EyeSim VR
User's Manual

EyeSim VR Team

November 3, 2017

[Revised March 23, 2025]

1 GENERAL INFORMATION

This simulator will let users simulate the robots' execution of functions specified in RoBIOS-7 file. It accepts and runs customized script files written in C, and simulates robot behaviors of each command. User can select or specify a world or maze file for the simulator to build the simulation environment accordingly. User can also create any number of robots of any kind provided in the robot models, and the robot added last will be controlled by the user script. Objects like cans and soccer balls can also be added to the simulation, and physical interactions between these objects and robots can also be simulated.

2 SYSTEM CONFIGURATION

Following is a table of tested OS and prerequisites for each OS. Please note that other system versions may also work, they are just not tested yet, if your system version cannot work properly, please file a bug according to section 5 or upgrade your system to the tested version.

The required supporting software should be installed in their default directory before start using the simulator.

Operating Systems	OS Version	Prerequisites
Windows	Windows 8.1, 10	None
Mac OS	10.10.X	Install Xcode and Xquartz
Linux	64bit	Install X11 library

3 GETTING STARTED

3.1 Installation

Mac OS:

(1) Download and install XCode:

https://itunes.apple.com/au/app/xcode/id497799835?mt=12

(2) Download and install XQuartz:
http://robotics.ee.uwa.edu.au/eyesim/ftp/aux/mac/ or
https://www.xquartz.org/

Note: Your system default XQuartz app may need to be removed and reinstall from above link to include X11.

(3) Run following command in terminal to link the x11 library:

```
sudo ln -s /opt/X11/include/X11 /usr/local/include/X11
```

(4) Download and install EyeSim for macOS:
http://robotics.ee.uwa.edu.au/eyesim/ftp/

(5) Download and unzip eyesimX (EyeSim Examples):
https://robotics.ee.uwa.edu.au/eyesim/ftp/

Windows:
(1) Download and install EyeSim for Windows:
http://robotics.ee.uwa.edu.au/eyesim/ftp/

(2) Download cygwin.zip and Xming.zip from:
http://robotics.ee.uwa.edu.au/eyesim/ftp/aux/win/

(3) Unzip cygwin.zip to C:\Program Files (x86)\eyesim\cygwin

(4) Unzip Xming.zip to C:\Program Files (x86)\eyesim\Xming

(5) Download and unzip eyesimX (EyeSim Examples):
https://robotics.ee.uwa.edu.au/eyesim/ftp/

Linux(64bit):

(1) Install X11 library using following command:

sudo apt-get install libx11-dev

(2) Download the latest EyeSim for Linux:
http://robotics.ee.uwa.edu.au/eyesim/ftp/

(3) Unarchive the .tar.gz file and run the 'install.sh' script

(4) Download and unzip eyesimX (EyeSim Examples):
https://robotics.ee.uwa.edu.au/eyesim/ftp/

3.2 System Menu

When the application is launched, a window as shown in Figure 3-1 will show up to let you select the resolution and graphics quality, and if ***windowed*** check box is ticked, the simulator will run in a window instead of full screen, you can view the keybinds for control the simulator by choosing the **input** tab.

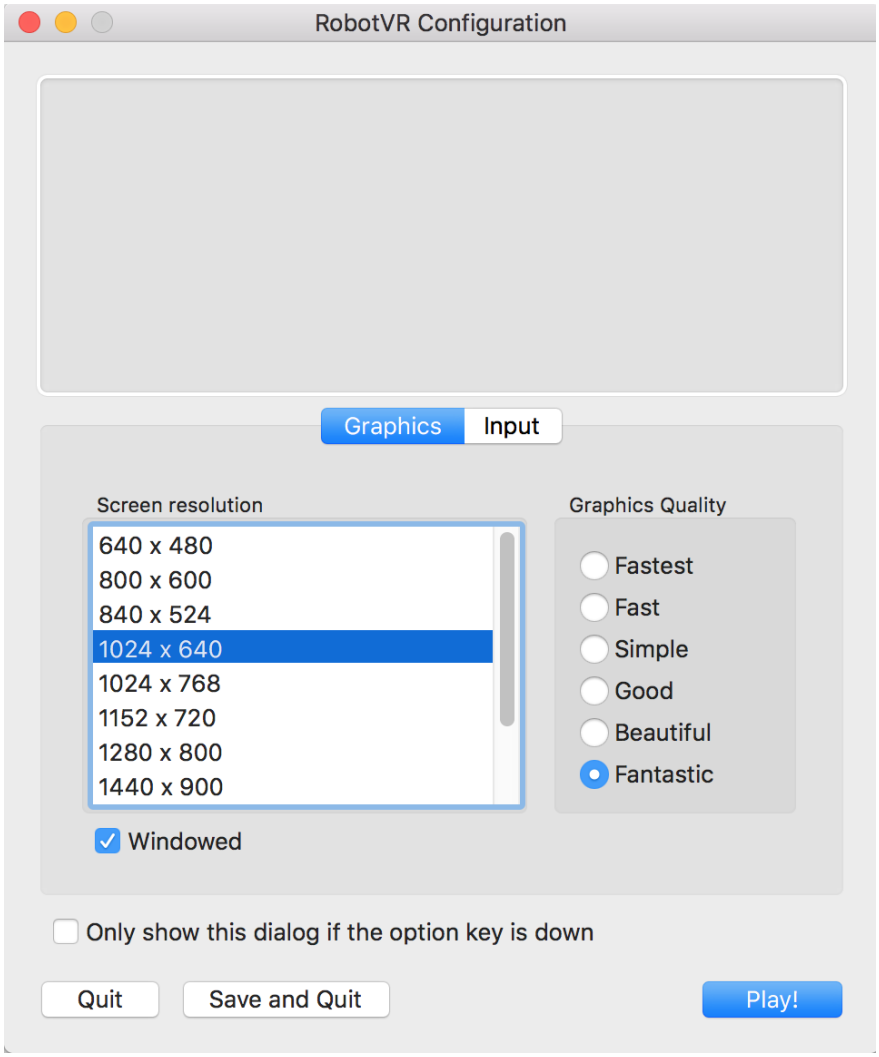


Figure 3-1 Configuration window for Mac

The simulator main window shows after the ***Play!*** button is clicked, as shown in figure 3-2. The black plane in the middle is where the simulation will be, and the menus on the upper-left corner help to set up the simulation.

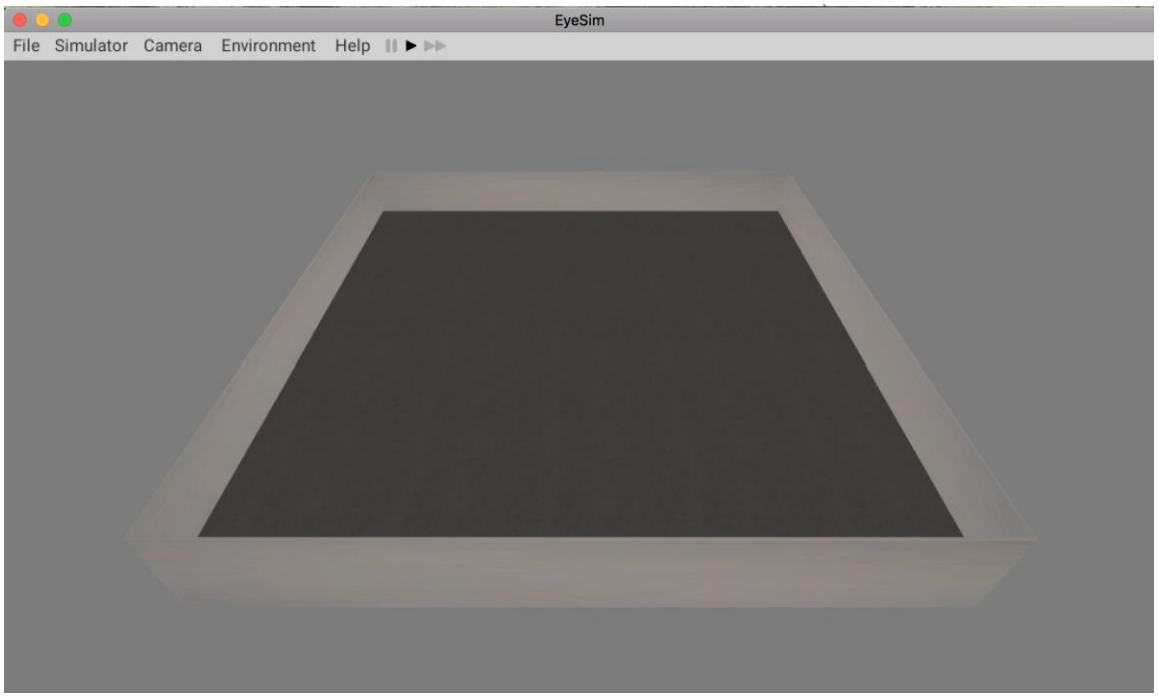


Figure 3-2 Simulator main window

File	Simulator	Camera	Environment	Misc	Help
Open Terminal	Add Robot	Birdseye View	Add Wall	1v1 Soccer	RoBIOS API
Load Sim	Add Object	Follow Object	Remove Wall	2v2 Soccer	View Log
Save Sim	Add Marker	Reset Camera	Paint Walls		About
Load World	View Robots				Changelog
Save World	View Objects				
Create World	Save State				
Reset World	Load State				
Load Object	Pause				
Settings	Resume				
Exit					

Table 3-1

This above table shows the menu functions of simulator, we can

load world, save world, add objects and robots using these menus, detailed information related to each of the menu item will be introduced later.

4 USING THE SYSTEM

Please make sure you have all the required files/folders as described in section 3.1 before you proceed. Please make sure you launch the simulator application to perform following supported functionalities.

4.1 Adjust the Viewpoint

To adjust the simulation plane to a better viewpoint, you can use →←↑↓ arrow keys and **w, s, a, d** to move the plane around, and use the number key x and z to zoom in and out.

You can have a birdseye view of the simulation by choose from menu **Camera** -> **Birdseye View**. And you can reset the camera to normal viewpoint by **Camera** -> **Reset Camera**.

4.2 Load/Reset/Save/Create World

To save the current world, select **File -> Save World**, simulator will save current world settings in a file called **SavedWorld.wld** in the root of the EyeSim folder.

To load a world/maze file as simulation environment, in the main simulator window, select ***File***-> ***Load World*** submenu and then in the popped-up file selector, navigate to and click on the intended world or maze file (with wld or maz extension). The simulator will build simulation environment according to the selected file.

To reset the world/maze environment to the default one, select ***File***-> ***Reset World*** submenu.

You can use the **File -> Create** World menu to adjust the

dimensions of the current world.

4.3 Add/Remove Wall

To manually create a customized world, you can select **Environment** -> **Add Wall** to add wall, then you need to click two points in the simulator plane to set the starting point and ending point of the wall.

To remove any wall, you can choose **Environment** -> **Remove Wall**, and then click on the wall to be removed from environment, the selected wall will turn red when you move cursor on to it indicating it has been selected.

4.4 Place Robots/Objects/Markers

To place a robot in the simulator, select ***Simulator***-> ***Add Robot*** then selection a robot to add to the simulation.

To place an object (a can or a soccer ball) in the simulator, select ***Simulator***-> ***Add Object*** , then select the kind of object you want to put in the simulation.

Then you can move the cursor with the robot/object stuck on it to any valid place (robot/object is highlighted in green to indicate a valid placement spot, while in red not valid), and click your mouse to place it.

Though ***Simulator***-> ***Add Marker***, you can add a colored marker point to the world in order to mark a position. You can also select a different color after the marker is placed, by double clicking on it and choose a color in the pop up inspector.

4.5 Inspect Robots/Objects/Markers

User can inspect the current status of any robot/object by double clicking on it, an inspector window will pop up.

The robot inspector window:

Figure 4-1 shows the inspector window of an object.

The **Camera** tab provides a camera image, and you can control the level of noise added to the image, by selecting **Salt and Pepper** and adjust the two parameters below.

The **PSD** tab shows the current psd readings of the robot, and you can also add errors to the PSD sensors by selecting **Error Enabled** and adjust the parameters of mean and std. Dev. of error. There's a toggle box called **Visualize Sensor**, when it is toggled, simulator will show the PSD sensor ray cast when performing the **PSDGet** command.

The **Driving** tab shows the current x,y coordinates of the robot and the rotation value Phi.

The **Control** tab can be used to select compiled simulation script files for simulation, and disconnect the control at any time.

The object inspector window:

Figure 4-2 shows the inspector window of an object, the information includes the name, id of the inspected object, x and y coordinates of the object against the lower-left corner of the simulation plane, and the rotation parameter.

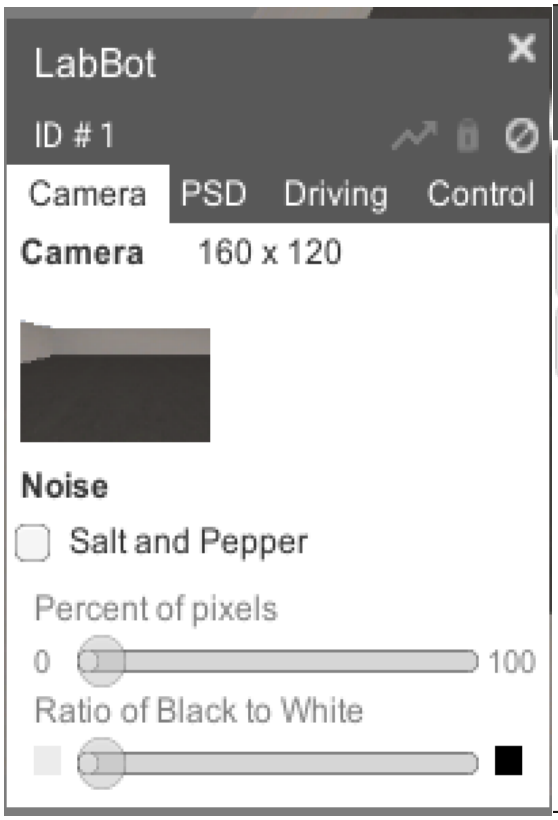


Figure 4-1 Inspector window for robots

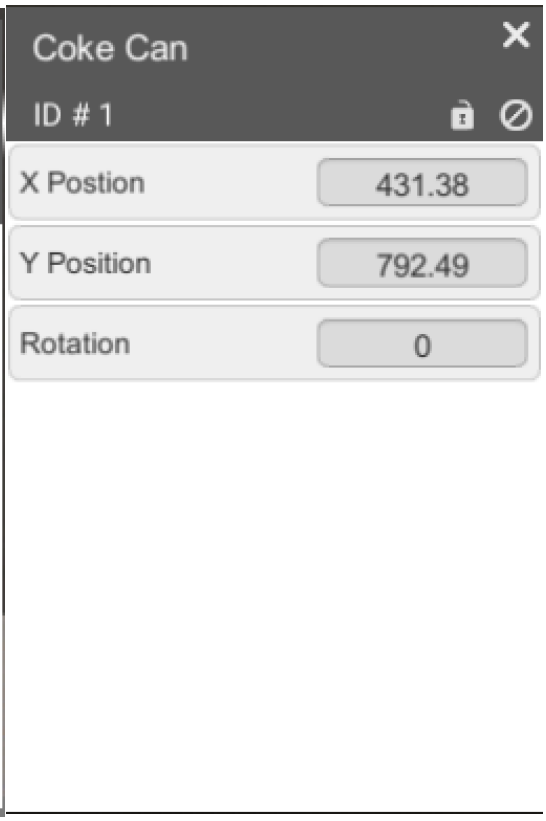


Figure 4-2 Inspector window for objects

4.6 Relocate/Rotate a Robot/Object/Marker

You can move a robot, object or marker to any new valid spot after they have been added to the simulation.

To move a robot/object, you can double click on the target to open the inspector window, then click on the  icon so it becomes grey, a marker doesn't require this setting to move.

If you want to move the robot/object/marker, you can click on the target and drag your cursor to the desired spot, then click again to place it.

If you want to rotate the robot/object, you click on the target and drag until it is picked up and turned green, then use - and = key to control its rotation.

4.7 Relocate/Rotate a Robot/Object/Marker to Specified Value

Sometimes you want to move a robot/object/Marker to a specified position or rotate a robot/object to a specified degree.

To specify the x,y position or rotation degree of a robot/object/marker, you need to select submenu ***Simulator***->***Pause*** to make the position or rotation data editable, then double click on the target to open inspector window, you can see now you can input or change the position and rotation value of the target, after you typed in each desired value, you need to click elsewhere to make it work, after finishing the relocation, you should select ***Simulator***->***Resume*** to return to normal mode.

4.8 View all Robots/Objects

You can inspect all robots/objects in the simulation easier by selecting ***Simulator-> View Robots/View Objects*** submenu, this will create a list of all the robots/objects in the simulation as shown in figure 4-3. You can inspect the current status of any robot/object by simply clicking on the target in the list, and an inspector window of that target will pop up.

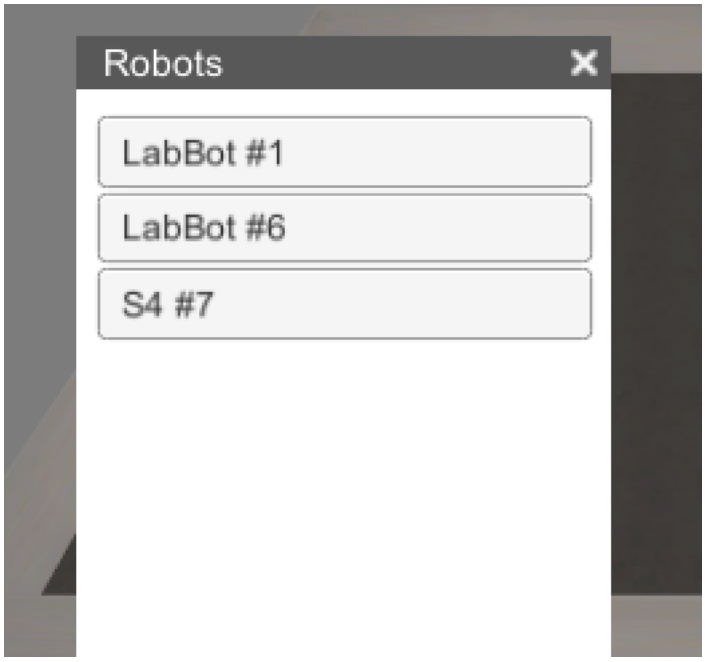


Figure 4-3 list of robots

4.9 Delete a Robot/Object/Marker

You can delete any robot/object/marker in the simulation by double clicking on the target to open the inspector window, then click the  icon to remove the target from the simulation.

4.10 Save Sim/Load Sim/Save State/Load State

During the process of your simulation, you may want to save the whole current simulation including the world, robots, objects and markers, you can select **File** -> **Save Sim**, simulator will automatically save the current simulation to a file called **SavedSim.sim**. You can restore the simulation at a later time by using **File** -> **Load Sim**, and select the SavedSim.sim file. Alternately, you can use **Similator** -> **Save State**, and **Simulator** -> **Load State** to quickly store and restore the simulation, without having to save and fetch from a sim file.

4.11 Create & compile scripts

To compile example scripts, go the subfolders under example

folder, where you can find a file called **Makefile** together with other script files in C, type following command in your terminal to compile all the scripts in this folder:

make

You will find one compiled file generated (in .x or .exe extension) for each script file.

To compile your own script file,

For Mac OS:

To compile your own script file You can create a folder called "myscripts" in eyesim folder. Then created a script file in C in this folder (for e.g. main.c).

Open your mac terminal and cd to this folder. Type in following command to compile the script files in this folder, you can also lauch a terminal window using **File** -> **Terminal** :

gccsim -o program main.c

The above command will generate an executable file called program in this folder.

For Windows:

Create a folder called **myscripts** at any place you want, and create a script file called main.c inside it.

In simulator main window, select submenu ***File/Open Terminal*** , and the Cygwin terminal will pop up.

You can see your current folder is called **tmp** under Cygwin folder, you can navigate to your c disk using command:

cd /cygdrive/c

Then further navigate to your **myscripts** folder, and run command.

gccsim -o program main.c

Advanced Windows Users (experience with cygwin / linux subsystem):

If you have an installed version of cygwin already on your machine,

or are experienced using the Linux subsystem for Windows, you can use either of these to run your programs. The eyesim libraries and header files are available by themselves.

For Linux:

To compile your own script file You can create a folder called "myscripts" in eyesim folder. Then created a script file in C in this folder (for e.g. main.c).

Open your terminal and cd to this folder. Type in following command to compile the script files in this folder:

gcc main.c - L ../lib -I ../include -leyesim -lX11 -lm -o program

The above command will generate an executable file called **program** in this folder.

Note 1: Appendix A shows a list of RoBIOS-7 functions that are not supported by the simulator, all other functions in RoBIOS-7 should work in the script.

Note 2: Following is a very simple sample script for your reference.

```
#include <stdio.h>
#include "eyebot.h"

int main() {
  VWStraight(600, 30);
  VWWait();
  VWTurn(180,10);
}
```

4.12 Start Simulation

Requirements:

There should be at least one robot in the simulation, if not, you should add a new robot to the simulation according to section 4.3.

There should be a compiled script file (with extension **.x**)placed in the folder, if not, you should first create a script file and have it compiled according to section 4.9.

There are two ways to control a robot with script.

From command line:

For Mac OS/Linux:

Open terminal and navigate the folder of any executable files you have compiled and run command: **./{your file name}** to run the executable script.

For Windows:

Open the cygwin terminal and navigate (cd) to the directory where the executable file (program.exe) is located , then run: **./program** in terminal.

A response in the terminal as shown in figure 4-4 is expected to show, indicating the simulation is running, and you can watch the simulation in the application.

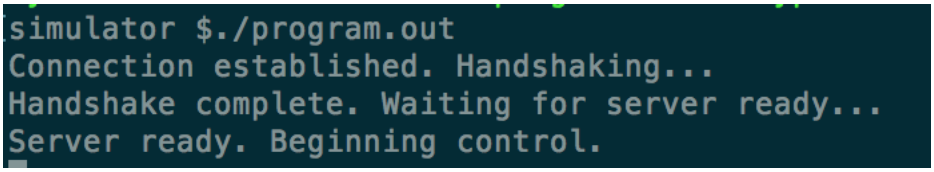


Figure 4-4 terminal response

Note: Currently only one robot can be controlled by the script, if you have multiple robots added to the simulation plane, only the last added robot will respond to the script.

Select control script from robot inspector window:

By selecting control script for each robot in the simulation, we can control multiple robots at the same time. Double click on any robot in the simulation, the inspector window will show up (as shown in figure 4-1). Click on the ***Select Control*** button, than navigate and select any compiled script file (in .x extension), the robot will simulation the

selected script. To terminate the execution of the rest script commands, just lick on ***Disconnect*** button in the inspector.

4.13 Pause/Resume/Speed up Simulation

You can pause and resume a simulation when the simulation is running, by select ***Simulator->Pause*** to pause the current simulation, and ***Simulator->Resume*** to resume the simulation. You can also click on the icons on menu bar :  to pause, resume and speed up the simulation.

4.14 Add Trace to Robot

You can add trace to the robot during a simulation by double clicking on the target robot to show the inspector window, then click on the  icon on the upper right corner of the inspector window to may it highlighted instead of grey, you will see a green trace added to the route it has covered.

5 LOADING FILES

EyeSim supports the loading of the following external files:

- .robi files for custom Robots
- .esObj files for custom Objects
- .wld or .maz files for custom environments
- .sim files for preconfigured simulations

These files are standard text documents, with the appropriate extensions, and contain a series of commands to pass to the simulator. Each command begins with a keyword, as is followed by arguments separated

by whitespace. An argument can be contained by double quotes (“”) if it contains a whitespace itself (such as a path to a file).

5.1 .robi Files

A .robi file specifies the parameters for a custom robot. Any line that begins with a # is treated as a comment, and is not processed by EyeSim. The keywords and corresponding arguments are as follows:

Keyword	**Arguments Specification** **Example**
drive	One of the following: DIFFERENTIAL_DRIVE ACKERMANN_DRIVE OMNI_DRIVE MANIPULATOR_DRIVE
name	Name of the robot MyRobot
mass	Mass in kg, followed by position centre of mass in mm (kg x y z) 5 0 30 -50
speed	Maximum linear velocity in mm/s 600
turn	Maximum rotational velocity in deg/s 300
model	Path to a .obj model, x y z offset (in mm), rotation about x y z axis (in degrees) “\Robots\Models\LabBot.obj” 14 0 0 0 90 0
axis	Distance between the centre of the robot, and the centre of the axis (vertical, horizontal in mm) 22.7 10.8

psd	Id number, PSD name, position relative to robots centre x y z (in mm), and rotation x y z (in deg) 1 PSD_LEFT 30 0 80 0 90 0
camera	Camera position relative to robot x y z (in mm), default pan and tilt (in deg), max pan and tilt (in deg) 40 50 70 0 0 90 90
wheel	Wheel diameter (in mm), maximum rotational velocity (in deg/s), encoder ticks per revolution, distance between wheels (track, in mm) 45 3600 540 70
lidar	position relative to robots centre x y z (in mm), and rotation x y z (in deg), angular range [1…360], tilt angle relative to driving plane (in deg, between -90 and 90), number of LIDAR points 0 0 0 0 0 0 180 10 360
thruster	Id number, thruster name, thruster diameter, max speed, position relative to robots centre x y z (in mm), and rotation x y z (in deg) 1 THRUSTER_LEFT 180 1000 -320 290 -170 90 0 0
fin	Id number, fin name, axis, max angle, size x y z (mm), and position relative to robot's centre x y z (mm) 1 FIN_UPPER Y 90 10 100 100 0 250 -190
buoyancy	Volume of robot (m^3) 0.012
turn_offset	Offset value -3

The first non-comment line of a .robi file must be the drive keyword and arguments. After a robot is loaded, it will be added to the Add Robot submenu (under Simulator), specified by the name parameter.

Manipulator Drive Specific Functions

parameters	The Denavit-Hartenberg parameters that specify the relative position of the joints and the axes of motion Ordered by d (mm), theta (degrees), r (mm), alpha (degrees). 300 90 0 0 : 100 0 100 90 : 300 0 0 0
scale	The scale of the arms and joints relative to the default 0.5
gripper	The presence of a gripper on the end of the manipulator (only keyword required)

More on Denavit-Hartenberg (DH) parameters: the example of

300 90 0 0： 100 0 100 90： 300 0 0 0

Represents a manipulator with 3 limbs. To specify that any of the limbs have variable axes of motion, add bounds to one or more of [d, theta, r, alpha] using brackets. For example:

300(−100 | 100) 90 0 0： 100 0 100 90 (V | V)： 300 (−300 | V) 0 0 0

In this example, the first arm is prismatic as it can move from its initial length of 300 an amount of -100 or +100. The second arm has a rotation joint attached at the end as its alpha parameter has no lower or upper bounds represented by (V | V). The third arm is prismatic like the first, but there is no upper bound represented by the 2nd value being a ‘V’ (-300 | V).

5.2 .esObj files

Custom objects can be loaded with .esObj files. These are simple world objects that interact physically with the robots. The keywords for this type of file is as follows:

Keyword	**Arguments Specification** **Example**
name	Name of the object Bottle
obj	Path to a .obj file “\Objects\Models\Bottle.obj”
scale	Scale of the object (modifies model size, positive number) 0.1
mass	Mass (in kg), centre of mass x y z (in mm) 1 0 0 0
collider capsule	Centre of capsule x y z (in mm), radius (in mm), height (in mm), vertical axis (character x, y, or z) 0 0 0 1 3 y
collider sphere	Centre of sphere x y z (in mm), radius (in mm) 0 0 0 1
collider box	Centre of box x y z (in mm) side length of box x y z (in mm) 0 0 0 2 2 2
buoyancy	Volume of object (m^3) 0.0012
fixed	N/A

An object consists of multiple colliders, to allow more complicated objects to be created. All the positions are relative to the centre of the model.

5.3 .wld and .maz files

Custom environments can be loaded with .wld and .maz files. A .wld file consists of a floor, walls, and an optional floor texture. The floor can be specified by the keyword floor followed by the width and height in mm:

```
floor 2000 2000
```

Or by the width and height keywords

```
width 2000
height 2000
```

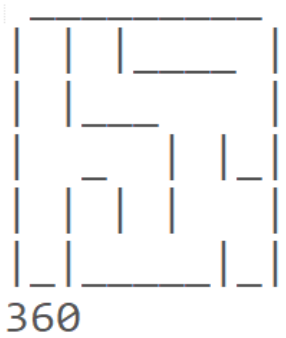
A texture to apply to the floor is specified by the keyword floor_texture, followed by a path to a .png file.

Walls have no keyword, and are simply lines with 4 numbers: x1 y1 x2 y2 eg:

```
0 0 1000 1000
```

Will create a diagonal wall from (0, 0) to (1000,1000).

Maze files, specified by .maz, contain an ASCII representation of what the maze looks like, using the characters | and _, with the very last line being the wall size in mm. An example of a maze:



Coloured markers may also be placed in the maze using the following characters:

Marker Colour	Char (no wall)	Char (with bottom wall)
Black	1	!
White	2	@

Grey	3	#
Red	4	$
Yellow	5	%
Green	6	^
Cyan	7	&
Blue	8	*
Magenta	9	(

5.4 .sim files

A sim file is used to store configurations, for quickly repeating experiments, by setting up the environment and objects. A sim file can consist of the following keywords:

Keyword	**Arguments Specification** **Example**
world	Path to a .wld or .maz file "\Worlds\MyWorld.wld"
robi	Path to a .robi file "\Robots\MyRobot.robi"
object	Path to an .esObj file "\Objects\MyObject.esObj"
object_name	Name of an actual object, position x y (in mm) rotation phi (in deg), path to executable (if a robot) labbot 1000 1000 0 "\Programs\Drive.x" can 1500 1500 0 Or in the case of manipulators:

	Keyword ‘manipulator’, path to robi file, position x y z (in mm), rotation alpha beta gamma (in deg), vehicle id to mount on (1st vehicle in sim file is id 1), path to executable manipulator “Robots\MyManipulator.robi” 100 0 0 0 45 45 mount 1 theta_set.x
marker	Position of marker x y (in mm), color of marker r g b (0 to 255) 1000 1000 255 0 255
settings	Toggle various in-game settings. VIS – Enable PSD visualization for all robots TRACE – Enable path tracing for all robots Parameters can be used together, e.g.: ‘settings VIS TRACE’

Note that for object_name, the keyword is the name of the object itself (ie labbot, s4, can, etc.) This can also be the name of a robot or object that has been loaded previously with the robi or obj keywords.

6 BUG REPORTING

If you have detected any bug or considered any function missing or needed to be improved while using this EyeSim-VR simulator, you are highly encouraged to report the bug or suggestion to us for further corrections and improvements.

We are using Bugzilla as an online bug-reporting system, you will see the main page of Bugzilla as shown in figure 5-1 by accessing following link: http://robotics.ee.uwa.edu.au/bugzilla/.

Figure 5-1

6.1 Create an Account/login

Create account

If it is the first time you are using this system, you should open a new account by clicking on the ***Open a New Account*** icon, then follow the steps to create your account. (note: sometimes it may take a few minutes for Bugzilla to send the registration email, please be patient). you will automatically get logged in after you have created your account.

Log in

You can log in directly if you have an account through the ***LogIn*** menu.

6.2 File a Bug

After logging in, you can click the File a Bug icon to report a bug. Then after clicking on **Robot-VR** as selected product, you will see a report form as shown in figure 5-2. Please select the specific component that is related to the bug, and **Severity** of the bug, **Hardware type** of the computer you are using and the **OS**.

In the **Summary** section, you are supposed to write a succinct

sentence summarizing the bug.

In the **Description** section, the details of the bug, like when it occu4rs, what is the setup of environment, what are the previous actions you have performed, what you suspect has led to the bug should be presented.

In the **Attachment** section, you can add attachment like a snapshot of the bug, the error message or some other supporting files. Then you can click on Submit Bug button to submit the bug report.

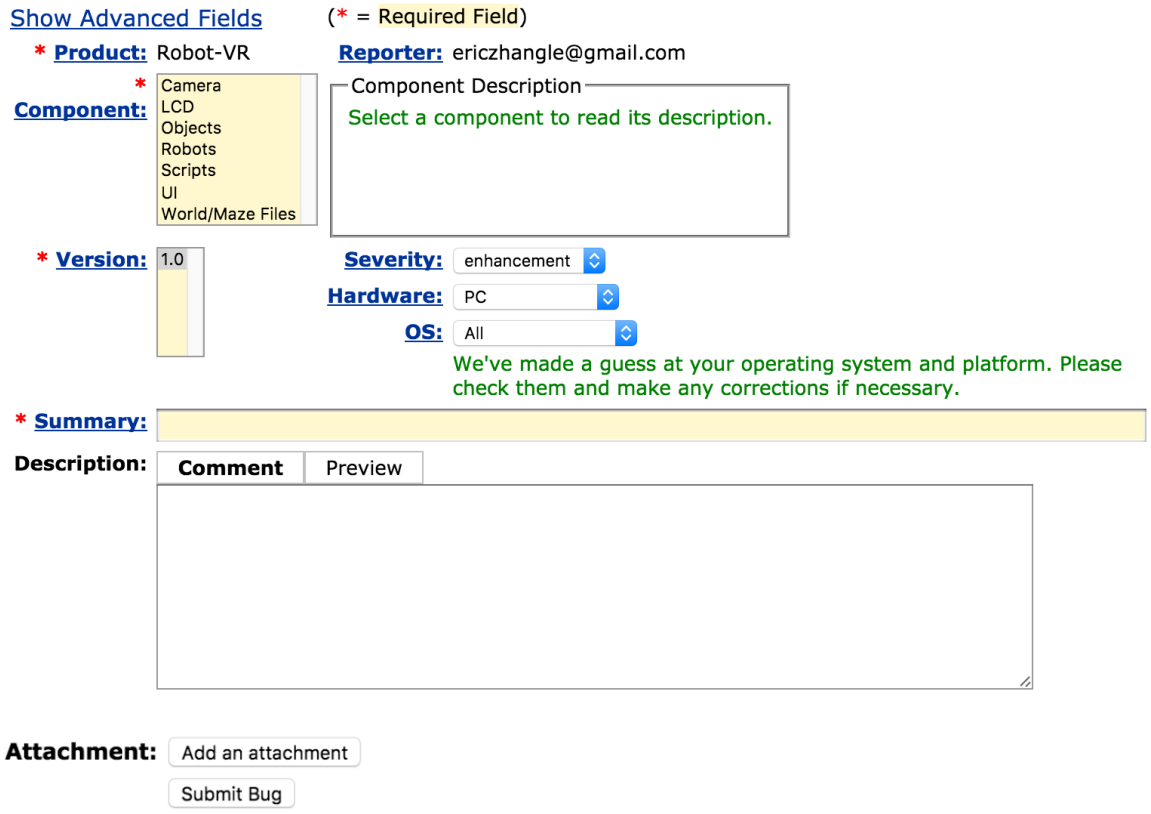


Figure 5-2 Bug report form

Appendix A

RoBIOS-7 Functions Not Supported List

Servos and Motors (2 of the functions in RoBIOS-7)

```
int ENCODERRead(int quad);                    // Read quadrature encoder [1..4]
int ENCODERReset(int quad);                   // Set encoder value to 0 [1..4]
```

USB/Serial Communication (all functions in RoBIOS-7)

```
int    SERInit(int interface, int baud,int handshake); // Init communication (see parameters below), interface
number as in HDT file
int    SERSendChar(int interface, char ch);           // Send single character
int    SERSend(int interface, char *buf);             // Send string (Null terminated)
char SERReceiveChar(int interface);                   // Receive single character
int    SERReceive(int interface, char *buf, int size); // Receive String (Null terminated), returns number of
chars received
bool SERCheck(int interface);                         // Non-blocking check if character is waiting
int    SERFlush(int interface);                       // Flush interface buffers
int    SERClose(int interface);                       // Close Interface
```

Digital and Analog Input/Output (all functions in RoBIOS-7)

```
int DIGITALSetup(int io, char direction);     // Set IO line [1..16] to i-n/o-ut/I-n pull-up/J-n pull-down
int DIGITALRead(int io);                      // Read and return individual input line [1..16]
int DIGITALReadAll(void);                     // Read and return all 16 io lines
int DIGITALWrite(int io, int state);          // Write individual output [1..16] to 0 or 1
int ANALOGRead(int channel);                  // Read analog channel [1..8]
int ANALOGVoltage(void);                      // Read analog supply voltage in [0.01 Volt]
```

```
int ANALOGRecord(int channel, int iterations);    // Record analog data (e.g. 8 for microphone) at 1kHz (non-blocking)
int ANALOGTransfer(BYTE* buffer);                 // Transfer previously recorded data; returns number of bytes
```

IR Remote Control (all functions in Robios-7)

```
int IRTVGet(void);                                // Blocking read of IRTV command
int IRTVRead(void);                               // Non-blocking read, return 0 if nothing
int IRTVFlush(void);                              // Empty IRTV buffers
int IRTVGetStatus(void);                          // Checks to see if IRTV is activated (1) or off (0)
```