The 2008 NUManoids Team Report

Naomi Henderson Steven P. Nicklin Aaron Wong Jason Kulk
Stephan K. Chalup Robert King Richard H. Middleton
Shekman Tang Alexander Buckley

November 1, 2008

NUManoids

School of Electrical Engineering & Computer Science
The University of Newcastle, Callaghan 2308, Australia
http://www.robots.newcastle.edu.au
&
National University of Ireland, Maynooth
Maynooth, Co Kildare, Ireland

Contents

1 Introduction

1.1 Overview

The NUbots from the University of Newcastle, Australia have had a strong record of success in the RoboCup Four Legged League since first entering in 2002. The Nubots achieved 3rd place in 2002 (Fukuoka), and again in 2003 (Padua) and 2004 (Lisbon). In 2005 (Osaka) with a complete redevelopment of the code, the NUbots came 2nd in a heartbreaking penalty shoot out against the German Team. With a strong 2005 base code, 2006 development focused on debugging and fine tuning. The result was a nail biting final against rUNSWift in Bremen where the NUbots won 7-3. Code development plateaued in 2007 and we lost the final in Atlanta against the Northern Bites, a relatively new team who was heavily influenced by our methods.

2008 was a year of change. The transition from the Sony ERS-7 robot to the Aldebaran Nao meant a total rewrite of the software system. In addition, two of our main team members left. Michael Quinlan was 'poached' by UT Austin and Rick Middleton moved to the National University of Ireland, Maynooth (NUIM). With a limited number of places available in the new Nao League we decided to collaborate with Rick and NUIM to form a joint team, the 'NUManoids'.

2008 was also a year of challenges. University payment issues, splitting the robot package and repair delays meant even that we had even less time to work on the robot than originally planned. In the end we had approximately 3 weeks 'hands on' time with a robot. International collaboration proved quite difficult in the short amount of development time.

The NUManoids preparation strategy was based on priorities. Our first priority was to chase the ball. This required the robot to see the ball, track the ball and walk towards the ball.

Once this was achieved our next aim was to kick towards a goal. Unfortunately we didn't get to this stage. Time restrictions and international collaboration meant that the integration of localisation was delayed until we arrived at the venue. We did not get a chance to properly test localisation hence we could not write behaviour that depended on it. However, by the final we had implemented behaviour that attempted to not kick towards our own goal.

A heightened stance and reduced stiffness of the joints was our advantage in game performance. Our walk style also doubled as a kick. Our walk comprised of

short sharp steps so there was no need to waste time lining up the ball for a kick. Our behaviour concept was to gain control of the ball and at least do something with it as quickly as possible.

We made it to the 2008 final of the 2-Legged Standard Platform League based on luck. We played GTCMUnited and the game came down to a penalty shoot out after a 0-0 draw. The penalty shoot out rules were; the ball is positioned 1.5m away from the the goal and the robot 50 cm behind the ball. There is no goal keeper and the quickest goal wins. If no goal is scored in 3 minutes the team who had the ball closest to the goal wins. GTCMUnited went first. After many attempts to kick the ball a goal was scored in 2min 30sec.

Then it was our turn. The robot took four steps towards the ball and stopped about 5cm in front of the ball. It panned its head and then continued to walk straight, kicking the ball the full length and into the goal. The goal was scored in 11 seconds. This year our advantage was focusing on the basic things. Our simple behaviour and reliable walk style proved to be all we needed.

1.2 Software Layout

Our 2008 system was based on our Aibo architecture. Our system consists of one control module *Nao* and four functional modules - *Vision, Localisation, Behaviour and Locomotion.*

The flow of information in our system can be seen in Figure 1.

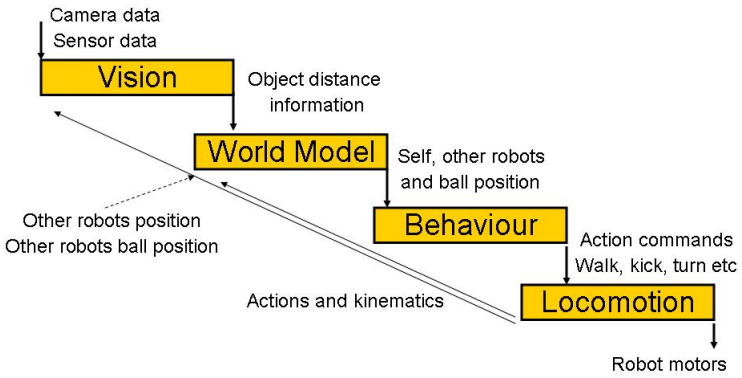

Figure 1: 2008 Software Architecture

2 Vision

The Nao vision system followed the process: *Input image, classify image, form blobs, sort colour blobs, combine soft blobs, find ball, find goals, line detection.* Even though the vision process was based on NUbot code the only section that could be reused was blob formation.

The 2008 vision module was very simple. This section details the main functions of the vision system and the reasons behind them.

2.1 Overview

Webots was used to develop the base vision system. Inputting the rgb image and classifying it using a perfect look-up table (LUT) allowed blob formation and basic object recognition to be tested which then allowed for localisation development to begin.

We then worked on developing the vision debug application *Eye of the NUManoid* (EOTN). What made this application different is the HSI classification method. Transforming the YUV image to HSI values and selecting the classification region in HSI allowed for an intuitive classification method and better separation of hue ranges.

Once we had a robot we were able to select camera settings using *Telepathe* and save image streams. We could then work on object recognition. It was very basic, however localisation was quite basic aswell.

2.2 Image Quality and Camera Settings

One good thing about the transition from the Aibo to the Nao was the improvement in camera quality. While the camera still had its share of problems, such as random swaps of U and V channels, random blackouts and random ghost camera settings, at least it made the separation of colours easier.

2.2.1 Camera settings

The Aibo camera has three camera settings and three values for each setting. The Nao camera has 6 main camerea settings with a large range of values for each setting. This means that the camera settings are very sensitive to lighting changes and not

only do colour tables need to be recalibrated for any change, but also the camera settings. At the RoboCup venue the camera settings and colour tables had to be recalibrated for both the main field and test field.

The robot is mobile with a movable head. This means that images are subject to blurring and lighting flare. Camera settings are chosen to minimise bluring and allow for best colour separation.

2.3 Colour Classification

Colour classification is the top tier in the software system. Any error in classification has a flow on effect to the entire system. Image pixels are mapped to a colour class by referencing a LUT. The environment of the Standard Platform League is colour coded so that objects are recognisable by colour. Rather than classify pixels based on object colour classes alone, additional 'soft' colour classes are used to allow additional colour information to be classified and to reduce the risk of false positive classification.

2.3.1 Soft Colour Classification

Soft colour classification [Quinlan *et al.*, 2005; Quinlan *et al.*, 2006; Henderson 2005]is used to classify regions of colour that belong to an object but is less saturated due to lighting conditions or overlaps with another colour class. This allows the decision about what object the soft colour belongs to, to be delayed until the entire image is procesed.

Using soft colour allows for the classification of 'noisy' shades of colour and allows valueable object size information to be kept. The risk of false positives is reduced by allowing true colour to be classed as the shades of colour that are highly saturated. Object decisions are based on the presence of true colour and then soft colour can be used for additional size information.

There are significant differences in the application of soft colour between the Aibo and Nao camera. The Aibo camera was low to the ground. This meant that objects were subject to heavy shadowing when upclose to an object. It also meant that when the ball was close to the robot it could fill the entire image. The camera quality was significantly worse than the Nao robot and there was significant overlap between objects of close hues. Soft colour was applied to separate the red uniform

and orange ball, the orange ball and yellow goal, the blue goal from the navy blue uniform and the blue goal from the green field.

While the heightened stance and improved camera quality of the Nao has improved colour separation, the change of uniforms has created new overlap. The shiny bright blue uniforms of the Nao has created a significant overlap of colour values with the blue goal. Similarly, the shiny red uniforms become less saturated with the reflection of light and the hue values shift closer to orange.

Soft colour is also applied to less saturated shades of colour. These shades are more likely to occur in background noise. Soft colour is used to classify additional bright and dark shades of red, orange, yellow and blue.

Blobs are formed for each colour class, including soft colours. Once the image is processed soft colour decisions are made based on blob arrangement in Soft Colour Filtering.

2.4 Blob Formation

Grouping of classified colours is used for the transfer of information from an image to object recognition. When forming blobs the colour, size and area information is required. Blobs are formed based on classified colour of the classified image. Undefined, white and field green and the soft colour shadow-black are not included in blob formation. This is due to the fact that size and shape information is not required for objects of these colours. It is also due to the nature of distribution of these colours. They are all abundantly spread throughout images and scattered; forming blobs on images would involve unnecessary processing.

Blob formation checks every pixel of the image when forming blobs, this means we can form blobs as small as 1x1, however, we only use blobs greater than or equal to 3x3 pixels. In the Aibo camera this method allowed us to see the ball the entire length of the field, however, with the elevated position of the Nao camera we can only see a ball about 3/4 of the field away.

2.5 Sort Colour Blobs

Soft colour blobs are filtered and kept if of sufficient size or overlapping with the corresponding true colour.

2.6 Combine Soft Blobs

Overlapping soft colour blobs are merged. Colour blob clusters are then made to prepare information for object recognition. Cluster variables include: exists, correct pixels, colour 2 correct pixels, min x, max x, miny, max y, height, width, area. Yellow blobs, shadow yellow blobs and all yellow blobs are clustered for yellow goal recognition. Blue blobs, shadow blue blobs and all blue blobs are clustered for blue goal recognition and blue robot separation. Red blobs and shadow red blobs are clusterd for red robot detection.

Clustering allows object recognition to have a region of interest. With the use of soft colour and the new goal size and shape most goal images are made of multiple blobs.

2.7 Ball Recognition

Ball recognition was based on previous code. The main difference is that we don't have to deal with the upclose ball situation. Instead we have to be able to see the ball at an elevated position and with additional glare shining off the ball. A confidence system is used to decide the ball. Confidence is weighted based on correct pixel to size ratio, size and circle fit.

2.7.1 Ball Distance, Bearing and Elevation Calculations

The raw values of bearing and elevation are calculated using the centre x and centre y of the blob in the image, while the raw distance is calculated using the width of the blob in pixels. These resulting values are relative to the camera. They are then transformed using the forward kinematics of the robot to give a relative location in terms of the robots reference frame.

2.7.2 Circle Fitting

Least squares circle fitting has been implemented to improve the distances, bearings and elevations of balls that are not fully visible in the image. There are two main parts to the circle fitting procedure. Firstly the selection of the points to be fit, followed by the fitting of a circle to these points. These points are gathered during one of various scan types depending on the positioning of the ball in the image. The

points are found by searching from the outside of the blob inwards until it finds pixels of the same colour of the blob, or in the case of the orange blobs also one of the soft colours close to orange. The directions for which scans have been created are: *Left to Right, Right to Left, Top to Bottom, Bottom to Top,Simultaneous Left to centre & Right to centre and Simultaneous Top to Centre & Bottom to Centre.* These can be seen in Figure 2. The reason the different scanning directions are needed is so that in images in which the view of the ball is cut off by the edge of the image, points are not selected along the edge of the image in turn biasing the circle fit. For more information on the least squares fitting function used see [Seysener 2003; Seysener *et al.* 2004].

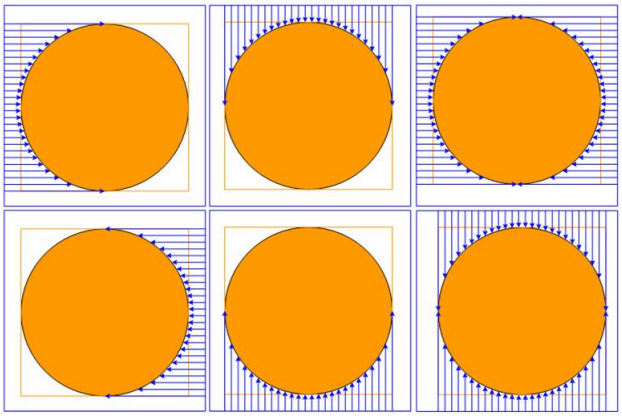

Figure 2: Ball scanning directions, (top left) left to right, (middle left) right to left, (bottom left) left and right to centre, (top right) top to bottom, (middle right) bottom to top, (bottom right) top and bottom to centre.

Once a circle has been fit to these points, the validity of the fit circle is determined. If the diameter of the circle is significantly lower that the width of the blob then it is assumed that the circle fit is not correct.

2.8 Goal Recognition

With the use of soft colour classification and due to obstructions on the field, the goal is often not seen as a single blob, but rather a cluster of blobs. A scan method was used on clusters to detect the gap type and calculate the width of posts. 3

horizontal scan lines were used to detect gap type and post width by measuring the pattern of colour and soft colour compared to non object colours.

Gap types detected are: no gap (unknown post), left (right post), right (left post), middle (both posts, no crossbar), bottom (both posts and crossbar) or undefined.

Three main variables were used to calculate goal confidence. Correct pixel to size ratio, cluster height to width ratio and goal gap detected. The amount of correct pixels required to pass as a goaL object depends on the size and shape of the cluster and also what gap type it has. For instance an image of a single upclose goal post will be detected as having no gap. Then it must pass a height to width ratio check and colour pixels to size ratio check to be recognised as a post. The ratio of correct pixels to size is different for each gap type.

2.8.1 Goal Distance, Bearing and Elevation Calculations

The raw distance of the goal is found by using the width of a post. With the new tall goals and the robot camera position looking down for a ball, the goal height is unreliable. If both goal posts are seen both widths are used to calculate the distance to each post.

The raw bearing and elevation are found using the centre of the post. The method to find the center of the post varies depending on the gap type. As with the ball these raw distances are translated back so they are relative to the hips of the robot. The elevation is not very reliable as the top of the post is often not seen.

2.9 Robot Recognition

Red robots are detected by clusters of red blobs. Blue robots are detected using the clusters of shadow blue blobs if blue goal tests have failed.

2.10 Line and Corner Point Detection

This year the algorithm used in previous years to detect field lines and corners [Quinlan *et al.*, 2006] was ported from the Aibo to the new Nao system. This required rewriting the function that transforms the found corner points in the camera plane to the field plane to suit the new robot kinematics and camera perspective. Otherwise, the algorithm is the same as the previous years for detection. Further

development this year involved uniquely identifying these found corner points for use in Localization.

2.10.1 Line Detection

The detection of field lines is approached in a completely different manner to that of other Objects. When searching for objects, sections of a specific colour are joined together to form a blob. While this works well for most objects, white causes problems due to its abundance in the image and hence is normally ignored. Also the thin nature of lines, and the robot's low Point Of View, makes missing pixels very common within the classified image. Finally the storing of blobs, as a square which bounds the entire object, does not store enough important information about the line.

To overcome these issues the detection of lines is based around the two unique features that field lines have; their long thin length and their green-white-green transitions. Using these two details the image can be sparsely searched, since the search line does not need to find every point of the line, just enough to re-create the line. Also the transition information allows many white pixels to be thrown out early in the detection process, thereby reducing the overall load on the processor. This reliance on points and not the entire line also allows the lines to be partially hidden behind another object without effecting their detection.

With these factors in mind, the image can now be efficiently searched in the following steps;

- The image is searched using a horizontal and vertical search grid. The search is restricted to the area in the image below the horizon line and picks out points of sharp contrast which have green-white then white-green transitions. These points are recorded in an array for later use.

- The found points are then checked against each other to form possible lines. Once candidates are found, more points are checked and added until a line is formed. All checks on the lines at this stage is done based on gradient to allow the fastest line formation.

- The lines are cleaned up to make sure all points actually fit on the line. Lines which have too large a number of points not contained within the final line are

removed.

- Lines are checked against each other to confirm that they are not just segments of larger lines. The allows lines to be formed that have a break in the middle, such as when a robot is on top of the line.

- Corner points are found by extending lines and finding their intersection locations. The intersections are checked to confirm virtual points are not found.

- An attempt is made to uniquely identify corner points from other objects within the image. While this is not always needed, if a unique identification is made the use of the corner point within localisation becomes much more efficient.

2.10.2 Corner Point Detection

The locating of these corner points alone in each frame is not enough to localize with as there are numerous type-L and type-T corners on each side and a complete mirror image of the field lines on each half end of the field. Unique identification of each corner point is required in our system. A rule based approach was used to perform this identification that filters out the possible corner identity based on what properties of the corner can be extracted from the image and in most cases also some vague knowledge of robot location and orientation.

The goal was not to provide data that Localization can rely upon solely but to fine tune the accuracy once approximate localization has been established. However this use of knowledge of location and orientation creates a loop between the Line Detection and Localization system components. This allows for the possibility of each component validating each other's incorrect data to exist. This being the case, the main focus was to write identification rules that are strict enough to eliminate the possibility of false positives. Also, the corner point identification rule written would block (that is, return an unknown corner or T point) unless the Localization confidence first came within 2 standard deviations of the probable orientation.

The identification relies upon the orientation of the seen corners, the combination of these corners and other lines in the image and the number of corners and lines seen. The Line Detection algorithm classifies each corner as either a type L or T and then attributes it with one of four possible orientations, pointing up/down or pointing

left/right. By disassembling the field lines in the image they could be examined for what types of corners and orientations make up what is seen. Following that, this information is then decoded to match what should be seen from the present known location. Avoiding false positives here partially takes care of itself as if the robot location or orientation data is wrong, the corresponding corner orientation classification will not match the rule.

For example, in Figure 3a taken from the simulator, the detection algorithm reports seeing 3 corners: 2 type L's of orientation 2 and 4, 1 type T of orientation 1 and 4 lines. This set of data is almost unique when compared to the other countless possible positions and perspectives from which to view the field lines and can therefore be decoded with brute force.

The only other occurrence is a duplicate on the opposite end of the field as both halves are identical. This the leaves the question, "which end of the field is the robot in?". Knowledge of the robots orientation could also solve this problem of determining which end of the field this scene is taken from. To be robust, the rules written require Localization to be certain of both field XY position and orientations to 2 standard deviations. This value could be increased in this case but for others closer to the mid field line, the probability of error increases as the two mirror images from each end come closer together. The necessity for this prior knowledge is best show pictorially.

In the first frame we have no other information to work with other than the lines themselves. In the second frame, the Goal Detection provides Localisation with enough information to determine which end of the field the robot is in and it is now possible to uniquely identify the corner points.

Another example of an identification rule would the identification of a mid field T corner. Conditional tests for seeing only: 2 field lines, corner count = 1, corner type = T, corner orientation = 3, robot X location is greater than zero, orientation is 135 degrees plus or minus 80 degrees and no goal post is seen, accurately reports the correct corner and filters out all others.

Unique identification rules were written for each corner for a selection of possible perspectives and rigorously tested in the Webots simulator. Possible sources of error discovered in testing were the extension of the penalty box line to join the side field line to create a phantom type T corner, the clipping of a type T corner into a type L by the edge of the camera frame and the center circle appears as a collection of

many corners (for example an octagon). Each of these sources of error were reliably filtered out by the corner combination decoding.

The simulation testing also showed great improvement in Localisation over simply relying upon goal detection alone. Once a goal had been seen and Localisation determined to some approximate area, the line detection would report a unique corner point and fix a very accurate position and orientation and maintain it while looking all around the field.

The identified point on the field is then sent to Localization after performing the following transformation.

$V_{cam} = [focLength, (IMG_{width}/2) - X_{img}, (IMG_{height}/2) - Y_{img}]$
Transformation through neck
V_1=V_{cam}*$M_{camTilt}$
V_2=V_1*M_{camPan}
Transform body tilt
V_3=V_2*$M_{bodyTilt}$
$\alpha = -ChestHeight/V_3[2]$
$distance = \alpha * \sqrt{V_3[0]^2 + V_3[1]^2}$
$bearing = arctan2(V_3[0], V_3[1])$

This inter system approach to the corner identification is shown in Figure 3 b and c. Green components demonstrate an improvement to this approach where one of the Localisation reset triggers becomes an external one based on an added component that monitors for robot relocation by hand by way of accelerometer readings (Teleport Detection if image is BW). In the Robocup competition robots do get picked up and moved, possibly to the other side of the field and turned around by 180 degrees when being penalized. The loop created between the Line Detection and Localisation could slow down the reset as it would take some time for the bad data to be dumped after locating a goal post and dealing with conflicting information. A specialised system component could ensure this does not happen.

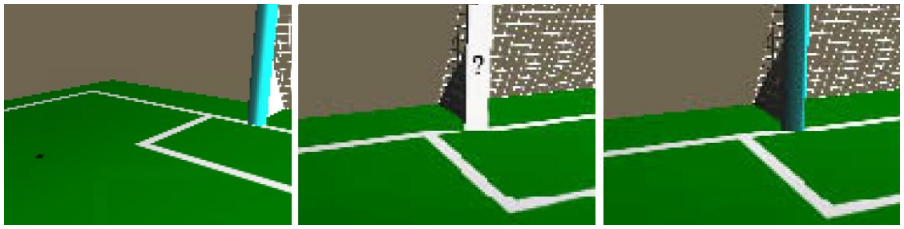

Figure 3: Images of corners for examples.

2.10.3 Code Implementation

Up until this year, Line Detection was a collection of functions within the Vision class itself. To reduce coupling, the Line Detection function and supporting functions were removed from the vision class and placed into a newly created Line Detection class. This Line Detection class is responsible for both line detection and the corner identification. These routines are still part of the vision system and the files can be found in the vision directory. No other NUManoids system component is dependent upon this Line Detection class to in order run and therefore it may be removed from the project build by editing the MakeFile in the root of the code body. Once a Line Detection object has been instantiated, the Line Detection function formlines() is called at the end ProcessFrame() within the class Nao. It requires no data to be passed into it and it outputs its results by directly modifying the elements found in the fieldObjects structure. The list of all possible corner points that can be evaluated can be found in the FieldObject.h file.

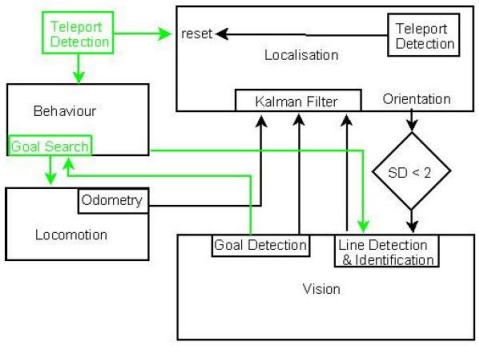

Figure 4: Software flow diagram.

#	Element	units	Description
Xhat[0][0]	x	cm	Robots own x-coordinate
Xhat[1][0]	y	cm	Robots own y-coordinate
Xhat[2][0]	θ	radians	Robots orientation
Xhat[3][0]	xb	cm	Ball x-coordinate
Xhat[4][0]	yb	cm	Ball y-coordinate
Xhat[5][0]	vxb	cm/sec	Ball x velocity
Xhat[6][0]	yb	cm/sec	Ball x-coordinate

3 Localisation

The main software interactions of this module are illustrated in Figure 5. VISION provides vision information (range and bearing) (corrected for head pan, tilt and roll) on observed goal posts and ball objects to LOCWM, beacons are no longer used in this year. LOCOMOTION provides odometry information and LOCWM fuses this data together with past data to form estimates of the robot location, heading, and ball location with uncertainty.

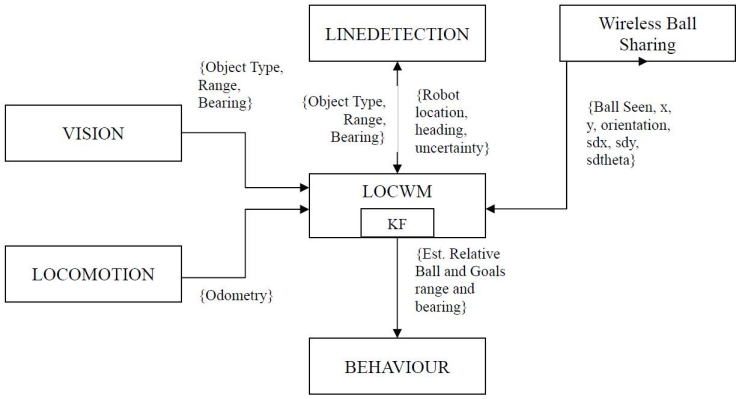

Figure 5: Main software interfaces to Localisation and World Modeling object (LocWM)

The state estimate information is stored in a 7x1 matrix called Xhat. The elements of Xhat are listed in Table 3:

These coordinates are all based on the right handed field coordinate system, with origin at the center of the field, y axis directed up the field towards the blue goal, x axis directed to the right of the y axis, and angles measured anti-clockwise from the x-axis.
The LOCWM code also includes a representation of the uncertainty. This is contained in a 7x7 upper triangular matrix S, where S is a matrix square root of the covariance of the estimates, more specifically, $S \times S^T = P$, where P is the covariance of Xhat.
The information from LOCWM is converted to a number of different variables used by other parts of the robot software. They are converted to Relative distance and bearing to the ball and goals by arctangent and geometry calculations and communicated to the behaviour control module BEHAVIOUR.
Each software modules send related information by writing to correspond variables defined in FieldObjects.h, software modules can use the information by importing FieldObjects.h, it consists of object types: self and teammates, ball, beacons, blue and yellow goal and goal posts, unknown goal posts, field corners, goal gaps and opponent robots. The objects that stored in the robot world model are the robot's own position, the ball position and the goal position. Each of these object is stored with x coordinate (cm), y coordinate (cm), orientation (radians -PI to PI), x coordinate estimate error, y coordinate estimate error and heading estimate error.

When LOCWM is first executed, it initialises parameters for a range of purposes. The parameters that are initialised include:

- reset calculation (OBJECT_ERROR_THRESHOLD, OBJECT_ERROR_DECAY, RESET_SUM_THRESHOLD, RESET_NUM_THRESHOLD).
- Kalman filter state XHAT and square root of covariance S.
- location of fixed field objects (x, y coordinates (cm) of goals, goal posts, corners and we assume Unknown goal posts to be in the center of the goal).

On every time frame, LOCWM performs the following sequence of operations:

1. it checks for game states after robot has been penalised, the KF is reset (to mid field in x direction with uncertainty of that direction 20cm, and large uncertainties in y and) .

FieldObjects.h					
Object Number	Description	Written From	Read by	Attributes used	Used in 2008?
0	Self	LOCWM	BEHAVIOUR LINES	WmX,wmY, wmOrientation, sdx, sdy	
1,2,3	Teammates	WIRELESS	BEHAVIOUR		No
4	Ball	VISION	LOCWM	VisionBearing, visionDistance, wmBearing, wmOrientation	
5,6	Yellow and Blue Beacons	VISION	LOCWM		No
7,8,9	Blue Goal, goal posts	VISION LINES	LOCWM LINES	VisionBearing, VisionDistance, wmBearing, wmOrientation,	
10,11,12	Yellow Goal, goal posts	VISION LINES	LOCWM LINES	VisionBearing, VisionDistance, wmBearing, wmOrientation,	
13,14	Unkown yellow and blue goal post	VISION LINES	LOCWM	VisionBearing, VisionDistance,	
15,16	Unkown L and T corner	LINES	LOCWM	VisionBearing, VisionDistance,	
17-22	Yellow goal side's corners	LINES	LOCWM	VisionBearing, VisionDistance,	
23,25	Center T corners	LINES	LOCWM	VisionBearing, VisionDistance,	
24	Center circle	LINES	LOCWM		
26-31	Blue goal side's corners	LINES	LOCWM	VisionBearing, VisionDistance,	
32-33	Goal gaps	VISION	BEHAVIOUR		NO
34-37	Opponent robots	VISION	BEHAVIOUR		NO
Globals.h					
	WirelessfieldObj	LOCWM	LOCWM LINES	Seen, x, y, sdx, sdy, sdtheta, orientation	
		LOCOMOTION	LOCWM	OdomForward, OdomLeft, OdomTurn	

2. It then scans through seen field objects (ball, goals, goal posts, identified corners), ignoring unknown objects (unknown goal posts and corners) and calls Kalman filter to run object measurement updates on the objects given the measurements provided from VISION (visionDistance and visionBearing) and converted to relative measurement by the following calculation done in the measurement updates:

$$R_{b_{rel}} = \begin{pmatrix} cos^2\theta_b R_{r_b} + r_b^2 sin^2\theta_b R_{\theta_b} & sin\theta_b cos\theta_b \left(R_{r_b} - r_b^2 R_{\theta_b}\right) \\ sin\theta_b cos\theta_b \left(R_{r_b} - r_b^2 R_{\theta_b}\right) & sin^2\theta_b R_{r_\theta} + r_b^2 cos^2\theta_b R_{\theta_b} \end{pmatrix}$$

And

$$S_{b_{rel}} = \sqrt{R_{brel}} = \begin{pmatrix} cos\theta_b \sqrt{R_{r_b}} & -sin\theta_b r_b \sqrt{R_\theta} \\ sin\theta_b \sqrt{R_{r_b}} & cos\theta_b r_b \sqrt{R_\theta} \end{pmatrix}$$

$$S_{b_{rel}} \times S_{b_{rel}}^T = R_{b_{rel}}$$

Where "r" means distance, "θ" means bearing to the object.

3. After all the seen field objects processed, it calls Kalman filter to run time update with the odometry information from LOCOMOTION

4. If wireless information is received from other robots, it calls Kalman filter to perform a special measurement update if the ball is not seen since last few frames.

5. LOCWM does some checks before it outputs results to the other modules:

Outliers - Outlier detection has been implemented on ball, goals and goal posts, corners so any measurements that are significantly different from the current estimate due to error in VISION or LINESDETECTION measurements do not get included in the Kalman Filter update.

Reset - Outlier counts for each object were keep recorded, if the amount of outliers done to goal posts or identified corners exceed a pre-defined OBJECT_ERROR_THRESHOLD recently, it calls Kalman filter to reset variances. This is implemented to deal with the shifting rule that robot can possibly 'kidnapped' or rotated which measurements

will always be ignored by Kalman Filter and end up in an error loop if this is not implemented.

Constraining robot position in the field (Clipping) – Due to noise in the measurements, the Kalman Filter may not always return values of (x, y) that are valid in the robot soccer application, that is, within the soccer field. The simplest way of solving this is to clip the individual coordinate that is out of range back into the field. A more insightful solution is to use the fact that there is an estimate of the certainty of the measurements and adjust the robot back onto the field taking into account the uncertainty and correlations in the variables. The boundary limit is set to 20cm off by the sideline, 40cm off by the goals side line.

And finally it calculates estimate relative distance and bearing of the ball and goals to the robot (wmBearing and wmDistance):

fieldObjects[obj].wmBearing $= unwrap(atan2(objY - robotY, objX - robotX) - robotOrientation)$
fieldObjects[obj].wmDistance $= \sqrt{(objY - robotY)^2 + (objX - robotX)^2}$
To be used in BEHAVIOUR, estimate robot location and uncertainty (wmX, wmY, wmOrientation, sdx, sdy) to be used in LINEDETECTION.

This year we have used a modern Kalman Filter algorithm for localisation and world modeling. The unscented Kalman filter (UKF) uses a deterministic sampling technique known as the unscented transform to pick a minimal set of sample points (called sigma points) around the mean. These sigma points are then propagated through the non-linear functions and the covariance of the estimate is then recovered. The result is a filter which more accurately captures the true mean and covariance. In addition, this technique removes the requirement to analytically calculate Jacobians, which for complex functions can be a difficult task in itself.

Unscented (Sigma Point) Measurement Update:
Given
$\hat{X}, S(S = \sqrt{P}), S_R(S_R = \sqrt{R})$,y(measurement) and constant k≈2
Algorithm to Update X̂,S

Step 1

$$w_i : i = 0...2nx; x_i : i = 0...2nx$$
$$w_0 = kk + nx; x_0 = \hat{X}$$
$$w_i = 12(k + nx); i = 1...2nx$$
$$x_i = \hat{X} + \sqrt{k + nx}S[.][i]; i = 1...nx$$
$$x_{i+nx} = \hat{X} - \sqrt{k + nx}S[.][i]; i = 1...nx$$

Step 2

$$m_x = [|\sqrt{w_0}X_0|\sqrt{w_1}X_1|...|\sqrt{w_{2nx}}X_{2nx}]$$
$$m_y = [|\sqrt{w_0}Y_0|\sqrt{w_1}Y_1|...|\sqrt{w_{2nx}}Y_{2nx}]$$
$$m_1 = [|\sqrt{w_0}|\sqrt{w_1}|...|\sqrt{w_{2nx}}]$$

Step 3

$$\bar{y} = m_y m_1^T$$
$$P_y = (m_y - \bar{y}m_1)(m_y - \bar{y}m_1)^T$$
$$P_{xy} = (m_x - \hat{X}m_1)(m_y - \bar{y}m_1)^T$$
$$K = P_{xy}(P_y + R)^{-1}$$

Update

$$S = ht(m_x - Km_y, KS_R)$$
$$\hat{X} = \hat{X} - K(\bar{y} - y)$$

where “ht” is the ‘Householder Transformation’

3.1 Robocup 2008

Given the time constraint and hardware issues, we have not been able to fully tested LINEDETECTION, localisation with Odometry information and wireless ball sharing on the real world situation, so they were not being used during the competition. Game controller was not able run wirelessly because of the interference by other wireless game controllers in the stadium, game states were manually controlled by pressing the NAOs chest button when the robot is not connected by cable to the game controller machine. In a result with the game rules, the robots were rarely get penalised.

4 Behaviour

4.1 Strategy

Moving to the Nao League, we aimed to carry over the overall strategies that had proved successful for us in previous years of the Aibo League. Our strategy involved Trying to find the ball as quickly as possible, moving as fast as possible towards the ball when it is seen, and reducing the amount of wasted time when at the ball by only reducing speed when absolutely required. This strategy works well when you can move at an equal or greater speed to your opponents. The basis behind this is that if you can get to the ball faster than your opponent you get much more possession. You are therefore able to keep control of the ball and give your opponents less chances to score against you.

Upon development of our walk we found that the ball would be effectively 'kicked' when the robot would walk into it. This played well into our overall strategy as it meant that we would not need to stop and kick the ball every time we wanted to move it, greatly reducing our time spent manipulating the ball. There are a few advantages to using this method. It allows the robot to continue moving at full speed. Also once the ball has been 'kicked' the robot is already moving towards its final destination, effectively giving you a head start on your opponents. In this way our motion was more like a form of dribbling in regular soccer than a shot. The major downside was that it was difficult to control the direction of the ball, with the only controlling factor being the direction of approach to the ball which still would not give an accurate angle depending on the manner in which the foot contacted the ball. This randomness both helped and hurt us during the competition, leading to goals both for and against.

4.2 Searching

To find the ball we used a simple searching method. The robot would pan its head twice from side to side, one pan with the robots head relatively level, the second with its head looking down. Following this the robot would then turn approximately 90 degrees. This was repeated 4 times to complete a full circle the robot then taking

a short walk to the next search location. This movement of location was to account for balls that were close to the robot, but could not be seen without bending over excessively.

4.3 Chasing

Chasing was used as the core component to our behaviour. Once a ball is seen the relative distance and bearing of the ball from the robot was used to calculate an arced walk of four steps. These four step arcs are generated as the previous one is completed. As the robot approached the ball it enters a region in which it cannot track the ball using only its head. Upon entering this region the robot would move over the position of the last seen ball. This results in a 'kick' and the robot would hopefully then be able to see the ball again and continue with its next chase.

To add some intelligence to this behaviour, the walk arc was calculated not to move to the ball, but to a position generated using the positions of the robot, the ball and the goals. The aim was to approach the ball while moving towards the opponents goals, thereby moving the ball towards the opponents' goals and away from our own.

5 Locomotion

Given the limited preparation time, and lack of prior experience with bipedal walking, the Aldebaran locomotion engine (alMotion) was used for all movement in 2008. This included the built-in *open-loop* walk engine, where only the functions `walkStraight` (always forward), `walkArc` and `turn` were called by behaviour.

There are two functions provided by alMotion that parameterise the robot's gait; `setWalkConfig` and `setWalkExtraConfig`. These parameters provide a small scope with which to change the geometry and frequency of the gait. Through manipulation of these parameters small improvements in speed or stability can be obtained.

The NAO uses very high gain position control to track trajectories produced by the walk engine. The trajectories produced by the simplified walk engine are imperfect. Instead of rigidly following these trajectories the joint stiffness was reduced, and hence the control gain was reduced, to allow the robot to 'settle' into a more

natural improved gait. This provided a greater extent with which the gait could be modified, and resulted in a walk that was both faster, and more stable, than the default parameters.

5.1 Optimisation of Walk Parameters

As an initial step, using an early version of the Alderbaran walk as a base, a slow walk was developed through the manual adjustment of walk parameters and global stiffness, that is, specifying a lowered stiffness value that is applied to every motor. The low-stiffness walk used in 2008 was then created through iterative improvements resulting from both varying the walk parameters, and setting the stiffness *individually* for each motor in the legs and arms. The effect of changing the walk parameters and stiffness values were not independent, for example, the ZMP offset walk parameters and ankle roll stiffness were strongly related.

Precedence was given to maintaining a low stiffness value in a joint. For example, consider the ankle pitch joint which had the lowest stiffness value in the legs (0.25). This low value meant that the robot would fall if it lent too far forward, consequently walk parameters were selected to maintain an upright stance while walking. Similar compromises were made for other joints in the legs. This process resulted in a considerable range of stiffness values, from 0.1 in the shoulders to 0.7 in the hip pitch joint.

5.2 Results

The speed of the low-stiffness walk was measured to be 13.9±0.2 cm/s. This was 60% faster than the default Aldebaran walk settings [Kulk, 2008].

The low-stiffness walk represents a 59% improvement in efficiency compared to the default Aldebaran walk [Kulk and Welsh, 2008]. This reduction in power consumption brings the battery current draw below the manufacturer's recommendation for both nominal and peak current, both of which are exceeded by the default walk settings. Similar results were seen in the individual motors; lowering the stiffness brings the motor current below their recommend nominal currents, whereas the default settings do not.

6 Final Soccer Results

1. **NUManoids (Australia and Ireland)**
2. GTCMUnited (USA)
3. Korettes (Greece)

7 Acknowledgments

The NUManoids are grateful to all colleagues, friends, previous members, fans and supporters of the team including the Faculty of Engineering and Built Environment, the School of Electrical Engineering and Computer Science, and CDSC at the University of Newcastle, Australia.
The NUManoids have been supported in part by the Science Foundation of Ireland under a Research Professor award: SFI 07/RPR/I177.

Links to the NUbots' publications can be found at the NUbots' webpage

`http://robots.newcastle.edu.au/`

References

[Quinlan et al., 2005] .J. Quinlan, S.P. Nicklin, K. Hong, N. Henderson, S.R. Young, T.G. Moore, R. Fisher, P. Douangboupha, S.K. Chalup, R.H. Middleton, and King. R. The 2005 NUbots Team Report. EECS technical report, University of Newcastle, 2005. Available at http://robots.newcastle.edu.au/publications.html.

[Quinlan et al., 2006] .J. Quinlan, S.P. Nicklin, N. Henderson, R. Fisher, F. Knorn, S.K. Chalup, R.H. Middleton, and King. R. The 2006 NUbots Team Report. EECS technical report, University of Newcastle, 2006. Available at http://robots.newcastle.edu.au/publications.html.

[Henderson, 2005] Henderson. Digital Image Processing In Robot Vision. Technical report, University of Newcastle, 2005. Available at http://robots.newcastle.edu.au/publications.html.

[Seysener, 2003] Seysener. Vision Processing for RoboCup 2003. Technical report, University of Newcastle, 2003.

[Seysener et al., 2004] . J. Seysener, C. L. Murch, and R. H. Middleton. Extensions to Object Recognition in the Four-Legged League. In D. Nardi, M. Riedmiller, and C. Sammut, editors, Proceedings of the RoboCup 2004 Symposium, LNCS. Springer, 2004.

[Kulk, 2008] . Kulk. Nao Walk Comparison [Video]. 2008. Available at http://www.youtube.com/watch?v=x9mCOxKE4I0.

[Kulk and Welsh, 2008] . Kulk and J. Welsh. A Low Power Walk for the NAO Robot. Proceedings of ACRA (to appear), 2008.