

# The PICRoboSim Handbook

Christopher Merck  
Department of Physics & Engineering Physics,  
Stevens Institute of Technology, Hoboken, New Jersey 07030, USA

September 29, 2007

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	What is PICRoboSim? . . . . .	2
1.2	Features at a Glance . . . . .	2
1.3	System Requirements . . . . .	2
1.4	Author Contact . . . . .	2
<b>2</b>	<b>Tutorials</b>	<b>3</b>
2.1	Tutorial 1: Getting Started . . . . .	3
2.1.1	Download PICRoboSim . . . . .	3
2.1.2	Open and Run in Visual Studio . . . . .	3
2.1.3	Driving Around . . . . .	3
2.2	Tutorial 2: Driving by Instruments . . . . .	6
2.2.1	The Sensor Suite . . . . .	6
2.2.2	Trusting the Sensors . . . . .	6
2.3	Tutorial 3: Automating the Robot . . . . .	7
2.3.1	A First Program . . . . .	7
2.3.2	Complex Movement . . . . .	7
2.3.3	Bumper Program . . . . .	8
2.3.4	Light-seeking Program . . . . .	8
2.3.5	Floor Sensor Module (FSM) . . . . .	9
2.3.6	Navigation Lamp/Sensor . . . . .	9
2.4	Tutorial 4: Writing a Successful Robot Control Program . . . . .	10
2.4.1	Modifying the Simulated Robot . . . . .	10
2.4.2	Adding Walls and Home Lights to the Arena . . . . .	10
2.4.3	Writing a Complete Program . . . . .	10
2.5	Tutorial 5: Using the Bumper Interrupt . . . . .	12
2.5.1	Using Global Variables in the Interrupt Function . . . . .	12
2.6	Tutorial 6: Using your Program in Design Lab . . . . .	14
<b>3</b>	<b>Reference</b>	<b>15</b>
3.1	Keybindings . . . . .	15
3.2	Built-in Functions . . . . .	15
3.2.1	High-level Functions (found in <i>STUDENT_Functions.h</i> ) . . . . .	15
3.2.2	Low-level Functions (found in <i>PICinit.h</i> ) . . . . .	16
3.3	Troubleshooting Errors . . . . .	17
3.3.1	Compile-time Errors . . . . .	17
3.3.2	Run-time Errors . . . . .	17

# Chapter 1

## Introduction

### 1.1 What is PICRoboSim?

PICRoboSim is a software package which provides a simulation of the Engineering Design I robot project environment. It allows students to learn to write robot control programs and demonstrates the simulated robot's behavior on-screen (see Figure 2.4). Additionally, C code written to control the simulated robot runs on the PIC-based robots used in Design I without modification.

### 1.2 Features at a Glance

- Qualitative and quantitative simulation of robot performance
- Manual control mode to help programmers develop their algorithms
- Near-100% code compatibility with c2cpp-based robot code used in Design I
- Allows customization of virtual robot to match student's actual design
- High portability through cross-platform libraries

### 1.3 System Requirements

Laptops provided to Stevens students run the simulator flawlessly. However, if you decide to run PICRoboSim on a different machine it is strongly recommended that it meet the following requirements:

**processor** 1*Ghz* dual-core or hyper-threading CPU

**memory** 512MB

**OS** Windows 2000/XP (Vista untested)

**compiler** Microsoft Visual Studio 2005

NOTE: While the current version of PICRoboSim only runs on Win32 it can be easily ported to UNIX systems because of use of standard C and the cross-platform SDL libraries.

### 1.4 Author Contact

Please direct any questions or suggestions regarding the simulator program to Chris Merck, reachable via email as [cmerck@stevens.edu](mailto:cmerck@stevens.edu).

# Chapter 2

## Tutorials

### 2.1 Tutorial 1: Getting Started

#### 2.1.1 Download PICRoboSim

To use PICRoboSim it must first be downloaded to your computer. The latest version of PICRoboSim may be downloaded at <http://personal.stevens.edu/~cmerck/picrobosim/download/>. Click on `PICRoboSim_1.0.0.exe` (or the most recent version) and save it. Then run it by double clicking the file (see Figure 2.1.1). Click unzip to complete the installation. It is highly recommended that you install to the default directory of `C:\PICRoboSim`.

#### 2.1.2 Open and Run in Visual Studio

In order to run the simulator, you need to open it in Visual Studio. Double click *My Computer*, browse to `C:\PICRoboSim\`, and double-click *PICRoboSim.sln* as shown in Figure 2.2.

Microsoft Visual Studio (MSVC) will load, displaying the source code of the simulator (see Figure 2.3). Don't worry if it looks complex, you are not responsible for learning the details of the simulator, just using it.

Once MSVC is loaded, press *F5* to run the simulator for the first time. You should see a screen similar to Figure 2.4. If you get any errors, make sure you double-clicked *PICRoboSim.sln* and not one of the other files in the directory.

#### 2.1.3 Driving Around

Get a feel for the environment by driving the virtual robot around with the arrow keys. Try steering the robot onto the away side of the arena and hitting both target lights. You win if you manage to knock both out. Congratulations! You have successfully set-up PICRoboSim on your system.

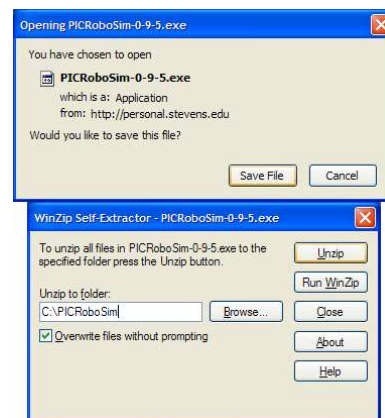


Figure 2.1: Downloading the software

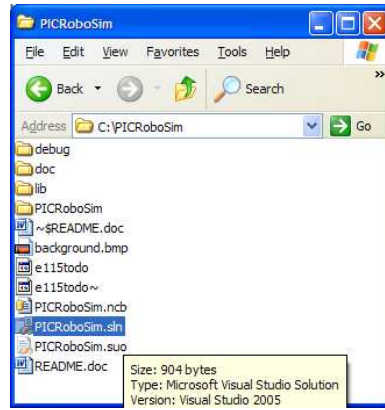


Figure 2.2: Browsing and opening the solution file

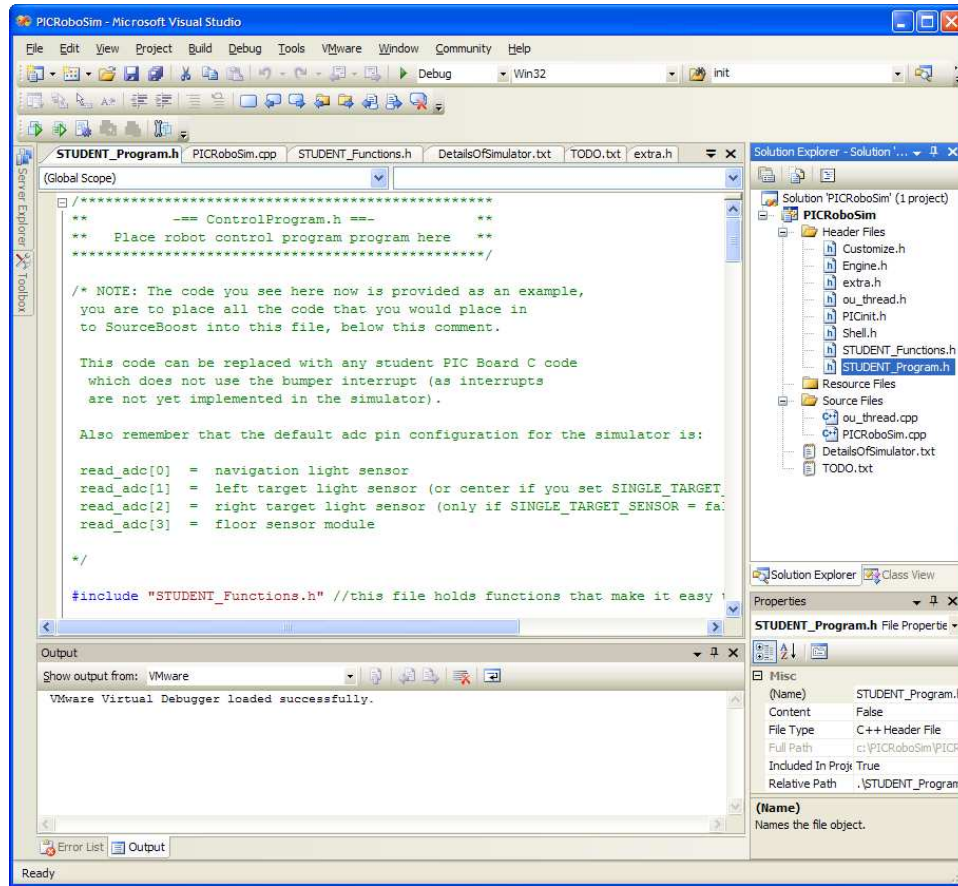


Figure 2.3: Microsoft Visual Studio (MSVC) with the simulator project opened and ready to run.

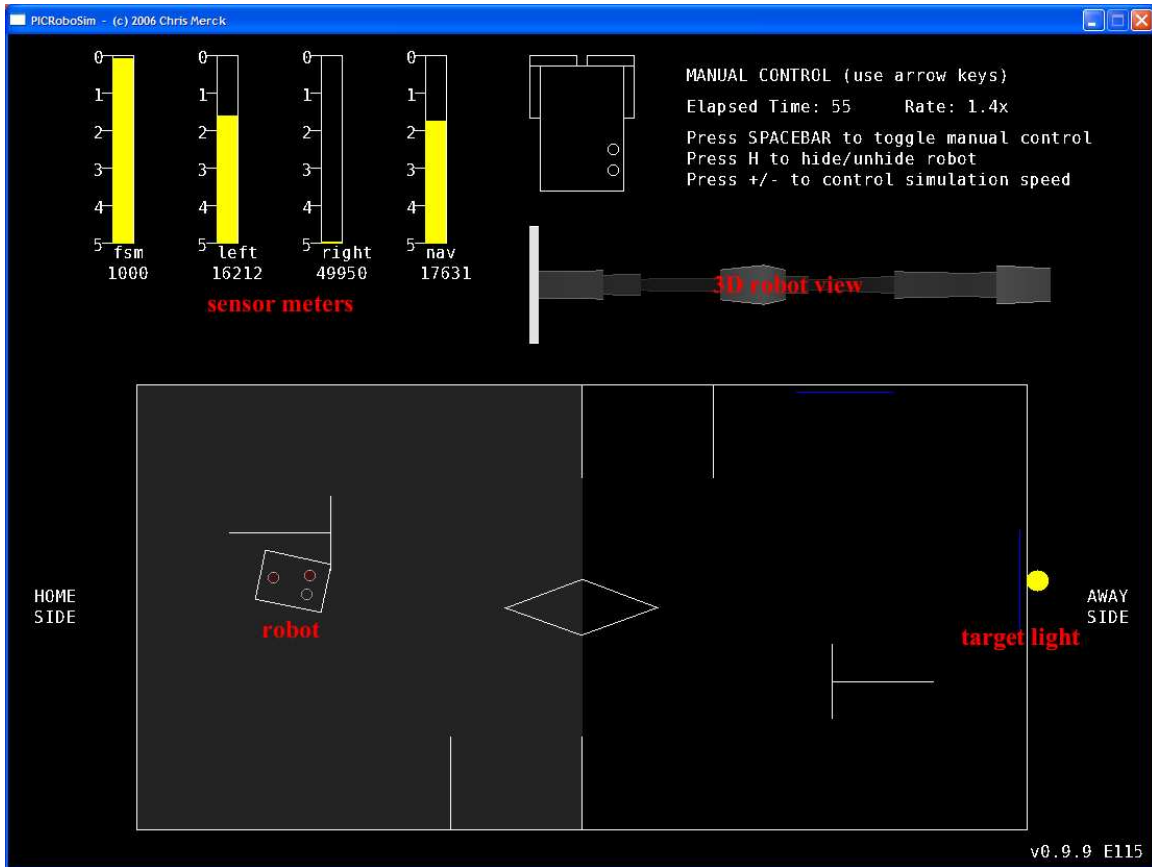


Figure 2.4: Simulator running under manual control

## 2.2 Tutorial 2: Driving by Instruments

### 2.2.1 The Sensor Suite

In Engineering Design I you will build a robot based around a common 2-wheeled chassis, a PIC micro-controller printed circuit board (PIC board), and a suite of bumper and light sensors. Your ultimate goal is to program the robot to eliminate two target lights in an arena in the shortest time possible as part of an end-of-semester competition. While you are free to build your Design robot in any way you choose, the robot in the simulator has a default sensor suite that should satisfy you (to customize the virtual robot see Section 2.4.3).

Run the simulator again. This time pay attention to the sensor indicators on the top of your screen. On the top-left are meters showing the state of the four light sensors on the simulated robot. On the far-left is the Floor Sensor Module (FSM) which reports the brightness of the floor, the second and third meters are the robot's two forward-looking light sensors (named left and right), and last is an upwards-looking light sensor to detect the navigation light on the ceiling. Drive near an activated light (yellow circle) and see how the light sensors respond. Also, below each light sensor is the actual numerical value reported by the sensor (the more light, the lower the value).

In addition to the light sensors your robot has two front bumpers. In the top-center of the screen there is a diagram of the robot with bumper indicators that turn red when a bumper is hit. On that same diagram are two LED indicators that you can use to help you write your program, but more on that later.

### 2.2.2 Trusting the Sensors

It is easy to eliminate both away target lights using the arrow keys when you can see the robot on the screen. However, when you write a program to automate the robot, you will have to trust the sensors. Press 'h' to hide the robot. Now try eliminating both target lights using only the sensors.

At first it may be hard to drive the robot without seeing it, but the sensors give you all the information you need. If you have trouble try these tips:

- You can find a target light by pivoting in one direction until both the left and right light sensors see about the same amount of light.
- If you see one of the bumper indicators (top-center) go red it means that bumper is hitting a wall. When that happens try moving in reverse for a moment and then pivoting in the opposite direction of the bumper that got hit. You should clear the obstacle.
- If you get frustrated, un-hide the robot by pressing 'h' again and reorient yourself.

If you can win at this little game with the robot hidden then you know an algorithm for winning, and should not have trouble programming the robot to succeed on its own.

## 2.3 Tutorial 3: Automating the Robot

Now that you are familiar with the simulation environment you can begin programming the robot to move on its own, using a program written in the C language.

### 2.3.1 A First Program

To write your own robot control program you need to edit the *STUDENT\_Program.h* file in the simulator. The program in this file is run when you activate autonomous mode. If the simulator is running press 'q' to exit and return to Visual Studio. On the right you should see the 'Solution Explorer' window with a listing of files. In this window browse PICRoboSim > Header Files > STUDENT\_Program.h and double-click that file. You should see the following code:

```
/*
*****
**          == STUDENT_Program.h ==
**   Write your robot control program here   **
*****
#include "STUDENT_Functions.h" //high-level functions
int main() {
    configurePIC(); //this function starts up the PIC micro-controller
    //ADD YOUR PROGRAM BETWEEN HERE...

    //... AND HERE!
    return 0;
}
```

Let's write a very simple program for your robot. In-between the comments telling you where to insert your program type `gofwd()`; . This function turns both motors on forward and leaves them on. Next you will run the program. To see the effect of your program press F5 to bring up the simulator and then hit the space bar to allow your program to take over control of the robot. The robot should move forward and hit a wall after a second or two. Read on to learn how to teach the robot to detect when it is hitting a wall.

### 2.3.2 Complex Movement

You will need to do more than move forward to accomplish your goal of knocking out the target lights. We will now program the robot to perform a complex motion: turning in a circle. We will replace your old one-line program with another short program that makes the robot repeatedly move forward and then turn to the right, roughly tracing out a circular path. You can use the following code or make your own version:

```
while (true) {
    gofwd();
    pause(200);
    pivotright();
    pause(200);
}
```

Two new functions are used in this program. First, there is `pause(200)` which waits for 200 milliseconds before executing the next command. The other new function, `pivotright()`, turns the left motor on forwards and sets the right motor to reverse resulting in a tight pivot about the center of the axle. These functions are wrapped in a `while` loop which will loop forever, making the robot move in a circle. Press F5 to run the simulator and see the effect of this new program. Try writing your own variations on the above code to make the robot do the following:

- do circles in reverse
- move in a zig-zag pattern

You will need to look up other movement functions to complete these tasks. For a list of all the functions available to you to control the robot check out Section 3.2.1 on high-level functions.

### 2.3.3 Bumper Program

To avoid hitting walls you will need to program the robot to check the bumper sensors. The virtual robot in PICRoboSim is equipped with two front bumpers. The state of these bumpers can be read using the `leftbumper()` and `rightbumper()` functions. They return true when the bumper is depressed and false otherwise.

The following code example is a basic bumper program and can be placed in the same place as the program from the previous section.

```

gofwd();                                //start out going forwards

while (1) {

    if (leftbumper() || rightbumper()) { //check for bumping either bumper
        gorev();                          //take evasive action
        pause(500);
        pivotright();
        pause(100);
        gofwd();
    }
}

```

Once you have placed this code into the `STUDENT_Program.h` file, press F5 to run it. Notice how the robot can now move around obstacles by responding to bumper presses.

The text to the right of the code (after the `//` characters) are comments which should help you understand how this program works. The one line which may require additional explanation is the one with the if-statement. The conditional expression of the if, contained in parenthesis, consists of tests for both the left and right bumpers combined with the or-operator (written '—'). The result is that when either bumper is hit, the code in the following block (between '{' and '}') is run, causing the robot to back up and turn before continuing forwards.

Try modifying the above program to turn to the right after hitting the right bumper, and turn left after hitting the left bumper. Notice how this improves or deteriorates the performance of the robot.

### 2.3.4 Light-seeking Program

To hone in on the target lights your robot needs to make use of its front target sensors. You have a left and a right target sensor at your disposal. By comparing the brightness of these sensors you can create an effective program to find the lights. Try the following program. You may need to help the robot out in manual mode if it gets stuck, or improve the code to add bumper checking.

```

while (1) {
    if (read_adc(1) > 30000 && read_adc(2) > 30000) //if it is dark, don't turn.
        gofwd();
    else if (read_adc(1) < read_adc(2)) //if not so dark, check for brighter direction
        arclleft(); // and arc that way.
    else
        arcrright();
}

```

This program uses the motor control functions `arclleft()` and `arcrightright()` which turn both motors on forwards, but at different speeds, introducing an arc to the robot's motion. Under this program, the robot will make alternating arcs until it strikes a light.

### 2.3.5 Floor Sensor Module (FSM)

The simulated robot is equipped with a Floor Sensor Module, or FSM, which detects the brightness of the floor beneath the robot. This is useful in determining whether the robot is in home or away territory. The FSM is wired to analog channel 3, so it can be read with the function `read_adc(3)`. Low FSM values correspond to the light side of the arena, and high values to the dark side. As an example of programming for the FSM, try the following code which makes the robot straddle the FSM. To run it, drive the robot near the light/dark boarder in manual mode and then enable the program.

```
while (1) {
  if (read_adc(3) > 25000) //compares FSM with a threshold value
    gorev();
  else
    gofwd();
}
```

While this program is a good demonstration of reading the FSM, it does little for your robot in the long-run. You can program your robot to turn about if it crosses back into home territory, keeping it on the away side where its targets are. This can be done using a variable to hold the home color and looking at the difference of the current color and the home color. When testing your FSM code make sure you try starting your robot on both dark and light colors. You can do this in the simulator by enabling the `SWAP_FLOOR_COLOR` option in *Customize.h*. Do this by double-clicking that file in the Solution Explorer and changing the `SWAP_FLOOR_COLOR` option to read `#define SWAP_FLOOR_COLOR true` (instead of false as before) and run the simulator.

### 2.3.6 Navigation Lamp/Sensor

Suspended one meter above the center of the arena is a light bulb called the navigation lamp. It can be used to help your robot determine where it is in the arena. The simulated robot has a navigation sensor looking up and forward at 45 degrees which is on analog channel zero. The details of the programming is up to you.

## 2.4 Tutorial 4: Writing a Successful Robot Control Program

The previous tutorial taught you all how to control the robot's motors and get input from various sensors. Now it is time to customize your robot and write a program which can complete the task of quickly and reliably knocking out the target lights. Doing this can succeed in the Engineering Design I competition at the end of the semester. While your team is still building your physical robot you can test out your program in this simulator.

### 2.4.1 Modifying the Simulated Robot

The simulated robot by default has two front light sensors and two front bumpers. However, your robot for Design lab may have a different configuration. If you wish, you can customize the simulated robot to more closely emulate your physical one.

To modify the position of light sensors, start up the simulator and enter the configuration screen by pressing 'c'. You will see a screen like in Figure 2.5. The rectangle in the center of the screen represents your robot. Using the mouse, click and drag the centers of the light sensors (red circles) around the robot. Then adjust the viewing angle of the sensors by clicking and dragging the edge of the sensor. Lastly, you can change the field of view of the sensor by pressing the UP and DOWN keys while the mouse is hovering over the sensor you wish to change. When you are satisfied with the configuration, press 's' to save and restart the simulator with the new configuration. If you don't want to apply changes, press 'q' to return to the simulation. Default settings may be restored by pressing 'd' at any time.

For more fine-tuned control over the simulation, you can set appropriate options in `Configuration.h`. You can make modifications such as adding a rear bumper, adding noise to the sensor readings, and change the physical dimensions of the robot.

By configuring the simulator in this way you can achieve a high accuracy in the simulation and reduce the number of changes you will need to make to your program to run it on the real robot. Read on to learn how to run your simulated program on the real robot.

### 2.4.2 Adding Walls and Home Lights to the Arena

To start, let's make the simulator look more like the real arena. First, turn walls on in the `Configuration.h` file. Do this by double-clicking that file in the Solution Explorer and changing the `TOGGLE_WALLS` option to read `#define TOGGLE_WALLS true` (instead of false as before). Now when you run the simulator you see walls in the arena - in the same places they are on the actual competition boards in the Design lab.

Also, in the real arena there are lights on the home side that you are not supposed to knock out. You can enable these by setting the `HOME_LIGHTS` option in `Configuration.h`, just as you did for adding walls. Try adding code to your program to avoid the home lights when in home territory!

### 2.4.3 Writing a Complete Program

To be succeed in knocking out the target lights the robot should take advantage of all the sensory information available to it. Load up the bumper program from the last tutorial and run it. Notice how it has difficulty handling the walls you just added. Think about ways of improving the way the robot responds to its bumpers being hit that would allow it to move around more freely.

When your bumper program works, try adding light-seeking code (like that in Section 2.3.4) without removing the bumper code. If done correctly, this makes the robot seek out the target lights and not get stuck on obstacles in the way.

Lastly, take advantage of the navigation and FSM sensors. Above all, be creative! The robot is only as clever as the programmer.

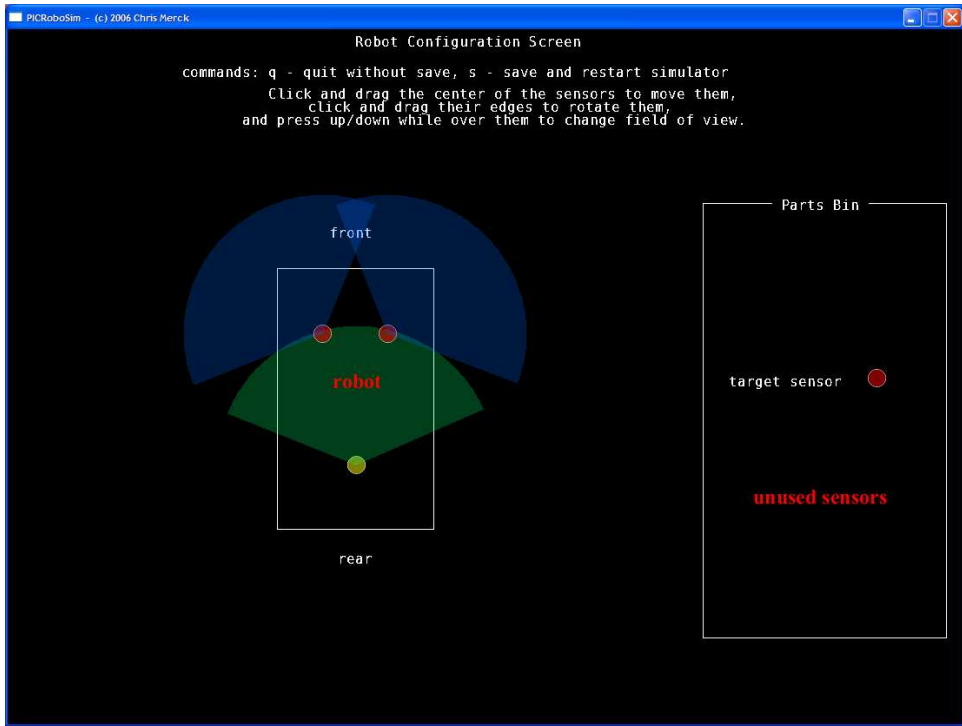


Figure 2.5: robot configuration screen

## 2.5 Tutorial 5: Using the Bumper Interrupt

The method for getting bumper input presented in Section 2.3.3 is called "polling". This means that you test the state of some input within your main loop. While this method is simple to use, some programmers prefer to use an "interrupt"-based input scheme instead. The way the interrupt works is that a special feature of the PIC watches the state of the bumper input pin, and when it goes high (that is, when the bumper is hit) it suspends your main program and runs a special `interrupt()` function. When the `interrupt()` function returns the main program resumes from where it left off. As a result, the main program does not need to check for bumpers, which can be very convenient depending on your programming style.

The bumper interrupt is used in PICRoboSim just like it is in SourceBoost for Design I: somewhere in your source file, but outside your `main()` function, you must declare a `void interrupt()` function, as well as use several other special functions, as described on WebCT under Robot Design Project > Constructing the Bumper Interrupt Support Board. Here is an example student main program using an interrupt which results in the same robot behavior as the one given for polling-based bumper control in Section 2.3.3:

```
int main() {
    configurePIC(); //this function starts up the PIC microcontroller

    enable_interrupt (INTE);    // Enable the External Interrupt Line
    enable_interrupt (GIE);    // Enable Global Interrupt

    gofwd();

    //todo: place more code here to control your robot!
    return 0;
}

void interrupt ( )            // This function will be automatically called upon an interrupt
{
    save_data ( ); //save environment now to enable restoration later

    //evade walls using bumper data
    if (leftbumper() || rightbumper()) { //check for bumping either bumper
        gorev(); //take evasive action
        pause(500);
        pivotright();
        pause(100);
    }

    clear_bit (INTCON, INTF); // Clear interrupt flag bit
    restore_data ( ); //restore system status to that before the interrupt occurred
    return;
}
```

### 2.5.1 Using Global Variables in the Interrupt Function

If you wish to use global variables between your main and interrupt function, you will need to place the definitions of these variables in a special file due to technical restrictions of the simulator. Instead of placing the definition for some global variable at the top of the `STUDENT_Program.h` file, place the definition in the `STUDENT_GlobalVariables.h` file. Note that you ONLY need to place this variable here

if you need to use it in both your main program AND in the `void interrupt()` function. Also, be sure NOT to initialize the variable when you declare it in `STUDENT_GlobalVariables.h`; instead, initialize it at the top of your `main()` function (this avoids a rather cryptic compile-time error).

## 2.6 Tutorial 6: Using your Program in Design Lab

When you have built your robot in Design Lab you can use the program you wrote in this simulator to control the robot. Before you can use the code you wrote in the simulator in SourceBoost you will need to copy over the motor control and bumper checking functions you have been using. To do this:

1. First open PICRoboSim to the *STUDENT\_Functions.h* file (use the Solution Explorer window on the right side of Visual Studio to select the file). This contains the high-level functions for motor control, bumper checking, etc. that you have used in the simulator. Select all the text in the file (CTRL+A) and copy it to the clipboard (CTRL+C).
2. Open SourceBoost IDE to your main file.
3. Paste the functions into SourceBoost (CTRL+V).

Now that you have the functions in SourceBoost, you need only copy your main program over from the simulator:

1. Return to PICRoboSim in Visual Studio, this time select the *STUDENT\_Program.h* file from the Solution Explorer.
2. Select all the text in the file (CTRL+A) and copy it to the clipboard (CTRL+C).
3. Open SourceBoost IDE to your main file and paste your program into SourceBoost *after* the functions you pasted in earlier.
4. Compile and try your code. Remember that there are some small differences between how code runs in the simulator and on the real robot, so you may have to make some changes to the code to get it to work how you want it to.

# Chapter 3

## Reference

### 3.1 Keybindings

The user can control aspects of the simulation using the following single-key commands (responses are given in the terminal window):

**space bar** toggles manual / automatic control of robot

**arrow keys** move robot in manual mode

**'h'** toggle robot visibility

**'r'** reset simulation

**'q'** quit

**'+'** increase timestep (cruder simulation)

**'-'** decrease timestep (finer simulation)

**'i'** skip more frames in video display (faster display)

**'j'** skip less frames in video display (smoother display)

### 3.2 Built-in Functions

There are two kinds of functions that come with the simulator: high-level functions that define conceptual things like motor control and bumper checking, and there are low-level functions that allow the high-level parts to communicate with the robots electronics.

#### 3.2.1 High-level Functions (found in *STUDENT\_Functions.h*)

**void gofwd();** both motors on forwards

**void gorev();** both motors on reverse

**void pivotleft();** pivots to the left in-place

**void pivotright();** pivots to the right in-place

**void arcleft();** performs a gradual turn to the left

**void arcright();** performs a gradual turn to the right

**void motorsoff();** stops the robot

**void yellow(int state);** turn on/off yellow LED (1 = on, 0 = off)

**void red(int state);** same for red LED

**bool leftbumper();** returns true if left bumper is depressed

**bool rightbumper();** returns true if right bumper is depressed

### 3.2.2 Low-level Functions (found in *PICinit.h*)

**void configurePIC();** configure 16F877A PIC to E121 course configuration

**void motorspeed (char motornum, char speed);** sets the motor speed from 50 100% duty cycle

**void putdata (int data);** sends over RS232 the ASCII equivalent of a number in the range of 0 – 65535

**void putchar (char data);** sends over RS232 a single ASCII character

**int read\_adc (char channel);** returns analog voltage in the range 00000-50000 (0-5V)

**char read\_input (char port, char Bit);** returns digital logic level of an input port/pin

**void output\_float(char port, char Bit);** configures a port/pin to be an input

**void output\_high (char port, char Bit);** sets specified port-bit to be an output, then sets it high

**void output\_low (char port, char Bit);** sets specified port-bit to be an output, then sets it low

**void pause (int time);** Pause in milliseconds (Input range 1-65025)

**void save\_data ();** Saves function data so interrupt doesn't overwrite

**void restore\_data ();** Restores data after interrupt

## 3.3 Troubleshooting Errors

There are two kinds of errors you may encounter while programming with PICRoboSim. There are compile-time errors, and run-time errors.

### 3.3.1 Compile-time Errors

Compile-time errors are those that occur when you press **F5** and the program fails to start. There is no pop-up window for a compile-time error, instead, you will see an Error List in the lower-most window of the Visual Studio window. If there are any warnings, indicated with a yellow sign with an exclamation point, these may be ignored. However, any errors, indicated by a red circle with an X in it, must be fixed before the program can run. To fix an error, double-click on its line in the Error List. Your cursor will be brought to the line near the error in the program. First check to see if the error was a syntax error (a typo) by looking near and especially above the line the Error List sent you to. Otherwise, you will have to read the error message carefully to decide what to do. Here is a list of common errors and their solutions:

#### 'BrainThread::X' : only static const integral data members can be initialized within a class

You tried to initialize a global variable when it was declared. To fix this error, declare, but do not initialize the global variable outside of your `main()` function. Instead, initialize the variable within your main function.

For example, instead of:

```
int this_is_a_global_variable = 0;
int main() {
    //main program...
}
```

Do this:

```
int this_is_a_global_variable;
int main() {
    this_is_a_global_variable = 0;
    //main program...
}
```

### 3.3.2 Run-time Errors

More difficult to diagnose and fix are run-time errors. A run-time error is an error which occurs while the simulator is running, and normally result in a pop-up box from Visual Studio and the simulator is forced to close. These errors typically occur when using pointers or arrays, and you try to access memory that has not been declared. After getting a runtime error, double check any arrays or pointers you may be using. (Remember that an array declared as `int MyArray[3]` has only *three* elements, `MyArray[0]`, `MyArray[1]`, `MyArray[2]`, not `MyArray[3]`.) Also, try un-doing recent edits by pressing Control-Z multiple times, or using `EDIT`; `UNDO` in Visual Studio, and recompile to see if the error was fixed.

If the error persists, there may be a bug in the simulator. Try copy and pasting your program to a backup file somewhere on your machine and reinstalling the simulator. If you still receive the error, email [cmerck@stevens.edu](mailto:cmerck@stevens.edu) for assistance. Include a full description of the error, the contents of your `STUDENT_Main.h` file, and a screenshot of the error message.