



THE UNIVERSITY OF  
NEW SOUTH WALES

SCHOOL OF  
COMPUTER SCIENCE AND ENGINEERING

**COMP3902: Special Project B**

**Robocup 2008**

**Exploration of Nao and its Locomotive Abilities**

**Aaron Tay (3160660)**

2008

## **Abstract**

Does the Nao, the new robot adopted for the Standard Platform League (SPL) in Robocup, have what it takes to live up to the success of the AIBO dogs? This report documents the development of the locomotive functions of the Nao as per the 2008 rUNSWift team architecture. After six months of testing and development, the Nao has demonstrated some promising results. Our results show the Nao is capable of basic manoeuvrability skills such as walking and turning in sequence.

## **Table of Contents**

Introduction	4
rUNSWift Architecture and Locomotion Module	5
The Hardware behind the Nao	7
Early Works of Nao	9
Nao's the time for NaoQi	12
Future Work	16
Conclusion	17
References	18
Appendices	19

## **Introduction**

Since its initial launch, Robocup has attracted academics, researchers, and robot enthusiast from around the world. In 2008, the Standard Platform League (SPL), traditionally known as the four-legged league with the Sony AIBO dogs, saw the transition from quadrupedal (four-legged) to bipedal (two-legged) humanoid robots. The successor for the SPL, developed by French robotics company, Aldebaran Robotics, is the 'NAO'.

Although the announcement of the new robot is simple, the transition for developers was far from it. With change comes great challenges and the new platform meant some key components from the previous seven years worth of work in the four-legged league became redundant. Locomotive and behavioural skills suffered the most due to mechanical and structural differences between the two platforms. Furthermore, as behavioural skills were so finely tuned to suit the AIBOs and relied on lower level components such as locomotion, it only served as a reference for the new platform. Finally, the field was changed, removing localisation beacons leaving only field lines and two goal posts; similar to a real soccer field.

This report documents the various events from early March up to and including the 2008 Robocup competition in Suzhou, China. Section 2 contains an overview of rUNSWift's 2008 architecture and the locomotive module. We will then briefly discuss in Section 3, the hardware specifications of the Nao and any limitations it has. Section 4 will highlight early ventures and results gained from the transition between simulation and the real world. Section 5 will outline NaoQi and how rUNSWift adjusted its system to accommodate for its introduction. Finally, Section 6 offers some suggestions for future work and offers some general feedback from the 2008 experience.

## 1. rUNSWift Architecture and the Locomotion Module

Largely due to its success in previous years, the Nao system architecture was initially based on the system employed by the AIBOs. This involved a multi-layered approach to modularising key components of the robot. These include Vision, Localisation, Locomotion and Behaviour. The former three are all low level modules which communicate directly with hardware and sensors compared to Behaviour which communicates with the lower level modules in order to achieve its purpose. A break up of this can be seen in Figure 1.0.



Figure 1.0: rUNSWift Architecture (Collien, D., Huynh, G.)

The Locomotion module was originally responsible for sensing, controlling and processing joint commands through the communication of a hardware interface library, ALMotion. Three basic moves were to be developed for the locomotion module: straight walk, in place turn and side step. These basic movements were sufficient to navigate around the field, thus allowing the Naos to play a game of soccer.

Throughout the life cycle of the project, the architecture has undergone some changes since its initial proposal. These were mainly to accommodate for the introduction of NaoQi (middleware software framework which allows communication between software and hardware devices), which came supplied with an open loop walk. An open loop walk means the walk being execute is simply following a set of commands which have been generated offline or pre-processed. Unlike closed-loop walks, they do not adjust or accommodate for environmental factors such as external forces. As such, a robot executing an open loop walk is prone to instability issues and could tumble.

Upon reception of our first robot, it was realised that if the robot fell down from a standing height, severe damage could be sustained to its hardware due to the poor quality of the plastic. In order to prevent or reduce the number of falls, a governing layer was proposed which was responsible for ‘closing the loop’ in the open loop walk. The main idea of this governing layer (later termed ‘walk governor’, see Figure 1.0) was that it would analyse data from the Inertial Measurement Unit (IMU) and Force Sensitive Resistors (FSR) and attempt to bring the robot to a ‘safe’ position if the sensor readings return values suggesting signs of instability (Li, Takanishi, Kato, 1991). The ‘safe’ position was a crouching stance which reduced the centre of mass, thereby restabilising the robot.

Data gathered from the IMU and FSR were difficult to interpret due to the high level of noise present in both devices. A high pass filter was used to counter this however there were still outliers which could not be interpreted with certainty. Due to the inconsistent and unreliable data gathered from the two devices, the governing layer’s main purpose was unable to be fulfilled.

The governing layer thus, eventually became the sole wrapper for the NaoQi walk actions with the added ability to queue walk sequences. This was necessary as NaoQi, executes walk commands instantly regardless of whether a previous walk command was issued. As such, we experienced various moments of twitching due to conflicting joint commands. Walk commands would be added and executed once a previous command sequence has been completed or terminated.

## 2. The Hardware behind the Nao

Although thoroughly detailed in the NaoWare Documentation, there are some areas of the Nao which need to be addressed. In relation to this report, these are the joints and lower torso of the robot itself.

Weighing in at just over 4.5kg (with its battery) the Nao is a fairly heavy robot compared to the AIBOs which weigh around 1.5kg. In addition, the distribution of weight on the Nao is almost divided evenly between its upper and lower body, with the majority of the upper body weight in the main torso body. This leads to stability issues as the centre of mass is placed higher above the ground, thus making it harder to balance. Since the Nao is used in the SPL, it is up to the developers to take this into account when implementing balancing algorithms such as an inverted pendulum or dynamic ZMP balancing. Both of which will be discussed later.

Overall, the Nao has 25 degrees of freedom (see Appendix 1). A degree of freedom is a place where the robot is able to change the position of a joint. There are 11 degrees of freedom in the lower torso, including the pelvis. Each leg has five degrees of freedom, two in the hip (pitch and roll), one in the knee (pitch) and two in the ankles (pitch and roll). Pitch allows a joint to be moved parallel to the sagittal plane, where as roll is parallel to the coronal plane. This is shown in Figure 2.0.

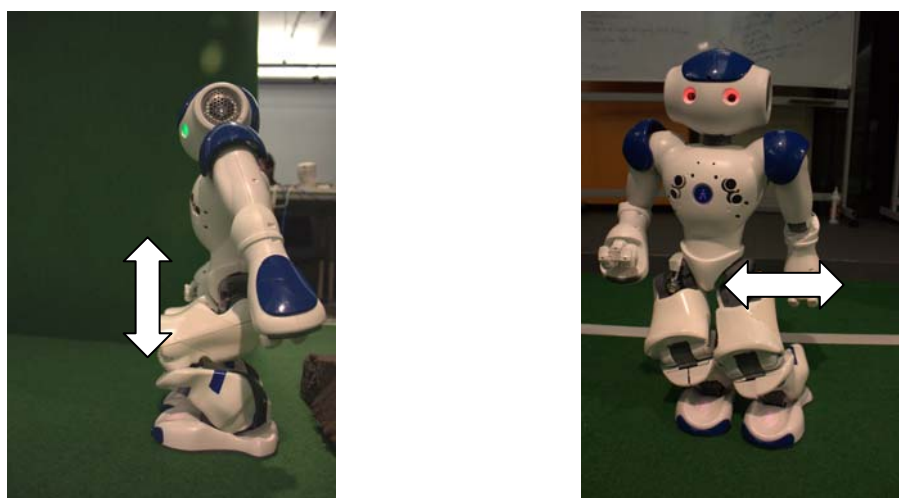


Figure 2.0: Pitch (left) and Roll (right) on the Nao

The motor responsible for the pelvis has a unique design which is not seen in conventionally robots. Unlike traditional robots which have a horizontal plane hip motor

and another vertical plane leg motor the Nao uses an inclined axis rotatory which allows its legs to move in a similar fashion to humans. The differences are shown in Figure 2.1. Although this design reduces the number of motors required for the hip (effectively one as opposed to three), there are performance issues and these affect the walks. However the walk parameters are able to be adjusted to accommodate for this and is discussed in Section 5.

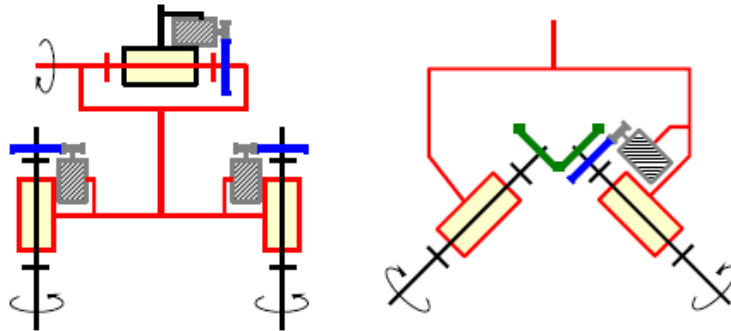


Fig. 3. Left-hand side: classical set of three rotary joints, one horizontal axis rotary joint at the waist and two vertical axis rotary joints for the legs. Right-hand side: coupled inclined axis rotary joints for the NAO pelvis.

Figure 2.1: Differences in hip motors on robots. (David Gouaillier, Vincent Hugel, Pierre Blazeovic, 2008)

Although briefly discussed, the Nao is equipped with an array of sensors including the Inertial Measurement Unit (IMU), Force Sensitive Resistors (FSR), ultrasound sonar, microphones and of course, a video camera. The location of these can be seen in Appendix 2. Unlike the IMU and FSR, the ultrasound is fairly accurate within a specific range (0.15m to 1.75m). This makes it useful for determining reliably whether there is an object in front of the robot or not. However, despite various attempts at its usage, the values retrieved by the ultrasound are always two cycles behind, making it slightly unreliable when quick decisions are required. This is due to the fact that to test if an object is in front, an ultrasound signal must be broadcast in one cycle. The receiver is then queried in the next cycle for the result.



### 3. Early works of Nao

Before the introduction of NaoQi, the interface between our software and the robot's motors was via a library provided by Aldebaran. This library was a lightweight API which gave us near full control on the robot's hardware. Over the 07/08 summer, the 'Taste of Summer Research' students, Steven Wong and Thurston Dang, worked on setting up the system in a robotics simulation program known as Webots (by Cybernetics). Work was done using this simulator until the actual robots arrived (10 weeks before the competition).

The first task we decided to tackle was a side stepping motion. There were many reasons for going for this before a walk:

- We only have to work with two dimensions in the Coronal plane. Since the robot does not move forwards we do not need to consider the Sagittal plane and thus dimensional complexity is reduced allowing us to use 2D trigonometry.
- Motions can be mirrored easily in both directions as well as reversed if properly implemented to give us a canned (open loop) side step in either direction.
- Useful skill for players to have when lining up to the ball. If a robot is near a ball, it will have to align itself around it such that it is facing the correct goal. A sidestepping technique is therefore necessary to circle it.

Two versions of the side step were produced (see Appendix 3). In the first approach, we assumed the robot's knees could not bend. This reduced the number of joints we had to modify, thereby reducing our complexity. The aim of this approach was to get the robot to move in one side ways direction without falling. Results showed that a side step motion is possible but could cause damage to the robot due to the falling nature of the side step. The second approach was much more successful in terms of speed and stability. In this approach we allowed the knee joints to be bent. The aim was the same as the first approach, but capable of moving in either direction easily. Results from this approach were more promising as moving in opposite directions was as simple as reversing the set of commands. The falling motion was also eliminated reducing possibly damage to the robot. It is important to note that both approaches resulted in a canned motion (following a set of joint commands with no feedback from sensors).

Another area explored using the simulator was the idea of a Proportional Integral-Derivative Controller (PID controller) in the area of balance control. This was necessary to develop a system which is capable of adjusting for instability and thus correct the balance of the robot while standing and in motion. At first, the robot was left standing and data was extracted from the simulated readings of the IMU and FSRs. Noise was averaged out via a high pass filter to remove outliers and then averages taken which was used as our desired values. The robot was then made to stand again but this time an external force was exerted upon it which was an attempt to push it off balance. The difference (error) in the reading between the desired and after the exerted force was used in Formula 1 to calculate the amount of error which existed. Results showed that the Integral component was overcompensating and thus increased the error, as such, it was removed.

$$\text{output} = K_p(P_{\text{desired}} - P_{\text{error}}) + K_d(D_{\text{desired}} - D_{\text{error}}) + K_I(I_{\text{desired}} - I_{\text{error}})$$

Formula 1: PID formula.  $K_I$  was set to 0 to eliminate it from the equation

P = joint angles. D is the derivative of joint angles over some time step (5 cycles). K is a constant

Using this error, leg joints were adjusted appropriately to reduce this error which usually involved trying to straighten the upper body by rolling or tilting joints in a direction opposite to the force being exerted.

The above approach, if successfully completed, could have been used to treat the robot as an inverted pendulum and create stable walks. The idea behind an inverted pendulum is that the upper body must be kept at a certain position (relative to the ground). The lower body is then responsible for maintaining this, by leg joint angle adjustments. The distance from the desired position and the current position would be the error used in the PID controller, and solving this problem can be likened to pole balancing experiments.

Although the simulator provided us with a means of testing our design decisions, results gains would not translate to the real world. This was due to various reasons:

- There was no friction in the simulator, for example, if the ball was kicked, it would roll and bounce indefinitely.
- The weight distribution of the robot was rather unrealistic given the actual data sheet of the robot's distributions. This led to calculations for the PID controllers being inaccurate and unreliable.

- The robot's motors have an infinite amount of power, and given the appropriate time, could move the robot to any position. This meant that a knee motor could support the entire weight of the robot, when in reality we know this could never happen. A perfect example is the get up routine which works in the simulator, but could not work on an actual robot.

Subsequent versions of the simulator addressed and seemed to solve these problems, however we did not pursue further into the use of the simulation as the amount of time required to readjust our system to work on the simulator and the actual robot would far outweigh its potential benefits, given the tight time constraints.

## 4. Nao's the Time for NaoQi

NaoQi is proprietary software designed by Aldebaran Robotics and “*is the kernel of Nao's intelligence*” as it provides the interface between software and hardware features of the Nao and replaced the previous library interface. It allows computers in a network to either execute programs locally or remotely on the Nao. From a locomotive point of view, NaoQi had many features such as calculating joint angles, their location in 3D space, balancing as well as an open-loop walk. As such, an investigative study was undertaken to explore the limits of these locomotive abilities.

### Aldebaran's Open Loop Walk

Aldebaran was rather generous by providing us with a walk which was relatively easy to tune and optimise. Provided were 4 types of walking actions; walking straight, turning on the spot, side stepping and walking around a circle (arc walk). A combination of these actions sufficed for playing a game of soccer at this year's competition, but we believe for future competitions these will not be enough.

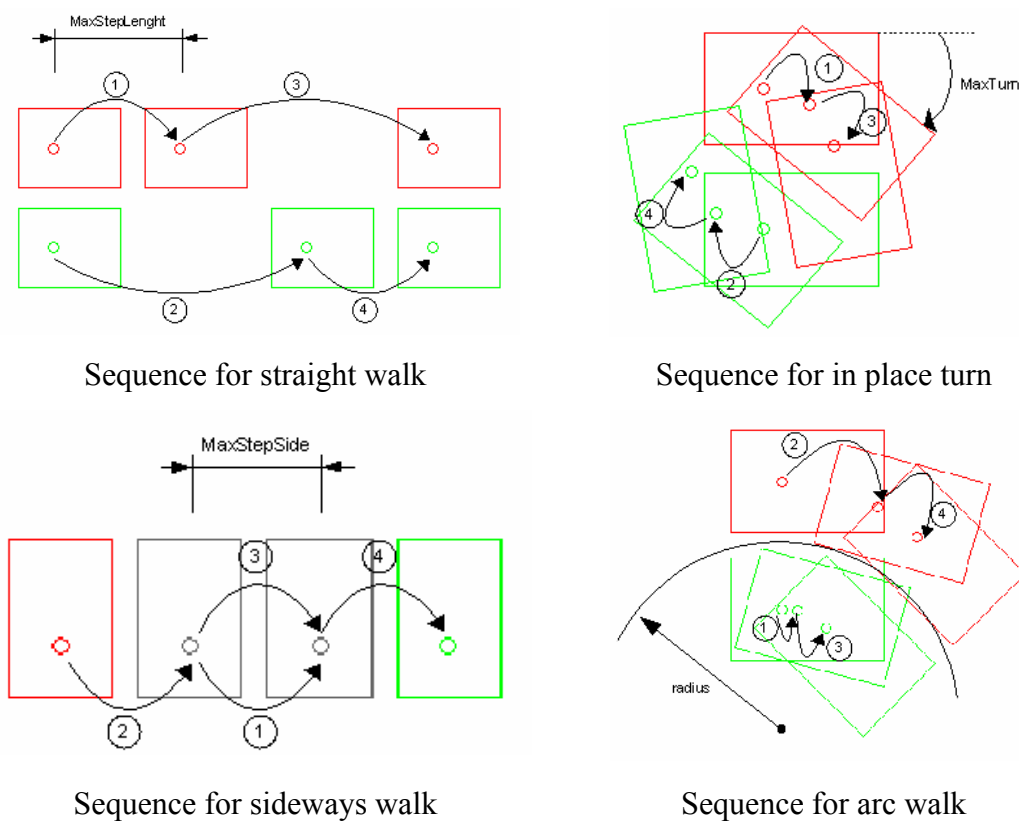


Figure 4.0: NaoWare documentation

Figure 4.0 shows the different sequences of steps each walk executes, as well as some of the information regarding the parameters used. The use of parameters in the walks is rather important as they affect how the walks are executed. For example, a walk with a higher centre of mass will have a larger stride (step) length, but may be more unstable. However, a walk with a lower centre of mass is able to walk faster but take smaller steps because it is more stable. Various tests were carried out and the results of each parameter on each walk can be seen in Appendix 4. It is worth noting that some of the results obtained from the tests were based on intuition more than concrete empirical results.

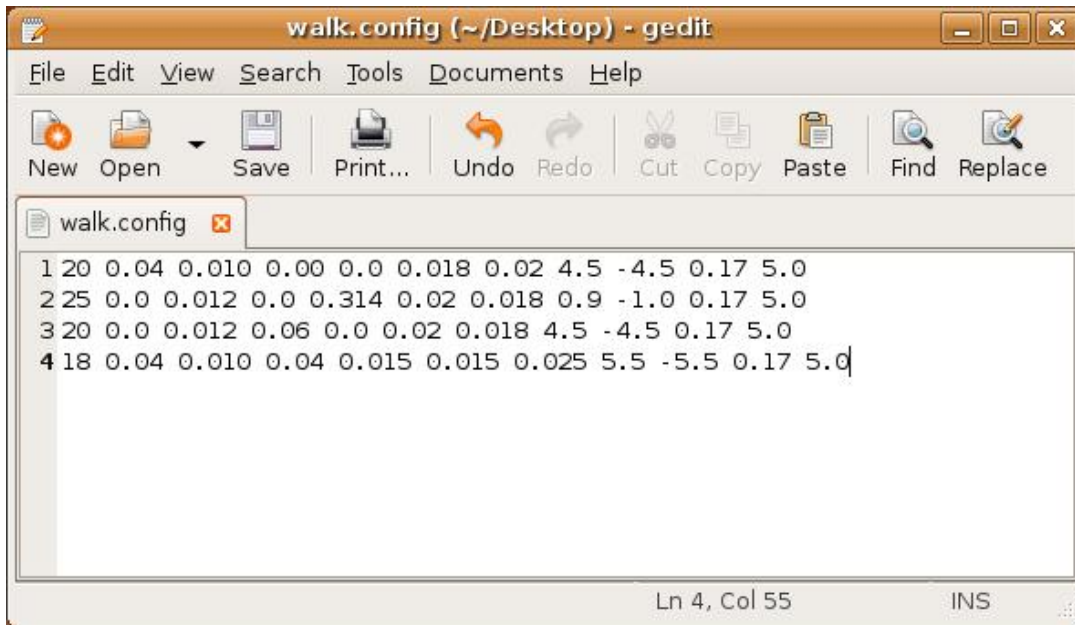
### **Tuning of the walk**

The initial walk provided by Aldebaran was not fast enough to live up to the rUNSWift name. As such optimisation on the parameters was necessary. Although a machine learning or downhill simplex approach would have sufficed for optimisation, time constraints prevented us from automating this process. As such, all tuning was performed manually by Gary Huynh and Aaron Tay.

Initially, walk parameters were hard coded into the code base of the system. This meant that a recompile was required each time a parameter was changed. From a time consumption point of view, this could take up to two minutes per change which, adds up very quickly leading to inefficient use of the robot as behaviour or vision testing needed to be done as well. In order to reduce this time, we developed an external file, 'walk.config', which was placed in the root directory of the Nao. This file contained all the walk parameters in a particular order. When our module was executed, it would read the values from the file and use those for all the walks in that instance. Reloading of a module takes at most five seconds. This reduction in time allowed us to manually tune at a quicker pace. The typical layout of the walk configuration file can be seen in Figure 4.1.

The structure of the file was simple and was based on the order in which parameters are required in the two main configuration functions. The first number refers to the number of cyclesPerStep. Adjusting this affects the speed of the robot's walk. The next six values are parameters for the setWalkConfig() function and are used to adjust the individual stepping features from forward and side length to speed and turn angle. Finally, the last

four values are for the `setWalkExtraConfig()` function which allow us to affect the upper body during the walk including the height from the ground as well as side and forward swaying motions.



```
walk.config (~/Desktop) - gedit
File Edit View Search Tools Documents Help
New Open Save Print... Undo Redo Cut Copy Paste Find Replace
walk.config
1 20 0.04 0.010 0.00 0.0 0.018 0.02 4.5 -4.5 0.17 5.0
2 25 0.0 0.012 0.0 0.314 0.02 0.018 0.9 -1.0 0.17 5.0
3 20 0.0 0.012 0.06 0.0 0.02 0.018 4.5 -4.5 0.17 5.0
4 18 0.04 0.010 0.04 0.015 0.015 0.025 5.5 -5.5 0.17 5.0
Ln 4, Col 55 INS
```

Figure 4.1: Typical layout of walk configuration file.

The approach we used to tune the parameters had a similar resemblance to a hill-climbing algorithm and this method comes naturally to humans. We would change the value of one parameter and see how that affects the walk. If the walk is more stable and faster (to some extent) then this set of parameters would be used in the next iteration. Sometimes the same parameter was changed multiple times to see how various values affect the walk given the other set of parameters as this allowed us to determine the dependencies as specified in Appendix 4.

Once we believed a decent set of parameters was found, tests were performed to compare the differences between our hand tuned parameters and Aldebaran’s supplied parameters. Appendix 5 shows a summary of results from a series of straight time trial tests. The parameters for the relevant walks are also detailed in the test results. Figure 4.2 shows the area on which the tests were conducted. All tests were conducted on our star playing Nao dubbed ‘Owen’. Overall, the results were promising as our tuned walks were about 20% faster than Aldebaran’s walks. It is important to note that due to the hand crafted nature of the robots, there are subtle differences between them which leads to the same set of parameters not necessarily working for all robots. As such, each robot must be tuned individually. Although both walks displayed deviations from a straight line walk, our

walk was more predictable than Aldebaran's. Finally, compared to Aldebaran and other teams such as the Newcastle Numanoids (2008 world champions), we believed a lower centre of mass provided more stability and less deviation from a straight path. When compared side by side, the height of the robot is noticeably different as can be seen in Figure 4.3.



Figure 4.2: Field with Nao. The Nao was made to walk straight on the red line. The width of the field is 4.0m

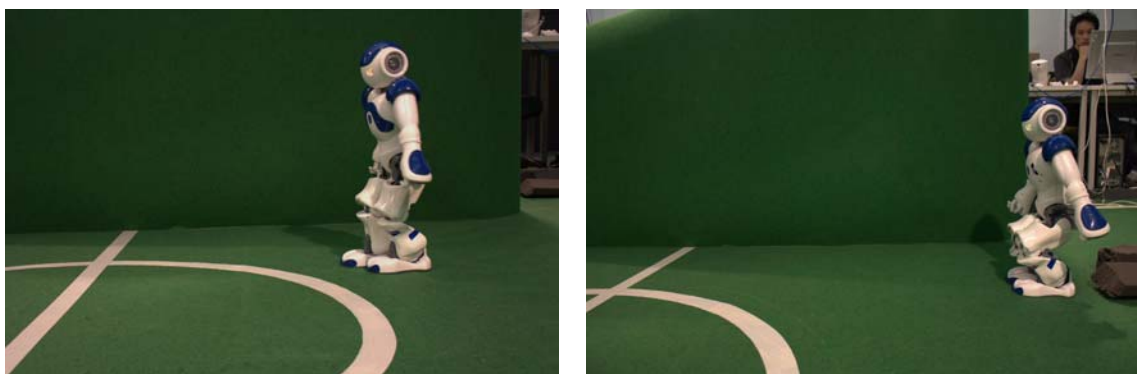


Figure 4.3: Walk with higher centre of mass (left) versus lower centre of mass (right)

## 5. Future work

If time allowed, there are many areas which need exploration. The first and foremost involves the communication with hardware devices directly. The Vision module already communicates directly with the video camera as the original driver has compatibility issues. If the Locomotion module was able to communicate with motors directly, the use of NaoQi would be eliminated and as such we would not be restricted nor have to rely on using it. Direct communication would also allow us to create more flexible walks which are able to be merged or changed dynamically. However, by directly communicating with the motors, there is a higher chance for hardware failures as everything we do will need to be double checked to ensure damage to the motors is kept to a minimum.

In the field of walks, my first proposal and initial idea of a side step would be to develop one which is capable of circling around a point. This approach would have a circle with centre,  $P$  where;  $P_x$  = distance from robot centre,  $P_y = 0$ . The robot would then side step around this point in a circle with radius  $P_x$ . Thus, as  $P_x$  approaches  $\infty$ , the more straight the side step as each step corresponds to some arc segment along the circle. Similarly, as  $P_x$  approaches 0 the more circular the side step becomes. It is also worth noting that for  $P_x < 0$  the same occurs except that the robot should circle the point facing outwards. This approach would enable the Nao to circle the ball, goal or even other robots in a manner similar to the AiBO sGetBehindBall() behaviour in the previous rUNSWift architecture.

Secondly, a walk which is able to be changed midway is necessary if we want to have a flexible soccer player. This could be achieved by following another approach used in the AIBOs. We would have a cache which holds the next set of actions to be carried out. If the robot has at least one foot on the ground and the other in a 'safe' position, then and only then should it be able to stop its current walk action and execute the action stored in the cache. The reason for the need for this safe position is that it allows the robot to make at least one step (thereby movement) between actions as well as prevent it from falling over due to frequent changing of actions.



## 6. Conclusion

Overall we can see that the Nao does have potential to succeed the AIBOs to the throne as the new platform for the SPL. However, there are some issues which need to be addressed in order for Nao to become widely accepted. These are mainly to do with design and hardware issues of the Nao.

We have seen that the weight distribution of the Nao is rather poor and does not lend itself easily with regard to stability. If some of the weight from the upper body was moved to the lower body, this might solve some of the stated problem, however even stronger motors would be required to handle a heavier load. The twist hip motor is another area which needs to be addressed.

Our robots, 'Owen' in particular, have shown that the Naos are capable of decent manoeuvres in terms of speed and stability. As such, there is potential for the competition to mature (speed-wise) into matches of similar standard to the humanoid league.

From a developer's perspective, NaoQi, although provided some useful features, is not practical for Robocup soccer. This is due to its inflexibility of the walk which is provided. If the walks could be changed or even merged in sequence with each other we might have adopted and continued to use the system. But because of the static nature of the sequencing of walks, it seems that parts of NaoQi, in particular the walking motions, are on their way to the scrap yard.

Once the above issues have been addressed in future revisions, only then can the Nao have hope in succeeding the AIBO, as the next generation platform for the Standard Platform League.

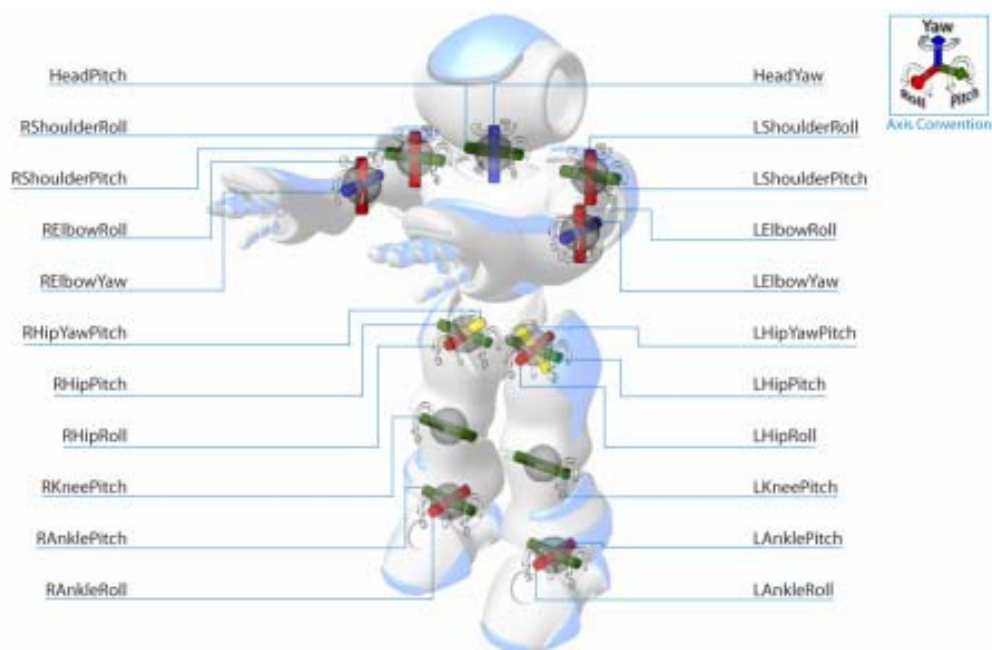
## Bibliography (background reading)

- Jacky Baltes and Sara McGrath and John Anderson, 2004, “*Active Balancing Using Gyroscopes for a Small, Humanoid Robot*”, 2nd International Conference on Autonomous Robots and Agents, December 13-15, pg 470-475
- Sven Behnke, Jörg Stücker, Michael Schreiber, Hannes Schulz, Martin Böhnert, and Konrad Meier, 2007, “*Hierarchical Reactive Control for a Team of Humanoid Soccer Robots*”, Computer Science Institute University of Freiburg, Germany, pg 635-642
- Stefano Carpin and Enrico Pagello, 2006, “*The challenge of motion planning for humanoid robots playing soccer*”, Proceedings of the Workshop on Humanoid Soccer Robots of the 2006 IEEE-RAS International Conference on Humanoid Robots
- David Gouaillier, Vincent Hugel, Pierre Blazevic Chris Kilner, Jérôme Monceaux, Pascal Lafourcade, Brice Marnier, Julien Serre, Bruno Maisonnier, 2008, “*The NAO Humanoid: a combination of performance and affordability*”, arXiv.org (arXiv:0807.3223v1 [cs.RO]), Cornell University Library
- Qinghua Li, Atsuo Takanishi and Ichiro Kato, 1991, “*A Biped Walking Robot Having A ZMP Measurement System Using Universal Force-Moment Sensors*”, IEEE
- Elliot Nicholls, 1998, “*Bipedal Dynamic Walking in Robotics*”, The University of Western Australia, Department of Electrical and Electronic Engineering, PhD Thesis
- Yu Okumura, Tetsuo Tawara, Ken Endo, Takayuki Furuta and Masaharu Shimizu, 2003, “*Realtime ZMP Compensation for Biped Walking Robot using Adaptive Inertia Force Control*”, IEEE October 2003
- Tak Fai Yik, 2007, “*Locomotion of Bipedal Humanoid Robots: Planning and Learning to Walk*”, The University of New South Wales, School of Computer Science and Engineering, PhD Thesis
- Special thanks to everyone on the 2008 rUNSWift team

- **Appendix 1**

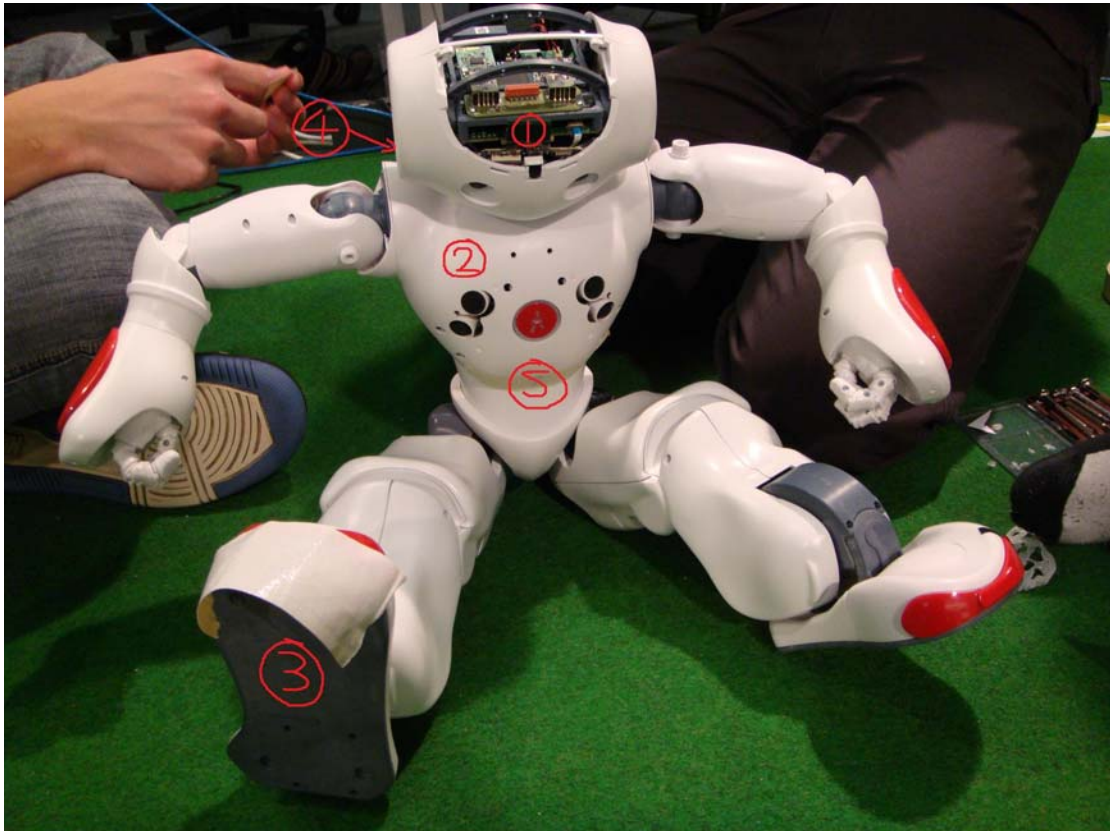
CHAIN NAME	JOINT NAME	MOTION	RANGE (degrees)
Head	HeadYaw	Head joint twist (Z)	-120 to 120
	HeadPitch	Head joint front & back (Y)	-45 to 45
LArm	LShoulderPitch	Left shoulder joint front & back (Y)	-120 to 120
	LShoulderRoll	Left shoulder joint right & left (Z)	0 to 95
	LElbowYaw	Left shoulder joint twist (X)	-120 to 120
	LElbowRoll	Left elbow joint (Z)	-90 to 0
LLeg	LHipYawPitch*	Left hip joint twist (Y-Z 45°)	-90 to 0
	LHipRoll	Left hip joint right & left (X)	-25 to 45
	LHipPitch	Left hip joint front and back (Y)	-100 to 25
	LKneePitch	Left knee joint (Y)	0 to 130
	LAnklePitch	Left ankle joint front & back (Y)	-75 to 45
	LAnkleRoll	Left ankle joint right & left (X)	-45 to 25
RLeg	RHipYawPitch*	Right hip joint twist (Y-Z 45°)	-90 to 0
	RHipRoll	Right hip joint right & left (X)	-45 to 25
	RHipPitch	Right hip joint front and back (Y)	-100 to 25
	RKneePitch	Right knee joint (Y)	0 to 130
	RAnklePitch	Right ankle joint front & back (Y)	-75 to 45
	RAnkleRoll	Right ankle right & left (X)	-25 to 45
RArm	RShoulderPitch	Right shoulder joint front & back (Y)	-120 to 120
	RShoulderRoll	Right shoulder joint right & left (Z)	-95 to 0
	RElbowYaw	Right shoulder joint twist (X)	-120 to 120
	RElbowRoll	Right elbow joint (Z)	0 to 90

Joint Limits as per NaoWare Documentation



Location of joints on the Nao as per NaoWare Documentation

## Appendix 2



Location of various key parts on the Nao.

- (1) Video camera
- (2) Ultrasound sensor (black dots)
- (3) Force Sensitive Resistors (inside feet)
- (4) Microphone
- (5) Inertial Unit (inside body)

## Appendix 3a

### Task:

Develop a canned motion for the robot which moves sideways in one direction whilst maintaining balance

### Terminology:

Inwards: towards the centre of the robot

Outwards: leaving the robot's body

### Movement Sequence:

1. Roll left ankle and hips outwards until the right leg is raised off the ground. This effectively shifts the centre of mass onto the left leg.
2. Roll right hip outwards until the right leg is almost parallel to the ground. Roll the right ankle at the same rate inwards so it remains parallel to the ground.
3. Roll left ankle and hips inwards until the right leg makes contact with the ground.
4. Roll right hip and ankles back to zero positions
5. Roll left hip and ankles back to zero positions

### Results:

- Robot was able to successfully move in one direction whilst maintaining balance. The resulting actions were neither fluid nor natural (from a human perspective).
- The side step motion abused the simulator's lack of friction and dragged the foot across the ground when returning to the neutral position.
- When the mobile foot came in contact with the ground the robot was 'falling' onto it. If such a motion were carried out under real conditions the leg could sustain damage or lose balance due to the falling motion.

### Discussion:

- When the side step motion was executed in reverse order, the robot would fall over. However, changing the joint angle to suit the opposite direction would solve this problem.

### Findings:

Simulator has very poor friction simulation.

In order to keep the upper body upright, the ankle roll and hip rolls must be kept equal to each other.

## **Appendix 3 cont...**

### **Task:**

Develop a canned motion for the robot which moves sideways in both directions whilst maintaining balance. Both directions refer to, using the same set of commands in reverse in order to achieve the same motion just in the opposite direction.

### **Movement Sequence:**

1. Roll hips and ankles to the left until the centre of the upper body is over the left foot.
2. Bending the left knee, move the right leg to the outwards (to the right) until it touches the ground, keeping the right ankle parallel to the ground at all times.
3. Set the joints of the right leg to the joints of the left leg and vice versa. This effectively shifts the centre of mass to the other leg whilst keeping the sequence of actions symmetrical.
4. Do the reverse of 1 but with the legs swapped.

### **Results:**

- The robot was able to successfully move sideways in both directions. Use of the knee joint enabled more fluid movement which resulted in a more natural motion.
- The speed of the transition between support legs was slightly fast and this may have adverse effects on the actual robot.
- There was a slight angle in the sideways motion to the right. The deviation was only noticeable if the robot was made to side step the entire length of the field. Deviation was around 10 degrees forwards.
- By reversing the order of commands, the robot was able to return to its initial position suggesting that the sideways motion is perfectly symmetrical.

### **Findings:**

- In order to keep the ankles parallel to the ground, hip roll and ankle roll must be equal. Furthermore, hip pitch + ankle pitch must equal knee pitch.
- Keeping the ankle parallel to the ground at all times meant that as soon as it made contact with the ground it would be flat and therefore more stable.

## Appendix 4 – Parameter Effects on Various Walks

Parameter	Effect on Straight Walk
distance	Distance to walk. Positive is forwards. Negative is backwards
stepsPerCycle	How long each step takes. 18 is the best minimum value found. Use 30 if walk is unstable, this will slow down the motions.
maxStepLength	How far forward the robot moves with each step.
maxStepHeight	How high each step can be. If stepLength is further, the step height must be high enough to allow for the step. Low step height reduces angle deviation.
maxStepSide	<i>No effect, leave as 0.0</i>
maxStepTurn	<i>No effect, leave as 0.0</i>
zmpOffsetX	How far forwards the robot is allowed to lean. Large values lead to instability
zmpOffsetY	How far to the sides is the robot allowed to lean. Large values lead to instability
LHipRollBacklash	<i>No noticeable effect, use default 4.5</i>
RHipRollBacklash	<i>No noticeable effect, use default -4.5</i>
hipHeight	Raises/lowers centre of gravity. Lower height leads to more stable walks. High heights allow for larger forward stepLengths.
torsoYOrientation	Affects lean on X-axis (contrary to documentation). Large values can lead to instability.

Parameter	Effect on in place Turn
angle	Angle to turn the robot. Angle is in RADIANS. Angles based on the trigonometric angle circle (i.e. positive is anti-clockwise)
stepsPerCycle	How long each step takes. 22 is the best minimum. Lower values lead to inaccurate turning
maxStepLength	<i>No effect, leave as 0.0</i>
maxStepHeight	How high each step can be. Should be proportional to stepTurn
maxStepSide	<i>No effect, leave as 0.0</i>
maxStepTurn	How many degrees each step changes the orientation of the robot. RADIANS. Use numbers which are proportional to $\pi$
zmpOffsetX	Affects forward lean. Large values lead to instability. Keep below 0.25
zmpOffsetY	Affects sideward lean. Large values may lead to faster turn. Keep below 0.25
LHipRollBacklash	Affects leftward (anti-clockwise) turning. Larger values give bigger turns. Reduces rightward turns. Usually less than rHipRollBacklash due to motor differences.
RHipRollBacklash	As above except for rightward turns. Reduces leftward turns.
hipHeight	Same as straight walk except for larger stepTurn
torsoYOrientation	<i>No noticeable effect, use default 5.0</i>

Parameter	Effect on Sideways Walk
distance	Distance to walk. Positive is to the right. Negative to the left
stepsPerCycle	How long each step takes. 20 is the best minimum.
maxStepLength	<i>No effect, leave as 0.0</i>
maxStepHeight	Same as straight walk, except for stepSide. Lower step heights reduce angle deviation
maxStepSide	How far to the side the robot moves with each step.
maxStepTurn	<i>No effect, leave as 0.0</i>
zmpOffsetX	<i>No noticeable effect, use default 0.20</i>
zmpOffsetY	Affects sideward lean. Large values may lead to faster sidesteps, but more instability.
LHipRollBacklash	Affects RIGHTward momentum. Higher values give faster movements to the right.
RHipRollBacklash	As above except for LEFTward momentum.
hipHeight	Same as straight walk except for larger stepSide
torsoYOrientation	<i>No noticeable effect, use default 5.0</i>

## Appendix 5 – Straight Walk Time Trial Results

### Results from various walks

Walk Type	Distance Travelled (m)							
	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0
rUNSWift Tuned	5.7s 0°	10.7s 5°	15.4s 10°	19.3s 15°	24.3s 21°	29.7s 24°	36.1s 30°	40.7s 33°
Aldebaran v1.0	7.9s 4°	(1)	(1)	(1)	(1)	(1)	(1)	(1)
Aldebaran v1.0 (slower)	9.0s 5°	17.1s 9°	24.0s 20°	31.5s 5°	40.0s 10°	47.0s -10°	54.3s - (2)	(1)
Aldebaran v2.0	7.1s 2°	13.5s -6°	19.1s 15°	25.9s 5°	31.5s 2.3°	37.7s 30°	43.3s 15°	(1)

#### Notes:

Results based on an average of 10 trials per successful walk.

Aldebaran v1.0 (slower) uses the same parameters except the cyclesPerStep parameter was set to 30 as opposed to the original 25.

(1) Robot fell almost immediately and thus results could not be obtained

(2) Robot fell but after walking a certain distance.

Results in the table represent how long the robot took to travel the distance and how much from the centre straight line the walk deviate from. Times are in seconds, Deviations are in degrees. Positive deviations means the robot deviated towards the right, and negative towards the left.

### Walk parameters - ordered as per section 4

#### rUNSWift tuned parameters ‘Owen Config’

20	0.04	0.010	0.0	0.0	0.018	0.02	4.5	-4.5	0.17	5.0
25	0.0	0.012	0.0	0.314	0.02	0.018	0.9	-1.0	0.17	5.0
20	0.0	0.012	0.06	0.0	0.02	0.018	4.5	-4.5	0.17	5.0
18	0.04	0.010	0.04	0.015	0.015	0.025	5.5	-5.5	0.17	5.0

#### Aldebaran Version 1.0 Walk (Initial release)

25	0.04	0.015	0.04	0.3	0.015	0.025	4.5	-4.5	0.19	5.0
25	0.00	0.012	0.00	0.314	0.02	0.01	4.5	-4.5	0.19	5.0
25	0.00	0.015	0.05	0.0	0.02	0.018	4.5	-4.5	0.19	5.0
25	0.04	0.015	0.04	0.015	0.015	0.025	4.5	-4.5	0.19	5.0

#### Aldebaran Version 2.0 Walk (Improved)

25	0.04	0.015	0.04	0.3	0.015	0.025	4.5	-4.5	0.19	5.0
25	0.00	0.012	0.00	0.314	0.02	0.01	4.5	-4.5	0.19	5.0
25	0.00	0.015	0.05	0.0	0.02	0.018	4.5	-4.5	0.19	5.0
25	0.04	0.015	0.04	0.015	0.015	0.025	4.5	-4.5	0.19	5.0