WSU Khepera Robot Simulator User's Manual

Version 7.2
March 24, 2005

Robotics in Education Group
Computational Autarkeia and Robotics Laboratory
Wright State University
Dayton, Ohio

Please send comments, questions, and suggestions to:

reg@carl.cs.wright.edu

Acknowledgements

A large number of people assisted in creating this software and manual. The original simulator was written by Steven Peretta. Various updates and enhancements, most notably the collaborative display sharing features, were added by Michael J. Cribbs, Brian Potchik, and Josh Howard. Many critical bug fixes were applied by Duane Bolick, who also authored the new robot remote control application and who developed an enormous number of programming examples. Aarti Raghavan provided critical assistance in editing and assembling documentation, including this manual.

Various agencies have funded this work over the years. We very much thank the National Science Foundation for their continuing support of this project. Wright State University and the Ohio Board of Regents have also provided much appreciated financial and material support for the courses developed around this software.

Finally, we would like to thank the many kind people who have (and continue to) send bug reports and suggestions. This is still very much a work in progress, and we rely critically on our users and to tell us what is and is not working.

WSU Khepera Simulator Licensing Agreement

The WSU Khepera Simulator is distributed under the Q Public License (QPL). It allows free use of the WSU Khepera Simulator for running software developed by others, and free use of the WSU Khepera Simulator for development of free/Open Source software.

THE Q PUBLIC LICENSE version 1.0

The intent of this license is to establish freedom to share and change the software regulated by this license under the open source model.

This license applies to any software containing a notice placed by the copyright holder saying that it may be distributed under the terms of the Q Public License version 1.0. Such software is herein referred to as the Software. This license covers modification and distribution of the Software, use of third-party application programs based on the Software, and development of free software which uses the Software.

Granted Rights

1. You are granted the non-exclusive rights set forth in this license provided you agree to and comply with any and all conditions in this license. Whole or partial distribution of the Software, or software items that link with the Software, in any form signifies acceptance of this license.

2. You may copy and distribute the Software in unmodified form provided that the entire package, including - but not restricted to - copyright, trademark notices and disclaimers, as released by the initial developer of the Software, is distributed.

3. You may make modifications to the Software and distribute your modifications, in a form that is separate from the Software, such as patches. The following restrictions apply to modifications:

    a. Modifications must not alter or remove any copyright notices in the Software.

    b. When modifications to the Software are released under this license, a non-exclusive royalty-free right is granted to the initial developer of the Software to distribute your modification in future versions of the Software provided such versions remain available under these terms in addition to any other license(s) of the initial developer.

4. You may distribute machine-executable forms of the Software or machine-executable forms of modified versions of the Software, provided that you meet these restrictions:

    a. You must include this license document in the distribution.

    b. You must ensure that all recipients of the machine-executable forms are also able to receive the complete machine-readable source code to the distributed Software, including all modifications, without any charge beyond the costs of data transfer, and place prominent notices in the distribution explaining this.

    c. You must ensure that all modifications included in the machine-executable forms are available under the terms of this license.

5. You may use the original or modified versions of the Software to compile, link and run application programs legally developed by you or by others.

6. You may develop application programs, reusable components and other software items that link with the original or modified versions of the Software. These items, when distributed, are subject to the following requirements:

    a. You must ensure that all recipients of machine-executable forms of these items are also able to receive and use the complete machine-readable source code to the items without any charge beyond the costs of data transfer.

    b. You must explicitly license all recipients of your items to use and re-distribute original and modified versions of the items in both machine-executable and source code forms. The recipients must be able to do so without any charges whatsoever and they must be able to re-distribute to anyone they choose.

    c. If the items are not available to the general public, and the initial developer of the Software requests a copy of the items, then you must supply one.

Limitations of Liability

In no event shall the initial developers or copyright holders be liable for any damages whatsoever, including - but not restricted to - lost revenue or profits or other direct, indirect, special, incidental or consequential damages, even if they have been advised of the possibility of such damages, except to the extent invariable law, if any, provides otherwise.

No Warranty

The Software and this license document are provided AS IS with NO WARRANTY OF ANY KIND, INCLUDING THE WARRANTY OF DESIGN, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Choice of Law

This license is governed by the laws of the State of Ohio in the United States of America. All disputes will be settled in Ohio courts.

1. Requirements

1.1 JDK Version

This simulator was developed using JDK 1.4.1. You should be able to use any JDK that is version 1.4.1 or higher. We have tested this simulator with Sun's JDK 1.4.1, 1.4.2, and 1.5.

You will find a listing of JDK packages at

http://java.sun.com

1.2 Operating System

This program has been extensively tested on most "popular" operating systems. These include Mac OS X, Linux, Solaris Sparc, Windows 98, Windows 2000, Windows NT and Window XP. Under each of these systems no performance problems were detected, although your results may vary depending on the underlying hardware.

2. Installation

2.1 Uncompressing Files

WSU_Sim_v7.2.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_7.2 directory within the current directory.

WSU_Sim_v7.2.tar.gz
If you are on a UNIX system, type the following at the shell prompt:

```
>gunzip WSU_Sim_v7.2.tar.gz
>tar -xvf WSU_Sim_v7.2.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_7.2 directory within the current directory.

2.2 Files and Directories

Each of the above distribution packages contains the following subdirectories and files:

/controllers – directory to place the controller you write
/source – contains controller source and template files
/html-docs – contains html documentation
/images – contains images used by the simulator
/manual – contains the simulator manual
/maps – contains maps saved the by the simulator
ksim.jar – executable jar file used to run the simulator
kremote.jar – executable jar file used to interface to remote robot

3. Introduction and Background

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 (ksim) is to provide a platform for "coarse-grained" controller development prior to testing on the actual Khepera robot. When it looks like your controller does what it's supposed to do under simulation, you can then run your code on an actual robot with kremote.

The simulated robot and its environment were designed to mimic the specific robotic hardware and testing environment used by WSU's Robots in Education Group (see Figure 1). This consists of a standard Khepera 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 affected 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.

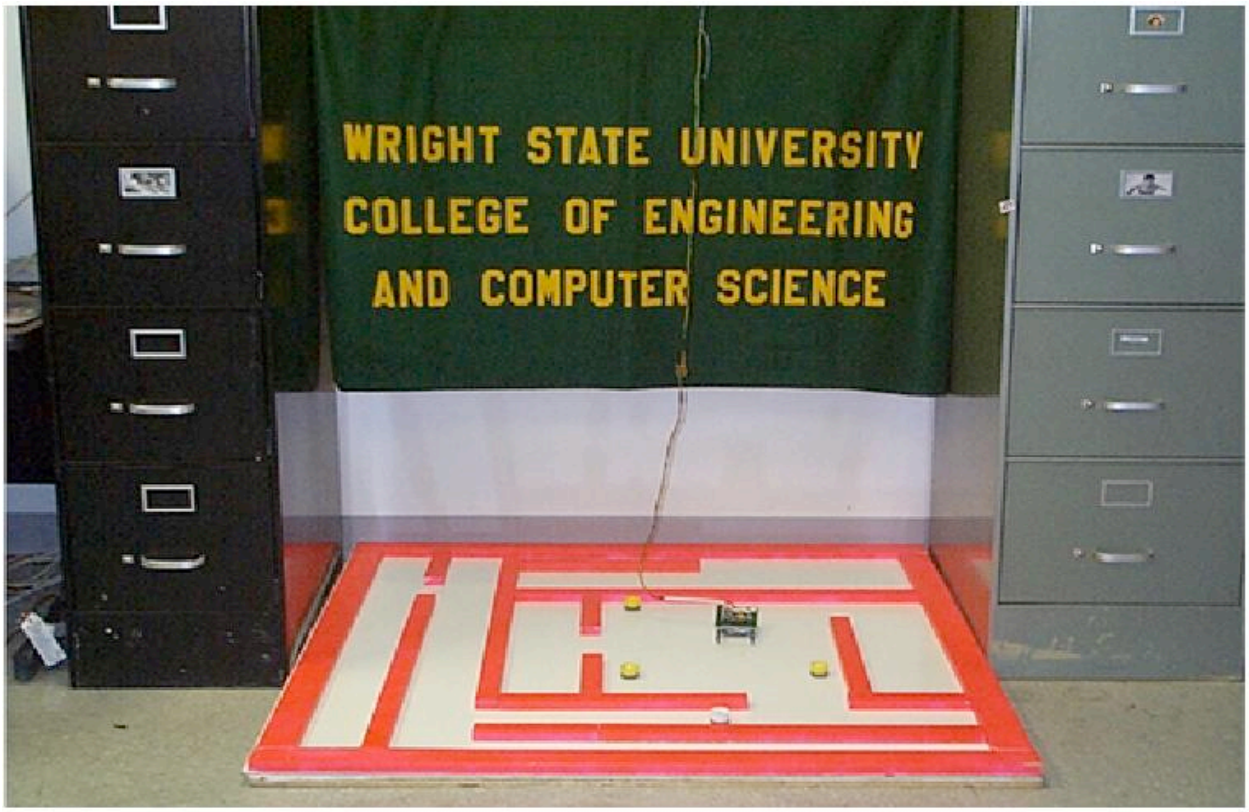

Figure 1: The robot arena used at WSU's NDAC lab.

The behavior the robot exhibits while it is running is dictated by the current controller. In both the ksim and the kremote programs, this controller runs as a separate thread – essentially executing as an endless loop. Typically, within this loop current sensor readings are sampled. Based on sensor values and internal program state, the robot's effectors are assigned values that cause it to act in the world. 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 simulator, see Section 6.

To start the simulator:

- Locate the home directory of the simulator software

- Start the Simulator
    - Windows and Mac OS X
      Double Click the executable jar file (ksim.jar). Alternatively, you can type "java –jar ksim.jar" from a command prompt. Do not type the quotes

    - Unix
      Type "java –jar JKSim.jar" from a command prompt

NOTE: When starting the program for the first time, it is possible to receive error messages from the Java VM. The most common problems experienced by new users are caused by incorrect system configurations for their Java environments. 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 or consult a local guru. Once your Java environment is properly configured, you will have no need to modify or care about any Java files outside of the simulation folder/directory.

5. GUI Components

The user interface is composed of nine main panels, or groupings of buttons and displays. 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.

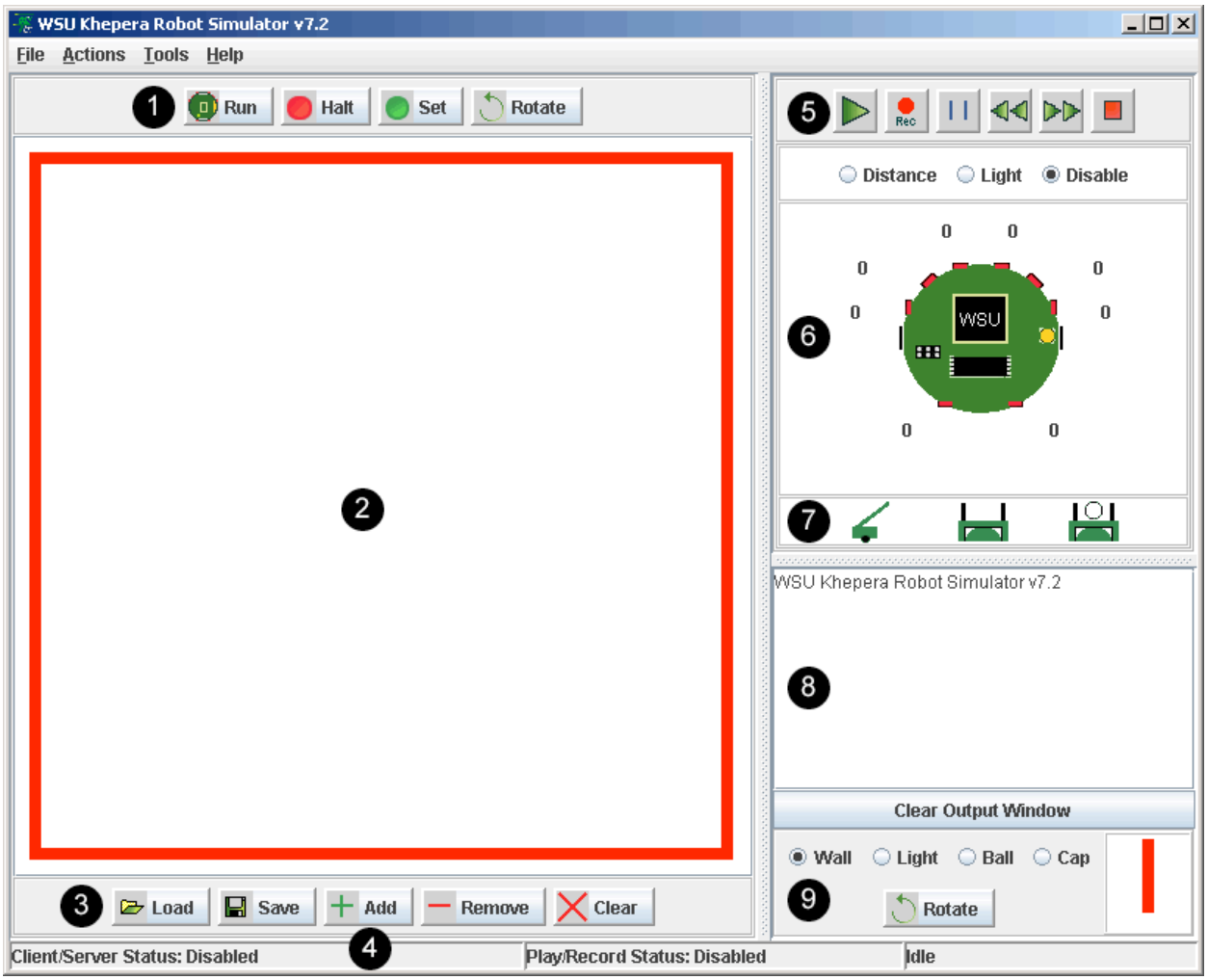

Figure 2: The user interface, 1) Robot Control Panel, 2) World Panel, 3) World Control Panel 4) Status Bar Panel, 5) Record/Playback Panel, 6) Sensor Display Panel, 7) Arm/Grip Status Panel, 8) Output Panel, 9) Object Selection Panel.

5.1 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: Select a controller to execute from the /controllers directory.

Halt: Terminate the currently running controller.

Set: Manually set the starting position of the robot. Once the Set button is clicked, just move the mouse into the world map and click to place the robot. Once the robot is placed you must deselect the Set button by clicking it again.

Rotate: Rotate the orientation of the robot by increments of 45°.

Figure 3: The Robot Control Panel.

5.2 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.

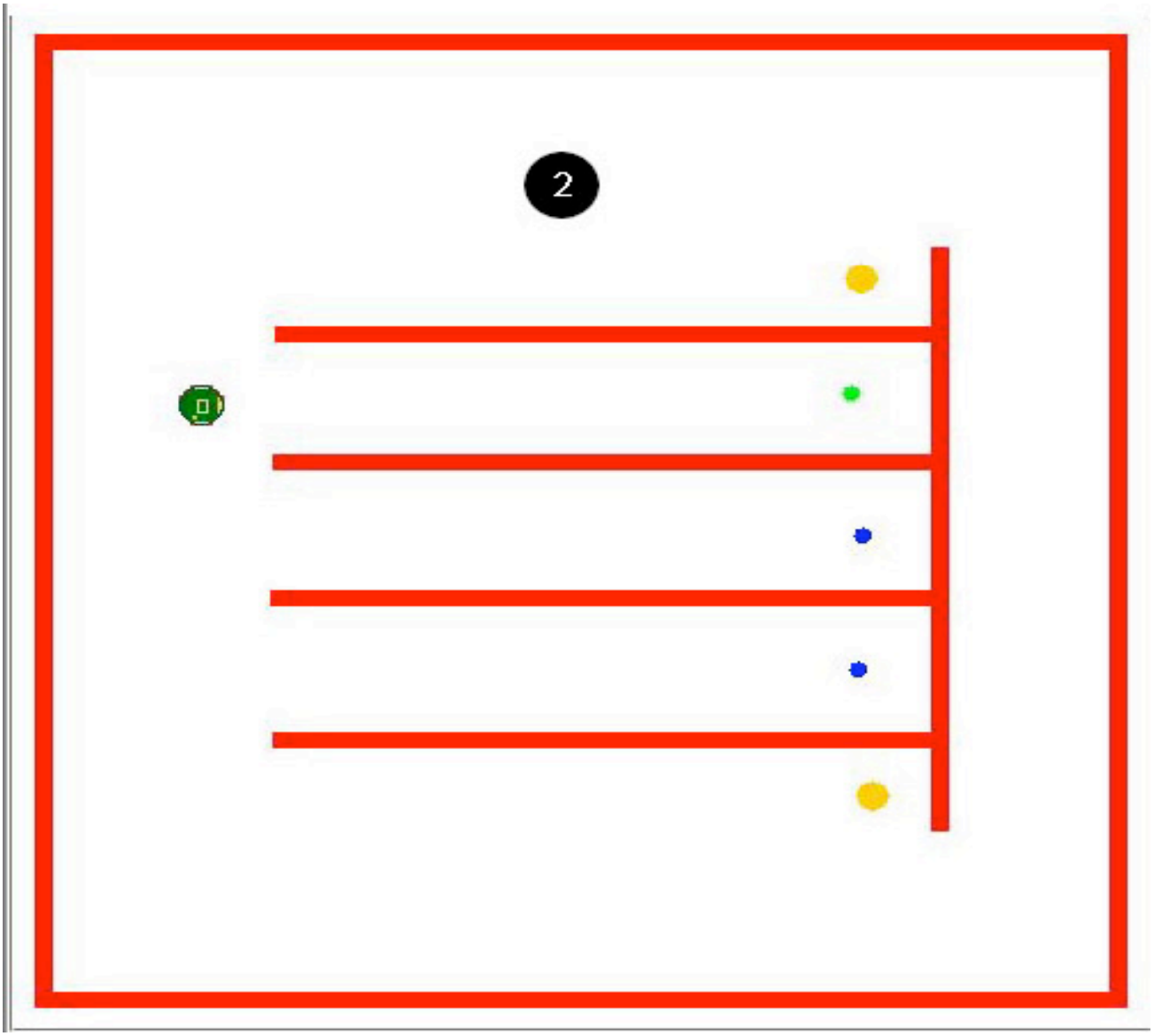

Figure 4: The World Panel.

5.3 World Control Panel

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).

Figure 5: The Robot Control panel.

5.4 Status Bar Panel

Left: The left side of the status bar displays the Client/Server Status.

Middle: The middle section of the status bar displays the Play/Record Status.

Right: The right side of the status bar displays the simulator status or the currently running controller.

Figure 6: The Status Bar Panel

5.5 Record/Playback Panel

Record: Allows the user to record actions of the current world panel. This includes the screen locations the robot, sensor display values as well as map, light, ball and cap object on the world panel. The user has the ability to set the record interval in the options screen.

Play: Allows the user to playback previously recorded files. A dialog pops up prompting the user to select a file to play. The play option then plays the appropriate file.

Pause: Allows the user to pause a currently playing file.

Rewind: Allows the user to rewind a currently playing file.

Fast Forward: Allows the user to fast forward a currently playing file.

Stop: Allows the user to stop a currently playing file.

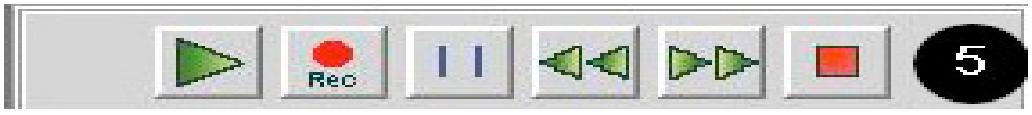

Figure 7: The Record Playback Panel

5.6 Sensor Display Panel

The sensor display panel shows you what the robot perceives while the simulation is running (see Figure 8). The sensor values for each of the eight IR sensors will be continually updated and displayed 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.

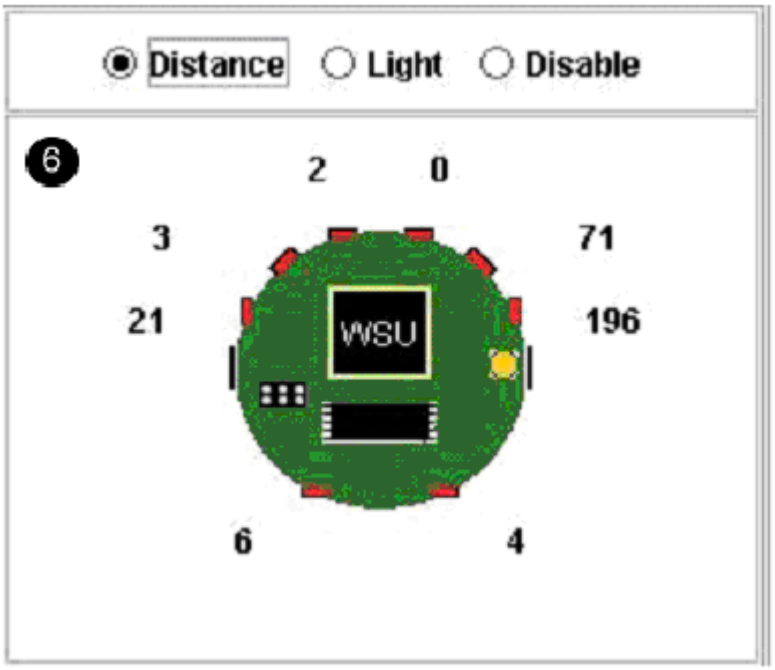

Figure 8: The Sensor Display Panel

5.7 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 9). 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.

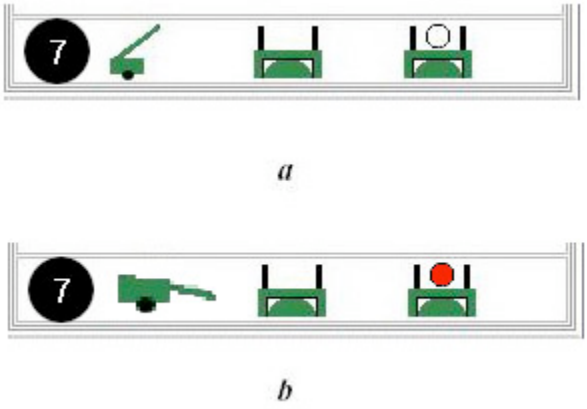

Figure 9: Two examples of gripper arm state depictions: a) arm up, gripper closed, and no object, b) Arm down, gripper open, and object present.

5.8 Controller Output Panel

The controller Output 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 receiving messages and data from the controller thread. The multi-line text area below the Arm Status Panel 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 10).

NOTE: This feature is currently disabled – It will be returned in a future version of ksim.

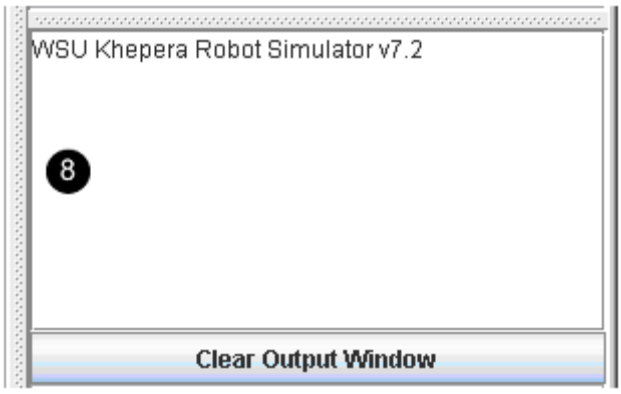

Figure 10: The controller Output panel.

5.9 Object Selection Panel

The object selection panel, mentioned below, is fairly intuitive (see Figure 11). 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.

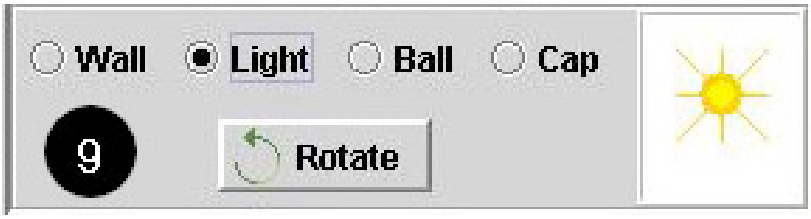

Figure 11: The object selection panel with a light selected.

5.10 Menu Items

At the top left-hand side of the GUI you will find some basic menu categories: Files, Actions, 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 "Actions", you will see a variety of actions that can be performed. The first section allows the user to run and halt a controller. The second section contains all of the Record/Playback features. The third section allows for Client/Server functionality.

Start Server: Start a server on the local simulator. This allows the local simulator to broadcast the actions of the currently running controller to clients that are connected to the server.

Stop Server: Stop the server. This will disconnect all connected clients as well.

Drop Client: Drop a client that the local simulator is currently servicing.

Start Client: Start a client on the local simulator. The client will connect to a server simulator and receive controller broadcasts. The HOSTNAME and PORT of the server simulator must be known.

Stop Client: Stop the local simulator client. This will end communication with the server simulator.

The "Tools" menu provides access to the options window where you can change different simulator options.

5.11 Options

Using the contents of the Simulator options window, you can modify the simulators internal parameters that affect distance and light sensing, as well as motor speeds. In addition you can set your record speed interval, network server transmission rate and server port number.

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 Khepera's with slightly different sensor averages.

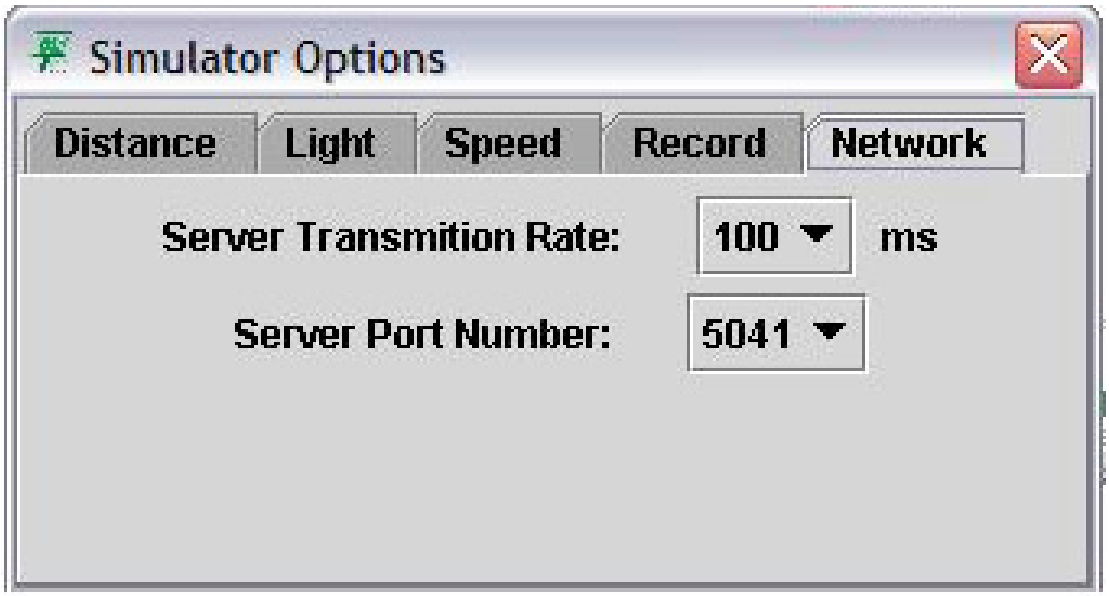

Figure 12: Simulator Options Panel with Network selected.

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 affect 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.

5.12 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. This can be done by opening the index.html file in the 'html-docs' subdirectory.

6. Writing a Controller

In order to write a controller for the simulator you must have the ksim.jar file. This archive is used as a library during the creation of a controller. In the /source directory there is a file called Template.java. This test controller will show you the basic format for writing a controller and provide relevant documentation to each part. The Template.java controller can be compiled and ran by the simulator. If you run the Template controller successfully the robot should spin in a circle.

Below is a list of tips that will help you during the creation of a controller.

- The controller file must import the simulator library. This can be done by adding the following import statement: "import edu.wsu.KhepheraSimulator.RobotController;"

- The controller name (class name and file name) can be anything.

- The controller class must extend the RobotController class.

- The controller class must define and implement the methods:
  - doWork()
  - close()

- The constructor can contain anything. This is also a good place to call "setWaitTime()" which sets the time to wait before the next call to doWork().

- Never call the wait() method to insert your own delays. Instead use the RobotController's sleep() method if a delay is needed.

- The JavaDoc documentation for the RobotController class will provide all the information needed to check the robots sensors and to control the robot.

7. Summary and Comments

For those 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 its 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 rise), 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.

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.

More information and ideas on how to build a robot controller can be found in the WSU WWW course or in any number of books on autonomous and/or mobile robots.

8. Known Issues

This application is currently in its seventh release. Even so, we consider the program to be fully functional – but not fully tested. The following items are known issues:

1) Maps can't be saved with lights. Maps that contain lights will appear to save and load correctly. However, the lights will NOT be handled correctly on reload. The work around is to save maps that do not contain lights. Place the lights manually when the map is loaded.

2) There are occasional GUI display glitches in the robot animation. This may be annoying, but should not prevent you from writing, testing, and observing the outcomes of your control programs.

If you find a bug, especially one that affects your ability to implement some controller, please e-mail:

reg@carl.cs.wright.edu

Please state what you have observed and attach a copy of the Java source of your controller program.