WSU's Khepera Robot Simulator

Manual for Version 6.4

Steve Perretta
sperrett@cs.wright.edu

Licensing Agreement

By obtaining this software, you agree to the following terms of use:

1. This license agreement applies to the KS software, the KS manual, and all files accompanying the distribution that have a copyright notice. Below, "KS" refers to any and all files associated with the WSU Khepera Simulator. "A work based on KS" refers to any work that contains any portion of KS, with or without modifications. "You" refers to the licensee.

2. You may copy and distribute verbatim copies of KS in any medium, provided that you retain all copyright notices, disclaimer of warranty, notices that refer to the license, and the licensing agreement. You may not solicit or accept payment for distributing KS. You may, however, charge the fair cost of the media you used to deliver it.

3. You are free to modify KS for your own personal use. You may not distribute your modified versions of KS. You may distribute modifications as separate files along with the standard KS distribution. For example, you may distribute a shell script or a patch file that modifies the standard KS code. You may not distribute the modified code itself.

4. You may not copy, modify, sublicense, distribute or transfer KS except as expressly provided under this license. Any attempt to otherwise copy, modify, sub-license, distribute or transfer KS is void, and will automatically terminate your rights to use KS.

5. By obtaining, distributing or modifying KS or a work based on KS, you indicate your acceptance of this license and all its terms and conditions.

6. Each time you redistribute KS or a work based on KS, the recipient automatically receives this license from the original licensor to copy, distribute or modify KS subject to terms and conditions within. You may not impose any further restrictions on the recipient's exercise of the rights granted herein.

7. KS is licensed free of charge. To the extent permitted by applicable law, there is no warranty on KS. The copyright holder and/or other parties provide KS "as is" without warranty of any kind, either express or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of KS is with you. Should the program prove defective, you assume the cost of all servicing, repair or correction.

8. In no event will any copyright holder, or any other party who may redistribute KS as permitted above, be liable to you for damages, including any general, special, incidental or consequential damages arising out of the use or inability to use KS (including but not limited to loss of data or data being rendered inaccurate or losses sustained by you or third parties or a failure of KS to operate with any other programs), even if such holder or other party has been advised of the possibility of such damages.

9. Incorporation of KS or portions of it in commercial programs require a separate agreement between the copyright holder and the Licensee in writing. Absent such an agreement, you may not use KS in any commercial product or use it in any for profit endeavor.

10. If you publish any academic paper, book, treatise or other work based upon experiments conducted using KS, you must include a suitable citation or acknowledgement of KS and its author.

TABLE OF CONTENTS

1. REQUIREMENTS

1.1 Java Version

This simulator was developed using jdk1.3.1, and makes heavy use of Swing components. You should be able to use any SDK that supports the Java 2 Platform. Alternatively, if you want to use JDK1.1, you can download and add the current Swing extension. Since I have not tested the simulator with anything but 1.3.*, it is recommended that you use 1.3.* or the current 1.4.*. Here is a listing various Java related internet resources that you may find useful:

You will find a listing of JDK packages at:

- http://java.sun.com/products/
- http://www.ibm.com/java/jdk/download/

You will find documentation on various JDKs (including installation help) at:

- http://developer.java.sun.com/developer/infodocs/

If you are unfamiliar with the Java language, on-line tutorials can be found at:

- http://java.sun.com/docs/books/tutorial/

If your platform is not supported by the products at Sun's site try these:

FreeBSD: http://www.freebsd.org
- use the local search engine here, using key words "java ports".

Linux: http://www.blackdown.org/
- although Sun supports Linux java packages, there are some alternatives here.

There are many different java ports around the web. If you can't find one using the preceding links, or you just want to see what else is out there, here is a possible starting point:

- http://home.intranet.org/~mouse/links/java.html

1.2 Operating System

This program has been extensively tested on most "popular" operating systems. These include Mac OS X, Linux, Solaris Sparc, Windows98, and WindowNT. Under each of these systems no performance problems were detected, although your results may vary depending on the underlying hardware. The program will run under Windows2000 and WindowsXP, but with varying performance problems possible. The observed behavior of the program on these systems is hardware dependent as well. Further developments of the simulator are currently underway to optimize it's performance on newer Microsoft operating systems.

2. INSTALLATION

2.1 Uncompressing Files

WSU_Sim_v6.4.zip

Simply unzip with an appropriate decompression utility (e.g. WinZip if you are using Windows). This will extract all files into a WSU_Sim_6.4 directory within the current directory.

WSU_Sim_v6.4.tar.gz

If you are on a Unix system, type the following at the shell prompt:

>gunzip WSU_Sim_v6.4.tar.gz
>tar -xvf WSU_Sim_v6.4.tar

(in Windows gzipped and/or tar files can be dealt with using some versions of WinZip, or WinRar - check the web for these utilities). This will extract all files into a WSU_Sim_6.4 directory within the current directory.

2.2 Files and Directories

Each of the above distribution packages contains a collection of class files that make up the main program, and a set of subdirectories containing auxiliary files. These include the 'controller', 'html-docs', 'images', 'manual', and 'maps' sub-directories. The directory tree created here serves primarily as an organizational structure for the files you will create, but you can store files wherever you want. The class files will be installed to the main WSU_Sim_6.4 directory; this is the directory you need to be in when starting the program.

The 'manual' sub-directory contains program documentation (at the moment you will find this file only). The 'maps' subdirectory contains some sample maze layouts that have been used in past assignments. You may want to store your own maps here as well.

The 'controllers' subdirectory contains example controller files: Robot_Example.java contains a full description of the method calls and static data fields that comprise your controllers interface to the main program, Robot.java.template contains the basic skeleton code that is common to all controllers, and Robot.java.io provides an example command driven controller. You may want to store backups of your control code in this subdirectory. The controller files you create need to be named Robot.java when you compile them, but obviously you can store them under any name you like (see the "RUNNING THE SIMULATOR" section below for more details).

You will see a Robot.class file in the main directory after installation. This is a pre-compiled controller that depicts a simple wall-following behavior. This file was added so that you can start running the simulator right away without having to write and compile your own controller first. You will want to delete or overwrite this class file when you begin developing your own control code. After reading this document you should run the program (if you haven't already) and make sure everything is executing smoothly before you write your own controller. If the robot animation and/or the sensor updates seem painfully slow, try using a JDK made by a different organization (see the "REQUIREMENTS" section for web related references).

3. INTRODUCTION

This program simulates the Khepera robot: a popular research tool used for implementing and testing various types of robotic controllers, and is developed and manufactured by K-team:

- http://www.k-team.com

The central purpose of this simulator is to provide a platform for “coarse-grained” controller development prior to testing on the actual Khepera robot. Because the real Khepara is a limited resource (we only have one hooked up at a time), you will most likely be spending the majority of your development time using the simulator. When it looks like your controller does what its supposed to do under simulation, you will then run your code on the actual robot, where any fine-tuning of the controller will be done on-line.

The simulated robot and it’s environment were designed to mimic the specific robotic hardware and testing environment used at WSU’s Neural Dynamics and Computation Lab (see Figure 1). This consists of a standard Khepara robot, its gripper-arm turret extension, and a 4’X 4' enclosed arena that may contain walls, lights, caps and/or balls. Walls and lights are assumed to be static objects in that they cannot be effected in any way by the robot. Thus if the robot were to drive into one of these objects, it would stop and possibly become stuck. Caps and balls are considered dynamic objects; they will move if the robot makes unintentional contact with them, and they can be directly manipulated by the robot’s gripper.

Figure 1: The robot arena used at WSU’s NDAC lab.

The behavior the robot exhibits while it is running is dictated by the current controller. In both the simulator and the serial port control programs, this controller runs as a separate thread - essentially executing as an endless loop. Typically, within this loop current sensor readings are sampled and based on their values (possibly along with the current state of the robot) the robot's effectors are assigned values. Sensors for our robot include 8 infrared sensors, which convey two types of data - object proximity and light intensity, wheel encoders for loosely determining position, and a gripper object sensor that indicates the presence of an object between the grippers. Effectors include wheel motor speeds, arm position, and gripper state (i.e. open or closed). For more information on the actual hardware specifications for the Khepera, go to:

- http://www.k-team.com/download/khepera.html.

Here you can find downloadable documentation for the standard Khepera, as well as the available extension modules (e.g. the gripper turret).

4. RUNNING THE SIMULATOR

There are actually two things involved in running this simulator successfully: understanding the GUI, and understanding how to write control code. For more information on writing control code for this program, see Section 6 and the example controller files in the controllers subdirectory. The rest of this section will deal with starting the program and understanding the graphical interface.

To start the program you need to first change to the main WSU_Sim_6.4 directory. If you are on a Unix system, you simply “cd” to this directory at the shell prompt. On Windows you will need to open either a window to DOS or the command prompt window, depending on your version of Windows. Once in the WSU_Sim_6.4 directory, type the following at the prompt:

```
> java RoboMain
```

Note that you must have a Robot.class file in this directory if you want to run the simulation. As noted previously, this release comes with a precompiled controller (i.e. Robot.class) that demonstrates wall following. If this file is still present, or you have compiled your own controller, just follow the instructions below to run the simulation.

When starting the program for the first time, it is not uncommon receive error messages from the Java VM. The most common problems experienced by new users are caused by incorrect system configurations for their Java environment. These problems include incorrect or nonexistent class path variable setting(s), incorrect or nonexistent path settings to Java executables (i.e. your Java’s bin directory), etc. Details regarding the installation and configuration of Java for different systems will not be dealt with in this document. If you are new to Java, please refer to the documentation pointed to by the links listed in Section 1.

5. GUI Components

The user interface is composed of seven main panels, or groupings of buttons and displays (see Figure 2). The functionality associated with each of these components can be discovered through trial and error, and for the most part should be fairly intuitive. The rest of this section describes the various components that make up the user interface. Although most of the information here may seem obvious, there are some quirks to controlling the robot and editing the environment that are described in this section, so please read on...

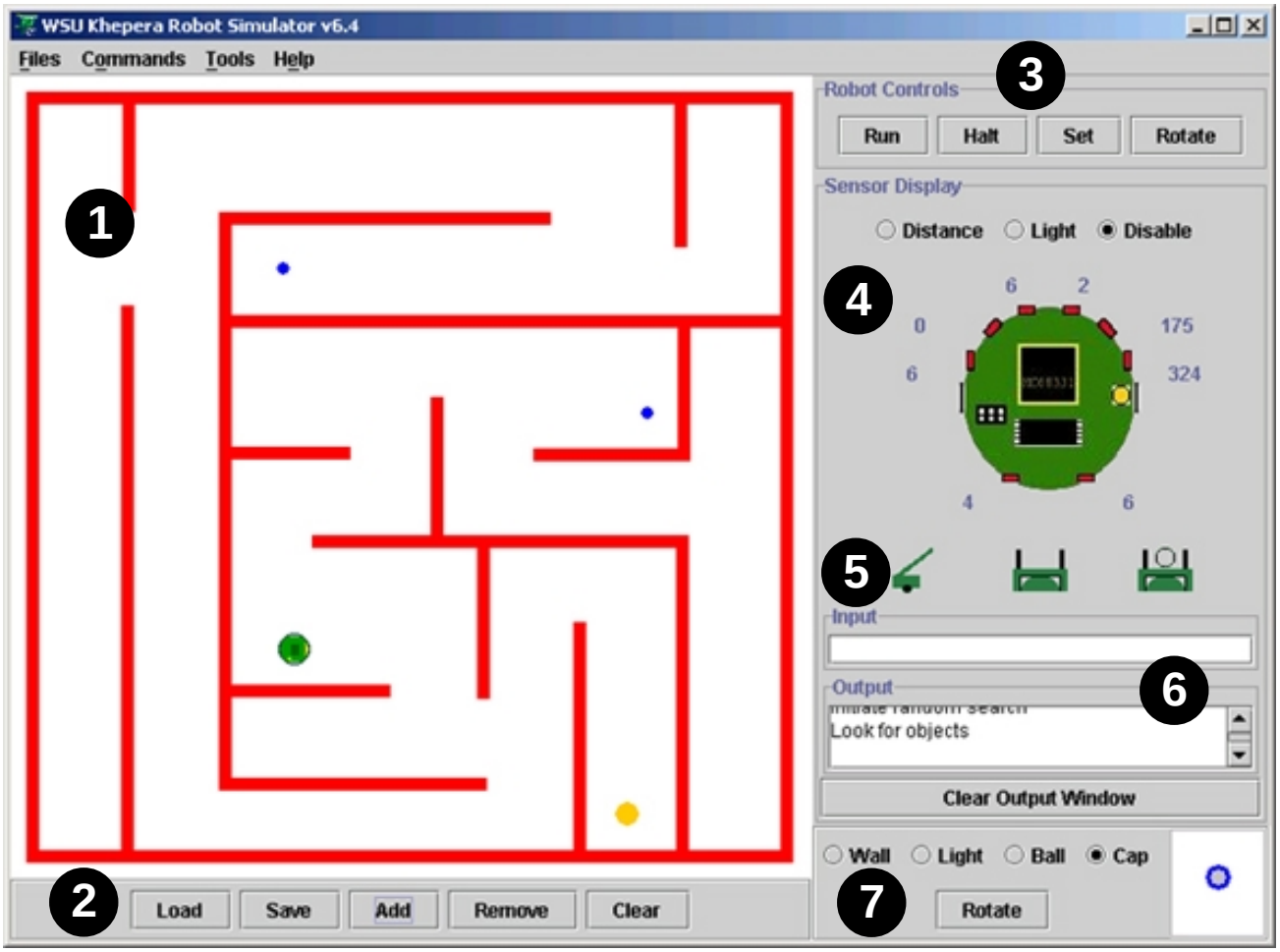

Figure 2: The user interface.1) World Panel, 2) World Control Panel, 3) Robot Control Panel, 4) Sensor Display Panel, 5) Arm Status Panel, 6) Controller I/O Panel, and 7) Object Selection Panel.

5.1 World Panel

The world panel displays the simulated environment and the robot. It is here that you place objects, position the robot and arrange wall sections. The world control panel consists of a row of buttons

that enable you to edit the environment as well as save or load those you have already created. The function of each button in this panel is listed below:

Load: brings up the load file dialog window. To load a previously saved world layout, just navigate to the directory where it is stored and select the file, then click on the load button.

Save: brings up the save file dialog window. To save the layout that is currently in the world panel, navigate to the directory you want to save to and type a name to identify the file in the text box under the current directory listing.

Add: clicking this button allows you to add an object or wall section to the world panel. Select the type of object you want to add from the object selection panel. Once the add button is selected, move the mouse cursor over the world panel. The type of object currently selected will appear with the mouse cursor. Click anywhere in the world panel to place the object. Make sure you "deselect" the add button when you are done.

Remove: clicking this button enables you to remove individual objects from the world panel. Select the type of object you want to remove in the object selection panel. Then double click on the actual object within the world panel to remove it. Like the add button, you must click the remove button to "deselect" it when you are finished.

Clear: removes everything from the world panel except the robot (if one is present).

5.2 Object Selection Panel

The object selection panel, mentioned above, is fairly intuitive (see Figure 3). By clicking on the appropriate radio button you select the object type to be added or removed. The window on the far right of this panel displays the object currently selected. To orient a wall section vertically or horizontally, click on the rotate button. Rotating objects other than wall sections essentially has no effect since these are all symmetrical.

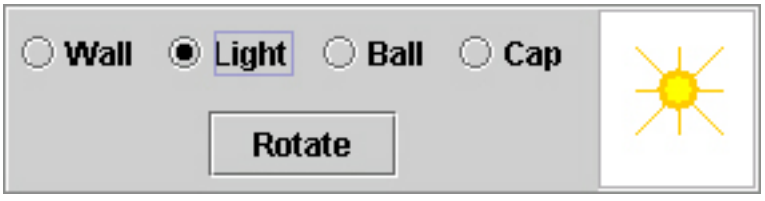

Figure 3: The object selection panel with a light selected.

5.3 Robot Control Panel

The robot control panel contains a row of buttons that enable you to initially position the robot, and start its execution. Here is each button's function:

Run: selecting this button starts the simulation (i.e. it executes your controller). Once selected you will see the robot interacting with the virtual environment you created in the world panel. The simulation will run until you either explicitly halt execution by clicking on the halt button, or implicitly kill the controller thread within your program. It is highly recommended that you end each simulation run using one of the methods above, as opposed to killing the entire program.

Halt: selecting this button will terminate the current execution of your controller.

Set: this button lets you place the robot in its initial position prior to starting the simulation. The behavior of this button is much like the add button in the world control panel. Move the mouse over the world panel and click where you want the robot placed. If you change your mind after placing the robot, just click again in the new location - the previously placed robot will disappear. When you have placed the robot in the desired location, deselect the button. If the robot is not set before the simulation starts, the robot will start from the center of the world panel. As of this release, this default starting position is fixed, therefore if any objects already occupy this space the robot may get stuck from the start. Thus it is highly recommended that you "pre-position" the robot.

Rotate: once the robot has been placed in the world, use this button to modify its orientation. Repeatedly clicking this button rotates the robot clockwise in 45 degree increments.

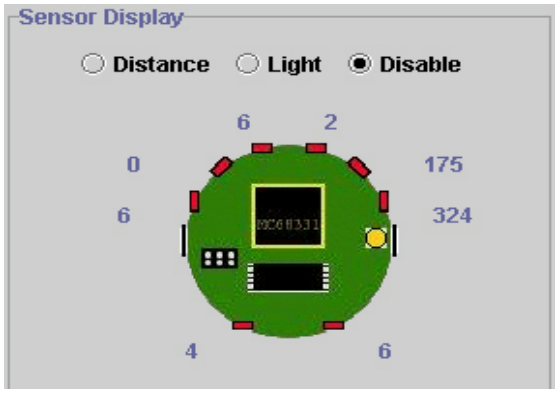

Figure 4: The sensor display panel.

5.4 Sensor Display Panel

The sensor display panel shows you what the robot is perceiving while the simulation is running (see Figure 4). The sensor values for each of the eight IR sensors will be continually updated and dis-

played next to its respective location on the robot image. To switch between light and distance readings, click on the appropriate radio button at the top of the panel. Distance values will range from 0 to 1023: 0 meaning nothing is detected and 1023 being the maximum value. For light readings the range will be from 512 to about 5: 500-512 implies total lack of ambient light, and 5-10 implies close proximity to a light source. You will notice that both light and distance readings will fluctuate to varying degrees even when there seems to be no apparent change in the robots relative position. This is done to simulate noise in the sensors. The third display option disables all sensor updates to the GUI. As of this version of the simulator, displaying sensor readings on the interface can cause a serious performance hit to the program. It is recommended that you display readings during testing and debugging sessions, but disable this feature at other times.

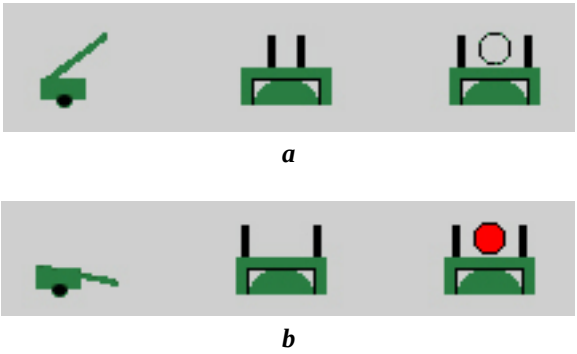

Figure 5: Two examples of gripper arm state depictions: a) arm up, gripper closed, and no object, b) arm down, gripper open, and object present.

5.5 Arm Status Panel

The arm status panel shows three icons that convey the current state of the arm, gripper and gripper sensor from left to right respectively (see Figure 5). Because there are certain states in which the arm and gripper unit are not graphically conveyed on the animated robot, the icons displayed here can help you determine what is happening at any given time. For example, the arm is not drawn on the simulated robot when it is in the “up” position. Therefore you will have to rely on these icons to determine if the gripper is open or closed (when the arm is down, however, the arm and gripper are drawn). Like the IR sensors, the gripper sensor is essential for detecting objects. By examining the icon on the far right (i.e. the gripper sensor icon), you can tell if the robot perceives anything between its grippers: a hollow circle between the grippers means nothing is currently there, while a red circle means that something is present. Certain conditions or states the robot may be in can be conveyed by combinations of icons. For instance, if the robot is carrying an object with the arm up, you will see the arm state icon depicting the arm in an upright position, the gripper state icon will show the gripper closed, and the gripper sensor icon will display a red circle between the grippers.

5.6 Controller Input/Output Panel

The controller I/O panel can be found on the lower right-hand section of the interface, just above the object selection panel (see Figure 2). The text boxes in this panel are used for sending and receiving messages and data to/from the controller thread. The single line text box at the top of this panel is used for sending commands and/or data to your controller - if such functionality is present in your code. The multi-line text area below the input field prints any output text generated by your controller. At the very bottom of the panel you will find a button that allows you to clear the output area (see Figure 6).

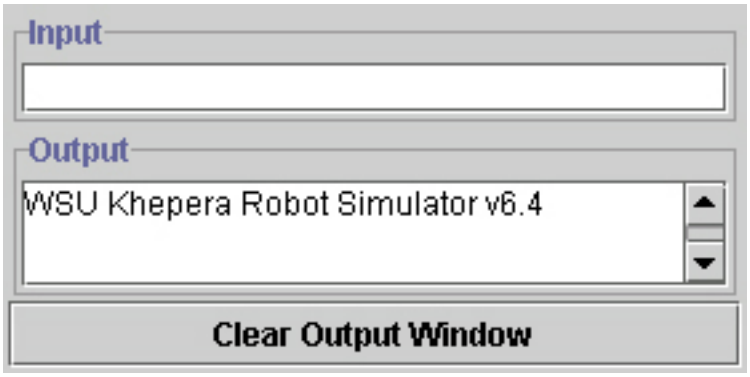

Figure 6: The controller I/O panel.

Focus is automatically given to the input field when the simulation is started. This means that when you type any characters they will appear in this field. You can determine if the text field is ready for input text by observing the presence (or absence) of a blinking text cursor on the far left side of the input field. As soon as you click on any other buttons, the focus on this component will be lost. In this case you have to move the mouse cursor over the text field and left click. This action will result in a blinking text cursor that is ready for keyboard input.

When text is written to the output area, it is presented line by line (assuming that you append a newline character to the end of each message you post within your control code). As text is written, it is placed on a new line at the bottom of the list consisting of all previous outputs - if any exist. New text sent from your controller will be immediately visible in the center of the output text area. To view any previously written text output, use the scroll bar to the right of the output area.

It should be noted again that you need to explicitly add this specific I/O functionality to your control code. See the API Section of the online help for more details.

5.7 Menu Items

At the top left-hand side of the GUI you will find some basic menu categories: Files, Commands, Tools, and Help. Under “Files” you will find alternatives to using the ‘Save’ and ‘Load’ buttons associated with the world control panel, as well as the program’s exiting command. Under “Commands”, you will see alternatives to using the robot control panel’s ‘Run’ and ‘Halt’ buttons, and a

menu item to clear the world panel's contents. The "Tools" menu provides access to the parameter adjustment window (see the subsection 5.8.1 for details). Finally, the "Help" menu lets you open the on-line help system (see subsection 5.8.2 for details).

5.7.1 Parameter Adjustment Window

Using the contents of the parameter adjustment window, you can modify the simulator's internal parameters that effect distance and light sensing, as well as motor speeds (see Figure 7). This window consists of 3 "stacked" sub-panels; each sub-panel can be accessed via its tab at the top of the window. Each sub-panel contains a slider bar that represents a particular program parameter and a sliding scale of possible values that the parameter can assume above and below its default value. The topmost panel provides access to IR sensor sensitivity levels as it pertains to proximity readings. Sliding the tab on this bar to the right makes the sensors more "sensitive" (i.e. they will detect obstacles at a greater distance), while sliding the tab to the left will have the opposite effect. The introduction of this feature was inspired by our experience with two different Khepera robots that seemed to exhibit different proximity sensor averages at the same distance from a given object. The current default setting accurately reflects the behavior of our present robot's sensors, but the flexibility enjoyed by having these values adjustable may benefit others who may own Kheperas with slightly different sensor averages.

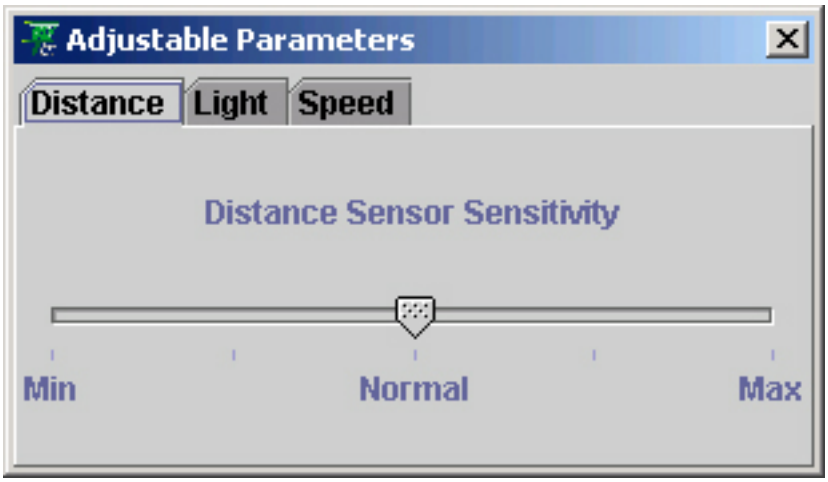

Figure 7: The parameter adjustment window.

The light sensor sensitivity panel essentially incorporates the same behavior as the distance sensor adjustment panel. The benefit of light sensor modification is given by the fact that lights used with the actual Khepera may exhibit varying degrees of intensity. This is particularly true of battery powered lights that may slowly dim over the life span of their power supply. By increasing or decreasing this parameter's value, you can indirectly simulate brighter or weaker light sources.

The bottom sub-panel provides access to wheel/motor parameters. Adjusting these values may be beneficial to you if you find yourself developing controllers for the simulator on two or more different computers - especially if processor speeds differ significantly between them. Since the speed of

your processor will probably effect the execution speed of the program, you may find that on slower machines fairly fast motor speed settings within the controller won't necessarily translate to a proportional speed in simulation runs. This way you can develop a controller in simulation on different computers that will translate well to the real Khepera without having to change certain values in your code; instead change parameter values.

5.7.2 Online Help Window

The online help window displays information about the simulator and controller programming. Most of the on-line help topics are addressed in this document.

Navigating through the on-line help contents is much like using a web browser - in fact the help content pages were written in HTML. Unlike a browser, however, all navigation through connected document pages is done by clicking links, as there are no 'Back', 'Forward' or 'History' buttons associated with the help window. At the bottom of each help page you will find links that take you back a page, or direct you to the main help topics page.

Since the help documents are written in HTML format, you can view them outside of the program. To do this, go into the 'html-docs' subdirectory and open the "index.html" file with your web browser.

6. WRITING THE CONTROLLER

Most of the issues involved with creating a controller for this program will be defered to the explanations and examples you will find in the Robot_Example.java, Robot.java.template, and Robot.java.io files found in the ‘controllers’ subdirectory. These files provide an overview of all the methods and static fields that make up the interface between the controller (Robot.java) and the simulator, as well as small example methods that show you some basic techniques. If you find you need more information use the online help. Here you will find information on the program’s API, and more. Once you have created a new controller it must be compiled, and the resulting class file (i.e. Robot.class) must be copied to the main WSU_Sim_6.4 directory where the other program .class files reside. Unless you are using a Java IDE, you will need to issue the command to compile your code at a command line prompt. In Unix this is simply the shell prompt, and in windows you use either a DOS shell window or a command prompt window depending your version of windows. To compile your new controller type the following:

```
>javac Robot.java
```

If your file compiled, you will see no output. If you see a message like “javac: command not found”, then you need to configure your path list to include executables in the bin directory under java’s main directory (refer to your java documentation for details). The final possible outcome would be a list of compile errors. If you are unfamiliar with the java language and see error messages you don’t understand, you should refer to some reference (check for references and how-to’s on the web). Even though your controller may compile, you may see some odd behavior exhibited during the simulation run. Outside of the overt behavior of the robot, the sensor reading displays, and the arm state icons, you may wish to have more information about what is going on during the run for purposes of debugging. Within your controller code you can insert print statements. If you print to “standard out” or “standard error”, your messages will be printed to the terminal or command window you started the program from. Alternatively, you may wish to write messages to a file.

7. SUMMARY AND COMMENTS

For those of you with little or no experience using a real Khepera (or any physical robot) here a brief list of things that may be taken for granted or unforeseen when you articulate your “expectations” via programming a controller:

1) Avoid obstacles at all cost when your robot is wandering around it’s environment. In the simulator, don’t expect to merely turn in order to maneuver your robot out of tight situations where it may be in contact with an unmoving obstacle.

2) When the Khepera’s arm is in the “down” position, it obstructs all but the two rear sensors. When you drop the arm, expect to “go blind”.

3) Sensors are noisy. Don’t expect a single snapshot of sensor readings within a short, discrete period of time to reveal an entirely accurate depiction of the robot’s surroundings.

4) The process of grabbing and carrying objects with the gripper and arm isn’t simple. It takes a small amount of time for the arm to lower (or raise), and the gripper to close (or open). Successive calls to these actuators in an attempt to grab an object, without some allowance for time in between, does not necessarily result in possessing the object. (See the Robot_Example.java file for examples).

5) In this simulator, you will notice that objects that are gripped will “disappear” (i.e. they are no longer rendered once the robot gains possession of them). Check the arm/gripper state icons to verify what is actually happening. When the object is eventually released, it will reappear wherever it was dropped.

6) Make liberal use of the robot’s wheel position values. These values, taken over time, are the best indicators of forward progress, and are critical when trying to determine if the robot is stuck.

7) Light sensor values reflect the relative proximity to a light emanating device in the immediate environment. In the simulated and real environments, obstacles (e.g. walls) are tall enough to obstruct light, even though the light source may be close. Don’t expect to detect light sources unless there is an unobstructed path between the robot and the light source within some relative distance.

8. KNOWN ISSUES

This application is currently in its sixth release. Thanks to students in Dr. Gallagher's CEG 759, CEG499 classes several bugs have been found and fixed, and a few minor ones have made it to the "to do list". So far students have been the only real beta testers beside myself. This essentially means that the program is fully functional, but not *fully* tested. Since the range of controls tested so far in this simulator has neither been complete nor extensive, you may very well find bugs in the program that have not yet been found. You may also notice certain "glitches" in terms of the GUI display and the robot animation, but none of these shortcomings should prevent you from writing, testing and observing the outcome your control programs. If you find a bug, especially one that effects your ability to implement some control, please e-mail me at: sperrett@cs.wright.edu . Please state what you have observed and attach a copy of your Robot.java file.