UNIVERSITY OF NEW SOUTH WALES

UNSW Undergraduate Research Student

Industrial Training Report

Youssef Hunter (z3242198)

2/16/2011

A brief outline of experimental robotics work undertaken over three months for the RoboCup Standard Platform League team rUNSWift, as part of the UNSW “Taste of Research” program.

Table of Contents

1 Introduction

1.1 RoboCup

RoboCup is an international research and education initiative that was established in 1997 to promote development and encourage research into the fields of experimental robotics and artificial intelligence. Competitive games of soccer are used to provide a standard problem for which a wide range of technologies can be integrated and examined. The game of soccer also provides a publicly appealing and formidable challenge with a significant long term goal:

> *"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."* – The Robocup Federation

1.2 Standard Platform League

RoboCup offers a number of various "leagues" in which teams can compete. In the Soccer Standard Platform League, all the teams use identical robots: the Aldebaran Nao humanoids (see Section 2.4.1, page 12). By doing so the teams can use state-of-the-art robots while focusing on the software development aspects of the competition, including: autonomous multi-agent collaboration, strategy, real-time reasoning, as well as sensor and vision systems.

1.3 rUNSWift

The University of New South Wales (UNSW) has fielded a team now known as ***rUNSWift*** at RoboCup since 1999, with an impressive record of success. The team has grown in size substantially over the years, and now consists of a diverse group of undergraduate and postgraduate research students, as well as research associates, lecturers, and the Head of the School of Computer Science & Engineering himself. I undertook a "Taste of Research" over the summer working as a member of the team from 15th November 2010 to 18th February 2011.

2 Projects

2.1 Foot Detection & Ball Stealing

2.1.1 Aims

My most significant project throughout the summer was my research project, and was the responsibility that dominated the vast majority of my time here. In a meeting with my research supervisor before the commencement of my term, we established that what the rUNSWift team lacked was an effective ball control strategy for close-encounters with other robots. This would give the team a great competitive advantage over other teams, so it became my research project. I then constructed a set of aims for my project, which were as follows:

- To design a method for detecting and identifying opponent robots' feet and their locations.
- To develop a more productive strategy of robot movement and kicks that uses this information to control the ball in close-proximity situations.

2.1.2 Induction

The first two weeks of my placement were an induction period of sorts, spent setting up and familiarizing myself with the robots and development environment. I was also reading and understanding team reports from previous years, as well as understanding the complex robot architecture and software design of the existing team code before embarking on my own project. This period was invaluable in helping me establish where and how I would eventually write my own project's code, and how it would fit within the complete program.

2.1.3 Edge Detection

The first step in my ball stealing project was to identify robot feet in the images the robot receives from its two onboard cameras. After experimenting with the robots on the field and examining their output, I established that the simplest method for achieving this would be to detect large white circles in the images of a certain radius. Initially my attempts focused around using or extending the pre-existing "region" code within the vision framework, which categorized images into various regions and provided the basis for many of the other vision modules[1]. After roughly a week of exploring this option however, I opted for a slightly different and simpler approach, with the primary difference being that I would use edge detection, and hence not have to rely on an often inaccurate colour classification.

Using the existing "Ball Detection" code as a basis, I set to work programming the edge detection that would eventually form the foundation of my foot detection. To remain efficient (to ensure that the vision module of the program could keep up with the 30 frames/second influx from the camera), I designed my edge detection to be as simple as possible. Pixels were compared with horizontally and vertically adjacent pixels and those that differed over a certain threshold were classified as edges.

The Ball Detection code that I had based my edge detection code on was designed for detecting the edges between the purple ball and green grass, and as such was not as useful for detecting the edges between the white feet and the green grass. I experimented by testing various other thresholds and methods of distinguishing between pixels, as well as examining the colour-plots[2] of some sample images. In doing so I eventually found a much more productive edge detection method (see Figure 1, page 6).

[1] Other vision modules include ball, robot and goal detection.
[2] A 3D plot within the YUV colour space, of all the pixels in an image.

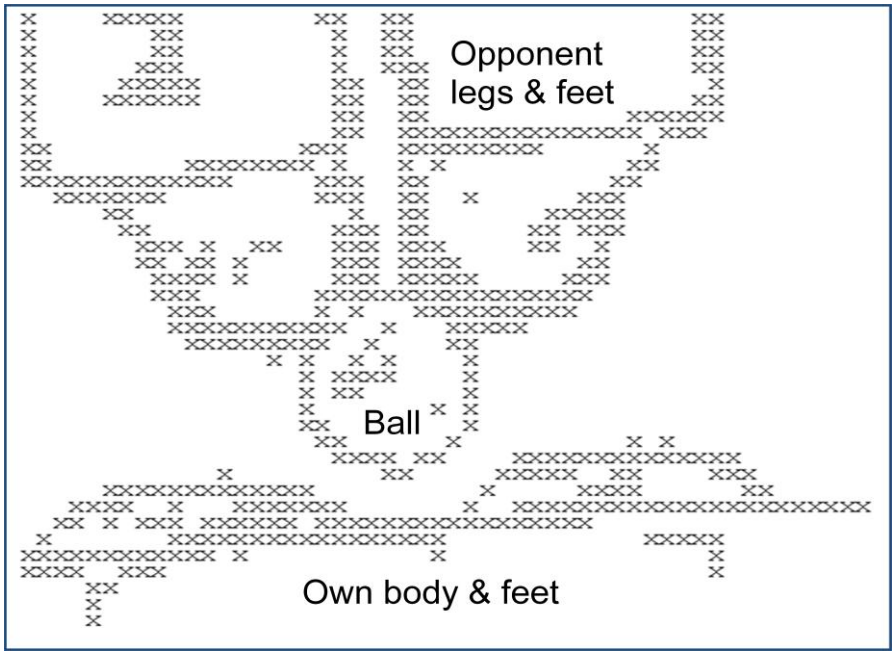

Figure 1: Edges detected within an image – looking down at opponent's feet and ball.

The low resolution of the above image is another result of the need for efficiency; a variety of resolutions were tested to find the optimum balance of speed and accuracy. Despite the raw images from the camera being 640x320 resolution, the opponent's feet and ball were clearly visible within edge detection at resolutions as low as 40x30 – and performing the edge detection calculations on every 16th pixel caused a significant reduction in processing time.

I also developed a variety of other methods to improve the efficiency of the edge/foot detection code. These included a number of sanity checks that prevented the foot detection code from running at all unless certain criteria were met, such as:

- If a ball could be seen in the image – there is little need for a ball stealing strategy if a ball is not visible.
- If the head is angled sufficiently downwards, and the "bottom camera" is in use – otherwise the chances of seeing relevant feet in the image are slim.
- If the robot's sonar sensors indicated the presence of any masses in front of the robot.

Another significant efficiency-concerned improvement I implemented was to ignore certain parts of the image (see Figure 2, page 8). This is achieved in two ways, by ignoring parts of the image that will likely contain glimpses of the robot's own body[3], as well as ignoring the region of the ball in the image[4]. Processing time is saved from not comparing these pixels in edge detection, as well as reducing the total amount of edges in images, which greatly reduces the amount of computation required in circle detection.

Testing and experimenting with the robots, combined with bug-fixing, optimizing, writing and refactoring the code meant that this was a fairly long process, and amounted to a roughly two week period in which the edge detection came together (three weeks when combined with the first week of exploring the region-based method).

2.1.4 Circle Detection

Once I had developed an efficient yet accurate method of detecting the edges within an image, the next week of development was dedicated to identifying circles within the edge-images I had created. My supervisor informed me of some relevant work a colleague had done in the previous year[5], and so I began using the same approach: using a simplified and targeted version of the Hough Circle Detection algorithm. The algorithm is targeted in the sense that it only looks for circles of a particular radius – the radius of the robot's foot, measured in pixels and obtained via experimentation. By examining each edge in the image, and assuming that they are part of a circle (with the pixels surrounding it being possible circle centres), the algorithm accumulates points for each pixel in the image as possible circle centres – with the pixels with the highest values being the most likely candidates for circles and thus foot centres.

[3] Initially I used the robot's kinematic chain and the pre-existing "exclusion zone"; however this was often inaccurate and simplified to a lower-bound that changed with the downward angle of the head.
[4] I ensured that the ball detection module is run before foot detection, so the ball location within the relevant frame is always known.
[5] One of the rUNSWift submissions for the 2010 "Open Challenge" of the Robocup was a black-and-white ball detection routine, which used the relevant circle detection method.

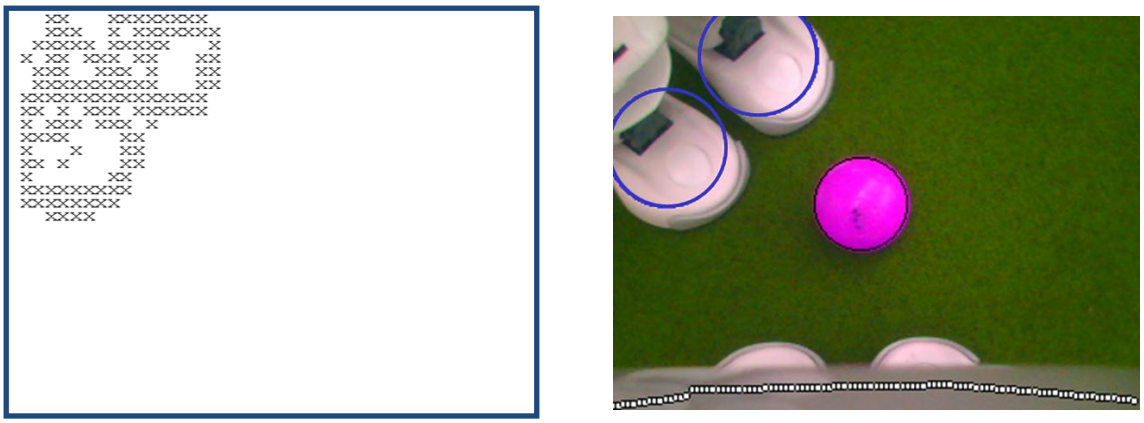

Figure 2: Edges and feet (blue circles) detected, with ball and own body ignored.

This method of circle detection worked well, however despite the targeted algorithm only examining circles of the desired radius the program was not efficient enough to run at the desired speeds. To greatly reduce computation, I used pre-calculated look-up tables to replace the costly Sin and Cos operations that were necessary. The low resolution of the images and size of the circles within them also meant that I could reconfigure the algorithm to only look at a mere fraction ($1/10^{th}$) of the possible candidate circle centres around any edge. The result was an efficient yet accurate enough method of detecting circles within images (see Figure 2).

2.1.5 Ball Stealing

Once I had successfully implemented the circle/foot detection, it was time to start using this new information in a productive way: the goal being to develop a behaviour that would have the robot detecting feet, identifying their position relative to the robot, and then kicking the ball in an appropriate direction. I first developed a filter for the foot information received from the vision module. Rather than looking at the information from each image individually, the filter allowed for the robot to average the readings over time. This gave the robot more consistent information on the foot locations, and was incredibly helpful in negating any false positives that occasionally occurred with distorted images.

The behavioural code I developed also had to categorize the location of the feet it received from the filter into various regions I defined[6], based on the heading of the observations – which are calculated relative to the robot from the position in the image and knowledge of the robot's current kinematic position. Once the foot location has been categorized, the behaviour I wrote will use this information to make a fairly simple decision: calling an appropriate motion depending on the category of the detected foot.

The behaviour first locates a ball via scanning with its head, approaches it, and then kicks the ball to the left if it senses the robot is more towards the right (see Figure 3, page 10), and vice versa for robots detected on the left. The behaviour can also dribble the ball forward if it does not detect feet or if it has detected them and categorized them as being far enough away to not present an obstacle.

In addition to writing the behavioural code to demonstrate the ball stealing mechanism, I also worked with a colleague[7] on preparing the relevant sidekicks that would be used by the behaviour to manoeuvre the ball. These robot motions required a great deal of testing and fine-tuning, both in the behaviour and the motion generator. Testing the foot detection and ball stealing behaviour so thoroughly, with the robot in motion and in a variety of scenarios was also useful in highlighting numerous issues, which were then addressed. The development and testing of the behaviour and motion required a further two weeks, totalling two months so far and marking the completion of the preliminary foot detection and ball stealing work.

[6] Far Left, Left, Centre Left, Centre Right, Right and Far Right were the regions used.
[7] A fellow Taste of Research student whose focus was the robot's walk engine, motion, and kinematics.

Figure 3: Aldebaran Nao performing a ball stealing manoeuvre – a left side kick.

2.2 Behaviours

The ball stealing behaviour code I wrote is only one of the many behaviour classes that constitute the larger collective behaviour module: the robot uses and transitions between all of these behaviours, known as "skills", to play the game of soccer. However during the course of my research term it was decided that the framework for the robot's behaviours was due for an overhaul (see Section 2.4.2, page 13). The new behaviour system allowed for brand new behaviours to be written more easily than previously, yet unfortunately the architecture was not compatible with the entirety of the vast number of legacy behaviours written for last year. For the next two weeks my primary responsibility was to rewrite/modify/update these behaviours to work with the new behaviour system.

I began rewriting and testing the new behaviours from the most logical starting point, working my way up through the simplest and most important skills. At first merely instructing the robot to stand up and indicate whether a ball was detected, then progressing onto scanning for the ball by moving the head through waypoints, followed by tracking the ball as it moves/rolls around, and walking towards the ball it has detected. I also rewrote my ball stealing behaviour for this new architecture.

Having me rewrite and experiment with the new behaviours almost from scratch also provided the team with a valuable fresh insight and new perspective on the behaviours – using the old behaviours as a basis for the new improved, modified and re-factored behaviours. As I was the first and only one at the time actively writing behaviours with the new behaviour system, I also contributed a lot to configuring and testing the recently updated behaviour architecture.

2.3 Poster Presentation and Wrap Up

The last two weeks of my term were spent bringing my research to a close. As a requirement for my Taste of Research Scholarship I had to participate in a Poster Presentation Session, which included attending two workshops and preparing an A2 poster that summarised and portrayed the research I had done over the summer, as well as preparing a 2 minute speech to accompany it. Attending the Poster Presentation Session gave me the chance to present my work to various Faculty of Engineering staff, as well as to other Taste of Research students.

As my term ended the team made another change to the robot's architecture, this time affecting the vision module. This meant that before I left the team I had to update the foot detection code I had written earlier, and ensure that it was compatible with this new system. This change also coincided with an alteration to the output of the debugging program Offnao (see Section 2.4.2, page 13), and also required me to rewrite the interaction between it and my foot detection module.

With my foot detection and behaviours updated for the new systems, and Offnao providing debugging output, my work on Ball Stealing was complete. In my final week I provided a final demonstration to the rest of the team, and instructed one of the members on its workings for future reference. My research provided an excellent proof of concept and basis for the team to incorporate the strategy into future play, and I eagerly await news of its performance in the upcoming 2011 RoboCup World Cup, to be held in Istanbul, Turkey this year.

2.4 Development Environments

The entirety of my work over the summer was focused on projects involving very similar development and testing environments. As such this section provides environment details referring to all the projects, outlined in sections 2.1, 2.2 and 2.3.

2.4.1 Hardware

The Aldebaran Nao humanoid robot (see Figure 4) is the "standard platform" of the RoboCup Standard Platform League, and as such is the main focus of the rUNSWift team. The team usually operates and tests development work by writing and compiling the "runswift" executable on personal machines and transferring it over a wireless or LAN network to a robot to run. The team also uses their personal machines for debugging: the robots maintain a link with the debugging program, sending data about the current system state.

The main features of the robot's hardware include:

- Geode LX800 processor
- 500MHz
- 512MB RAM
- 32MB Memory Stick
- 300,000-pixel digital cameras (640 x 480 resolution, 30fps)
- Wireless LAN (standard)
- Linux

Figure 4: Aldebaran Nao humanoid

2.4.2 Software

The majority of the rUNSWift system is written in C++, with the exception being the behavioural module which is written in Python. The system has recently been modified to incorporate SWIG (Simplified Wrapper and Interface Generator) to help facilitate the transition of information between the two languages, allowing for updates and changes to the structure of the other modules (including vision and motion) to be much more efficiently accessed by the Python behaviour module.

The team cooperate using a centralised and well organised repository, taking full advantage of the revision control system Git. This is especially relevant and useful as at any one time a number of students would be working on separate modules of the system. A “master” copy of the system is kept ready at all times, always prepared to be transferred to the robots and play a full game of soccer. Only work that is thoroughly tested and ready is pushed to the master branch, which is pulled by others on a regular basis to ensure compatibility issues are kept to an easy-to-manage minimum.

The team’s own debugging utility “offnao” is also written in C++. When connected to the robot it receives a steady stream of information concerning the state of the robot, including sensor values, a map of the field with identified objects, vision images and their processed perceived contents. As functionality is added to the system, the debugger can also be updated to display the data it receives in the desired format. Figure 2 (see page 8) demonstrates offnao superimposing blue circles over the raw image, in the location my foot detection method has determined there to be apparent feet.

3 Relating to Studies

3.1 Software Engineering

This experience has been an invaluable example of a Software Engineering project that I have learnt much from. rUNSWift is a diverse team of collaborating individuals who meet at least once a week to discuss progress and strategy; it exposed me to a larger team based project than I had experienced before, as well as introducing me to various aspects of undertaking research.

Despite providing me with very little “formal training”, my term did provide an opportunity to expand my knowledge of the programming languages C++ and Python through practice and experimentation, as well as the project development and testing process.

3.2 Artificial Intelligence

Quite obviously the RoboCup competition and rUNSWift project are closely linked to the field of Artificial Intelligence, and I was able to build on the knowledge I gained in my elective COMP3411 Artificial Intelligence course.

3.3 Computer Vision

The Computer Vision principles and methods I used to aid in my development of the foot detection system will be relevant to the Computer Vision course I intend to undertake soon.

4 Conclusion

In conclusion, my term with the rUNSWift RoboCup team has been an interesting and rewarding experience. My work on developing a foot detection method and ball stealing manoeuvre has helped me build on prior knowledge, as well as introduced me to new fields of computing – whilst also providing an invaluable *taste* of research.