THE UNIVERSITY OF NEW SOUTH WALES

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

rUNSWift 2D Simulator; Behavioural Simulation Integrated with the rUNSWift Architecture

Beth Crane

z3299448

bethc@cse.unsw.edu.au

Stephen Sherratt

z3334112

ssteve@cse.unsw.edu.au

COMP3901 (Special Project A)

August 19, 2013

Supervisor: Dr. Bernhard Hengst

Assessor: A. Prof. Maurice Pagnucco

Abstract

Simulation is necessary for the efficient and economical testing of robotic behaviours, and is in wide use throughout the field. There are many competing products and open source systems available for use with RoboCup SPL, each with their own drawbacks. This report approaches the problem of redesigning the existing rUNSWift architecture to allow for cleaner code and abstraction of hardware interaction in the higher-level behavioural code. This abstraction aims to enable easy replication of the actuators and sensors in a simulation environment, moving the focus away from low level integration issues to machine learning intelligent real-time strategic roles and role-switching behaviour. Also explored are the design choices involved in such a process, and a detailed explanation for future use and development of the architecture and simulator is included.

Contents

1. **Introduction**
    1.1. RoboCup
    1.2. RoboCup Soccer Standard Platform League
    1.3. Team rUNSWift
    1.4. Development Environment
    1.5. Document Overview
2. **Simulation Background**
    2.1. Role of Simulation in Development
    2.2. Available RoboCup Soccer Simulators
        2.2.1. Official Simulation League
            2.2.1.1. The RoboCup Soccer Simulator
            2.2.1.2. SimSpark
        2.2.2. SimRobot
        2.2.3. NaoSim
    2.3. Previous rUNSWift Simulation Development
        2.3.1. Simulation and Machine Learning of Physical Movements
        2.3.2. Simulation for Higher Level Behavioural Learning
    2.4. Simulator Aims
3. **Simulation Development**
    3.1. Language
    3.2. Game Library
    3.3. Physics Engine
    3.4. Coordinate System
    3.5. Relationship between Simulator and rUNSWift Codebase
4. **rUNSWift 2D Simulator**
    4.1. Setup
        4.1.1. Requirements
        4.1.2. Usage
    4.2. Maintenance
    4.3. Future Work
        4.3.1. Machine Learning Framework
        4.3.2. Noisy Data
        4.3.3. Physics Complexity
5. **NewSkillz Behavioural Codebase Restructure**
    5.1. rUNSWift Behaviours
    5.2. Motivation
    5.3. Goals
    5.4. In Real Life (IRL) API
    5.5. NewSkillz Delegation Tree
    5.6. Future Work
        5.6.1. Finish NewSkillz Port
        5.6.2. Clean Codebase
        5.6.3. Leaf-Locking
6. **Contribution of Team Members**
    6.1. Simulator

6.2. Codebase Restructure
6.3. Report
7. Conclusion

Chapter 1

Introduction

1.1 RoboCup

RoboCup is an international robotics competition aiming to further robotics and artificial intelligence research through different challenges. Within the RoboCup Soccer challenge there are 5 leagues which each aim to progress the field and achieve the goal stated below;

"By mid-21st century, a team of fully autonomous humanoid robot soccer players shall win the soccer game, comply with the official rule of the FIFA, against the winner of the most recent World Cup."[1]

A number of rule changes are implemented each year in order to stretch the league and continue pushing competitors towards the ultimate goal of the competition. The biggest of these saw the league move from 4 to 2 legged robots in 2008, but more common changes include colour changes, adjustments to lighting conditions and increases to field size.

1.2 RoboCup Soccer Standard Platform League

The Standard Platform League (SPL)[2], within the RoboCup Soccer Challenge, tests purely software advances. The organising committee select a standard model robot for each team to develop on, and hardware modifications are strictly forbidden.

Since 2008 the standard model has been the Nao, flagship product of Aldebaran Robotics[3]. The Nao is a 58cm humanoid robot designed for multiple purposes including: research, education, childhood development and RoboCup. Input is received through its 2 cameras and its sonar, infrared, tactile and pressure sensors. Nao interacts with the world through electric motors and actuators, and has speech capabilities and LED lights for communication with other robots or users.

[1] (2010). Objective « Robocup. Retrieved August 02, 2013, from http://www.robocup.org/about-robocup/objective/.
[2] (2010). Standard Platform League - RoboCup Federation Wiki. Retrieved August 03, 2013, from http://wiki.robocup.org/wiki/Standard_Platform_League.
[3] (2009). Nao - Home - Corporate - Aldebaran Robotics | Accueil. Retrieved August 17, 2013, from http://www.aldebaran-robotics.com/en/.

1.3 Team rUNSWift

UNSW has entered RoboCup SPL each year since 1999. Initially competing as UNSW United before changing to rUNSWift[9], UNSW has competed through a number of rule changes, modifying or restarting the codebase as needed.

The rUNSWift codebase in its 2013 form exists as 100,000 lines of C++ for interfacing directly with the robot's sensors and actuators, combined with a 5,000 line Python base for high-level behavioural skills. The rUNSWift codebase was released to the public in 2010[8], creating a frozen snapshot of well documented, working code. The fundamentals of the architecture established for interfacing with the hardware of the robots is unchanged, and as such the 2010 snapshot serves as a useful reference tool when aiming to understand or modify sections of this code.

1.4 Development Environment

rUNSWift primarily uses the Aldebaran Naos and a to-scale physical field for testing and development, augmented with specialised simulation environments for particular projects.

1.5 Document Overview

This report is divided into sections documenting the development of the rUNSWift 2D Simulator, a section discussing the architectural restructure of the rUNSWift behaviour code and a section explaining the contributions of team members to this project.

Chapter 2

Simulation Background

2.1 Role of Simulation in Development

Ideally as much initial development as possible would be done through simulation in order to reduce damage to the robots and reduce the dependency on an accurate field environment. Replicating the 6m x 9m field is a hard task in and of itself, modifying it each year to follow the rule changes takes time away from development.

Simulation environments that allow for easy adjustment of factors such as field variables, lighting conditions or number of players are highly valuable for testing and development.

The increased number of tests that can be run due to the ability to run games continuously and increase speed faster than real-time allows simulation environments to gather far more data than is available only with physical robots.

This extra data, combined with the ease of resetting the environment and the control over the output and analysis of the data make it a highly attractive tool that is especially useful for machine learning superior strategies. Machine learning strategies via simulation is not an end goal; more work is required in translating the learnt behaviours onto the physical robots, an area that is being increasingly studied[4].

2.2 Available RoboCup Soccer Simulators

The advantages of simulation in robotics are widely accepted[2,6], alongside a healthy awareness of the dangers of discounting the discrepancies with reality[6]. Many teams incorporate simulators into their RoboCup development[7] - both for work on the Simulation League and others. A number of robotic soccer simulators are available for use, whether agent-generic or designed specifically to replicate the Nao hardware.

2.2.1 Official Simulation League[4]

The RoboCup Simulation League features both a 2D and a 3D competition, each with their own official simulator.

---

[4] (2010). Simulation « Robocup. Retrieved August 03, 2013, from http://www.robocup.org/robocup-soccer/simulation/.

2.2.1.1 The RoboCup Soccer Simulator[5]

The 2D simulator plays 2 teams of 11 agents against each other, and runs with very simple graphics. Each robot has its own world view and the corresponding noisy data that comes with that. Client agents connect to the server via UDP/IP - introducing an extra layer of complexity.

2.2.1.2 SimSpark[6]

SimSpark is the official simulator of the 3D simulation league, developed for its inception in 2004. As a *"generic physical multiagent simulator system for agents in three-dimensional environments"*[7], agents must be written specifically for the framework, or ported over from other languages. The resource-intensive 3D graphics processing renders it too slow for our purpose of running fast, successive simulations.

2.2.2 SimRobot[8]

Bremen University develop and maintain their own simulator designed for testing robot behaviours, with a focus on use within RoboCup[7]. Their simulator is feature-rich and available on multiple platforms, however we were unable to properly test or use the simulator due to build issues when compiling.

2.2.3 NaoSim[9]

Aldebaran Robotics, the company that produce the Nao, provide a specifically designed simulator for testing Nao behaviours on. NaoSim uses the exact code running on the robots, a huge advantage, however only simulates 1 robot per environment. The lack of team play rules it out as an option for strategic development.

2.3 Previous rUNSWift Simulation Development

Having noted the inadequacies of the above simulators, previous rUNSWift team members have experimented with developing in-house simulators to better meet the needs of the team.

2.3.1 Simulation and Machine Learning of Physical Movements

Calvin Tam explored using SimSpark to produce realistic agent movements, focussing on the need for accurate movement for the development of higher level strategy.[10]

---

[5] (2002). The RoboCup Soccer Simulator | Free software downloads at ... Retrieved August 03, 2013, from http://sourceforge.net/projects/sserver/.
[6] (2007). Wiki - Simspark. Retrieved August 03, 2013, from http://simspark.sourceforge.net/wiki/.
[7] (2005). SimSpark - SourceForge. Retrieved August 04, 2013, from http://simspark.sourceforge.net/.
[8] (2002). SimRobot - Robotics Simulator. Retrieved August 04, 2013, from http://www.informatik.uni-bremen.de/simrobot/index_e.htm.
[9] (2013). NAOsim — NAO Software 1.12.5 documentation. Retrieved August 03, 2013, from https://community.aldebaran-robotics.com/doc/1-12/software/naosim/index.html.

Bernhard Hengst has done extensive research on using simulation to develop machine-learnt strategies for stability and precision walking[5], and Dan Padilha has explored implementing some of these learned behaviours onto the Naos.[9]

2.3.2 Simulation for Higher Level Behavioural Learning

Yongki Yusmanthia designed a simulator capable of running the 2011 rUNSWift behavioural code, highly integrated with the system design[11]. To account for direct references to rUNSWift modules, such as `blackboard` or `robot`, files with those same names were created and constrained to containing the exact same data.

This simulator is still within the rUNSWift repository, however it is broken with the current architecture.

2.4 Simulator Aims

One of the rule changes for 2013 was to increase the number of robots each team could field from 4 to 5. Our task for rUNSWift was to focus on improving team strategy, requiring repeatedly testing the interactions of multiple robots in numerous situations; ideally something to simulate and apply machine learning rules to.

Finding the previous rUNSWift simulators broken with the current architecture[11], or not suitable for a task focussed purely on high level behaviours, it was determined that building a new simulator would improve the efficiency with which we could improve strategic play.

The specific goal of the simulator was to allow for very quick, very simple data gathering about the relative effectiveness of different behavioural strategies. As the strategies and skills are what is to be learnt, they must translate 1:1 from the robots to the simulator; ideally there will be no modifications between the behaviour code that runs on each system. A common API between the simulator and the robots is necessary for this to be feasible.

Rather than aiming to be generic and flexible, the number one priority is that the behaviours plug and play without needing re-coding. As such, the simulator will be dependent on the rUNSWift architecture not changing dramatically; language changes will render it useless, but changes to the behaviour files will not cause disruptions as long as any changes to the API are reflected in the simulator implementation.

Chapter 3

Simulation Development

3.1 Language

We evaluated Python as continuing to be the best language for the behaviour codebase. The behaviour codebase and this reasoning is further discussed in chapter 5.

The number one design point of the simulator being to smoothly integrate with the skills, the most sensible choice for that was also Python; porting the code is a barrier to ease-of-use, and reduces speed. Python is a well-supported language with extensive libraries and documentation, complemented by a community of packages developed for graphical systems and a number of physics engines to choose from. Keeping the language the same between the skills and the simulator reduces the barrier-to-entry for team members to extend the project; no new languages will be required to understand the workings of the simulator.

3.2 Game Library

There are a number of available game libraries written for Python. From reading through the documentation and tutorials of each, as well as trawling a number of python game development forums and posts[10] [11] [12] [13] [14], a compiled list of advantages and disadvantages for the more established libraries is found in table 3.1.

[10] (2012). Pygame? Pyglet? Something else entirely??? : Python - Reddit. Retrieved August 09, 2013, from http://www.reddit.com/r/Python/comments/15lz1m/pygame_pyglet_something_else_entirely/.
[11] Erik Horton (2013). Python Game Development - Pygame vs Pyglet Pt. 1 | Funinstall. Retrieved August 09, 2013, from http://funinstall.blogspot.com/2013/01/python-game-development-pygame-vs.html.
[12] Leontius Adhika Pradhana (2008). Making Games with Python: Which Library To Use, pygame or pyglet ... Retrieved August 09, 2013, from http://leapon.net/en/making-games-with-python-which-library-to-use-pygame-or-pyglet.
[13] (2008). Differences between Python game libraries Pygame and Pyglet ... Retrieved August 09, 2013, from http://stackoverflow.com/questions/370680/differences-between-python-game-libraries-pygame-and-pyglet.
[14] (2010). PyWeek - A Change of Plans - Panda3D's Performance Just Not ... Retrieved August 09, 2013, from http://www.pyweek.org/d/3430/.

Game Library	*Advantages*	*Disadvantages*
PyGame[15]	• Large support community • Simple and stable system • Tutorials for tying in physics engines • Multi platform installer • Cross platform	• Last stable release 2009
Pyglet[16]	• Benefits from OpenGL advanced capabilities • Hardware accelerated • Lightweight framework • Allows drawing of multiple windows • Cross platform	• Lacking community support • Assumes knowledge of OpenGL
PySoy[17]	• Minimal hardware and software requirements	• Focussed on 3D game texturing and rendering • Does not support Python 2.x
Python-Ogre[18]	• Contains a physics library within its package system	• Focusses on 3D gaming • Quite a heavyweight system to work with
Panda3D[19]	• Fast - processing done in C++ • Written specifically for Python development • Multi-platform installers	• Resource intensive due to fast processing • Long install process
Blender3D[20]	• Advanced rendering capabilities • Combines well with other libraries	• Overly complex for this project
Kivy[21]	• Heavily accelerated • Cross platform	• Lack of documentation • Overly complex for this project

Table 3.1 Advantages and Disadvantages of Python Game Libraries

[15] (2003). News - pygame - python game development. Retrieved August 08, 2013, from http://www.pygame.org/.
[16] (2006). pyglet. Retrieved August 17, 2013, from http://www.pyglet.org/.
[17] (2006). PySoy, a 3D Cloud Game Engine for Python. Retrieved August 08, 2013, from http://www.pysoy.org/.
[18] (2008). Python-Ogre | High performance gaming and graphics library for ... Retrieved August 08, 2013, from http://www.pythonogre.com/.
[19] (2010). Panda3D - Free 3D Game Engine. Retrieved August 08, 2013, from https://www.panda3d.org/.
[20] (2007). Python Scripting - Blender. Retrieved August 08, 2013, from http://www.blender.org/education-help/python/.
[21] (2008). Kivy: Crossplatform Framework for NUI. Retrieved August 08, 2013, from http://kivy.org/.

Major consideration was given to Pyglet for the ability to spawn multiple windows, a potentially useful feature for future development of a machine learning framework, however we selected PyGame for the rUNSWift 2D Simulator due to its stability and extensive documentation and tutorials. The lack of recent and future releases is a downside, but not of critical importance to a project such as this.

3.3 Physics Engine

Pygame is commonly paired with one of three physics engines[22]; Pybox2D, a wrapper for Box2D (C++); Pymunk, a wrapper for Chipmunk (C); or PyODE, a wrapper for Open Dynamics Engine (C/C++). PyODE is employed in some of the aforementioned simulators from section 2.2.

Advantages and disadvantages of each are listed below, however for a project of this size the benefits of one over the other are near negligible.

Physics Engine	*Advantages*	*Disadvantages*
PyBox2D[23]	• Active community support • Constrained to physically accurate behaviour • Greater scope for collision complexity	• Requires SWIG, an extra installation
Pymunk[24]	• Flexible collision callbacks • More 'Pythonic' than PyBox2D	• Fewer features than PyBox2D[25]
PyODE[26]	• Used in other simulators	• Designed for 3D Physics • Lacking active community

Table 3.2 Advantages and Disadvantages of Python Physics Engines

We selected PyBox2D due to its ease of setup and scope for collision handling. It has previously been used by rUNSWift team member Yongki Yusmanthia, in his simulator described in section 2.3.2.

---

[22] (2006). PythonGameLibraries - Python Wiki. Retrieved August 10, 2013, from http://wiki.python.org/moin/PythonGameLibraries.
[23] (2008). pybox2d - 2D Game Physics for Python - Google Project Hosting. Retrieved August 11, 2013, from http://code.google.com/p/pybox2d/.
[24] (2012). pymunk — pymunk 3.1.0 documentation. Retrieved August 10, 2013, from http://www.pymunk.org/.
[25] (2009). Print Page - Chipmunk Physics and Box2D comparison - TIGSource Forums. Retrieved August 11, 2013, from http://forums.tigsource.com/index.php?action=printpage;topic=9318.0.
[26] (2003). PyODE. Retrieved August 10, 2013, from http://pyode.sourceforge.net/.

3.4 Coordinate System

The rUNSWift physical Nao agents vision system describes the world using a (0,0)-centric coordinate system; e.g. the centre of the field is (0,0). Each team views their own goal as `(-Field.length/2,0)` and positive y goes to the left when facing the opposition team's goal. See Figure 3.1 for a visual depiction.

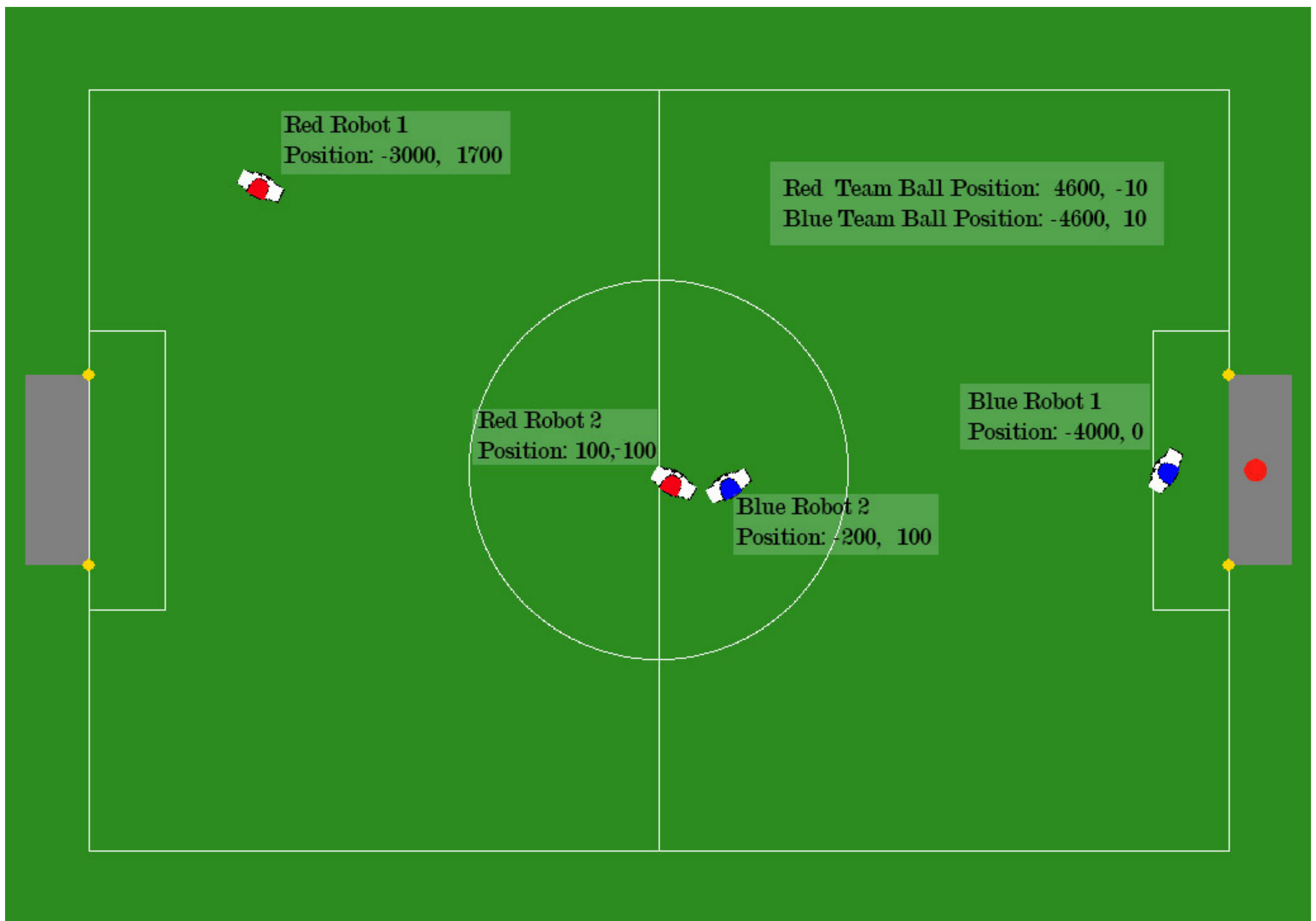

Figure 3.1 Robot viewpoint; team-relative coordinate system

Pygame draws its 'screen' using the top left corner as (0,0). For consistency between the Naos and the simulated agents, coordinates within the simulation software treat the world as (0,0)-centric. To adjust for this, and for each team having an individual interpretation of positions, translation functions had to be implemented for the 5 discrepancies;

1. (x,y) adjustment for drawing images onto the screen
2. Negation of blue team player's x coordinates for drawing/physics
3. Negation of red team player's y coordinates for drawing/physics
4. The blue team player's negation of the ball coordinates
5. Each team's negation of the other team's coordinates

The (x,y) adjustment is abstracted away into the Drawable class in `drawable.py`, with each coordinate passed to the draw() function adjusted to be `(x + Field.length/2 + Field.border, y + Field.width/2 + Field.border)`.

The blue x coordinates and the red y coordinates are negated in the draw() function in `robot.py`, before the robot image is rotated and passed off to be adjusted.

The SimulatorSense class, within `simulator_sense.py`, handles the point-of-view discrepancies, adjusting between coordinate systems depending on the team of the robot. The affected functions are ball_position() and obstacles().

3.5 Relationship between Simulator and rUNSWift Codebase

The design structure chosen aimed to correlate as closely to a real game as possible. Classes were created to represent physical objects, and to reflect the structure of the software running on the robot. Figure 3.2 depicts the simulator design, with reference to physical counterparts.

The fundamental difference between the two is within the IRL API implementation (see section 5.4); the robot IRL classes are simply wrappers for calls to blackboard modules - for instance game controller - or to `robot.py`, the 'Python module' that exists as part of the conversion between the C++ and Python sections of the codebase. The blackboard system is explained in more detail in the team reports, as well as in Yusmanthia's thesis[1, 11].

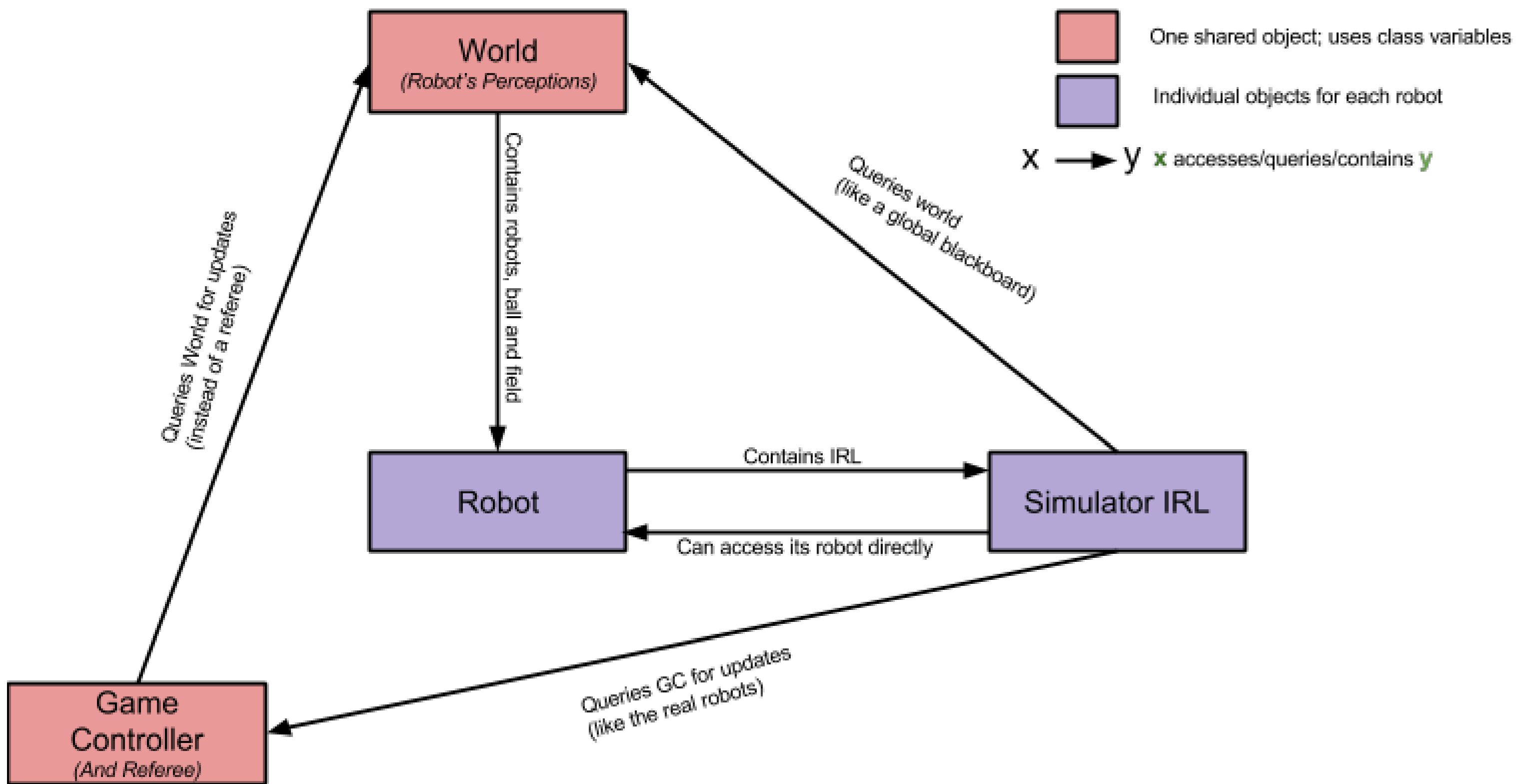

Figure 3.2 Simulation class diagram with references to rUNSWift counterparts

Chapter 4

rUNSWift 2D Simulator

4.1 Setup

4.1.1 Requirements

1. Download and install the latest 2.x.y version of Python[27]
2. Download and install the latest Pygame for your Operating System[28]
3. Download and install Pybox2D[29]
    - For OS X this may require first installing PCRE[30]
4. Clone the rUNSWift repository
    - Switch to branch new_skillz

4.1.2 Usage

Once the requirements are installed and the correct git branch is selected, running the simulator is as simple as navigating to directory `rUNSWift/bsimulator` and running `` `python simulator.py` ``. The skills currently located in the `rUNSWift/bsimulator/skills` directory will automatically be used. This folder is currently a symlink to the `rUNSWift/image/home/nao/behaviours/skills` directory.

[27] (2011). Download Python. Retrieved August 06, 2013, from http://www.python.org/getit/.
[28] (2002). Downloads - pygame - python game development. Retrieved August 06, 2013, from http://www.pygame.org/download.shtml.
[29] (2010). install - pybox2d. Retrieved August 06, 2013, from http://pybox2d.googlecode.com/svn/trunk/INSTALL.
[30] PCRE - Perl Compatible Regular Expressions. Retrieved August 06, 2013, from http://www.pcre.org/.

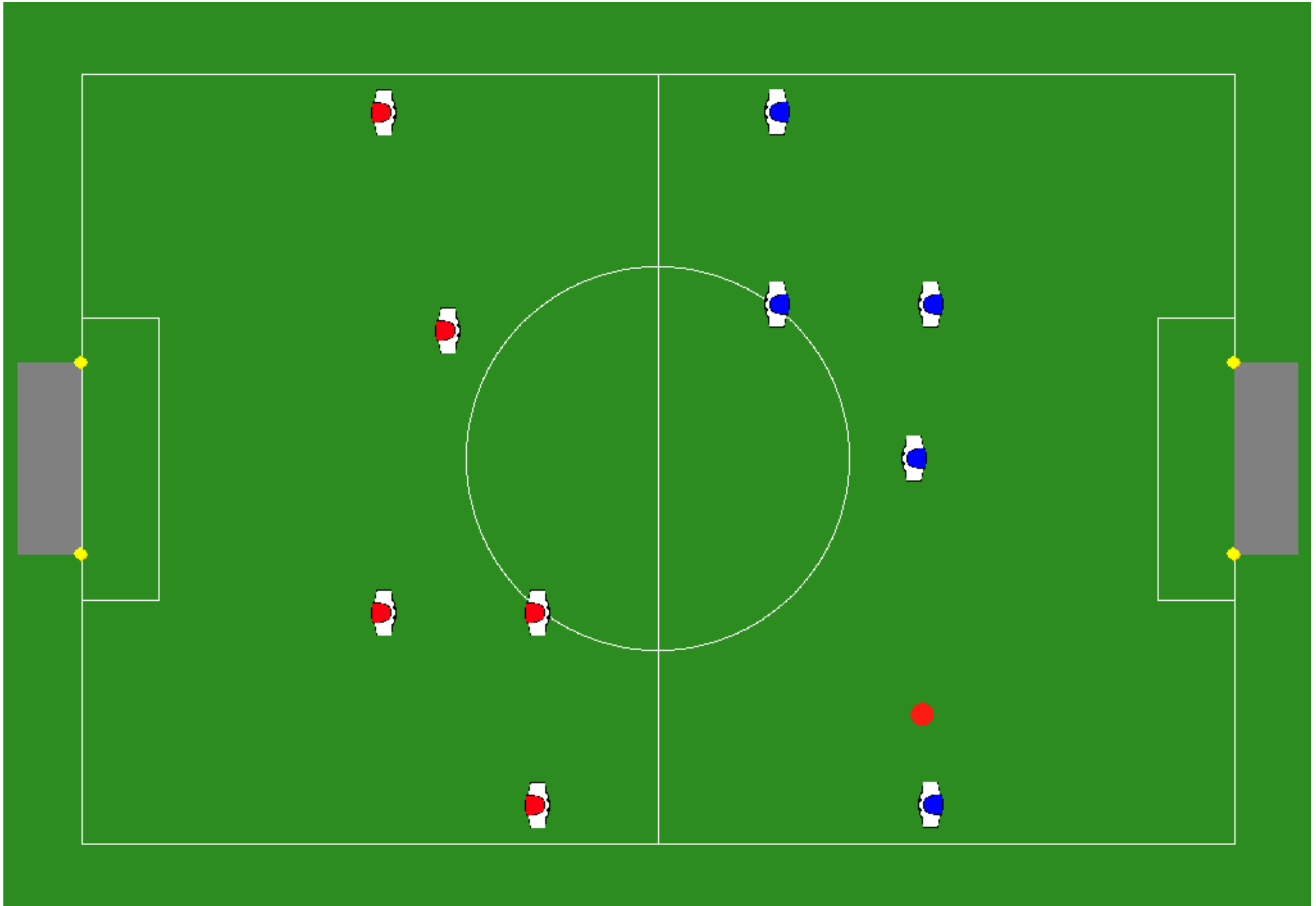

Figure 4.1 Image of simulator just after first ball kick

4.2 Maintenance

Major architectural changes to the rUNSWift codebase, e.g. moving away from Python, will render the simulator useless. For use with the current architecture, changes made to `rUNSWift/image/home/nao/behaviours/skills/(robot_sense.py|robot_action.py)` that affect functionality or add new functions must be mirrored in `rUNSWift/bsimulator/(simulator_sense.py|simulator_action.py)`.

4.3 Future Work

Machine Learning Framework

Having created a simulator integrated within an understandable code base, the next step would be to increase the sophistication of the system. Our recommendation would be to focus on a wrapper framework to enable fast customisation of the simulation settings and data output, later tied in with a built-in reinforcement learning system to be toggled on or off by the user.

Potential settings for this framework would be:

- Data output (to file) of a number of data points - e.g. penalties, points scored for/against, etc
- Faster pure data output *versus* watching the games on the graphical interface
- Selecting where the source skills files are located
- Selecting the number of simulations to run
- Toggling learning on and off

Noisy Data

Modelling the probabilistic nature of knowledge known to the Naos and implementing a similar system for unknown and approximate data on the simulator would, by decreasing data accuracy, hugely increase its accuracy in reflecting real games.

Physics Complexity

Another area for future interest could be improving the complexity of the physics; with improved collision detection and better accuracy in object movements lower level skills, such as a more efficient walk-to-point, could be trained as well as the intended role-switching and positioning skills.

Chapter 5

NewSkillz Behavioural Codebase Restructure

5.1 rUNSWift Behaviours

The Python behaviours codebase, introduced in section 1.3, controls the high-level decisions of the Nao on the field. The behaviours defined in this section control everything from how the robot lines up for the ball to whether it acts as a Forward or a Defender and if it should move out of the way of a teammate.

We consider Python to be the most suitable language for this codebase, even aside from the advantage of staying with the previously-established choice, for the following reasons:

- The ability to update code whilst it is running on the robots is a huge advantage for testing and at competition
- A lot of previous rUNSWift work, notably by David Claridge[3], has been devoted to ensuring Python works well with the C++ low-level code
- High level behavioural code merits a high level language for ease of development

5.2 Motivation

From the 2012 rUNSWift Codebase[1] the 2013 team inherited a wealth of strategic play and skills, allowing us to establish an idea of the essential components for a successful behaviour strategy; a mix of behaviours from direct, basic skills like 'Walk to Point' and 'Kick Ball' to complicated player roles such as 'Goalie' and 'Striker'.

Having gained this knowledge, we determined that a codebase restructure would be more economical than trying to make adjustments to the current system; the existing codebase was plagued with magic numbers, rampant redundancy, idiosyncratic hard-coded edge cases and code segments leftover from previously discarded strategies, as well as lacking comments and documentation.

```
380        if ballx < 147 and abs(self.angleDir - targetDir) < radians(9):
381          if self.useLeft and bally > -110 and bally < -75:
382            self.arrived = True
383          if not self.useLeft and bally < 110 and bally > 75:
384            self.arrived = True
385        if ballx > 168 or abs(bally - targety) > 45 or abs(self.angleDir - targetDir) > radians(12):
386          self.arrived = False
387
388        # Walk back a bit first if it's next to us but not forward enough
389        if ballx < 125 and abs(bally - targety) > 42:
390          targetx = targetx + 25
```

Code 5.1 Code segment from `Master/Shoot.py` riddled with magic numbers

```
242      nearestDist = 9999999
243      striker = None
244      for i in range(0, len(blackboard.receiver.data)) :
245         buddy = blackboard.receiver.data[i]
246         if (buddy.playerNum != blackboard.gameController.player_number and
247             buddy.uptime > 0):
248            dist = buddy.ballPosRR.distance
249            if not math.isnan(dist) and dist < nearestDist:
250               (nearestDist, striker) = (dist, buddy)
251      if nearestDist - myDist > -500 - buff and nearestDist > 300:   #clear if striker is further than us
252         return True
```

Code 5.2 Code segment from `Master/Goalie.py` with messy blackboard calls

5.3 Goals

Through restructuring the aim was to create a clean and well-documented codebase that would reduce confusion and learning-curve time for future rUNSWift teams, and enable faster debugging at competition.

The NewSkillz restructure serves the dual purpose of introducing the API structure necessary for the simulator and reshaping the design from a non-hierarchical state machine with no knowledge of its predecessor state to a more intuitive and documentable tree-like structure.

5.4 In Real Life (IRL) API

The API enables easy re-implementation of functions that interact with the hardware through the blackboard; a requirement for implementing the simulator's 'hardware', and a simple way to reduce duplicated functions and abstract confusing naming conventions (e.g. `blackboard.localisation.robotObstacles.rr`) out of the skills code.

Figure 5.1 depicts the different branches the IRL path takes for two separate implementations, each running the same skill.

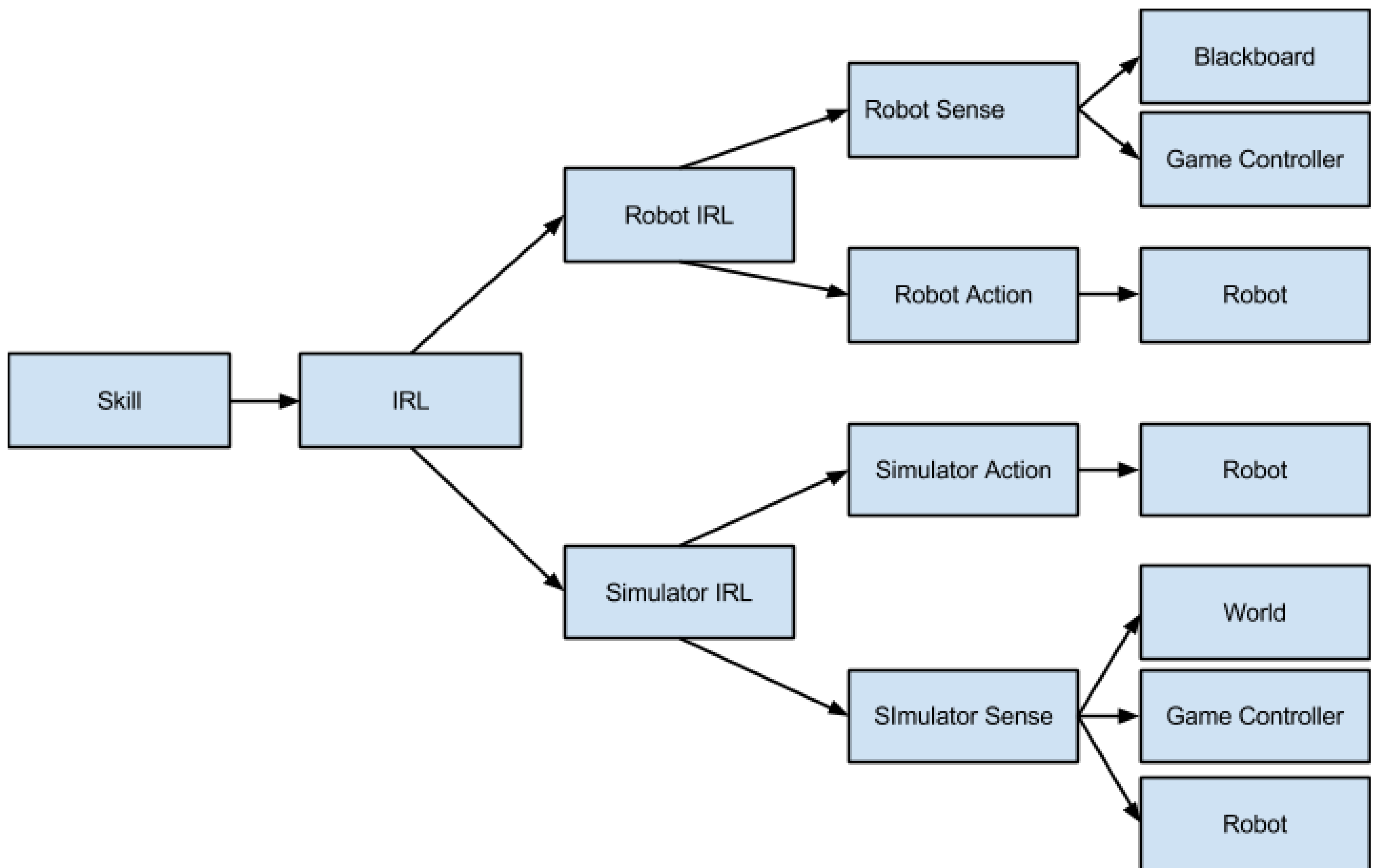

Figure 5.1 The separate streams of the IRL API

5.5 NewSkillz Delegation Tree

Within the NewSkillz hierarchy there are two types of divisions;

1. 'Public' vs 'Private
    a. 'Public' classes are referenced both within the file and outside, these are indicated with a name corresponding to the filename
    b. 'Private' classes are referenced only by other classes within its file
2. Internal vs Leaf nodes
    a. Internal nodes are indicated with a 'delegate' function, called each clock tick. These are calculating functions that do not affect the actuators.
    b. Leaf nodes are indicated with a 'tick' function, called each clock tick. These skills typically involve an 'action command' that affects the robot's actuators (`e.g. self.irl.action.walk()`).

Every skill in the path from root node to current-state node is accessed each clock tick, starting from the root node down. The 'delegate' functions are called on each node in the path, allowing the path to shift at any level. The bottom node in the path will eventually be a leaf node, and its 'tick' function will be accessed rather than its (non-existent) 'delegate'.

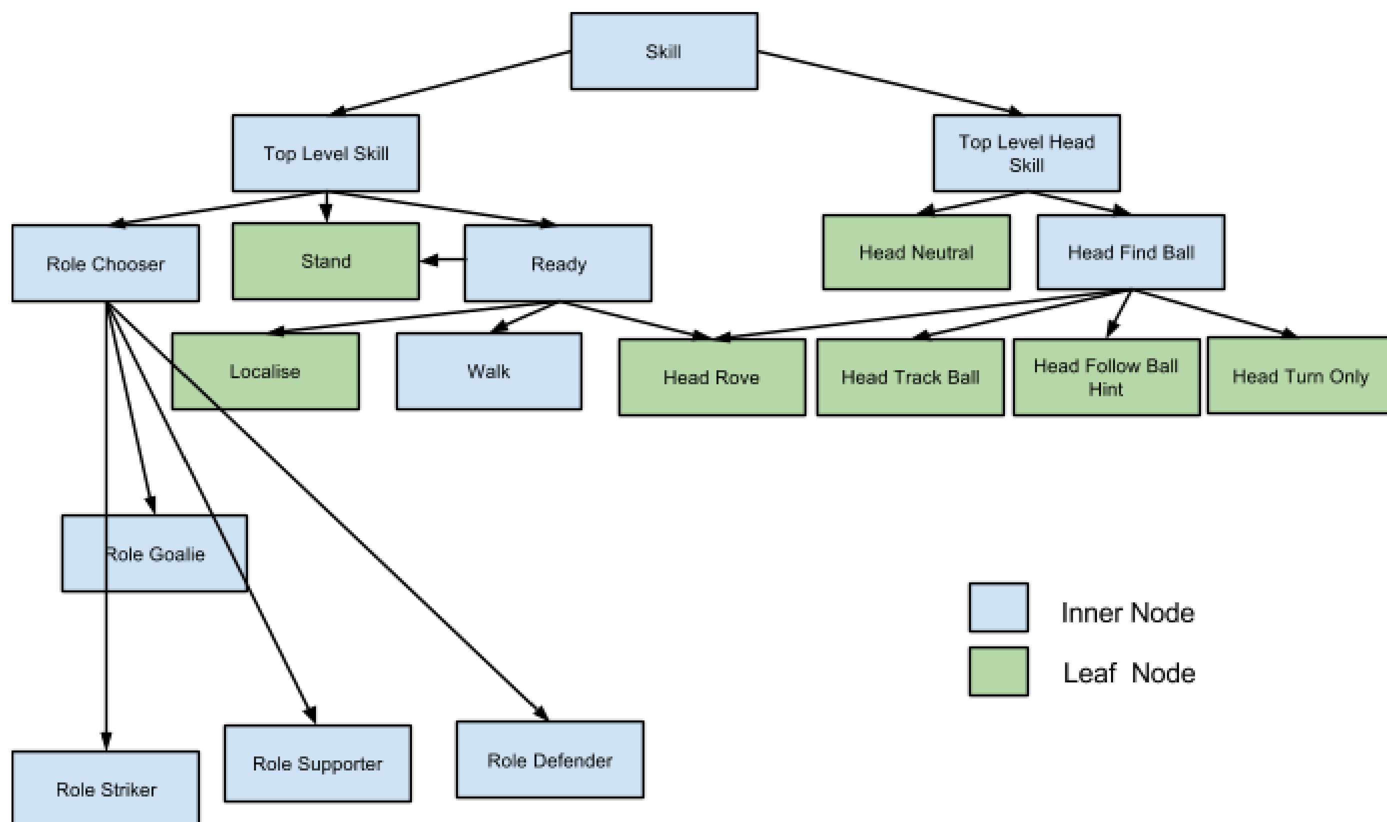

Figure 5.2 An overview of the Skills Delegation Tree

The differences between the 2012 framework and the 2013 framework have been nicely visualised by Dan Padilha[9].

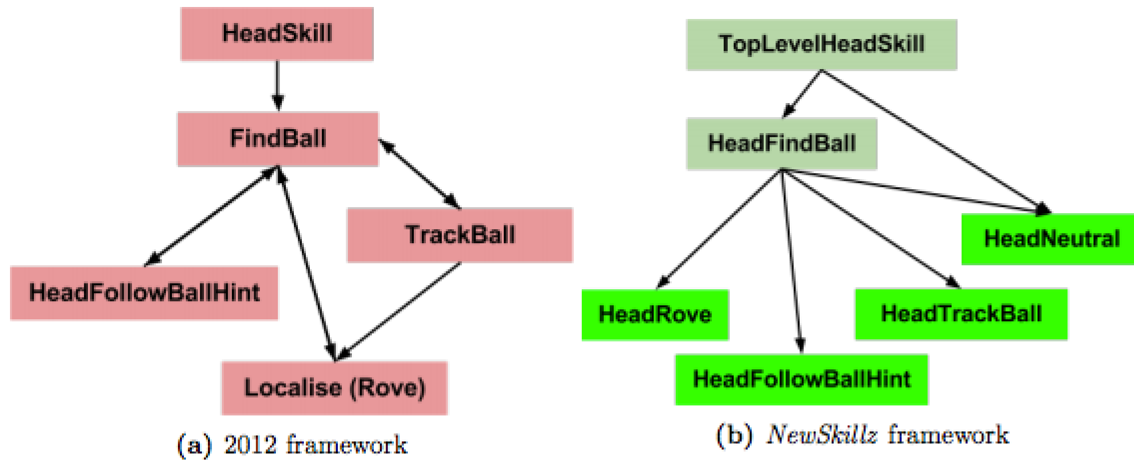

Figure 5.3. Pictorial representation of the contrast between 2012 and 2013 behaviour frameworks

5.6 Future Work

Finish NewSkillz Port

The 2012 behaviour code[1] was ported to the NewSkillz framework in a simplified Proof-of-Concept. The founding structure for the new architecture is in place, with these simple and clean skills. More testing is necessary to further build up the complexity of the skills and improve the level of play - an ideal project would be devising a roles template for different actions to be learnt with the simulator.

Clean Codebase

Another direction for further work in this area is continuing to clean up the codebase; devising elegant ways to reduce redundancy and increase how intuitively it is received and understood when new team members first approach it.

Leaf-Locking

Leaf nodes, described in section 5.5, implement actions. Currently there is no system in place to prevent a higher up node switching the node path whilst the robot is in the middle of implementing a leaf node action.

Chapter 6

Contribution of Team Members

6.1 Simulator

The simulator was worked on by Alexander Whillas, Stephen Sherratt and Beth Crane. The classes necessary to display the field were created by Alexander, with the remaining work split equally between Stephen and Beth. Stephen concentrated on the physics and computational geometry of the simulator, whilst Beth designed and ported the API, as well as establishing the overall simulator design and implementing the game controller.

6.2 Codebase Restructure

Developing the design of the framework was largely done by Jack Murray, in collaboration with Stephen Sherratt and Beth Crane.

The implementation of the framework, porting old skills to new skills, was split between Jack Murray, Dan Padilha, Stephen Sherratt and Beth Crane. Work was not divided cleanly; the job was a fast-paced team effort with frequent pair programming and tasks being allocated whenever a previous one was finished.

6.3 Report

This report was written and compiled by Beth Crane.

Chapter 7

Conclusion

The aim of this project was to develop a system for improving high level behaviours for the RoboCup SPL, specifically real-time strategic role-switching. The rUNSWift 2D Simulator forms the foundation of this system, and this report shows where future work can be applied in order to fully benefit from the integrated design. Usage of the new architecture and simulator will assist future rUNSWift teams in gaining an understanding of the system and developing sophisticated, machine-learnt strategies to continue progressing the field of robotics at the RoboCup SPL.

References

1. Anderson, P., Chatfield, C., Harris, S., Hua, R., Hunter, Y., Li, S., Liu, R., Roy, R., Teh, B., Hall, B., Hengst, B., Pagnucco, M., Sammut, C. (2012). *RoboCup 2012 Standard Platform League*. Available online: http://cgi.cse.unsw.edu.au/~robocup/2012site/reports/Robocup2012rUNSWiftTeamDescription.pdf
2. Carpin, S., Lewis, M., Wang, J., Balakirsky, S., & Scrapper, C. (2007, April). *USARSim: a robot simulator for research and education*. In Robotics and Automation, 2007 IEEE International Conference on (pp. 1400-1405). IEEE.
3. Claridge, D. (2011). *Generation of Python Interfaces for RoboCup SPL Robots*. Taste of research report, The University of New South Wales. Available online: http://cgi.cse.unsw.edu.au/~robocup/2011site/reports/Claridge-Python.pdf
4. Domingues, E., Lau, N., Pimentel, B., Shafii, N., Reis, L. P., & Neves, A. J. (2011). *Humanoid behaviors: from simulation to a real robot*. In Progress in Artificial Intelligence (pp. 352-364). Springer Berlin Heidelberg.
5. Hengst, B. (2012). *Reinforcement Learning of Bipedal Lateral Behaviour and Stability Control with Ankle-Roll Activation*. In WSPC Proceedings, 2013.
6. Jakobi, N., Husbands, P., & Harvey, I. (1995). *Noise and the reality gap: The use of simulation in evolutionary robotics*. In Advances in artificial life (pp. 704-720). Springer Berlin Heidelberg.
7. Laue, T., Spiess, K., & Röfer, T. (2006). *SimRobot–a general physical robot simulator and its application in robocup*. In RoboCup 2005: Robot Soccer World Cup IX (pp. 173-183). Springer Berlin Heidelberg.
8. Lewis, M., Wang, J., & Hughes, S. (2007). *USARSim: Simulation for the study of human-robot interaction*. Journal of Cognitive Engineering and Decision Making.
9. Padilha, D. (2013). *rUNSWift Robocup SPL 2013 Special Project A Report*. The University of New South Wales.
10. Tam, C. (2012). *Developing Agent Behaviour in the SimSpark Simulator.* Special Project A, The University of New South Wales.
11. Yusmanthia, Y. (2011). *Simulation and Behaviour Learning for RoboCup SPL.* Thesis, The University of New South Wales. Available online: http://cgi.cse.unsw.edu.au/~robocup/2011site/reports/Yusmanthia-Simulation.pdf
12. *rUNSWift code release, team report, apt repository and 64-bit CTC*. (2010). Retrieved August 10, 2013, from https://mailman.cc.gatech.edu/pipermail/robocup-nao/2010/000263.html
13. *UNSW RoboCup Team*. (2013). Retrieved August 2, 2013, from http://www.cse.unsw.edu.au/help/students/Student-Projects/RoboCup/