UNIVERSITY OF TARTU

FACULTY OF SCIENCE AND TECHNOLOGY

Institute of Technology

Rainer Keerdo

Developing the behaviours for use in RoboCup 2015 based on rUNSWift architecture.

Bachelor's thesis (12 ECS)

Supervisor:

Assoc. Prof. Gholamreza Anbarjafari

Tartu 2015

Abstract

The aim of this thesis is to evaluate the use of rUNSWift 2014 Codebase as a potential method for improving the general gameplay performance in RoboCup 2015.

At first, a review was done on the rules of the RoboCup Standard Platform League. The actual capabilities of the Aldebaran Nao platform were also looked at to better understand the system being used and how it ties together with the control software. The set-up time on a clean installation was also evaluated.

The work performed consists of evaluating the rules, the platform, the tools and then using them to develop the behavioural module for use in RoboCup 2015. This was done mostly in lab conditions on a scaled RoboCup field using real Nao robots to develop the system and test their compatibility with RoboCup control system.

The resulting behaviour code allows for individual behaviour with improved walking and kick stability and provides a platform for future robotics students and researchers wishing to participate in RoboCup 2014 as a part of team Philosopher.

Contents

Chapter 1

Introduction

For the past seven years Aldebaran Nao platform has been used as the robot for use in RoboCup Standard platform league. During this time, there have been several revisions to both the hardware and software used - there have been upgrades to the computational power available, the cameras, sensors and the operating system itself.[19] Although progress has been made, the current level is still too far from the final goal of the RoboCup competition - playing against the best human football team and winning. This idea started the initial humanoid robot league, which made it's first appearance in the 2002 RoboCup held in Fukuoka, Japan . In 2008 the Nao robot was chosen as the new system for use in RoboCup Standard Platform League (SPL), replacing the older Sony four-legged robot dog Aibo.[9] [25]

The robots playing football in SPL (and other leagues as well) must be fully autonomous - meaning they cannot accept any external commands other than the ones allowed by rules - game control system signals and messages from the coaching robot in the case of SPL. This creates a need for better multi-modal systems, wireless communication protocols and localization systems, which is a good model for conducting research - the teams who are all trying to find better solutions to current problems in robotics get to compete and in the end the best methods can be chosen based on the results of actual field testing.

University of Tartu first competed in RoboCup SPL in 2014 in Sao Paulo, Brazil, where they managed to score a goal in a real match. As the amount of manpower available for development is significantly lower than top competitors and we lack the experience. For comparison, the 2014 champion rUNSWift has been accumulating experience by competing in RoboCup since 1999.[3]

1.1 Problem description

The main problem stems from the fact that both of the previous developers are not in frequent contact with the current team - they are not available on-site to coach new members on the structure of the code. The learning curve for an established system in an unknown programming language (LUA) was deemed too steep and time-consuming and thus started a search for alternative methods for improvement.

1.2 The purpose of the research

The purpose of this research is to adopt a new codebase for both improvements in robot motion and kick and to make future developments easier by providing access to the robot's core functionality via the programming language Python. Another benefit of the new codebase was the ease of installation - the set-up took noticeably less time than the full set-up of Austin Villa codebase used last year. The end goal is successfully competing in the international event RoboCup 2015, which will be held in Hefei, China and improving our results compared to last year's performance.

Chapter 2

Robot World Cup Initiative (RoboCup)

RoboCup is an international event held once a year with different leagues based on the competition. There are different leagues for humanoid robots, multi-wheeled soccer robots, simulated robotics and in 2008, there were also two leagues in SPL.

2.1 Goals of RoboCup

The main purpose of RoboCup is to promote research in the fields of both robotics and artificial intelligence. By providing a challenging long-term goal, it is also possible to popularize the field of robotics for future generations, thus providing more engineers and scientists to work on real-time robotics systems. The main problems that are currently being worked on include autonomous activity in a dynamic real-time environment, distributed control systems and non-symbolic sensor readings. Tackling these problems is vital to handling real-world complexities, even in a rather controlled environment. By introducing more random and complex factors with every revision to the rules, research must be conducted on new methods for sensor fusion, behavior, learning systems, multi-modal systems, vision and many more.

The end goal of RoboCup has been defined as the ”Landmark Project”, which has a 50-year deadline. It is stated, that by the year 2050, a team fully autonomous humanoid robots will beat the winners of the most recent FIFA World Cup. The idea is to follow in the footsteps of previous landmark achievements - 50 years from the first powered flight to the Apollo mission and another 50 years from the first digital computer to supercomputer Deep Blue beating the human chess champion Garry Kasparov. RoboCup Initiative aims to take the next big leap and accelerate the research done on both general robotics and humanoid robotics.[12]

2.2 Humanoid League

The first RoboCup humanoid league event happened in 2002, when even packing sufficient processing power on a humanoid robot was difficult. Even though there were existing humanoid robot platforms, such as Honda Asimo, they were unavailable and even if they had been available for purchase, the price tag would have been prohibitively expensive.[9]

2.3 Standard Platform League

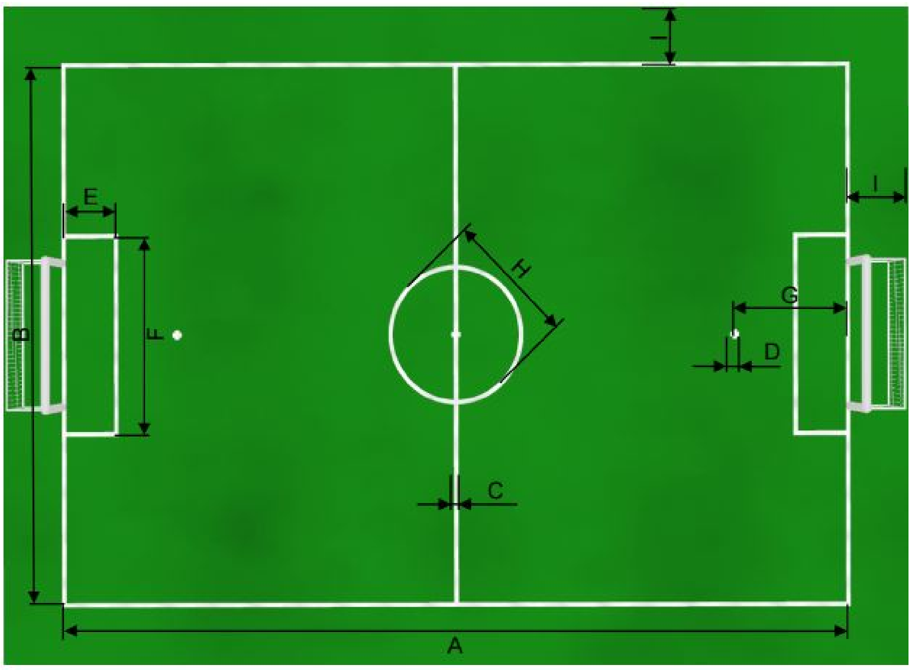
RoboCup Standard Platform league was held it's first competition in 1998, using the four-legged Sony Aibo robots. Since 1998, the rules, including the amount of players on the field and the field size have changed considerably. Since 2009 the use of Sony Aibo four-legged robots has been discontinued and for now only the Aldebaran Nao platform is used.[24][11]

2.3.1 SPL 2015 rules

According to the structure of the 2015 rule book, this section has been divided into four parts covering the most important rules of 2015 SPL - environment, robot players, game process, illegal actions and the major changes since 2014.

Environment

The size of the carpet used for the game is 10.4 meters by 7.4 meters, while the playing field itself is a rectangular area measuring 9 x 6 meters. There is a continuous white line running through the middle of the field along the y-axis of the field. In the center is a 1.5-meter diameter circle, inside which the ball will be placed during both the initial and subsequent kick-offs. The full specifications of the field can be found in Figure 2.1.



ID	Description	Length (in mm)	ID	Description	Length (in mm)
A	Field length	9000	E	Penalty area length	600
B	Field width	6000	F	Penalty area width	2200
C	Line width	50	G	Penalty mark distance	1300
D	Penalty mark size	100	H	Center circle diameter	1500
			I	Border strip width	700

Figure 2.1: The RoboCup SPL Field[10]

At each side of the field there is a goal with the following parameters : it has a height of 0.8m, width of 1.5m and a depth of 0.5m. The front of the goal is constructed from two goal-posts and a connecting crossbar with a 0.1m diameter.

The color of all the constructs of the gate except the net is white. The full construction of the gate can be seen in Figure 2.2. The colors used on the field are either white (for lines and goals) and green (the rest of the field). Lightning conditions provided by the venue are fixed - however, even though only ceiling lights may be used, the colors can be affected by outside light conditions caused by either the sun or failure of venue light bulbs. The ball used on the field is an orange street hockey ball with a 65mm diameter that weighs 55 grams. These balls are also available commercially.[10]

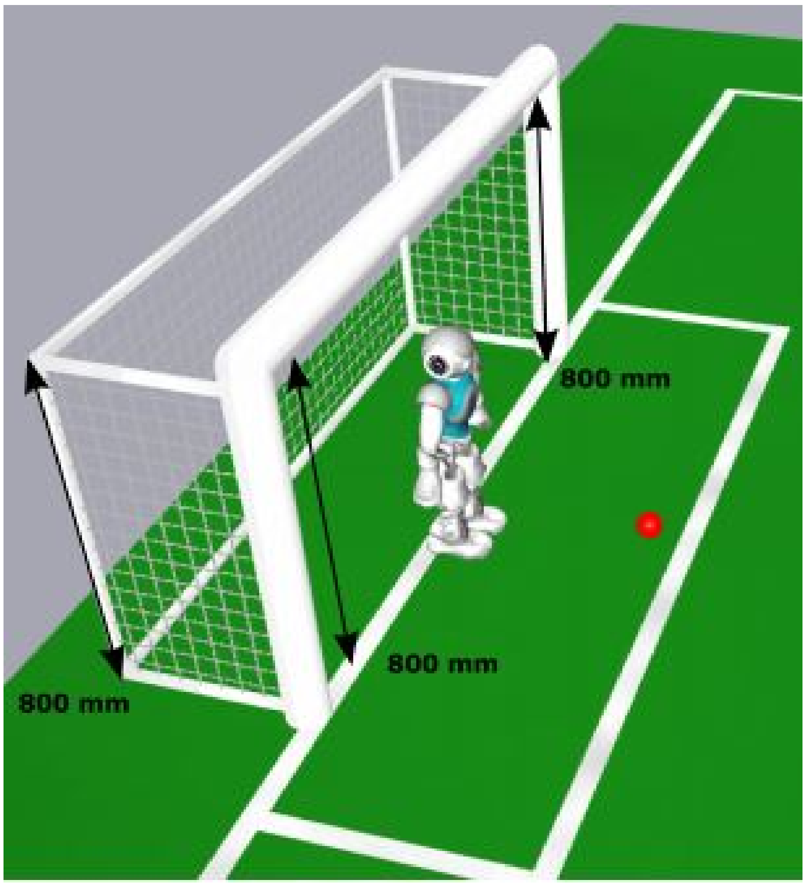


Figure 2.2: A 3D representation of the gate structure[10]

Players

There are a total of 10 players on the field, 5 in each team. The hardware of the robots is standardized, meaning all teams must use the same platform. The standard is an unmodified gray Aldebaran Nao robot - all hardware changes are illegal. This includes off-board sensing capabilities and off-board processing systems. However, there are some exceptions, although they are mainly cosmetic and some serve the purpose of providing quality of life improvements. Such of these changes include the adding of player numbers, sponsor logos, and adding robot and team names.[10]

The playing robots can be divided into two groups according to the rules, with a third type being a coach robot who is not counted among the five. The goalkeeper is only allowed to touch the ball with it's arms while inside the penalty box, but may otherwise kick the ball. Only one other robot (a field player) is allowed inside the penalty box during the game. The field players are numbered from 2 to 6, with 6 being the number for a substitute player. The goalkeeper is always labeled number 1.[10]

The field players can be separated by using colored jersey shirts. During the 2013 and 2014 RoboCup SPL, two main colors were used - cyan and magenta. In the 2015 rule book, each team can design their own jersey, but it must comply with the same rules as the previous ones. The style used is a non-restrictive tank top that must allow the chest LED to shine through. All the players on the same team must wear identical jerseys. The coloration of the jersey can have two different colors - one primary, which must be at least 80% of the total area. The rest of the jersey can be of a secondary color - team logos are also included in this 20%. The main restrictions on the main coloration state that the main color cannot be either of the field colors, orange, red nor light gray. The restrictions that apply to the secondary color are simpler - the only forbidden colors are orange, red or green. Only the coaching robot's jersey may ignore these rules, as the robot itself is not located on the field.

During the competition, the robots must play fully autonomously, meaning that no human input or control is allowed. However, the robots may communicate between themselves using either acoustic communication methods or networked messages. While there are no restrictions on the acoustic communications, the wireless system is does have some limitations. The wireless communication used the UDP protocol and each team gets one port - 10000 plus the team number. Using this channel, robots may communicate using the provided official access point. All teams will be assigned a range of static IPs, which will be announced along with the SSIDs and WEP keys at the competition site.[10]

The wireless messages are also standardized and the message format can be found in the file *RoboCupGameControlData.h*, which is provided along with the GameController program. Each robot is allowed to send a maximum of five messages per second, with the exception of the coaching robot, who is directly connected to the GameController interface and may send one message every 10 seconds. The messages sent by the coaching robot must also fit inside 80 bytes of data, be human-readable and include no numbers. The main structure for the coaching robot is also defined within a header packaged together with the GameController named SPLCoachMessage.h. If the coaching robot does not fill these criteria, it will be unable to assist the team in any way.[10]

Game Process

The general gameplay process is divided into two half-times and one half-time break, all of the stages lasting 10 minutes. During the half-time break, the sides are also changed. Before and during the game, the robots can switch between 6 different states - initial, ready, set, playing, penalized and finished. In the initial state, the robots are not allowed to move except standing up. In this state the team color and kick-off status can also be changed using the button interface. In the ready state, the robots walk to their positions on the field. This state can last for a maximum of 45 seconds and is ended earlier if the judge deems that there will be no more significant process. During the set state the robots must once again stop moving while waiting for the playing state signal. However, their heads may still move. During the playing state, the robots are playing soccer and pressing the chest button once will send them to the penalized state, during which the robot is not allowed to move in any manner. The last state is the finished state, which occurs after 10 minutes of gameplay. Each of these states must also be shown externally via the chest LED : off during initial and finished states, blue LED for initial, yellow for set, green for playing and red for penalized. The possible state transitions are graphically described in Figure 2.3.[10]

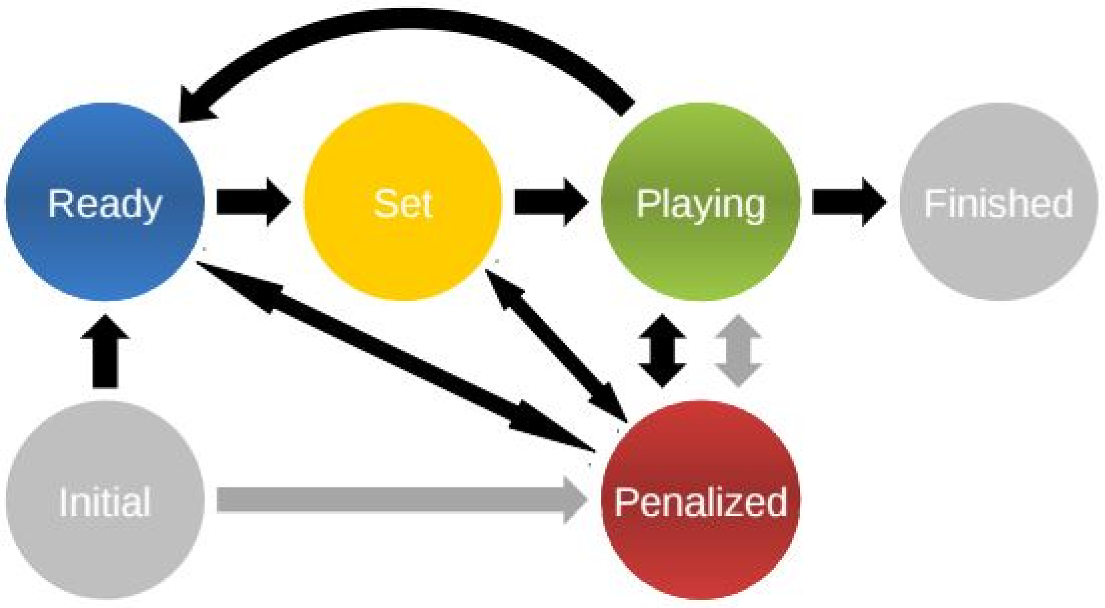


Figure 2.3: The GameController state machine.[10]

At any time during the game, a team may request a robot pick-up due to a hardware or software malfunction. During this time, all changes to hardware and software are allowed, and it is also possible to replace a broken robot with a substitute.[10]

At certain points during the game, such as after a goal has been struck, the game has been declared stuck or before the next half-time, either team is allowed to call for a timeout. This timeout has the same rules as the robot pick-up, meaning that the team can make changes to both robot hardware and software. Either team can call for one timeout per game and the maximum duration of said timeout is 5 minutes.[10]

Illegal actions

During the game, there are various illegal actions that the robot may not perform. The first involve types of locomotion other than walking on two legs, such as crawling. If a robot displays such behaviour, they are removed with the request for pick-up ruleset. At any point during the game, team members are not allowed to interact with the robots manually, either by physical contact or some other communication methods. Another illegal move after a certain time is ball holding. A field player can hold the ball for up to 3 seconds, while the goalkeeper may do so for up to 10. Ball holding is defined with the help of the robot's convex hull projection onto the ground. If over half of the ball falls inside this area, it is described as ball holding. Violating this rule results in standard removal penalty.
Other illegal activities include :

- Robots failing to stand up after an extended amount of time
- Taking a stance wider than the robot's shoulders for over 5 seconds
- Having the coach robot leave it's seated position
- Pushing other players forcefully
- Having a field player or a goalkeeper use it's arms to touch a ball outside the penalty box
- Damaging the field or being deemed as a threat to any spectators
- Leaving the field carpet
- Entering the penalty area after two robots including a goalkeeper are inside of it
- Jamming either wireless, acoustic or visual information

Most infractions to these rules result in the robot being removed in accordance to the standard robot removal procedure, which states that a robot will be put in the penalized state and then let back into gameplay after 45 seconds. After the 45 seconds have passed, the robots may re-enter the field from their own side and from the sideline further away from the ball.[10]

Rule changes 2015

Every year, some new technical challenges are introduced into the rules. This year, the major rule changes include changes to the goal colors, jersey designs, coach limitations, lighting conditions and starting signals. As of 2015, both of the goals are colored white - the same color as the field lines. Each team may design their own jersey with certain limitations, described in field player paragraph. The coach can now send up to 80 bytes of data, up from a previous 40. The messages sent are not delayed anymore either. From this year onward, the changes in lighting are not controlled anymore - people may perform flash photography and the outside lighting is not accounted for anymore. Finally, from the quarter finals onwards, the game starts with a whistle - only after a 15 second delay is the GameController signal sent to the robots.[26]

Chapter 3

Nao humanoid robot platform

Aldebaran Nao is a general-purpose humanoid robot meant for use in various fields - be it as an interactive assistant, for use in education or high-end robotics research.[18] In 2013 Aldebaran launched an initiative called ”Autism Solution for Kids”, which aims to use the Nao platform to help teach young children with autism.[4] In this work both the 21-DoF (Degree of Freedom) and 25-DoF 2012 NaoV4 robots were used.

3.1 Hardware of the Nao platform

The main processor, located inside the robot’s head, is a 1.6GHz Z530 Intel Atom. The whole system is powered by a 27.6 watt-hour battery. The newer Nao V5 system has an improved battery that has an energy storage capability of 48.6 watt-hours.[1] The storage on board the system includes 1GB of Random Access Memory (RAM), 2GB of flash memory for system usage and up to 8GB of flash memory for user dedicated purposes.[17][5] The Nao platforms used have 1 to 3 tactile sensors, depending on the degrees of freedom, 2 cameras which can capture up to 30 frames per second at a 1280x960 resolution in YUV422 color coding[27], four pressure sensors under each foot, two sonar devices, a speaker on each side of the head and two microphones.[22] The Inertial Measurement Unit (IMU), which helps stabilize the robot has two gyroscopes and one three-axis accelerometer. The gyroscopes provide 5% precision at an angular speed of approximately 500 degrees per second and the accelerometer provides 1% of accuracy at an approximate acceleration of 2G.[16]

3.2 Software on the Nao

The main operating system used in this project is the OpenNAO 1.14.5 - it is an embedded Linux distribution based on Gentoo that has been developed to fit the specific needs of the Nao platform. The default username and password are always ”nao” and the default administrator account is root. By default, connecting to the robot is done over SSH (Secure Shell). However, it must be noted, that while one can use any account to log into the robot, connecting as a root user over ssh is impossible. The OpenNAO platform provides various linux programs, such as connman, which is used for managing the networks and NAOQi, the core software that allows reading robot data and manipulating the robot.[21]

NAOQi is a cross-platform, cross-language system that allows the developer to create systems in either C++ or Python and on Linux, Windows and Mac. The NAOQi system acts as a broker for the low-level libraries. The NAOQi loads the names of the required modules on startup from a file called ”autoload.ini”, which loads the libraries required for the requested modules. The module itself is typically a , providing functions to be used by the developer. This concept is illustrated in Figure 3.1.

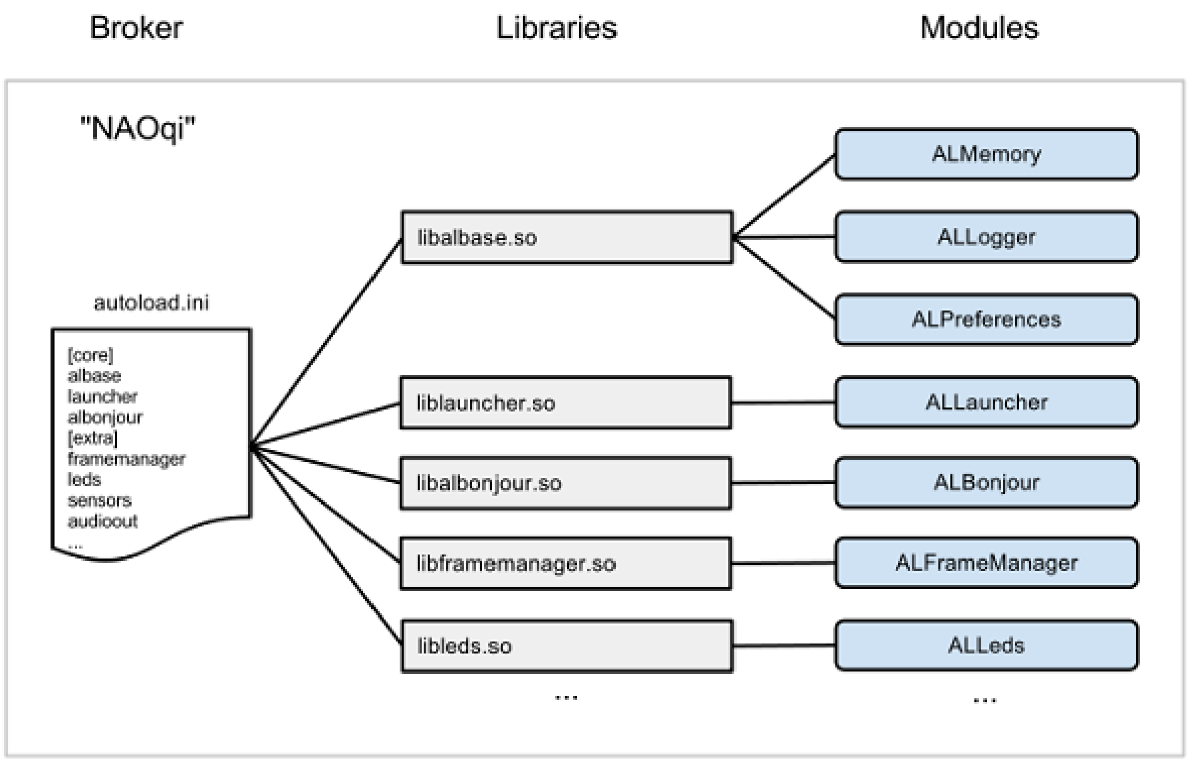


Figure 3.1: Structure of the NAOQi modules.[20]

Depending on the way they are loaded, these modules can be divided into two categories : remote and local. Remote modules are compiled executable files which can run outside the robot and are easier to debug, but at the cost of being less efficient. Local modules are compiled as libraries and cannot be used in an environment outside the actual robot. All of the calls to the modules can be made as either blocking calls or non-blocking calls. Blocking calls wait for the process to complete until returning a value, while non-blocking calls start a new thread and allow the main process to do other things while waiting for a return value. In the NAOQi environment, all modules can share the same memory. located in ALMemory, which provides thread-safe access to multiple modules at once.[20]

Chapter 4

Overview of the system used

4.1 Working environment and tools

For the purpose of generating code, compiling, testing and loading it onto the robots a Linux operating system distribution was installed. In this thesis the author has chosen to use Linux Mint v16 ”Petra” 32-bit edition. The initial setup was done by installing the operating system on a new partition instead of putting the development environment on a virtual machine. For setting up the development environment, developer tools such as Eclipse Luna, JetBrains PyCharm and SmartGit were installed. The setup of the codebase only required the user to install three dependencies - zlib, glib and cmake. The developers from rUNSWift have created a cmake script that allows us to bypass the manual part of the setup - it will try to automatically generate the necessary file structures, download the missing libraries and install the cross toolchain that will be later used for cross-platform compilation. The pre-existing scripts also allow us to easily load the code onto Nao robots without any tools but the Linux bash.

For calibrating and receiving real-time information or saving it for later use, a tool called ”offnao”, developed by team rUNSWift of the University of New South Wales, was used. The tool offers mostly the same functionality as UTNaotool used in the previous year does, but it has also fixed some problems, such as the program crashing if certain tabs are activated in the wrong order. The tool provides access to any Nao robot currently running rUNSWift soccer software and on the same network as the host computer. The system is able to stream raw images, location data, sensor data and the robot log files in real-time and can also save the stream data for later use. The offnao tool is also used to calibrate the robot vision and sensors by invoking a specialized behaviour type and reading the offset values from the sensor tab. The general layout of the offnao tool can be seen in Figure 4.1.

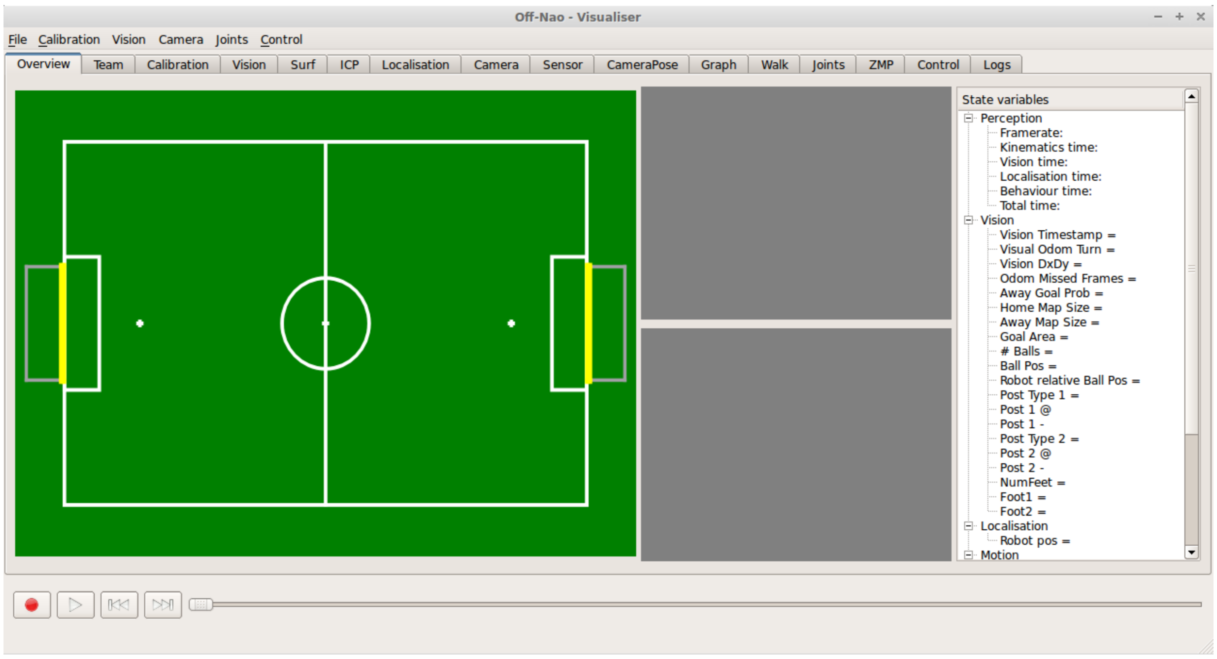


Figure 4.1: The offnao debugging and control tool.

4.2 Modules

The main test field is a scaled-down version of the original RoboCup SPL carpet, measuring 80% of the original carpet in every way. Most of the tests have been conducted under controlled lighting conditions. As the carpet itself is wrinkled in many places, it has given us the ability to test the walking behaviours in extreme environments.

The language used in development was mostly Python, as the C++ modules worked without any problems. There was also no need to adjust the existing modules. The C++ core engine is exposed to the Python via Boost Python libraries, giving Python a high-level access to the core via a central blackboard. The Python part of the soccer software can also send out it's own information into the C++ engine. The Boost Python library allows for easy reconfiguration of exposed modules and also allows function calls from either Python to C++ or vice versa.[7]

4.3 General architecture of the rUNSWift codebase

In the system used, all robots act on their own - each one has their own finite state machine, meaning that they all obtain information on their own, rather than depending on a central controller. The only information that the robots communicate amongst themselves are their positions and each robot's calculated ball position. On a lower level, the rUNSWift architecture uses an abstraction layer called ”libagent” over the NAOQi-provided Device Communication Manager (DCM), which in turn provides access to individual motor and sensor values. The libagent communicates with the main core using a shared memory block - similar to the structure used by Austin Villa. The runswift executable is not directly tied to the NAOQi broker, but instead uses the information provided by libagent. This allows us to turn off the motion thread for complete off-line testing. The runswift binary runs six main threads - perception, motion, off-nao transmitter, naotransmitter, naoreceiver and gamecontroller receiver. Most of these threads have frame limiters, such as the naotransmitter, which can only run at 5 frames per second (5 messages per second rule). The general flow of information is described in Figure 4.2.[3]

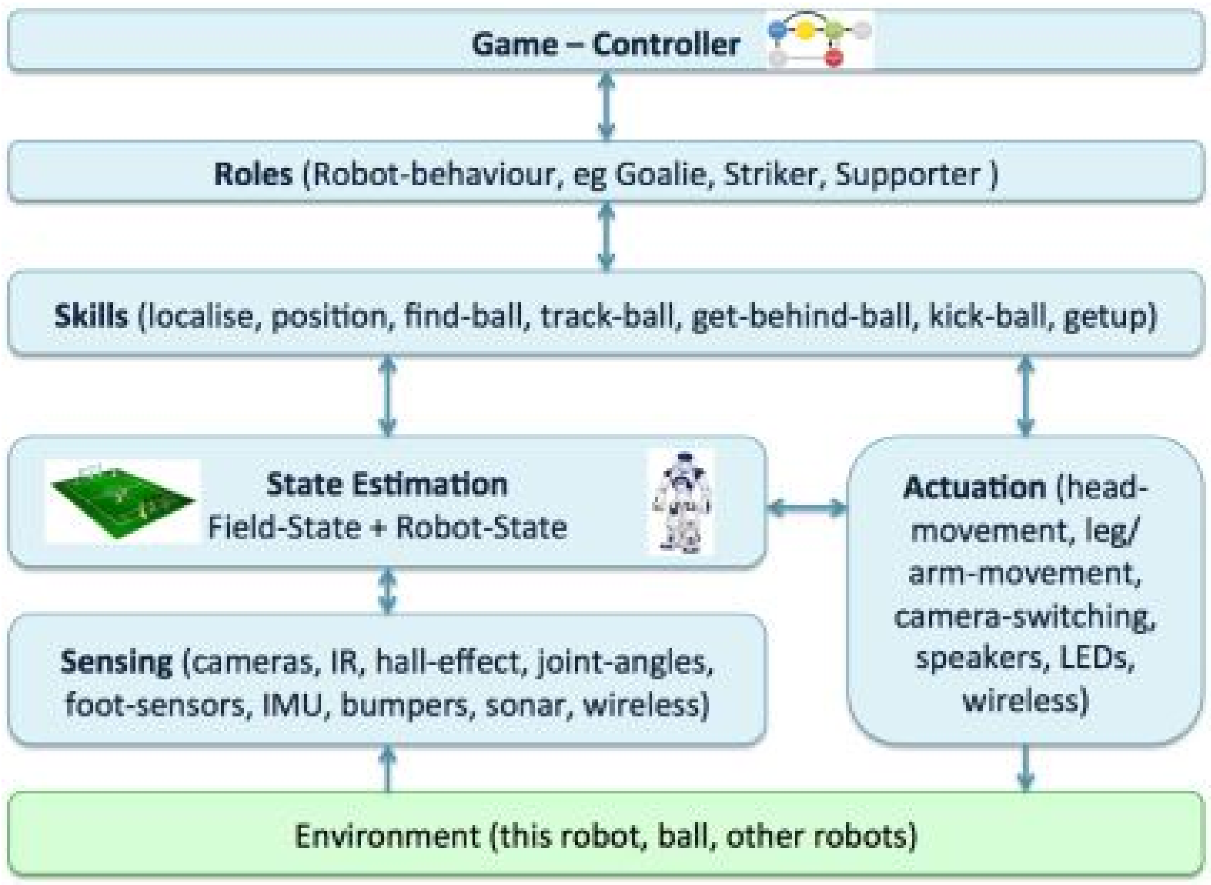

Game – Controller
Roles (Robot-behaviour, eg Goalie, Striker, Supporter)
Skills (localise, position, find-ball, track-ball, get-behind-ball, kick-ball, getup)
State Estimation
Field-State + Robot-State
Actuation (head-movement, leg/ arm-movement, camera-switching, speakers, LEDs, wireless)
Sensing (cameras, IR, hall-effect, joint-angles, foot-sensors, IMU, bumpers, sonar, wireless)
Environment (this robot, ball, other robots)

Figure 4.2: The architecture of the rUNSWift codebase.[3]

4.4 Modules

The rUNSWift architecture distinguishes between six different modules, each run inside a different thread with a certain frame limiter. All of the threads communicate with a central blackboard, which also acts as an access point for Python behaviours. As some of these threads only handle sending information over the network to other robots or receiving it, they will not be discussed in further detail. However, both the perception and motion threads will be discussed in further detail, as most of the core engine research and development conducted by team rUNSWift has been conducted on these.

4.4.1 Perception

In general, perception handles processing the read image and then generating useful data based on further object detection and filtering. On a lower level, it could be divided into three distinct parts - image acquisition, object detection and localization. This thread could be classified as the most important thread, as vision is the primary sensory input for the whole game process.

As the Nao platform does not offer a lot of computational power, the vision process must be well-optimized on a low level. In rUNSWift architecture, this is done by first sub-sampling the image and generating saliency images. This process allows to save a lot of otherwise unnecessary processing power while still retaining enough information for object detection. Even though the main object recognition is still greatly based on colors, the rUNSWift team has added an additional layer of detection by using the the edges found in the image. This allows for some robustness due to poor venue lighting, which will most likely play a more substantial role than ever before in 2015.[26][3]

For object detection the system relies on multiple algorithms. For detecting field lines, circles and edges, Random Sample Consensus (RANSAC) is used. Line intersections found using this algorithm are further used in localization. For ball detection, first the main fovea areas of both bottom and top camera are analyzed. If the ball is not found within those areas, after which the algorithm analyzes the color histograms of pre-generated sub-sampled images. After an area of interest has been found, the edges of the area are mapped and by using RANSAC a circle fitting is attempted.[8]

Since both of the goals are the same color, an algorithm called Speeded Up Robust Features (SURF) is used. As the algorithm is too expensive to be used on a 2-dimensional image in real-time, only a one-dimensional line is used. SURF uses a pre-built visual information database to compare the 1D vector extracted from the image to obtain information on gate position. SURF is also used as the base for a visual compass to track which way the robot is facing. This can be used to decide if the goal belongs to our team or the enemy team.[2]

For robot localization, a multi-modal extended kalman filter is used. It uses the information from the camera (line intersections, central circle) and previous states. The kalman filter is also used to track the ball and retain it's position even if the robot cannot see it. Thus, it can be said that the kalman filter tracks a total of 7 parameters : x,y coordinates of the robot, heading of the robot, x and y coordinates of the ball and the velocity of the ball in both directions. On every observation, noise is added to the measurements taken. To get better results, a list of independent states is used, where every state has it's own state mean vector and a covariance matrix. Each of these states has it's own weight, which indicates the confidence of the mode matching the true state (multi-modal filter). To further enhance the results, the localization results of other robots on the network are also used. This filter is broadcasts the filter results 5 times over the network to other robots, which use the broadcast data as a new observation.[13]

4.4.2 Motion

The motion module is run at 100 frames per second - the reason for this, according to the team's official documentation, is to maintain the stability of the robot. The motion module can be divided into three submodules - one for reading the sensor values, one for generating new joint values based on sensor input and requests and the final system which uses the generated values to change the robot's stature.[3]

The modules are all connected together via a central MotionAdapter - on their own, they act as independent modules, all accepting input and offering output to a centralized controller, meaning they have no direct connection between each other. The MotionAdapter owns Touch, Generator and Effector systems as objects, using function calls to pass information between them. The MotionAdapter exchanges this information with the blackboard - a watcher thread runs the whole cycle every 10 milliseconds. The whole flow of information can be seen illustrated in Figure 4.3.[14]

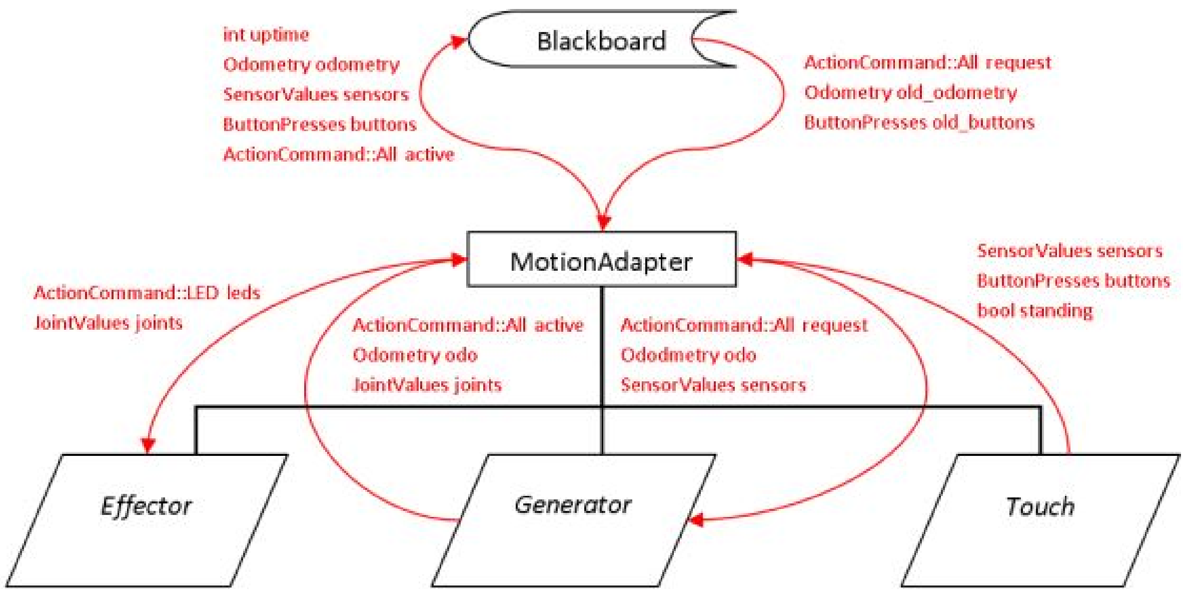


Figure 4.3: The architecture of the motion engine.[14]

The walk processor generates the motions according to three parameters - forward, left and turn. The maximal values accepted for each are 300, 200 and 1.5. Both forward and left parameters are measured in millimeters per second and the turn is measured in radians per second. The walking system itself is an open-loop omni-directional walk, meaning the robot can perform backwards turns and other complicated movements. This walk processor can be easily accessed via action requests from the Python level.[14]

4.5 Behaviour module

All of the high level behaviour is managed from a system written in Python. Using Boost's Python library, the blackboard is exposed to the Python and in a C++ adapter, the Python behaviour module is activated on each tick, happening at 30 frames per second. Inside the behaviour module, during the initial function call, a World object is created, which will be passed to the first loaded skill - usually Default. How the World object is handled afterwards, is left to the developer. This structure allows the system to only update the blackboard inside the World on each tick, making the system simpler. The behaviour module comes packaged with multiple helper systems, such as a system for reading global access data (*Global.py*), a field- and player constant file (*Constants.py*) and other various utilities (vectors, field geometry classes, mathematical utilities and a timer). This thesis was largely based around building new behavioural systems and re-writing some existing structures for ease of access.[23]

Chapter 5

System development

5.1 Methodology

As it is difficult to evaluate the abilities of a new codebase, we could not establish a metric for most of research and development. There were many factors which came into play - the documentation of the system, the readability of the core code and the pace at which new behaviour code can be written at a high level. Since the codebase uses Python as the high-level behaviour language, new team members can also be taught the basics of the structure quicker than before and the object-oriented structure available in Python offers a lot more capabilities than Lua.

5.2 Initial assessments and trials

The initial assessment on the viability of changing the codebase was conducted based on comparisons - the deciding factors were the readability of the core code, the documentation available (including comments) and the behaviour level language and link. As the documentation provided with the previous codebase proved to be lacking, it was difficult to understand the system without any external advice provided by the developers who had worked on it. Same could be applied to the behaviour modules, which lacked comments and a clear documentation. In comparison, rUNSWift codebase had a github wiki, which described in great detail how the modules worked and how to write code based on their system.

Another important factor was the set-up time and difficulty. As Austin Villa codebase required some libraries that were not available anymore in the linux package managers, they required manual compilation, which can be at times difficult and time-consuming. Due to this fact the development environment was set up as a virtual machine, which could at times lead to networking issues and slows down the development on older machines (time taken to compile the code, development tools are less responsive etc). By using the documentation and scripts provided by rUNSWift team, it was possible to have a fully functioning installation of Linux with all the necessary tools set up within approximately 2 hours.

The first trials involved calling basic behaviours - walking and kicking. It took a total of approximately three weeks to get from the first contact with the new system to the robot successfully performing a kick - something which took approximately two months for the developers working with Austin Villa codebase. As the initial results for both kick strength and walk speed looked promising, it was decided that for the best results and future potential, a transition to the new codebase is necessary.

5.3 Documenting the blackboard interface

Blackboard is the central information management system, much like the shared memory used in Austin Villa code. As there was no direct documentation on what the blackboard contained, a github wiki was created, where each of the submodules in blackboard was then documented. For most of the blackboard modules, the data types were also included. This process allowed us to quickly find information that would have required going through header files before. This module can be seen in Figure 5.1.[6]

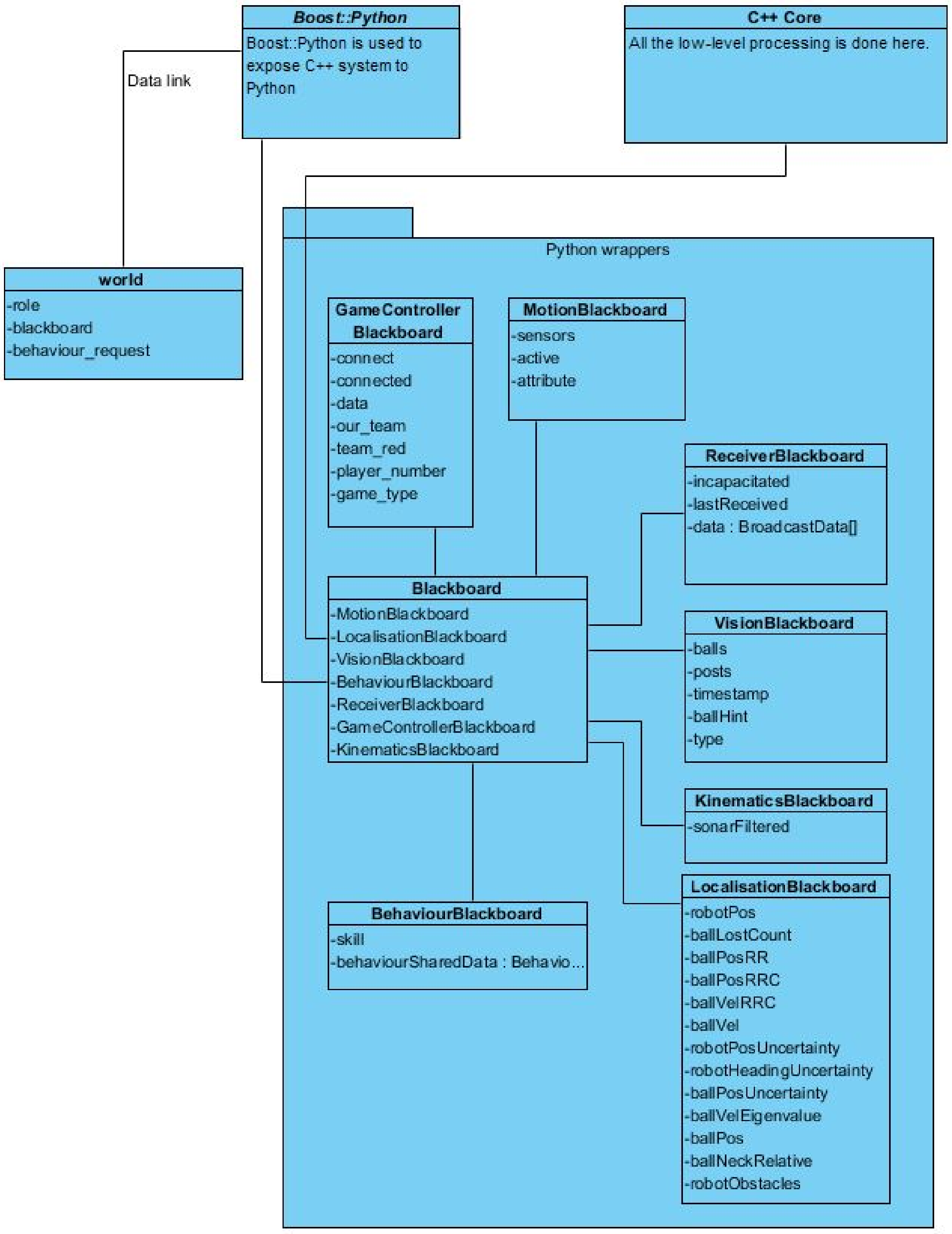


Figure 5.1: The structure of the blackboard and it's connections to the system.

5.4 Documenting the motor sensor array

As the blackboard provides motor position is only provided as an array filled with values, they needed to be documented. To do this, we created a python system that would print out the values of each joint along with it's number and documented them in the public github wiki.

5.5 Base behaviour code structure

If the runswift executable is loaded without any parameters, the behaviour code executed in default will be the first Python file in the roles folder. To get the robots to play a specific role, a new file, called "robotname.role" was created. During the initial loading of Python modules, this file is read and the value inside is stored as the robot's role. These can value from 1 to 5, with 1 being the goalkeeper. The Default behaviour handles the transitions between GameController states - each time the state is changed, a new behaviour is created and loaded. All the robots share the common states except Playing - the reason for that being future development, where each role will have a dynamic behaviour and the robots are able to change their roles. The current set-up for the behaviour structure can be seen in Figure 5.2.

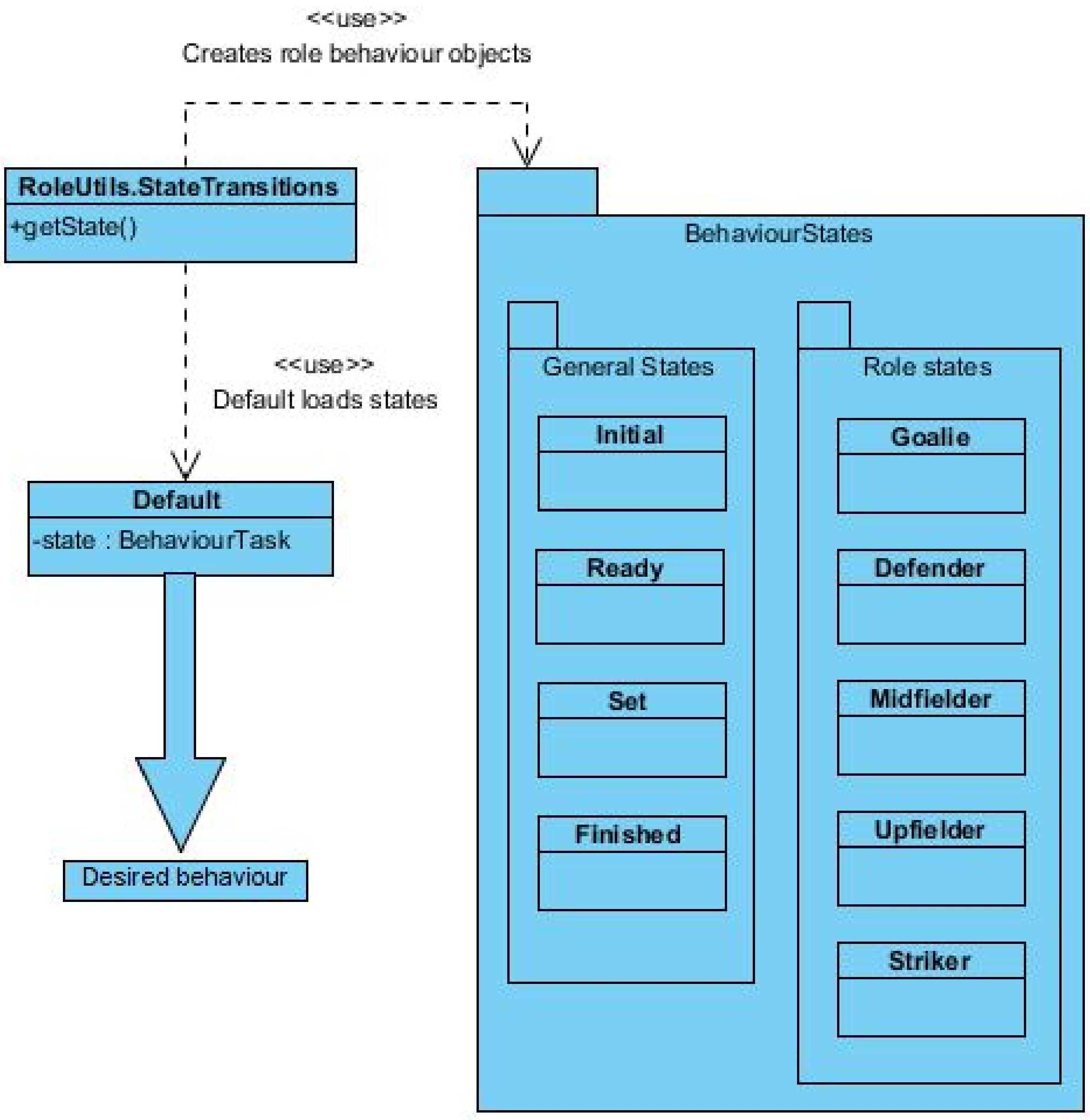


Figure 5.2: The behaviour loading system.

5.6 Low level behaviours

Low level behaviours are defined as basic behaviours written in Python - while they still use the C++ core functionality to fulfill their task, they are used by other high level behaviour tasks. Such tasks in the current system include lining up a kick, scanning the field for either localization purposes or finding the ball and walking from point A to point B.

5.6.1 Walking from point to point

For walking from the robot's point to another point on the field, a system called "Walk2Point" was used. This system can accept field coordinates or robot-relative coordinates, but it does not try to limit the speeds given to the C++ core. The solution was to use a simplex shape to divide the inputs. While this may limit the robot speed in extreme environments (high turn and movement values requested), it helps stabilize the robot's walk. The scaling can be described by the following formula :
Given parameters forward (f), left (l) and turn (t)

$$sumDiv = (f/maxF + l/maxL + t/maxT)$$

$$\begin{cases} f = f; l = l; t = t & sumDiv < 1.1 \\ f = \frac{f}{sumDiv}; l = \frac{l}{sumDiv}; t = \frac{t}{sumDiv}t & sumDiv >= 1.1 \end{cases}$$

By applying these rules, a large increase in the stability of the walk and turn was noticed. However, it may result in unwanted behaviour such as moving sideways.

5.6.2 Scanning

In general, the scanning can be divided into three distinct sub-behaviours :

- Tracking
- Scanning
- Localizing

The tracking module is meant for holding a stable ball position if the ball can be seen. This is done by continuously calculating the necessary next frame yaw and pitch positions and using the same parameters to calculate the speeds for head movement.

In scanning, an experimentally-deduced scan speed is used to scan the whole possible field of view from one side to the other. If the head reaches one maximum, the vertical position is also changed so that the robot could perform the scan on the area right in front of it.

The localization scan is the slowest scan - it is similar to the ball-finding scan, but as localization relies on features that are easier to confirm in a stable image over multiple frames. To achieve this, the scan behaviour moves the head in around 45-degree increments and then stops for some time. This gives the localization module over 10 frames to process and improves the results greatly.

5.6.3 Lining up a kick

The initial aiming system was developed to imitate the system used in Robotex soccer robots - after gaining control of the ball, the robot rotates until a gate is found and then prepares the kick. This proved to be unreliable, as the robot may not even see the gate, which would cause the robot to get stuck. The next version featured an aiming vector and finding the robot's location in relation to the vector. This vector was constructed between the ball and the gate. The aiming algorithm used the robot's distance from the vector and an additional check for finding if the robot is on the wrong side of the vector. If the distance to the vector was smaller than a set threshold, the kick would take place. However, this proved problematic as the distance from the aiming vector varied greatly.

The current solution was developed by using two aiming vectors - one between the ball and the target, and another one between the robot and the ball. By constructing this system, it is possible to use the following formula to find the cosine of the angle between the robot and the ball to target line. The formula to find the cosine is as follows (u and v represent the 2 vectors) :

$$cos(\varphi) = \frac{\overrightarrow{u} \times \overrightarrow{v}}{\|u\| \times \|v\|}$$

Now it is possible to turn to only one side based on where the robot is currently situated to reach the kick position without walking around the whole ball : if the angle is over 180 degrees, walk to whichever side takes the robot towards 360 degrees and if the robot is in the 0-180 degree range, walk until the robot reaches the kick threshold. Testing has proven the accuracy of this algorithm, and with a stable enough localization it can hit the ball towards the intended target most of the time.

5.7 High level behaviour structure

On the highest level, the behaviour is divided into different states, one for each GameController state and one playing state per robot's role. While each of these role-specific states is currently only individual behaviour, which is the same across all robots, they were created with future developments in mind, as it is easier to base new developments on this structure.

5.7.1 Initial state

During the initial state, the robot uses a low stiffness stand to conserve power. In this state the robot is not allowed to move it's head.

5.7.2 Ready state

In the ready state, the robots are allowed to move to their designated starting positions. This state is also limited by time - meaning that the localization is very important to reach the designated points in time. The robot uses an array of points stored in Constants which tells each role where it's starting position is, based on which team has the kickoff. If our team has the kickoff, it is very important to note that the ready state positioning plays an even greater role - it allows the robot to get the first kick off without any problems and then return to playing the game.

5.7.3 Playing state

The playing state can start off with either our kickoff or the enemy team kickoff. As the structure used uses a central task management system for checking if the robot is in it's own zone and localized, it cannot be used while the kickoff timer is still active or it is our kickoff. When the kickoff is done, the two front line robots are allowed to move into their respective zones. As the previous year's experience proved, team behaviour in a noisy environment is difficult - it is much simpler to implement a robust system based on completely individual behaviour.

Finished state

In the finished state the only important part is to stop the locomotion to conserve battery life and preserve the joints. In that sense, it is similar to initial.

Individual strategy

As the individual strategy defined in in 2014 proved to work reasonably well, it was re-implemented. It is based on two defending robots near the team's own goal and two robots on who are playing as the offensive players on the other side of the field. The zones are defined by using the central point on the field and field-relative coordinates, such as field length and width, which are defined in a header file. These field zones can be seen in Figure 5.3.

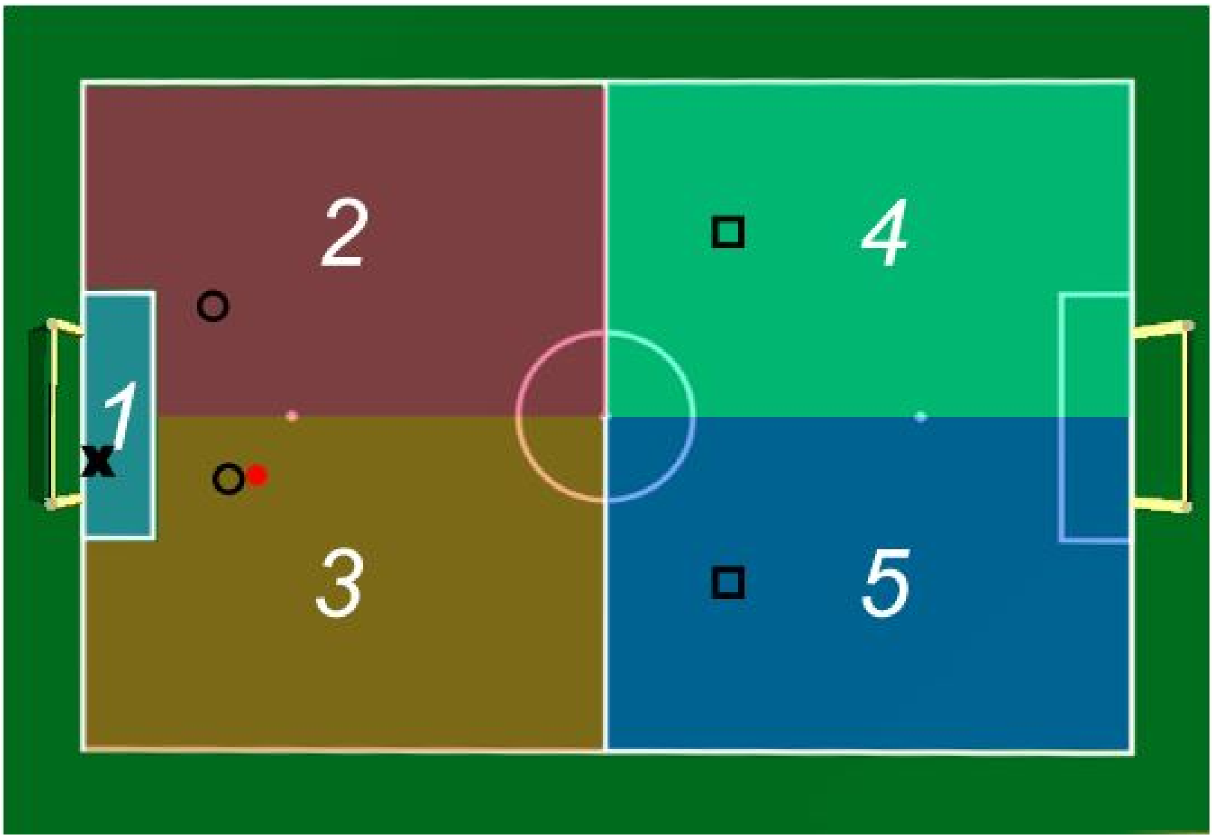


Figure 5.3: The field zones - circles represent defenders and squares represent attackers.[15]

The defending robots stand in their zones, passively waiting for the ball to pass the central line, at which point they will actively try to kick the ball back to their opponent's side. The robots on the offense try to actively chase the ball and kick it into the goal.

The main upgrades that have been implemented so far involve new functionality that enables the robot to detect obstacles and provide information on the nature of the obstacle. This was made possible by the system designed by rUNSWift that uses multiple sensor input to classify obstacles such as robots and goals posts. Even though the use of this system is limited in the current strategy, we can use the distance information to evaluate the distance between the ball and our team's other player at a close range. Doing this enables us to actively prevent two of our own robots approaching the same ball at close range. The current state machine prototype for individual strategy can be seen in figure 5.4.

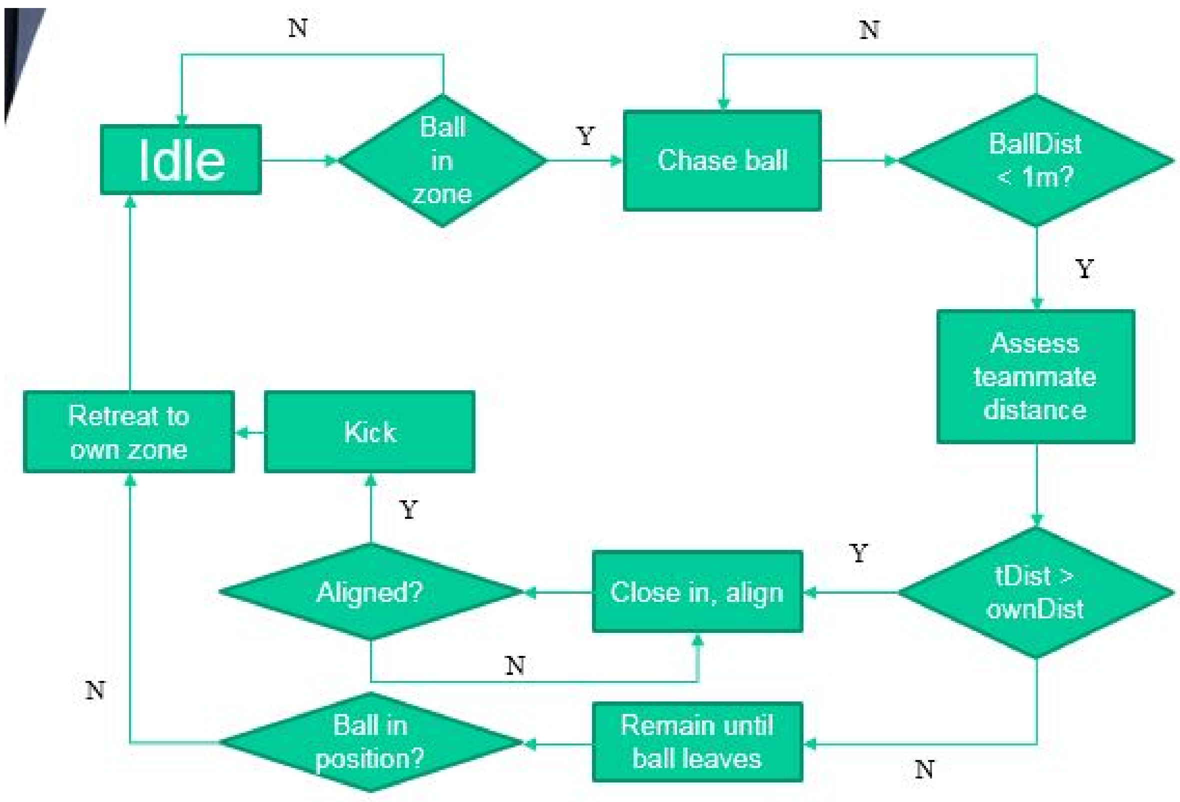


Figure 5.4: The state machine used as a prototype.

However, it must be mentioned that any and all results prior to RoboCup 2015 cannot be considered conclusive as it is impossible to test the systems without actual gameplay in a proper 5 versus 5 match with RoboCup conditions.

The strategy also needs to be able to handle some specific rules, especially the ones related to player pushing and penalty zones, along with other special cases (localization failure, finding the robot outside it's own zone). For this reason, a task manager is used.

Task manager

To prevent any illegal actions, a general-purpose task manager was designed. The task manager handles the transition between gameplay and general tasks, such as moving the robot back inside it's own zone and localizing. To do this, all the gameplay tasks are wrapped inside a management-level behaviour task, which then handles the calls to both the task manager and gameplay tasks. The management-level behaviour task calls the task manager on each tick to see if there is an active task or if a task has just been finished. In the transition stage, which is generally meant to transition between different states, the current_task inside the management system is updated accordingly. If the task manager has just exited a task, the task manager's GetGameplayState will be used to generate a new gameplay state and during the normal process, each gameplay state has to handle it's own transitions. During each tick, the task manager is called first and then a check is made to see if there is an active task. If there is no task active, the gameplay behaviour is executed. The full schematic of the Task Manager is detailed in Figure 5.5.

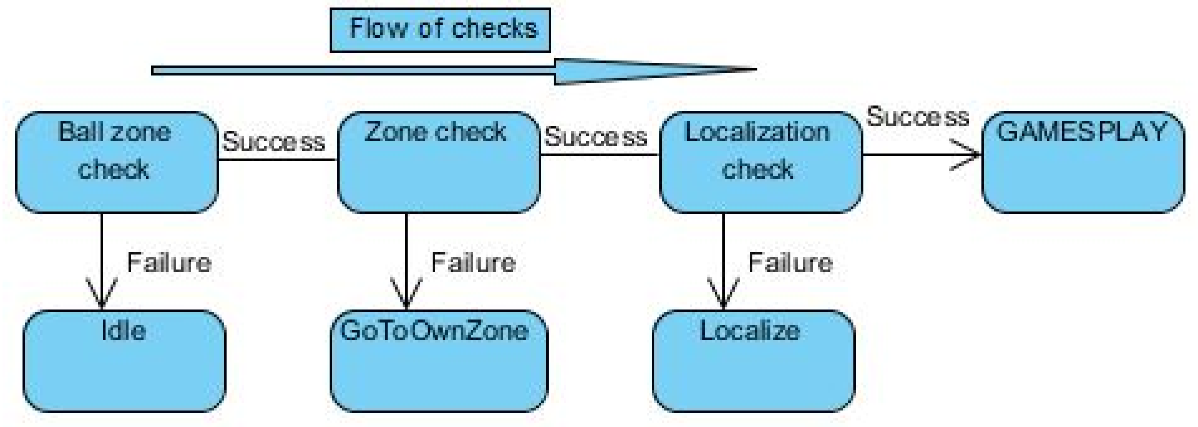


Figure 5.5: Control flow of the Task Manager

This system design allowed us to save a lot of space in individual behaviour code and prevent the main state machine from getting overly complicated. When compared to the state machine used in the previous year, the main state only has 6 states and the rest is being managed by the task manager.

Summary

The RoboCup Standard Platform League has two teams, each consisting of five robots play football against each other in a semi-controlled setting. The robots used have the same hardware and modifications are not allowed.

The purpose of this thesis was to find a method to improve the overall performance displayed during 2014 RoboCup and implement the method(s).

During the course of the project, a new codebase, developed by team rUNSWift, was evaluated, tested and then adopted as it offered improvements compared to the Austin Villa codebase used in 2014. As the codebase offered only basic core functionality, a behaviour module needed to be implemented to offer both low- and high-level behaviours. The behaviours developed provide low-level functionality for movement, ball alignment and targeting and high-level functionality for basic soccer gameplay according to RoboCup 2015 rules.

The individual strategy mimics the system used in 2014 with the main difference being the ability to recognize our teammates and then use that information to avoid collisions while trying to hit a ball that is in the common playing area of the two robots.

The kick and walk performance appear more stable, as they are both dynamically generated using rUNSWift's motion system. The walk is also offers greater configurability and needs careful calibration for tuning the input parameters.

Kokkuvõte

Meeskonna rUNSWift süsteemi põhjal käitumisloogika arendamine 2015 RoboCup võistluse jaoks.

RoboCup Standardplatvormi liigas vistlevad kaks viiest robotist koosnevat meeskonda omavahel jalgpalli poolkontrollitud tingimustes. Kõigil robotitel on sama riistvara ning seda ei tohi muuta.

Käesoleva bakalaureusetöö eesmärgiks oli leida meetod 2014 RoboCupil demonstreeritud oskuste parandamiseks ning seejärel viia ellu vajalikud muudatused.

Projekti käigus hinnati, testiti ning implementeeriti meeskonna rUNSWift kirjutatud koodibaas, mis pakkus uuendusi võrreldes 2014 Austin Villa koodibaasiga. Kuna süüsteem pakkus vaid tuumikfunktsionaalsust, oli vaja arendada tühja käitumismooduli peale nii madala- kui ka kõrge taseme käitumisloogika. Madala taseme loogika pakub meetodeid liikumiseks, palli suhtes joondumiseks ning selle sihtimiseks. Kõrge taseme loogika käsitleb tegelikku jalgpallimängu kasutades madala taseme süsteeme ning jälgides RoboCup 2015 SPL reeglistikku.

Individuaalstrateegia on enamjaolt 2014 süsteemi koopia, pakkudes uuenduslikkust meeskonnaliikmete ning muude takistuste vältimiseks. Selle info abil on õimalik vältida kokkupõrkeid pallilöögi ajal, kui pall peaks olema kahe roboti jagatud tsoonis.

Löögi- ning kõndimisoskused on märgatavalt stabiilsemad, sest nad on alati dünaamiliselt koostatud, kasutades rUNSWift'i liikumisgeneraatorit. Kõndimine pakub palju rohkem konfiguratsioonivõimalusi ning tänu sellele vajab hoolikat kalibratsiooni sisendparameetrite valikul.

Bibliography

[1] Aldebaran. *Unveiling of NAO Evolution: a stronger robot and a more comprehensive operating system*. URL: https://www.aldebaran.com/sites/aldebaran/files/press-releases/cp_nao_evolution_en_def.pdf (visited on 04/10/2015).

[2] Peter Anderson and Bernhard Hengst. *Fast Monocular Visual Compass for a Computationally Limited Robot*. 2013. URL: http://cgi.cse.unsw.edu.au/~robocup/2014ChampionTeamPaperReports/20120000-AndersonHengst-VisualCompass-RobocupSymposium2013.pdf.

[3] Jayen Ashar et al. *RoboCup SPL 2014 Champion Team Paper*. 2014. URL: http://cgi.cse.unsw.edu.au/~robocup/2014ChampionTeamPaperReports/20141221-SPL2014ChampionTeamPaper.pdf.

[4] *Ask Nao—New Way of Teaching?* URL: https://asknao.aldebaran.com/ (visited on 04/10/2015).

[5] *Battery*. URL: http://doc.aldebaran.com/1-14/family/robots/battery_robot.html (visited on 04/12/2015).

[6] *Blackboard*. URL: https://github.com/TeamPhilosopher/philosopher-2015/wiki/Blackboard (visited on 05/07/2015).

[7] *Boost.Python Index*. URL: http://www.boost.org/doc/libs/1_58_0/libs/python/doc/index.html (visited on 04/05/2015).

[8] Carl Chatfield. *rUNSWift 2011 Vision System: A Foveated Vision System for Robotic Soccer*. 2011. URL: http://cgi.cse.unsw.edu.au/~robocup/2014ChampionTeamPaperR 20110825-Carl.Chatfield-VisionFoveated.pdf.

[9] Humanoid League Technical Committee. *Development of the league*. URL: http://www.robocuphumanoid.org/league/history/ (visited on 04/07/2015).

[10] RoboCup Technical Committee. *RoboCup Standard Platform League (Nao) Rule Book*. URL: http://www.informatik.uni-bremen.de/spl/pub/Website/Downloads/Rules2015.pdf (visited on 04/07/2015).

[11] RoboCup Technical Committee. *RoboCup Standard Platform League (Nao) Rule Book*. 2009. URL: http://www.informatik.uni-bremen.de/spl/pub/Website/Downloads/Rules2009.pdf (visited on 04/06/2015).

[12] The Robocup Federation. *Objective*. URL: http://www.robocup.org/about-robocup/objective/ (visited on 04/07/2015).

[13] Sean Harris. *Localisation - Multi-Modal Extended Kalman Filter*. 2014. URL: https://github.com/UNSWComputing/rUNSWift-2014-release/wiki/Localisation (visited on 05/10/2015).

[14] Bernhard Hengst. *rUNSWift Walk2014 Report Robocup Standard Platform League*. 2014. URL: http://cgi.cse.unsw.edu.au/~robocup/2014ChampionTeamPaperReports/20140930-Bernhard.Hengst-Walk2014Report.pdf (visited on 04/25/2015).

[15] Kristian Hunt. *HUMANOIDROBOTI ALDEBARAN NAO JALGPALLITARKVARA KITUMISLOOGIKA ARENDAMINE*. 2014. URL: http://www.tuit.ut.ee/sites/default/files/tuit/arvutitehnika-thesis-bsc-2014-hunt-kristian-text-20140529.pdf (visited on 05/10/2015).

[16] *Inertial Unit*. URL: http://doc.aldebaran.com/1-14/family/robots/inertial_robot.html (visited on 04/12/2015).

[17] *Motherboard*. URL: http://doc.aldebaran.com/1-14/family/robots/motherboard_robot.html (visited on 04/12/2015).

[18] *NAO*. URL: https://www.aldebaran.com/en/humanoid-robot/nao-robot (visited on 04/10/2015).

[19] *Nao technical overview*. URL: http://doc.aldebaran.com/1-14/family/robots/index_robots.html (visited on 04/10/2015).

[20] *NAOqi framework*. URL: http://doc.aldebaran.com/1-14/dev/naoqi/index.html (visited on 04/12/2015).

[21] *OpenNAO - NAO OS*. URL: http://doc.aldebaran.com/1-14/dev/tools/opennao.html (visited on 04/12/2015).

[22] *Robot version and Body Type*. URL: http://doc.aldebaran.com/1-14/family/body_type.html (visited on 04/12/2015).

[23] Ritwik Roy. *Behaviour*. 2015. URL: https://github.com/UNSWComputing/rUNSWift-2014-release/wiki/Behaviour (visited on 02/15/2015).

[24] *Sony Quadruped Robot Football League Rule Book*. 1998. URL: http://www.informatik.uni-bremen.de/spl/pub/Website/History/Rules1998.pdf (visited on 04/07/2015).

[25] *Standard Platform League History*. URL: http://www.informatik.uni-bremen.de/spl/bin/view/Website/History (visited on 04/07/2015).

[26] *Summary of Major Rule Changes for 2015*. URL: http://www.informatik.uni-bremen.de/spl/bin/view/Website/MajorRule2015 (visited on 04/07/2015).

[27] *Video Camera*. URL: http://doc.aldebaran.com/1-14/family/robots/video_robot.html (visited on 04/12/2015).

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina, Rainer Keerdo

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose

   "Developing the behaviours for use in RoboCup 2015 based on rUNSWift architecture."

   mille juhendaja on Gholamreza Anbarjafari

   (a) reprodutseerimiseks säilitamise ja üldsusele kättesaadavaks tegemise eesmärgil, sealhulgas digitaalarhiivi DSpace-is lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;

   (b) üldsusele kättesaadavaks tegemiseks Tartu Ülikooli veebikeskkonna kaudu, sealhulgas digitaalarhiivi DSpace'i kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile;

3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tartus, **22.05.2015**