

iRobot Create Tutorial for CS20

Ethan J. Eldridge

April 11, 2012

Contents

1	Overview	2
2	Installation and Configuration	2
2.1	Locating the Necessary Files	2
2.2	Installation on a MAC	3
3	Verifying the Installation	3
4	Familiarizing yourself with the Simulator	3
5	Configuration Files	6
6	Creating Map Files	7
6.1	Making a map manually	8
6.2	Making a map using the MapMakerGUI	8
7	References	9

List of Figures

1	Click Set Path	2
2	Clicking the add with SubFolders button	2
3	Selecting a toolbox in the browse dialog	3
4	The iRobot Create Simulator	4
5	The Configuration File Creator GUI	7

1 Overview

This tutorial provides a quick and easy reference for using the iRobot Create simulator and robot kit.

2 Installation and Configuration

This section assumes you're using the Windows operating system. For directions to configure the iRobot Create on a Mac, refer to the next section.

2.1 Locating the Necessary Files

To use the iRobot Create you must first install two toolkits. The iRobot Create Matlab Toolkit can be found at

<http://www.usna.edu/Users/weapsys/esposito/roomba.matlab/MatlabToolboxiRobotCreate.zip>

and the iRobot Matlab Simulator can be found at

<http://sourceforge.net/projects/createsim/files/latest/download?source=files>.

Once both of these files have been downloaded, extract the compressed files to your Matlab's toolkit directory. The directory is commonly located at: C:\Program Files\MATLAB\R2011b\toolbox If you receive a permissions error when trying to place files at this location, then you'll have to unzip the files to your desktop, navigate to the above path directory in explorer, and then drag and drop the folders to the directory, possibly saying yes to any administrative control warnings. Once both of these folders are placed in the directory start Matlab.

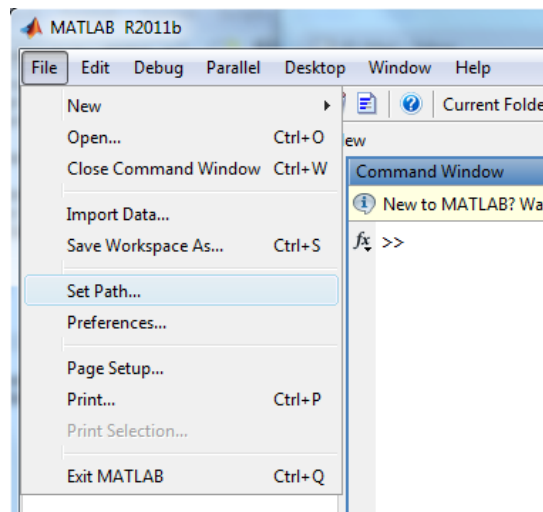
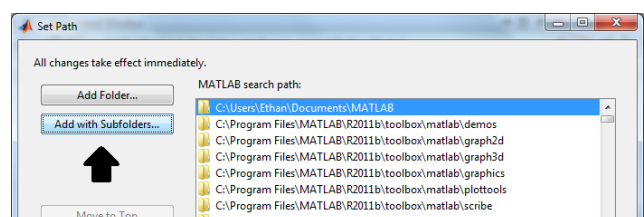


Figure 1: Click Set Path

Next, click file and select Set Path... this is shown in figure 1. A new window will open. This set Path dialog window controls which folders are in Matlab's search path which is used to locate files efficiently. Click the button to add a folder with subfolders, pictorially indicated in figure 2.

Once you've clicked this button, simply navigate to the toolbox directory and add the two folders you downloaded and placed there! Figure 3 shows

2



one of these folders being selected. Click Ok on all the dialogs and then click save on the set path window. This may prompt an administrative ok, just say yes and wait for MATLAB to finish up.

Now that you're done with that, return to the command window and move on to the next section.

2.2 Installation on a MAC

The installation of the toolboxes on a Macintosh is similar to the the windows installation. First, open safari or a similar web browser and navigate to the two pages:

- <http://www.usna.edu/Users/weapsys/esposito/roomba.matlab/MATLABToolboxiRobotCreate.zip>
- <http://sourceforge.net/projects/createsim/files/latest/download?source=files>

This should download two zip archives to your default downloads directory. Navigate to that directory and unzip the two folders to separate folders. Place these folders somewhere you'll remember them. You can place them into MATLABs package, but you won't be able to navigate to them from MATLAB, so place them somewhere safe. Next, start up MATLAB. Next, the instructions are the same as the Windows version, but with different filepaths. These paths are determined by where-ever you placed the files.. Click File and set path as in figure 1. And then click the add with subfolders button indicated in figure 2. Using the browsing dialog that appears, navigate and select each of the two folders (one at a time, opening the add subfolders dialog twice) as in figure 3. Click the save button and you're done.

3 Verifying the Installation

To test your installation type `SimulatorGUI` into the command line and press enter. MATLAB should be busy for a moment and then open the simulator. This large box should look excruciatingly similar to figure 4. Each control on the simulator is well labeled, and decently intuitive.

4 Familiarizing yourself with the Simulator

Once you've followed the instructions in the previous sections, it's time to actually start familiarizing yourself with the controls. Let's start with a brief run down of each of the controls.

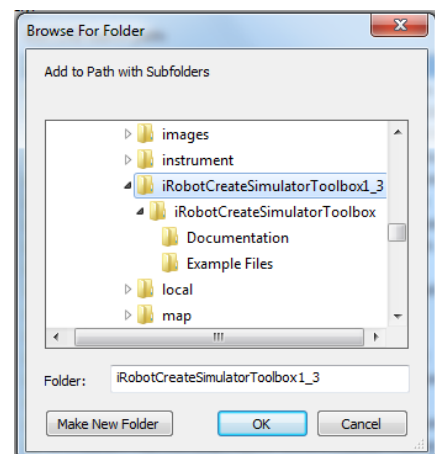


Figure 3: Selecting a toolbox in the browse dialog

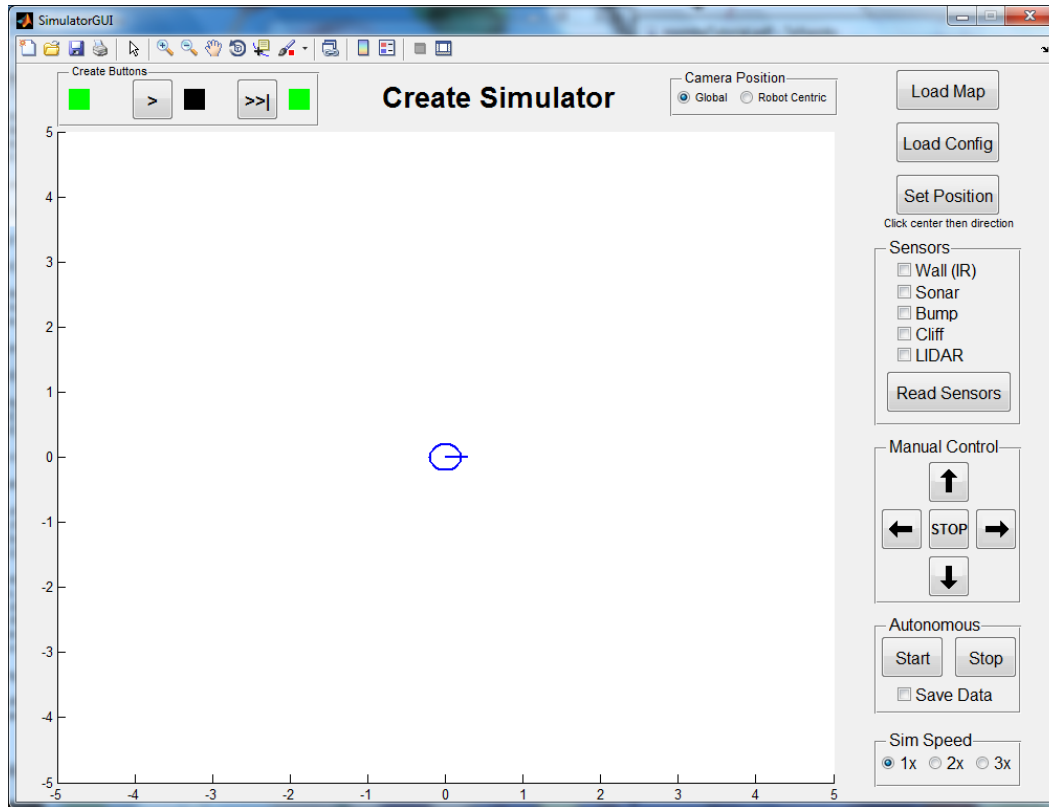
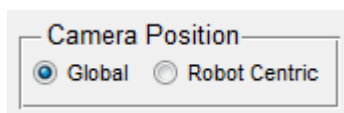
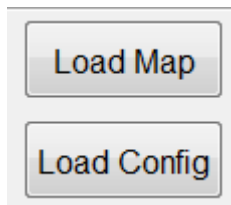


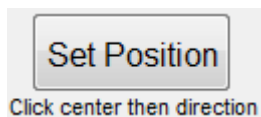
Figure 4: The iRobot Create Simulator



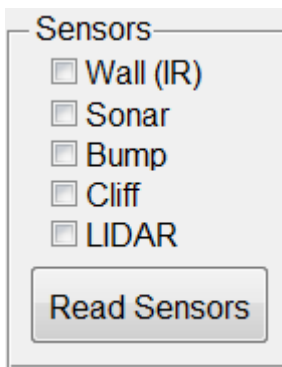
This radio button determines how the camera moves when the robot moves. If it's set to global, the robot will move relative to the screen. Robot Centric will keep the robot in the center of the screen always, and you'll see the numbers along the axes changing as the robot moves. For large maps it may be useful to use Robot Centric, and for smaller maps Global will allow you to see where the robot is in absolute position to the environment better.



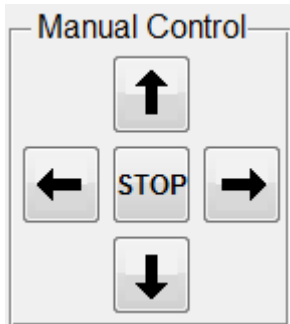
Both of these buttons are self explanatory, clicking either will open a dialog to select a Map or Configuration file for the simulator. Map files are covered in §6, and configuration files are covered in §5. For now, ignore these until we load the example files later on in the tutorial.



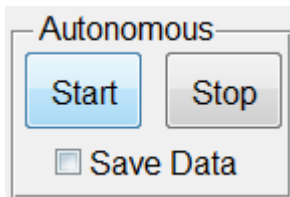
Click this button to place the robot anywhere on the screen you'd like, in whichever direction you'd like. Once you've clicked the button, click on the screen where you'd like to place the robot, then click again in the direction you'd like the robot to face. You can click outside of the white figure space and it will be placed there, as the placement code is not limited to the white figure space.



The check boxes in the sensor control display the robots sensors graphically. This does **not** toggle whether or not the sensors are read by the robot or not. Only if you can see the field of view of each sensor during execution time. Clicking the Read Sensors button will output all sensor data to MATLAB's command window. The only exception being the LIDAR information which is displayed in a figure that is robot centric.

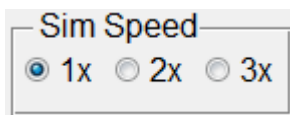


You can control the robot either by pressing the buttons on the manual control panel, or with the keyboard. WASD and the directional arrow keys can control the direction and forward/backwards velocity of the robot. You can stop the robot by pressing the space,enter, or esc key.



These two buttons are going to be your best friends once we start writing autonomous control programs! The Start button opens a dialog to select a control program. Once one has been selected, the program will execute until it completes, or you press the stop button. During this time manual control is disabled. So if your program involves an infinite loop you'll have to close the simulator depending on a few things. If you press the stop button, the next time your code calls a function from the create toolbox, then the code will stop. So if your infinite loop calls a toolbox function, you're fine. If not, you'll have to kill the simulator and possibly MATLAB.

If you check the save data box, then the output data(time history, orientation, information passed between control and simulator programs) will be saved to a file in the current directory. The name of this file will be SimulatorOutputData_*.mat, where the star is some number such that no files will be overwritten.



According to online documentation, it is recommended that this feature **not** be used. As it can cause issues with data output, any calculations that involve tic and toc, as well as more inaccurate physics supplied by the simulator. That being said, this toggle will speed up the simulator itself without changing how the robot perceives himself. In analogy, this would be akin to using a large time step size instead a smaller one, within the course of a second one could get more information using a smaller step size (1x in simulator) or the calculation could be more of an approximation and run faster (3x in simulator).

5 Configuration Files

Creating a configuration file is very simple, you can either type

`ConfigMakerGUI` at the command line or you can open a `.txt` file and use the following conventions. To create one through a text document, use notepad or emacs or an equivalent¹ and type the sensor name, a space, the mean of the noise applied to a sensor, a space, and the standard deviation of the noise.

You can do this for `wall,cliff,odometry,sonar,lidar`, and `camera`. Spelled all lowercase in your file. To include a communications delay, leave a blank line between the last sensor and use `com_delay`, space, the decimal number you would like to delay by. If you use the GUI for creating a configuration file, simply use type in your desired values for each sensor and then click save to export the file. Each sensor is discussed below:

- **Communication Delay:** This parameter specifies the response time for the Roomba. The delay time is in seconds, and increasing will cause delay for all function calls to the Roomba. Note this means all function calls, not just action commands but sensor readings as well.

During manual creation you can specify a value for this like so: `com_delay 0.1` will result in a tenth of a second delay to all functions.

- **Wall Sensor:** This parameter changes the range of the Roomba's wall sensor. The mean of the noise provides how much of an offset from the standard range there will be. The standard range is defined as `rangeIR` in the constants of `CreateRobot.m`. The standard deviation property here gives the variance in the effective range of the affected sensor.

During manual creation you can specify a value for this like so: `wall 0.001 0.0020`

- **Cliff Sensor:** When this sensor has no noise the simulator will read 1.5 when over a line, and 21.5 when not. During physical robot simulations this may not be the case because of lighting, floor color, materials, or any number of outside conditions. Setting a mean that changes the sensor output to act more like the real thing might be handy. The standard deviation adds an amount of variance to the sensors.

During manual creation you can specify a value for this like so `cliff 0.000 0.000`

- **Odometry Sensor:** The odometry sensor noise affects both the angle and distance sensors of the Roomba. During physical simulations if the distance reading is off by an amount (reads 1.1 when the robot has traveled 1 meter for example), then setting the mean noise will signify the offset amount in the reading (0.1 in this example). If the standard deviation of readings from either distance or angle sensors are off, then setting the noise standard deviation should be set to reflect the variance in the sensor data. For example, after turing 6 radians, the sensor reads only .3 radians, then the noise std. dev. should be set to 0.05 to show 5% variation.

During manual creation you can specify the value like so `odometry 0.03 0.005`

¹Do not use Microsoft word or any rich text editor!

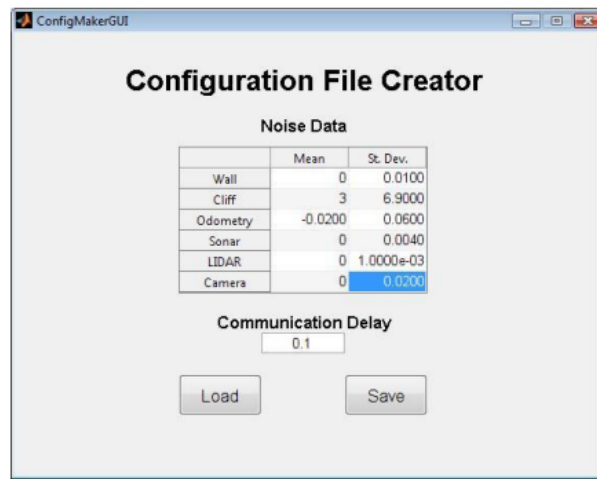


Figure 5: The Configuration File Creator GUI

- **Sonar Sensor:** The noise and standard deviation of the sonar noise specifies the difference of the readings of the sonar from the actual distances. This is the mean and standard deviation of the actual errors in readings, and like the other sensors should be set after experimenting with a physical robot if necessary.

Setting this value in manual creation can be done by the command `sonar 0.000 0.0000`

- **Lidar Sensor:** This sensor's noise functions exactly the same as the Sonar's noise. Specifying the difference of the reading from each point on the lidar range from the real distance.

In Manual configuration this can be set like so `lidar 0.000 0.002`

- **Camera:** The noise mean specifies the difference between the distance readings, and the standard deviation specifies the difference of angle to beacons. This is done by percentage. The camera noise and the odometry noise function very similarly.

In manual configuration files this property can be set like so `camera 0.000 0.1`

Setting the noise to 0 models an ideal robot and should be used during initial coding of control programs for the Roomba in order to fully understand the mechanics of the robot. Once physical robots are considered, noisy data can help perfect control programs intended for real life use.

In manual configuration files, you may include comments using `%` at the beginning of the line. You may also put blanks and spaces anywhere and the ordering of rows do not matter. If you enter a sensor name incorrectly you will receive a warning at the command window upon loading your configuration file. The *only* acceptable sensor names are as follows: wall, cliff, odometry, sonar, lidar, camera.

6 Creating Map Files

To create a map for use in the simulator you can either create one manually or using the graphical user interface through the command `MapMakerGUI` at the command window of matlab. If you make a map manually, you'll need to use a text editor that doesn't

6.1 Making a map manually

Maps can be made in a text file for precise control of positioning map elements. While this is more tedious than using the GUI, a better representation of a real world scenario (such as making sure lengths are correct relative to each other in a maze or obstacle course) is very important for transferability from virtual to real world applications. There are a few formatting rules to follow when creating a map: Comments and blank lines are allowed in the file, any text line that does not start with %,wall,line,beacon, or virtwall will result in a warning from the command window. Each of the map elements are defined below:


- **Walls** are defined by the keyword wall at the beginning of the line followed by cartesian coordinates of the first point, and then the second point. An example is `wall 2.5 0 5.1 2.1`. If a Roomba runs into a wall the bump sensor will go off, walls can be detected by lidar, sonar, and wall sensors.
- **Lines** are defined by the keyword line at the beginning of a text line. Similarly defined like walls, two cartesian points specify all information. Example: `line -1.0 3.4 50.2 2.4`. Lines are picked up by cliff sensor, and change the signal strength. Because the Roomba will only read a line while it's passing over it, setting com delay to be high in configuration can result in these sensors not reporting accurately.
- **Beacons** can only be detected by the camera and are typically colored paint or paper on the ground. When the Roomba goes over it the beacon is detected and matched to it's ID string. The keyword beacon followed by the coordinate of the beacon, the color vector, and the id string are parsed into a map from a file. For more information on color vectors type `help colorspec` or `doc colorspec` at the command line. An example of a beacon would be: `beacon 4.5 2 [1.0 0.0 1.0] beacon1`
- **Virtual Walls** are infrared light emitters that can range between a power output of one to three. To define the wall the keyword virtwall followed by the coordinate of the emitter, the angle (in radians) of the direction of the light relative to the positive x-axis, and the output power (1 to 3) must be specified. Example: `virtwall 1.0 1.0 -2.45 2`


Once your map file is created you can load it using the load map button in the simulator. If there are any errors in your map warnings will be displayed on the command line. Typically, maps are on a grid from -5 to +5 in both x and y directions, however this is not a limit on the size of a map. Maps can be any size but be aware that depending on the camera settings for the simulation your Roomba may wander out of the viewing area. You can use the standard figure controls to move the map around during the simulation as well as in the MapMakerGUI.


6.2 Making a map using the MapMakerGUI

Using the GUI included in the simulator is very intuitive. The directions are clearly labeled in the top right of the window that appears after executing the `MapMakerGUI` command. Some important notes are that while you can use ctrl-z (apple or command-z on Apple Macs) or the undo button to reverse changes you have made, a load, clear, or save cannot be undone. Also, there is no redo button so any changes reversed are permanent. Another thing to keep in mind is that buttons for both the beacon and virtual wall will use the settings to their right when you click the button to place the object. You must change these settings to your desired settings before clicking the button!

Below is a brief overview of what each button in the simulator does:

-  The **Shape** controls modify the type of object you can create using the line and wall tools. Clicking the straight line button will allow you to place straight segments of Wall or Line. The middle button, continuous line segments allow for you to click each point you'd like to connect with a line, one after another. To exit this mode simply press any button on the keyboard. The rightmost button, the Square enables one to create square Walls or Lines of any size by clicking the two opposing diagonal corners.²

-  The **beacon** button should only be clicked when the settings to the buttons right are selected. You can choose a color for the beacon as well give a unique label for it from the settings of the button. Once the button is clicked you can place the beacon down by clicking on the map where you want it to be. If you wish to place multiple buttons at once, press the button once and then press the space bar for each additional beacon you want to place down. Once the button is clicked, you must put down a beacon. If you mistakenly place a beacon down simply undo the last action.

-  A **virtual wall** is used in much the same way a beacon is. The button should be pressed after the settings are selected, and repeatedly pressing the space bar will enable you to place a large number of virtual walls at once. Once you've selected the range setting from the 3 options, click once where you'd like the object to be placed. Then click in the direction you'd like the object to project its virtual wall.
- The **undo**, **clear**, **load**, and **save** buttons are all self explanatory. But it is worth mentioning again that any of these buttons being pressed cannot be undone (including an undo- there is no redo button).

7 References

Information in this tutorial was collected through the use of google and self testing. Notable websites include

http://www.irobot.com/filelibrary/create/create%20manual_final.pdf

<http://web.mae.cornell.edu/hadaskg/CreateMATLABsimulator/User%20Guide.pdf>

<http://web.mae.cornell.edu/hadaskg/CreateMATLABsimulator/Code%20Documentation.pdf>

http://www.usna.edu/Users/weapsys/esposito/roomba.matlab/Matlab_Toolbox_iRobot_create_doc.pdf

All images were screen-captured by the author. You'll definitely want the MATLAB Toolbox for the iRobot Create available at

²This can be either the top left and bottom right corners, or the top right and bottom left corners. It doesn't matter which.