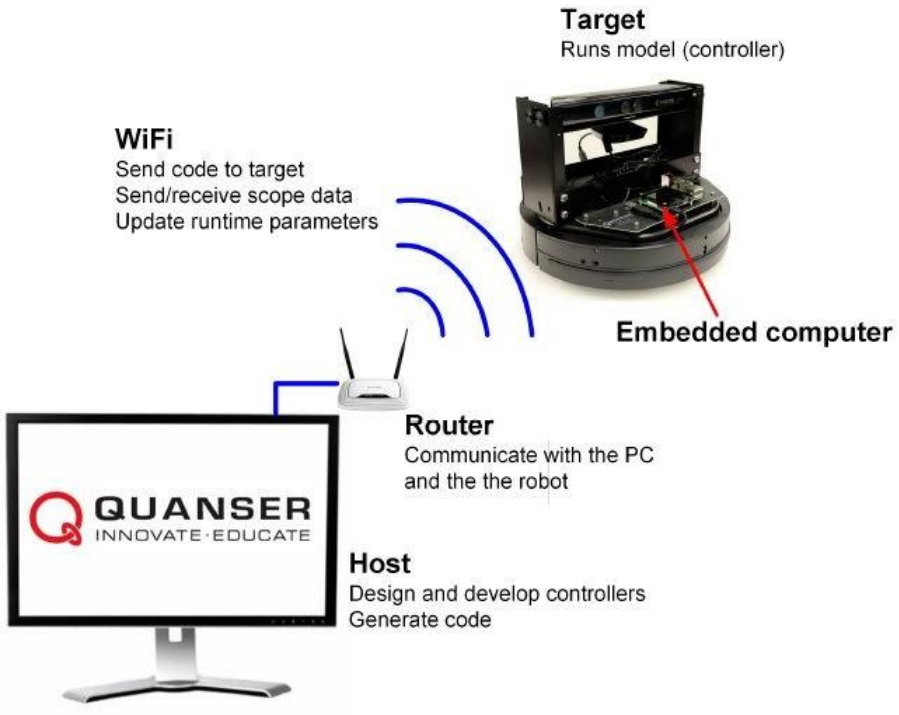


Figure 1.6: Communication between a computer with QUARC installed and the QBot

To communicate with the QBot, several QUARC blocks available in the Quanser Library in Simulink will be used. The following blocks will be, in particular, most useful :

1. **Hardware In the Loop (HIL) Initialize block:** it configures the drivers and hardware interface for the QBot.
2. **HIL Read block:** it is used to read sensory data.
3. **HIL Write block:** it is used to drive the two motors.
4. **Host Initialize:** it can be used for external input devices such as a keyboard (Host Keyboard) or joystick (Host Game Controller).
5. **Kinect Initialize block:** it is used to initialize the Kinect sensor. Maximum frame-rate and resolution are set in this block.
6. **Kinect Get Image block:** it captures RGB data from the Kinect sensor.
7. **Kinect Get Depth block:** it captures depth data from the Kinect sensor.
8. **Video Compressed Display block:** it transmits compressed input data (RGB or depth) from the QBot to the PC and displays them on the monitor.

All the aforementioned blocks will be seen in the Simulink files supplied for this lab. The complete list of available QUARC blocks available in the Quanser Library is available on the following site:
docs.quanser.com/quarc/documentation/quarc_block_categories.html

1.2 Getting started

The purpose of this section is to get you started with the QBot mobile robot.

1.2.1 Matlab 2020a and supplied Simulink files

To get started, follow the set of instructions below:

1. Open Matlab 2020a and NOT any other version.
2. From the Public folder on the local disk C:, go to the folder `TP_5A_SIA`.
   the files required for this lab are stored here.
3. Once inside the folder, create a new folder named with the initials of your 2 (or 3) names: `Name_Initials`.
4. Copy all the files and folders included in the folder `TP_5A_SIA` in your folder.
5. From the Matlab workspace, set your working directory to be: `TP_5A_SIA/Name_Initials`.
   From now on, this will be your "working-directory".

In the `User_guide` sub-folder, you will find the following two pdf files
- Quick Start Guide.pdf
- User Manual.pdf
Please refer to these documents any time.

Before launching any Simulink file, it is recommended to follow the steps below:

1. **Setting up the QBot;**
2. **Setting up and verifying the network connections;**
3. **Configuring QUARC for the QBot target.**

1.2.2 Setting up the QBot

Turn on the robot by using the on/off button at the rear of the robot.

1.2.3 Setting up and Verifying the Network Connections

Follow the steps to setup and verify the network connection:

1. Power up and turn on the wireless router by using the button at the rear of the router.
2. Wait for about 1 mn for the wireless network to establish and allow the QBot to automatically connect to the wireless router.
3. Use the Command Prompt ("invite de commandes") to verify the connection between the router and the computer. Ping the router address: `ping 192.168.2.1`.
4. Ping then the robot address to verify the connection between the robot and the wireless router. The robot address is indicated on the QBot robot near the Raspberry Pi.

It is important to verify the network connection anytime during the Lab. Repeat the tests above in case your Simulink program does not work anymore. If the robot does not communicate with the computer, see the Troubleshooting Section at the end of this document.

1.2.4 Configuration of QUARC: How to Run a Simulink Model

The steps to configure QUARC are as follows:

1. Open **Matlab** and open the **Simulink** file entitled **QBot2e_Explore_Sensors.mdl**.

2. Click on **QUARC** (as shown in Figure 1.7), then select **Options**.

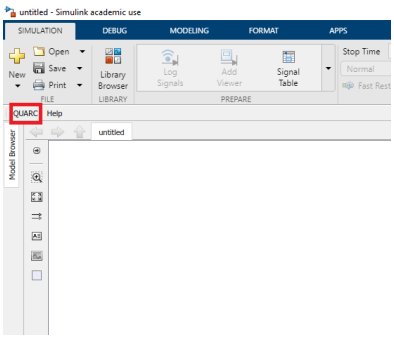


Figure 1.7: Button QUARC in Simulink

3. To setup the default target address for all linux-raspberry-pi-3 targets, go to the **QUARC menu** select **Options** then **Code Generation**. **The system target type** parameter should be set to "quarc_linux_pi_3.tlc", as shown in Figure 1.8.

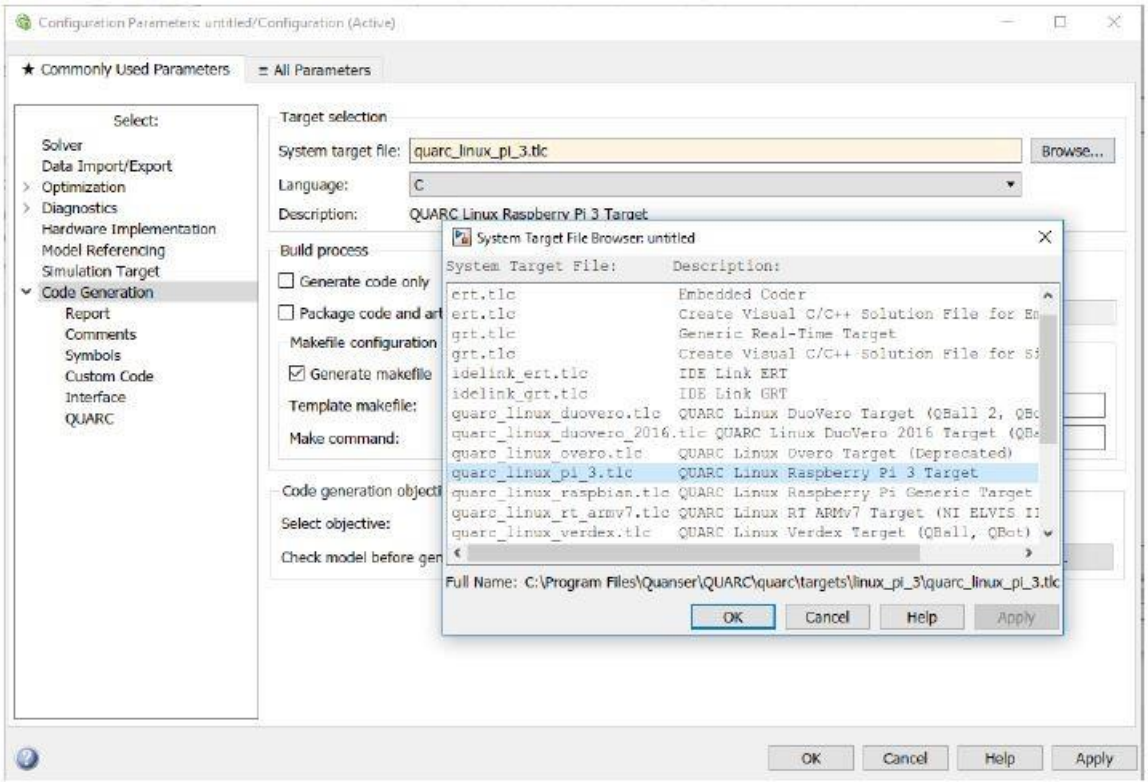


Figure 1.8: Setup for the code generation

4. Choose **Code Generation > Interface** on the left hand panel. Setup the **MEX-file arguments** as shown in Figure 1.9. The argument should be: **'-w -d /tmp -uri %u','tcpip://192.168.2.XX:17001'**. Replace the XX by the appropriate number for your robot.

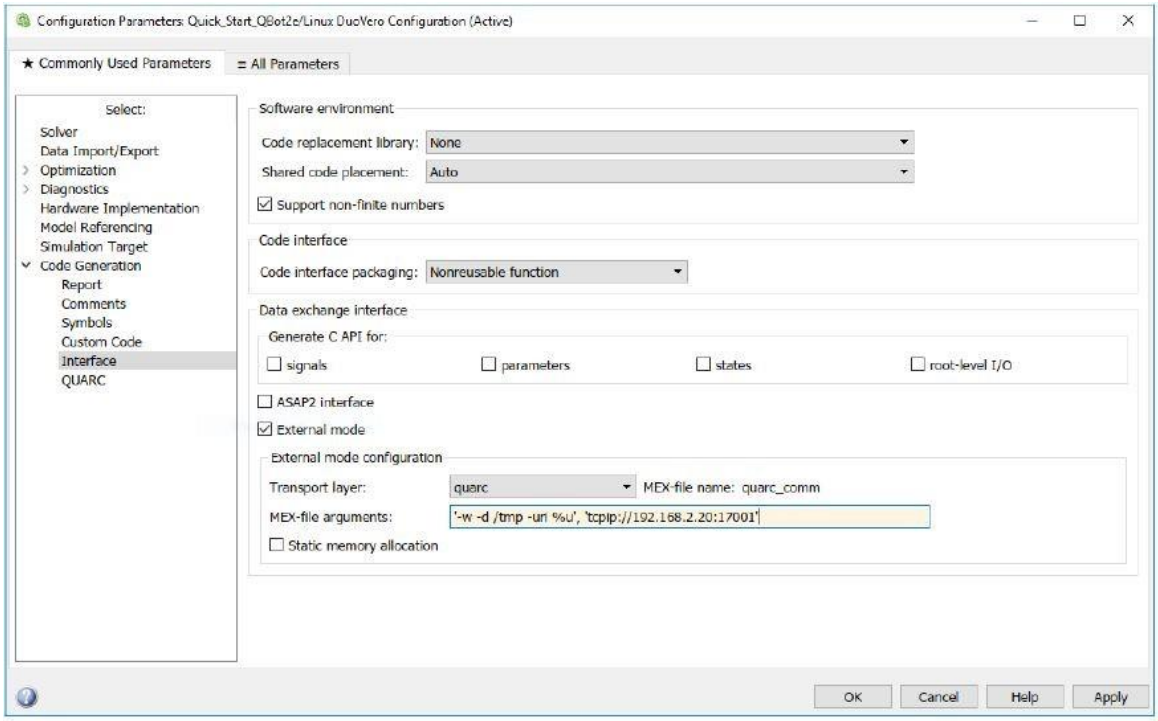


Figure 1.9: Setup of the Interface option for code generation in QUARC

Alternatively, go to **QUARC menu** then **Preference** replace the Default Model URI with the IP address of the desired target robot: **"tcpip://192.168.2.XX:17001?nagle=no,keep_alive=1"**.

⚠ **If some blocks deliver an error message; in particular the Host Initialize block, try the following :**
go to line "Source of URI", indicate "Specify via dialog (do not evaluate)"; then go to line "URI Upon which to listen" and set "**tcpip://192.168.2.XX:18001**".

Once the configuration of QUARC is complete, a Simulink model may be built, downloaded and then start/stop on the Qbot target by following the instructions below

Starting with MATLAB R2019b, Simulink has greatly simplified building, downloading, connecting, starting, stopping, and disconnecting of real-time code with external mode operation.

To build, download, connect, and start the model for real-time external operation, click the **Monitor & Tune** button on the *HARDWARE* tab.

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.

To stop the real-time code execution, click the **Stop** button on the *HARDWARE* tab and Simlink will stop the real-time code on the target and disconnects from the target.

Figure 1.10: Procedure to build/stop a Simulink model on the Qbot target

Further information about the procedure to build a model is available on the Quanser site: docs.quanser.com/quarc/documentation/quarc_procedures.html#external_mode_operation

1.2.5 Basic Input/Output Tests

Explore the Chassis Sensors

The Simulink model for these experiments is **QBot2e_Explore_Sensors.mdl**, shown in Figure 1.11.

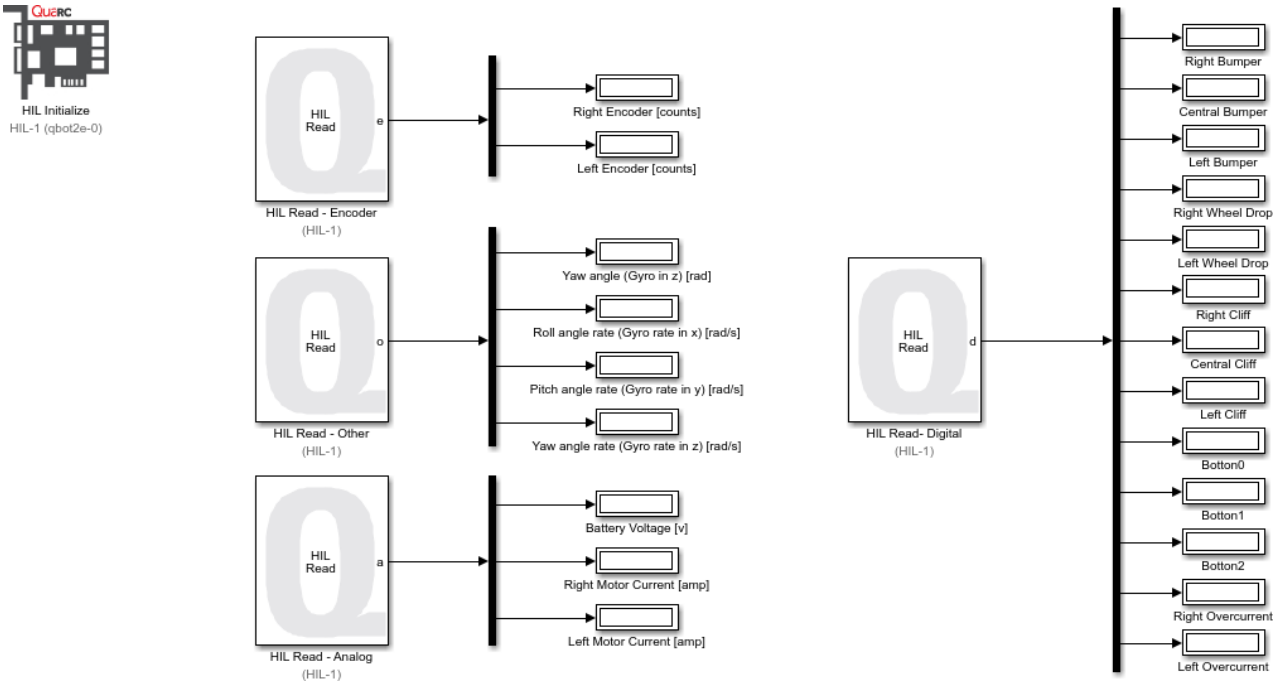


Figure 1.11: Snapshot of the model **QBot2e_Explore_Sensors.mdl**

The QBot is equipped with several binary digital sensors including impact bumpers, wheel drop sensors, cliff sensors, and buttons. These sensors can be used for various custom applications including obstacle avoidance and path following. The digital sensors are measured using the QUARC HIL Read block, and are accessed on digital channels 28-38. The sensor

mapping is outlined in the User Manual, and is shown in Figure 1.11.

The chassis also includes over-current sensors to ensure that the motors are not damaged due to improper use. These sensors can be measured on channels 39 and 40.
To track the status of the QBot battery, and for possible closed-loop motor control several power sensors are provided. The battery voltage can be measured using the QUARC HIL Read on Analog channel 0. The two motor current measurements can also be measured on Analog channels 4 and 5.

Task

Compile and run the **QBot2e_Explore_Sensors.mdl** file. Once the QBot beeps, indicating that the initialization routine is complete, trigger several of the sensors to familiarize yourself with their operation. Close the Simulink file.

Explore the LED Commands

The Simulink model that will now be used is called **QBot2e_Explore_Commands.mdl**, shown in Figure 1.12.

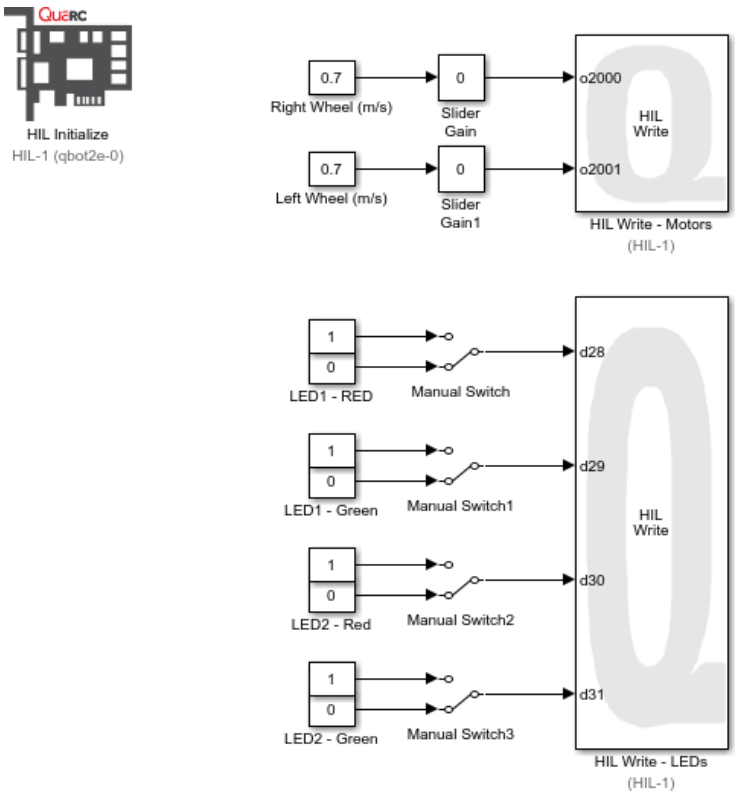


Figure 1.12: Snapshot of the model **QBot2e_Explore_Commands**

As shown in Figure 1.13, there are two programmable LEDs on the top of the QBot, that can be illuminated as either green or red. The LEDs are commanded on digital lines 28 through 31 as outlined in the QBot User Manual, and as demonstrated in the model shown in Figure 1.12.

Figure 1.13: QBot LEDs

Task

Compile and run the **QBot2e_Explore_Commands.mdl** file. Once the QBot beeps, indicating that the initialization routine is complete, toggle the switches that control the commands sent to each LED to familiarize yourself with their operation. Do not modify the slider gains for the left and right drive wheels; this will be explored in the next sub-section.

1.2.6 Explore the Actuators: DC Motors

The QBot mobile robot is driven by two high-performance DC motors with encoders, located coaxially in a differential drive configuration. The two motors are commanded using the QUARC HIL Write block shown in Figure 1.12. The right wheel command is sent on channel 2000, and the left wheel command on channel 2001. The maximum command that is recommended to each motor is 0.7 m/s.

Task

Compile and run the **QBot2e_Explore_Commands.mdl** file. Once the QBot beeps, follow the procedures outlined below to familiarize yourself with the motor operation.

1. Place the Qbot in the middle of the room.

2. Use the slider to gradually increase the left wheel velocity command to 0.5 m/s. The robot should begin to rotate about the left wheel.

3. Slowly decrease the right wheel velocity to -0.5 m/s, with the left wheel command set to 0.5 m/s. The QBot should begin to spin in place.

4. Slowly increase the right wheel velocity to 0.2 m/s, the robot should now begin driving in a small circle.

For more information on mapping the rotational speed of the wheels to deterministic motion of the chassis, please refer to the Kinematics models and locomotion Section. Close the Simulink file.

1.2.7 Explore the Sensors: Encoders

The Simulink model for the 3 upcoming sub-sections is **QBot2e_Explore_Sensors.mdl**, shown in Figure 1.11. Open it.

The QBot is equipped with two high resolution wheel encoders to track the rotation of each wheel. The wheel encoder values for the right and left wheels are read using the QUARC HIL Read block on encoders channels 0 and 1 respectively, as shown in Figure 1.11.

Compile and run the **QBot2e_Explore_Sensors.mdl** file. Once the QBot beeps, follow the procedures outlined below to familiarize yourself with the encoders.

1. Slowly move the robot forward and backward and track the direction of each encoder. Then rotate the robot clockwise and counter-clockwise.

2. The encoders on the QBot measure 52 counts per rotation at the motor, which when translated through the gearbox yields 2578 counts per rotation at the wheel. To convert the wheel rotation counts to wheel rotations in radian add a gain of 0.0024 to each measurement. You can also add a gain of 35 representing the wheel radius in mm to translate the rotation of the wheels to linear movement of the robot base in each wheel frame. Try adding these gains to check the rotation of each wheel against the movement of the robot base.

1.2.8 Explore the Sensors: Gyroscope

The QBot is equipped with a three axis gyroscope that tracks the angular rate of the roll, pitch, and yaw of the robot. For simplicity, the yaw angle of the robot is output on channel 1002 using the QUARC HIL Read block, along with the roll, pitch, and yaw rates on channels 3000-3002. The proper configuration of these measurements is shown in Figure 1.11.

Compile and run the **QBot2e_Explore_Sensors.mdl** file. Once the QBot beeps, follow the procedures outlined below to familiarize yourself with the gyroscope.

1. Slowly rotate the robot clockwise, and counterclockwise and observe the changing gyroscopic values.

2. Pitch and roll the robot and observe the measured angular rates. If needed, take note of the conventions which all follow a conventional right-hand rule.

1.2.9 Explore the Sensor: Microsoft Kinect

To enable the QBot to easily perform classic mobile robotic algorithms and applications including visual servoing, mapping and localization, obstacle avoidance, and path following, the QBot is equipped with a Microsoft Kinect which utilizes an infrared laser projector and monochrome CMOS sensor for depth measurement, and an RGB camera for image processing. The camera provides image and depth capture at a frame rate of up to 30 fps and resolution of 640 x 480 pixels. The depth sensor has a range of 0.5 to 6 m. The sensor has a horizontal field of view (FOV) of 57 degrees, and vertical FOV of 43 degrees. The sensor can also be pivoted vertically by up to 21.5 degrees to allow the sensor to measure objects outside of the conventional horizontal field. The depth of objects are measured as a monochrome value depending on their distance from the sensor as illustrated in Figure 1.14.

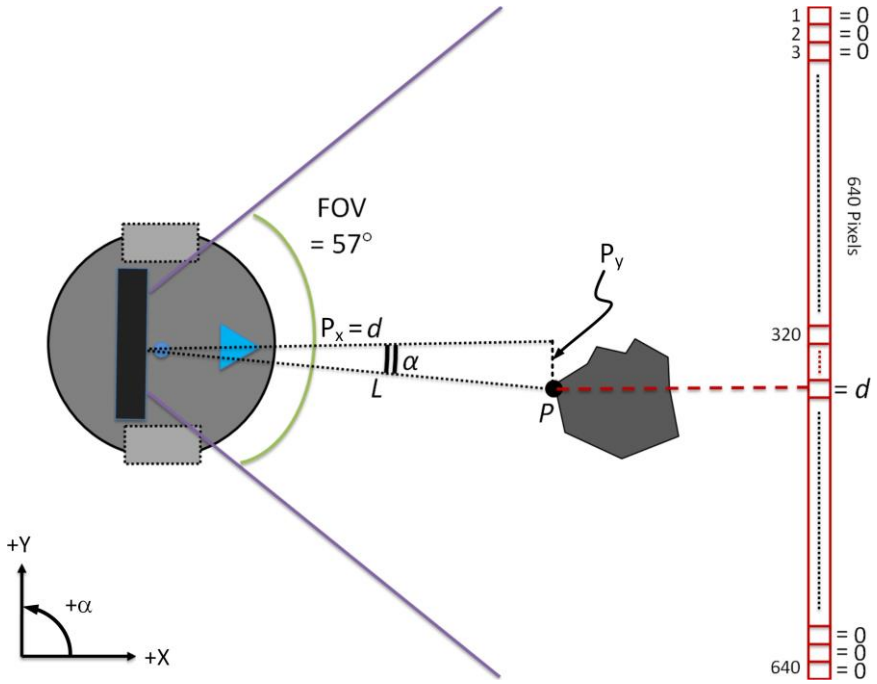


Figure 1.14: QBot Depth Measurement

The Simulink model that is used to illustrate Kinect sensor measurement is **QBot2e_Explore_Vision.mdl**, shown in Figure 1.15.

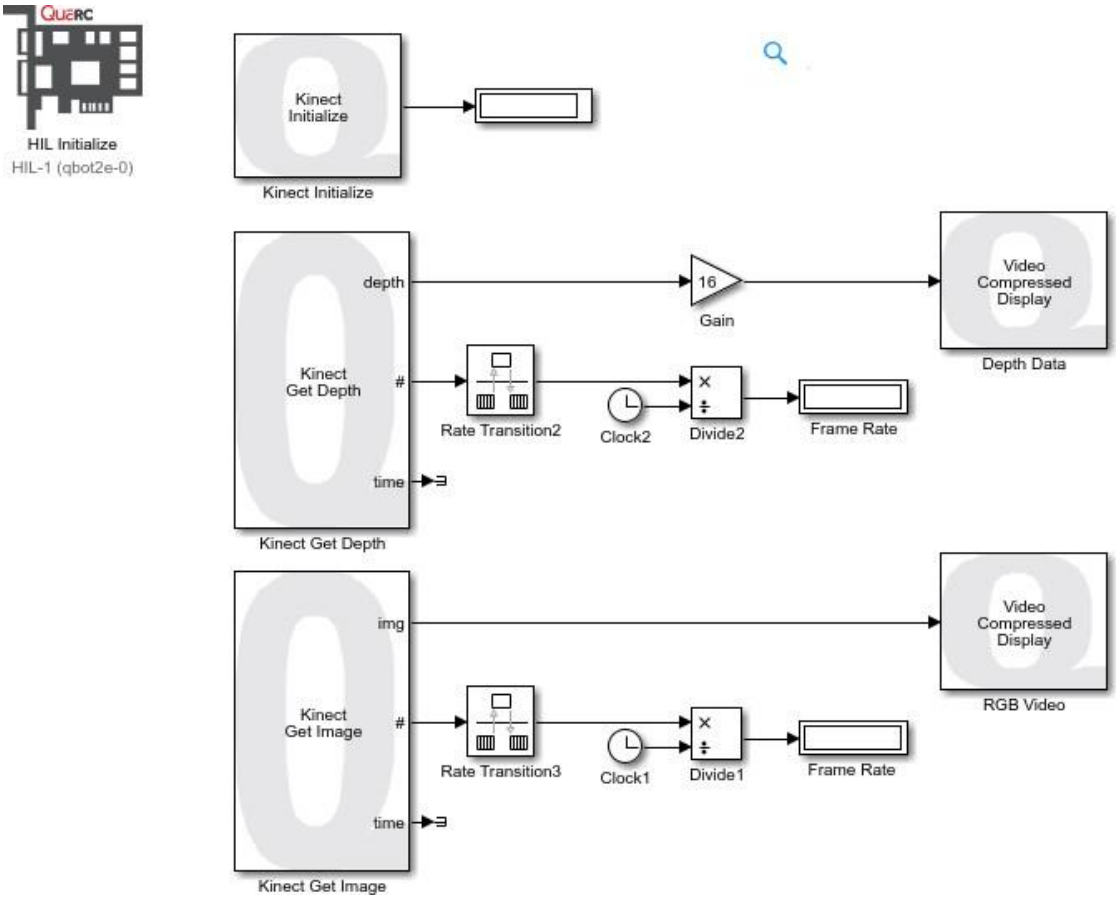


Figure 1.15: Snapshot of the model **QBot2e_Explore_Vision.mdl**

Before running the Simulink model, there are several configuration settings that should be observed to ensure proper deterministic operation of the sensor. Begin by making sure that the trigger duration in the Signal and Triggering sub-menu of the External Mode Control Panel is set to **2**, as shown in Figure 1.16.

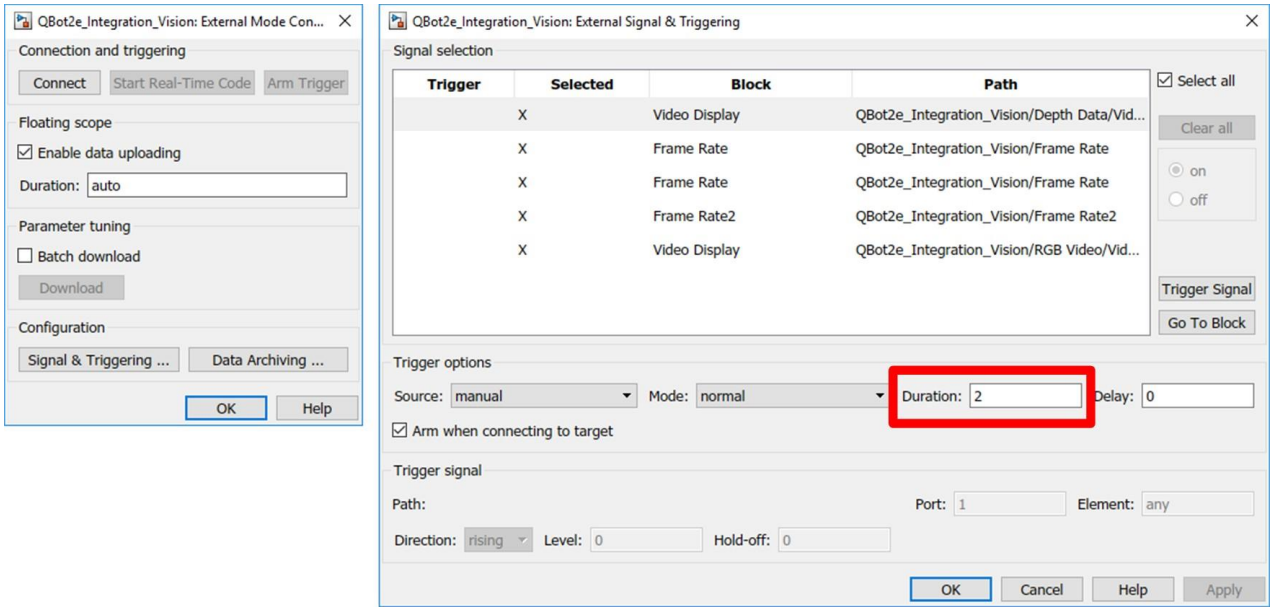


Figure 1.16: External mode triggering duration settings

To achieve a desired frame rate, use the Sample Time parameter in the Kinect Initialize, Kinect Get Depth, and Kinect Get Image blocks. The parameter is specified in seconds. By default, the Sample Time parameter is set to **qc_get_step_size * ceil(0.033 / qc_get_step_size)**, where it attempts to acquire images at the maximum rate of 30 fps. As an example, assuming the sample time of the model is qc_get_step_size = 0.005 s, to acquire images at a say 10 fps (100 milliseconds per frame), the Sample Time parameter should be set to **qc_get_step_size*20** = 100 milliseconds per frame.

For improved image playback it is recommended to use the Video Compressed Display as shown in 1.15. The Video Compressed Display block displays video in a window on the host. However, it uses image compression internally to minimize the bandwidth required to transmit the raw image from the target model to the host. It is designed for typical video frame rates. The video can be paused and resumed, and the current frame may be saved to disk as an image. This block has much higher performance than the Display Image block because it does not use MATLAB figure windows. However, as a result it cannot be used to drive axes in a MATLAB GUI.

Compile and run the **QBot2e_Explore_Vision.mdl** file. Once the QBot beeps, indicating that the initialization routine is complete, make sure that the Depth Data and RGB Video windows are visible. If they are not open, double click on each of the display blocks to open the viewers. With the model running, move the robot and place objects in front of the robot to familiarize yourself with the operation and capabilities of the vision sensor.

1.3 Keyboard Manual Drive

1.3.1 Basic Wheel Drive Mode

In this experiment, you will command the left and right wheels independently and observe the motion of the robot and vision data from the Kinect sensor.

The Simulink file is **QBot2e_Keyboard_Basic_Wheel_Drive.mdl**.

In this file, we use, as seen in the blue frame, the HIL Initialize block to configure the interface options for the QBot, the 3D video Interface for the Kinect sensor, the Host Keyboard for the keyboard interface.

If you double click on the QBot Basic Motor Commands and Sensor Measurement sub-system block, and then **QBot2e_IO_Basic**, you will find **HIL_Write** and **HIL_Read** blocks used to drive the motors and read from the sensors. Take time to explore the model. You can right-click on the blocks and select help to find more useful information regarding each block.

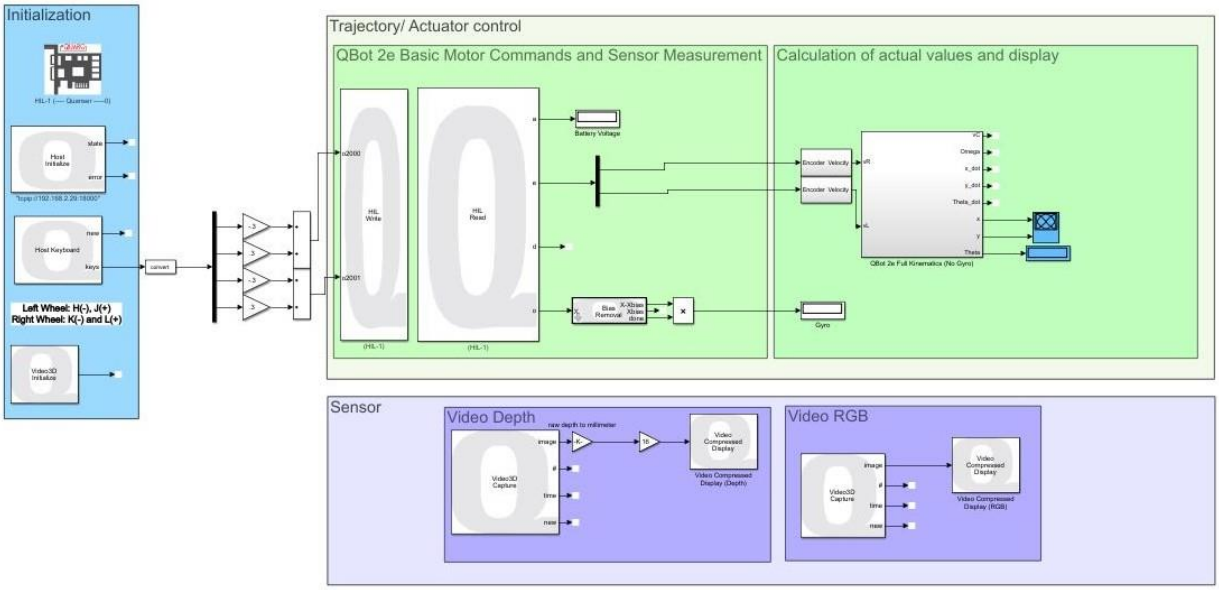


Figure 1.17: Snapshot of the Simulink *QBot2e_Keyboard_Basic_Wheel_Drive.mdl* file

The *KinectGetDepth* and *KinectGetImage* blocks are used to read the depth and RGB data from the Kinect sensors. We then use the *V ideoCompressedDisplay* blocks to display these images.
The keyboard controls for this experiment are as follows:

- **Left wheel rotational speed command**: H (-) and J (+)
- **Right wheel rotational speed command**: K (-) and L (+)

Tasks

1. Compile the Simulink file

2. Double-click on the XY Figure block in blue on the right, as well as on the *VideoCompressedDisplay*(*Depth*) and *VideoCompressedDisplay*(*RGB*) blocks.
3. Use the keyboard to command the robot.
4. Describe how the speed of the left/right wheels relate to the actual motion of the robot (forward/backward motion, left/right turn, etc.).
5. Stop the file.

1.3.2 Normal Drive Mode

In this experiment, you will drive the robot in a conventional manner where the robot commands correspond to move forward/backward, and turn left/right, similar to the way you would control any vehicle.

The Simulink file for this experiment, as shown in Figure 1.18, is **QBot2e_Keyboard_Normal_Drive.mdl**.

The QUARC blocks used in this file are similar to the ones described above.

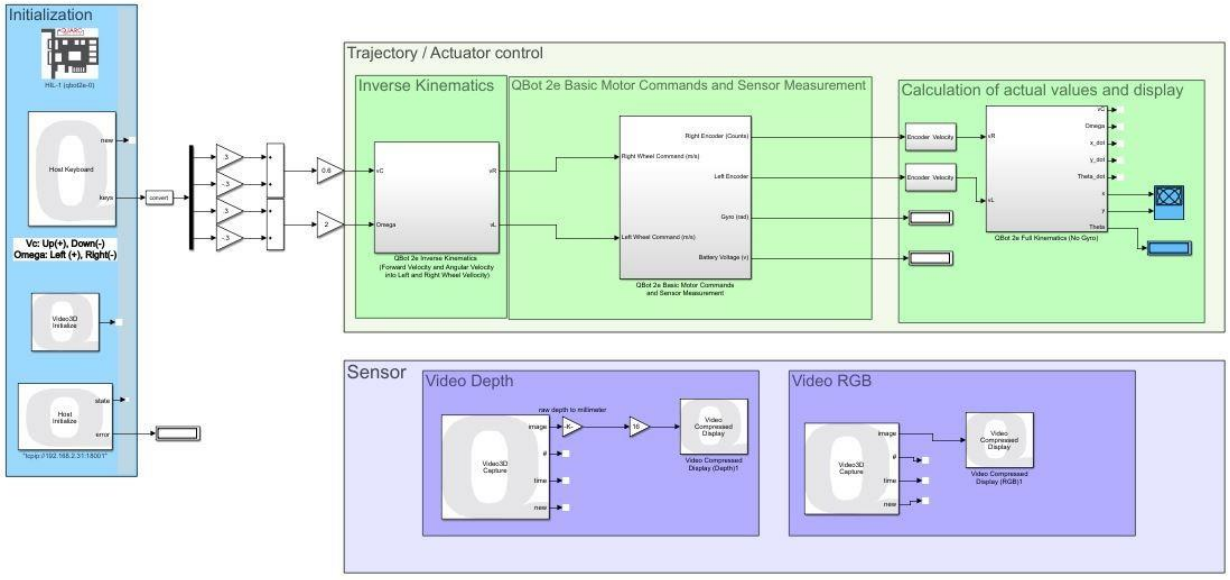


Figure 1.18: Snapshot of the Simulink *QBot2e_Keyboard_Normal_Drive.mdl* file

Two customized blocks are also used that apply the inverse kinematics and forward kinematics models of the device, the details of which are presented in the next Section. Keyboard controls for this experiment are as follows:

- **Linear velocity command**: Up arrow (+) and Down arrow (-)
- **Angular velocity command**: Left arrow (+) and Right arrow (-)

Tasks

After turning on the robot and connecting to the **Quanser_UVS-5G** wifi network, follow the steps below to run the Simulink file:

1. Open the Simulink file, compile it and run it.
2. Double-click on the **XY Figure**, **Video Compressed Display (Depth)** and **Video Compressed Display (RGB)** blocks.
3. Use the keyboard keys to command the robot. Up arrow to move forward, down arrow to move backward, left arrow to turn left, and the right arrow key to turn right.
4. Observe the RGB and depth images, as well as the XY figure, and report your observations. RGB image shows the sequence of images captured from the Kinect sensor with the requested capture rate. Depth data shows the distance of the objects in the field-of-view (FOV) of the Kinect sensor and can be visualized with an image.
5. Describe the benefits of controlling the vehicle in normal mode.
6. Put your QBot on the yellow line at the indicated place and make a lap while timing yourself.
7. Stop the Simulink file.

1.4 Kinematic Models and Locomotion

The purpose of this experiment is to study the basic motion behaviour of the Quanser QBot mobile platform.
Kinematics is used to determine the appropriate actuator commands for the robot. You can get a full and clear explanation about the derivation of the kinematic model for the differential drive and car-like model by watching the following video:
www.youtube.com/watch?v=LrsTBWf6Wsc

1.4.1 Differential Drive Kinematics

Differential drive kinematics is the mathematical relationship that maps the independent motion of the wheels to the overall movement of the robot chassis. This fundamental topic is the foundation of all mobile robot control, in that it is chiefly responsible for the predictable mobility of the robot. In this experiment you will investigate the differential drive kinematics of the Quanser QBot mobile platform.

The QBot is driven by a set of two coaxial wheels. These wheels are actuated using high-performance DC motors with encoders and drop sensors. To determine the relationship between the independent motion of the two wheels and the motion of the overall robot, we begin by modeling the motion of the robot about a common point.

Let the radius of the wheels be denoted by r, and the wheel rotational speed be denoted by w_L and w_R for the left and right wheel respectively. The linear speed of the two wheels along the ground is then given by the following equations:

$$v_L = w_L r \tag{1.1}$$

$$v_R = w_R r \tag{1.2}$$

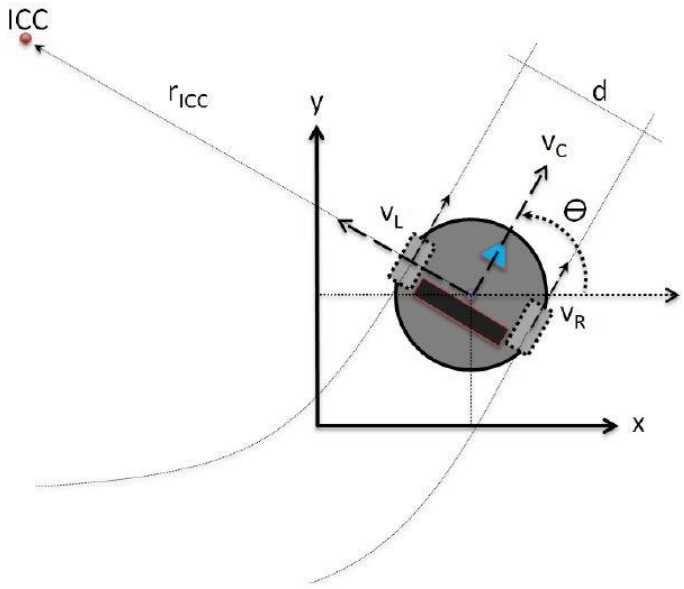


Figure 1.19: QBot Reference frame definitions

Assuming there is no wheel slippage, the QBot can move along the horizontal plane in straight or curved trajectory, as well as spin on a spot, by varying the relative speed between the left and right wheels.

Since we are assuming that the wheels are not subject to slip, the motion of the wheels are constrained to move along their forward and backward directions. This, together with the inherent constraint that is imposed by the robot chassis coupling the two wheels together, means that all robot chassis rotations must be about a point that lies along the common wheel axis. For example, if only one of the two wheels rotates, the robot would rotate (pivot) about the non-moving wheel. On the other hand, if both wheels rotate at the same speed, the robot rotates about a point infinitely far from the robot. This center of rotation is known as the Instantaneous Center of Curvature (ICC) as shown in Figure 1.19.

1.4.1.1 Background

Let r_{icc} be the distance measured from the center of the robot chassis, which is halfway between the left and right wheels, to the ICC. If d is the distance between the left and right wheels, ϑ is the heading angle of the robot, and v_C is the (forward/backward) speed of the robot chassis center, the motion of the QBot chassis can be summarized by the following equations:

$$v_C = \dot{\vartheta} r_{icc} \tag{1.3}$$

$$v_L = \dot{\vartheta} \left(r_{icc} - \frac{d}{2} \right) \tag{1.4}$$

$$v_R = \dot{\vartheta} \left(r_{icc} + \frac{d}{2} \right) \tag{1.5}$$

Notice that v_C, v_L and v_R are all defined along the same axis, which lies in the forward/backward direction of the chassis. Given the wheel speed, v_L and v_R, the robot speed, v_C, the angular rate, $w_C = \dot{\vartheta}$, and the distance from ICC, r_{icc} ,we can relate the motion of the wheels to the motion of the robot using the forward kinematic model for the differential drive system, as shown in Figure 1.20.

$$v_C = \frac{v_R + v_L}{2} \tag{1.6}$$

$$w_C = \dot{\vartheta} = \frac{v_R - v_L}{d} \tag{1.7}$$

$$r_{icc} = \frac{d}{2} \frac{v_R + v_L}{v_R - v_L} \tag{1.8}$$

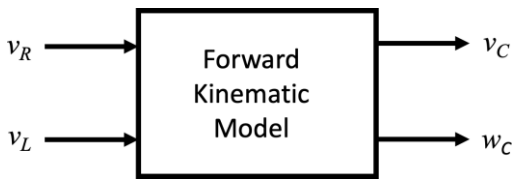


Figure 1.20: Block diagram of the forward kinematic model

Given the robot speed, v_C and the angular rate, $w_C = \dot{\vartheta}$, we can compute the wheel speed, v_L and v_R, of the robot using the inverse kinematic model for the differential drive system, as

shown in Figure 1.21.

$$v_R = v_C + \frac{1}{2} d\, w_C \tag{1.9}$$

$$v_L = v_C - \frac{1}{2} d\, w_C \tag{1.10}$$

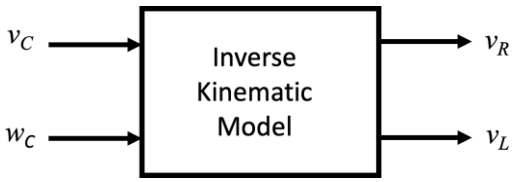


Figure 1.21: Block diagram of the inverse kinematic model

Tasks

The Simulink file for this experiment is **QBot2e_Diff_Drive_Kinematics.mdl** a snapshot of which is shown in Figure 1.22.

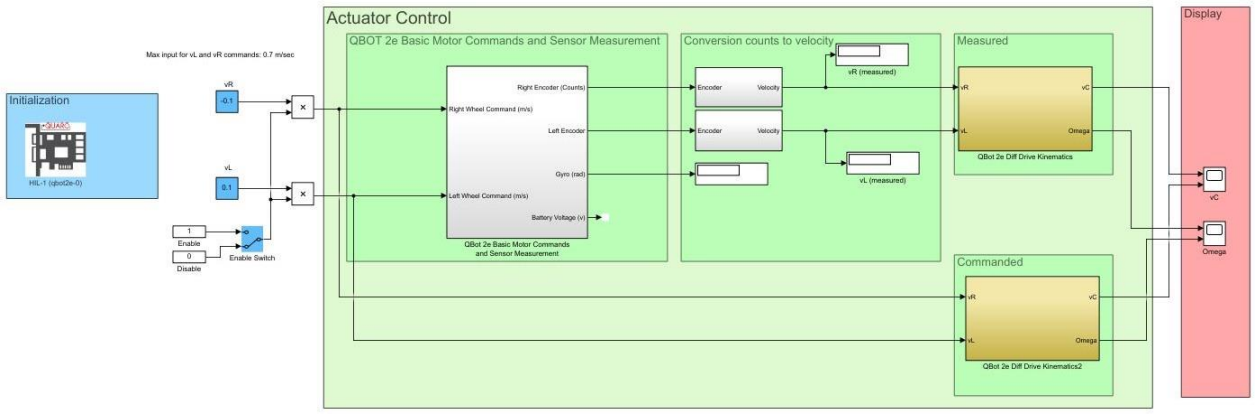


Figure 1.22: Snapshot of the Simulink **QBot2e_Diff_Drive_Kinematics.mdl** file

The QBot diff drive kinematics block, highlighted in yellow, receives the left and right wheel velocities as input and computes the forward/backward speed and angular velocity of the robot chassis center.

1. By using equations (1) to (5), derive (6), (7) and (8).

2. Compile and run the Simulink file.

3. Wait 5 seconds until the QBot has fully initialized, and then enable the movement of the robot using the manual enable switch shown in blue in Figure 1.22.

4. Set both left and right wheel velocity set points, highlighted in blue, to 0.1 m/s. Run the file and observe the linear and angular velocities calculated by using (5) to (7). What are the values of r_{icc} and w_C when the left and right wheels are moving at the same speed (i.e. $v_L = v_R$)? What do your results indicate about the relationship between r_{icc} and w_C?

5. Change the right wheel velocity to - 0.1 m/s and keep the left wheel velocity command at 0.1 m/s. What is the value of r_{icc} when the left and right wheels are moving at the same speed but in the opposite direction (i.e. $v_L = -v_R$)? What do these results indicate? Does the relationship identified earlier hold?

6. What does it mean when r_{icc} is negative?

7. What does it mean when w_C is negative?

1.4.2 Forward and Inverse Kinematic Models

The objective of this experiment is to investigate the forward and inverse kinematic model of the Quanser QBot mobile platform. Forward kinematics is used to determine the linear and angular velocity of the robot in the world coordinate frame given the robot wheel speeds. Inverse kinematics on the other hand, is used to determine the wheel commands needed for the robot to follow a specific path at a specific speed. Inverse kinematics is an essential tool for mobile robotics as it bridges the gap between a navigation and path planning module, and actual robot locomotion.

A typical forward kinematic model that computes the robot chassis speeds, v_C, and turning rate, w_C, from the wheel speed, v_R and v_L, with wheel separation distance, d, for a robot with differential drive system like the QBot is given by (6) and (7).

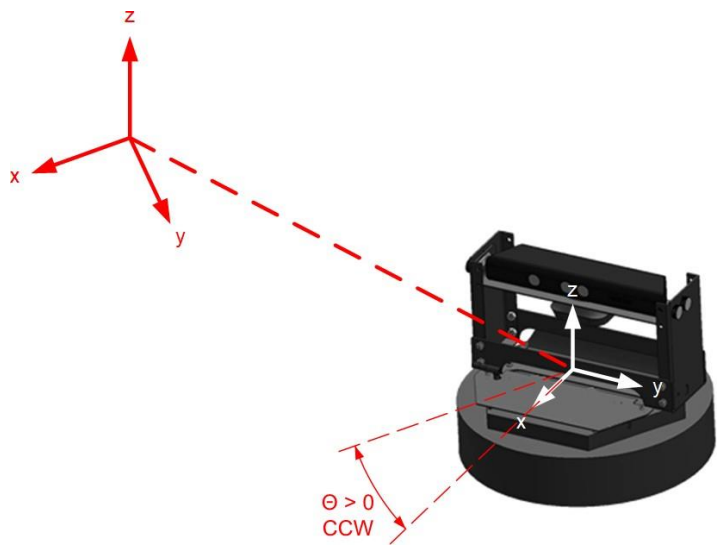


Figure 1.23: Robot chassis local and global reference frames for a robot with a heading ϑ

Implicit in the derivation of the above kinematic model is the use of a local frame of reference. In other words, the chassis speed, v_C, is expressed in the forward/backward (heading) direction of the robot chassis and not the global frame that would be used in a map of the environment. Since the robot chassis heading changes when the angular rate is non-zero, $w_C = \dot{\vartheta}$, we need to apply a transformation to the differential drive kinematics model in order to compute the robot chassis motion with respect to the global reference frame. For a robot with a heading, ϑ, the transformation required is the following rotation matrix:

$$R = \begin{bmatrix} \cos\vartheta & -\sin\vartheta & 0 \\ \sin\vartheta & \cos\vartheta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1.11}$$

This transformation maps motion expressed with respect to the robot chassis local frame to the corresponding motion in the global frame.

The corresponding inverse mapping is given as follows:

$$R^{-1} = \begin{bmatrix} \cos\vartheta & \sin\vartheta & 0 \\ -\sin\vartheta & \cos\vartheta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1.12}$$

1.4.2.1 Forward Kinematic Model

We define a state vector, S, as the position, x and y, and the heading, ϑ, of the robot chassis. Its definition and rate of change are given as follows:

$$S = \begin{bmatrix} x \\ y \\ \vartheta \end{bmatrix} \tag{1.13}$$

$$\dot{S} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\vartheta} \end{bmatrix} \tag{1.14}$$

The x and y axes lie in the "ground" plane that the robot primarily travels in. The heading, ϑ, is measured about the vertical z axis, which is defined as positive pointing upwards. The heading is zero, ($\vartheta = 0$), when the robot chassis forward direction aligns with the global x axis. The time-derivative of the states can be expressed in terms of the robot chassis speed, v_C, and angular rate, w_C, as follows:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\vartheta} \end{bmatrix} = R \begin{bmatrix} v_C \\ 0 \\ w_C \end{bmatrix} = \begin{bmatrix} v_C \cos\vartheta \\ v_C \sin\vartheta \\ w_C \end{bmatrix} = \begin{bmatrix} \frac{1}{2}(v_R + v_L)\cos\vartheta \\ \frac{1}{2}(v_R + v_L)\sin\vartheta \\ \frac{1}{d}(v_R - v_L) \end{bmatrix} \tag{1.15}$$

Equation (1.15) represents the forward kinematic model for the QBot that computes the linear speed, ($\dot{x}$ and $\dot{y}$), and turning rate, (w_C), of the robot chassis given its heading (ϑ), and wheel speed (v_R and v_L).

Similarly, the position of the Instantaneous Center of Curvature (ICC) in space, (x_{ICC} and y_{ICC}), expressed with respect to the global reference frame can be obtained as follows:

$$\begin{bmatrix} x_{ICC} \\ y_{ICC} \\ 0 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 0 \end{bmatrix} + R \begin{bmatrix} r_{ICC} \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} x - r_{ICC}\sin\vartheta \\ y + r_{ICC}\cos\vartheta \\ 0 \end{bmatrix} = \begin{bmatrix} x - \frac{d(v_R+v_L)}{2(v_R-v_L)}\sin\vartheta \\ y + \frac{d(v_R+v_L)}{2(v_R-v_L)}\cos\vartheta \\ 0 \end{bmatrix} \tag{1.16}$$

Equation (1.15) can be useful for path planning or obstacle avoidance algorithms.

Tasks

The Simulink file for this experiment is called **QBot2e_Forward_Kinematics.mdl**; a snapshot of which is shown in Figure 1.24.

The forward kinematic blocks, highlighted in yellow, receive the left and right wheel velocities as inputs and compute the linear velocities in the world coordinate frame. These blocks are used on both commanded wheel velocities giving us the ideal robot speed, as well as on the measured wheel velocities, resulting in the actual measured robot velocity.

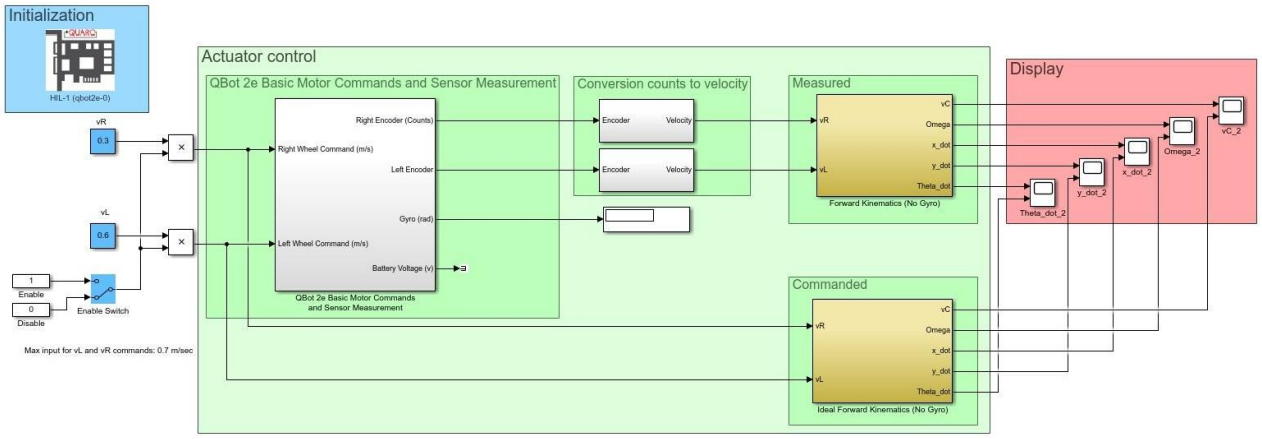


Figure 1.24: Snapshot of the Simulink file **QBot2e_Forward_Kinematics.mdl**

1. Compile and run the Simulink model
2. Wait until the QBot has fully initialized (you should hear the QBot startup chime), then enable the movement of the robot using the manual switch highlighted in blue.
3. Set the left and right wheel velocity commands, highlighted in blue, to 0.6 m/s and 0.3 m/s accordingly. Run the model and observe the ideal and measured linear and angular velocities in the world coordinate frame ($\dot{x}$, $\dot{y}$ and $\dot{\vartheta}$).
4. What is the shape of the robot trajectory when the right wheel is commanded to travel at twice the speed of the left wheel (i.e. $v_R = 2v_L$)? Comment on the effect of changing the value of v_L on the robot chassis trajectory.
5. Compute the required constant wheel speeds v_R and v_L to generate a trajectory with a constant turning rate of $w_C = 0.1$ rad/s and a constant turning radius of $r_{ICC} = 1$m. Implement the wheel speed command on the robot chassis, observe and explain the resulting chassis trajectory. Compare the desired turning rate and radius to the measured turning rate and radius.

1.4.2.2 Inverse kinematic model

If you want the robot to follow a certain path or drive at a given speed, you need to send appropriate left and right wheel commands to the robot. The inverse kinematics model computes the required wheel speed to obtain a desired robot chassis speed v_C, and angular rate ϑ. It is obtained by solving (6) and (7) together for the wheel speed v_R and v_L and is given as follows:

$$\begin{bmatrix} v_R \\ v_L \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \\ v_C - \frac{1}{2} d w_C \end{bmatrix} \quad c$$

Tasks

The Simulink file for this experiment is called **QBot2e_Inverse_Kinematics.mdl**; a snapshot of which is shown in Figure 1.25.

In this file, the Inverse Kinematics block for the QBot is shown in yellow and the input commands for v_C and w_C are highlighted in blue.

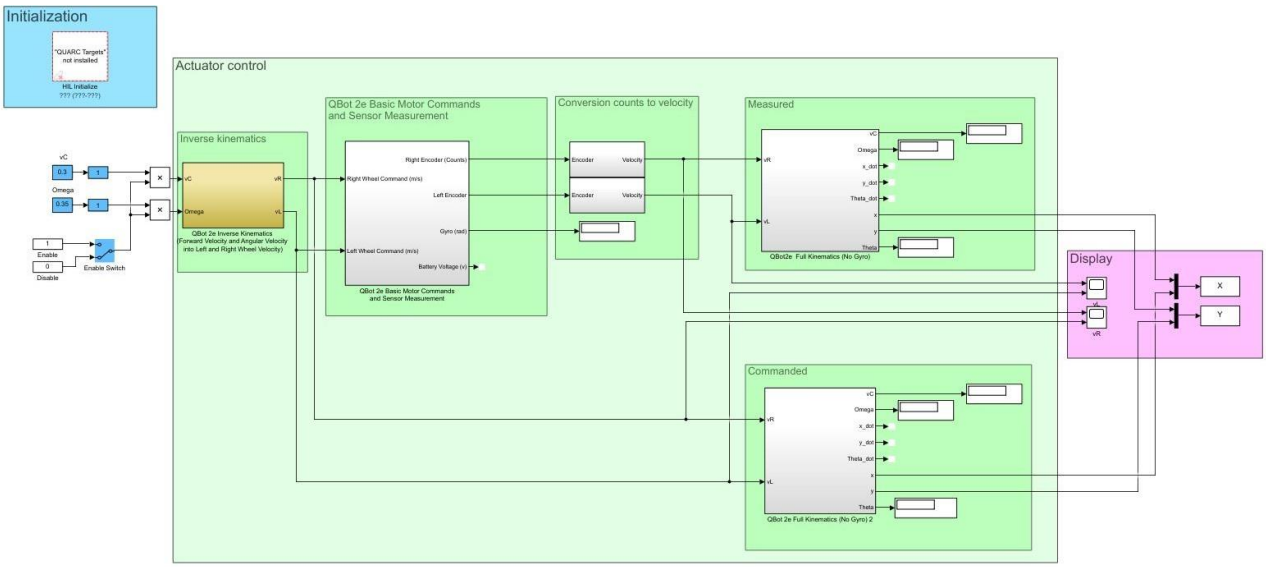


Figure 1.25: Snapshot of the Simulink file **QBot2e_Inverse_Kinematics.mdl**

1. Compile and run the file
2. Wait until the QBot has fully initialized, then enable the movement of the robot using the manual switch.
3. Set the desired forward speed $v_C = 0.1$ m/s and the desired turning rate $w_C = 0.1$ rad/s.
4. Run the model.
5. Record and observe the corresponding wheel speed commands v_R and v_L and the measured values. Compare them with the wheel speeds computed in the Forward Kinematics experiment.
6. Set the desired forward speed $v_C = 0$ m/s and the desired turning rate $w_C = 0.2$ rad/s and run the model again. Observe the behaviour of the robot as well as the measured v_L and v_R signals

1.5 Odometric Localization and Dead Reckoning

The objective of this section is to explore the concept of Odometric Localization by using the QBot platform.

1.5.1 Background

Odometric Localization, also known as Dead Reckoning, is the estimation of a robot's position and orientation (pose) based on the measured or estimated motion of the robot.

In the case of the QBot mobile robot, the procedure for odometric localization involves estimating the wheel speeds, ($v_R(t)$ and $v_L(t)$), based on encoder data or the integrated gyro. The forward kinematics model is then applied to estimate the robot chassis' linear speed, $v_C(t)$, and angular rate, $w_C(t)$. The data is then integrated over time starting from a known initial location, ($x(0)$ and $y(0)$), and heading, $\vartheta(0)$, to obtain an estimate of the robot chassis' pose.

This approach to localization is the most basic methodology used in mobile robotics, but is still routinely applied in industrial robotics applications that do not require high-fidelity location estimation, or as a redundant backup system for validation and error detection.

1.5.2 Equations of Motion

Given the robot chassis state vector, $S(t)$, and its rate of change, $\dot{S}(t)$, expressed in the global inertial frame, the robot pose at time, t, can be computed as follows:

$$S(t) = \begin{bmatrix} x(t) \\ y(t) \\ \vartheta(t) \end{bmatrix} = \int_0^t \dot{S}(t)\, dt \tag{1.18}$$

For the QBot, given the wheel separation, d, the heading, ϑ, and the wheel speed, v_R and v_L, the equations of motion for odometric localization are given by:

$$S(t) = \begin{bmatrix} x(t) \\ y(t) \\ \vartheta(t) \end{bmatrix} = \int_0^t \begin{bmatrix} \frac{1}{2}(v_R(t) + v_L(t)) \cos\vartheta(t) \\ \frac{1}{2}(v_R(t) + v_L(t)) \sin\vartheta(t) \\ \frac{1}{d}(v_R(t) - v_L(t)) \end{bmatrix} dt \tag{1.19}$$

In MATLAB/Simulink, it is easy to use the built-in integration blocks to solve the odometric calculations. However, for low-level languages, we need to employ other integration methods. For example, for a small time step, δt, the above QBot equations of motion can be approximated as a first-order Taylor series expansion:

$$S(t+\delta t) = S(t) + \dot{S}(t)\delta t = \begin{bmatrix} x(t) \\ y(t) \\ \vartheta(t) \end{bmatrix} + \begin{bmatrix} \frac{1}{2}(v_R(t) + v_L(t)) \cos\vartheta(t) \\ \frac{1}{2}(v_R(t) + v_L(t)) \sin\vartheta(t) \\ \frac{1}{d}(v_R(t) - v_L(t)) \end{bmatrix} \delta t \tag{1.20}$$

1.5.3 Trajectory Errors

The Simulink file for this experiment, shown in Figure 1.26, is called **QBot2e_Odometric_Localization.mdl**.

The Simulink model implements an open-loop trajectory controller for the QBot. The specified path consists of the following segments:

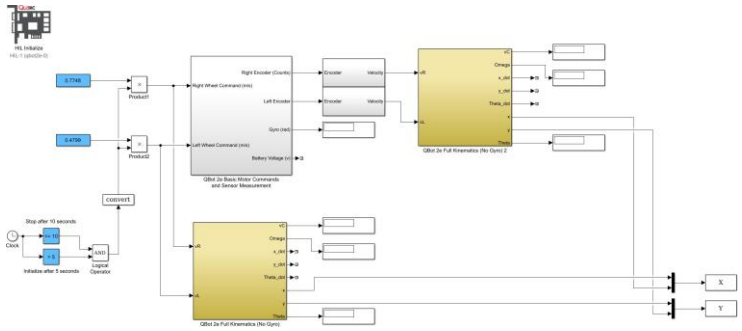


Figure 1.26: Snapshot of the controller model **QBot2e_Odometric_Localization.mdl**

- Rotate in a large circle with a radius of 0.5 m.
- Stop when the robot has returned to the initial position.

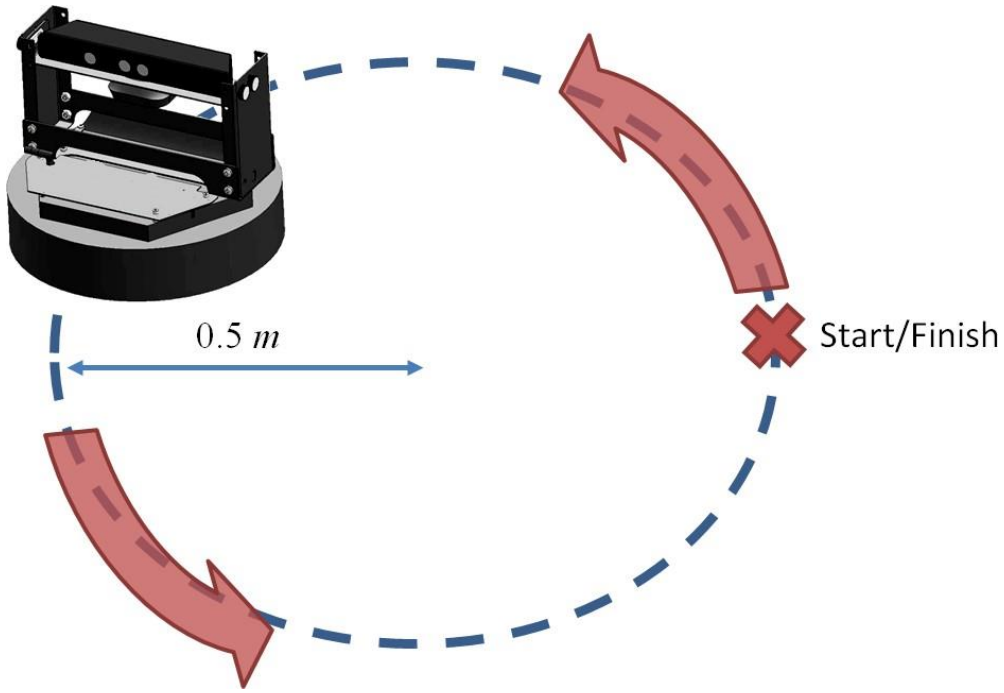


Figure 1.27: Path following controller desired path

At the end of the specified path, the QBot is expected to have returned to approximately the starting point.

Tasks

1. Place the Qbot robot is the middle of the room.
2. Mark the starting location and heading, prior to running the Simulink model.
3. Compile and run the Simulink model, then follow the procedure outlined below. Make observations on the motion of the robot, and answer the associated questions.
4. Once the QBot has fully initialized (upon hearing the QBot start-up chime), it will begin to move.

5. Measure and record the final position and heading of the robot chassis with respect to the starting position and heading. A MATLAB function called **plotXY.m** has been provided to generate an appropriate plot for the path analysis.

   - What is the error between the desired end-point of the robot, and the actual final *x*-*y* position? Recall, the *x* axis is in the forward/backward direction of the robot and the *y* axis is in the left/right direction.
   - What is the heading error?

   The XY plot should look similar to the figure below

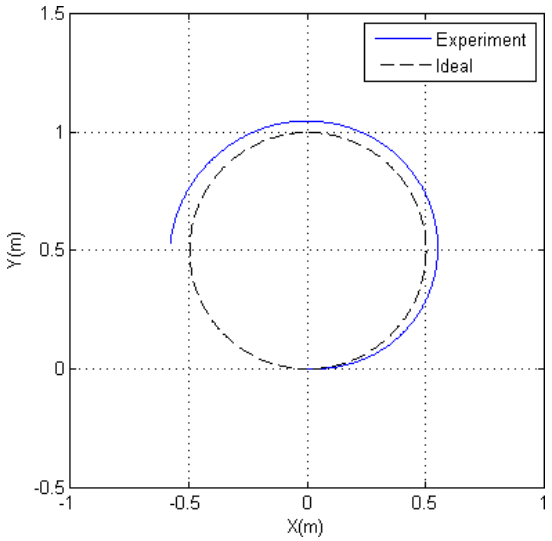


Figure 1.28: Measured and ideal robot path for 5 seconds.

   The robot travels less than the ideal calculated distance due to the speed tracking error mostly at the beginning of the motion when the robot is trying to overcome the friction and has not yet reached the desired speed. This error depends highly on the type of flooring that causes the friction. In Figure 1.28, the robot has traveled almost 3/4 of the full circle, and therefore, would require more time to reach the final destination.

6. Determine the theoretical right wheel and left wheel commands, and time required, for the QBot to travel in a straight line 2 meters in length. Enter your values into the appropriate fields in the Simulink model, indicated in blue and shown in 1.27.

7. Repeat steps 1 and 2, and record your new results. Modify the wheel commands and time values until the robot is able to reach the correct position.

8. Identify and discuss the different factors that contribute to the path/trajectory tracking error. Specifically, note the error reported by the Simulink model and the error based on direct physical measurements.

1.6 Vision guided control

The objective of this experiment is to study computer vision for robotic applications using the QBot mobile platform.

1.6.1 Computer Vision and Image Processing

The field of computer vision is chiefly concerned with the processing of digital images using computers. It normally consists of the development of processes and algorithms whose inputs and outputs are images, and extract attributes from images such as lines, corners, specific colors, and object locations. For the field of robotics, image processing is often used for navigation and mapping, but can also be used for more advanced topics including facial recognition, dynamic path planning, etc.

The objective of this experiment is to explore some useful image processing techniques using the Quanser QBot Mobile Platform.

A digital image can be defined as a twodimensional function, $f(x, y)$, where x and y are spatial coordinates, and the amplitude of f at any pair of coordinates (x, y) may be a scalar or a three element vector. When the scalar represents a value proportional to the energy of the visual spectrum of the electro magnetic (EM) field at the coordinate (x, y), the image is called a grayscale image. On the other hand, when the vector amplitude represents the energy of red, green, and blue colours in the visible EM spectrum, the image is called a colour image or RGB (RedGreenBlue).

In digital image acquisition, photo sensors are arranged in a 2D array where each photo sensor indicates a point in the discrete spatial coordinate and is called a picture element or pixel. The spatial coordinate and is called a picture element or pixel. The electrical signals from photo sensors are digitized using a quantizer to produce a digital image which can be stored in a memory chip.

Therefore, when working with images in this experiment, you will be dealing with mxn (for grayscale images) or mxnx3 for colour images, where m is the numbers of pixels in a row (number of columns) and n is the number of the pixels in a column (number of rows).

Once an image has been acquired, it can be processed. Image processing fundamentally involves the manipulation of digital images using a computer. Image processing techniques are widely used for visual based control of mobile vehicles. The following subsections briefly describe selected image processing techniques covered in the QBot experiments.

There are several types of image processing, we are going to explore three methods:

- **Image thresholding**
- **Edge detection**;
- **Blob analysis**.

1.6.1.1 Image thresholding

Thresholding is an operation that is often used to isolate specific colours or brightness levels in an image.

For more information:
https://en.wikipedia.org/wiki/Thresholding_(image_processing)

The Simulink file for this experiment is **QBot2e_Image_Processing_Color_Thresholding.mdl** shown in Figure 1.29

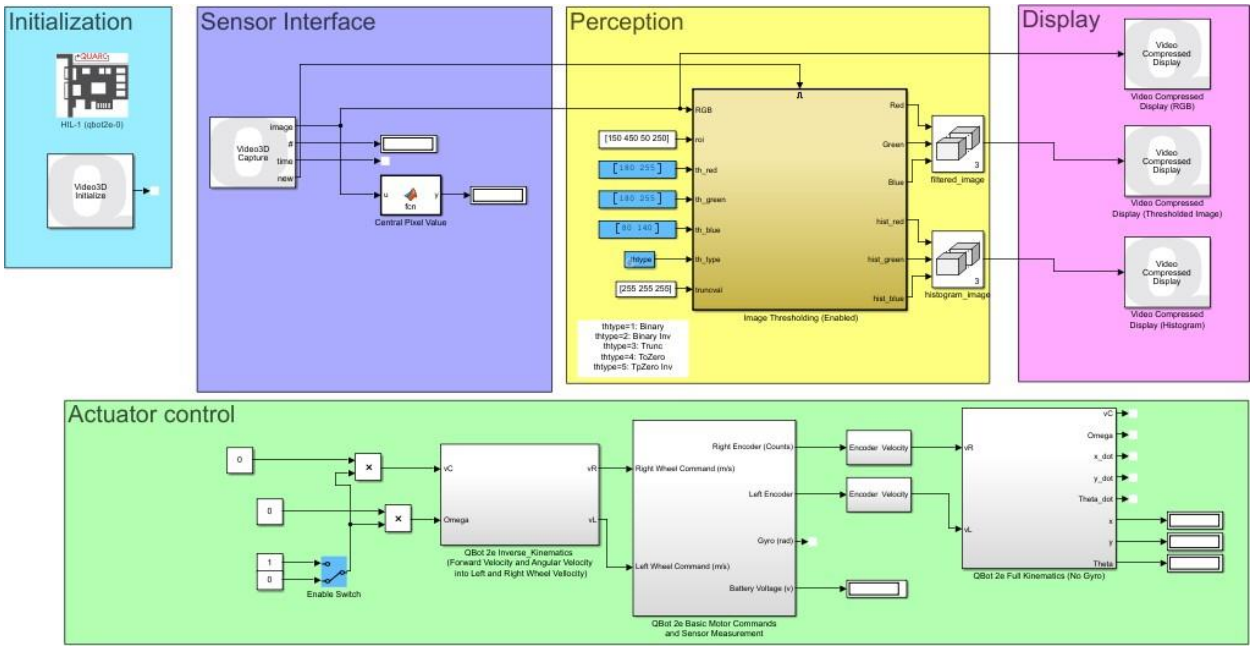


Figure 1.29: Snapshot of the Simulink **QBot2e_Image_Processing_Color_Thresholding.mdl** file

The Image Thresholding block, in yellow, implements the thresholding algorithm using the different methods described in the background section. This block receives the frame, region of interest (*roi*), thresholds on all three color channels (*th_red*, *th_green*, *th_blue*), the method (*th_type*) and the truncation values (*thuncval*) required for the truncation approach. This block outputs the processed image through RGB channels as well as the histograms.

Follow the procedure outlined below. Make observations about the effect of the algorithms on the generated images, and answer the associated questions.

1. Open the file, **QBot2e_Image_Processing_Color_Thresholding.mdl**, and make sure the manual switch (Enable switch) is set to zero. Compile the file and run it.
2. Double click on the three Video Compressed Display blocks in your model.
3. Find or create a colored sheet of paper, and cut out a 10 cm x 10 cm square.
4. Bring the colored sheet close to the QBot . Scroll your mouse over the image in the video display and note the RGB data in the title for a few points on the card. Tilt the card in various orientations and observe the changes in the measured colors. Then define a range for the RGB values that represent the color of the card.

5. Inspect the Video Compressed Display (Histogram) window, and observe the changes in the output when you bring objects with different colors, including the colored piece you just made, in front of the robot. Can the histogram help you find the [*minmax*] ranges (the histogram is scaled by a factor of 10)? Save a snapshot of the image in each case, and discuss the results. You can save a snapshot by using the Save icon.

6. Using the range that you found, set the [min max] values of the three threshold parameters, *th_red*, *th_green*,and *th_blue*.

7. If the filtering does not seem good enough, try tuning the threshold values until the colored piece of paper is completely filtered out.

8. Doubleclick on the *th_type* variable, and select the various options onebyone, observing what each method does to the results. Save a snapshot in each case, and discuss the results.

1.6.1.2 Edge detection

An edge is a set of similar, connected pixels that lie on the border between two regions. This method allows you to detect the edge of different shapes.

For more information:
https://en.wikipedia.org/wiki/Edge_detection

The Simulink file for this experiment is **QBot2e_Edge_Detection.mdl** shown in Figure 1.30

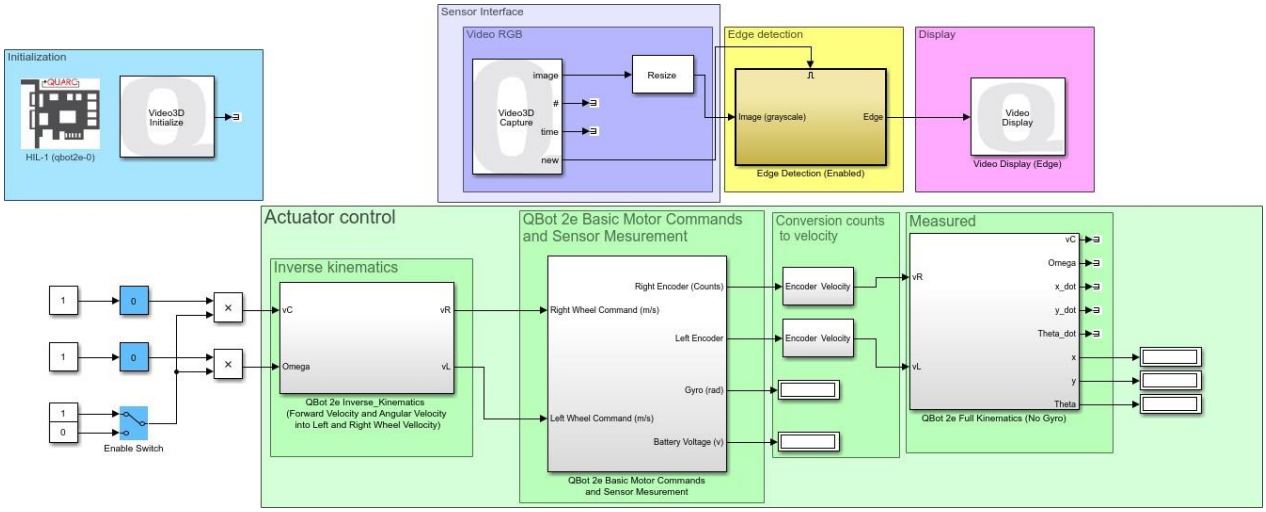


Figure 1.30: Snapshot of the Simulink QBot2e_Edge_Detection.mdl file

The Edge Detection subsystem, shown in yellow, processes the image using the mask convolution technique described in the Background section. Doubleclick on this block, open the embedded mathscript and find the masks and convolution functions used in the code.

Follow the procedure outlined below. Make observations about the effect of the algorithms on the generated images, and answer the associated questions.

1. Make sure the manual switch (Enable switch) as well as all the sliding gains (shown in blue) are set to zero.
2. Double click on the Video Display (Edge) block to open up the figure window.
3. Compile the file and run it once it is downloaded to the target.
4. Look at the resulting image showing the edges. Toggle the enable switch, and slowly change the slider gain related to Omega. Observe the effect on the generated image, and record your observations.
5. Toggle the manual switch to disable the algorithm, and stop the file.
6. Double click on the Edge Detection block to open the embedded mathscript. Compile the updated code and go through the steps above to access the comparative performance of your solution.
7. What would need to be changed if you were asked to implement a different edge detection technique?
8. Describe how you would extract the edges out of the image.

1.6.1.3 Blob analysis

Blob analysis makes it possible to detect the different regions that make up the image.

Fore more information:
https://en.wikipedia.org/wiki/Blob_detection

The Simulink file for this experiment is **QBot2e_Image_Proc_Blob_Tracking.mdl** shown in Figure 1.31

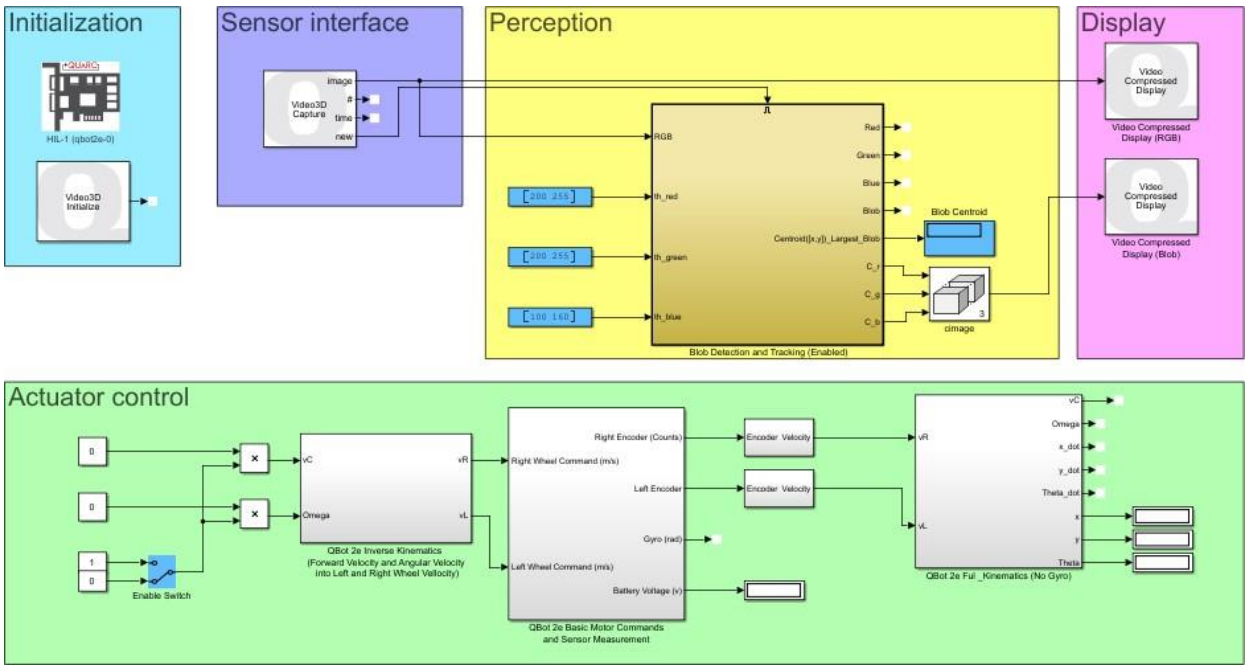


Figure 1.31: Snapshot of the Simulink **QBot2e_Image_Proc_Blob_Tracking.mdl** file

The yellow subsystem called **QBot2e_Image_Proc_Blob_Tracking** includes a thresholding algorithm on all three color channels, a blob filter (8connectivity default value) and a third block that finds the centroid of the largest blob.

Follow the procedure outlined below. Make observations about the effect of the algorithms and answer the associated questions.

1. Set the [min max] values of the three threshold parameters (highlighted with blue) that you found in the first part of the lab.

2. Make sure that the manual switch (Enable Switch) is set to zero.

3. Double click on the Video Compressed Display blocks.

4. Compile and run the model and look at the video display window.

5. Bring your colored piece in front of the robot and see if it is completely detected by the robot. Move if back and force, up and down and look at the Blob Centroid display (in blue). If the object is not fully detected as a blob, tune the [*minmax*] threshold values until your object is fully detected as one blob.

6. Explain how the blob centroids could be used to plan the motion of the robot.

1.6.2 Reasoning and Motion Planning

The reasoning and motion planning stage of a vision-based robotic application uses different image processing algorithms in order to generate appropriate motion commands for the robot. The objective of this experiment is to explore how these methods can be implemented on the Quanser QBot Mobile Platform.
The goal of this experiment is to use a set of image processing algorithms to interpret the environment around the robot, and create appropriate motion commands for the robot. As an example, you will learn how the QBot can follow a line on the floor.

1.6.2.1 Reasoning Based on Image Features

Image features, such as blob centroids, edges, corners, etc. are utilized primarily in robotics to make reasoned statements as to the environment around the robot, and an appropriate course of action. In order to create an algorithm that can make appropriate decisions, you will often need to remove much of the detail from the data that is provided by the image capture device in order to create clear judgments about the state of the robot and the environment. This act of using available data to create conditional statements is the basis for much of artificial intelligence.

Reasoning is the highest level of vision-based motion planning, extending several lower level image processing techniques. In this experiment, we will use blob centroids as âfactsâ about the environment that indicate where the next goal for the robot is, and generate motion commands based on ârulesâ for appropriate robot motion.

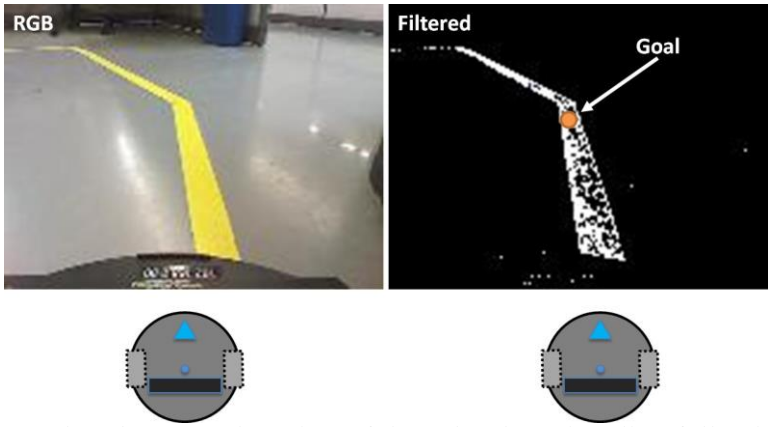


Figure 1.32: Goal and current location of the robot base in a line following scenario.

1.6.2.2 Motion Planning

Motion planning generally involves the creation of motion commands for the robot based on a series of goal positions, or waypoints. For this experiment experiment, the motion planning algorithm uses the centre of the current blob as the next goal for the position of the robot (command or setpoint), as well as the current location of the robot to create the motion path. Given that the Kinect sensor is mounted on top of the robot, and can see directly in front of the robot (when the Kinect is tilted down), we can always assume that the current location of the robot is a fixed distance below the last row of the image in the image coordinate frame. The goal (blob centroid) as well as the current location of the robot (below the last row of the image) in the image coordinate frame.

However, there are others trajectory tracking methods.

Tasks

The file for this part of the lab is **QBot2e_Image_Proc_Line_Following.mdl**. This file uses image processing algorithms, explicitly blob filtering, in order to find a line, locate the goal, and controls the robot to reach the target.

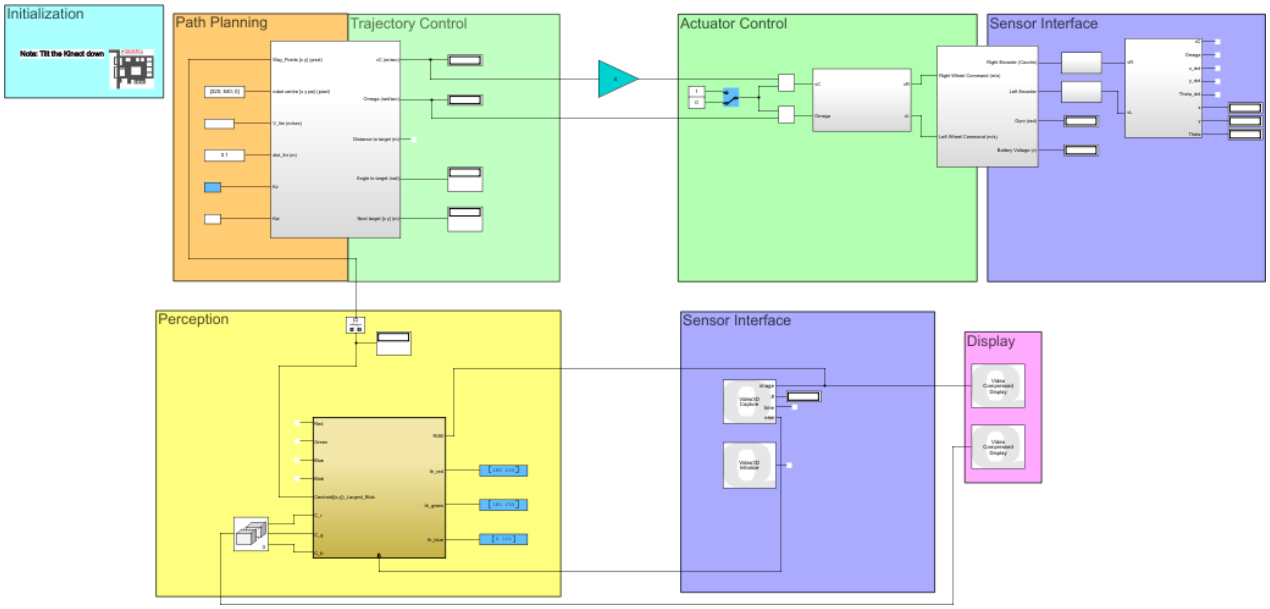


Figure 1.33: Snapshot of the **QBot2e_Image_Proc_Line_Following.mdl**

1. Open the Simulink model, **QBot2e_Image_Proc_Line_Following.mdl**, and make sure the enable switch (Manual Switch) is the actuator control block set to zero. Compile the model and run it.

2. Doubleclick on the two Video Compressed Display blocks.

3. Pick a color tape of your choice, and tape in the floor to make a line for the robot to follow as shown in Figure 1.32.

4. Put the robot on the floor right in front of your start point of the line. Tune the [*minmax*] values of the three threshold values, *th_red*, *th_green* and *th_blue*, highlighted in blue, while looking at the Video Compressed Display window so that the line is clearly filtered in the resulting image. To achieve good results move your cursor over different points in the video display and use the RGB information in the title to estimate the RGB range of values for this card. Once the tuning is successful, enable the manual switch and see if the robot can follow the line. You can change the K_v gain (in blue), which is the control gain of the robot. The larger this value, the faster the robot will move. Note that by increasing K_v, you might make the controller unstable.

1.7 Conclusion

Summarize and reflect on what you have seen and learned in this lab.

1.8 Troubleshooting

How to fix some issues that may appear:

1. **If the router does not communicate with the computer**: follow the instructions in "Quick Start Guide" (in the Folder User_guide).

2. **If the robot does not communicate with the computer:**

   i - Verify the robot is turned on.

   ii - Open "invite de commandes" and ping network :"ping 192.168.2.1".

   iii - Verify that there is not a second router in the room. **If there is**: verify that the robot does not communicate with the second network: ping the robot from the computer that could be connected to the second router.
   If the robot communicates with the second router: turn off the second router (switch at the rear), wait and ping the address of the robot on the first computer. After the connection is established: turn on the second router.

3. **For any error:**:

   i - Verify that *"C : \ProgramFiles\Quanser"* is added to the Matlab path.

   ii - Verify that *"C : \GW"* is added to the Matlab path.

   iii - Delete the files that do not ": ".mdl", ".m" or ".jpg"

   iv - Delete the folder "slprj" and "name_of_program_quarc_linux_pi_3".

   v - Quit and restart Matlab.

   vi - Reboot the computer.

4. **If a Message Box pops up, saying the file is not added to the path, here is the method to follow** : Click Add To Path

English to French glossary

bandwidth	:	bande passante
castor	:	roulette
chime	:	carillon
crane	:	grue
closed-loop system	:	système bouclé
cut-off frequency	:	fréquence (ou pulsation) de coupure
damped frequency	:	pulsation amortie
damping ratio	:	coefficient d'amortissement
drag	:	traînée
feedback	:	contre-réaction
feedback system	:	système à contre-réaction
hoisting device	:	dispositif de levage
impulse response	:	réponse impulsionnelle
integral wind-up	:	emballement (de l'action) intégral
input	:	entrée
gain	:	gain
heading angle	:	angle de cap
linear time-invariant (LTI)	:	linéaire invariant dans le temps
motor shaft	:	arbre moteur
output	:	sortie
overdamped	:	sur-amorti
overshoot	:	dépassement
rise time	:	temps de montée
road grade	:	inclinaison de la route
robot arm joint	:	articulation d'un bras de robot
root locus	:	lieu des racines
setpoint	:	consigne
settling time	:	temps de réponse
steady-state gain	:	gain statique
steady-state response	:	réponse en régime permanent
steering	:	direction
step response	:	réponse indicielle
stream	:	courant
yaw angle	:	angle de lacet

time-delay	:	retard pur
time-invariant	:	invariant dans le temps
transient response	:	réponse transitoire
throttle	:	accélérateur
undamped	:	non amorti
undamped natural frequency	:	pulsation propre non amortie
underdamped	:	sous-amorti