EXPERIMENT 3: COMPUTER VISION AND VISION-GUIDED CONTROL

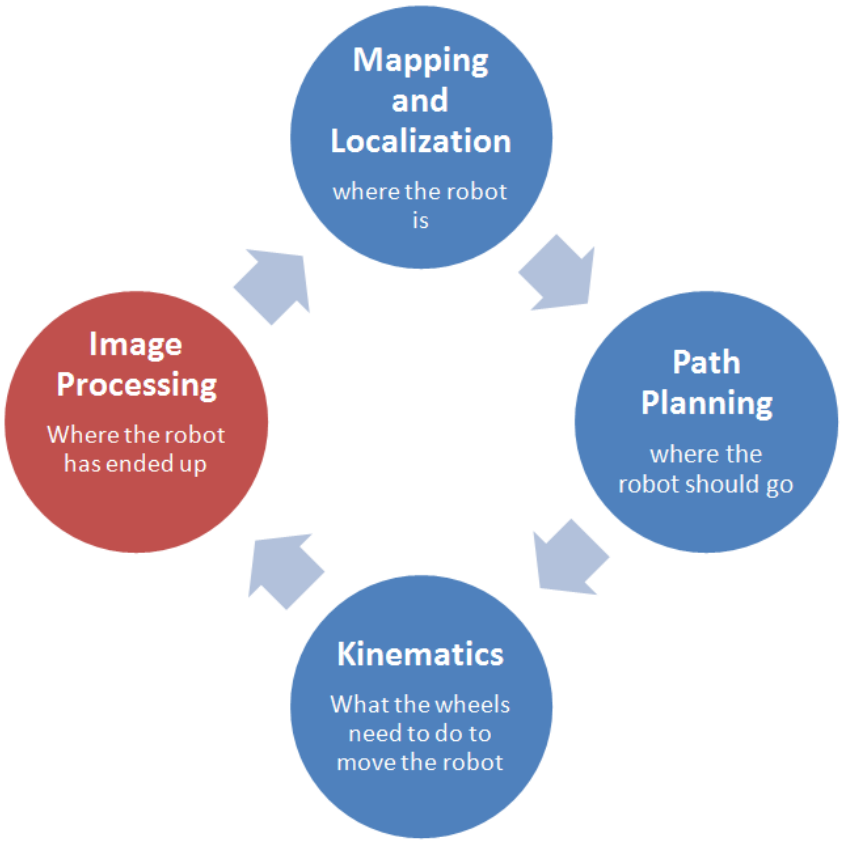
The objective of this experiment is to study computer vision for robotic applications using the QBot 2. The following topics will be studies in this laboratory.

Topics Covered

- Image Processing Techniques
- Reasoning and Motion Planning

Prerequisites

- The QBot 2 has been setup and tested. See the QBot 2 Quick Start Guide for details.
- You have access to the QBot 2 User Manual.
- You are familiar with the basics of Matlab® and Simulink®.



Vision and image processing is used to detect properties of the environment around a robot

IMAGE PROCESSING

The field of digital image processing 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 exercise is to explore some useful image processing techniques using the Quanser QBot 2 Mobile Platform.

Topics Covered

- Image Thresholding
- Edge Detection
- Blob Analysis

1 Background

Visual sensing plays a key role in robotic applications. Mobile robots use visual feedback to build an internal representation of the environment, which is used in the decision-making process of the robot for motion control. Figure 1.1 describes a typical vision-based robotic application which consists of the following steps

1. **Perception**, which includes image acquisition and processing
2. **Localization and Path Planning**
3. **Motion Control**, which includes kinematics and motion control

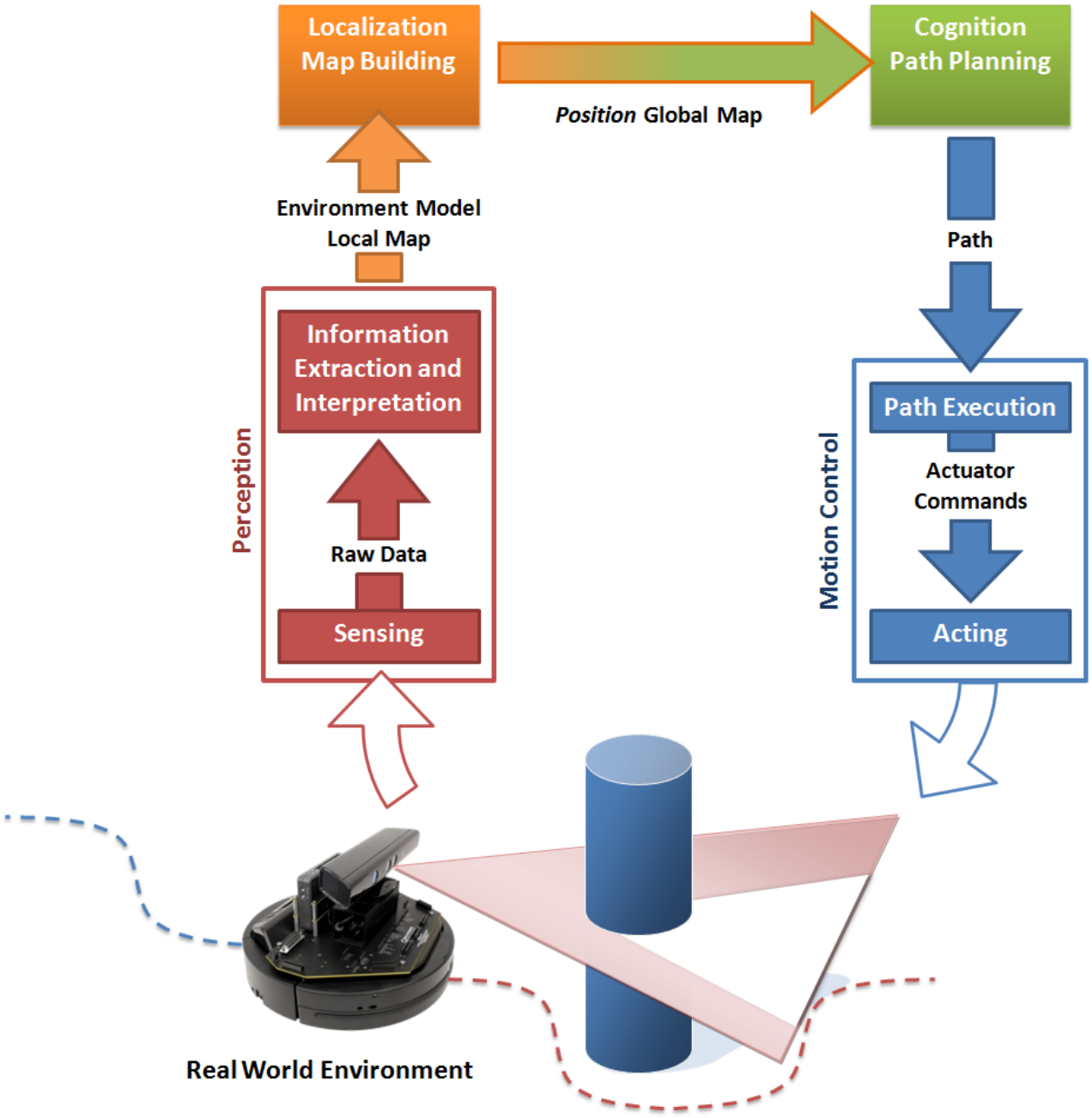


Figure 1.1: Vision-based robotic applications.

The following sections briefly describes the individual components of the block diagram:

1.1 Perception: Image Acquisition and Processing

A digital image can be defined as a two-dimensional 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 gray-scale 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 (Red-Green-Blue).

In digital image acquisition, photo sensors are arranged in a 2-D array where each photo sensor indicates a point in the discrete 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 laboratory, you will be dealing with $m \times n$ (for gray-scale images) or $m \times n \times 3$ 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 sub-sections briefly describe selected image processing techniques covered in the QBot 2 experiments.

1.1.1 Image Thresholding

Thresholding is an operation that is often used to isolate specific colours or brightness levels in an image. In general, a spatial domain process like thresholding is denoted by the following expression:

$$h(x, y) = T\big(f(x, y)\big), \tag{1.1}$$

where $f(x, y)$ is the input image, $h(x, y)$ is the processed image, and T is the thresholding operation that can be implemented on the image pixels in one of the following ways:

- Binary thresholding:
$$T\big(f(x, y)\big) = \begin{cases} 255 & \text{if } th_1 \leq f(x, y) \leq th_2 \\ 0 & \text{otherwise} \end{cases}$$
- Binary thresholding inverted:
$$T\big(f(x, y)\big) = \begin{cases} 0 & \text{if } th_1 \leq f(x, y) \leq th_2 \\ 255 & \text{otherwise} \end{cases}$$
- Truncate to a value
$$T\big(f(x, y)\big) = \begin{cases} trunc & \text{if } th_1 \leq f(x, y) \leq th_2 \\ f(x, y) & \text{otherwise} \end{cases}$$
- Threshold to zero:
$$T\big(f(x, y)\big) = \begin{cases} f(x, y) & \text{if } th_1 \leq f(x, y) \leq th_2 \\ 0 & \text{otherwise} \end{cases}$$
- Threshold to zero, inverted:
$$T\big(f(x, y)\big) = \begin{cases} 0 & \text{if } th_1 \leq f(x, y) \leq th_2 \\ f(x, y) & \text{otherwise} \end{cases}$$

where the threshold range is defined with $[th_1 th_2]$ and $trunc$ denotes an arbitrary level of truncation. The above equations can be used for gray-scale image thresholding directly. For colour image thresholding, similar functions can be applied to each channel of the image. For example, binary thresholding for RGB images can be written as:

$$T\big(f(x,y)\big) = \begin{cases} f(x,y) & \text{if range-condition } = true \\ 0 & \text{otherwise} \end{cases} \tag{1.2}$$

in which range-condition can be written as the logical AND of three conditions, $th_{r,1} \leq f(x,y) \leq th_{r,2}$, $th_{g,1} \leq f_g(x,y) \leq th_{g,2}$ and $th_{b,1} \leq f_b(x,y) \leq th_{b,2}$ where r, g and b subscripts denote the red, green and blue channels, respectively.

1.1.2 Edge Detection

An edge is a set of similar, and connected pixels that lie on the boundary between two regions. Edge detection is performed by convolving an input image with gradient masks. The convolution process involves computing the sum of products of mask (also called window or kernel) coefficients with the gray levels contained in the region encompassed by the mask. For example, Figure 1.2 shows a 3x3 moving mask on an image, where the mask center $w_{0,0}$ coincides with the image location at (x,y). The following equation describes how the processed image, $h(x,y)$, is calculated based on the gray value of the original image $f(x,y)$ at (x,y) using the convolution procedure when the mask size is $(2p+1) \times (2q+1)$.

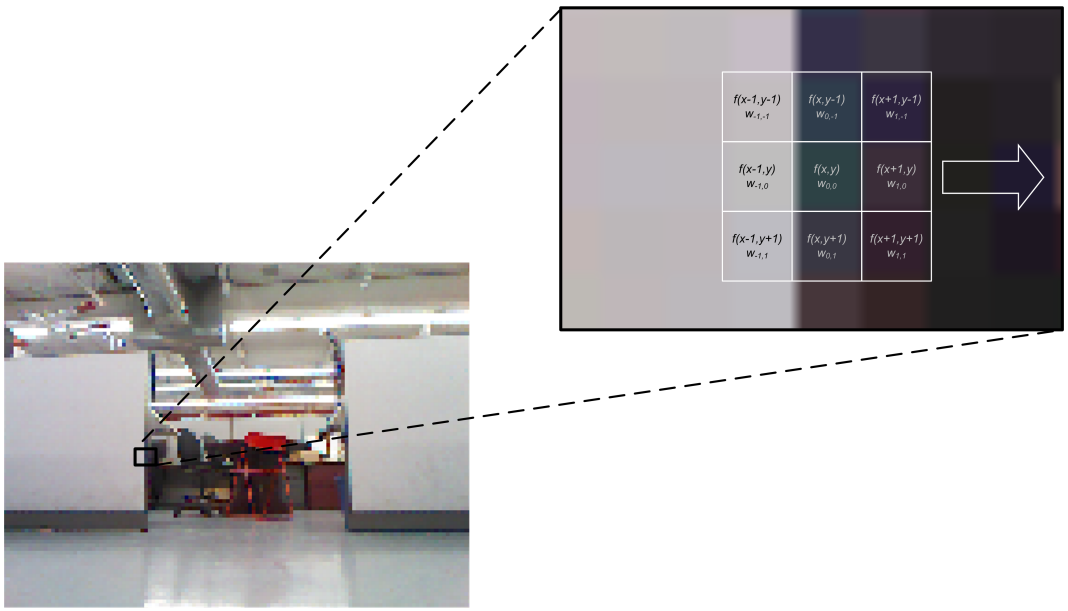


Figure 1.2: Mask operation in image processing.

$$h(x,y) = \sum_{-p \leq i \leq p} \sum_{-q \leq j < \leq q} w_(i,j) \times f(x+i, y+j) \tag{1.3}$$

The edge detection procedure employs horizontal, w_x, and vertical, w_y, gradient masks to determine image gradients $G_x(x,y)$, $G_y(x,y)$ in x and y directions, respectively. The overall gradient image is defined by the following:

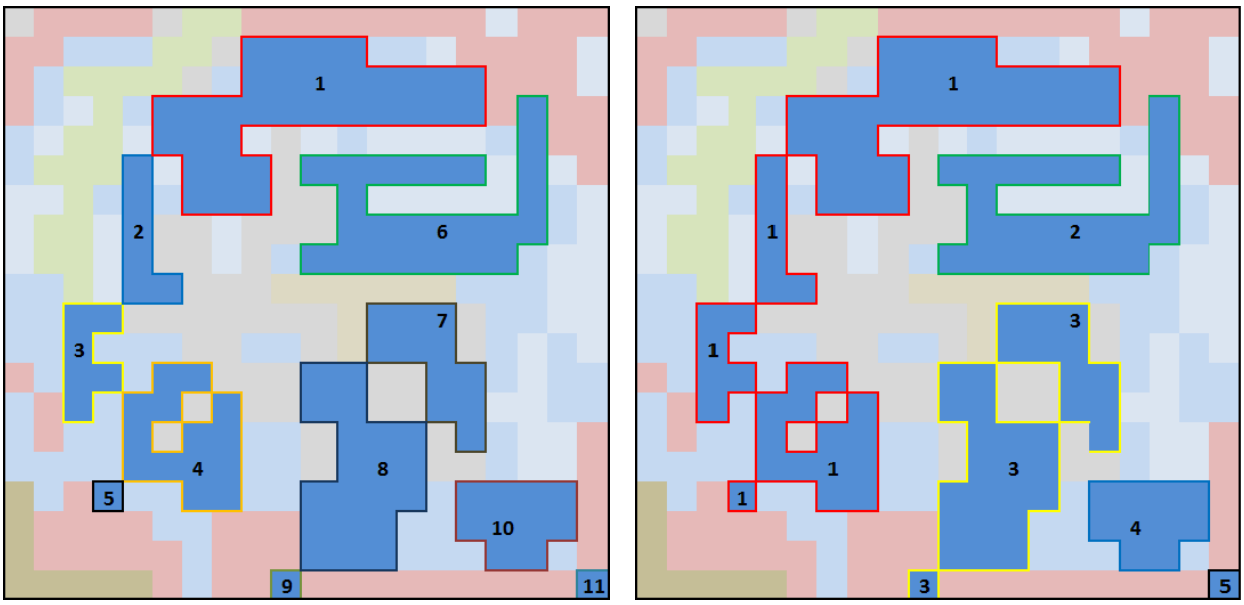
$$G(x,y) = \sqrt{G_x^2(x,y) + G_y^2(x,y)}, \tag{1.4}$$

where $G_x = conv\big(f(x,y), w_x\big)$ and $G_x = conv\big(f(x,y), w_y\big)$. Different implementations exist for gradient masks which determine the type of the edge detection algorithm. For instance, *Sobel* edge detection method uses the following

gradient masks of size 3x3.

$$w_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, w_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{1.5}$$

1.1.3 Blob Analysis



(a) 4-Connectivity

(b) 8-Connectivity

Figure 1.3: Example of blob analysis connectivity types when finding blue coloured blobs.

Blob analysis involves the segmentation of images based on connected components (blobs), and analysis of various blob properties, e.g., area and centroid of the blobs. The shape of the connected components may vary based on the type of connectivity, which is commonly a 4 or 8-connected type. A 4-connected type refers to pixels that are neighbors to every pixel that touches one of their edges. These pixels are connected horizontally and vertically. 8-connected pixels are neighbors to every pixel that touches one of their edges or corners. These pixels are connected horizontally, vertically, and diagonally. Figure 1.3 visualizes the 4 and 8-connectivity concepts in digital images.

2 In-Lab Exercise

2.1 Image Thresholding

The first controller model that is used for this lab is “QBot2_Image_Processing_Color_Thresholding.mdl” shown in Figure 2.1.

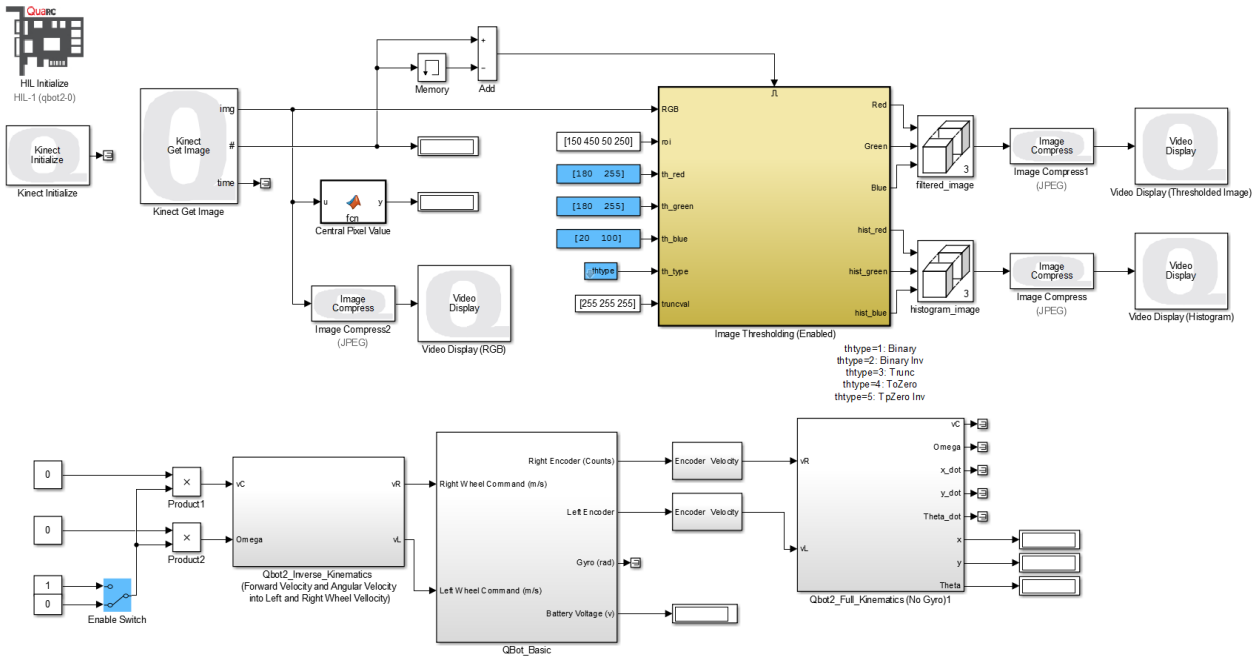


Figure 2.1: Snapshot of the controller model “QBot2_Image_Processing_Color_Thresholding.mdl”

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 supplied model, QBot2_Image_Processing_Color_Thresholding, and make sure the manual switch (Enable switch) is set to zero. Compile the model and run it.
2. Double click on the three *Video Display* blocks in your model.
3. Find or create a colored sheet of paper, and cut out a 10cm x 10cm square.
4. Bring the colored sheet close to the QBot 2. 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 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 [min max] ranges (the histogram is scaled by a factor of 10)? Save a snap-shot of the image in each case, and discuss the results. You can save a snap-shot 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. Double-click on the “th_type” variable, and select the various options one-by-one, observing what each method does to the results. Save a snap-shot in each case, and discuss the results.

2.2 Edge Detection

The next controller model for this lab is “QBot2_Edge_Detection.mdl” shown in Figure 2.2.

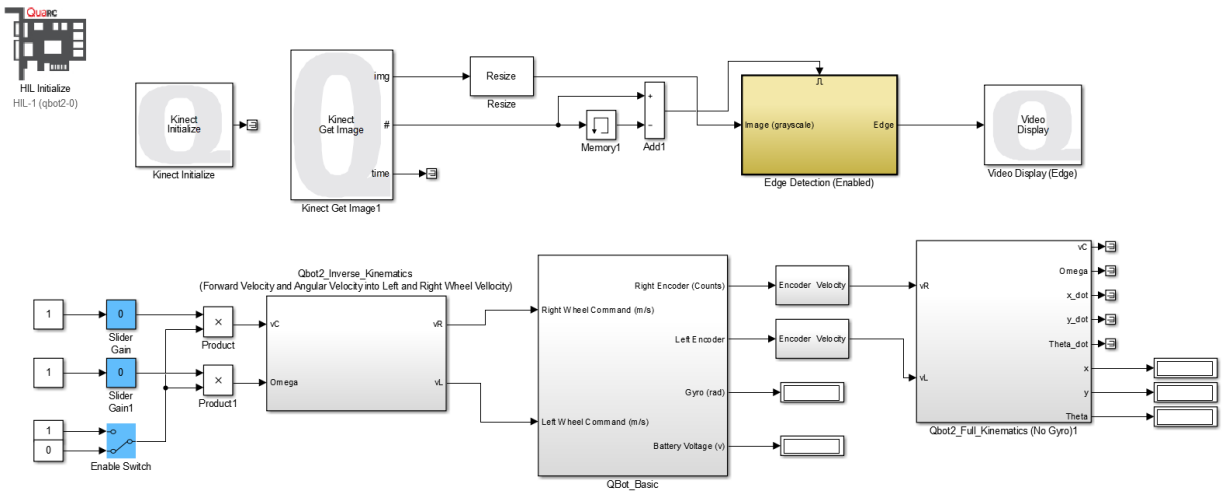


Figure 2.2: Snapshot of the controller model “QBot2_Edge_Detection.mdl”

The “Edge_Detection” subsystem, shown in yellow, processes the image using the mask convolution technique described in the Background section. Double-click 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 model 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 model.

6. Double-click on the “Edge Detection” block to open the embedded mathscript. Comment out the MATLAB convolution functions and implement your own convolution function based on the masks and convolution functions in the Background section (Equation 1.3). 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.

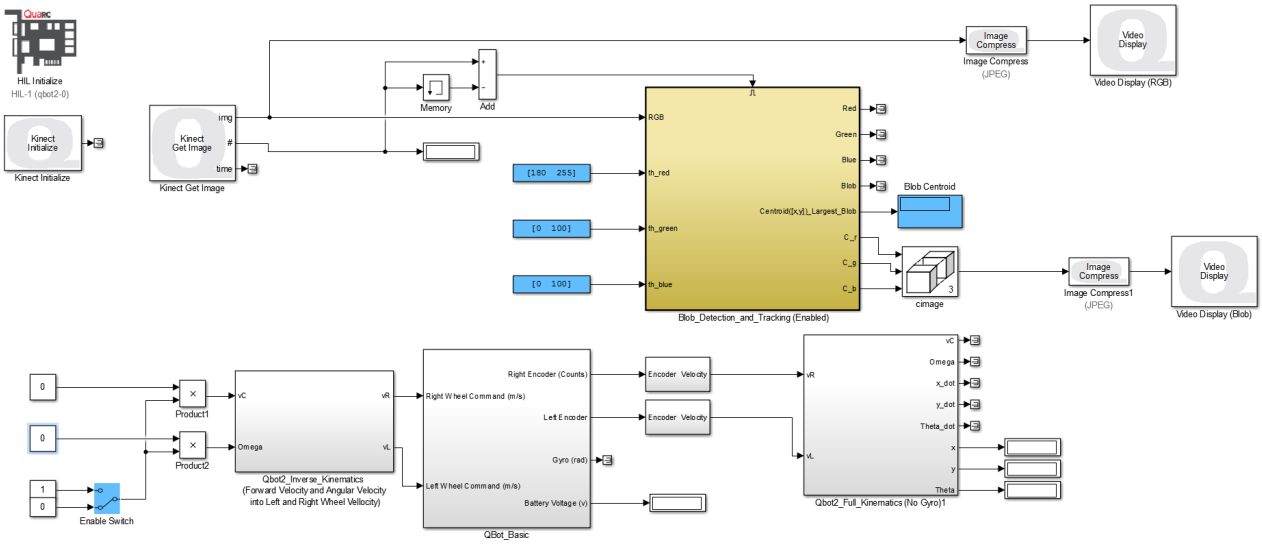


Figure 2.3: Snapshot of the controller model “QBot2_Image_Proc_Blob_Tracking.mdl”

2.3 Blob Detection

The model for this part of the lab is called “QBot2_Image_Proc_Blob_Tracking.mdl” shown in Figure 2.3.

The yellow subsystem called “Blob_Detection_and_Tracking” includes a) a thresholding algorithm on all three color channels, b) a blob filter (8-connectivity default value) and c) 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 Display” block.
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 [min max] 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.

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 exercise is to explore how these methods can be implemented on the Quanser QBot 2 Mobile Platform.

Topics Covered

- Reasoning based on the image features
- Motion Planning

1 Background

The goal of this lab 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 2 can follow a line on the floor.

1.1 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 laboratory, we will use blob centroids as “facts” about the environment that indicate where the next goal for the robot is, and generate motion commands based on “rules” for appropriate robot motion.

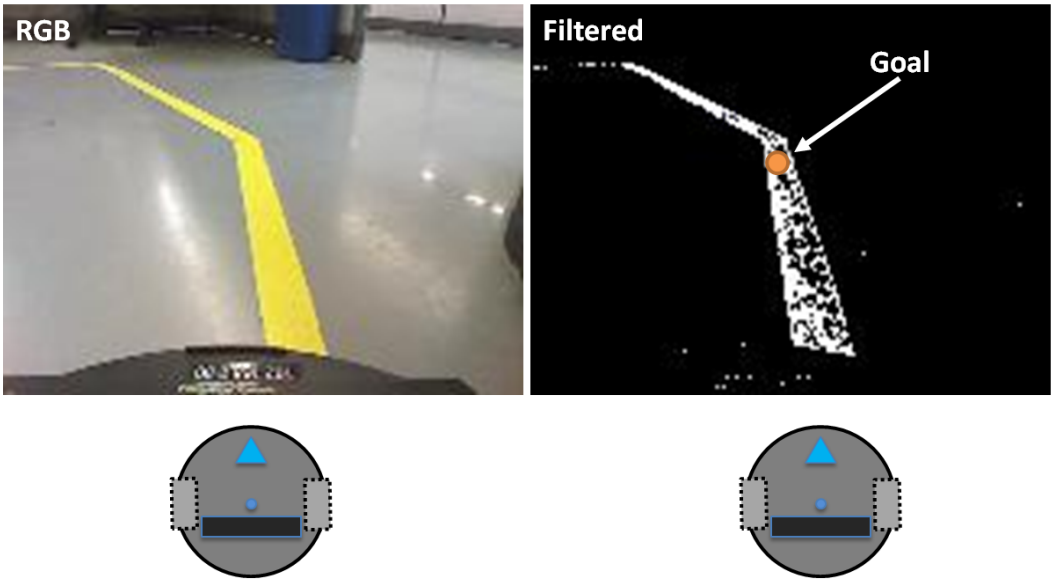


Figure 1.1: Goal and current location of the robot base in a line following scenario.

1.2 Motion Planning

Motion planning generally involves the creation of motion commands for the robot based on a series of goal positions, or way-points. For this laboratory experiment, the motion planning algorithm uses the centre of the current blob as the next goal for the position of the robot (command or set-point), 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. Figure 1.1 shows 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.

2 In-Lab Exercise

The model for this part of the lab is “QBot2_Image_Proc_Line_Following.mdl” shown in Figure 2.1. This model uses image processing algorithms, explicitly blob filtering, in order to find a line, locate the goal, and controls the robot to reach the target.

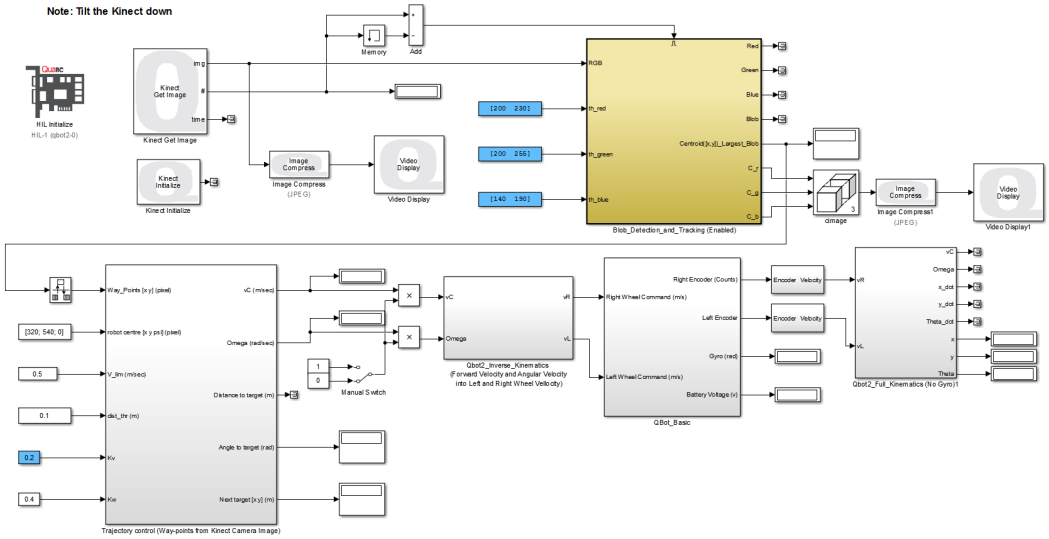


Figure 2.1: Snapshot of the controller model “QBot2_Line_Following.mdl”

Note: In this lab you need to tilt down the Kinect sensor so that the robot can see right in front of its chassis.

1. Open the supplied model, QBot2_Image_Proc_Line_Following.mdl, and make sure the manual switch (Enable switch) is set to zero. Compile the model and run it.
2. Double click on the two “Video Display” blocks in your model.
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 .
4. Put the robot on the floor right in front of your start point of the line. Tune the [min max] values of the three threshold values, th_red, th_green and th_blue, highlighted with blue, while looking at the “Video 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 Kv 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 Kv, you might make the controller unstable.

Quanser Inc.
119 Spy Court
Markham, Ontario
L3R 5H6
Canada
info@quanser.com
Phone: 1-905-940-3575
Fax: 1-905-940-3576

Printed in Markham, Ontario.

For more information on the solutions Quanser Inc. offers, please visit the web site at:
http://www.quanser.com