THE UNIVERSITY OF WESTERN AUSTRALIA

Achieve International Excellence

Robot Simulators and the Porting of EyeSim from Windows to Mac

Advanced Embedded Systems Group

Author: Jessen Harry Beinart

Supervisor: Professor Thomas Bräunl

27/10/2014

Abstract

The Advanced Embedded Systems project’s goal is to update the EyeBot hardware and software. EyeSim is the Eyebot’s simulator; a multiple mobile robot simulator that runs Eyebot programs and models a robots movement and sensors. The purpose of this project involves the porting of the EyeSim simulator from Windows to Mac Operating Systems. A brief overview of simulators, their history and operation is covered. An in-depth look at the workings of the EyeSim simulator is presented. The author provides the porting process, including version control, the installation of the latest versions of open source libraries, bugs and their solutions and debugging methods. The final results are presented, along with suggestions for further work for the updating of EyeSim.

Acknowledgments

The author would like to thank:

His supervisor Professor Thomas Bräunl, from whom I have learnt a lot over the course of the project.

His fellow Advanced Embedded systems team members: Thomas Smith and Stuart Howard for the help they gave him.

His friends and family for the support they have given him throughout the project.

Table of Contents

Introduction

The EyeBot, shown in Figure 1, is a small programmable mobile robot with a number of on-board sensors. It is used in research and teaching at UWA. It has also been used in a variety of other configurations such as walking, aerial and underwater robots. The previous version of EyeBot is controlled using the 'EyeCon', a 32 bit Motorola M68332 embedded controller that provides an interface for a digital camera (Bräunl, 2008). This is old and slow and needs to be updated. The new EyeBot controller has been chosen: the Raspberry Pi. The overall goal of this project is to modernize the EyeBot software and hardware to make it more functional, user friendly and up to date.

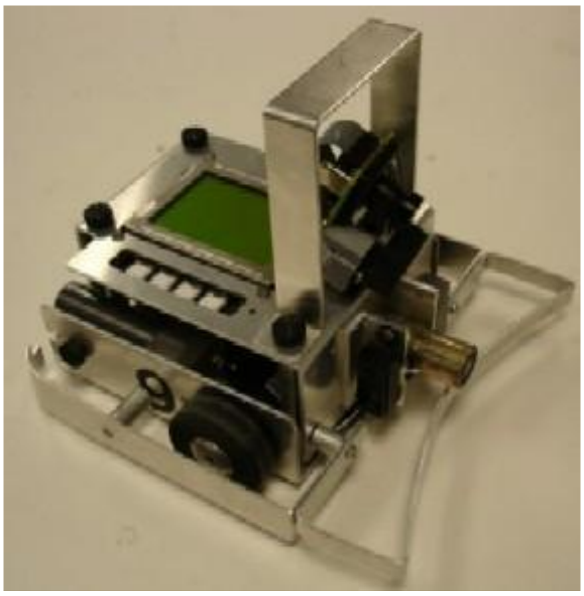
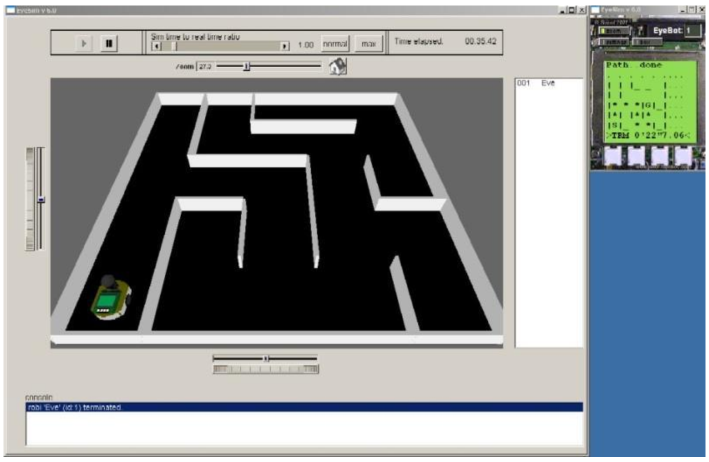


Figure 1: The EyeBot (Bräunl, 2008)

To achieve this goal a number of tasks need to be completed, namely:

- Port the operation of the EyeBot from the Motorola controller to the Raspberry Pi, as the Raspberry Pi is faster, newer and more robust.
- Update the operating system library functions (called RoBIOS), particularly the camera functions, to accommodate the new onboard computer.
- Manufacture a USB expansion board to interface between the Raspberry Pi and inputs/outputs.
- Update the EyeBot Simulator (EyeSim) to reflect the physical changes to the EyeBot.

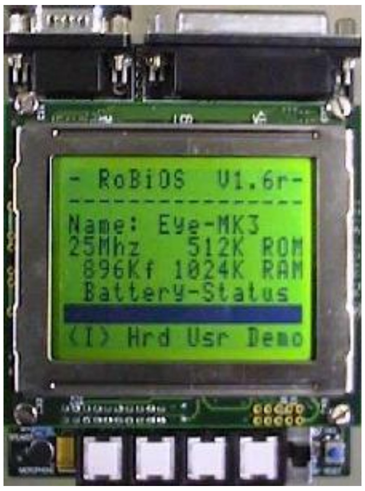
The purpose of this particular project is updating the EyeSim simulator.



```
robi 'Eve' (id 1) terminated
```

Figure 2: A view of EyeSim on Windows (Bräunl, 2008)

The current version of EyeSim (version 6) runs on the Windows operating system, and its display is shown in Figure 2. The updated version is required to run on both Windows and Mac OS X. This will allow more people to access the software. The simulator's Graphical User Interface (GUI) also needs to be updated to reflect the changes in the new EyeBot GUI. The old GUI is displayed in Figure 3. The Raspberry Pi interfaces with a larger colour touch screen compared to the black and white non-touch screen of the previous version. Finally the changes to the RoBIOS library functions and new functions needed to be implemented in the simulator.



```
- RoBiOS  V1.6r-
----------------
Name: Eye-MK3
25Mhz     512K ROM
896Kf 1024K RAM
Battery-Status
(I) Hrd Usr Demo
```

Figure 3: A view of the old EyeBot menu (Bräunl, 2008)

Literature Review

Simulators are an important part of robotic study, research and development (Aaron Staranowicz, 2011). They play a role in static manipulators, mobile robots and manufacturing processes in automated factories. A number of commercial and open-source robot simulators are available to the public.

In recent years there has been an increased need for robot simulators that are accurate and robust and easy to use, that will simulate robots, sensors, errors, and control in virtual environments (Bräunl, 2008). This need is driven by the rise of modern robotic applications which require autonomous operation, many in close proximity to humans for example in human-robot interaction (Murphy, 2004), medical robotics (Dan Stoianovici, 2003) or in robotics for assistive environments (Kim, 2009) In these cases, as well as many others it is important to test the efficiency, robustness and safety of new algorithms by dependable and realistic methods. Other uses are for algorithm optimization and educational purposes, where real robots may be impractical or too expensive.

Robot simulators have a number of uses, namely:

- Testing robustness and performance of robot design by changing specific environmental conditions (Aaron Staranowicz, 2011).
- Simulators allow for the conducting of robot research without the need for actual robots. Studies can be carried out even if the physical model does not actually exist (Neto, 2010).
- Simulations allow for rapid prototyping. The entire method of building a robot can change significantly through the use of simulations. Simulations can deal with and make "on the fly" changes to a robots build without the need to physically rebuild the entire robot. This is more time and cost effective.

The benefits of robot simulation are:

- They allow for reduced cost (Neto, 2010). Physical robots can be expensive and simulations may be adequate to achieve desired objectives.
- Simulators can perform thousands of repetitions, faster than real world applications where inertia must be overcome. This environment is useful to program and model neural networks and genetic algorithms. These need to gather sizeable amounts of training data. Real world vehicle runs can be difficult to train in, as they can take long, and might be dangerous (Bräunl, 2008).
- Simulation can be executed in a "perfect" environment with specified sensor and actuator errors. This allows for testing of a robot and algorithms for robustness in a near real-word scenario (Bräunl, 2005). This also allows for proof of concept in

ideal conditions.

- The simulation of hundreds of robots can be executed (Webots, 2014). This allows for the development of swarm algorithms.

The difficulties robot simulators present are:

- The making of unrealistic assumptions about the robot, the environment or the mechanics.
- The obscuring of a real problem a robot might face.
- Simulators are different to the real world. For example simulating balancing robots is impractical as there are complex mechanics and too many factors at play. The disparity between the real world and simulators means that a model that can balance on the simulator might not balance in the real world and vice versa.

The robotics community has acknowledged the need for more open-source robotic simulator and interfacing software (Knoll, 2009) (Bruyninckx, 2008), and has started to increase its interest in these tools and their applications (Aaron Staranowicz, 2011).

Detailed below are some popular robot simulators, a brief description and their relation to EyeSim:

Player/Stage

Player/Stage (PS) (Player Project, 2014) is a free open-source software project, released under the GNU General Public Licence. It provides tools for single and multi-robot interface and control (Aaron Staranowicz, 2011). Player supports a large range of robotics platforms and sensors. Player is available for Windows, Linux and Mac OSX.

Stage allows for 2-D simulation of robotic platforms and sensors. Stage's robotic models are computationally economical, and therefore allow many robots to be simulated at the same time, as seen in Figure 4. Stage has the potential to simulate hundreds of thousands of robots at the same time (Aaron Staranowicz, 2011). Stage attempts to run at real time (B.P Gerkey, 2003), but will run slower if robotic models take longer to update. Stage is available on Mac and Linux. Stage relates to EyeSim in that EyeSim is also a multi robot simulator, useful for indoor environments. Both Stage and EyeSim rely on FLTK library for the graphical user interface.

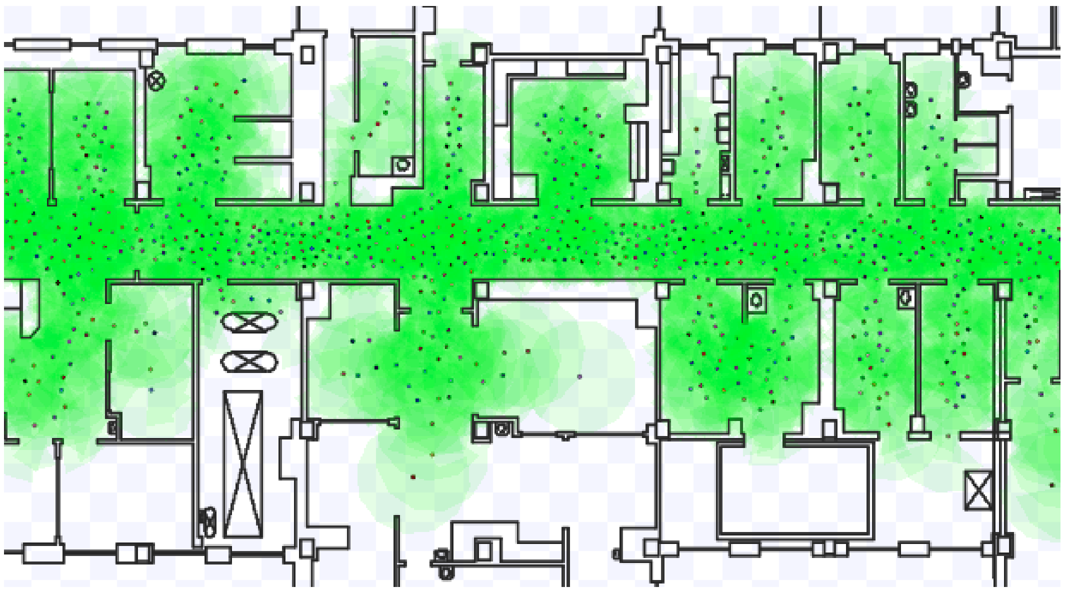


Figure 4: A view of Stage (Player Project, 2014)

Gazebo

Gazebo (Gazebo, 2014)is a 3-D simulator, a view of which is presented in Figure 5. Gazebo has been developed to be used in conjunction with Player. Gazebo can utilize the functions from Player/Stage without modifications. Gazebo uses an Object-Orientated Graphics Engine (OGRE) (Assaf Raman, 2014)and an Open Dynamics Engine (ODE) (Smith, 2014) to render 3-D environments, objects and robots. This allows Gazebo to accurately replicate the environment a robot may confront (Howard, 2004). All simulated objects have mass, friction and other characteristics, that allow for realistic behaviour upon interaction e.g. if knocked or pulled etc. Gazebo is available on both Mac and Linux. Gazebo uses OpenGL for the rendering graphics; similarly EyeSim uses GL or OpenGL for the same purpose.

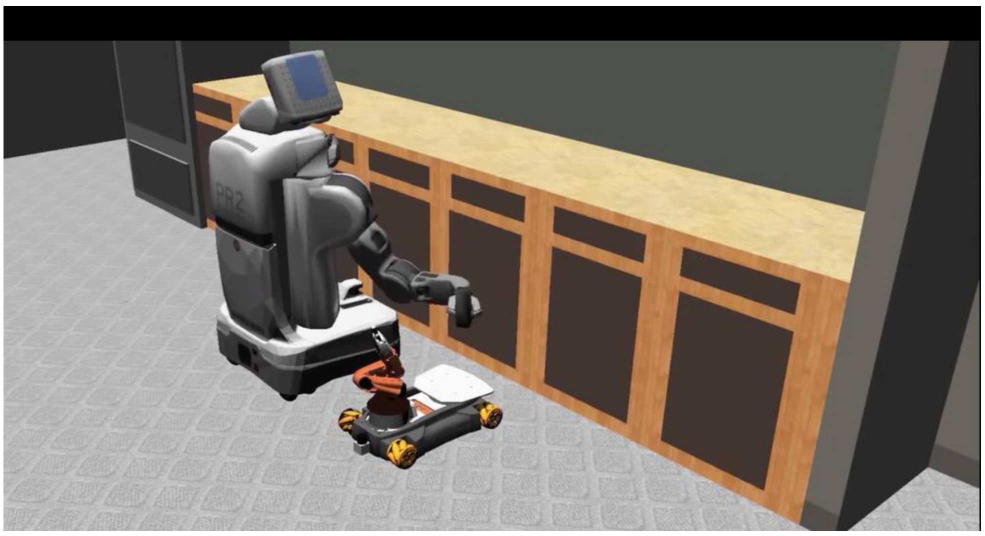


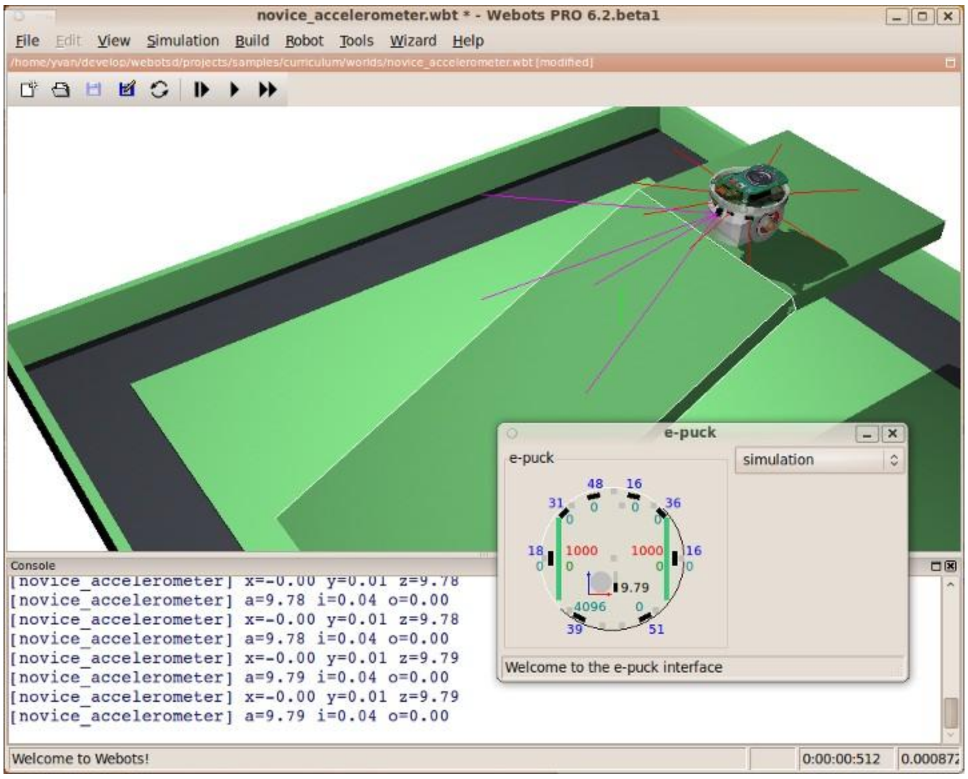
Figure 5: A view of Gazebo (Gazebo, 2014)

ROS

Robot Operating System (ROS) (ROS, 2014)is an open-source operating system that provides hardware abstraction, message-passing between processes, low-level device control and package management. ROS has a growing community of researchers that contribute to the expansion of ROS through code repositories which are available on its website (Aaron Staranowicz, 2011). ROS supports Python, C++, Octave and LISP (Quigley, 2009). Control code is implanted in ROS using nodes, messages, topics and services to perform task. ROS is available on Mac, Linux and with limited functionality on Windows.

Webots

Webots (Webots, 2014)is a commercial 3-D simulator, a view of which is depicted in Figure 6. Models have many customizable attributes such as their shape, mass, friction and texture. They are provided by the Open Dynamics Engine (ODE) Library (Smith, 2014). Webots can simulate mobile, humanoid and quadruped robots as well as a variety of sensors. A view of a mobile robot model is shown in Figure 7. Webots can transfer programs directly to real world robots via Bluetooth (Aaron Staranowicz, 2011). Programs are coded in C, C++, Java, Python and Matlab. Webots is available on Windows, Mac and Linux.


novice_accelerometer.wbt * - Webots PRO 6.2.beta1
File
Edit
View
Simulation
Build
Robot
Tools
Wizard
Help
/home/yvan/develop/webotsd/projects/samples/curriculum/worlds/novice_accelerometer.wbt [modified]
e-puck
e-puck
simulation
48
16
31
36
18
16
1000
1000
9.79
4096
39
51
Console
0:00:00:512
0.00087

```
[novice_accelerometer] x=-0.00 y=0.01 z=9.78
[novice_accelerometer] a=9.78 i=0.04 o=0.00
[novice_accelerometer] x=-0.00 y=0.01 z=9.78
[novice_accelerometer] a=9.78 i=0.04 o=0.00
[novice_accelerometer] x=-0.00 y=0.01 z=9.79
[novice_accelerometer] a=9.79 i=0.04 o=0.00
[novice_accelerometer] x=-0.00 y=0.01 z=9.79
[novice_accelerometer] a=9.79 i=0.04 o=0.00
```

Welcome to the e-puck interface

Welcome to Webots!

Figure 6: A view of the Webots interface (Webots, 2014)

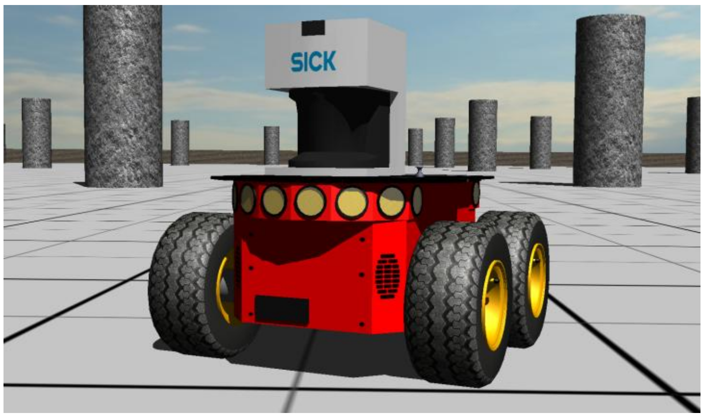


Figure 7: A Webots mobile robot model (Webots, 2014)

TeamBots

TeamBots (TeamBots, 2014) is an open-source 2-D mobile robot simulator that specializes in multi-robot control systems. It has been used to simulate robot soccer, maze solving and capture the flag competitions as well as navigation of city roads. The soccer setup is displayed in Figure 8. Programs can be ported directly to the Nomadic Technologies' Nomad 150 robots. TeamBots runs in a java environment, which has attracted many complaints. Despite these criticisms, developers have found that it runs fast enough for its intended purposes. TeamBots is available on Windows, Mac and Linux. EyeSim is similar to TeamBots in that EyeSim programs can run on the EyeBot unchanged, as well as possessing a top down view.

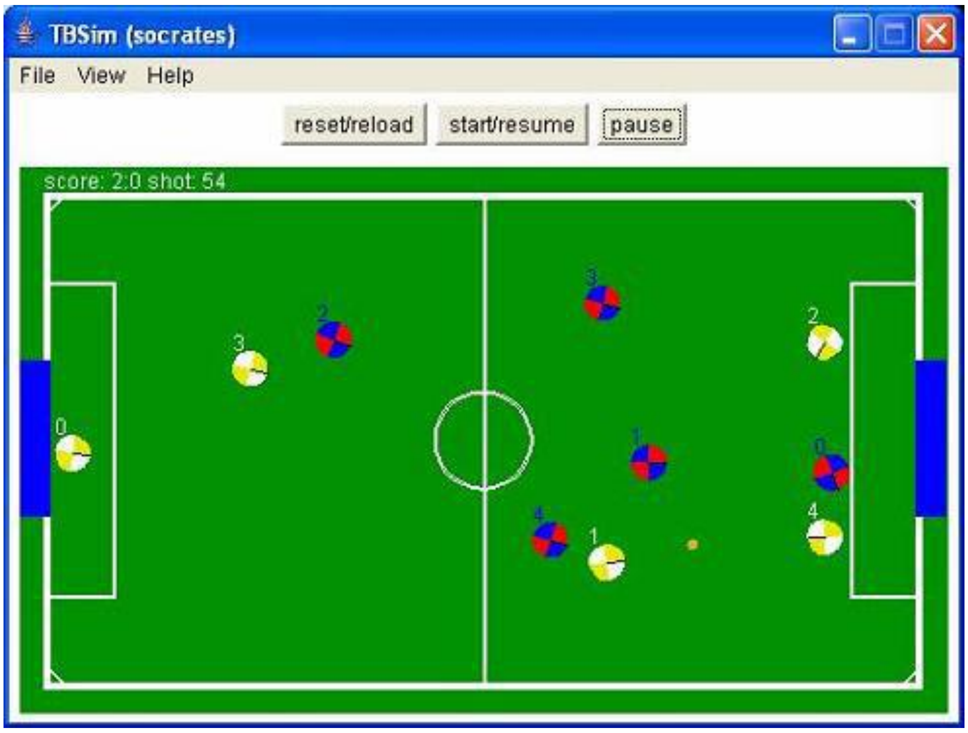


Figure 8: TeamBots soccer map (TeamBots, 2014)

Below in Table 1 is a comparison of a number of robot simulators

Table 1: Characteristics of a Range of Robotic Simulation Software

	License Type	OS	Simulator Type	Programming Language	Portability	Sensors	Graphical User Interface
Webots	Commercial	Linux, Mac, Win	3-D	C, C++, Java, Matlab, Python	Yes	Odometry, range, camera, GPS	Yes
Player/Stage	Open-Source	Linux, Mac, Win	2-D	Player(any) Stage(C, C++, Python, Java)	Yes	Odometry, range,	No
Gazebo	Open-Source	Linux	3-D	C, C++, Python, Java	Yes	Odometry, range, camera	No
ROS	Open-Source	Linux, Mac, Win	2-D, 3-D	C++, Python, Octave, LISP, Java, Lua	Yes	Odometry, range, camera	No
Simbad	Open-Source	Linux, Mac, Win	3-D	Java	Limited	Vision, range, contact	No
CARMEN	Open-Source	Linux	2-D	C, Java	Yes	Odometry, range, GPS	No
USARSim	Open-Source	Linux, Win	3-D	C,C++,Java	Yes(Using Player)	Odometry, range, camera, touch	Yes
MRDS	Open-Source	Win	3-D	VPL,C#,Visual Basic, JScrpit,Iron-Python	Yes	Odometry, range, camera	Yes
MissionLab	Open-Source	Linux	3-D	VPL	Yes	Odometry, range,	Yes
TeamBots	Open-Source	Linux, Mac, Win	2-D	Java	Yes	Odometry, range	No
EyeSim	Open-Source	Mac, Win	2-D	C,C++	Yes	Odometry, range, camera	Yes

A detailed examination of EyeSim

EyeSim uses a copy of a real robots API (application programming interface) and features a simulation of all its sensors and actuators, adjustable error models and the generation of a virtual camera image. The system makes use of the RoBIOS (Robot Basic Input Output System) API. EyeSim has been used for time-consuming tasks, such as experiments in Neural Networks and Genetic Programming (Bräunl, 2008), (J Du, 2003)

EyeSim implements the two levels of driving (Andreas Koestler, 2004), that are available on RoBIOS as follows:

- A High level driving controller for basic driving operations such as driving in straight and curved segments, and on the spot turning.
- A low level driving system for simulation of motor actuators, shaft encoders and feedback for different driving models such as differential drive, Ackerman steering and omni-directional drive.

EyeSim differs from most simulation implementations (Braunl, 1997)that run simulations as separate programs or processes that communicate with the application via a message passing. EyeSim dynamically links the simulation program at run-time and provides all API system functions for reading data from sensors and motors. The system relies on FLTK for the GUI (graphical user interface), Commoncpp for cross platform file operations and OpenGL for graphics.
For each simulation, the system accepts a number of parameters. A number of optional parameters are detailed in Figure 2below.

Table 2: EyeSim Parameters

Parameter	**Function**
-r simTimeRatio	Sets the simulation to real time ratio
-s time	Starts the simulation after specified number of seconds
-d	Enables debugging mode
-m	Start simulation with minimized GUI

The simulation also requires a "Sim File" (simulation file), as seen in Figure: 9 below, which is a file that again contains a number of parameters.

```
include ../path.txt


# world description file (either one maze one world)

world  %sim%/worlds/worlds/Soccer1998.wld

# additional obstacles: x,y,theta, friction parameter, image file

object  800  800   0  0.0 %sim%/worlds/objects/can/can.ms3d


# robi description file
robi    %sim%/robots/SocBot/S4.robi  DriveDemo.dll  400 400 0
```

Figure: 9 EyeSim Sim File

These parameters include a compulsory environment file path, a compulsory "Robi" file path accompanied by an optional location parameter and a compulsory simulation program (dynamic library) file path. It may also contain optional object files and location parameters (Andreas Koestler, 2004)
Environment files describe the driving scenery and can be of the type "world", or the simpler "maze" type. Robi files describe the robot type and change the graphical model displayed. Objects are .ms3d (MilkShape 3-D) files.

Scope of Project

The aim of the project is simple: the porting of EyeSim from the Windows operating system to the Mac operating system.

The scope of the project is:

- Achieve the same functionality on Mac as the Windows version. This requires all demo programs to run on Mac, the buttons on the simulator to work and the way the simulator is started to be the same.
- EyeSim should be cross compatible between operating systems. There should be an easy way to move from compiling the source code on Windows to Mac and vice versa. The way functions are implemented on different machines

should be similar. Different methods should not be used to accomplish the same task on different machines.
- Maintaining version control. The same files should be used for each function, no matter which machine it is running on. It is too arduous and inefficient to keep track of different files and methods for each version of EyeSim.

Process

The process of porting EyeSim from Windows to Mac involved the following steps:

1) Updating and installing open source libraries
2) Editing Makefiles
3) Editing EyeSim source code
4) Porting scripts
5) Debugging executable

The objectives of this cross compilation process were:

- To minimise the differences between the platforms.
- Have only one file for each task e.g. one simulation file instead of one Windows simulation file and one Mac simulation file.
- All platforms work in the same way.
- Achieve same functionality across all platforms.

Method

This project was completed on a Mac OS X Mavericks (10.9.5). Debugging was done using ‘Lldb’. ‘Gdb’, the program previously used, has been replaced by Lldb on Mac. Debugging was done by first localizing the glitch in the source code. Breakpoints were set at these locations to further find the lines of code causing problems. Once the problem was identified further inspections were made, either into local variables, using Lldb, or research on the functions used was carried out.

Open Source Libraries

EyeSim relies on two open source libraries: Commoncpp and FLTK. Open source libraries are helpful as they provide extra functionality for free. A problem they do present is currency. Open source libraries have the possibility of becoming out of date due to a lack of maintenance. The Application Program Interfaces (API’s) that the unmaintained software relies on may change or be replaced. This results in incompatibility. There are a number of solutions to this issue. These are:

1) Pay for a library that is updated
2) Write your own library which is time consuming, or
3) Find a new open source library that is updated.

Commoncpp

The Commoncpp framework was utilized in EyeSim for portable threading and file operations (Yurii Rashkovskii, 1998)The version (Commoncpp2-1.6.2) had known problems compiling on OS X so the most up to date version was used (Commoncpp2-1.8.2). Despite Commoncpp claims of portability, it provided compilation problems. The latest update to the framework was November 2010 (GNU, 2014). Changes to the source code were made to allow for compilation. These changes are detailed in Appendix A. Following these changes the library was built and installed.

FLTK

FLTK (pronounced "fulltick") is a cross-platform C++ GUI toolkit (Spitzak, 2014). The latest version of FLTK, released December 2012, was used, as the current version was not compatible with the updated Commoncpp library. The FLTK source code did not compile at first and, as a result had to be debugged. The changes made to the source code are detailed in Appendix A. After these changes the library was built and installed.

Makefile

After the open source libraries compiled, a new folder, entitled ‘mac’, was added to the EyeSim source code directory and the Makefile was edited. Changes were made to the names of the updated libraries, as well as dependencies. The Windows GL libraries were changed to OpenGL, which are available on Mac.

Editing the EyeSim source code

The next step was to compile the EyeSim source code, with the new Makefile. Detailed below are the problems that were encountered and their solutions that were developed.

1) The GL library is available on Windows, but not on Mac. The Mac equivalent is OpenGL. All instances of GL header files needed to be appended. The way this was done, for cross compilation, was as follows:

```
#if defined (_WIN32)
#include <GL/glu.h>
#else
#include <OpenGL/glu.h>
#endif
```

This issue occurred in the following files:

- engine/meTexture.h
- engine/meViewport.h
- src/Main3DView.h
- engine/meSphere.h
- engine/me.h

2) Both engine/me.h and engine/meComponent.h included the header windows.h, which is not available on Mac. This was dealt with in a similar way to the way the GL/OpenGL problem was resolved as follows:

```
#if defined (_WIN32)
#include <windows.h>
#endif
```

3) At the top of some header files there was the macro definition:

```
 #define WIN32 1
```

This was removed to allow for quick cross compilation. It is inefficient to change these macros every time you change machines. Any dependencies on this macro were changed to "_WIN32" or "_APPLE_".

The following files had the WIN32 definition removed:

- engine /me.h
- engine /me3DVector.h
- engine /meBMPImageReader.h
- engine /meCamera.h
- engine /MeCollision.h
- engine /meColor.h
- engine /meContainer.h
- engine /meCuboid.h
- engine /meGenVector.h
- engine /meTexture.h
- engine /meVeiwpoint.h
- engine /meWorldSettings.h

4) In engine/meComponent.h (line 85) and src/sim.h (line 716) single lines of code did not compile on Mac. Conditional compilation was used to check if the operating system was Windows.

In src/Synchronization.cpp (line 94) the following code was added:

```
#define EXPORT_FUNC extern "C" __declspec(dllexport)
```

Line 106 was changed from:

```
pthread_mutex_t temp ={0, 0, 0, PTHREAD_MUTEX_RECURSIVE/*_NP*/,
__LOCK_INITIALIZER};
```

to:

```
pthread_mutex_t temp =PTHREAD_MUTEX_INITIALIZER;
```

After these changes EyeSim compiled.

Debugging the EyseSim Executable

After EyeSim compiled there were a number of runtime errors. Detailed below are the problems that were encountered and their solutions that were developed:

1) In src/Synchronization.cpp the CSemaphore() function crashes EyeSim. Windows supports unnamed semaphores such as sem_init() and sem_destroy(), however Mac does not (Singh, 2001). Only named semaphores can be used, therefore the sem_open() and sem_close() functions are used, and replaced the unsupported function.

2) In src/thread adapter.cpp abort() was called terminating with uncaught exception of type ost::DSO*. DSO is the class to dynamically load object files cross platform and is defined in the Commoncpp library (GNU, 2012). This class is utilized for opening and loading the dynamic library simulation file for each robot, on non-Windows machines. The issue was not able to be solved by fixing the direct cause of the problem. The way it was resolved was by implementing a similar approach that the Windows method used. The Windows method used the function LoadLibrary() found in the dlfcn.h header file. The new Mac implementation used the dlopen() function also found in the dlfcn.h header file. The Mac implementation needed to create the function pointer ‘sim_main’.

In Windows it is executed as follows:

```
sim_main = (cfunc) GetProcAddress ( (HMODULE)hLib, "main" );
```

while in Mac it is executed as follows:

```
sim_main= (int(*)())dlsym(hLib,"main");
```

The GetProcAddress() function is defined in the Windows header Winbase.h and there is no corresponding Mac definition. This is where the Commoncpp library should be beneficial, to be used as the Mac equivalent. However, as seen before, the issues could not be resolved and a new approach was taken.

3) Once EyeSim was able to load dynamic libraries a problem arose with the actual libraries. EyeSim would load the library and then try to execute the simulation program, but would exit with the following error:

```
Undefined symbols for architecture x86_64:
  "_KEYGet", referenced from:
     _main in DriveDemo-14a743.o
```

Each time a RoBIOS function was called, EyeSim could not find their definitions, which were defined in the EyeSim application itself in files such as src/rb_lcd.cpp. The difficulty here is the difference in the way .dll and .dylib files work. Dynamic libraries are bound at runtime, whereas static libraries are added at the linking phase as seen in Figure 10 and Figure 11 below.

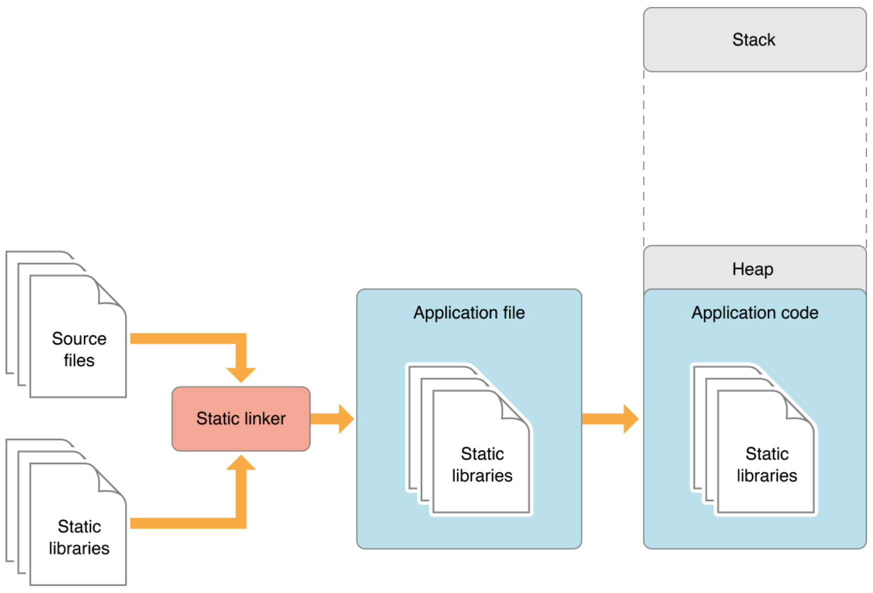


Figure 10: Linking of Static Libraries (Apple, 2014)

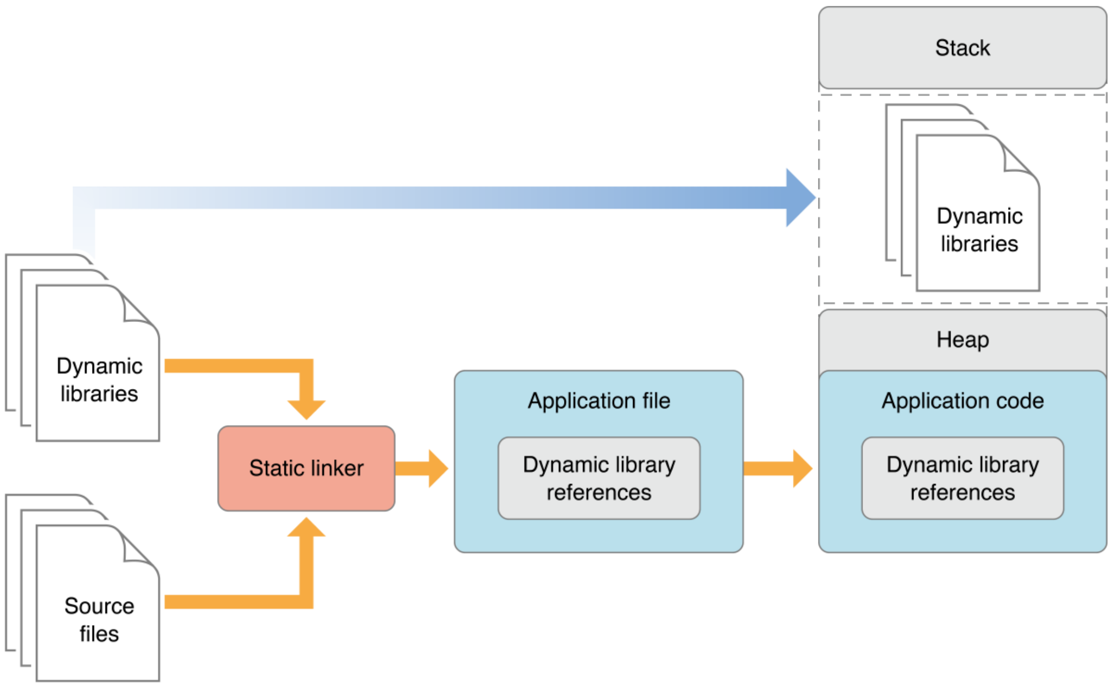


Figure 11: Linking of Dynamic Libraries (Apple, 2014)

Windows uses .dll files while Mac uses .dylib files. The main difference between these is what is visible by default from each file type. Files of type .dylib export the language level linkage, which means symbols that are "extern" are available for linking when .dylibs are pulled in. It also means that, at resolving .dylib files is a link step, the loader doesn't mind which .dylib file a symbol comes from. It just searches the specific .dylib files.

By contrast .dll files are an operating system feature, completely separate to the link step. The .dll file has no dangling references. Instead, an admission to methods, functions and data goes through a lookup table. This means that DLL code does not have to be fixed up at runtime to refer to the program's memory. Instead, the lookup table is changed at runtime to point to the functions and data.

To create the .dylib files the following lines of shell script were used:

```
g++ shared -fno-common -Wall -dynamiclib -undefined suppress -
flat_namespace -I../include $*
```

Sim.h had the following code added at line 47:

```
#elif defined __APPLE__
#define EXPORT_FUNC extern "C" __attribute__ ((visibility ("default")))//
#else
```

This code segment ensured that the symbols in the dynamics libraries were visible.

A lot of care needs to be taken when creating .dylib files. Use of the otool and install_name_tool are important Terminal utilities to examine the visibility of symbols and ensure the correct install paths and versions. These tools are discussed in the section 'Installing EyeSim'.
'Installing EyeSim'.

4) EyeSim then crashed with the exception type 'EXC_BAD_ACCESS (SIGSEGV)'. This error is a segmentation fault, and in this case is caused by trying to call an OpenGL command without a 'context'. An OpenGL context is a data structure that stores all the states of an OpenGL instance (Apple, 2014). To remedy this issue a call to CGLGetCurrentContext() is made, which returns the current rendering process. This ensures that any subsequent OpenGL commands have a context associated with them, and do not cause a segmentation fault.

Main3DView.cpp is edited at line 67 in the following way:

```
#ifdef __APPLE__
CGLGetCurrentContext();
#endif
```

The CGLGetCurrentContext() function is defined in OpenGL.h, so this header is included in Main3DView.h as follows:

```
#ifdef __APPLE__
#include <OpenGL/OpenGL.h>
#endif
```

5) When loading World files, mazes were not loading properly. Maze files are text files with symbols representing the maze. The root cause of the problem is that the Windows and the Mac use different newline characters. Windows uses '\n' while Mac uses '\r'. This meant that when the parser came to an '\r' character, it did not treat it as a newline. To fix this, a new case was added to the switch-case statement in the following way, so that the map drawing function would stop drawing at the appropriate time:

In world.cpp line 1422 and line 1270

```
#ifdef __APPLE__
case '\r':
#endif
```

Scripts

Scripts and Makefiles are used to compile the simulation programs,.
Batch files are utilized on Windows machines, but these are not available on Mac devices. However shell scripts are available. A new shell script called gccsimMAC.sh was created, with the same functionality as the gccsim.bat batch file. The updated Makeincl file decides which script to execute.

The Makefile would include the file Makeincl. This script recognizes which operating system it is on and defines common operation. The pseudo code for the Makeincl is as follows:

```
ifeq ($(shell ver),)
 PLATFORM = UNIX
 COPY = cp -f
 RM = rm -f
 TMPDIR = /tmp

else
 PLATFORM = DOS/Windows
 COPY    = copy
 RM      = del
 TMPDIR  = .
endif
```

The line that would decide which Operating System it is currently on, would not work on Mac and so none of the script worked. The command ‘ver’ was changed to ‘uname,’ which provides the operating system name on both Windows and Mac machines (GNU, 2012). Another two definitions were added, namely GCCSIM and LIBSUFFIX. GCCSIM defines the name of the script to run i.e. the gccsim batch script for Windows or gccsimMAC.sh shell script for Mac. To run shell scripts the command ‘sh’ precedes the script name, so for Mac the definition of GCCSIM is:

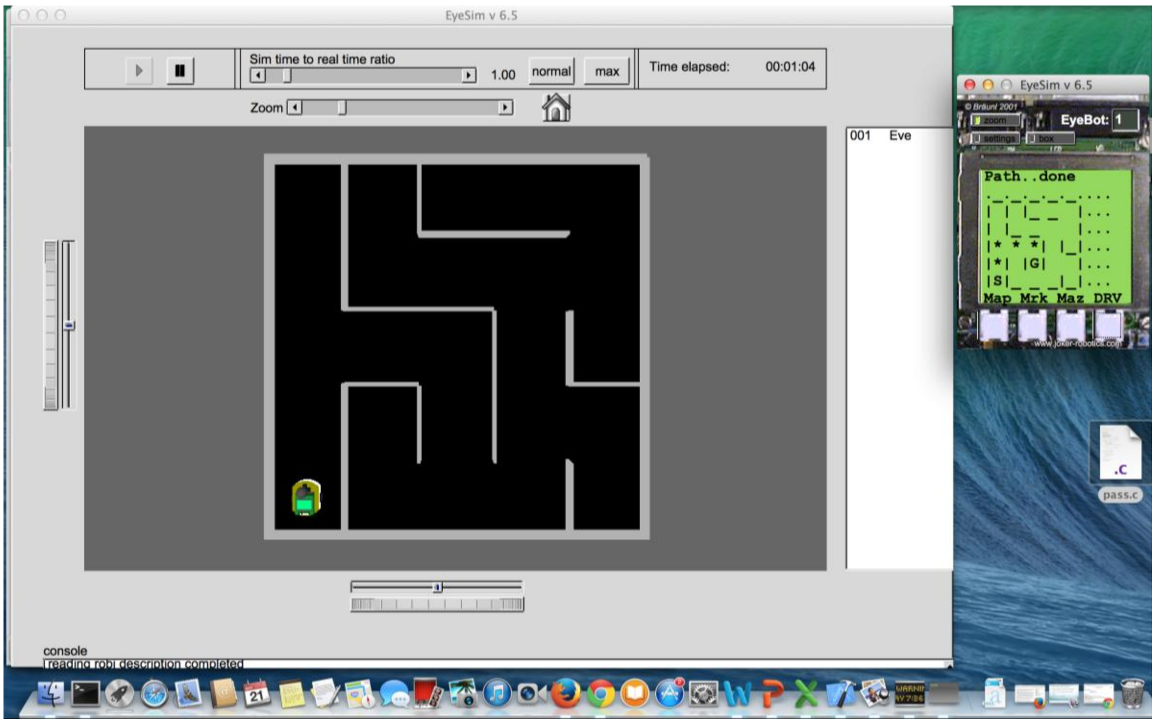
```
GCCSIM = bash gccsimNEW2.sh
```

LIBSUFFIX is the suffix of the dynamic library pertaining to the current Operating System. It is used to create the file names in the Makefile file. On Windows it is defined as .dll and on Mac as .dylib.

Results and Discussion

The open source libraries, FLTK and Commoncpp, were compiled and installed on Mac. Part way through this project, after FLTK was installed correctly, it stopped working. Attempts at trying to build FLTK again produced compilation errors. During this time, Xcode released an update, and it was thought that this update caused the issue.

Porting of EyeSim was successful and it now compiles and runs on Mac as seen in Figure 12 Most functionality has been retained, although with some issues detailed in the sub-section ‘Current Problems with EyeSim on Mac’.



```
reading robi description completed
```

Figure 12: A view of EyeSim running on Mac

The Scripts and Makefiles work correctly. An important goal of cross compilation is using the same file over all machines. This means that only one change needs to be made if there is an update, instead of changing a different file for each platform.

The ‘gccsim’ script is used to work out the linking option needed to create a dynamic library, to plug into EyeSim. For Windows, Batch scripts only need the file name to be entered into the command line in order to execute the script. To run Shell scripts on Mac, the command “sh” or “bash” needs to precede the filename. For Windows the Makeincl file creates the GCCSIM variable as follows:

```
GCCSIM = gccsim
```

On Mac, GCCSIM is as follows:

```
GCCSIM = bash  gccsimMAC.sh
```

On Mac the compiler executes the gccsimMAC.sh script, with the above implementation, as desired.

All the demo programs that are provided with EyeSim compiled on Mac.

The following demonstration programs ran as expected:

- LowLevelDrive
- RandomDrive
- DetectObject
- DriveDemo
- Wireless
- DriveToBall
- MazeSearch

The following programs did not run correctly:

- Cluster
- MultiTask
- Nerual
- Pantilt
- Randomdrive6
- SoccerDemo
- Genetic prog

The reason that these programs did not run correctly is due to the texture problem as discussed below in the 'BitMaps' sub-section. This is more a problem of 3D models not loading, than problems with the programs themselves. These 3D models are objects that are declared in the Sim File.

Current Problems with EyeSim on Mac

On Windows, a click to play icon was created, which launched a simulation command line tool. Clicking on a Sim file would run that simulation. Currently this feature has not been implemented on Mac. In its present state, Eyesim is launched from the terminal in the following way:

```
$./eyesim  SimName.sim
```

Terminal's "iconutil" can be used to create the required icon (Apple, 2014).

Sim files in their current form are not cross-platform compatible. Sim files include the file path.txt, which gives the location of the EyeSim application and the worlds, robots and object files. Path.txt has one line:

```
define sim /path/to/eyesim
```

In batch files, sim is called as %sim%. This works correctly and points to the file location needed. However on Mac, this does not work. When the “sim” variable is called as $(sim), it is interpreted as “$(sim)” and not the contents of the variable sim. A solution to this issue has not been found yet.

BitMaps

The textures for objects and world floors are produced from Bitmap images of file type .bmp. These images are only available on Windows (Microsoft, 2014). The MS3D (MilkShape 3-D) model needs to be changed from using the .bmp file format to a more portable format such as JPEG. This feature is not integral to the running of EyeSim, nevertheless it has a superior look when the floor is the correct colour and objects appear to be more realistic. A MilkShape 3-D licence is needed to save .ms3d files. The floor texture images were saved in JPEG format, but the image would not appear. EyeSim has a member function meBMPImageReader, which reads bitmaps, and an equivalent needs to be created to read other file types. Another project group is currently working on replacing MilkShape with Blender, another modelling software package.

The pause button presently hangs the program when pushed. It should stop the robots from moving, pause the dynamic library program, and pause time. It is not understood why this occurs. It is theorized that there is a problem with the threading, and that the threads become out of sync.

A feature that could be implemented in the future is the ability to manipulate the users viewpoint. Currently the view is constrained to a 2-D top view. A 3-D view would work in a similar manner to the way a simulated robot’s camera field of view is calculated and displayed.

Installing EyeSim

Upon trying to use EyeSim on another Mac computer a number of faults arose, which are all caused by the same problem: dynamic libraries. While making EyeSim on a new Mac, linking errors arose from the dependant dynamic libraries. When trying to run an already made EyeSim, all dependant dynamic libraries were not found. These problems were due to the fact that dynamic libraries need to be ‘installed’ in their current directory. That is if you copy a dynamic library to a new location or machine (thereby changing its absolute path) it has not been installed in the new location. The shell command:

```
$ otool –L foo.dylib
```

is used to investigate a dynamic library and display its install path.

The command:

```
$ install_name_tool –id foo.dylib
```

and

```
$ install_name_tool –change OLD/path/name/foo.dylib
NEW/path/name/foo.dylib
```

are used to install a dynamic library to a directory. Using the 'otool' command once again will verify the install path change.

A script was created to execute this process of changing the install paths of all dependencies so that EyeSim will work on new Macs, as shown in Appendix B.

The way EyeSim is deployed on Mac is by copying the EyeSim executable, the include headers, robot files, world files, object files, compiling scripts, and a dependencies folder. This dependencies folder contains all the .dylib files that EyeSim relies on. Before running EyeSim the 'installScript.sh' script needs to be executed, so that the dynamic libraries are installed to the correct path, and can be used.

Future Work

There are still many facets of EyeSim that need to be upgraded. The graphical user interface (GUI) needs to be upgraded in both its physical appearance and function. The new Raspberry Pi and Beagle Board screens are touch-sensitive and this functionality needs to be implemented. FLTKs DEVICE_TOUCH and DEVICE_MOUSE events can be utilized (Spitzak, 2014).

The size of the screen is different for the Raspberry Pi, Beagle Board and the old EyeBot. This should be represented in the simulator. In the Simulation file, an optional parameter could be constructed to indicate the desired screen size, or use a preset size. Another project group is currently working on this aspect of the simulator.

As the RoBIOS functions are updated to manage the new hardware, so too, the EyeSim's versions of the RoBIOS functions need to change. These are implemented in the ‘rb’ source files e.g. rb_lcd.cpp. It is important to note that the EyeSim’s version of the RoBIOS functions are high level and only apply the result of a function not the inner workings. For example the drive function in the real world turn the motors on and then the robot would move as a result. In the simulator, there are no motors so the robot is manipulated forward. Once again another project group is currently working on this aspect of the simulator.

The next stages of development of EyeSim should be moving away from Commoncpp and FLTK. FLTK can perhaps be replaced with ‘Qt’, a more up-to-date and portable software package. EyeSim can also be ported to Linux and this should not be difficult as Mac OS X is similar to Linux. Other features like 3-D viewing and the option to change viewpoint can be included.

Appendix A

Changes to compile and install Commoncpp-1.8.1 on Mac

In inc/string.h
Line 734 change to:
friend __EXPORT std::istream &getline(std::istream &is, String &str);//, char delim = '\n', size_t size = 0);

Line 115 change to:
public:

In src/applog.cpp
Line 48 add:
#include <sys/types.h>
#include <sys/stat.h>

In inc/file.h
Line 63 add:
#include <cc++/serial.h>

Comment out line 80

Line 153 – 155 change to:
accessReadOnly = O_RDONLY,GENERIC_READ,
accessWriteOnly = O_WRONLY,GENERIC_WRITE,
accessReadWrite = O_RDWR,GENERIC_READ | GENERIC_WRITE

In inc/serial.h
Line 61 change to:
#define INVALID_HANDLE_VALUE   -1

Changes to compile and install FLTK-1.3.2 on Mac

In Type_FL.cpp
Line 39 change to:
friend Fl_Widget *make_type_browser(int,int,int,int,const char *l);

Appendix B

Script to install .dylib libraries - installScript.sh

This script is used to install EyeSim's dependent dynamic libraries.

```
#!/bin/bash

LIBPATH="$( cd "$( dirname "${BASH_SOURCE[0]}" )" && pwd )"

DYLIBS="libfltk_forms.dylib libfltk_gl.dylib libfltk_images.dylib libfltk.dylib
libccgnu2.dylib libccgnu2-1.8.0.dylib"

for DYLIB in ${DYLIBS} ; do
LIBFILE=${LIBPATH}/${DYLIB}
DYLIBID=`otool -DX ${LIBPATH}/$DYLIB`
NEWDYLIBID=${NEWLIBPATH}/${DYLIB}
install_name_tool -id ${NEWDYLIBID} ${LIBFILE}
install_name_tool -change ${DYLIBID} ${NEWDYLIBID} ../eyesim
done
```

Bibliography

Aaron Staranowicz, G. L. M., 2011. A survey and comparison of commercial and open-source robotic simulator software. p. 56.
Andreas Koestler, T. B., 2004. Mobile Robot Simulation with Realistic Error Models.

Apple, 2014. *Dynamic Library Programming Topics: Overview of Dynamic Libraries.* [Online]
Available at:
https://developer.apple.com/library/mac/documentation/developertools/conceptual/dynamiclibraries/100Articles/OverviewOfDynamicLibraries.html#//apple_ref/doc/uid/TP40001873-SW2
[Accessed September 2014].

Apple, 2014. *OpenGL Programming Guide for Mac: Working with Rendering Contexts.* [Online]
Available at:
https://developer.apple.com/library/mac/documentation/graphicsimaging/conceptual/opengl-macprogguide/opengl_contexts/opengl_contexts.html#//apple_ref/doc/uid/TP40001987-CH216-SW12
[Accessed August 2014].

Assaf Raman, J. B., 2014. *OGRE – Open Source 3D Graphics Engine.* [Online]
Available at: http://www.ogre3d.org/
[Accessed 4 10 2014].

B.P Gerkey, R. V. a. A. H., 2003. The player/stage project: Tools for multi-robot and distributed sensor systems. *Proceedings of the 11th International Conference on Advanced Robotics,* pp. 317--323.

Bräunl, K. W., 2005. Mobile Robots between Simulation and Reality. *Servo Magazine,* 3(1), pp. 43-50.

Braunl, S., 1997. Mobile robot simulation with sonar sensors and cameras. *Simulation,* 69(5), pp. 277--282.

Bräunl, T., 2008. *Embedded robotics: mobile robot design and applications with embedded systems.* Berlin: Springer.

Bruyninckx, H., 2008. Robotics software: The future should be open. *IEEE Robotics Automation Magazine.*

Dan Stoianovici, R. H. T., 2003. Medical robotics in computer-integrated surgery. *Robotics and Automation, IEEE Transactions on,* 19(5), pp. 765--781.

FLTK, 2014. *Download - Fast Light Toolkit (FLTK).* [Online]
Available at: http://www.fltk.org/software.php
[Accessed 2014].

Gazebo, 2014. *Gazebo.* [Online]
Available at: http://gazebosim.org/
[Accessed 1 10 2014].

GNU, 2012. *GNU Bayonne 2: ost::DSO Class Reference.* [Online]
Available at: http://www.gnutelephony.org/doxy/bayonne2/a00083.html
[Accessed October 2014].

GNU, 2014. *Commoncpp Change Log.* [Online]
Available at: http://www.hyperrealm.com/commoncpp/ChangeLog.txt
[Accessed 2014].

Howard, N. K., 2004. Design and use paradigms for gazebo, an open-source multi-robot simulator. *IEEE/RSJ International Conference on Intelligent Robot and Systems,* Volume 3, pp. 2149--2154.

J Du, T. B., 2003. Collaborative Cube Clustering with Local Image Processing. *Proc. of the 2nd Intl. Symposium on Autonomous Minirobots for Research and Edutainment.*

Kim, L. B., 2009. An empirical study with simulated adl tasks using a vision-guided assistive robot arm. *IEEE International Conference on Rehabilitation,* pp. 504--509.

Knoll, H., 2009. *Workshop on Open-Source Software.* [Online]
Available at: http://www.openrtp.jp/icra09_workshop/
Michael Abbott, R. A. e. a., 2014. *Building C and C++ Extensions on Windows.* [Online]
Available at: https://docs.python.org/2/extending/windows.html#differences-between-unix-and-windows
[Accessed June 2014].

Microsoft, 2014. *BITMAP structure (Windows) - Msdn.microsoft.com.* [Online]
Available at: http://msdn.microsoft.com/en-us/library/windows/desktop/dd183371(v=vs.85).aspx
[Accessed 22 9 2014].

Murphy, R. R., 2004. Human-robot interaction in rescue robotics. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on,* 34(2), pp. 138--153.

Neto, P. M., 2010. Robot path simulation: a low cost solution based on CAD. pp. 333--338.

Player Project, 2014. *Player Project.* [Online]
Available at: http://playerstage.sourceforge.net/

Quigley, M. C. K. G. B. F. J. F. T. L. J. W. R. a. N. A., 2009. ROS: an open-source Robot Operating System. *International Conference on Robotics and Automation Workshop on Open Source Software.*

ROS, 2014. *ROS.org | Powering the world's robots.* [Online]
Available at: http://www.ros.org/
[Accessed 3 10 2014].

Scacchi, W., 2002. Understanding the Requirements for Developing Open Source Software Systems. *IEE Proceedings,* Febuary.p. 29840.

Singh, A., 2001. *Mac OS X Internals: A Systems Approach.* s.l.:Addison-Wesley Professional.

Smith, R., 2014. *Open Dynamics Engine - home.* [Online]
Available at: http://www.ode.org/
[Accessed 10 2014].

Spitzak, B., 2014. *Fast Light Toolkit - Fast Light Toolkit (FLTK).* [Online]
Available at: http://www.fltk.org/index.php
[Accessed 7 10 2014].

TeamBots, 2014. *TeamBots(tm).* [Online]
Available at: http://www.teambots.org/
[Accessed 3 10 2014].

Webots, 2014. *Webots: robot simulator - Features.* [Online]
Available at: http://www.cyberbotics.com/features
[Accessed 12 9 2014].

Yurii Rashkovskii, D. S. C. V. J. C. S. C., 1998. *GNU Operating System.* [Online]
Available at: http://www.gnu.org/software/commoncpp/
[Accessed 12 05 2014].