Software Requirements Specification

Swinbots - Swinburne RoboCup Team
SEP Group 1, 2004

Author(s):	Andrew Walker - 1262106
	James Fraser - 1261371
	Shane Harvie - 1262246
	David Reed - 1266659
	Luke Pitcher - 1262769
	Nickolas Wanke - 1229338
	Dejan Zigic - 1265636
	Michael Henderson - 1261487
Last Modified:	October 4, 2004
Version:	*Revision* : 1.62

Modification History Table

Date	Version	Description	Author
25/02/04	0.1	Added Document Framework	AW
01/03/04	0.2	Added Additional Requirements	AW
03/03/04	1.0	Test update	AW
03/03/04	1.3	Update for document revision histories	AW
15/03/04	1.4	Updated Functional Requirements	DZ
15/03/04	1.5	Fixed the Broken Build	AW
18/03/04	1.7	Grammar changes	LP
18/03/04	1.7	Update of the tactics Functional Requirement	LP
18/03/04	1.8	Fixed Compile Error	LP
18/03/04	1.8	Update of the tactics Functional Requirement	LP
21/03/04	1.10	Edited Document Purpose and Scope	SH
21/03/04	1.12	Added headings from BSE Checklist	SH
21/03/04	1.13	Added more headings from BSE Checklist	SH
22/03/04	1.15	Maintainability and extendibility non functional requirements added	LP
22/03/04	1.16	Use Cases added	DZ
22/03/04	1.17	Performance and Reliability Req's added to non-functional req's	DZ
22/03/04	1.18	updated last few Functional Requirements for Tactics	DR
23/03/04	1.19	Added in the rest of the Tactic Functional Requirements	LP
25/03/04	1.20	Minor change to wording in Tactics Functional Requirements	LP
25/03/04	1.21	Added FRs for Tactics Debug Tool, removed physics sim related material, added environment relationship diagram	SH
25/03/04	1.22	Added to Environment Relationship section, minor consistency changes	SH
26/03/04	1.23	Added to user task analysis	SH
26/03/04	1.24	Added content for User Task Analysis	MH
26/03/04	1.25	Added to UTA the steps of each of the tasks	SH
26/03/04	1.26	Added business org, constraints, user acceptance criteria	SH
27/03/04	1.27	Added to User Interface section	MH
27/03/04	1.28	Better Image Process Requirements. Still incomplete	AW
28/03/04	1.29	Added List of Tables and Figures, added to mod history	SH
28/03/04	1.30	Added to Product overview, mod history, Doc overview, and various other sections	SH
29/03/04	1.31	Another shot at improving the image processing functional requirements	AW
29/03/04	1.32	Changed client acceptance criteria	SH
29/03/04	1.33	Added mockups for user interface	SH
29/03/04	1.34	Amended UTA and UI sections to match UI mock-ups	MH
30/03/04	1.35	Added and altered subsubsection tags and placed newpage for each Functional Requirement	DR

Modification History Table

Date	Version	Description	Author
30/03/04	1.36	Added Data Flow Diagrams (Captions need to be changed)	DR
30/03/04	1.37	Changed subsubsection headings in functional requirements	LP
30/03/04	1.38	Initial embedded section	NW
30/03/04	1.39	RF Server updates, minor changes	NW
31/03/04	1.40	Updated Data Flow Diagram captions	DR
31/03/04	1.41	Changed 'goalie' to 'Goal-Keeper' and Functional Req headings (where neccessary)	DR
31/03/04	1.42	Changed Functional Req to have bullets for Pre and Post conditions and number lists for Normal Operating Procedure	DR
31/03/04	1.43	Changes made as per 4:30 review 31/03/04	LP
31/03/04	1.44	Included a note to refer to UTA document	LP
01/04/04	1.45	Updated reference to Appendix	LP
02/04/04	1.46	Change made to PASS and SHOOT functional reqs to include orientation and updated mod history	LP
03/04/04	1.47	Updated image processing requirements	AW
04/04/04	1.48	Repaired Mod His Tables	DZ
04/04/04	1.49	General Editing and QA check	JF
09/04/04	1.50	Corrected order of entries in the definitions	AW
09/04/04	1.51	Updated definitions of DirectX, DirectShow and filters	AW
09/04/04	1.52	Removed ALL data manipulated and reports generated sections	AW
09/04/04	1.53	Continued mods from review (added intelligence def., clarified some others)	JF
13/04/04	1.54	Modified to Shane's mark-up from the review	DZ
27/04/04	1.55	Addressed remaining BSE defect log items	JF
27/04/04	1.56	Fixed calibration / thresholding issue	JF/AW
06/05/04	1.57	Fixed blank lines and updated revision history	JF
21/07/04	1.58	added appendix with updates	JF
14/08/04	1.59	Added changes to section 5.5 to the appendix (After Release Changes)	DZ
01/10/04	1.60	Minor changes to After release changes	SH
02/10/04	1.61	Added abstract and intended audience	SH
03/10/04	1.62	Added changes to embedded requirements	JF

Abstract

This document defines the software requirements for the 2004 RoboCup team. It describes the system that is to be produced and the major sub-systems that will constitute the final product. It provides an analysis of all tasks that will be performed by the user. Each of the major sub-systems is then broken down into a list of functional requirements. Interface requirements and non-functional requirements are also defined. Some use cases that were used during the requirements elicitation stage are presented.

This document forms an agreement between the stakeholders of the project (defined below) regarding the functionality to be implemented in the final product. It is also intended to be read and understood by members of both past and future Swinburne RoboCup teams, and interested peer groups.

Audience

The target audience for this document is:

- The Client — Dr Ali Bab-Hadiashar
- Project Team Members
- Project Supervisors

Contents

List of Figures

1 Introduction

1.1 Document Purpose

This document presents the software requirements for the control software of the Swinburne RoboCup team, 2004.

The target audience for this document is:

- The Client — Dr Ali Bab-Hadiashar
- Project Team Members
- Project Supervisors

This document forms an agreement between the above parties regarding the functionality to be implemented in the final product. It is also intended to be read and understood by members of both past and future Swinburne RoboCup teams, and interested peer groups.

This document is the main deliverable output from the Requirements Elicitation stage. It is intended to be a baseline to supply all requirements information to the Client. It will provide a foundation for subsequent software assessment and approval. Furthermore, it also provides the development team with a basis for on-going software design.

1.2 Document Scope

This document details the functional, non-functional and user interface requirements as specified by the client and the rules of the RoboCup competition. The functional requirements have been divided into six modules based on the modules defined in the 2004 Robocup Project Plan. They are as follows:

- Image Server
- Robot Client
- Tactics Engine
- Radio-Frequency Server
- Embedded Software
- Tactics Debug Tool

All of the above software products play an integral role in the success of the RoboCup project. Each component represents a 'module' of an overall software system, and as such, each component must be designed and implemented to operate seamlessly within this system.

The requirements specification for the Mechanical and Electronic Hardware will not be covered in this document.

1.3 Requirements Elicitation Team

The Requirements Elicitation team will consist of all RoboCup 2004 team members; they are:

- Andrew Walker
- James Fraser
- Shane Harvie
- David Reed
- Luke Pitcher
- Nickolas Wanke
- Dejan Zigic
- Michael Henderson

By involving all team members in the Requirements Elicitation effort, an understanding of the project by the team as a whole will be promoted. Each team member will, at different times throughout the project, undertake the role of Systems Engineer and Developer, so inclusion of people responsible for these roles is done by default.

The client will also be consulted during the Requirements Elicitation process on a number of occasions:

- Initial consultation to help define functional requirements
- Review of first draft of SRS
- Final sign-off of SRS

This document will also be subject to both informal and formal review by the RoboCup 2004 Development Team and the BSE students.

1.4 Reference Documents

SRS_checklist.pdf (author unknown, available on Blackboard) 2003 RoboCup SRS (SRS.pdf)

Whilst not used explicitly in the creation of the SRS, the following documents may prove useful for understanding the requirements:

- 2004 RoboCup Project Plan (PP.pdf)
- 2004 RoboCup Software Quality Assurance Plan (SQAP.pdf)
- Justification for the Omission of the Physics Simulator (PhysicsSimOmissionJustification.pdf)

1.5 Terminology Used

1.5.1 Definitions

Average propagation time: Time between an image being taken (frame from overhead camera), and the receipt of the subsequent image, showing the result of the command which was derived from the previous image. Referring to Figure 4, the average propagation time is the sum of the time taken by each of the individual components to complete one full cycle.

Calibration (Image Processing): Adjustment of image processing system parameters, according to variable environmental conditions including extraneous/ambient lighting.

Client: End user for whom the product is designed and developed; in this case: Dr Alireza Bab-Hadiashar, from the School of Engineering and Science, Swinburne University of Technology.

Colour Space: A colour space is a way of uniquely describing a colour in an image. Certain colour spaces are more suitable for particular applications, such as TV transmissions or describing output to a computer monitor. Example colour spaces include RGB (one byte each for red, blue and green intensity — making 24 bits total information per pixel) and YUV, which is also known as YCbCr (one byte for luminance/brightness, and two bytes for chrominance/colour (blue and red))

DirectShow: A subset of DirectX focused on audio and video streaming, which can be adapted for use with real-time image processing.

DirectShow Filter: Software development using the DirectShow API relies upon interconnected software modules which are known as filters. An example of a DirectShow filter would be an MPEG codec.

DirectShow Transform Filter: A DirectShow filter specialised for operations involving the transformation of video from one state to another. An example of a transformation filter is a filter for correcting contrast in an image.

DirectShow Source Filter: Filters which provide data. Examples include file sources, TV capture cards and Digital Video cameras. Source filters interact with lower level hardware drivers if required.

DirectShow Sink Filter / Renderer / Null Renderer: Filters which accept data. In the case of Renderers, the final output is presented as an image or a sound. A Null renderer or sink filter accepts all data, but ignores it.

DirectX: Microsoft API for multimedia applications. Has sections which deal with video, audio, rendering and input.

Dribbler/Roller: Device used on the robot, enabling it to acquire and control the ball, with a view to maintaining possession whilst in motion.

F180 League: The RoboCup league in which the Swinburne team aims to compete in the future. It is also known as the 'small size' league.

Image: A digital image, comprised of many pixels.

Image Format: A description of the internal layout of bytes for pixels.

Intelligence: In the context of RoboCup, the ability of a robot to act in a manner that seems 'rational', i.e. cursory human observation would conclude that the robot is acting in a way congruent with its instructions, such as: moving to a specific point along the shortest (or best) path, blocking an opposition kick, or shooting at the correct goal. Blatantly 'dumb' actions, such as going to the wrong point on the field, trying to kick the ball without first having possession of the ball etc., constitute an unintelligent RoboCup robot.

Kicker: Device used by the robot to effect a kick on the ball.

Maintainability: Characteristic of software pertaining to easy maintenance and updating.

Pixel: Contraction of the term 'Picture Element', which refers to the discrete elements ('dots') of a digital image. Pixels are represented digitally as bytes, the format of which is dependent upon the type of camera from which the image was acquired.

RoboCup: International competition where entrants develop and build teams of robots to contest soccer matches.

Segmentation: The process of identifying and separating the pixels of an image into groups based on a classification method, allowing meaningful representation for further processing.

Tactics: Strategies to be employed during soccer matches, incorporated into the AI system.

Tactics Engine: Module of the system which is responsible for analysing the Image Processing Data, and deciding on team and individual robot strategies.

Verifiability: The degree to which a software system is conducive to being verified against stated requirements.

Thresholding: A threshold is a boundary condition for a classification method. With regards to the image processing for RoboCup, a threshold defines the limits which determine whether a pixel will be placed into a given classification group. Thresholding is the process of applying these thresholds to the image.

1.5.2 Acronyms and Abbreviations

IP Image Processing

PC Personal Computer

PP Project Plan

RF Radio Frequency

SDLC Software Development Life Cycle

SQAP Software Quality Assurance Plan

SRS Software Requirement Specification

TBA To be announced

TP Test Plan

UID User Interface Design

UTA User Task Analysis

1.6 Constraints

The constraints which were placed upon this Requirements Elicitation effort are as follows:

- The functional requirements of the Tactics Engine depend largely on the type of tactics rules implemented. Since the decision about which rule-method to use is yet to be made, only the basic 'high level' functional requirements have been defined, which would apply to any rule-method chosen. Development of prototypes of the Tactics Engine will enable more precise definition of the requirements for this module.
- The above item also influences the definition of the Tactics Debug Tool software requirements. Whilst it was possible to develop some reasonably detailed requirements for the Tactics Debug Tool, refinements may need to be made once the Tactics Engine requirements have been fully developed.
- General time constraints mean that more abstract functions, such as 'response to referee signals', were deliberately left out to afford sufficient time for developing the fundamental functionality of the system.
- Forced use of image processing hardware has resulted in the functional requirements for this module being very specific.

1.7 Overview of Business Organisation

The business organisation is Swinburne University of Technology, Higher Education Division. Two major goals of Swinburne are:

- To educate students and prepare them for the workforce
- To attract future students to the university so that the first goal can be achieved

The RoboCup project addresses both of these goals. Whilst enabling the students that are involved in the project to develop and demonstrate the skills they have gained, it also serves the purpose of promoting the university. This will be achieved primarily through the Open Day Presentation, which is the major deliverable for the project.

1.8 Document Overview

This document — the Software Requirements Specification (SRS) — identifies the functional and non-functional requirements of the system. It also provides a User Task Analysis, and Client Acceptance Criteria. It has 10 major sections and 2 appendices:

- **Introduction** (Section 1) provides an overview of the entire SRS document, the project and the product being specified.
- **Product Overview** (Section 2) describes, in more depth, the product that is to be produced. It gives an overview of the total system functionality.
- **User Task Analysis** (Section 3) provides a comprehensive analysis of all tasks that will be performed by the user. This section assists in the development of Functional Requirements.
- **Functional Requirements** (Sections 4-8) lists those requirements which are essential for the functionality of the system, for each of the 5 major modules comprising the system.
- **Interface Requirements** (Section 9) describes the requirements for the user interface, including mock-ups of the proposed user interfaces.
- **Non-Functional Requirements** (Section 10) provides the requirements which are to be met by the system, but do not explicitly affect the functionality of the system.
- **Use Cases** (Section 11) Provides some scenarios for anticipated uses of the system.
- **Relationship between the Product and the Environment** (Appendix A) shows a diagram representing the relationships between the product and the environment.
- **Data Flow Diagram** (Appendix B) shows a diagram representing the flow of data in the system.

2 Product Overview

2.1 Background

RoboCup is an international event held annually, the stated purpose of which is: “By the year 2050, develop a team of fully autonomous humanoid robots that can win against the human world soccer champion team”. This event requires participants to design, develop and build teams of robots capable of playing a game of soccer; there are different leagues specifying various rules for robot size, field size and ball type. The Swinburne RoboCup team is aiming to enter the ‘F180’ league, the rules of which

stipulate that robots must fit inside a cylinder of 180mm diameter. Furthermore, teams using 'global vision' (such as Swinburne, as opposed to having on-board cameras for each robot) must have robots of height no greater than 150mm. This league uses an orange golf ball as the soccer ball.

Swinburne University began its RoboCup campaign in 2003, building six robots from scratch. In addition to building the physical robots, the team also addressed setup needs, implemented basic RF communication, image processing and some simple embedded code. These robots display minimal intelligence, and many improvements are certainly necessary for this year.

The resources required to bring the team to competitive level are not available this year. As a result, our client would like to pit this year's robots against last year's, providing a clear demonstration of the progress made. As the RoboCup project at Swinburne is intended to be ongoing (this being its second year), all technical developments and products must be reproducible, extensible and maintainable; there should be no 'disposable' elements.

2.2 General Product Functionality

As discussed above, the system involves the following software components:

- Image Server
- Tactics Engine
- Radio-Frequency Server
- Embedded Software
- Tactics Debug Tool

A high level description of the system functionality is as follows:

1. The Image Server takes an image of the field, including all of the robots (both Swinburne and opposition), the ball and the goals.
2. The Image Server identifies the position of each of the robots — both friendly and opposition — and the ball.
3. The Image Server sends position data to the Tactics Engine.
4. The Tactics Engine processes the data and decides on a team objective, and a course of action for each of the robots.
5. The Tactics Engine sends the robot instructions to the RF server.
6. The RF server sends the instructions to the Embedded software.
7. The Embedded software processes the instructions and causes the robots to carry-out their instructions.

The Tactics Debug tool enables the developers to analyse the tactics decisions which are made during a game. The Image Server will log the robot and ball positions during a game, saving this data to a file. After the game has finished, the Tactics Debug tool will 'play back' the game from this file, showing the robot positions for each frame, and a corresponding visual representation of the tactics decision process. This will enable users to evaluate the tactics decision in the context of the recorded robot positions.

The purpose of this sub-system is to enable validation (deficiencies in the Tactics Engine will be exposed), and verification (the team can check the performance of the Tactics Engine against its requirements specification), in addition to determining how effective the collaboration is between the Tactics Engine

and other sub-systems. The Tactics Debug tool will also assist in assessing the effectiveness of the collection of tactics which have been supplied to the robots.

Appendix B shows the data flow diagram for the system.

2.3 Relationship between this product and other products

This product is an integration of the modules discussed above and is a stand-alone system. It does not interact with other products.

2.4 Client Acceptance Criteria

The system must, at least, implement the following:

- the 'go to' capabilities of the robots (go directly to a field position);
- the 'kick to' capabilities of the robots (kick to a field position/goal);
- the 'block' capabilities of the robots (block an incoming ball, for goal-keeper); and
- team tactics that will lead to the robots behaving intelligently.

3 User Task Analysis

See User Task Analysis 2004 document (UTA.pdf).

4 Functional Requirements - RF Server

The RF server receives packets from a UDP socket, and then broadcasts those packets across the RF communication channel. Unlike the Image server, it does not need to maintain a client list.

4.1 Receive robot instruction packet (HIGH)

Receive a robot instruction packet.

Function Name

RF_RECEIVE

Detailed Description

This function receives data from the Tactics Engine to broadcast to a robot.

Pre-conditions

- RF is initiated and ready to receive data

Normal Operating Procedure

1. Accept data from Tactics Engine
2. Prepare to transmit data

Post-conditions

- Instruction packet available on the send buffer

Exceptions

Corrupt Packet

4.2 Broadcast robot instruction packet (HIGH)

The instruction packet is broadcast via Bluetooth RF.

Function Name

COMMS_TRANSMIT

Detailed Description

This function transmits data to robots based on commands sent from the tactics engine.

Pre-conditions

- Instruction packet is available on send buffer

Normal Operating Procedure

1. Extract instruction packet from send buffer
2. Generate RF packet
3. Transmit data via RF

Post-conditions

- RF packet sent to applicable robot(s)

Exceptions

None

4.3 Receive robot data packet (HIGH)

Description

Packet is received via RF.

Function Name

COMMS_RECEIVE

Detailed Description

A packet is downloaded via Bluetooth RF communications from the robot.

Pre-conditions

- RF Packet is received from a robot

Normal Operating Procedure

1. Decode and verify packet
2. Add packet to receive buffer

Post-conditions

- Valid packet available on receive buffer

Exceptions

Received packet corrupt

5 Functional Requirements - Embedded software

The embedded software manages all robot based awareness, i.e. movement, control of kicker and dribbler, and communications with the host PC(s).

5.1 Receive robot instruction packet (HIGH)

Description

Receives an RF packet from the PC.

Function Name

ROBOT_RECEIVE_PACKET

Detailed Description

The robots are controlled via continuous commands sent from the host PC. These packets arrive via Bluetooth RF communications; packets are downloaded and executed (acted upon) by the robot.

Pre-conditions

- Robot is operational
- Communications channels are established

Normal Operating Procedure

1. Interrupt triggered — incoming packet
2. Packet downloaded into buffer
3. Header, footer examined, packet validity verified
4. Packet parsed to extract instruction and any accompanying data
5. Instruction executed

Post-conditions

- Data received successfully, and executed properly

Exceptions

Corrupt/invalid packet

5.2 Send robot data packet (MEDIUM)

Description

Sends an RF packet to the Host PC.

Function Name

ROBOT_SEND_PACKET

Detailed Description

Creates a correctly structured communications packet, and transmits over the Bluetooth RF link.

Pre-conditions

- Communications channels are established

Normal Operating Procedure

1. Communications packet built based on response command and any applicable data
2. Packet sent via Bluetooth RF link

Post-conditions

- Data transmitted successfully

Exceptions

None

5.3 Stop (HIGH)

Description

Halts the robot.

Function Name

ROBOT_STOP

Detailed Description

Causes the robot to immediately stop all its tasks, and wait for further instructions.

Pre-conditions

- 'Stop' instruction received

Normal Operating Procedure

1. Stop all operations

Post-conditions

- Robot stopped

Exceptions

None

5.4 Reset (HIGH)

Description

Resets the robot.

Function Name

ROBOT_RESET

Detailed Description

Stops the robot, re-establishes communications links, and resets all robot parameters to initial operating conditions.

Pre-conditions

- 'Reset' instruction received

Normal Operating Procedure

1. Stop all operations
2. Reset robot

Post-conditions

- Robot reset successfully

Exceptions

Communications failure

5.5 Determine the required velocity for each wheel (HIGH)

Description

Calculate desired wheel speeds.

Function Name

ROBOT_WHEEL_VELOCITIES

Detailed Description

Robot receives translational and rotational velocity commands, and determines the wheel speeds which will cause the robot to realise these 'resultant' velocities.

Pre-conditions

- Translational and Rotational velocities received

Normal Operating Procedure

1. Apply inverse-kinematic equations to received velocities
2. Output wheel speeds to drive control loop

Post-conditions

- Drive system has updated desired speed for all three wheels

Exceptions

None

5.6 Control speed of a wheel (HIGH)

Description

Maintains the actual speed of a wheel based on the desired speed value.

Function Name

ROBOT_UPDATE_WHEELSPEED

Detailed Description

Each of the robot wheels have an encoder for speed feedback, and a PID digital control loop is used to modulate the motor current to maintain a particular set speed. This function must be called on a cyclic basis to provide 'continuity' in the control loop.

Pre-conditions

- Wheel speed set-point is a valid value

Normal Operating Procedure

1. Read wheel speed set-point
2. Perform PID error calculation on desired wheel speed and actual wheel speed
3. Set desired wheel speed to newly-calculated speed

Post-conditions

- Actual wheel speed adjusted closer to desired speed

Exceptions

Actual wheel speed outside tolerances on set speed, PID failure

5.7 Grab ball (HIGH)

Description

Collects the ball.

Function Name

ROBOT_GRAB_BALL

Detailed Description

Gets the robot ready to 'grab' a ball by turning on the dribbler bar.

Pre-conditions

- 'Grab ball' instruction received

Normal Operating Procedure

1. Turn on dribbler bar

Post-conditions

- Dribbler bar turned on

Exceptions

Dribbler motor failure

5.8 Determine whether ball is in possession (MEDIUM)

Description

Tests for ball possession by robot.

Function Name

ROBOT_BALL_GRABBED

Detailed Description

Utilises feedback to determine if the ball is presently in contact with the dribbler bar.

Pre-conditions

- Dribbler bar active
- Ball possession queried

Normal Operating Procedure

1. Check feedback to see if ball is in possession
2. Return result

Post-conditions

- Result successfully returned

Exceptions

None

5.9 Shoot ball (HIGH)

Description

Shoots the ball.

Function Name

ROBOT_SHOOT_BALL

Detailed Description

Activates the kicker in order to effect a kick on the ball.

Pre-conditions

- Robot has possession of ball
- 'Shoot ball' instruction received

Normal Operating Procedure

1. Turn off dribbler bar
2. Trigger kicking mechanism

Post-conditions

- Kicker mechanism 'fires'

Exceptions

None

5.10 Pass ball (HIGH)

Description

Passes the ball to another robot.

Function Name

ROBOT_PASS_BALL

Detailed Description

Orients the robot according to feedback sent from the Host PC, then kicks the ball toward a particular robot.

Pre-conditions

- Robot has possession of ball
- 'Pass ball' instruction received

Normal Operating Procedure

1. Orients robot towards target
2. Trigger kicker mechanism

Post-conditions

- Kicker mechanism 'fires'

Exceptions

None

5.11 Determine whether collision has occurred (MEDIUM)

Description

Check for robot collision.

Function Name

ROBOT_COLLISION_QUERY

Detailed Description

Polls the robot's sensors to determine whether their readings suggest a collision with another robot or the edge of the playing field.

Pre-conditions

- None

Normal Operating Procedure

1. Read bump/tilt/acceleration sensors
2. Process sensor readings and interpret the data
3. Return result

Post-conditions

- Result successfully returned

Exceptions

Sensor failure

5.12 Collision response (MEDIUM)

Description

Automatic robot response to a collision.

Function Name

ROBOT_COLLISION_RESPOND

Detailed Description

If the robot's sensors suggest that a collision has occurred, the robot will move a short distance in the direction opposite to that in which the collision occurred.

Pre-conditions

- Collision status queried
- Collision has occurred

Normal Operating Procedure

1. Move robot in opposite direction to sensed collision
2. Stop robot after short preset distance

Post-conditions

- Robot moved away from source of collision

Exceptions

None

6 Functional Requirements - Image Server

The Image Server will provide image capture, image processing, image analysis and communications. It will be comprised of two pieces of software – the Image Server, and a 'DirectShow' transform filter which abstracts the complexity of video capturing procedures. The video capture and processing requirements are based on the experience of the 2003 RoboCup team, and are strongly influenced by the existing hardware (Unibrain 'FireWire' cameras) and the target platform (Windows 2000).

Communications for the Image server will be maintained using a simple application-layer protocol based on UDP. Because UDP is a 'connectionless' protocol, there needs to be implemented a simple system to maintain a single client.

6.1 Capture data from the camera (HIGH)

The camera is configured for streaming, and data from the camera is captured.

Function Name

CAPTURE_CAMERA_DATA

Detailed Description

A DirectShow filter graph capable of transmitting data to an external application is constructed. Appropriate filters are added to the graph, and the graph is started.

Pre-conditions

- COM has been initialised
- A Camera is connected to the image processing PC
- No other applications are currently using the camera

Normal Operating Procedure

1. A DirectShow graph is constructed
2. A source filter is constructed
3. A grabber filter is constructed
4. A null renderer filter is constructed
5. The filters are added to the graph
6. The source filter and grabber filter are connected
7. The grabber filter and the null renderer filter are connected
8. The graph is started

Post-conditions

- The filter graph has been constructed
- The filter graph is running

Exceptions

None

6.2 Input from the camera is received by DirectShow (HIGH)

Data captured by the camera will be transmitted to the image server for processing.

Function Name

TRANSMIT_CAMERA_DATA

Detailed Description

Data from the Unibrain FireWire camera, connected to the image processing computer, will be captured using the camera's drivers in association with DirectShow. Output from the DirectShow filter graph will then be redirected to the Image Server.

Pre-conditions

- The DirectShow filter graph is running

Normal Operating Procedure

1. Data from the camera arrives
2. The image is transmitted to the Image Server
3. Data is transmitted to a sink filter

Post-conditions

- Image Data has been transferred to the external application

Exceptions

None.

6.3 Process image from the camera (HIGH)

The image processing will find and identify shapes.

Function Name

IMAGE_PROCESSING

Detailed Description

The low-level image processing of input from the camera will examine the pixels of the input image to identify the significant shapes.

Pre-conditions

- Colour calibration information is available
- A filter graph has been constructed
- A frame is available

Normal Operating Procedure

1. Camera data is received
2. The image is converted into an appropriate Colour Space
3. The image is separated into segments by colour
4. The shapes in the image are identified

Post-conditions

- The significant regions of the current frame will have been identified

Exceptions

None

6.4 Analyse shapes from the image processing stage (HIGH)

The medium-level image processing of input from the camera will measure detailed information about the angles and positions of objects on the field.

Function Name

SHAPE_PROCESSING

Detailed Description

Supplied shapes will be grouped according to colour, position, area and other significant features. Important information will be extracted by using high-level information about RoboCup. Measurements of angle and position will take place as appropriate; it is not feasible to determine angles of opposition robots as teams will be using different robot-angle marking schemes.

Pre-conditions

- The regions in the image have been identified

Normal Operating Procedure

1. Analyse Image Regions
2. Classify Image Regions
3. Measure Positions of Robots and Balls
4. Measure Angles of friendly robots

Post-conditions

The positions (and angles where appropriate) of robots and balls will have been measured

Exceptions

None

6.5 Client connection (HIGH)

The image server receives communication from a client, which will subsequently receive output.

Function Name

COMMS_ACCEPT_CLIENT

Detailed Description

The image server receives a packet from a potential client. The sender of the packet becomes the client if the packet is authenticated. The connection packet also contains information on the team of the connecting client (yellow or blue).

Pre-conditions

- A client is not currently connected

Normal Operating Procedure

1. The image server receives a control packet
2. The control packet is authenticated
3. The image server sends an acknowledgement packet
4. The image server receives a location to stream data to
5. The image server stores the client's team

Post-conditions

- A client is connected
- The client team has been set

Exceptions

If packet authentication fails, the image server waits for another client control packet

6.6 Write to client (HIGH)

The processed image data is broken down to x, y, theta and type information, which is to be transmitted to a client.

Function Name

COMMS_WRITE_CLIENT

Detailed Description

Upon completion of the image processing and analysis, information extracted from a frame will be transmitted to a connected client.

Pre-conditions

None

Normal Operating Procedure

1. A data packet based on the output from IMAGESERVER_COMMUNICATIONS_ACCEPT_CLIENT is transmitted to the client

Post-conditions

None

Exceptions

If no client is connected, or no team has been selected, processed data will be discarded

6.7 Calibrate colours (HIGH)

The image server will provide the facility to calibrate the colours used in identifying important information in the image processing.

Function Name

CALIBRATE_COLOURS

Detailed Description

Calibration of colours will be implemented as a separate tool. This requirement will be implemented as an offline process which will be conducted prior to starting the system.

Pre-conditions

None

Normal Operating Procedure

1. An input image is supplied
2. The user guides the selection of colour calibration parameters
3. The user saves the colour calibration parameters

Post-conditions

- Output describing the calibrated colours has been written to a file

Exceptions

None

6.8 Calibrate field area (HIGH)

The location of the main features of the field relative to the image will be adjustable

Function Name

CALIBRATE_FIELDAREA

Detailed Description

Calibration of field area will be implemented as a separate tool. This requirement will be implemented as an offline process which will be conducted prior to starting the system.

Pre-conditions

None

Normal Operating Procedure

1. An input image is supplied
2. The user guides the selection of field area
3. The user saves the field area parameters

Post-conditions

- Output describing the field area has been written to a file

Exceptions

None

7 Functional Requirements - Robot Behaviour/Tactics

7.1 Stop robot (HIGH)

Sends instruction to stop a robot immediately.

Function Name

STOP_ROBOT

Detailed Description

Initial instruction to stop a robot in its current location on the field. The RF server is told to send the stop information to a particular robot, then the RF server relays the information required.

Pre-conditions

- Robot is currently moving

Normal Operating Procedure

1. Send information to RF server for packet to contain robot information and stop command

Post-conditions

- RF server receives information for packet to contain

Exceptions

None

7.2 Move robot to set location (HIGH)

Sends instruction to move a robot to a specified location on the field.

Function Name

MOVE_ROBOT

Detailed Description

Initial instruction to move a robot within the field to a designated position. The RF server is sent the command to send a packet to a particular robot containing a specified position, then the RF server relays the information required.

Pre-conditions

- Robot communication channels are established

Normal Operating Procedure

1. Send information to RF server for packet to contain robot information, move command and position information

Normal Operating Procedure

- RF server receives the information for packet to contain

Exceptions

None

7.3 Stop all robots (HIGH)

Instruction to stop all robots immediately.

Function Name

STOP_ALL_ROBOTS

Detailed Description

Initial instruction to stop all the robots in their current location on the field. The RF server is told to send the stop information to the robots. Then the RF server relays the information required.

Pre-conditions

- Robot communication channels are established

Normal Operating Procedure

1. Send information to RF server for packet to contain stop command

Post-conditions

- RF server receives the information for packet to contain

Exceptions

None

7.4 Set goal direction (HIGH)

Set the goal that the team is to attack.

Function Name

SET_DIRECTION

Detailed Description

Set up the processing such that the team is heading in the correct direction on the field. This is essential to the setup and running of a game.

Pre-conditions

- None

Normal Operating Procedure

1. Direction to attack is set by the user

Post-conditions

- Correct goal direction is defined

Exceptions

None

7.5 Kick goal or shoot (HIGH)

Instruction for a robot to kick the ball with maximum speed.

Function Name

SHOOT

Detailed Description

Instruction for robot to orientate and kick the ball with maximum speed from current location on the field. The RF server is told to send the shoot information (orientation and shoot) to a particular robot, then the RF server relays the information required.

Pre-conditions

- Robot has possession of the ball

Normal Operating Procedure

1. Get required orientation for kick
2. Send information to RF server for packet to contain robot information and shoot command

Post-conditions

- RF server receives information for packet to contain

Exceptions

None

7.6 Kick to player - pass (HIGH)

Initial instruction for a robot to kick the ball with determined speed.

Function Name

PASS

Detailed Description

Instruction to orientate and kick the ball with specified speed from current location on the field. The RF server is told to send the pass information to a particular robot. Then the RF server relays the information required.

Pre-conditions

- Robot has possession of the ball

Normal Operating Procedure

1. Get orientation information for where the ball is to be passed
2. Send information to RF server for packet to contain robot information and kick command

Post-conditions

- RF server receives information for packet to contain

Exceptions

None

7.7 Allocate role to robot (MEDIUM)

Robot is allocated a role to play in the current game.

Function Name

ALLOCATE_ROLE

Detailed Description

Each robot shall be allocated a role as either an attacker, defender or goal-keeper for each game. Each role has certain characteristics that the robots are expected to follow.

Pre-conditions

- None

Normal Operating Procedure

1. Tactics/behaviour engine is told what the role of the robot is by the user.

Post-conditions

- Role is allocated to the robot

Exceptions

None

7.8 Select strategy (MEDIUM)

Select a strategy to use for play.

Function Name

SELECT_STRATEGY

Detailed Description

A strategy for a situation in play shall be selected and used to communicate information from PC to the robots. Multiple strategies will be available.

Pre-conditions

None

Normal Operating Procedure

1. Analyse the current state of the field
2. Make a decision on best overall strategy
3. Set each robot's strategy mode according to the overall strategy

Post-conditions

- The robots all have an appropriate strategy for the next frame based on the current frame

Exceptions

None

7.9 Update robot according to strategy (MEDIUM)

Robot reacts in accordance to strategy selected.

Function Name

UPDATE_ROBOT

Detailed Description

The robot shall react to a situation in accordance to the strategy selected and its surrounding environment. PC shall communicate information to robot over the RF.

Pre-conditions

- The Robot's strategy mode has a valid value

Normal Operating Procedure

1. Get the robot's strategy mode
2. Make a decision on a desirable next action for the robot
3. Carry out the action

Post-conditions

- A decision was made and sent to the Robot

Exceptions

None

7.10 Get strategy (MEDIUM)

Return the currently selected strategy.

Function Name

GET_STRATEGY

Detailed Description

Query to return the details for the current overall strategy that is guiding the system decisions; could include overall goal, individual robot goals, available robot strategy modes etc.

Pre-conditions

- Overall strategy mode has a valid value

Normal Operating Procedure

1. Get the current overall strategy mode
2. Return it's text description (from file)

Post-conditions

None

Exceptions

None

8 Functional Requirements - Tactics Debug Tool

8.1 Display visual representation of environment (MEDIUM)

Display a visual representation of the environment data.

Function Name

DISPLAY_ENVIRONMENT_DATA

Detailed Description

This function will display a visual representation of the RoboCup environment data — that is: the robot, ball and field positions. It will be a 2–D birds-eye view, equivalent to that obtained from the camera.

Pre-conditions

- Environment data is available to display

Normal Operating Procedure

1. Retrieve environment data
2. Display field
3. Display friendly robots
4. Display opposition robots
5. Display ball

Post-conditions

- Visual representation of environment data is displayed on the screen

Exceptions

None

8.2 Display visual representation of team objective data (MEDIUM)

Display a visual representation of the data that led to the choice of a team objective.

Function Name

DISPLAY_TEAM_OBJECTIVE_DECISION_DATA

Detailed Description

This function, combined with functional requirement *Display visual representation of environment* will enable the user to map a given team objective decision to the environment data that brought about that decision (i.e. the robot, ball and field positions).

Pre-conditions

- Environment data is available to map

Normal Operating Procedure

1. Retrieve environment data
2. Re-create team objective decision data
3. Display visual representation of decision data on the screen

Post-conditions

- Visual representation of team objective data is displayed on the screen

Exceptions

None

8.3 Display visual representation of robot decision data (MEDIUM)

Display a visual representation of the data that led to the choice of a robot objective.

Function Name

DISPLAY_ROBOT_DECISION_DATA

Detailed Description

This function, combined with functional requirement *Display visual representation of environment* will enable the user to map a given robot objective decision to the environment data that brought about that decision (i.e. the robot, ball and field positions).

Pre-conditions

- Environment data is available to map

Normal Operating Procedure

1. Retrieve environment data
2. Re-create robot decision data
3. Display visual representation of decision data for individual robot on the screen

Post-conditions

- Visual representation of robot decision data is displayed on the screen

Exceptions

None

8.4 Update to next frame (MEDIUM)

Updates to the next image processing frame.

Function Name

NEXT_FRAME

Detailed Description

This function updates the image processing frame to be analysed to the next frame that was logged during game play.

Pre-conditions

- There is a frame available to read

Normal Operating Procedure

1. Change the current frame to the next frame that was logged

Post-conditions

- Current frame has been updated to the next frame

Exceptions

None

8.5 Update to previous frame (MEDIUM)

Updates to the previous image processing frame.

Function Name

PREVIOUS_FRAME

Detailed Description

This function updates the image processing frame to be analysed to the previous frame that was logged during the game.

Pre-conditions

- There is a previous frame available to read

Normal Operating Procedure

1. Change the current frame to the previous frame that was logged

Post-conditions

- Current frame has been updated to the previous frame

Exceptions

None

9 Interface Requirements

9.1 Interfaces with other Products

The RoboCup system is self-contained; there are no external products needed.

9.2 User Interface Requirements

The system will provide multiple Graphical User Interfaces. All of these will allow input from the user via keyboard and mouse operations. Output is through the PC's monitor and, in some cases, through the physical actions of the robots.

9.2.1 Image Server

The image server will not provide a Graphical User Interface. A correctly configured image processing filter will, however, be able to output a video stream using the DirectShow GraphEdit tool.

9.2.2 Colour Calibration Tool

The Colour Calibration Tool is a separate application that is responsible for setting the video colour threshold levels. The Calibration Tool interface requirements are as follows:

- Provide a 3–dimensional plot of the colour data being collected from the camera.
- Allow the user to create a set of colour definitions (e.g. white, green, etc.).
- Each colour definition will be shown on the 3–dimensional plot as a wire-frame rectangular prism.
- The start and end points of each of the prism's dimensions can be adjusted using 6 slider bars.
- There will be a function to write these values to a text file for later reading by the system.

A mock-up of the Colour Calibration Tool User Interface is shown in Figure 1.

9.2.3 Tactics Debugging Tool

The Tactics Debugging Tool will provide a Graphical User Interface which displays the robot positions, and a visual representation of the tactics decision process.

The Tactics Debugging Tool user interface requirements are as follows:

- Provide a means by which to specify the file — recorded in-game by the Image Server — for reading game data.
- Provide a button to step forward through the recorded data.
- Provide a button to step backwards through the recorded data.
- Provide a means (possibly a slider) by which to set the rate at which data is cycled (zero setting would cause the buttons to work for one frame at a time).
- There will be a view to show the current positions of the robots and the ball.
- There will be a view (possibly several different styles) to represent the decision making data.

A mock-up of the Tactics Debugging Tool User Interface is shown in Figure 2

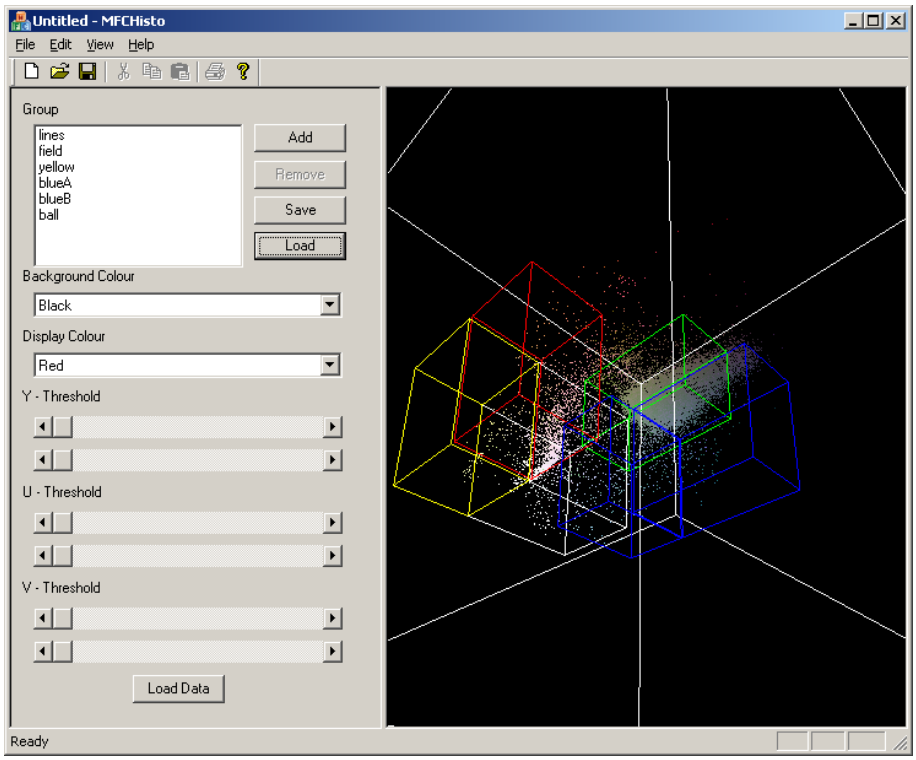

Figure 1: Colour Calibration Tool UI Mockup

9.2.4 Tactics Engine

The Tactics Engine will display the incoming information, and provide a facility to change the properties of the robots.

The Tactics Engine user interface requirements are as follows:

- Provide a 'System Settings' dialog for setting such things as which ports to use to connect to the image server.
- Provide a means (possibly a menu item) to initiate connection with the image server.
- Provide a button to start game play (Kick-off).
- Provide a button to stop game play.
- Provide a button to emergency stop to halt all robots and shut down as much of the robot's circuitry as possible.
- Provide a button for the 'go-to' command.
- Provide a button for the 'kick-to' command.
- Provide a slider for setting kicking speed.
- Provide a button to enter 'test goal-keeper mode'.
- Provide a means to resume game play (perhaps the stop button turns into a resume button once stopped).
- Provide a view of the playing field, with options to have unprocessed image (actual video), processed image (graphical representation), or no image (save cpu resources).

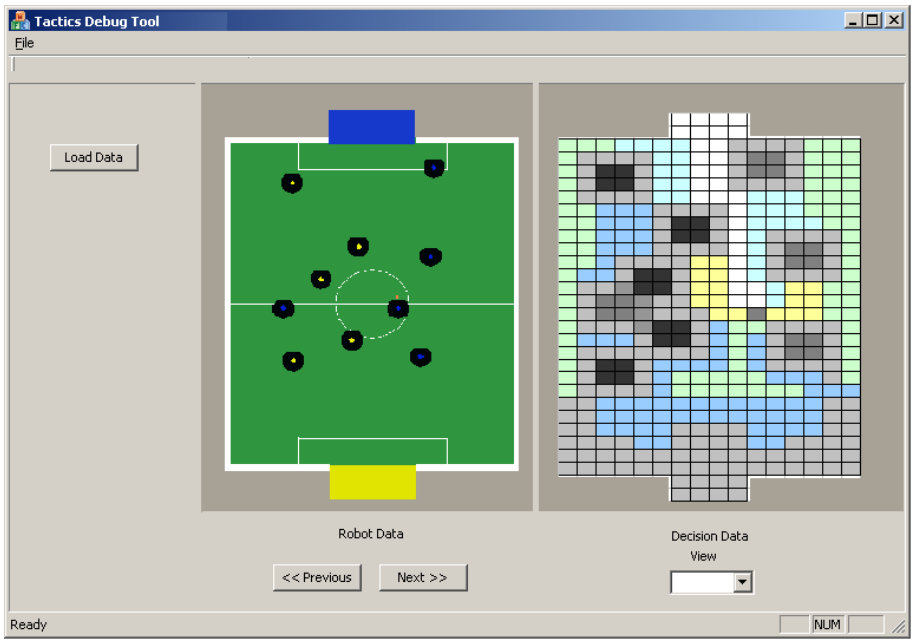

Figure 2: Tactics Debug Tool UI Mockup

- When the view is set to the processed image view, the robot objects should be 'clickable' and, when selected, that robot's information will be displayed.

A mock-up of the Tactics Engine User Interface is shown in Figure 3

10 Non-Functional Requirements

In general, any interfaces that are to be used during the development process will be required to be comprehensive in their features.

10.1 Performance Requirements

The following performance requirements were set by the 2004 RoboCup team:

- Image processing to produce more than 20 frames-per-second.
- RF to minimise latency to less than 20ms (time between packet sent and valid data available for use).
- Packet loss in RF to be minimised (measured once sending of packets has commenced).
- Average propagation time less than 150ms.
- Time taken for a Tactics Engine processing decision to be less than the average propagation time.
- The Image processing to be out by less than 5 pixels per frame.
- Robots follow exact commands outputted by the Tactics Engine at least 70 percent of the time.

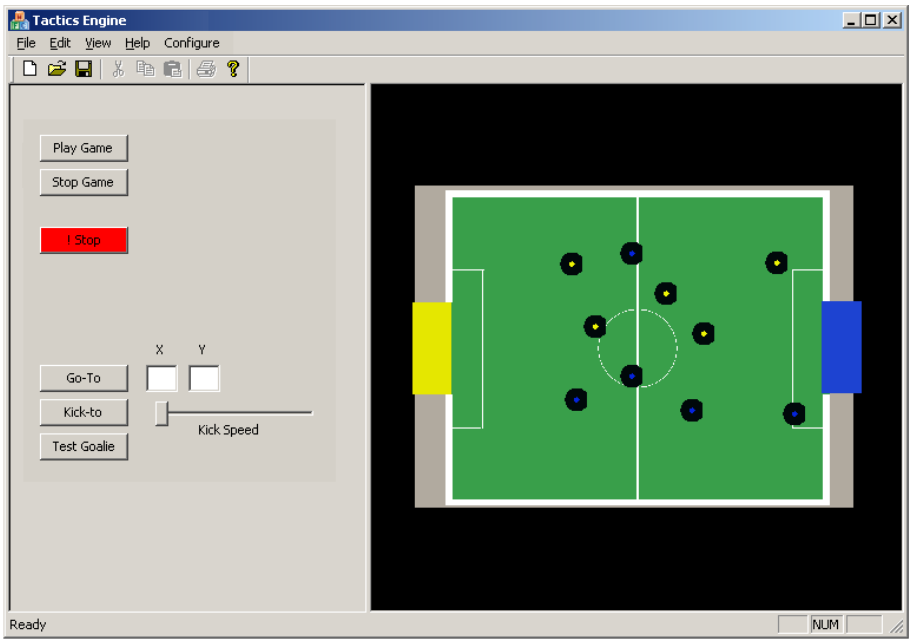

Figure 3: Tactics Engine UI Mockup

10.2 Reliability Requirements

The RoboCup system needs to address the following reliability issues:

- Be able to recover from a crash by restarting.
- To be tolerant of changing light conditions:
  - Gradual, minor changes in light conditions, such as changes in daylight
  - Brief, large changes in light, such as from a flash of a camera
- To be tolerant of electrical noise:
  - Fluorescent lights
  - TAFE robots during open day
- Tolerant of impacts during both game play and transportation
- No memory leakage of the data in the system
- Keeping log of all recent information, which is of particular importance before a crash
- Minimal CPU usage footprint of all critical modules, including Tactics Engine and Image Processing

10.3 Security Requirements

No security requirements apply to the 2004 RoboCup project.

10.4 Maintainability and Extensibility Requirements

The following Maintainability and Extensibility requirements are applicable to the 2004 RoboCup project:

- Documentation of final products, and the steps taken to achieve these, must be clearly specified.
- The software architecture will be such that the software components are broken-up into libraries/modules. This is to promote understanding, reuse and testing.
- Document not only the software but also the robotics components of the project. This will aid understanding due to the influence the robotics components have on the software design.
- Focus on tool-based development; this shall make using the software (and its reuse) easier.
- Focus on wizard-based code generation techniques, the result of which is to make code changes quicker and easier.
- Automate build and test procedures. This will help shorten development life-cycles and encourage team members to build and test more often.

10.5 Portability Requirements

No portability requirements apply to the 2004 RoboCup project.

11 Use Cases

The Use Cases described below are the scenarios specific to the basic functional requirements of the 2004 RoboCup robots, as specified by the client.

11.1 Go to location

The most important case is getting the robot to 'GO TO' the correct location, handled as follows:

1. Image Processing sends the Tactics Engine all of the relevant field information.
2. Tactics Engine uses the information to decide on the best path to get the robot to its desired location.
3. Tactics Engine sends the relevant commands to the RF module.
4. RF module sends the commands to the robot.
5. Robot receives the instructions and converts them to desired wheel velocities.

11.2 Kick to location

Kicking to a specific location, which is important for kicking goals and passing, is handled by the system as follows:

1. Embedded software is told by the specific sensing hardware that the ball is in the correct kicking location.
2. The Embedded software sends the relevant instructions to the RF module.
3. The RF module sends the instruction to the Tactics Engine.
4. Tactics Engine confirms via the Image Processing that the ball is in possession and decides on the best location to kick the ball.

5. Tactics Engine sends the relevant command to the RF module.
6. RF module sends the instructions to the robot.
7. Robot receives the instructions and converts them to desired wheel velocities to set its kicking path, and kicks the ball accordingly.

11.3 Goal-Keeper defends the goal

The following steps are the ones taken by the systems to allow the goal-keeper to defend the goal from an opposition shot:

1. Image processing detects the position of the ball and passes the images on to the Tactics Engine.
2. Tactics Engine establishes that the ball is travelling toward the goal by analysing the frames.
3. The Tactics Engine sends movement instructions to the RF module, specifically so that the robot moves into the ball's trajectory.
4. RF module sends the instructions to the robot.
5. Robot receives the instructions and converts them to desired wheel velocities.
6. Robot moves into ball's trajectory and stops the ball.

11.4 Defence

The Defence tactic of the system is handled as follows:

1. Image Processing sends the Tactics Engine the relevant field information.
2. Tactics Engine establishes that the friendly team needs to be in a defensive position (generally, when the opposition has the ball).
3. Tactics Engine selects 'Defence' as its objective.
4. Tactics Engine determines individual robot behaviours to meet objective.
5. Tactics Engine outputs to the RF the robot commands as decided above.
6. The RF sends the robot commands to the individual robots.
7. Robots receive the commands and convert them to actions.

11.5 Attack

The Attacking tactic of the system is handled as follows:

1. Image Processing sends the Tactics Engine all relevant field information.
2. Tactics Engine establishes that the friendly team is in an attacking position (perhaps one of the team members has the ball).
3. Tactics Engine selects 'Attack' as its objective.
4. Tactics Engine determines individual robot behaviours to meet objective.
5. Tactics Engine outputs to the RF the robot commands as decided above.
6. The RF sends the robot commands to the individual robots.
7. Robots receive the commands and convert them to actions.

Appendices

A Relationship Between Product and Environment

The environment which is relevant to the system is all that is encompassed by the field structure — that is: the green carpet that forms the field surface, the white lines, the boundary walls, the team's robots, the opposition's robots, the ball, the goals and the lighting. Changes in any of this environment are input into the system via the image processing. This data is transformed into tactics decisions via the Tactics Engine, which outputs instructions via the RF to the robots. The robots then change the environment according to these instructions. This process is shown in the context diagram in Figure 4 below.

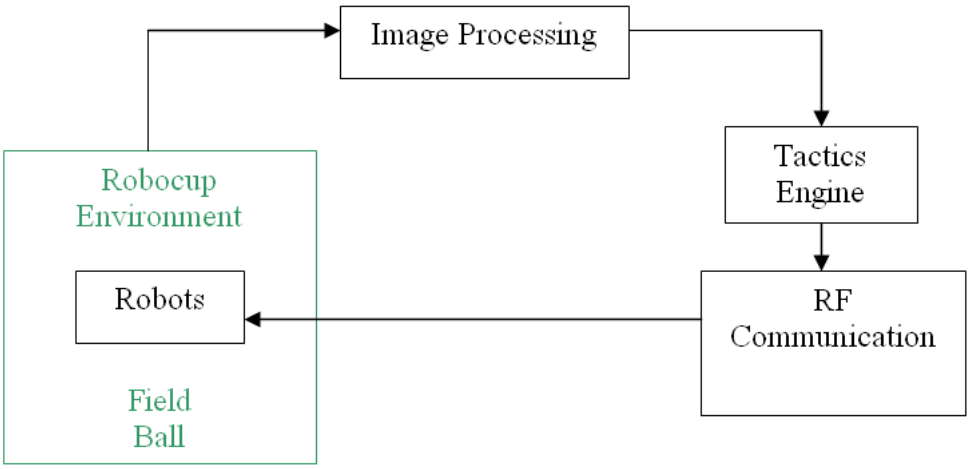

Figure 4: System Environment Interaction Diagram

B Data Flow Diagrams

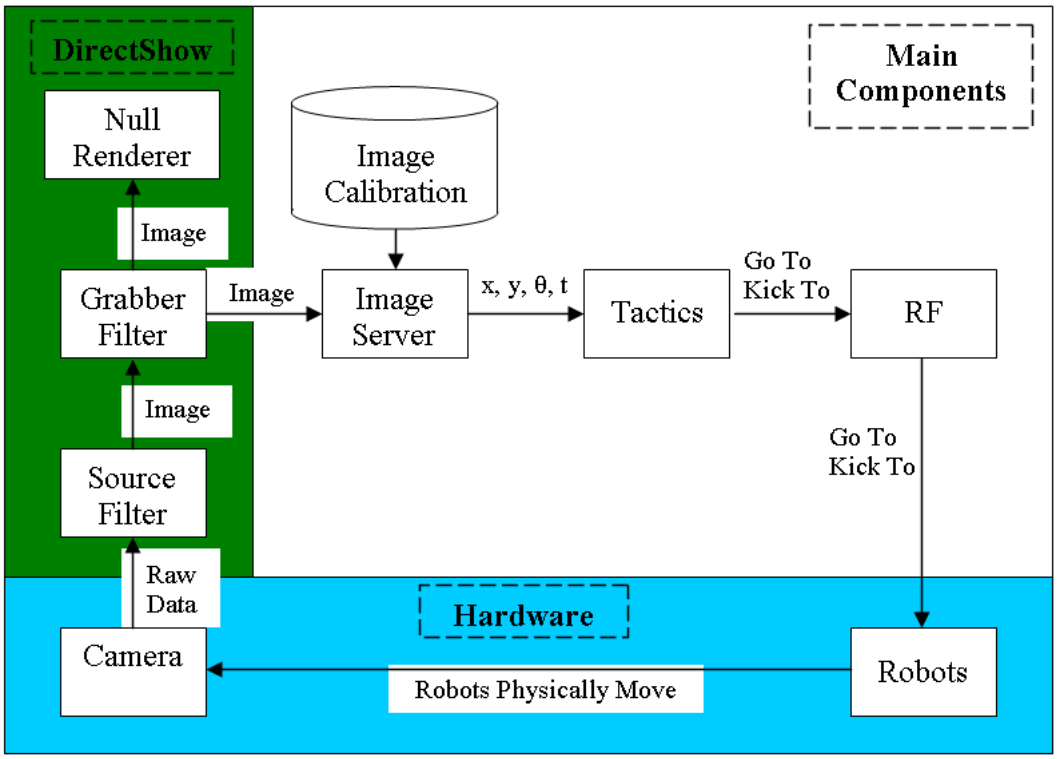

Figure 5: Whole System Data Flow Diagram

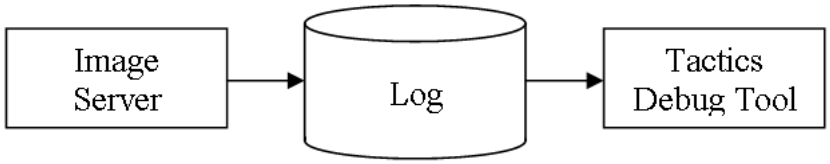

Figure 6: Tactics Debug Tool Data Flow Diagram

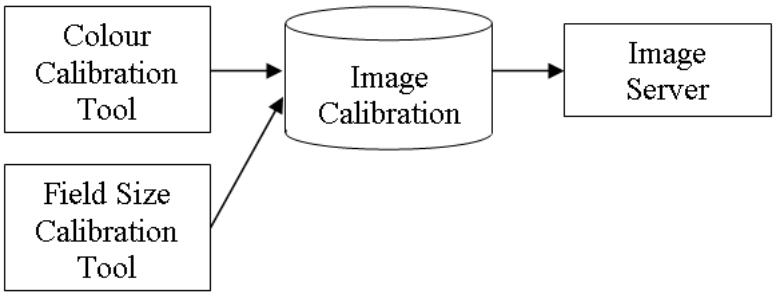

Figure 7: Image Calibration Data Flow Diagram

C Required "After Release" Changes

This section outlines the required changes to the SRS as a result of changes in specification throughout the project. The body of the SRS remains the same as that specified before design and coding began. This section outlines the changes we would make with the benefit of hindsight.

Tactics Engine

It was very difficult to specify the requirements of the tactics engine module. Since many of the functions relate to the type of implementation chosen (and therefore the software design), and these decisions had not been made at the time of intitial creation of the SRS, the task proved difficult. The following section outlines the changes that would be made should the SRS be re-written.

- A function to update the robot of its current position is required. This will be very similar to the "move" function, only the tag on the packet will be different, to indicate to the robot that the coordinates contained within the packet represent the current position, rather than the desired position.
- The pass function looks to be too difficult at this stage and as such, will not be implemented explicitly (a kick which is not aimed at the goal may, in effect, serve as a pass).
- The "allocate role" function will be more dynamic than that outlined within the SRS. Since the scope of the project has been changed to play three-on-three games rather than five-on-five, the two non-goalkeeper robots will change their roles dynamically. During attack, one robot will be an aggressive attacker, whilst the other will be a passive attacker. The same applies in defensive mode - one robot will be aggressive, whilst the other will be passive. So the "allocation of roles" function is still required, only it will be ran many times during a game, depending on the game state.
- A function to turn the roller on is required
- A function to turn the roller off is required

Tactics Debug Tool

The functional requirements specified for the Tactics Debug Tool are still relevant, however more functions are required to fully describe the component. These are as follows:

- Select robot for display
- Select map component for display (eg Line-of-sight map, ball map)

- Select "white" display for highest potential (change the colour of the grid element with the highest potential to white)
- Select map-type for display (eg ball-in-dispute map, defensive map, etc.)
- Drag robot to new position
- Refresh map according to movement of robot

Embedded Software

Changes to section 5.4

It was determined that no robot reset function is required. The robot may be started and stopped by sending consecutive STOP commands, but if a reset is required, it is simple enough to power-cycle the robot hardware.

Changes to section 5.5

After prolonged discussion, the 2004 team decided that it was no longer viable for the Tactics Engine to send required velocities to robots. Instead, the Tactics Engine is to send desired positions (absolute x and y co-ordinates, as well as orientation angles), and the embedded software is to determine appropriate wheel velocities using inverse kinematics — and position PID control — in order to get the robot to that field position. Hence, the function name is now ROBOT_POSITION and will process the data the same way as specified in section 5.5, except it will apply inverse kinematics to the position and angle to determine the required wheel speeds.

Changes to section 5.9

The requirement states that this function turns off the roller before kicking the ball. It was found that more consistent and accurate kicks are possible by leaving the roller running during kicks; therefore, this function will not be required to turn off the roller.

Changes to section 5.10

Passing the ball proved to be way out of scope for the intelligence system this year. Due to this, there is no longer a requirement for a robot ball-passing function.

Changes to section 5.11

The team has been unable to acquire appropriate collision detecting sensors (e.g. accelerometers) in time for Open Day. Without these sensors, there is little point in having (and no ability to test) a collision detection function.

Changes to section 5.12

Similarly, without collision detection, there is no requirement to implement a collision response function.