Sim.I.am: A Robot Simulator
Coursera: Control of Mobile Robots

Jean-Pierre de la Croix and Matthew Hale

Last Updated: March 9, 2016

Contents

1   Introduction

This manual is going to be your resource for using the simulator with the programming assignments featured in the Coursera course, *Control of Mobile Robots* (and included at the end of this manual). It will be updated from time to time whenever new features are added to the simulator or any corrections to the course material are made.

1.1   Installation

Download `simiam-coursera-week-X.zip` (where X is the corresponding week number for the assignment) from the *Optional Programming Assignment* page under *Week X*. Make sure to download a new copy of the simulator **before** you start a new week's programming assignment, or whenever an announcement is made that a new version is available. It is important to stay up-to-date, since new versions may contain important bug fixes or features required for the programming assignment.

Unzip the `.zip` file to any directory.

1.2   Requirements

You will need a reasonably modern computer to run the robot simulator. While the simulator will run on hardware older than a Pentium 4, it will probably be a very slow experience. You will also need a copy of MATLAB.

Thanks to support from MathWorks, a license for MATLAB and all required toolboxes are available to all students beginning in Week 2 and lasting until the end of the course. **You can easily put off the optional Week 1 MATLAB assignment until Week 2 without falling behind.** Check the *Installing MATLAB* page accessible from the *Week 2* page for a link that will let you install MATLAB on your computer.

1.3   MATLAB Help

Any questions about using MATLAB for the programming assignments should be posted to the course discussion forums.
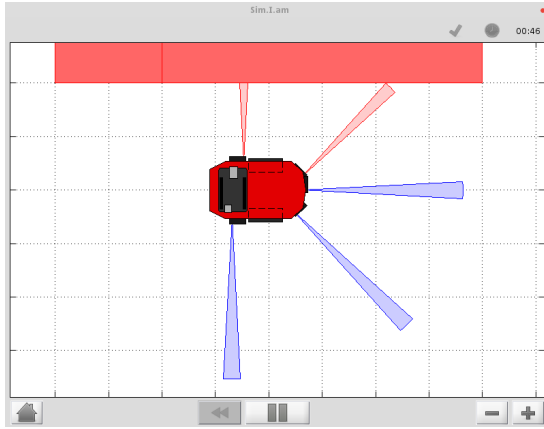
(a) Simulated QuickBot
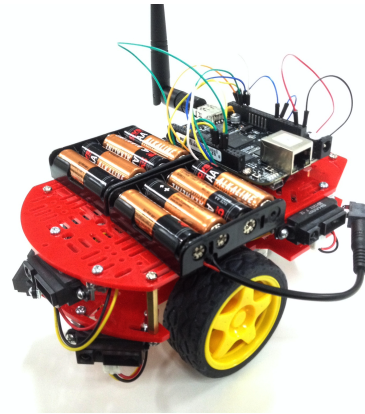
(b) Actual QuickBot

Figure 1: The simulated QuickBot and the mobile robot it is based upon.

2 Mobile Robot Simulator

The programming exercises use a simulated version of a robot called the QuickBot. The simulated QuickBot is equipped with five infrared (IR) range sensors, of which three are located in the front and two are located on its sides. The simulated QuickBot has a two-wheel differential drive system (two wheels, two motors) with a wheel encoder for each wheel. You can build the QuickBot yourself by following the hardware videos located under the *Course Resources* tab. **These hardware videos are not part of the course and their content is therefore unsupported by course staff. We provide them as a courtesy to learners in the course who may be interested in them.**

Figure 1 shows the simulated and actual QuickBot mobile robot. The robot simulator recreates the QuickBot as faithfully as possible. For example, the range, output, and field of view of the simulated IR range sensors match the specifications in the datasheet for the actual Sharp GP2D120XJ00F infrared proximity sensors on the QuickBot.
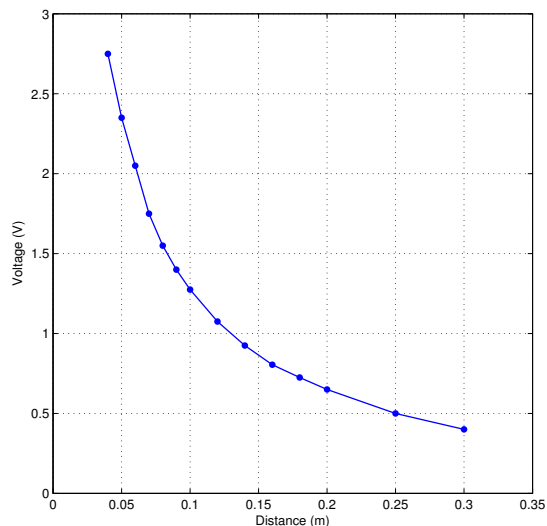
2.1 IR Range Sensors

In this section we cover some of the details pertaining to the five simulated IR sensors onboard the simulated QuickBot. The orientations (relative to the body of the QuickBot, as shown in Figure 1a) of IR sensors 1 through 5 are $90°, 45°, 0°, -45°$, and $-90°$, respectively. IR range sensors are effective in the range 0.04 m to 0.3 m only. However, the IR sensors return raw values in the range of $[0.4, 2.75]V$ instead of the measured distances. Figure 2a demonstrates the function that maps these sensors values to distances. To complicate matters slightly, the BeagleBone Black onboard the physical QuickBot digitizes the analog output voltage using a voltage divider and a 12-bit, 1.8V analog-to-digital converter (ADC). To faithfully recreate the QuickBot in simulation, we simulate the effect of this digitization. Figure 2b is a look-up table to demonstrate the relationship between the ADC output, the analog voltage from the IR proximity sensor, and the approximate distance that corresponds to this voltage.

Any controller can access the IR array through the `robot` object that is passed into its `execute` function. For example,

```
ir_distances = robot.get_ir_distances();
for i=1:numel(robot.ir_array)
  fprintf('IR #%d has a value of %d', i, robot.ir_array(i).get_range());
  fprintf('or %0.3f meters.\n', ir_distances(i));
end
```

It is assumed that the function `get_ir_distances` properly converts from the ADC output to an

Distance (m)	Voltage (V)	ADC Out
0.04	2.750	917
0.05	2.350	783
0.06	2.050	683
0.07	1.750	583
0.08	1.550	517
0.09	1.400	467
0.10	1.275	425
0.12	1.075	358
0.14	0.925	308
0.16	0.805	268
0.18	0.725	242
0.20	0.650	217
0.25	0.500	167
0.30	0.400	133

(a) Analog voltage output when an object is between 0.04m and 0.3m in the IR proximity sensor's field of view.

(b) A look-up table for interpolating a distance (m) from the analog (and digital) output voltages.

Figure 2: A graph and a table illustrating the relationship between the distance of an object within the field of view of an infrared proximity sensor and the analog (and digital) ouptut voltage of the sensor.

analog output voltage, and then from the analog output voltage to a distance in meters. The conversion from ADC output to analog output voltage is simply

$$V_{\text{ADC}} = \left\lfloor \frac{1000 \cdot V_{\text{analog}}}{3} \right\rfloor.$$

NOTE: For Week 2, the simulator uses a different voltage divider on the ADC; therefore, $V_{\textbf{ADC}} = V_{\textbf{analog}} \cdot 1000/2$. This has been fixed in subsequent weeks!

Converting from the the analog output voltage to a distance is a little bit more complicated, because a) the relationships between analog output voltage and distance is not linear, and b) the look-up table provides a coarse sample of points on the curve in Figure 2a. MATLAB has a `polyfit` function to fit a curve to the values in the look-up table, and a `polyval` function to interpolate a point on that fitted curve. The combination of the these two functions can be use to approximate a distance based on the analog output voltage, and this will be done as part of the programming assignment in Week 2.

2.2 Differential Wheel Drive

Since the simulated QuickBot has a differential wheel drive (i.e., is not a unicyle), it has to be controlled by specifying the angular velocities of the right and left wheel (v_r, v_l), instead of the linear and angular velocities of a unicyle (v, ω). These velocities are computed by a transformation from (v, ω) to (v_r, v_ℓ). Recall that the dynamics of the unicyle are defined as

$$\begin{aligned} \dot{x} &= v \cos(\theta) \\ \dot{y} &= v \sin(\theta) \\ \dot{\theta} &= \omega. \end{aligned} \tag{1}$$

The dynamics of the differential drive are defined as

$$\dot{x} = \frac{R}{2}(v_r + v_\ell)cos(\theta)$$
$$\dot{y} = \frac{R}{2}(v_r + v_\ell)sin(\theta) \qquad (2)$$
$$\dot{\theta} = \frac{R}{L}(v_r - v_\ell),$$

where R is the radius of the wheels and L is the distance between the wheels.

The speed of the simulated QuickBot can be set in the following way assuming that the `uni_to_diff` function has been implemented, which transforms (v, ω) to (v_r, v_ℓ):

```
v = 0.15; % m/s
w = pi/4; % rad/s
% Transform from v,w to v_r,v_l and set the speed of the robot
[vel_r, vel_l] = obj.robot.dynamics.uni_to_diff(robot,v,w);
obj.robot.set_speeds(vel_r, vel_l);
```

The maximum angular wheel velocity for the physical QuickBot is approximately 80 RPM or 8.37 rad/s and this value is reflected in the simulator. It is therefore important to note that if the simulated QuickBot is controlled to move at maximum linear velocity, it is not possible to achieve any angular velocity, because the angular velocity of the wheel will have been maximized. Therefore, there exists a tradeoff between the linear and angular velocity of the QuickBot: *the faster the robot should turn, the slower it has to move forward.*

2.3   Wheel Encoders

Each of the wheels is outfitted with a wheel encoder that increments or decrements a tick counter depending on whether the wheel is moving forward or backwards, respectively. Wheel encoders may be used to infer the relative pose of the simulated robot. This inference is called **odometry**. The relevant information needed for odometry is the radius of the wheel (32.5mm), the distance between the wheels (99.25mm), and the number of ticks per revolution of the wheel (16 ticks/rev). For example,

```
R = robot.wheel_radius; % radius of the wheel
L = robot.wheel_base_length; % distance between the wheels
tpr = robot.encoders(1).ticks_per_rev; % ticks per revolution for the right wheel

fprintf('The right wheel has a tick count of %d\n', robot.encoders(1).state);
fprintf('The left wheel has a tick count of %d\n', robot.encoders(2).state);
```

For more information about odometry, see the Week 2 programming assignment.

3   Simulator

Start the simulator with the `launch` command in MATLAB from the command window. It is important that this command is executed inside the unzipped folder (but not inside any of its subdirectories).
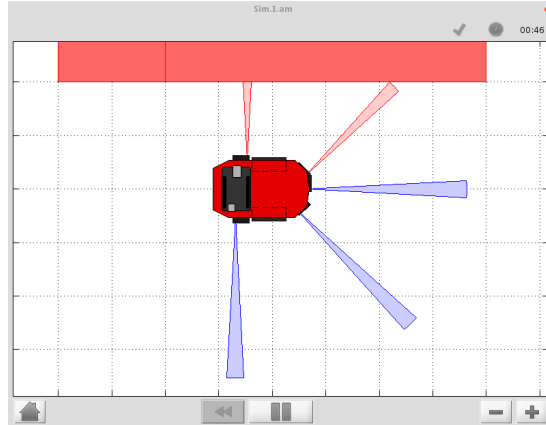

Figure 3: `launch` starts the simulator.

Figure 3 is a screenshot of the graphical user interface (GUI) of the simulator. The GUI can be controlled by the bottom row of buttons. The first button is the *Home* button and returns you to the home screen. The second button is the *Rewind* button and resets the simulation. The third button is the *Play* button, which can be used to play and pause the simulation. The set of *Zoom* buttons or the mouse scroll wheel allows you to zoom in and out to get a better view of the simulation. Clicking, holding, and moving the mouse allows you to pan around the environment. You can click on a robot to follow it as it moves through the environment.

4 Programming Assignments

The following sections serve as a tutorial for getting through the simulator portions of the programming exercises. Places where you need to either edit or add code are marked off by a set of comments. For example,

```
%% START CODE BLOCK %%
   [edit or add code here]
%% END CODE BLOCK %%
```

To start the simulator with the `launch` command from the command window, it is important that this command is executed inside the unzipped folder (but not inside any of its subdirectories).

4.1 Week 1

This week's exercises will help you learn about MATLAB and robot simulator. **If you do not already have access to MATLAB, a student license will be made available to registered participants of this course starting in Week 2. You can easily put off the optional Week 1 MATLAB assignment until Week 2 without falling behind.**

1. Install the GUI Layout Toolbox using the instructions provided in the *Preparing for Programming Assignments* page under *Week 1*.

2. Familiarize yourself with the simulator by reading this manual and downloading the robot simulator posted on the *Optional Programming Assignment 1: Instructions* section under *Week 1*.

3. Start the simulator by running the `launch` command from the command window. If the GUI Layout Toolbox has been installed correctly (instructions for this can be found in the *Preparing for Programming Assignments* page under *Week 1*), the home screen should appear. Pressing the Play button will then show a robot which is stationary (it will move once we design controllers for it next week!).