The Pennsylvania State University

The Graduate School

College of Engineering

A DYNAMIC ACTION SELECTION MECHANISM FOR AN AUTONOMOUS AGENT WITH A MODEL FOR ITS EMOTIONAL STATE

A Thesis in

Aerospace Engineering

by

Vidullan Surendran

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

December 2015

The thesis of Vidullan Surendran was reviewed and approved* by the following:

*Signatures are on file in the Graduate School

ABSTRACT

A goal based dynamic action selection mechanism incorporating a model for emotions was developed for use with autonomous agents. An autonomous agent was developed to test the action selection mechanism by recreating the scenario of an animal foraging for food while avoiding predators. Four emotions of anger, fear, happiness, and surprise were modelled which were affected by events such as finding food, encountering a predator, encountering a boundary wall, finding a safe area, and being in a state of low health. The model incorporated a reward prediction module that altered the effect of an event based on the error between when an event occurred and when it was predicted to occur. The model also included a decay term that resulted in the emotions returning to their steady state values unless there was continual reinforcement through the occurrence of events. Four different temperaments referred to as irate, timid, cheerful, and anxious were used to study the effect of differing temperaments on the emotions.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

COTS - Commercial Off The Shelf

MHz - Megahertz

KB - Kilobytes

PWM - Pulse Width Modulation

mA - Milliamps

USB - Universal Serial Bus

I^2C - Inter-Integrated Circuit

SPI - Serial Peripheral Interface

V - Voltage

A - Amp

mm - Millimeter

SoC - System on a Chip

MB - Megabyte

GB - Gigabyte

SD - Secure Digital

IMU - Inertial Measurement Unit

s - Second

ADC - Analog to Digital Converter

LED - Light Emitting Diode

KHz - Kilohertz

IC - Integrated Circuit

NiMH - Nickel Metal Hydride

IR - Infrared

fps - Frames per Second

UART - Universal Asynchronous Receiver Transmitter

GHz - Gigahertz

Mbps - Megabits per second

DCM - Direction Cosine Matrix

m/s^2 - Meter per second squared

ASM - Action Selection Mechanism

RPE - Reward Prediction Error

ACKNOWLEDGEMENTS

I would like to express my gratitude to Professor Long for his encouragement and guidance throughout my graduate studies. I am especially thankful for his faith in me even when progress was slow and, pushing me to develop my research and teaching abilities.

I am particularly grateful for the assistance provided by Mr. Richard Ahul with fabricating hardware and his help dealing with logistical issues as promptly as possible. I would like to thank Professor Melton for his valuable feedback on the report and his infectious enthusiasm. I would also like to thank Professor Lesieutre for the teaching opportunities the department has provided. My special thanks are extended to all the staff in the aerospace department for their patience and invaluable help during my time at the university.

I wish to acknowledge Kaushik Basu for his help troubleshooting electrical issues and Ripudaman Ripudaman for his help verifying the mathematical models used in this work.

Finally, none of this would have been possible without the unwavering love and support of my family and friends.

Chapter 1
Introduction

Affective computing is an exciting new field of study that seeks to understand and develop systems that can recognize and simulate human emotions. R.W. Picard [1] highlighted the importance of the field by explored neurological studies that indicated human cognition was intrinsically linked with emotions. She also argues that that the development of affective computing is critical to advancing emotion and cognition theory.

Breazeal and Brooks [2] make an argument for the inclusion of emotions into robotics saying that in the future, robots might be able to interact with humans on an emotional level and will be commonplace in our households. They considered cognition and emotions to be two distinct systems that evolved in intelligent creatures under social and environmental processes to aid in optimal functioning. Cognition is deemed to be responsible for interpreting the world whereas emotions are deemed to be responsible for evaluating the value of events. Emotions thus help prioritize concerns while minimizing distractions. They argue that emotion-inspired mechanisms would help robots function better in complex, unpredictable environments by modulating cognition.

Kismit, a sociable humanoid robot demonstrates emotional intelligence by being able to engage people in expressive face to face interactions. It is able to perceive social cues from visual and auditory input, and respond through facial expressions, body posture, gaze direction and vocals [3]. Valery Karpov [4] demonstrated that complex psychological phenomena implicated in the generation of emotions and temperaments in humans can be imitated by simple means. The author used Simonov's information theory of emotions to implement an autonomous robotic architecture. The robots used in the study were experimentally shown to exhibit distinct types of temperaments such as melancholic, choleric, sanguine, and phlegmatic.

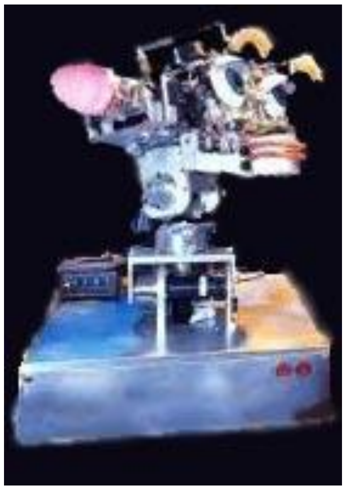


Figure 1-1 Kismit, a humanoid robot developed by the sociable machines project

The simulation of human emotions is greatly complicated by the fact that there is no accepted model that explains and predicts the wide range of emotions we experience. There still is no census in the literature on the number of base emotions. Paul Ekman [5] proposes a list of 15 emotions, each representing a family of emotions. A study on dynamic facial expressions of emotion by Jack et al. [6] challenges this notion by suggesting that basic emotion communication comprises fewer categories. It is clear that our understanding of the subject is still in its infancy.

A study on momentary subjective wellbeing by Rutledge et al. [7] resulted in a computational and neural model of happiness referred to as the 'Happiness Equation' in popular culture. They showed that momentary happiness in response to a probabilistic reward is explained by the combined influence of the reward expectations and the prediction errors from the expectations; not by current task earnings as one would naively conclude. Long et al. [8], [9] adapted this model to simulate the six 'universal' emotions of fear, anger, sadness, happiness, disgust, and surprise in cognitive mobile robots. The emotion and temperament engine they developed was incorporated into SS-RICS which is a cognitive architecture developed at the Army Research Laboratory [8].

This study continues in this vein by considering the integration of an emotions engine with an action selection mechanism suitable for an autonomous agent. In recent times there has been a shift towards distributed, mobile, and small form factor computing spurred on by the reducing cost of electronics. There was found to be a tenfold increase in the number of instructions per second that can be achieved per dollar every 5.6 years [10]. The widespread availability of low cost computing prompted a focus on developing a system that could be run with limited computational resources.

Chapter 2
Hardware

The hardware that makes up the system used here can be divided into components used in the robot, and those in the command center used to remotely control it. A commercial off the shelf (COTS) approach to the project was preferred to enable multiple robots to be built efficiently in terms of cost and time. A small mobile robotics platform was also preferred so it could be used in future robot swarm research.

1. Robot Platform - Dexter

The robotic platform developed here is called Dexter and the capabilities required of it were:

1. A steerable platform
2. Space for installation of required sensors and computing modules
3. Small form factor defined as having a footprint under 400 cm^2
4. Imaging capabilities
5. Analog inputs to handle inputs from at least 4 sensors
6. Digital communication protocol - I2C, SPI
7. Battery operation for a maximum of an hour
8. Wireless communication

To accomplish these goals, a variety of solutions were assessed and ones that were cost effective and easily available were selected.

A. Chassis

The DFRobot 2WD mobile platform for Arduino shown in Figure 2-1 and Figure 2-2 was purchased from Robotshop.com and is used as the chassis for the robot. It is designed for use with an Arduino microcontroller board and is controlled by two differential drive motors. A caster ball wheel provides a tri-cycle wheel configuration adding stability as shown in Figure 2-1. The motors included were unmarked, but judging by their electrical (5 volts (V) and 1 ampere (A) Max) and physical characteristics are most likely 180-size brushed motors. A gearbox is used to increase motor torque [11].

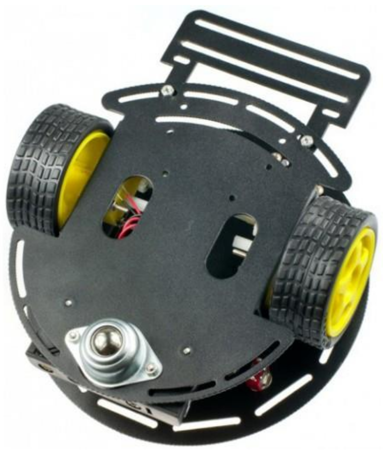


Figure 2-1 Underside view showing tricycle configuration [11]

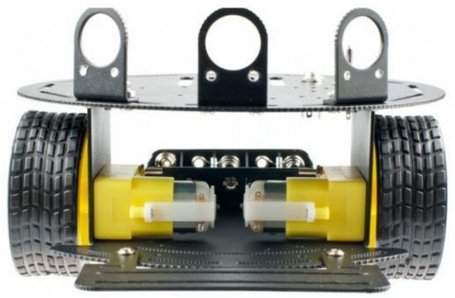


Figure 2-2 Front view of DFRobot 2WD platform [11]

B. Microcontroller

The Arduino Uno Rev. 3 based on the ATmega328 clocked at 16 megahertz (MHz) was chosen as it is compatible with the chassis used. It has 2 kilobytes (KB) of static random access memory, 32 KB of flash memory and 1 KB of electrically erasable programmable read-only memory [12].

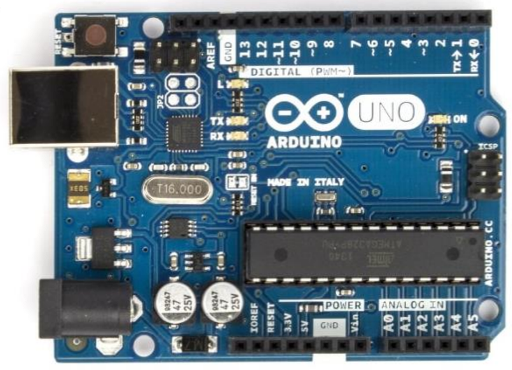


Figure 2-3 Arduino Uno Rev.3 [12]

The Uno has 14 digital input/output (I/O) pins along with 6 analog inputs. Of the 14 I/O pins, 6 provide pulse width modulated (PWM) outputs. Each pin is rated to source or sink current at 20 milliamperes (mA). It also provides universal serial bus (USB) communication through a serial bus, inter-integrated circuit (I^2C), and serial peripheral interface (SPI) capabilities. Battery operation is possible with a recommended input voltage range of 7 V to 12 V, and an operating voltage of 5 V.

The advantage of using this board was the large installed user base, support, and a plethora of libraries available on the internet. Software repositories such as GitHub located at https://github.com/ and the Arduino Playground located at http://playground.arduino.cc/ contain libraries, example code, user projects, tutorials, and other educational material that help one get familiarized with the platform.

C. Motor Controller

Due to electrical limitations, the Uno's pins cannot directly drive motors and an external solution was needed. The Arduino Motor Shield [13] is a two channel Arduino pin compatible driver board which allows the output pins of the Uno to drive inductive loads.

It is based on the L298 dual full-bridge driver and allows a maximum output current of 2 A per channel. The board requires power from an external source capable of providing at least 4 A and has an operating voltage range of 5 V to 12 V.

This extension board is connected to the Uno via male header pins facilitating communication between them. Motors or other loads are connected to the screw terminals of the board and the voltage supplied to them can be linearly varied using PWM signals generated by the Uno.

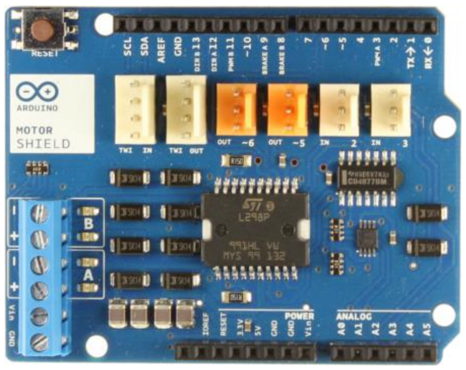


Figure 2-4 Arduino Motor Shield [13]

a. Modification of Pinout

The Motor Shield was originally designed to be pin compatible with the Uno's I/O pins as a plug n' play extension board. Certain I/O pins which have dual functionality as SPI pins are used as digital inputs by the motor shield. Since SPI functionality was required, the Motor Shield was rewired to use other digital I/O pins freeing up the SPI pins. This is documented in Table 2-1.

Table 2-1 Rerouted pins of motor controller

Function	Channel	Original Pin	Rerouted Pin
Direction	Channel A	D12	A4
	Channel B	D13	A5
PWM	Channel B	D11	5

D. Function Assigned to Arduino I/O Pins

Table 2-2 documents the I/O pins in use on the Uno.

Table 2-2 Function assigned to I/O pins

Pin	Function
0	Serial RX
1	Serial TX
3	Motor Shield A PWM
5	Motor Shield B PWM
8	Motor Shield B Brake
9	Motor Shield A Brake
10	SPI Slave Select
11	SPI MOSI
12	SPI MISO
13	SPI SCK
A0	Motor Shield A Current Sensing
A1	Motor Shield B Current Sensing
A4	Motor Shield A Direction
A5	Motor Shield B Direction

E. Micro Computer

The Uno is not capable of wireless communication or image processing unless expansion boards are used. The cost and space efficient solution was to use a computing module such as the Raspberry Pi whose dimensions are 85 millimeters (mm) by 56 mm [14].

The Raspberry Pi is a system on a chip (SoC) running an ARM1176 processor capable of operating at a clock speed of up to 1000 MHz. The SoC includes a VideoCore IV graphics processing unit, 16 KB of level 1 cache, 128 KB of level 2 cache and 256 megabytes (MB) of random access memory. It runs Raspibian which is a Linux based operating system based on Debian. Its power consumption is approximately 3.5 Watts which makes it suitable for mobile operation [14].

Figure 2-5 Raspberry Pi Model B [15]

The operating system is run from a 2 gigabyte (GB) secure digital (SD) card and a USB dongle is used for wireless connectivity. A USB hub is used to increase the number of available ports to five allowing the connection of a keyboard and mouse.

F. USB 2.0 Hub

The number of USB ports on the Raspberry Pi is expanded using a powered 4 port USB 2.0 hub.

Figure 2-6 4 port USB 2.0 hub

G. Inertial Measurement Unit (IMU)

The MinIMU-9 v3 produced by Pololu is used to keep track of the robot's absolute orientation to aid in navigation and maneuvering. This IMU board contains a 3-axis gyroscope, 3-axis accelerometer, and a 3-axis magnetometer. Its operating supply range is 2.5 V to 5.5 V at 10 mA, and it uses I^{2}C to facilitate two way communications with the board.

Figure 2-7 Pololu MinIMU-9 v3 [16]

The accelerometer and magnetometer provide one 12-bit reading per axis while the gyroscope provides a 16-bit reading per axis. It also provides adjustable sensitivities and for this project a sensitivity of ±250° per second (s), ±2 g, and ± 1.3 gauss were used for the gyroscope, accelerometer, and magnetometer respectively [16].

H. Analog to Digital Converter (ADC)

While the Arduino does provide 5 analog inputs using a built-in 8-bit ADC, only a sampling rate of approximately 10 kilohertz (KHz) is achievable. To obtain audio at a minimum of 4 KHz for sound localization, the Nyquist-Shannon sampling theorem [17] dictates a sampling frequency of 8 KHz per mic. This necessitated a total sampling rate of at least 16 KHz.

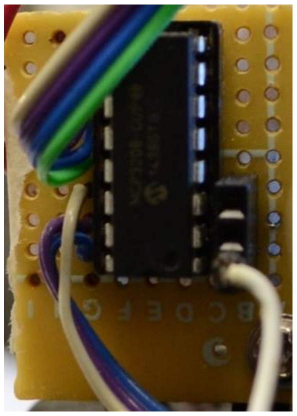


Figure 2-8 MCP3008 ADC

The MCP3008 is a 5 V analog to digital converter integrated circuit (IC) with 8 input channels. It allows a maximum sampling rate of 200 KHz at 10-bits of precision and was ideal for this project [18]. Communication with the IC is achieved using the SPI communication protocol.

I. Matrix Display

An 8x8 light emitting diode (LED) matrix driven using a MAX7219 common cathode display driver is used to display messages and data values. The schematic and bill of materials of the fabricated circuit board is provided in the Appendix on page 89. The board is powered with 5 V and data is sent to it using SPI.

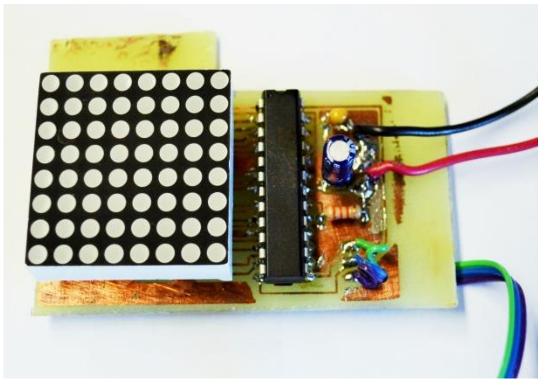


Figure 2-9 LED display along with driver circuit

J. Battery

To obtain an approximate current requirement, the average current draws of the Raspberry Pi and Uno were taken under maximal processing load. The motors were assumed to run continuously for an hour at 75% of their maximum power rating. The Web Cam was limited to a maximum of 500mA as it was powered by USB. The rest of the components had current draws that totaled to less than 50mA and were not separately considered.

Table 2-3 Current draw by component

Component	Current Requirement
Raspberry Pi Model B	700mA
Arduino Uno	300mA
Motor Shield	1500 mA
Web Cam	500 mA
Other components	50 mA
Total	3050 mA

Figure 2-10 5 cell Ni-MH battery pack

Based on the current draw in a worst-case scenario, a custom 5-cell 6.0V battery pack made up of Elite SC 3300 milliamp hour nickel-metal hydride (NiMH) cells was selected to give a run time of at least one hour. Under typical conditions this battery pack was found to permit operation for about an hour and a half.

K. Step-Down Voltage Regulator

The battery pack used provides 6 V when fully charged as mentioned in Chapter 2.1.J. Due to the pack voltage and peak voltage drop with discharge, a constant 5 V supply cannot be guaranteed. To obtain a regulated supply, a switching step-down voltage regulator is used. It is rated at 3.5 A, has an input voltage range of 4.5 V to 24 V, and provides a regulated 5 V or 3.3 V output [19].

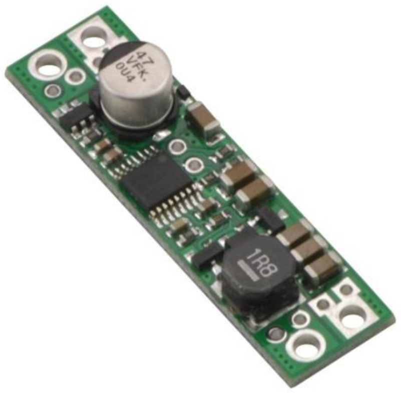


Figure 2-11 Pololu D15V35F5S3 [19]

L. Infrared (IR) Proximity Sensor

In order to aid in collision avoidance and to compute distances to objects, a Sharp GP2Yx IR sensor is used. This is an analog sensor with an output non-linearly varying from 3.1 V at 10 cm to 0.4 V at 80 cm [20]. A lookup table correlating distance values to sensor output was experimentally obtained and is used to extrapolate distance values.

Figure 2-12 Sharp GP2Y0A21YK [20]

M. Webcam

A Logitech C170 USB webcam is used to provide images for the vision system. It provides an optical resolution of 640x480 at 1.3 Megapixels and allows a maximum of 30 frames per second (fps) [21].

Figure 2-13 Logitech C170 [21]

2. Hardware Communication

The protocol used for communication between the various components is detailed here. A block diagram of the overall system is shown in Figure 2-15.

Arduino Uno and Motor Shield

The motor shield requires a PWM signal to set the speed of the motor along with two digital signals; one to set the direction and one to toggle the brake for each motor channel. The relevant pins were connected as listed in Table 2-2.

Analog Sensors

The analog signals generated by the sensors are fed to the input channels of the MCP3008 ADC.

MCP3008 and Arduino Uno

SPI is used to allow these two devices to communicate and the relevant pins of the MCP3008 were connected to the Uno as listed in Table 2-2.

Raspberry Pi and IMU

The IMU is connected as a slave device to the I^2C bus on the Raspberry Pi.

Raspberry Pi and LED Matrix

The MAX7219 is connected to the CE0 chip select pin and SPI pins on the Raspberry Pi.

Raspberry Pi and Arduino Uno

The hardware universal asynchronous receiver/transmitter (UART) on the Uno is used to send serial data to the Raspberry Pi over a USB connection. A protocol was developed to enable transfer of data between the two devices with the Raspberry Pi acting as the master device, and the Uno acting as the slave device. All commands sent are 12 bytes long which allowed the transmission of 6 unsigned short integers as they are defined to be 2 bytes long. In other words, each command consists of 6 numbers which are used to control the Uno.

Motor Control: In order to set the speed of the motors the 12 bytes of data are formatted in the manner shown in Figure 2-14. The first 6 bytes contain information for motor A and the last 6 for motor B. Let us consider the 6 bytes or 3 unsigned short integers sent about motor A. The first number controls the direction of motor A. If it is a ‘0’ the motor rotates clockwise and if it is a ‘1’ it rotates counter clockwise. The next number control the duty cycle of the PWM signal sent to motor A and has a range of 0 to 255. ‘0’ represents a 0% duty cycle whereas ‘255’ represents a 100% duty cycle. The third number switches on or off motor A’s electronic brake. If it is a ‘0’ the brake is disabled and if it is a ‘1’ it is enabled. Thus using 6 numbers, we gain complete control over both motors.

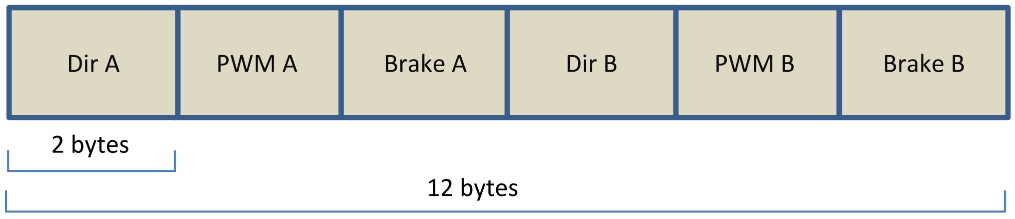


Figure 2-14 Composition of the command sent from the Raspberry Pi to the Uno

Note: motor A and motor B have been wired with reversed polarities such that while ‘0’ causes motor A to rotate clockwise, ‘0’ causes motor B to rotate counter clockwise.

For example, say we have to move the robot forwards at 50% of its speed. To do this we need both motors to rotate in the opposite direction and a PWM duty cycle of 50%. We would send the following numbers: ‘1’, ‘50’, ‘0’, ‘1’, ‘50’, and ‘0’. This would cause motor A to rotate at 50% of its speed counterclockwise and motor B to rotate at 50% of its speed in the clockwise direction cause the robot to move forward.

Consider another case where we have to turn the robot at 75% of the speed. To do this we need one motor to operate while the other is stopped. We would send the following numbers: ‘0’, ‘0’, ‘1’, ‘1’, ‘179’, and ‘0’. This will cause motor A to be stopped by the electronic brake and motor B to rotate at 75% of its speed in the clockwise direction causing the robot to turn.

IR Readings: In order to obtain IR sensor readings, the Raspberry Pi sends six numbers as seen in the motor control case but the number for Dir A is set to ‘9’. The other 5 numbers are arbitrary. As the direction can only be ‘0’ or ‘1’, the number ‘9’ instructs the Uno to poll the IR sensor and send the 16-bit reading obtained to the Raspberry Pi.

3. Block Diagram of the Robot

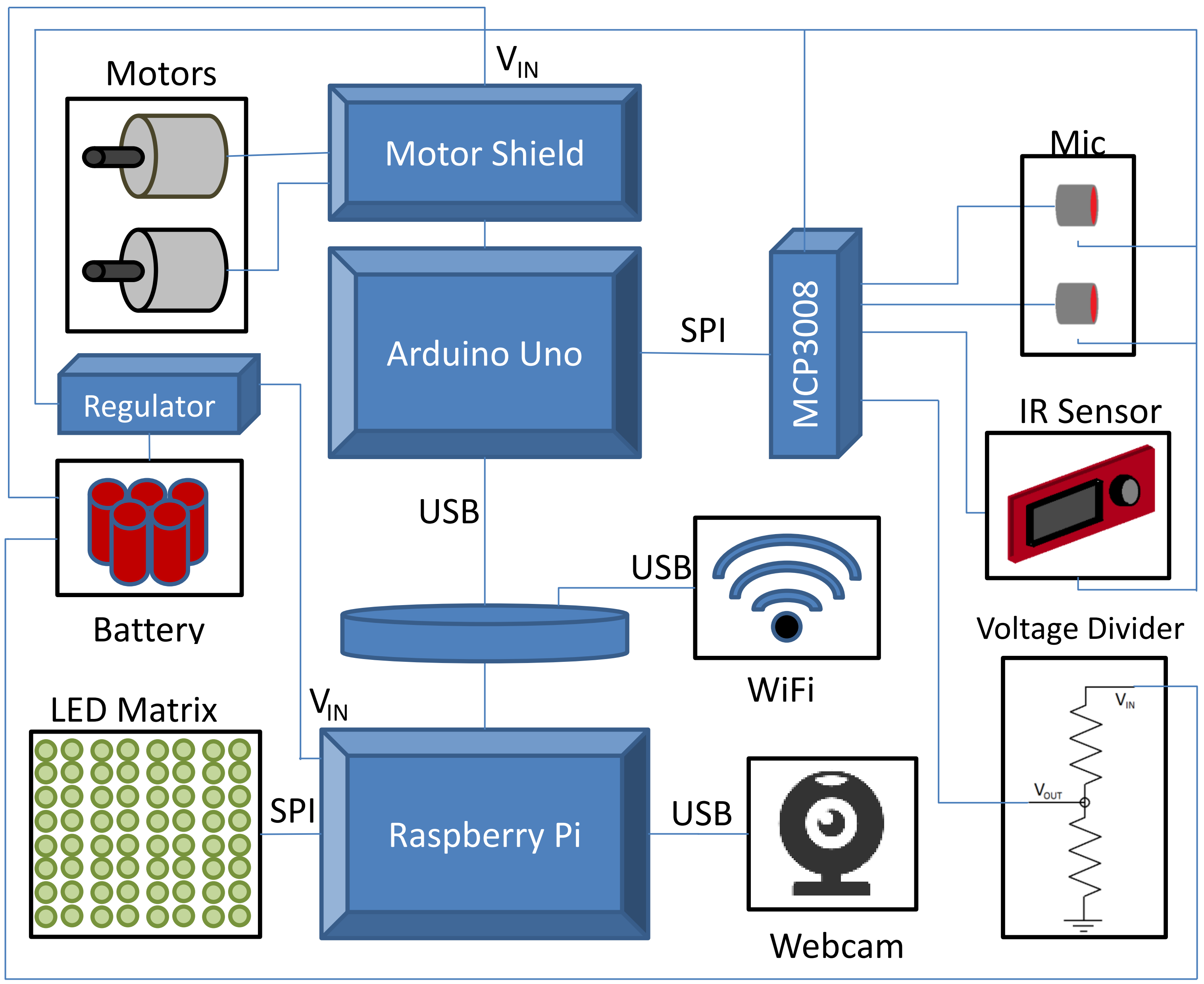


Figure 2-15 Block diagram of Dexter showing hardware connections and communication protocols used

4. Complete Robot

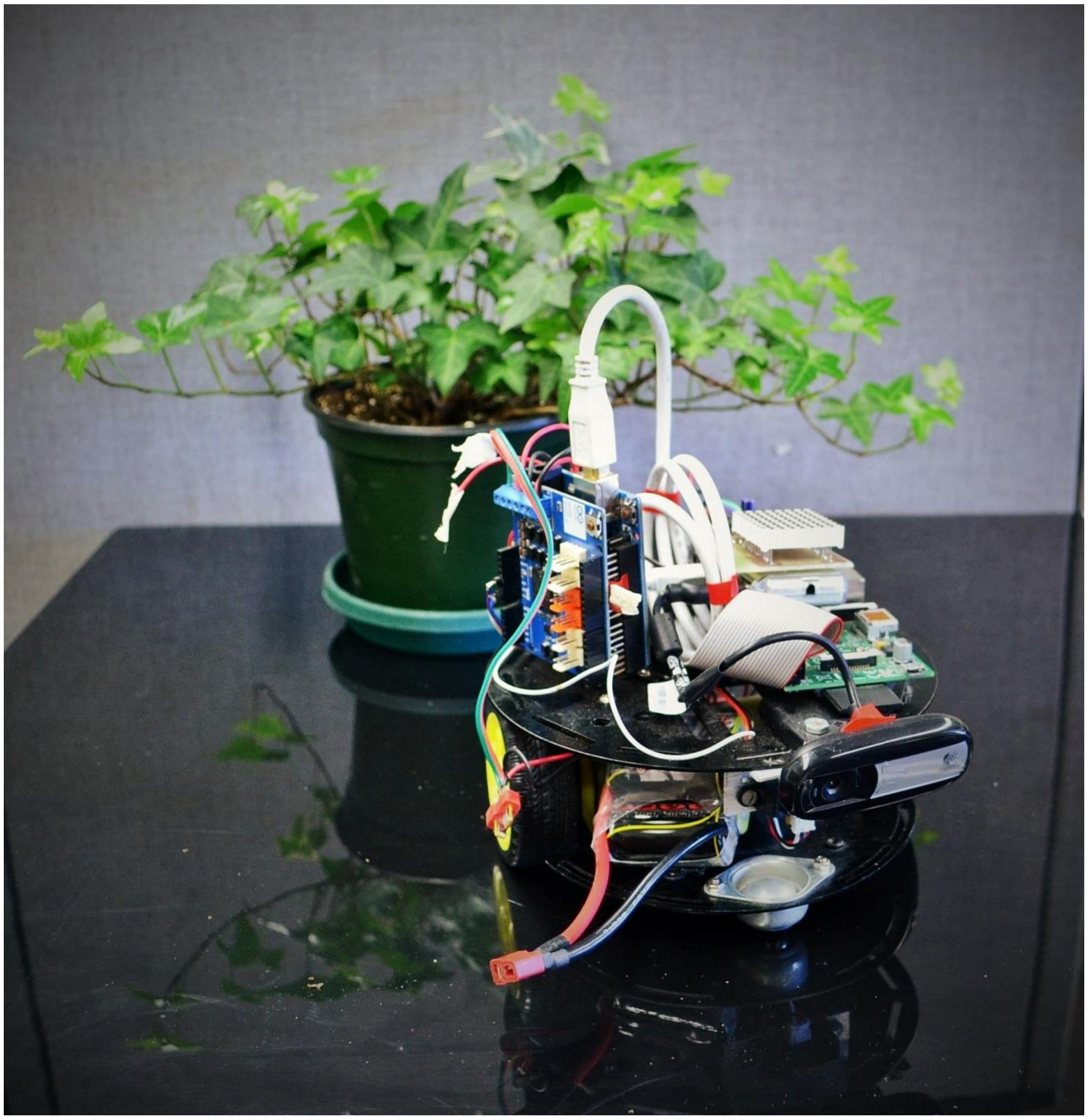


Figure 2-16 Dexter in its natural habitat

5. Command Center

The command center is made up of a computational node and a network. It receives data from the robot, processes it, and transmits control commands to it.

A. Hardware

a. Router

A wireless network was setup using a NETGEAR N150 router operating in the 2.4 gigahertz (GHz) band. The router can sustain a signal rate of up to 150 megabits per second (Mbps) [22]. No encryption is used on the network as sensitive data is not expected to be transmitted. This eliminated dropped connections that were frequently experienced when encryption was used.

Figure 2-17 NETGEAR N150 [22]

b. Computational Node

An IBM ThinkPad T30 which is a 14.1" laptop with a single core 2.4 GHz Pentium 4 processor and 1 GB of RAM is used [23]. The operating system used is Debian Wheezy (Linux Kernel 3.2). Connection to the router is established using an Ethernet cable.

Figure 2-18 ThinkPad T30 [23]

B. Communication with the Robot

The command center exchanges data with the robot over the wireless network using TCP/IP. The command center acts as the server and listens for connections while the robot acts as the client and connects to the server. Once connected together data is exchanged using the two user defined data structures ‘RobotParam’ and ‘BallData’ described in Chapter 6.2. To aid in synchronization of transfers, a protocol was implemented that allowed the robot to communicate whether it was ready to receive or transmit data. Table 2-4 lists the integers that the robot transmits along with what they represent.

Table 2-4 Protocol used to communicate with the Raspberry Pi

Integer	Description
1	Ready to send data to command center
3	Ready to receive commands from command center

The protocol operates in the following manner:

Command Center: The command center constantly listens for an integer from the robot. If it receives the integer ‘1’, it then prepares to receive the data structures ‘RobotParam’ and ‘BallData’. If it receives the integer ‘3’, it sends out the structure ‘RobotParam’ containing the control commands.

Robot: When the robot has gathered all data to be sent to the command center, it sends out the integer ‘1’ followed by the data using the structures ‘RobotParam’ and ‘BallData’. Then it sends out the integer ‘3’ and prepares to receive control commands.

Chapter 3
Problem Definition

The goal in this study was the implementation of an autonomous system that could recreate the behavior and emotional state changes of a small animal foraging for food. Immediately it becomes apparent that constraints have to be placed on this to make it a feasible exercise.

One of the most primal needs of an animal is food. An animal needs to forage for food to satisfy its energy needs and seeks to do so while avoiding predators. It would then presumably try to return to a safe area. If we consider this case, the tasks the robot has to accomplish is reduced to:

- Being able to search an area for food sources
- Avoiding predators
- Returning to a safe area

Now we consider the representation of emotions. The system should:

- Quantify emotions mathematically
- Provide means to track the values of the emotions
- Define situations in which the emotional state would be altered
- Keep track of such situations and update the emotional state
- Alter the behavior of the robot in response to its emotional state

. To further simplify the problem we only consider the four emotions of anger, fear, happiness, and surprise.

1. Test Setup

To recreate the scenario of a foraging animal, a certain type of object is assumed to represent food, a predator, and a safe area respectively. The objects used were differently colored foam balls having a diameter of approximately 10 cm. Table 3-1 lists what each colored ball represented.

Table 3-1 Concept represented by foam balls

Color	Represents
Green	**Food**
Red	**Predator**
Dark Purple	**Home/Safe Areas**

The robot was placed inside an enclosed space similar to the one shown in Figure 3-1 where balls representing safe areas, food, and danger were randomly arranged. One of the safe areas was randomly chosen as the starting position of the robot. It is also assumed that the health of the robot decays linearly with respect to time.

Searching for food would involve a search and scan of the area until green balls are identified. If one is found, the robot would track towards the ball till it got close enough to 'consume' it. This is simulated as a time dependent increase in the robot's health for as long as it stayed in close proximity to a food ball.

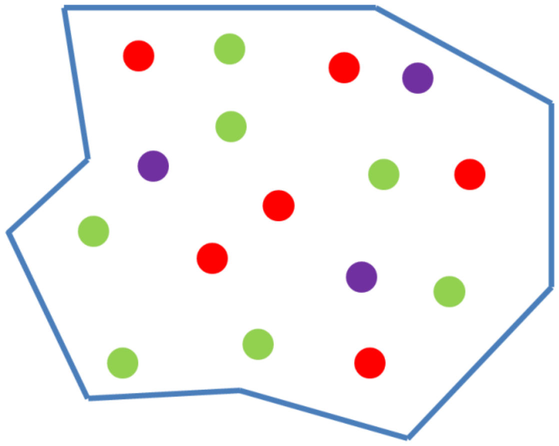


Figure 3-1 Operating Environment

At all times the robot would also have to scan for red balls. Whenever one is identified it would have to take evasive manoeuvers to avoid it. Only stationary predators were considered and the red balls do not change their position.

Once the robot recharged its health it would then seek out a safe area. This would involve a search and scan of the area similar to finding food. Once at a safe area, the robot does nothing until its health decays to a level that triggers a repeat of the cycle.

Chapter 4
Vision System

Completion of the tasks and navigation of the operating environment is highly dependent on the robot's ability to identify spherical objects. To this extent a vision system was developed focusing on the identification of spherical objects and differentiation based on color.

Thus we can define the requirements of the vision system as:

- Detection of spherical balls of a given diameter
- Differentiation based on color
- Deduction of the distance between a ball and the robot

1. OpenCV

In order to aid with image processing the OpenCV library is used as it provides a comprehensive set of methods written in C/C++ for such purposes. In addition to algorithms, it also provides data structures that can be used to store image data, methods to transform image data between color spaces, and routines to obtain images from hardware peripherals.

A. Modules Used

The following OpenCV modules are used in the image processing code:

core - defines basic data structures and basic functions used by all other modules

highgui - an interface to video capturing, image and video codecs, and user input capabilities

B. Installing OpenCV on the Raspberry Pi

Firstly, dependencies were installed using the following Linux commands [24]:

```
sudo apt-get -y install build-essential cmake cmake-curses-gui pkg-config libpng12-0
libpng12-dev libpng++-dev libpng3 libpnglite-dev zlib1g-dbg zlib1g zlib1g-dev pngtools
libtiff4-dev libtiff4 libtiffxx0c2 libtiff-tools libeigen3-dev
```

```
sudo apt-get -y install libjpeg8 libjpeg8-dev libjpeg8-dbg libjpeg-progs ffmpeg libavcodec-
dev libavcodec53 libavformat53 libavformat-dev libgstreamer0.10-0-dbg libgstreamer0.10-0
libgstreamer0.10-dev libxine1-ffmpeg libxine-dev libxine1-bin libunicap2 libunicap2-dev
swig libv4l-0 libv4l-dev python-numpy libpython2.6 python-dev python2.6-dev libgtk2.0-dev
```

Then the source code release of OpenCV 2.4.10 was downloaded from the sourceforge software repository found at: http://sourceforge.net/projects/opencvlibrary/files/opencv-unix/2.4.10/opencv-2.4.10.zip/download.

The source was then unpacked and the build was configured using CMake which is a build system generator. This allowed OpenCV to be customized for the hardware platform in use, in this case the Raspberry Pi. Commands that were used to do this were [24]:

```
unzip opencv-2.4.10.zip
cd opencv-2.4.10
mkdir release
cd release
ccmake ../
```

Once the required make files were generated, the project was built and installed using [24]:

```
make
sudo make install
```

C. Compiler Flags

Code using OpenCV, the core, and highgui modules must be linked to their respective libraries by passing the following linker options to the compiler:

```
-lopencv_highgui -lopencv_imgproc -lopencv_core
```

D. Functionality Used

This is a list of OpenCV structures and methods used in the image processing algorithm along with a brief description of their functionality.

a. Data Structures [25]

Mat - A class that represents an n-dimensional dense numerical array used usually to store image pixel data.

*Size_* - Template class used to specify the height and width of an image or rectangle

Point2f - Class used to store coordinates of 2D points

Point3f - Class used to store coordinates of 3D points

Vec4i - Class used for short numerical vectors of integers with 4 components

VideoCapture - Class used for video capture from files, image sequences or cameras

Scalar - Template class for a 4 component vector

b. Member Functions of Class VideoCapture [26]

open( ) - Used to open a specified device for video capturing

isOpened( ) - Used to check if object has been initialized

read( ) - Grabs, decodes, and returns the next video frame

set( ) - Sets the class property specified to an input value

c. Filtering Methods [27]

erode() - Performs a morphological erosion on an input image using the structuring element defined, and stores the output image in a specified container.

dilate( ) - Performs a morphological dilation operation on an input image using the structuring element defined, and stores the output image in a specified container.

d. Structural Analysis Methods [28]

findContours( ) - Finds contours in an input binary image and returns a vector of points describing each one.

approxPolyDP( ) - Approximates a polygonal curve with a specified precision

contourArea( ) - Returns the area enclosed by a contour described by a vector of points

minEnclosingCircle( ) - Finds the coordinates of the center and the radius of a circle of minimum area enclosing a 2D point set

e. User Interface Methods [29]

namedWindow( ) - Creates a window

imShow( ) - Displays an input image in the specified window

f. Miscellaneous Methods

cvtColor( ) - Takes an input image, converts it into the color space specified, and stores it in an image container. The input and output images were stored using objects of type Mat [30].

inRange( ) - Checks if the elements of an input array lie between the lower bounds and upper bounds specified. It stores the value ‘255’ if an element meets the condition, and ‘0’ otherwise in an output array of the same type and size as the input [31].

2. Imaging Algorithm

Figure 4-1 shows the major steps involved in processing each image. Processing a single frame for one color was found to take approximately 0.8 seconds. Since there were 3 differently colored balls to be identified, the vision system operated at an average of 4 fps. This low frame rate is a consequence of the limited computational power of the Raspberry Pi.

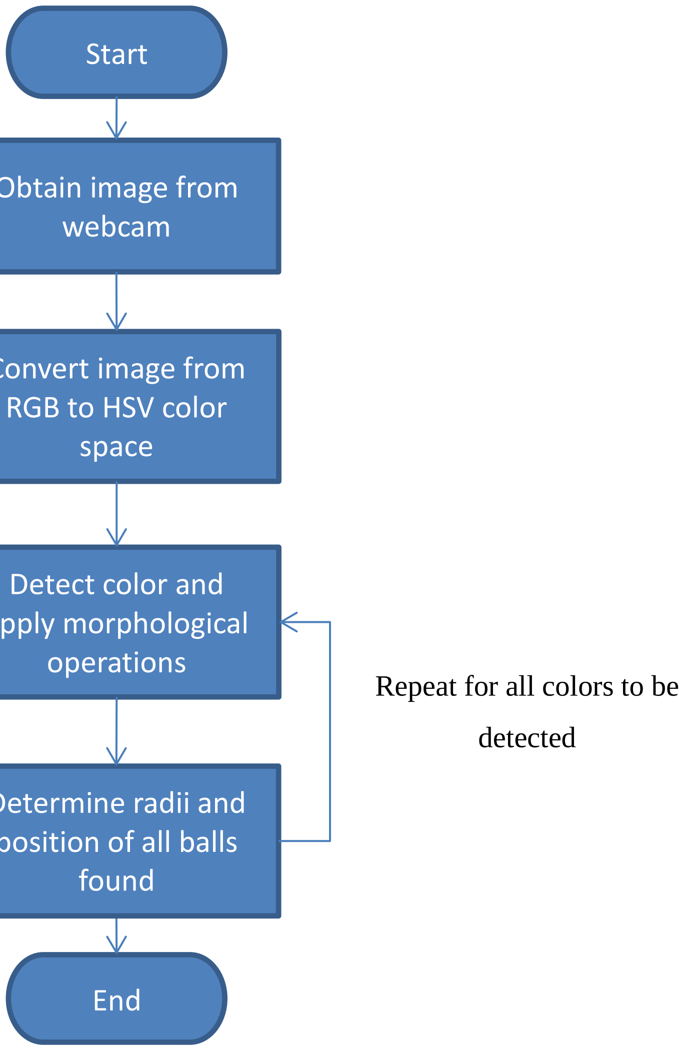

Start
Obtain image from webcam
Convert image from RGB to HSV color space
Detect color and apply morphological operations
Determine radii and position of all balls found
Repeat for all colors to be detected
End

Figure 4-1 Flowchart showing the major steps of the object detection algorithm

A. Obtaining Images

Images are obtained from the webcam at a frame rate of 5 fps and a resolution of 640x480. The frame rate was chosen to be slightly higher than the maximum fps that could be processed by the vision system to prevent this becoming a performance bottleneck.

The camera firmware was found to continuously obtain images which were stored in a FIFO buffer. This feature could not be disabled as access to the firmware was not available. Since the time taken to process each frame is significant, this meant images obtained were at least 0.25 seconds old. When the robot was in motion it would ‘see’ an object as being where it was 0.25 seconds ago as opposed to its current location. As a consequence, it would constantly be lagging the target.

To remedy this, dummy images are constantly obtained and discarded between each step of the algorithm. A timer keeps track of the time elapsed between frames used by the vision system. As the camera’s frame rate is known, the number of frames accumulated in the buffer can be predicted and discarded. While this increases the time taken to identify the 3 colors by 0.1 s to 0.2 s on average, it effectively eliminates the problem.

B. Color Detection

Using the RGB color space for color detection is not preferred as the amount of red, green, and blue in an image is dependent on lighting conditions. The values of the three color channels could change if the intensity of the ambient lighting is varied. Ambiguities where an object of a single color was detected to be multicolored were common. In contrast, the HSV color space where colors are represented as a combination of hue, saturation, and value was found to be minimally affected by changes in ambient lighting. The HSV color space separates the image intensity from the color information providing robustness from varying light intensity and shadows.

The webcam natively captures images in the RGB space which are subsequently converted to the HSV space. A threshold function is used to create a binary image where all the pixels corresponding to the color of interest are denoted as white and the rest are black. The images obtained from the webcam tend to be noisy leading to irregular boundaries on the detected balls. Morphological transformations of opening followed by closing smoothens the boundaries.

Each color to be detected required the creation of a binary image as the threshold values are unique to a particular color. Threshold values used to detect the colors of interest are listed in Table 4-1.

Table 4-1 HSV values of the colors of interest

	Hue	**Saturation**	**Value**
Red	0-13	130-255	0-255
Green	35-65	100-155	50-255
Purple	90-120	25-140	50-255

C. Radius and Position Detection

Contours are drawn around pixel clusters that represent objects of a color. The area enclosed by each of these contours is then calculated. If below a threshold area, then the contour is ignored. This facilitates noise rejection and allows control of the furthest distance at which objects are identified. If the area of the contour is greater than the threshold then we consider it as a positive identification and proceed to draw an enclosing circle of minimum area around it. The coordinates of the center and radius of this circle are considered to be the same as that of the object.

D. Example Scenario

Consider an image captured by the webcam of a green object ball shown in Figure 4-2. The goal of the vision system is to correctly identify the color, to calculate the coordinates of the ball with respect to the viewing area, and approximate the radius of the ball in pixels.

First the image was converted to the HSV color space. Next binary images were created for the three colors of interest as shown in Figure 4-3. Since there were no red or purple pixels with values defined in Table 4-1, the binary images for these colors contain solely black pixels. There was a lack of discernable noise in the binary images due to the background being well differentiated from the colors of interest.

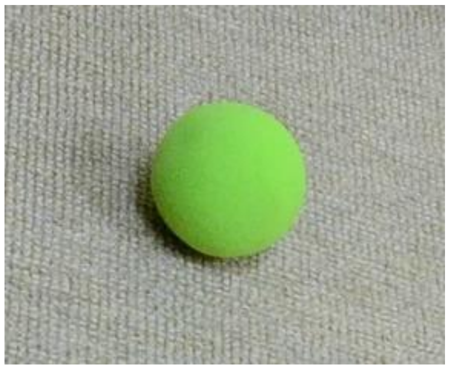


Figure 4-2 Image of a green object ball

The green binary image captured the majority of the pixels that represent the ball. The portion of the ball that was covered in shadows was not identified as the values of the pixels in those regions were outside the threshold values.

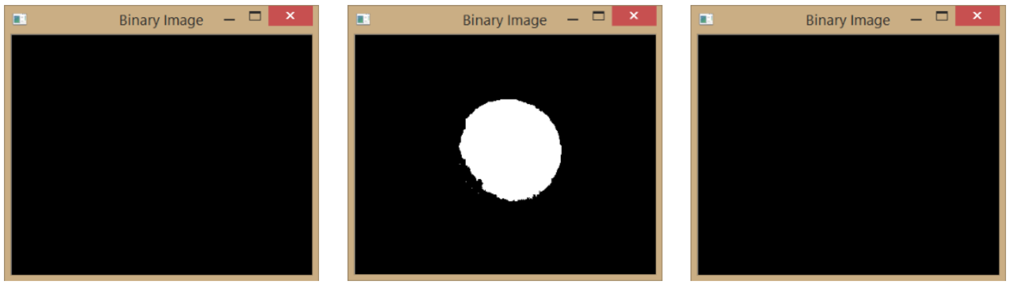


Figure 4-3 Binary images for the color red, green and purple respectively

Morphological operations were performed to smoothen out the boundary of the identified region. In this scenario the boundary was already well defined and no noticeable differences were present after the operations as seen in Figure 4-4.

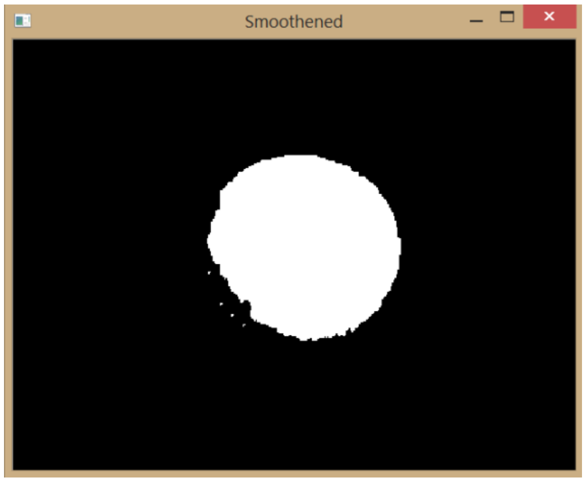


Figure 4-4 Binary image after morphological operations

Next contours were generated to enclose each of the pixel clusters in the binary image. For illustrative purposes a drawing of the contours generated was created as shown Figure 4-5. Each of the 5 contours has been drawn in a different color to aid in identification. Those contours enclosing less than 5000 pixels were ignored as shown in Figure 4-6. This noise rejection eliminated the possibility of the system falsely identifying 5 objects in an image containing 1 object.

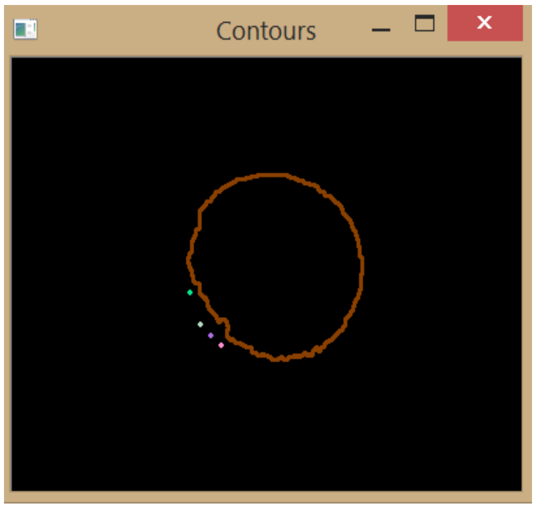


Figure 4-5 Contours drawn around pixel clusters

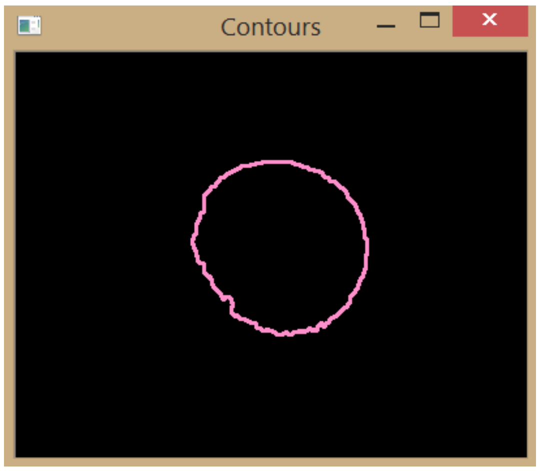


Figure 4-6 Contours enclosing at least 5000 pixels

Finally a circle of minimum area was drawn around the contour as shown in Figure 4-7. The center of the circle had an x coordinate of 198 pixels and a y coordinate of 152 pixels. Its radius was 71.5 pixels. This was considered to be the position and radius of the object.

Figure 4-8 overlays the contour and circle over the original image to confirm the validity of the estimated radius and position of the object.

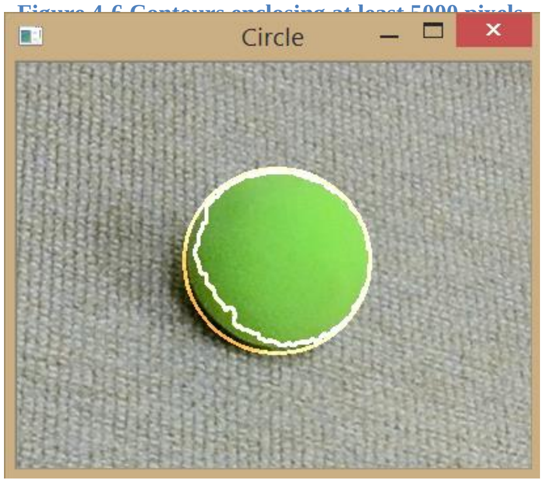


Figure 4-8 Original image overlaid with the contour and circle

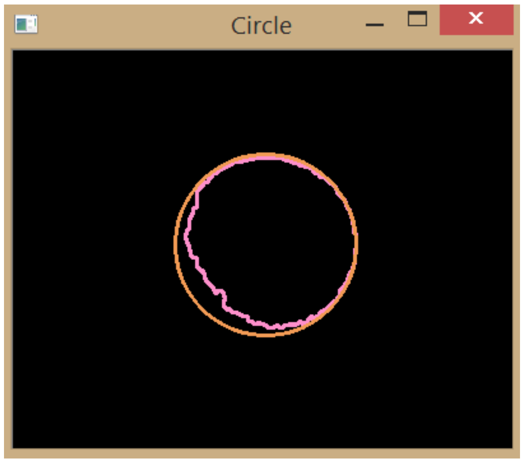


Figure 4-7 Enclosing circle drawn around contour

Chapter 5
Navigation System

In order for the robot to maneuver effectively it had to be able to track its orientation. Information regarding the change in its orientation is used to control the angle of the robot's turns and allows it to be pointed towards objectives. This information is also used in a feedback loop that keeps the robot travelling in a straight line. Without constant correction, the robot was found to travel in a random curved path due to differences in the electrical characteristics of its motors.

1. Coordinate Frames

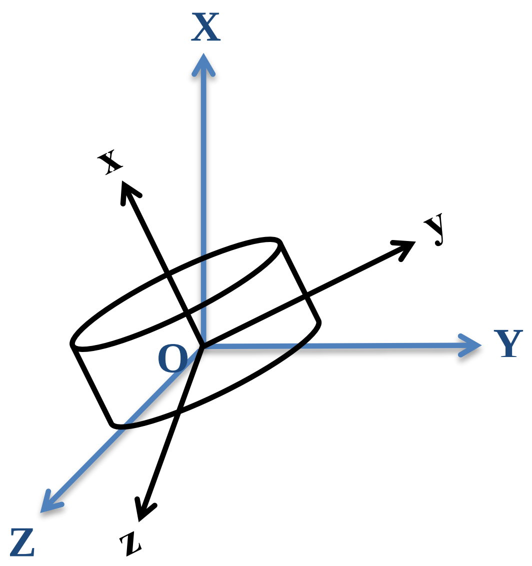


Figure 5-1 DCM Coordinate frames

Orientation kinematics deals with calculating the orientation of a body relative to a global coordinate system. A global coordinate frame O_{XYZ} and a body centered coordinate frame O_{xyz} both sharing a common origin O are defined as shown in Figure 5-1. Let $\hat{\imath}, \hat{\jmath}, \hat{k}$ denote unit vectors for the body frame and $\hat{I}, \hat{J}, \hat{K}$ denote unit vectors for the global frame.

In the body frame we define the unit vectors as

$$\hat{\imath}^B = \begin{bmatrix}1\\0\\0\end{bmatrix}, \qquad \hat{\jmath}^B = \begin{bmatrix}0\\1\\0\end{bmatrix}, \qquad \hat{k}^B = \begin{bmatrix}0\\0\\1\end{bmatrix} \tag{5-1}$$

These unit vectors can also be written in terms of the global frame coordinates

$$\hat{\imath}^G = \begin{bmatrix}i_x^G\\i_y^G\\i_z^G\end{bmatrix}, \qquad \hat{\jmath}^G = \begin{bmatrix}j_x^G\\j_y^G\\j_z^G\end{bmatrix}, \qquad \hat{k}^G = \begin{bmatrix}k_x^G\\k_y^G\\k_z^G\end{bmatrix} \tag{5-2}$$

Consider the components of unit vector $\hat{\imath}^G$. The components i_x^G, i_y^G, i_z^G are lengths of the projection of the vector onto the X, Y, and Z axis of the global frame respectively.

$$i_x^G = |\hat{\imath}^G| \cos\left(X^G, \widehat{\imath^G}\right) = \left|\hat{I}^G\right| |\hat{\imath}^G| \cos\left(\hat{I}^G, \hat{\imath}^G\right) = \hat{I}^G . \hat{\imath}^G \tag{5-3}$$

As long as both vectors are represented with respect to the same coordinate frame, the scalar dot product of the two unit vectors remains the same. That is,

$$\hat{I}^B . \hat{\imath}^B = \hat{I}^G . \hat{\imath}^G \tag{5-4}$$

The superscripts have thus been omitted to aid in readability for the y and z coordinates.

$$i_y^G = |i| \cos(Y, i) = \hat{J} . \hat{\imath} \tag{5-5}$$

$$i_z^G = |i| \cos(Z, i) = \widehat{\mathrm{K}} . \hat{\imath} \tag{5-6}$$

Similarly we can write the other two body frame unit vectors in terms of global frame coordinates as

$$\hat{\jmath}^G = \begin{bmatrix}\hat{I}.\hat{\jmath}\\ \hat{J}.\hat{\jmath}\\ \widehat{K}.\hat{\jmath}\end{bmatrix} \tag{5-7}$$

$$\hat{k}^G = \begin{bmatrix}\hat{I}.\hat{k}\\ \hat{J}.\hat{k}\\ \widehat{K}.\hat{k}\end{bmatrix} \tag{5-8}$$

Combing (5-4), (5-5), (5-6), (5-7), and (5-8) into matrix form we get

$$[\hat{\imath}^G \quad \hat{\jmath}^G \quad \hat{k}^G] = \begin{bmatrix} \hat{I}.\hat{\imath} & \hat{I}.\hat{\jmath} & \hat{I}.\hat{k} \\ \hat{J}.\hat{\imath} & \hat{J}.\hat{\jmath} & \hat{J}.\hat{k} \\ \hat{K}.\hat{\imath} & \hat{K}.\hat{\jmath} & \hat{K}.\hat{k} \end{bmatrix} \tag{5-9}$$

We then obtain the direction cosine matrix (DCM)

$$DCM^G = [\hat{\imath}^G \quad \hat{\jmath}^G \quad \hat{k}^G] = \begin{bmatrix} \cos(\hat{I},\hat{\imath}) & \cos(\hat{I},\hat{\jmath}) & \cos(\hat{I},\hat{k}) \\ \cos(\hat{J},\hat{\imath}) & \cos(\hat{J},\hat{\jmath}) & \cos(\hat{J},\hat{k}) \\ \cos(\hat{K},\hat{\imath}) & \cos(\hat{K},\hat{\jmath}) & \cos(\hat{K},\hat{k}) \end{bmatrix} \tag{5-10}$$

Following a similar procedure we can express the global frame unit vectors in terms of the body frame as

$$\hat{I}^B = \begin{bmatrix} \hat{I}.\hat{\imath} \\ \hat{I}.\hat{\jmath} \\ \hat{I}.\hat{k} \end{bmatrix}, \qquad \hat{J}^B = \begin{bmatrix} \hat{J}.\hat{\imath} \\ \hat{J}.\hat{\jmath} \\ \hat{J}.\hat{k} \end{bmatrix}, \qquad \hat{K}^B = \begin{bmatrix} \hat{K}.\hat{\imath} \\ \hat{K}.\hat{\jmath} \\ \hat{K}.\hat{k} \end{bmatrix} \tag{5-11}$$

Arranging the cosines in matrix form we obtain

$$DCM^B = [\hat{I}^B \quad \hat{J}^B \quad \hat{K}^B] = \begin{bmatrix} \cos(\hat{I},\hat{\imath}) & \cos(\hat{J},\hat{\imath}) & \cos(\hat{K},\hat{\imath}) \\ \cos(\hat{I},\hat{\jmath}) & \cos(\hat{J},\hat{\jmath}) & \cos(\hat{K},\hat{\jmath}) \\ \cos(\hat{I},\hat{k}) & \cos(\hat{J},\hat{k}) & \cos(\hat{K},\hat{k}) \end{bmatrix} \tag{5-12}$$

Using the DCM matrices we can now determine the coordinates of any arbitrary vector defined in terms of body coordinates in terms of global frame coordinates, and vice versa.

A. Properties of the DCM Matrices

From inspection we see that the two matrices shown in (5-10) and (5-12) are transposes of each other.

$$DCM^{B^T} = DCM^G \tag{5-13}$$

Multiplying (5-10) and (5-12) yields

$$DCM^G.DCM^B = DCM^{B^T}.DCM^B = \begin{bmatrix} \hat{I}^{B^T} \\ \hat{J}^{B^T} \\ \hat{K}^{B^T} \end{bmatrix} [\hat{I}^B \quad \hat{J}^B \quad \hat{K}^B] \tag{5-14}$$

$$DCM^{B^T}.DCM^B = \begin{bmatrix} \hat{I}^{B^T}.\hat{I}^B & \hat{I}^{B^T}.\hat{J}^B & \hat{I}^{B^T}.\hat{K}^B \\ \hat{J}^{B^T}.\hat{I}^B & \hat{J}^{B^T}.\hat{J}^B & \hat{J}^{B^T}.\hat{K}^B \\ \hat{K}^{B^T}.\hat{I}^B & \hat{K}^{B^T}.\hat{J}^B & \hat{K}^{B^T}.\hat{K}^B \end{bmatrix} \tag{5-15}$$

Equation (5-16) shows that the scalar product of a unit vector with its transpose is unity.

$$\hat{I}^{B^T}.\hat{I}^B = \left|\hat{I}^{B^T}\right|\left|\hat{I}^B\right| \cos\left(\hat{I}^{B^T}, \hat{I}^B\right) = 1 \tag{5-16}$$

Using (5-15) and (5-16) we can show that

$$DCM^{B^T}.DCM^B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = I_3 \tag{5-17}$$

This leads us to conclude that the DCM matrices are orthogonal.

B. Relationship to Euler Angles

Since the end goal was to obtain the pitch (ϕ), yaw (ψ), and roll (θ) orientation of the robot, the relationship between the DCM matrix and its Euler angle representation was used.

$$DCM = \begin{bmatrix} cos\theta cos\psi & sin\phi sin\theta cos\psi - cos\phi sin\psi & cos\phi sin\theta cos\psi + sin\phi sin\psi \\ cos\theta sin\psi & sin\phi sin\theta sin\psi + cos\phi cos\psi & cos\phi sin\theta sin\psi - sin\phi cos\psi \\ -sin\theta & sin\phi cos\theta & cos\phi cos\theta \end{bmatrix} \tag{5-18}$$

Using (5-18) the orientation of the body can be calculated as

$$\phi = \tan^{-1} \frac{DCM[3][2]}{DCM[3][3]} \tag{5-19}$$

$$\theta = -\sin^{-1} DCM[3][1] \tag{5-20}$$

$$\psi = \tan^{-1} \frac{DCM[2][1]}{DCM[1][1]} \tag{5-21}$$

C. Renormalization

As we are numerically integrating the IMU readings, errors in the matrix elements will lead to a violation of the orthogonality condition of the reference frame axes.

Consider two orthogonal vectors A and B. Since they are orthogonal, the dot product of A and B must be 0. Thus the error can be defined as

$$error = A.B = A^T B \tag{5-22}$$

We then correct for this rotating each of these vectors away from each other by half the error. It is found that this yields a lower residual error compared to assigning the error to either one vector [32].

$$A_{corrected} = A - \frac{error}{2} B \tag{5-23}$$

$$B_{corrected} = B - \frac{error}{2} A \tag{5-24}$$

Now consider a third vector that is to be orthogonal to the corrected A and B vectors. We can use the cross product to define it as

$$C_{corrected} = A_{corrected} \times B_{corrected} \tag{5-25}$$

A Taylor expansion is used to normalize the corrected A, B, and C vectors. This reduces the computational load as compared to using its magnitude [32].

$$R_{normalized} = \frac{1}{2}(3 - R_{corrected} \ \cdot \ R_{corrected})R_{orthogonal} \tag{5-26}$$

2. Inertial Measurement Unit

The IMU described in Chapter 2.1.G contains a 3-axis MEMS accelerometer and gyroscope. Using I^{2}C the values of its hardware registers can be read to obtain a 12-bit number representing the acceleration, and a 16-bit number representing the angular velocities in the three axes. These values are normalized using constants provided in the manufacturer’s datasheet to obtain values of acceleration and angular velocity in metric units.

A. Accelerometer Reading

When the accelerometer is at rest the acceleration acting on it corresponds to the gravity vector. The value of this vector is known to be approximately 9.81 meters per second squared (m/s^2) along the negative z-axis in the global coordinate frame. The accelerometer registers the x, y, and z components of this acceleration in the body frame.

B. Gyroscope Reading

The gyroscope output is the angular velocities of the body in the three axes, i.e. ω_x, ω_y and ω_z. We also know the time interval δt between readings. The angle of rotation about the axes is then given by

$$\delta\theta_x = \omega_x \delta t, \qquad \delta\theta_y = \omega_y \delta t, \qquad \delta\theta_z = \omega_z \delta t \tag{5-27}$$

Linear velocity can be computed from the angular velocity using

$$\hat{v} = \widehat{w} \times \hat{r} \tag{5-28}$$

The linear displacement is then given by

$$\delta r = v\delta t = (\omega \times \hat{r})\delta t \tag{5-29}$$

Using (5-27), (5-28), and (5-29), the linear displacement of the body due to rotations about the three axes can be expressed as

$$\delta\hat{r} = \delta\hat{r}_x + \delta\hat{r}_y + \delta\hat{r}_z \tag{5-30}$$

$$\delta\hat{r} = \left(\omega_x \times \hat{r} + \omega_y \times \hat{r} + \omega_z \times \hat{r}\right)\delta t \tag{5-31}$$

$$\delta\hat{r} = \left(\left(\omega_x + \omega_y + \omega_z\right) \times \hat{r}\right)\delta t \tag{5-32}$$

The linear velocity resulting from the three rotations can then be computed from

$$\hat{v} = \frac{\delta\hat{r}}{\delta t} = \left(\omega_x + \omega_y + \omega_z\right) \times \hat{r} \tag{5-33}$$

For larger rotations the order of rotation becomes significant and they cannot be summed up. By ensuring that the time interval (δt) between gyroscope readings is small we can assume that the rotations experienced by the robot are very small. Assume $\hat{r}_t$ is the initial position of a vector at time t and δt is the time interval between gyroscope measurements. Then,

$$\hat{r}_{(t+\delta t)} = \hat{r}_t + \hat{r}_t \times d\theta_t \tag{5-34}$$

Using (5-33) and (5-34) in combination with DCM^G defined by (5-10) we arrive at

$$DCM^G(t+\delta t) = DCM^G(t)\begin{bmatrix} 1 & -d\theta_z & d\theta_y \\ d\theta_Z & 1 & -d\theta_x \\ -d\theta_y & d\theta_x & 1 \end{bmatrix} \tag{5-35}$$

3. Orientation Tracking Algorithm

A gyroscope cannot measure the absolute orientation of a body with respect to the global frame, but we can compute the orientation of an object at a future time given its initial orientation and angular velocities obtained from the gyroscope. An accelerometer on the other hand measures the earth's gravity vector and thus can be used to determine the orientation of the body frame with respect to the global frame.

Accelerometer readings tend to be affected by linear accelerations such as vibrations and need to be averaged over longer periods of time to obtain a stable reading. Gyroscopes on the other hand tend to drift over long periods of time which introduces an offset into the readings. Thus we use the gyroscopic readings primarily to estimate the body's change in orientation and use weighted accelerometer readings to correct for gyroscopic drift.

The steps used to accomplish this are

1. Start with a DCM matrix that corresponds to an initial orientation of zero yaw, pitch, and roll with respect to the global frame. This corresponds to an identity matrix.
2. Every time a gyroscope reading is taken we update the DCM matrix using (5-35)
3. The orientation of the body is calculated using (5-19), (5-20) and (5-21)
4. The DCM matrix is renormalized using (5-23), (5-24), and (5-26)

4. Feedback Control Loops

Flowcharts depicted in Figure 5-2 and Figure 5-3 show how the orientation readings are used to turn the robot by a required angle, and correct its course to keep it moving in a straight line respectively.

A. Turning

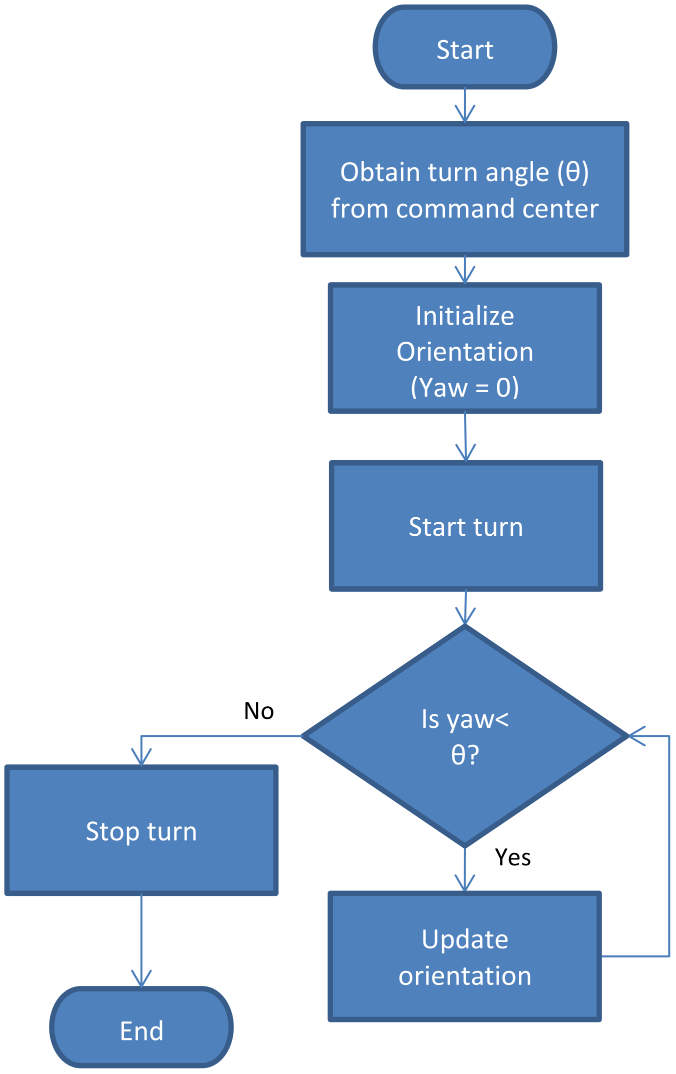

Start
Obtain turn angle (θ) from command center
Initialize Orientation (Yaw = 0)
Start turn
Is yaw< θ?
No
Yes
Stop turn
Update orientation
End

Figure 5-2 Flowchart of feedback loop used to turn the robot by a desired angle

B. Movement in a Straight Line

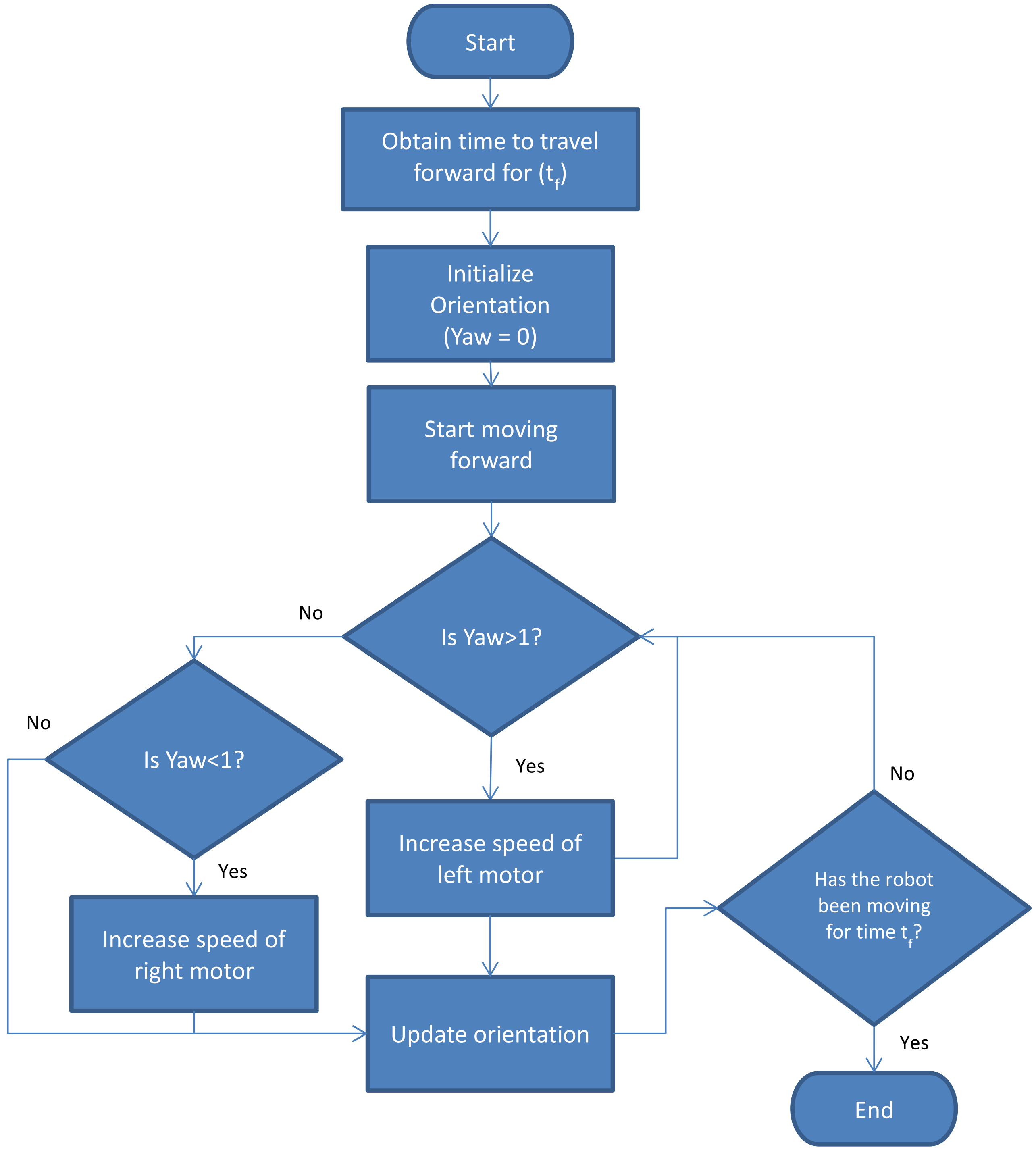

Start
Obtain time to travel forward for (t_f)
Initialize Orientation (Yaw = 0)
Start moving forward
Is Yaw>1?
No
Yes
Is Yaw<1?
No
Yes
Increase speed of left motor
Increase speed of right motor
Update orientation
Has the robot been moving for time t_f?
No
Yes
End

Figure 5-3 Flowchart of control loop used to straighten forward motion of the robot

Chapter 6
Action Selection Mechanism

In order for the system to be autonomous it had to decide what to do. In order for it to be successful it had to intelligently select appropriate actions based on external and internal stimuli. A suitable action selection mechanism (ASM) would also be required to replicate the behavior described in Chapter 3. Dynamic planning methods compute the next action to be taken based on the current internal and external state. This type of ASM is ideal when limited computational resources are available. On the other hand, to replicate the behavior required a goal driven architecture was preferred [33]. To this end a hybrid architecture was implemented in the final system.

1. Overview

The behavior to be implemented has a set of goals each of which is decomposed into subgoals. Each subgoal is associated with a dynamic plan that the ASM selects in order to accomplish the goal. This modular structure permits reuse of plans. Each plan is decomposed into sequential steps and implemented as a function. Dynamic plans for seeking a ball, avoiding a danger ball, avoiding a boundary wall and scanning an area were defined. Dynamic plans are interchangeably referred to as plans in this document. Table 6-1 shows the relationship between goals and subgoals. To implement the plan associated with each subgoal, condition-action rules are used similar to an expert system. Rules define the 'knowledge' the system possesses and in this system were either factual or procedural. Factual knowledge as the name suggests consists of facts such as the color of a ball defining if it relates to food, danger or a safe area. Procedural rules govern how actions the robot take will be carried out. For example, if an objective ball is encountered then move towards it. Conflict resolution in case of competing goals is handled using priorities where certain goals have a higher priority than others. For example if a danger ball is seen in front of an objective ball then avoid the danger instead of moving towards the objective.

Two enumerations containing a list of all goals and subgoals are used by the system to keep track of its current state.

Goals - Overall objective to be achieved by the system. In this case they were food, home, avoid and none.

Subgoals - Actions taken that lead to the fulfilment of a goal. In this case they were ball_find, ball_seek, ball_eat, sleep, avoid_ball and avoid_wall.

Table 6-1 Description of goals, subgoals and possible combinations

Goal	Description	Subgoals Possible	Description
food	The robot needs to find a food ball	ball_find	Find an objective ball
		ball_seek	Move toward a found ball
		ball_eat	Recharge robot health
home	The robot needs to find a safe area ball	ball_find	Find an objective ball
		ball_seek	Move toward a found ball
		sleep	Do nothing
avoid	The robot needs to avoid a danger ball	avoid_ball	Avoid a ball obstructing robot
		avoid_wall	Avoid a boundary wall
none	The robot has no objective	sleep	Do nothing

The inputs to the ASM consist of the robot's state variables, and Cartesian coordinates of identified objects obtained from the image processing module. An event handling function and an interrupt implemented in software handle flow control. The event handler executes the appropriate plan based on the current goal, subgoal and system state. For example, if the robot's health is low and the goal is none, the action taken would be to scan for food balls. The interrupt handler can stop the currently executing goal and instead execute another one. Once complete, it then reloads the previous goal and subgoal. For example, if the robot was searching for food and encountered danger the goal would be changed from 'food' to 'avoid'. The ASM would then set the appropriate subgoal and load the dynamic plan used to evade danger. Once evaded, the handler would reload the previous goal ('food'), subgoal ('ball_find'), and plan.

2. Input Data

The input data required by the ASM is transmitted from the robot to the command center using user defined data structures. The data structure ‘RobotParam’ stores the robot’s health, motor speed, angle to turn by if the command is to turn, IR sensor reading, and the command generated by the system.

```
struct RobotParam{
    unsigned short int health;
    unsigned short int speed;
    unsigned short int turnDeg;
    short int irDist;
    char comm;
};
```

The data structure ‘BallData’ is used to transmit the positional data of balls that have been identified. It stores the id which helps identify the type of object, the number of objects identified followed by their 2D coordinates in the Cartesian plane, and their estimated radii. The maximum number of objects that can be identified in a single frame is limited to 5.

```
struct BallData{
    char id;
    int count;
    int x[5];
    int y[5];
    int r[5];
};
```

Each unique object to be identified requires one variable of type ‘BallData’ to be transmitted. For example to identify red, green, and purple balls we would require three variables of the type.

3. Definition of Dynamic Plan Function

The definition of a plan has to follow a specific format such that it is compatible with the event handler and software interrupt. Let us consider the behavior of the robot when it encounters a boundary wall. We will assume that the appropriate response is to turn away from the wall.

Further we will assume that the direction and angle of the turn can be random as long as it successful in steering the robot away from the wall.

If we sub divide the response required into sequential steps, one solution would be:

1. Stop the robot to prevent it from colliding with the wall
2. Randomly chose either a left or right turn
3. Turn by an arbitrarily chosen angle
4. Check if the wall is still blocking forward motion
5. If so, go to step 2
6. Else, the robot has successfully avoided the boundary wall

Figure 6-1 shows the code used to implement the above logic in a manner that is compatible with the ASM. The variable ‘goalStep’ is used to track which step is currently being

```
void Action::avoidWall(){

    switch(goalStep){
    case 1:
        dexter.comm='h';
        goalStep++;
    break;
    case 2:
        if(dexter.irDist>15){
            goalStep++;
            dexter.comm='h';
        }
        else{
            if(rand()%2==0)
                dexter.comm='l';
            else
                dexter.comm='r';

            dexter.turnDeg=(45);
        }
    break;
    case 3:
        if(interruptStatus==2)
                interruptStatus=3;
        scans=1;
        dexter.comm='h';
    break;

    }
}
```

Figure 6-1 Dynamic plan used to avoid boundary walls

executed. Step 1 is implemented in case 1. Steps 2, 3, 4, and 5 are implemented in case 2 as it checks if the IR sensor detects an obstruction in front of the robot. If so, then the boundary wall is still present and the robot is turned either left or right by 45°. If not, execution moves on to the next step. Case 3 instructs the interrupt handler that avoidance is complete and the previous plan can be resumed.

Such decomposition allows the flow of control to be returned to the event handler at the end of each step. The event handler in turn can check if conditions that require this particular response to be halted arise. In case a situation that requires immediate attention such as a danger ball, or a goal with a higher priority arises, control is passed to the interrupt handler which then takes the appropriate action. Otherwise, the current plan is called and the next step is executed.

Plans to be used with interrupt handler - If a plan function is to be called by the interrupt handler its final goal step must set the 'interruptStatus' flag variable as seen in 'case 3' of Figure 6-1. This cues the interrupt handler to restore the previous system goal, subgoal, goal step, and plan function. Otherwise, the system will not resume its previous goal.

A. Detailed Breakdown of Defined Plan Functions

Each time a plan function is called all unindexed steps are carried out followed by the goal step stored in the counter variable. It is to be noted that the flowcharts that follow seek to provide an overall picture of the structure of these functions. As such they do not take into account the interactions between the event handler, interrupt, and the plan function. In addition, in the actual system certain steps are broken down further. The flowcharts omit these steps to prevent confusion that would arise from unnecessarily complicated charts. The descriptions on the other hand do not omit these steps and provide a complete picture of the functions.

a. Searching for Food

Figure 6-2 illustrates the main steps using a flowchart.

Goal Step Index	Process	Description
	Check if food ball count is greater than 0. If yes, go to goal step 3 Else, go to goal step 1	If a food ball exists, the robot does not have to search for one and can proceed to seek it. Otherwise, start at goal step 1.
1	Randomly chose a left or right turn, and a turn angle that is a multiple of 15. Set this as the robot's command. Increment goal step.	A multiple of 15 was chosen as the turn had to be large enough to make sure the new image obtained did not overlap the previously surveyed area.
2	Move forward for 0.5 seconds Run this step twice. Once this step has been run twice go back to goal step 1.	The function uses a static variable so that even when the function is out of scope the number of times this step has been repeated can be tracked. Thus the robot moves forward for a total of 1 second.
3	Call the 'seek' function to get close enough to the food ball. If seek is successful go to goal step 4. Otherwise call the 'scan' function If this also fails then go to goal step 1.	If this goal step has been reached it means that a food ball has been found. Thus we should be able to seek it. If we cannot that means due to errors in the system we have overshot the ball. Thus if we scan the area we should be able to find it again. If unfortunately we are unable to find it even after scanning we go back to the goal step 1 and search again.
4	Stop the robot from moving.	This is to prevent the robot from losing the ball or overshooting it once close enough.
5	Recharge the health. Call the emotion engine function to register this as an event.	Since we are close enough to food we can now recharge the health as the robot can be considered to be eating the food. We also send the emotion engine data signifying that the robot has found food.

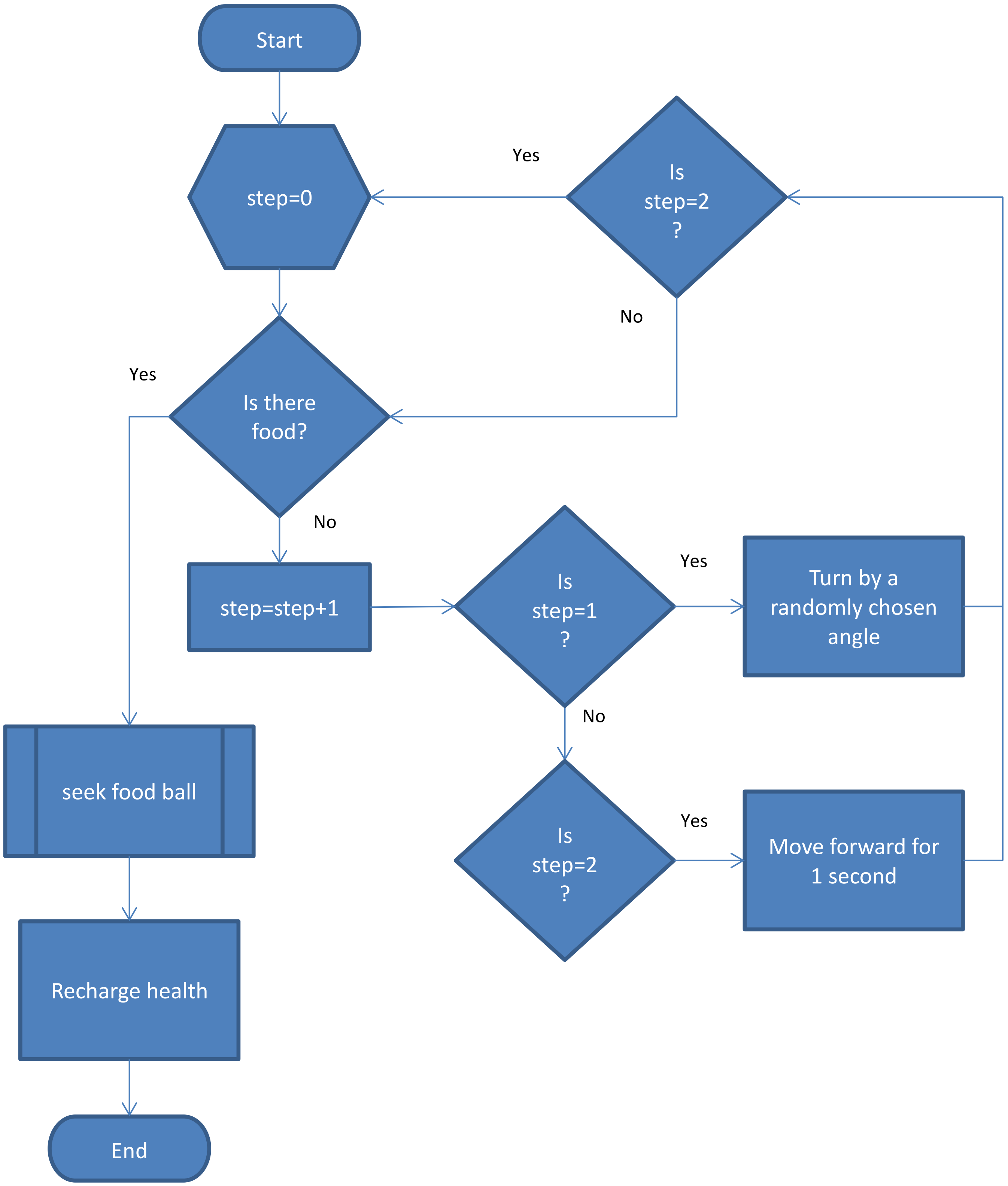


Figure 6-2 Flowchart of steps involved in searching for food

b. Searching for a Safe Area Ball

Figure 6-3 illustrates the main steps using a flowchart.

Goal Step Index	Process	Description
	Check if safe ball count greater than 0. If yes, go to goal step 3. Else, go to goal step 1	If a safe area ball exists, the robot does not have to search for one and can proceed to seek it. Otherwise, start at goal step 1.
1	Randomly chose a left or right turn and a turn angle that is a multiple of 15. Set this as the robot's command. Increment goal step.	A multiple of 15 was chosen as the turn had to be large enough to make sure the new image obtained was not overlapping the previously surveyed area.
2	Move forward for 0.5 seconds. Run this step twice. Once this step has been run twice go back to goal step 1.	The function uses a static variable so that even when the function is out of scope the number of times this step has been repeated can be tracked. Thus the robot moves for a total of 1 second.
3	Call the 'seek' function to get close enough to the food ball. If seek is successful go to goal step 4. Otherwise call the 'scan' function. If this also fails then go to goal step 1.	If this goal step has been reached it means that a food ball has been found. Thus we should be able to seek it. If we cannot that means due to errors in the system we have overshot the ball. Thus if we scan the area we should be able to find it again. If unfortunately we are unable to find it even after scanning we go back to the beginning step and search again.
4	Stop the robot from moving. Set the goal of the robot to none and subgoal to sleep. Call the emotion engine function to register this as an event.	This is to prevent the robot from losing the ball or overshooting it once close enough. Then we set the goal of the system to none as the robot has completed its goal. We also send the emotion engine data signifying that the robot has reached a safe area.

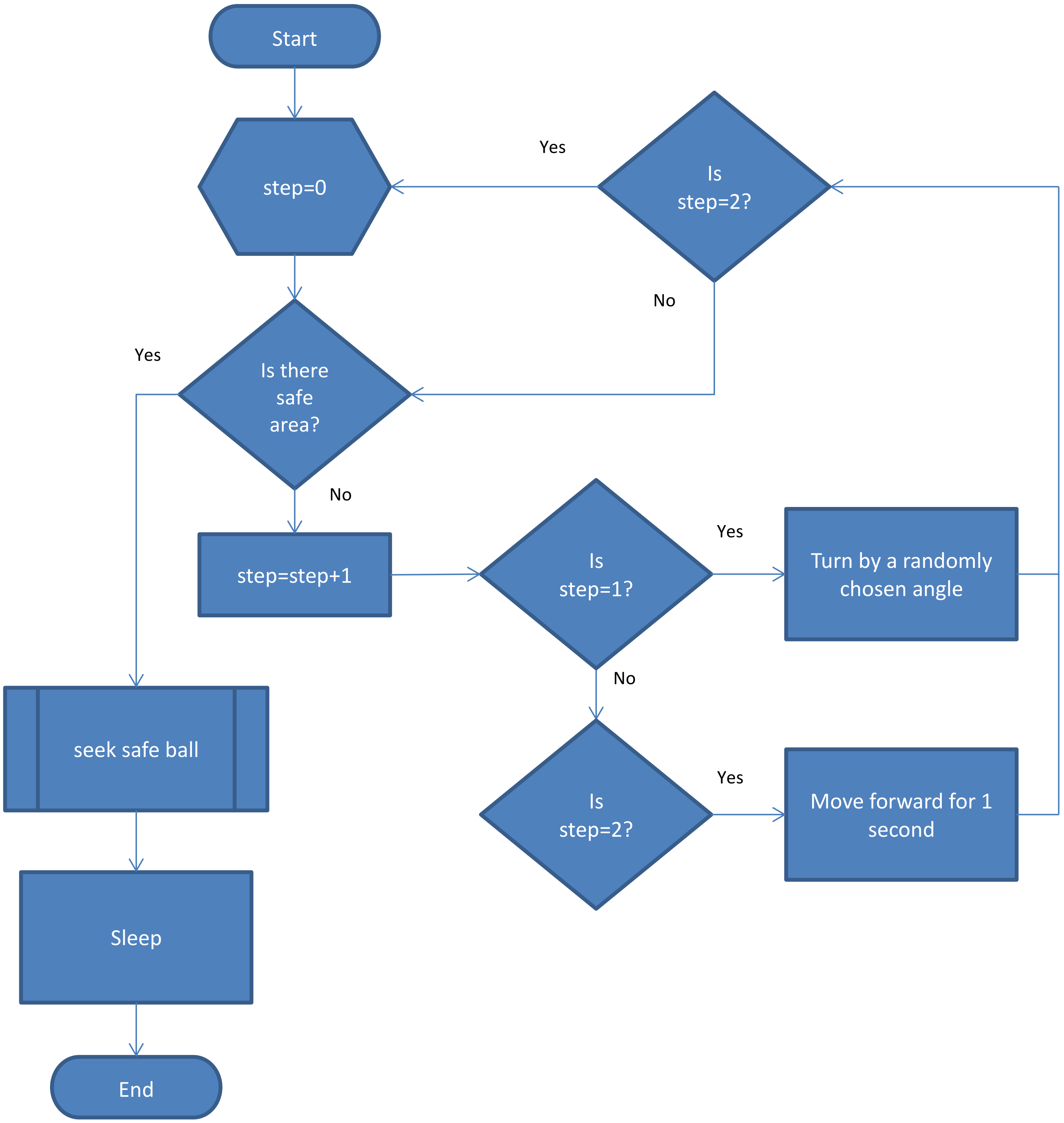


Figure 6-3 Flowcharts of steps involved in finding safe areas

c. Seeking an Objective Ball

Unlike the other functions, the seek function is not based on goal steps. It is written using if-else statements and as such all steps are executed each time the function is called. Figure 6-4 illustrates the main steps using a flowchart.

Goal Step Index	Process	Description
	Check if IR sensor distance is less than 9cm.	This condition means that the robot is close enough to the ball.
	Find the x and y coordinates of the closest objective ball.	If multiple balls of the same color are identified we will move towards the one that is closest to the robot.
	Calculate the angle by which the robot must turn to face the ball head on. Calculate the distance between the ball and the robot.	Based on the coordinates and the pixel count of the camera used we can estimate the angle by which we must turn to face the ball. We can also estimate the distance to the ball using the same principle.
	If the angle is less than 9 degrees move forward. Set the time to move forward as time required to move 75% of the distance calculated.	If the angle to be turned to face the ball is less than 9 degrees we move forward towards the ball. This prevents an infinite loop where the robot oscillates while trying to reduce the angle to zero. We only move 75% of the distance as this allows us to readjust course if necessary.
	If the ball is to the left of the center of field of view then set the direction of turn to the left. Otherwise the turn direction is set to the right. The turn angle is set to 75% of the angle calculated.	If the angle is greater than 9 degrees we turn to minimize this angle. We turn towards the direction that minimizes this angle, i.e. we turn towards the ball. We only turn 75% of the angle that was calculated so as to allow us to re adjust course if necessary.

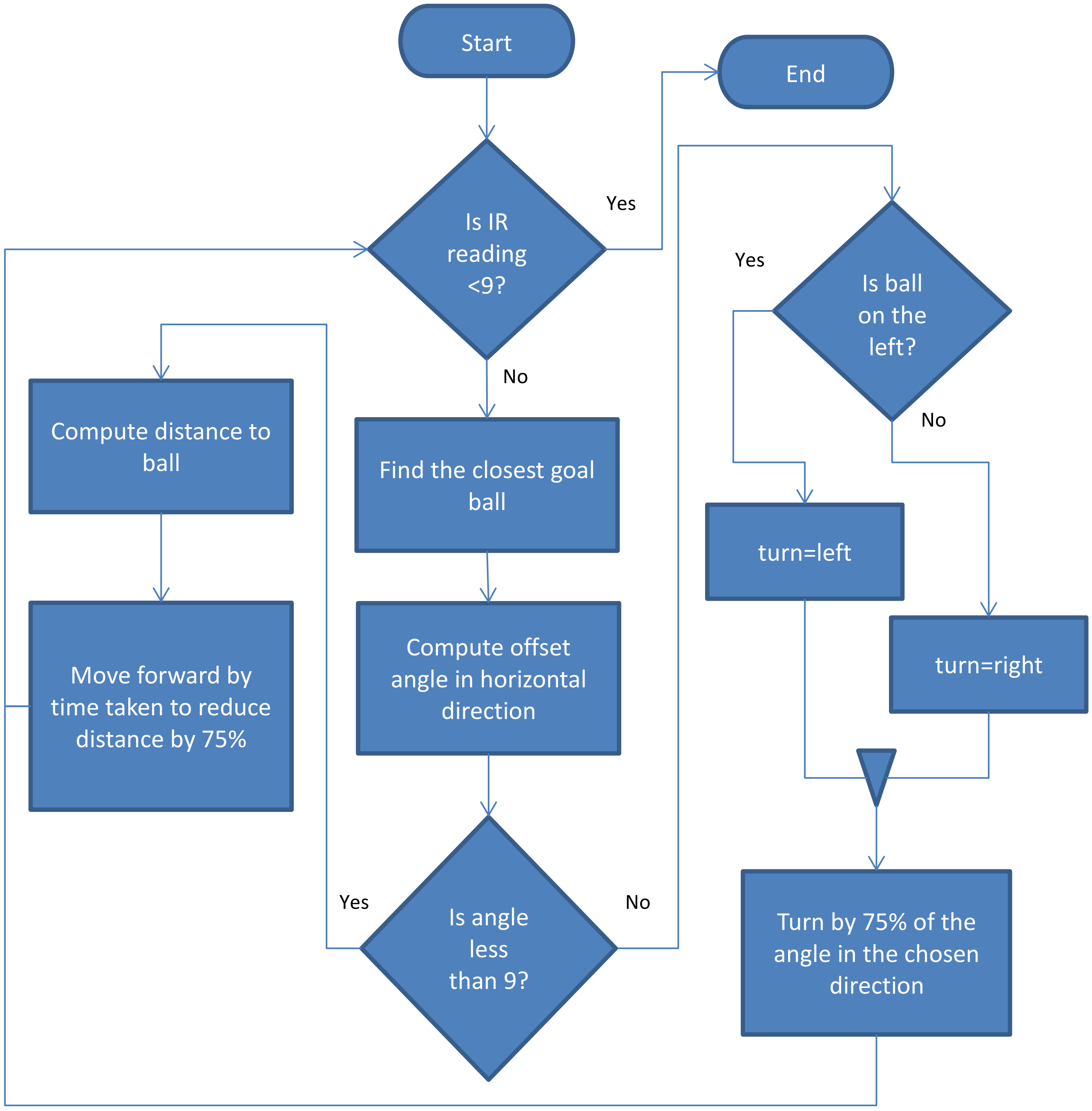

Start
End
Is IR reading <9?
Yes
No
Find the closest goal ball
Compute offset angle in horizontal direction
Is angle less than 9?
Yes
No
Compute distance to ball
Move forward by time taken to reduce distance by 75%
Is ball on the left?
Yes
No
turn=left
turn=right
Turn by 75% of the angle in the chosen direction

Figure 6-4 Flowchart of steps involved in seeking an objective ball

d. Scanning Around the Robot to Find a 'Lost' Objective Ball

This function scans the immediate 90 degree arc around the robot for objective balls. It is used when an objective ball goes out of the field of view during the seeking phase due to an overshoot. Overshoots were found to be most likely due to a turn angle that was too great and not because the robot drove past the objective ball. As such a 90 degree arc was sufficient in practice. Figure 6-5 illustrates the main steps using a flowchart.

Goal Step Index	Process	Description
	Check if objective ball count is greater than 0. If yes scan successful.	If the count is not zero then a ball has been found.
1	Turn 45 degrees to the right.	We have chosen to scan 90 degrees around the robot as it was found to be highly unlikely that the robot has overshot the ball.
2	Turn 15 degrees to the left.	
3	Turn 15 degrees to the left.	
4	Turn 30 degrees to the left.	This prevents rescanning the area directly in front of the robot.
5	Turn 15 degrees to the left.	
6	Turn 15 degrees to the left.	
7	Turn 45 to the right.	Return back to the starting position.
8	Scan unsuccessful.	An objective ball has not been found.

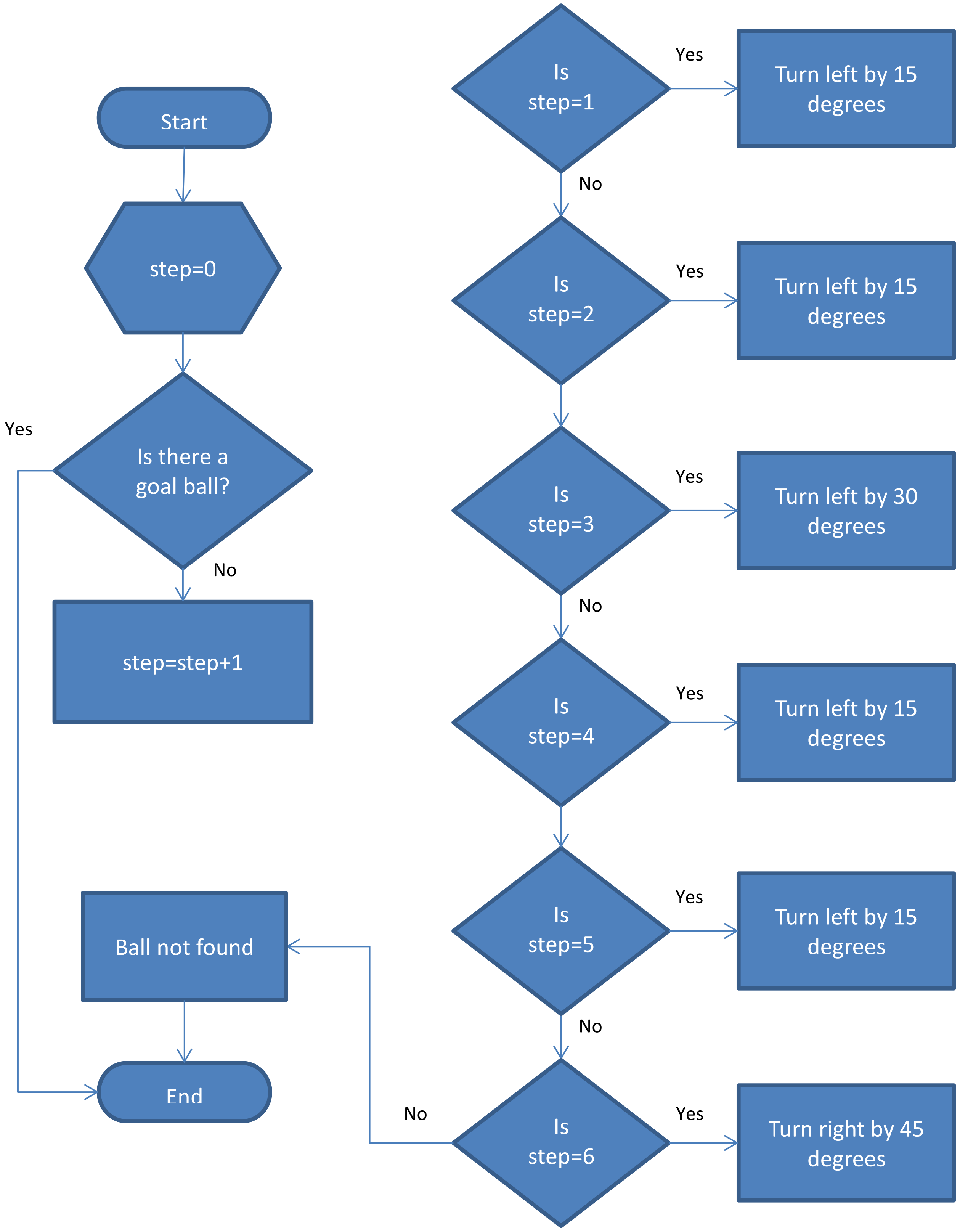


Figure 6-5 Flowchart of steps involved in scanning 90 degrees around robot

e. Avoiding Danger Balls

This function is used solely with the software interrupt system as whenever the robot has to avoid a ball the interrupt system is called. Thus goal step 5 is concerned with disabling the software interrupt which in turn will resume the previous goal that the system was trying to complete. Figure 6-6 illustrates the main steps using a flowchart.

Goal Step Index	Process	Description
1	Check if a danger ball is closer to the robot than an objective ball. If the danger ball is closer and multiple danger balls exist find the closest one. If the danger ball is to the left of the robot go to goal step 2. Otherwise go to goal step 3.	If both a danger and objective ball have been identified but the danger is further than the objective, the robot is not in danger.
2	Set the turn direction to be to the right and the turn angle to be a multiple of 30 degrees. Go to goal step 4.	Turning away from danger. A turn angle greater than 30 degrees ensures that the robot would not collide with the danger ball.
3	Set the turn direction to be to the left and the turn angle to be a multiple of 30 degrees. Go to goal step 4.	
4	Move forward for 0.5 seconds. Repeat this goal step twice then increment goal step.	The function uses a static variable so that even when the function is out of scope the number of times this step has been repeated can be tracked. Thus the robot moves for a total of 1 second.
5	Disable the software interrupt.	Once the robot has evaded the danger, the previous goal can be resumed.

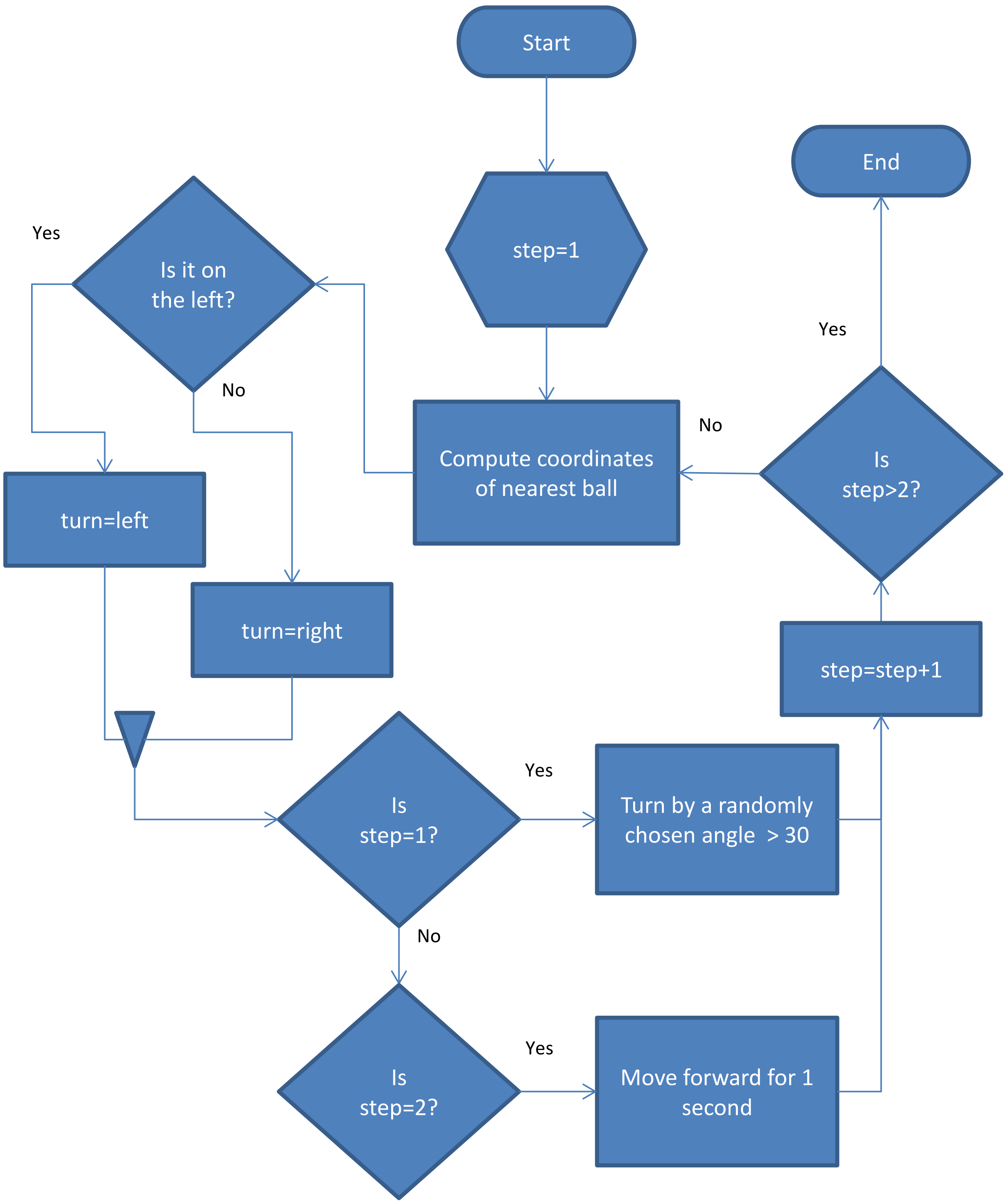


Figure 6-6 Flowcharts of steps involved in avoiding a ball

f. Avoiding Boundary Walls

This function is another one solely used with the software interrupt system as whenever the robot has to avoid a boundary wall or obstruction the interrupt system is called. Thus goal step 5 is concerned with disabling the software interrupt which in turn will resume the previous goal that the system was trying to complete. Figure 6-7 illustrates the main steps using a flowchart.

Goal Step Index	**Process**	**Description**
1	Stop the robot.	This prevents the robot from crashing into the obstruction.
2.	If IR distance is greater than 15 then go to the next goal step. Otherwise randomly turn by 45 degrees to either the left or the right.	If IR distance is greater than 15 then the sensor does not detect an obstruction close enough to the robot to cause alarm. If an obstruction is detected turn by 45 degrees. It was found in practice that this angle was large enough to avoid walls and allow exploration along them.
3.	Disable the software interrupt.	Once the robot has evaded the danger, the previous goal can be resumed.

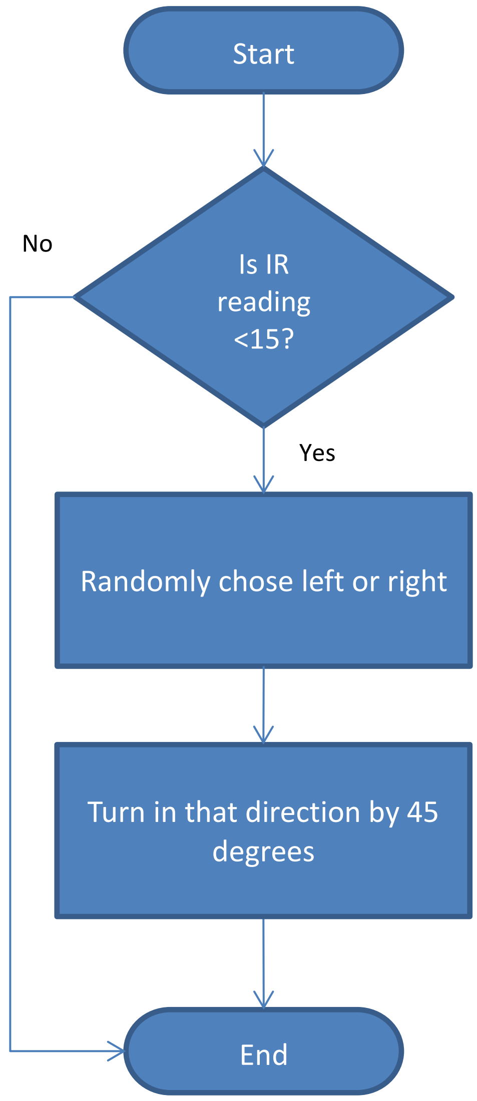

Start
Is IR reading <15?
No
Yes
Randomly chose left or right
Turn in that direction by 45 degrees
End

Figure 6-7 Flowchart of steps involved in avoiding the boundary walls

4. Event Handler

This handler is in charge of receiving inputs, loading the appropriate plan function, setting the system goal, and returning commands to be sent to the robot. It effectively integrates the plans and software interrupt together. The operations carried out by the handler are

1. Check for danger
2. Enable the software interrupt if necessary
3. Set a plan function based on the current goal
4. Call the set plan function
5. Send event data to the emotions engine
6. Return control commands to be sent to the robot

The event handler uses a function pointer to execute plan functions. When a particular plan is to be executed the pointer is redirected to the memory location of the plan function. Since the handler uses a pointer to execute the plan functions it does not need to know what function is currently being executed. This allows other parts of the ASM to change the plan function currently being executed.

The handler has to check for any immediate danger to the safety of the robot. In this case it would be a danger ball or a collision with the boundary wall. This is accomplished using distance readings obtained from the IR sensor along with verifying that the number of detected balls of a certain color

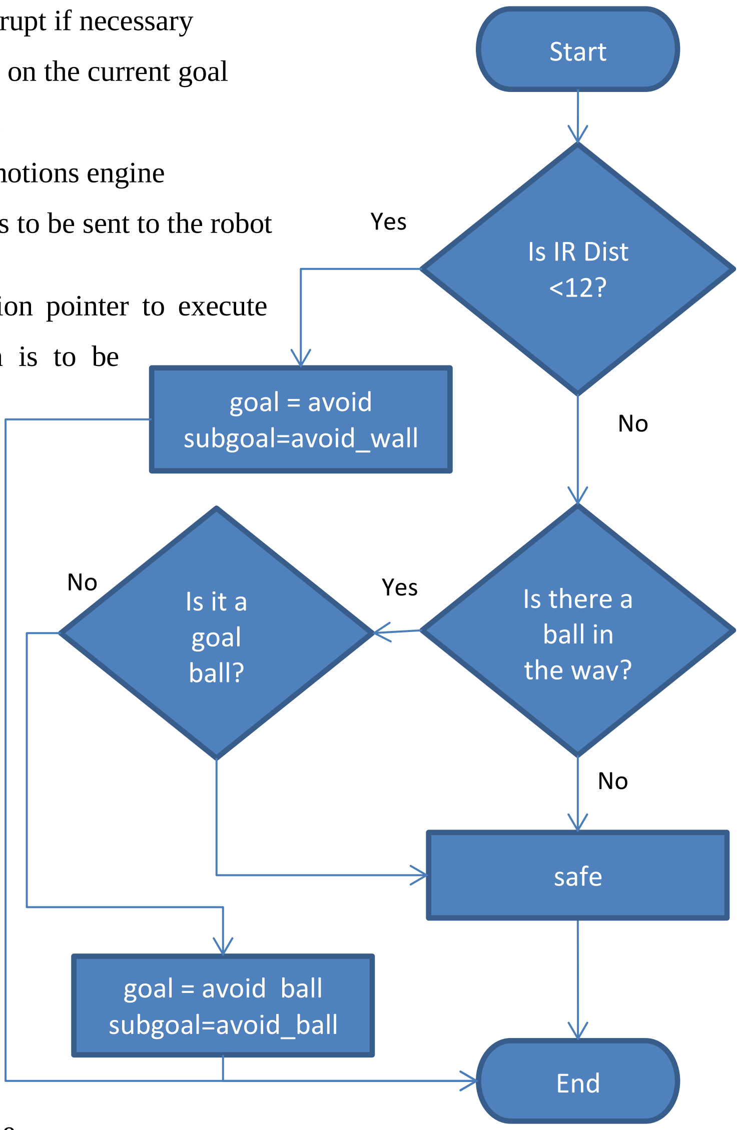

Start
Is IR Dist <12?
Yes
No
goal = avoid subgoal=avoid_wall
Is there a ball in the way?
Yes
Is it a goal ball?
No
No
safe
goal = avoid ball subgoal=avoid_ball
End

Figure 6-8 Flowchart of steps involved in checking for danger

is zero. This process is illustrated in Figure 6-8. In case a danger is detected the handler calls the interrupt handler which then changes the system goal and subgoal. Based on this an appropriate plan function is selected.

The default system goal at the beginning of time is defined to be ‘none’. As time progresses state variables change leading to a change in the system goal. When the conditions shown in Table 6-2 are met, the handler changes the current system goal to the specified goal and the corresponding plan is executed.

Table 6-2 Conditions resulting in change of system goal

Current Goal	State Variable	New Goal
none	health < Threshold	food
food	health > 99	home

The event handler stores commands generated by the ASM in the ‘RobotParam’ data structure which is then sent to the robot.

In case pre-defined events discussed in Chapter 7 occur, the handler sends information regarding the event to the emotions engine. Events are usually defined to occur when a goal is completed. The handler thus sends data identifying the type of event and the number of computational steps taken to complete the goal.

5. Interrupt Handler

The definition of goals, subgoals, and the use of discretized plan functions allows the simulation of a software based interrupt. This feature allows certain plans to be prioritized over others. In this scenario only plans related to avoidance of danger were to be prioritized and consequently are used with the interrupt handler.

The interrupt handler is called by the event handler if it detects an immediate threat to the robot by setting an interrupt flag. The goal and subgoal are set based on the type of danger as seen in Figure 6-8. The interrupt handler then saves the state of the system and loads the appropriate plan function based on the goal and subgoal. The event handler then executes this plan function.

For example let us assume that the current system goal is ‘food’ with the subgoal ‘ball_find’. This means that the robot is searching the environment for a green ball. During this search if it encounters a red danger ball the following events would occur.

1. The event handler would detect the danger and an interrupt flag would be set to ‘1’ enabling the interrupt
2. The interrupt handler would save the current system goal, subgoal, goal step, and plan function pointer
3. The interrupt handler would change the system goal to ‘avoid’, subgoal to ‘avoid_ball’, goal step to 1, and goal function pointer to the appropriate avoidance plan function. The interrupt flag would be changed to ‘2’ signifying a return to normal operation.
4. The event handler would run the avoidance routine as per usual
5. When the routine is complete, the interrupt flag would be changed to ‘3’
6. This would prompt the interrupt handler to restore the original system state, i.e. change the system goal back to ‘food’ and so on
7. The interrupt flag would then be reset to disable the interrupt

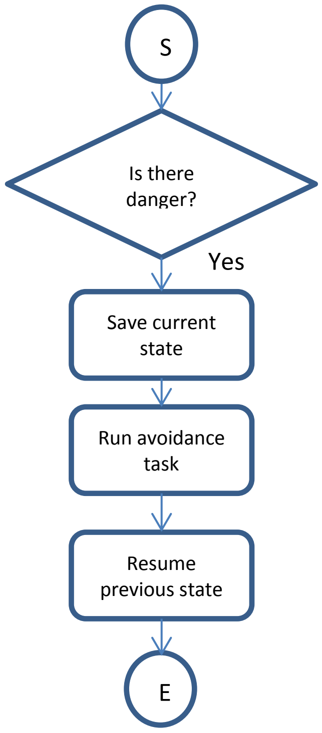


Figure 6-9 Interrupt flow of control

Chapter 7
Emotions Engine

1. Computational Model

In their study on momentary subjective well-being, Rutledge et al. [7] obtained the computational model shown in (7-1). They showed that momentary subjective well-being was explained by the cumulative effects of not only recent reward expectations but also prediction errors resulting from those expectations. In the model CR represents certain rewards, EV their expected value and RPE the reward prediction error (RPE).

$$\begin{aligned} Happiness(t) = w_0 + w_1 \sum_{j=1}^{t} \gamma^{t-j}\, CR_j + w_2 \sum_{j=1}^{t} \gamma^{t-j}\, EV_j \\ + w_3 \sum_{j=1}^{t} \gamma^{t-j}\, RPE_j \end{aligned} \qquad (7\text{-}1)$$

Long et al. [8], [9] modified this model to incorporate emotions and temperaments into cognitive mobile robots. Their model shown in (7-2) considered positive and negative rewards and its effect on the robot's emotional state. The winner take all approach taken meant that the emotion with the highest value was considered as the robot's emotional state. In this model R_{ij}^{+} represents positive rewards while R_{ij}^{-} represents negative rewards.

$$Emotions(t)_i = w_{0_i} + \sum_{j=1}^{t} \gamma_i{}^{t-j} (w_{1_i} R_{ij}^{+} + w_{2_i} R_{ij}^{-}) \qquad (7\text{-}2)\ [8]$$

The emotional model used in this research study is shown in (7-3) and incorporates the RPE term from (7-1) into (7-2). The positive or negative effect of an event on the emotion is represented by the term R_j. The reward prediction error is represented by the term RPE. w_0, w_1, and w_2 are weighting factors. γ is a decay factor that governs the impact of past events on the current emotional state.

$$Emotion(t)_i = w_{0_i} + \sum_{j=1}^{n} \gamma_i^{(t_f - t_j)} w_{1_i} R_j + \sum_{j=1}^{n} \gamma_i^{(t_f - t_j)} w_{2_i} RPE_j \qquad (7\text{-}3)$$

2. Emotions and Temperament Constants

A distinction has to be made between temperaments and emotional states. Temperaments are considered to be biologically based and derived from genetic predispositions, maturation, and experience. They are expected to be relatively stable over time. Emotions from a functionalist approach are defined as a person's readiness to establish, maintain or change one's relationship to his or her changing circumstances. By contrast to temperamental variability, emotional reactions can be enduring or brief [34].

Psychology studies tend to disagree, and there is no real consensus on the total number of emotions experienced [35]. As such the four emotional states deemed to most likely be affected by the test scenario were modelled in this study. They were namely anger, fear, happiness, and surprise. If required, the engine supports the addition of an arbitrary number of emotions.

Each emotion is assigned weighting factors and a decay factor that correspond to equation (7-3) as shown in Table 7-1. To better illustrate the system, the weighting factors were experimentally selected to result in significant changes in emotional values. The steady state value (w_0) of the emotions was set as zero for the same reason. Together these values define the temperament of an agent.

Table 7-1 Emotion constants

Emotion	w_0	w_1	w_2	γ
Anger	0	1.9	0.15	0.92
Fear	0	1.9	0.15	0.92
Happiness	0	2.3	0.15	0.92
Surprise	0	1.7	0.15	0.88

3. Components

The emotions engine is made up of three major components; short term memory, RPE module and the command modifier.

A. Short Term Memory

The emotional state of an agent is deemed to depend on its internal state and external stimuli such as events. Events are defined as interactions with the environment that result in a change of the ASM goal or subgoal. An example would be the robot encountering a danger ball while searching for food because this causes its goal to change from 'food' to 'avoid'. The emotions engine records such events in memory using a first in first out queue. It also stores the number of time steps taken to complete the ASM goal that led to the event along with the reward value for the event.

The system permits storage of an arbitrary number of events. Due to the decay component (γ) of the emotional model it was found that a memory length of six events was optimal and any additional events had a negligible impact.

a. Events

Table 7-2 lists all defined events along with their reward values. The values have been tweaked to obtain the required behavior from the agent. Table 7-3 lists the ASM goal and subgoal combination that results in a type of event being registered. For example, the event 'danger encountered' is triggered when the ASM's goal is set to 'avoid' and the subgoal is set to 'avoid_ball'. This occurs when the agent encounters a danger objective ball. A special case is the 'health too low' event which can be continuously triggered irrespective of the goal and subgoal as long as the health falls and remains below a certain threshold.

Table 7-2 Emotion modifiers

	Anger	Fear	Happiness	Surprise
Danger encountered	0	+12	-5	+10
Found Food	-1	-1	+5	0
Returned to safe area	-5	-5	+5	-5
Wall encountered	+3	0	0	+5
Health too low	0	+2	-2	0

Table 7-3 Events and their associated goals and subgoals

Event	Goal	Subgoal
Danger encountered	avoid	avoid_ball
Found Food	food	ball_eat
Returned to safe area	home	sleep
Wall encountered	avoid	avoid_wall
Health too low	(any)	(any)

B. Reward Prediction Error (RPE) Module

Momentary subjective wellbeing was found to depend not only on an event but errors in predicting the occurrence of said event. Unexpected events have a higher impact on the value of an emotion [7]. Since the robot's cognitive architecture lacked long term memory and learning capabilities, the average number of time steps taken for an event to occur in the past was used to predict when it could be expected to occur again. The difference between this prediction and the actual time taken for the event to occur is considered to be the RPE. The ASM handles time step tracking and passes this information to the emotions engine when an event occurs

a. Time step tracking

We have seen that each type of event is associated with a particular ASM goal and subgoal. This fact is used to track the number of time steps taken for an event to occur. The time step at which a goal is started, say goal X, is recorded as the start time. When eventually the subgoal changes to one that triggers an event, the number of time steps since the start time for which the system goal was set to X is considered to be the time taken for that event to occur. Due to the interrupts being able to temporarily change the goal in order to execute another plan, care had to be taken to not overestimate the time taken for an event to occur. Let us consider an example to illustrate this problem and show how the system avoids this pitfall:

- At time step 15 the robot's health falls below the threshold of low health
    - Goal is set to 'food' and the subgoal is set to 'ball_find'
    - The goal has just changed to 'food' and the system sets this time step as the start time.
- At time step 25 the robot encounters a boundary wall
    - Goal is set to 'avoid' and the subgoal is set to 'avoid_wall'
    - The goal has changed and the system stops attributing further time steps towards goal 'food'. So far 10 time steps have been spent under the goal of 'food'
- At time step 27 the wall has successfully been evaded
    - Goal is set to 'food' again and the subgoal is set to 'ball_find'
    - The system resumes counting time steps for the goal of 'food'. The count for the number of time steps spent under a goal of 'food' remains at 10
- At time step 33 the robot sees a food ball
    - Goal is set to 'food' and the subgoal is set to 'ball_seek'
    - We continue counting as the goal is still 'food'. The count is now increased by 6 for a total of 16 steps spent under the goal of 'food'
- At time step 53 the robot reaches the food ball
    - Goal is set to 'food' and the subgoal is set to 'ball_eat'
    - An event of the type 'found food' is registered
    - Since an event has been registered under the goal of 'food' the counting is stopped. The count is increased by 20 time steps for a total of 36 time steps spent under the goal of 'food'

Therefore, the number of time steps taken for the event ’food found’ to occur is 36. If only the start time (15) and end time (53) was considered, the system would have erroneously concluded that it took 38 time steps.

At the start of an agent’s life each type of events is assumed to take 25 time steps to occur. An exception to this is the event of type ‘Health too low’ which is considered to take 300 time steps to occur. These values were used as they were found to be the average number of time steps taken in the majority of cases. As events of a type occur during operation, this prediction is updated. In our example it took 36 steps for the ‘found food’ event to occur. Initially it was predicted it would take 25 steps to find food. The prediction is updated using (7-4).

$$prediction = \frac{previous\ prediction + time\ taken}{2} \tag{7-4}$$

The new prediction would be that it will take 30 time steps for the event of type ‘food found’ to occur when searching for food in the future. The RPE module stores predictions for all events defined.

C. Command Modifier

As described in Chapter 7.3.A, the short term memory stores the last 6 events, time steps taken, and the reward values. The time steps taken and the event type are sent to the RPE module which calculates the RPE. Using equation (7-3) the instantaneous value of the four emotions are calculated and in combination define the emotional state of the robot. These calculations are carried out every time step regardless of if an event has been registered.

The command modifier allows the emotions engine to modify the internal state and the commands sent to the robot using conditional logic or statements. In this study we have chosen to apply the following modifiers,

- $Speed = current\ speed + value_a + value_f - value_h$

Where $value_a$, $value_f$, and $value_h$ denote the numeric values of the emotions anger, fear, and happiness respectively. This modifier increases the speed of the robot when the value of anger or fear increases and decreases it when happiness increases. This shows how all emotions can compete against each other to affect behavior.

- $if\ speed < 90\ then\ speed = 90$

This modifier prevents the speed of the robot from reducing too much.

- $if\ value_f > 40\ then\ randomly\ stop\ the\ robot's\ motion$

This simulates a timid robot, by randomly sending it stop commands if the value of fear becomes greater than 40. The is similar to the tendency of an animal to momentarily freeze when frightened.

- $if\ surprise > 40\ then\ make\ a\ 530°\ turn$

This simulates a surprised robot, by making it turn around if the value of surprise gets larger than 40. This is similar to an animal being startled and a robot spinning around was not only amusing, but easily observed.

Chapter 8
Results

The effects of the different type of events on the emotions and robot state variables are considered in Chapter 8.1, the RPE module is considered in Chapter 8.2, and varying temperaments are considered in Chapter 8.3. Events are signified on the relevant plots using the letters shown in Table 8-1. A negative RPE denotes an unexpected event while a positive RPE denotes an event that was predicted to occur sometime in the past.

Table 8-1 Events and their signifiers

Event	Denoted by
Danger encountered	a
Found Food	f
Returned to safe area	b
Wall encountered	w
Health too low	h

1. Effect of Events on the Emotional State

To clearly illustrate the effect of the different types of events three test cases were considered; only food balls, only danger balls, and equal number of both type of balls. All three test cases were experimentally run for 300 time steps.

A. Six Food Balls and Two Safe Areas

This test highlights the increase in happiness when the robot eats food and reaches a safe area. Figure 8-1 shows the test setup, Figure 8-2 shows the variation in the individual emotions and Figure 8-3 shows the variation in the robot's speed (dependent on the emotions of anger, fear and happiness) and its health. Table 8-2 lists all events that were registered along with their time steps and the calculated RPE.

Table 8-2 RPE of events for a scenario consisting of food balls and safe areas

Step	14	30	51	61	90	91	92	93	94	102	114	228	262	263	264	265	290
Event	w	w	w	w	f	f	f	f	f	w	b	w	f	f	f	f	w
RPE	-11	-3	4	-9	55	29	16	9	6	27	-6	99	-44	-21	-9	-3	-14

From the data we can infer that:

- At time step 10 the robot's health drops below the threshold of 75 and it started to seek food.
- It encountered the boundary walls at time steps 14, 30, 51 and 61. Due to the RPE, the impact of the same type of event on the emotions was varied dependent on the frequency of occurrence. The first three encounters were 14, 15, and 20 time steps apart and had an RPE of -11, -3, and 4. This is seen as a reduction in the impact of wall events on the emotions. The fourth encounter occurred 10 time steps later and had a larger impact as the robot did not 'expect' to encounter a wall so soon. The RPE this time was -9.
- The robot reached a food ball at time step 90 and took 5 time steps to completely recharge its health as can be seen in the spike in health to a value of 104 at time step 94. There was an increase in happiness and reduction in anger and fear during this period.
- Once its health was full, the robot sought out a safe area during which time its emotions decayed to their steady state values.
- A wall was encountered at time step 102 that spiked anger and surprise. Since it had been 41 time steps (RPE of 27) since the last time a wall was encountered, its impact was greatly reduced.
- At step 114 a safe area was reached increasing happiness while reducing fear, anger, and surprise. For approximately the next 125 steps the robot rested in the safe area until the value of its health dropped below the threshold at time step 225.
- The robot encountered a wall at step 228. The RPE was 99 as it had been 126 time steps since the last wall event occurred. This caused the RPE module to

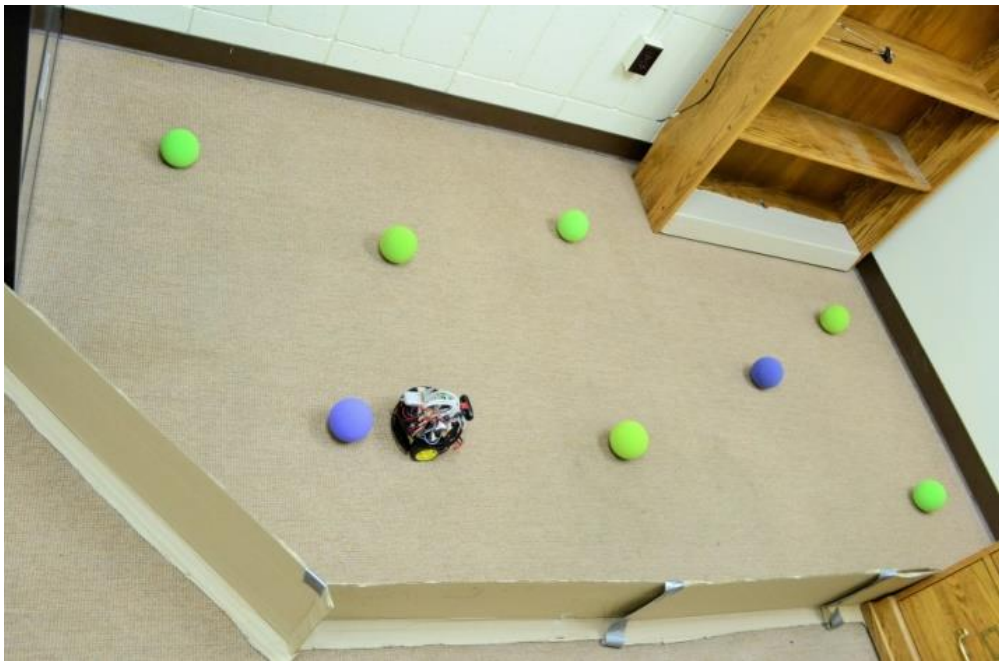


Figure 8-1 Scenario consisting of food balls and safe areas

reduce the effect of this event on the emotions by a large enough amount to cause a net reduction in anger and surprise.

- When the robot reached a food ball at step 262 the effect on the emotions was greater than the first time. The first time it took the robot 80 time steps (RPE of 55) to find a food ball whereas this time it found one in 37 steps (RPE of -44). The change in the emotions led to a large enough reduction in robot speed that it was capped at 90.
- At time step 290 another wall was encountered this time taking 62 time steps (RPE of -14) and affected the emotions as expected.

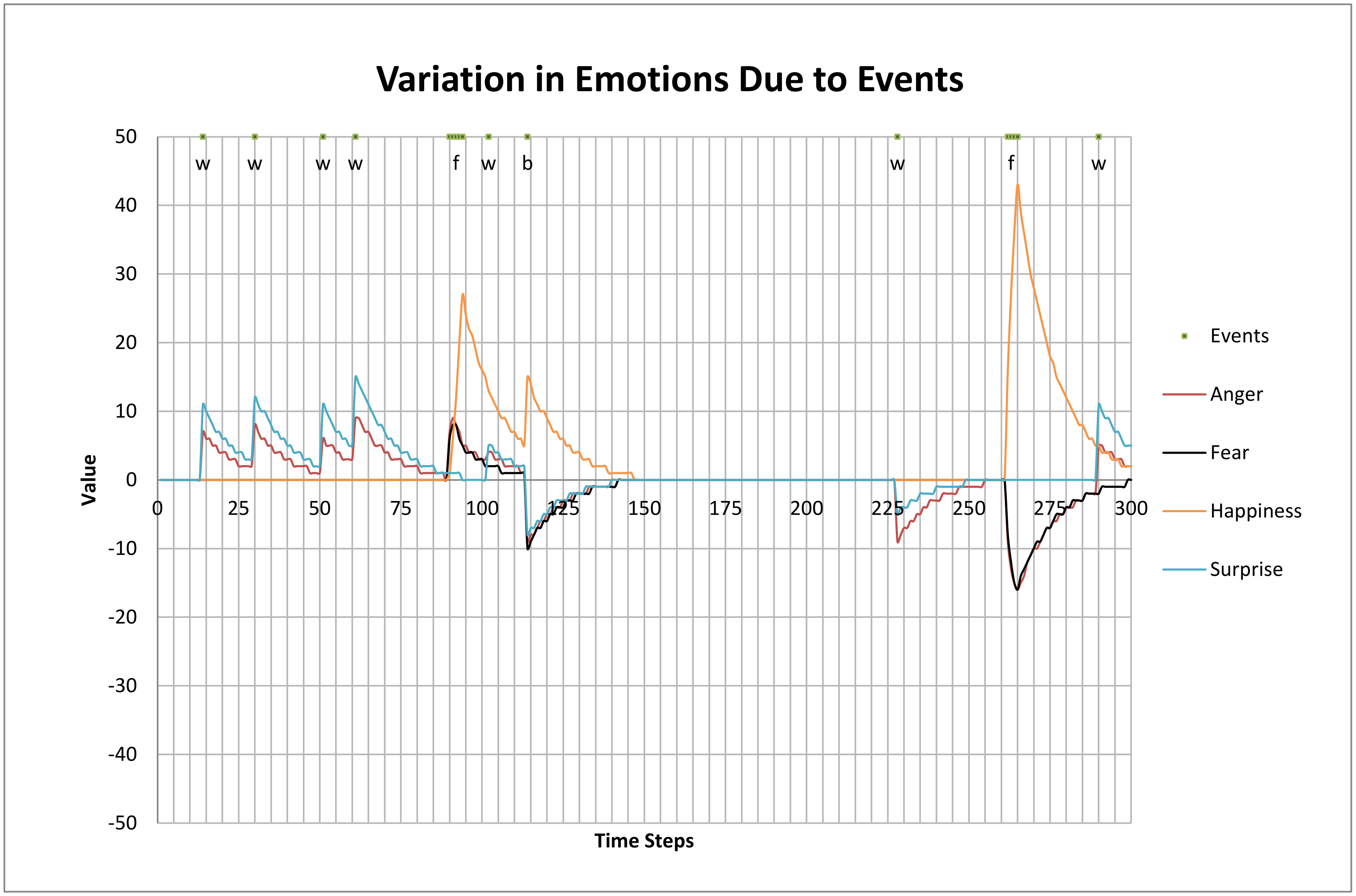


Figure 8-2 Shows the variation in the values of the four emotions for a test scenario consisting of 6 food balls and 2 safe areas

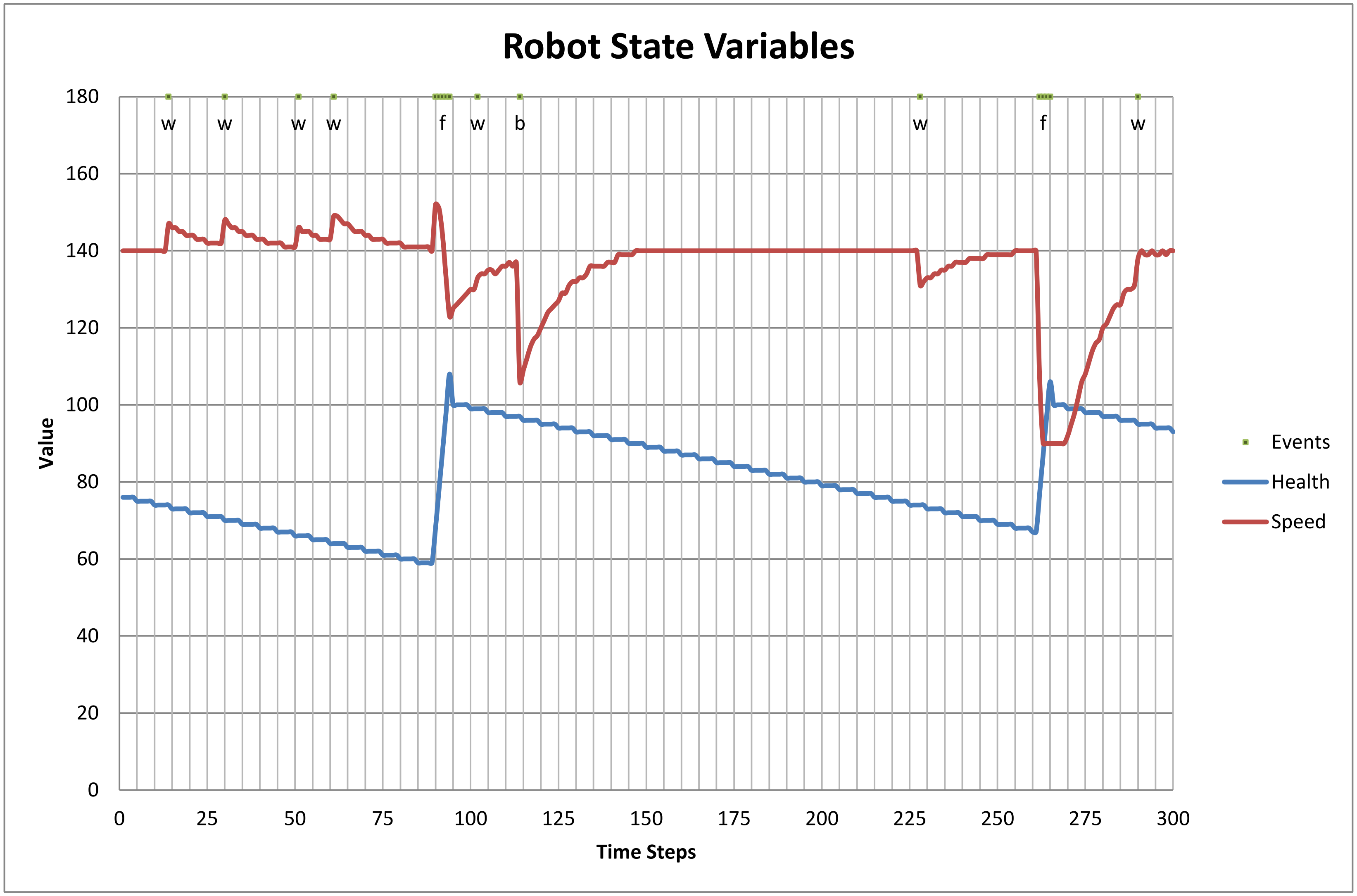


Figure 8-3 Shows the variation in the speed and health of the robot for a test scenario consisting of 6 food balls and 2 safe areas

B. Six Danger Balls and Two Safe Areas

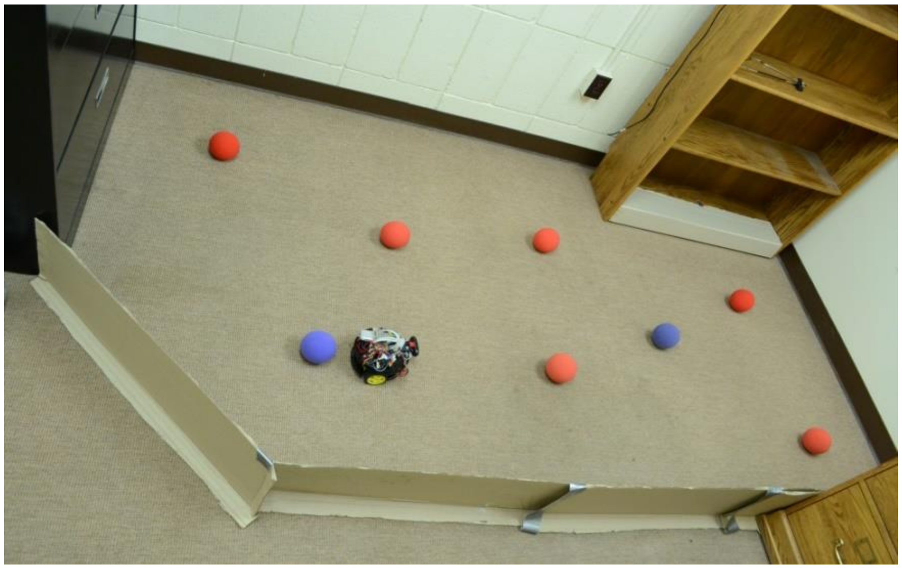


Figure 8-4 Scenario consisting of danger balls and safe areas

This case highlights the interaction with danger balls which leads to an increase in fear and surprise, and a reduction in happiness. Figure 8-4 shows the test setup. Figure 8-5 shows the variation in the individual emotions and Figure 8-6 shows the variation in the robot's speed and its health. Table 8-3 lists all events that were registered along with their time steps and the calculated RPE.

It is to be noted that the event of 'Health too low' that was registered every time step from 185 to 300 is omitted from the table. It is denoted on Figure 8-5 and Figure 8-6 as a yellow line.

Table 8-3 RPE of events for a scenario consisting of danger balls and safe areas

Step	**15**	**21**	**28**	**35**	**42**	**61**	**70**	**78**	**85**	**91**	**97**	**104**	**110**	**114**
Event	w	w	w	w	w	w	w	w	a	a	w	a	w	w
RPE	-10	-14	-6	-3	-1	12	-4	-3	60	-49	10	-17	-1	-9
Step	**149**	**167**	**175**	**183**	**191**	**197**	**216**	**222**	**235**	**242**	**248**	**260**	**276**	**289**
Event	w	a	w	w	a	w	w	w	w	a	w	w	w	w
RPE	27	42	5	-15	-18	-1	5	-1	2	18	1	0	4	-1

From the data we can infer that:

- Similar to the previous case once the health dropped below the threshold of 75 at time step 10, the robot started searching for food balls.
- It encountered a boundary wall at time steps 15, 21, 28, 35, and 42. The RPE led to a lower and lower impact with subsequent wall event seen as the value of anger and surprise plateauing. The next time it had to avoid the wall was at time step 61 (RPE of 12) nineteen steps later. The impact of this event on the emotions was diminished. It then had to avoid the walls again at time steps 70 and 78 and the RPE resulted in the peaks leading to a net increase in anger and surprise.
- A danger ball was encountered at time step 85 leading to a spike in all emotions but happiness. The robot encountered the danger ball again at step 91 and this time the corresponding emotional change was much larger due to the RPE.
- The robot continued to encounter the walls and danger balls. The danger ball it encountered at time step 167 had a much smaller impact on the emotions as it had been 66 steps since the previous time a danger ball had been encountered. The RPE was 42 for this event.
- Since there were no food balls the robot could not recharge its health leading to a state of extremely low health. At time step 185 the health dropped below the low health threshold of 40. This resulted in the event of ‘health too low’ being triggered from time step 185 to 300 (‘h’ denotes the start of the event and the yellow line the duration). By default the system assumes that the health will not drop low enough to trigger this event. So when it does occur, the effects of the RPE initially cause a large spike in fear and happiness. This was seen from time step 185 to 190. These emotions then decayed to a steady state value of 15 for the rest of the test run.
- During the period of low health the system encountered a danger ball at time steps 190 and 241. It also encountered the boundary walls 8 times. These events as well as the low health event cumulatively affect the emotions during this period.

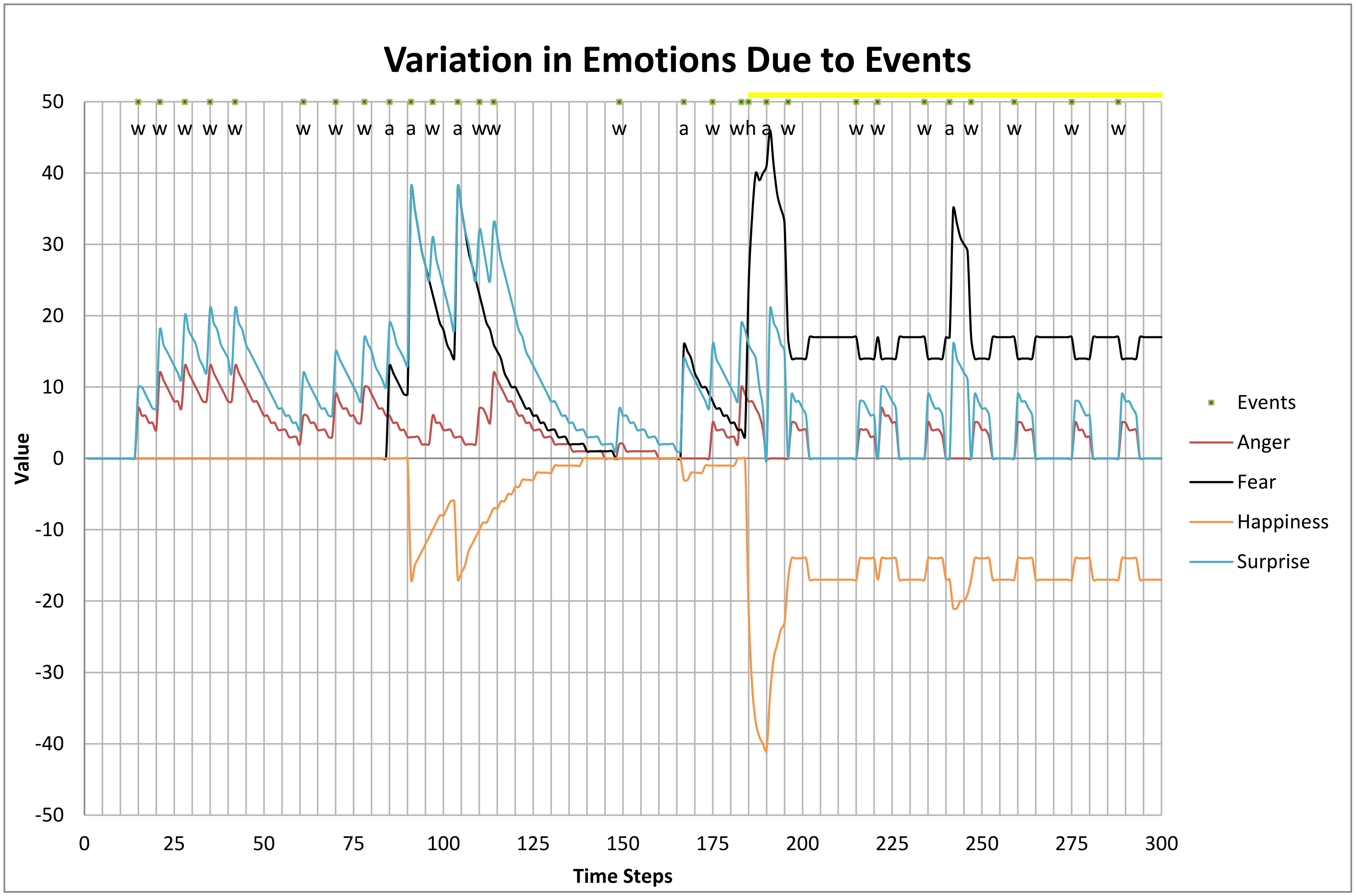


Figure 8-5 Shows the variation in the values of the four emotions for a test scenario containing 6 danger balls and 2 safe areas

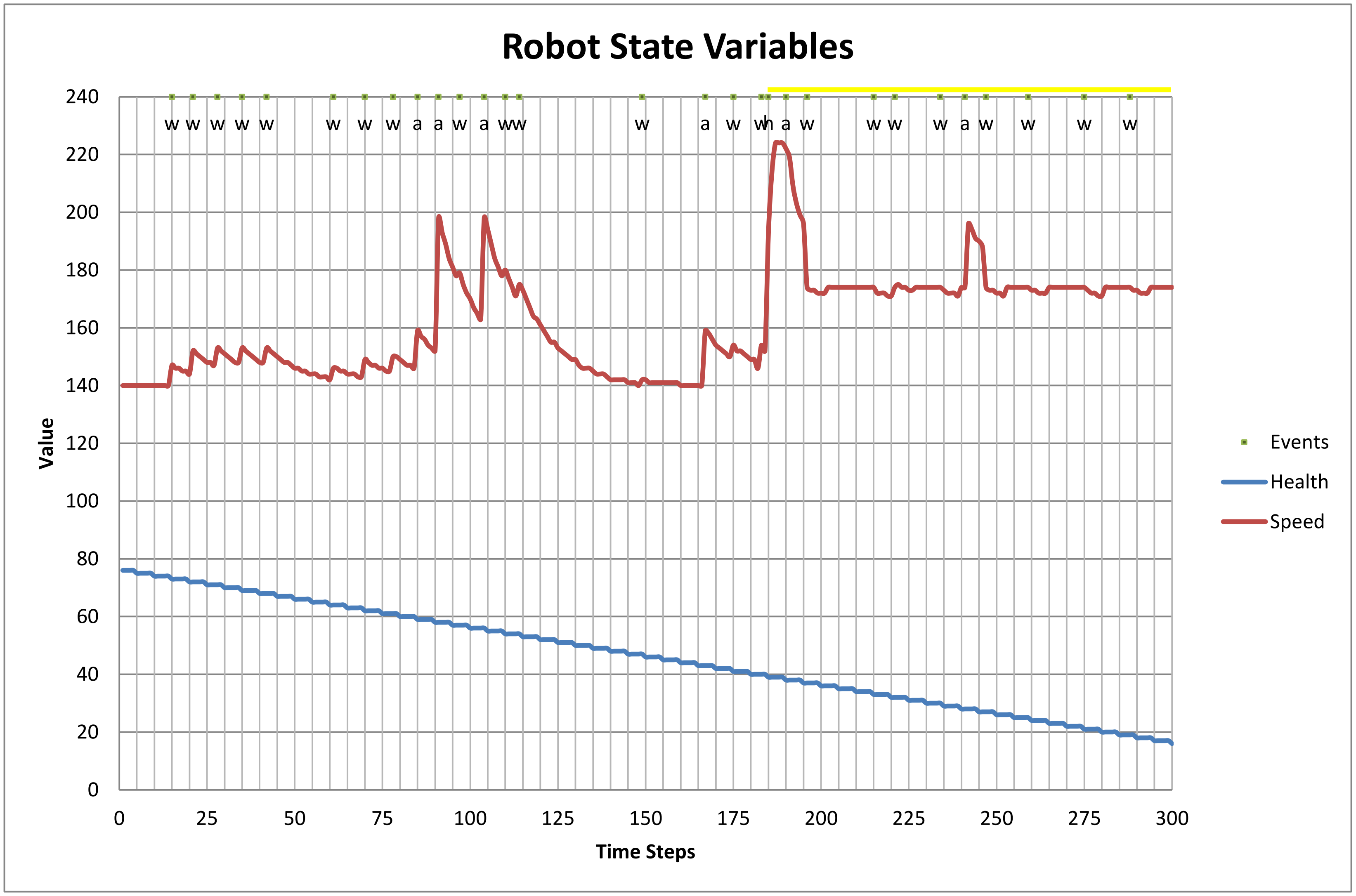


Figure 8-6 Shows the variation in the speed and health of the robot for a test scenario containing 6 danger balls and 2 safe areas

C. Three Food Balls, Three Danger Balls, and Two Safe Areas

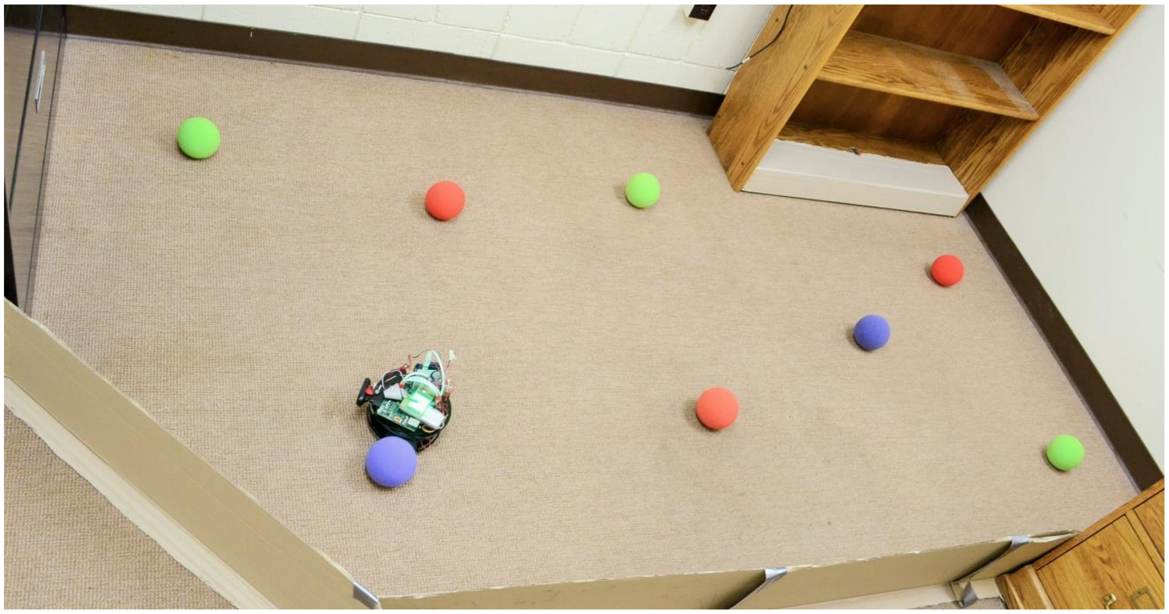


Figure 8-7 A scenario containing two safe areas, and three food and danger balls

The final scenario illustrates how the emotions and robot state vary during the course of a scenario containing all three types of balls. Figure 8-7 shows the test setup. Figure 8-8 shows the variation in the individual emotions whereas Figure 8-9 shows the variation in the robot's speed and its health. Table 8-4 lists all events that were registered along with their time steps and the calculated RPE.

Table 8-4 RPE of events for a scenario containing two safe areas and three food and danger balls

Step	**22**	**23**	**24**	**35**	**43**	**51**	**59**	**85**	**92**	**107**	**146**	**158**	**181**	**204**
Event	f	f	f	w	w	w	w	a	w	w	a	w	w	w
RPE	-13	-5	-1	10	-22	-11	-5	60	23	-6	6	33	-11	-4
Step	**214**	**217**	**225**	**231**	**241**	**242**	**248**	**249**	**255**	**256**	**260**	**266**	**294**	
Event	b	w	w	w	f	w	f	w	f	w	f	f	w	
RPE	164	-14	-11	-7	12	2	13	-3	14	-1	12	12	31	

From the data we can infer that:

- Once the health dropped below the threshold of 75 at time step 10, the robot went searching for food and found it at time step 22 that is in 12 steps. The RPE led to a larger spike in happiness and a large reduction in robot speed.
- During its search for a safe area it had to avoid the boundary walls, food balls, and the danger balls each of which affected the emotions as seen in the previous scenarios.
- The robot finally found a safe area at time step 214. The initial prediction by the RPE module was that a safe area would be reached in 25 time step. In this case since that led to an RPE of 164, instead of an increase in happiness and reduction in anger, surprise, and fear we see the opposite effect.
- Since it took a large number of time steps to return to a safe area by that time its health had dropped below the threshold of 75. This caused it to immediately start searching for a food ball.
- At around time step 241 there was an anomaly where the robot incorrectly identified a food ball as a boundary wall momentarily before correctly recognizing it as a food ball. This is seen as a food event and a wall event one after another a few times on Figure 8-8.

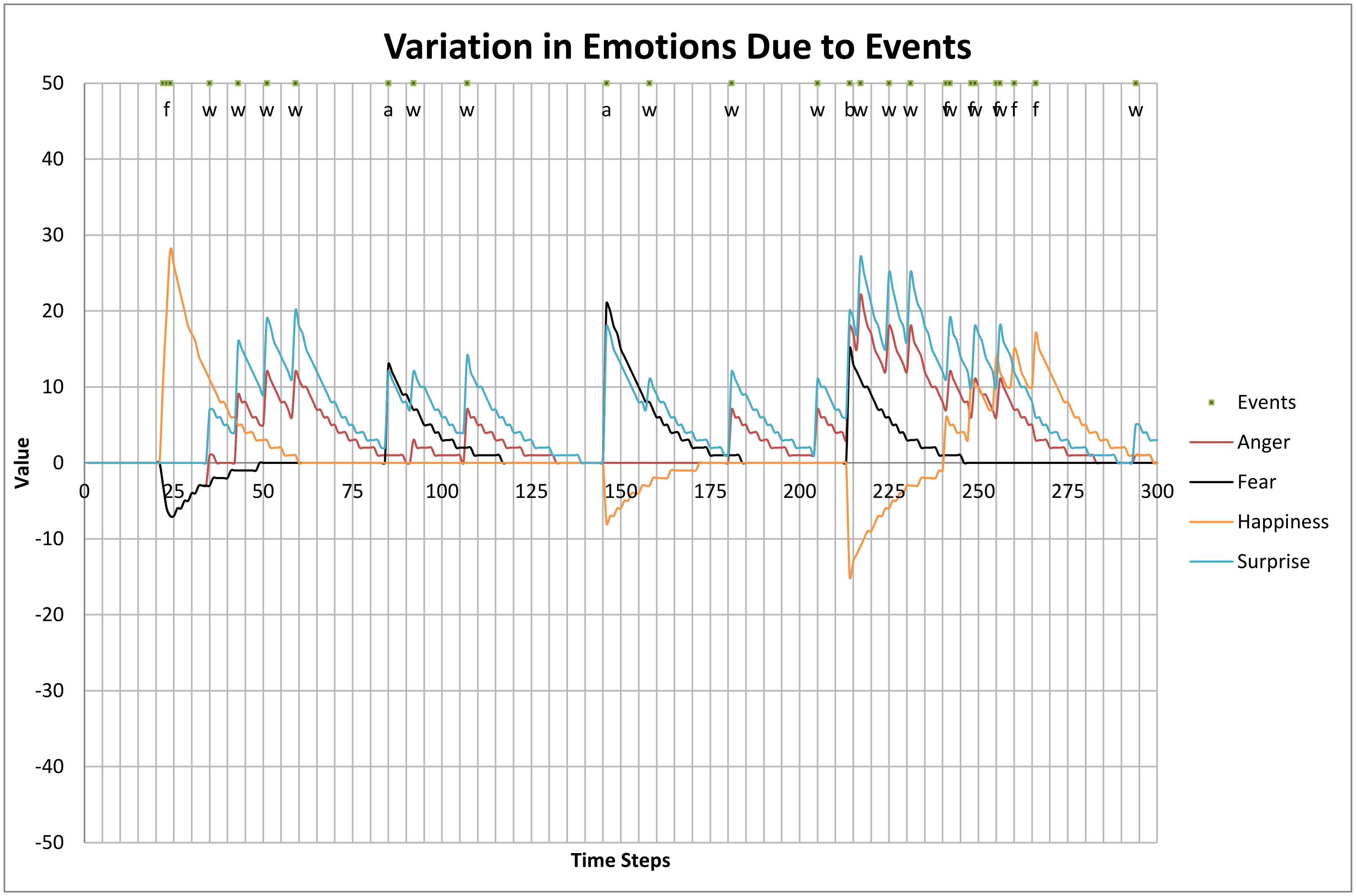


Figure 8-8 Shows the variation in the values of the four emotions for a test scenario containing 2 safe areas, and 3 food and danger balls each

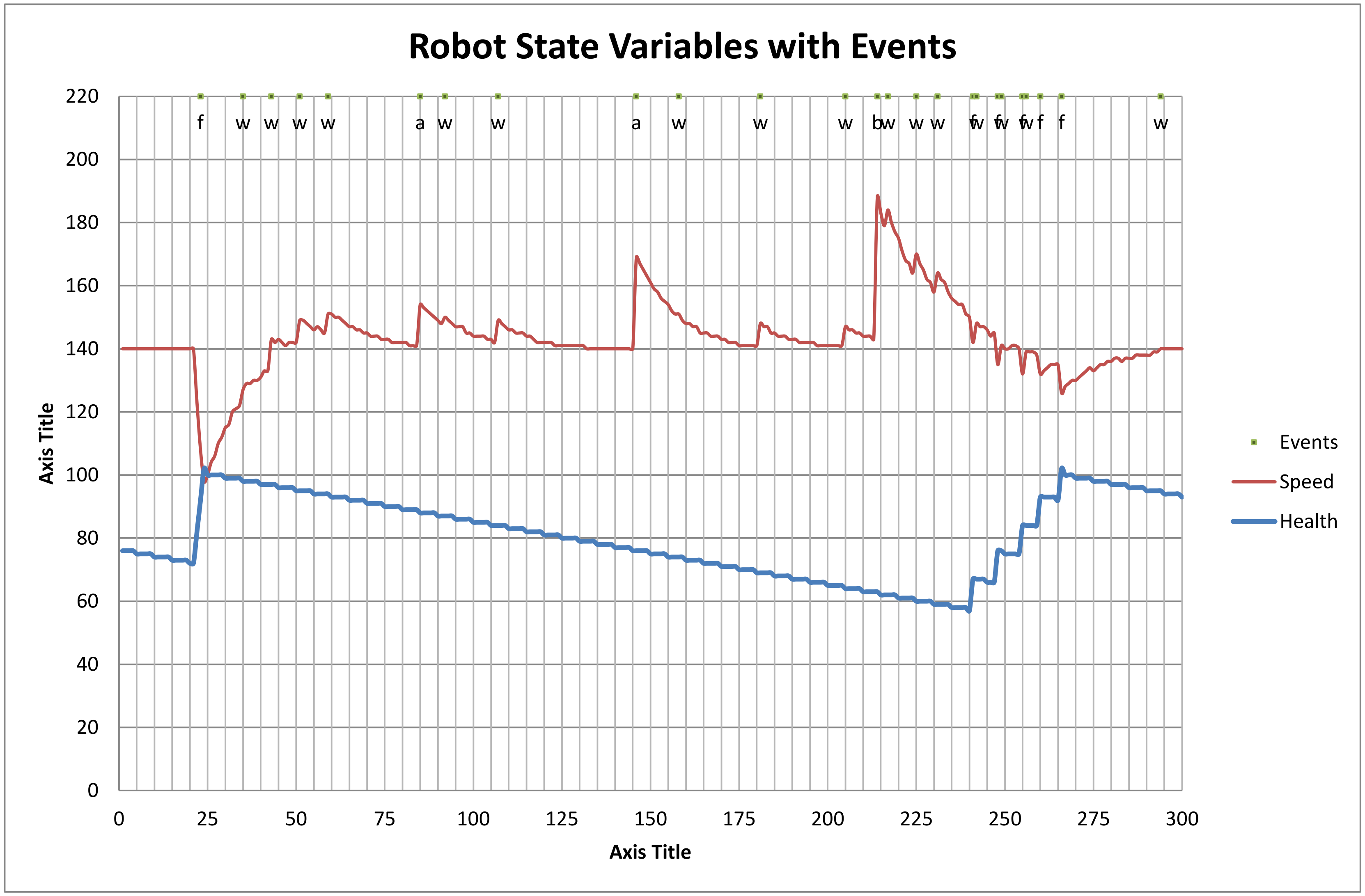


Figure 8-9 Shows the variation in the speed and health of the robot for a test scenario containing 2 safe areas, and 3 food and danger balls each

2. Effect of the RPE Module

Unless we compare the exact same scenario with the RPE module enabled and disabled, it is very difficult to observe the emotional state adjustments caused by the RPE module. The dynamic plans of the action selection mechanism implemented in this study are not affected by the emotional state of the robot. On the contrary, the ASM selects plans which in turn affect the emotional state based on the events that occur during the execution of the plans. By recording all ASM, event, and state data, it was possible to simulate the emotional state. This data was collected for the test conducted in Chapter 8.1.C (scenario consisting of three food balls, three danger balls, and two safe areas) and the emotional state that would have resulted with the RPE module disabled has been simulated.

Figure 8-10, Figure 8-11, Figure 8-13, and Figure 8-12 compare the variations in anger, fear, happiness, and surprise with and without the RPE module respectively. Using Table 8-4 we study some key events that illustrate the effect of the RPE module:

Food event at time step 22 (RPE of -13)

This event happened sooner than predicted by 13 time steps. Due to this fact, the RPE module caused this event to have a larger impact. Anger and fear were reduced to greater extent and happiness was increased further than would have without the RPE module.

Wall event at time step 35 (RPE of +10)

This event happened later than predicted by 10 time steps. Thus its impact was reduced. This is seen as anger and surprise values not being as large as they would have without the RPE module.

Wall event at time step 43, 51, 59 (RPE of -22, -11, -5)

We observe that the impact of the wall at step 43 is greater than that at step 51 and so on. When the RPE module was disabled, the impact of all 3 wall events was exactly the same.

Danger ball at time step 85 (RPE of 60)

Since the robot encountered a danger ball much later than it had anticipated, the effects of this event on fear and happiness were reduced when the RPE was active as compared to when it was not.

Safe area ball at time step 214 (RPE of 164)

The extremely large prediction error meant that instead of an increase in happiness that would have been expected when the robot returned to a safe area, there was a reduction with the RPE module switched on. There was also an increase in anger and surprise, basically the opposite of the effect this event would have had on the emotions had the RPE module been disabled.

We can conclude from this that the RPE module is crucial in modelling the psychological effect of expectations governing the effect of an event. Without it, an event would affect the emotional state exactly the same way regardless of when it occurs.

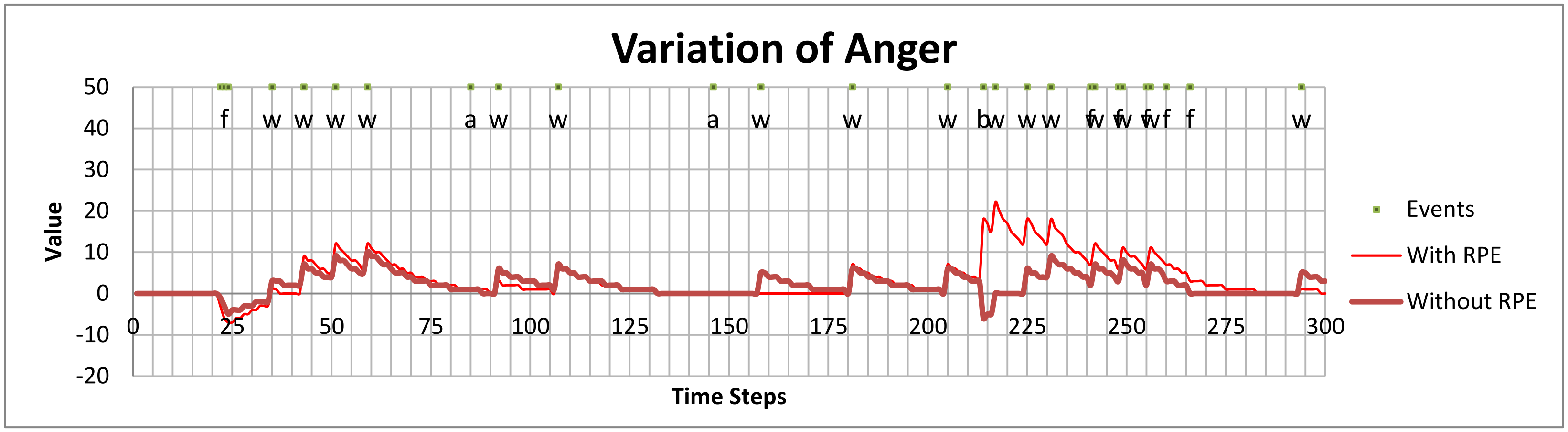


Figure 8-10 Comparison of the variation of anger with and without the RPE module for a test scenario containing 2 safe areas, and 3 food and danger balls

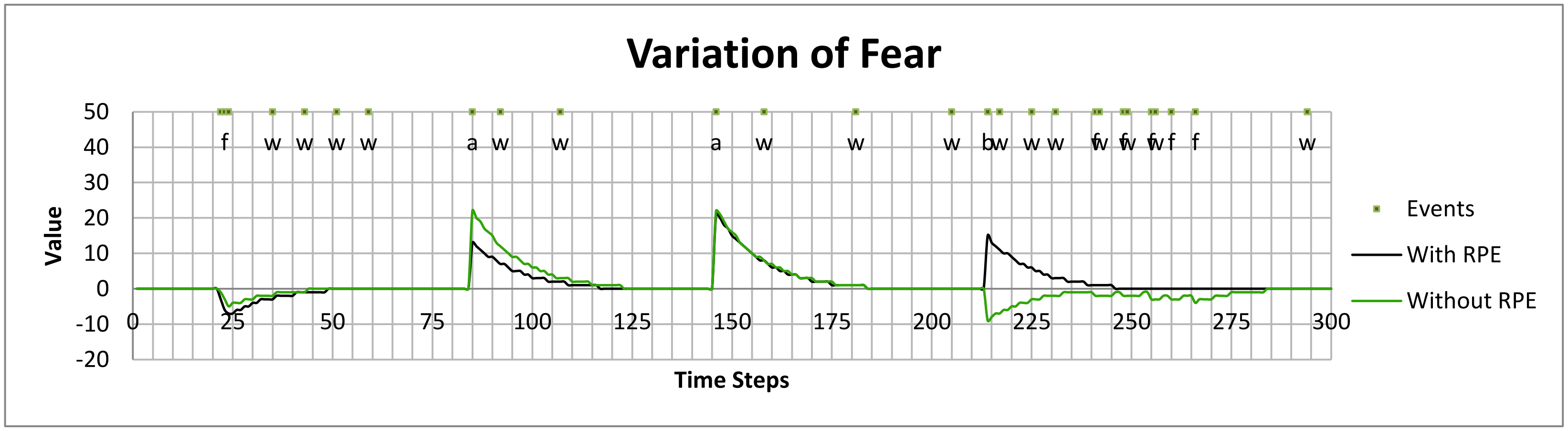


Figure 8-11 Comparison of the variation of fear with and without the RPE module for a test scenario containing 2 safe areas, and 3 food and danger balls

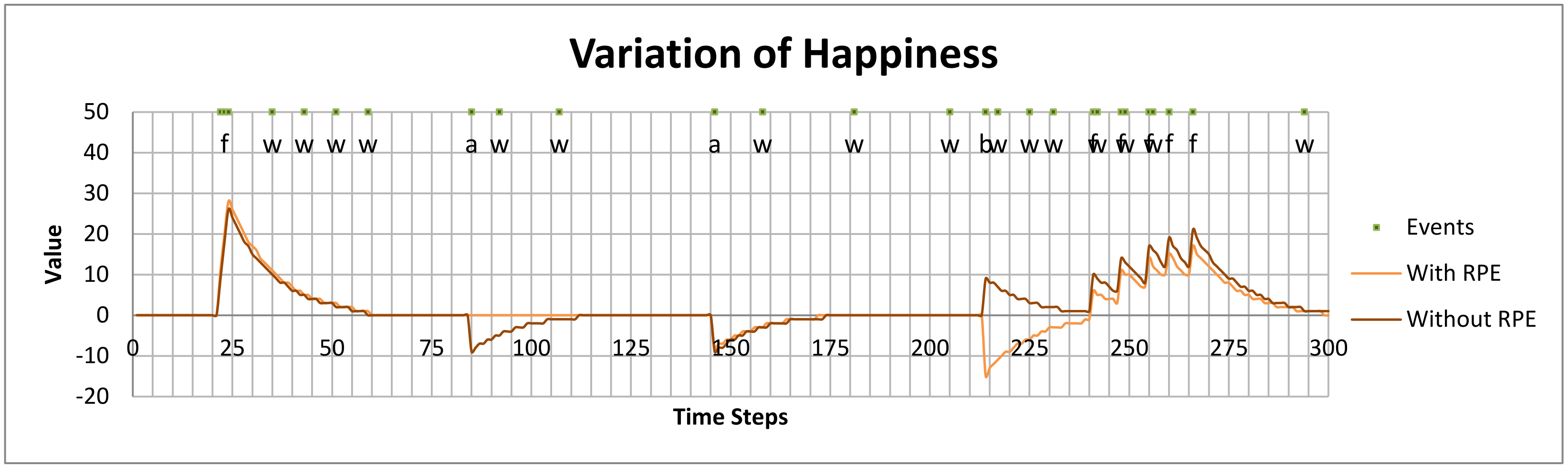


Figure 8-13 Comparison of the variation of happiness with and without the RPE module for a test scenario containing 2 safe areas, and 3 food and danger balls

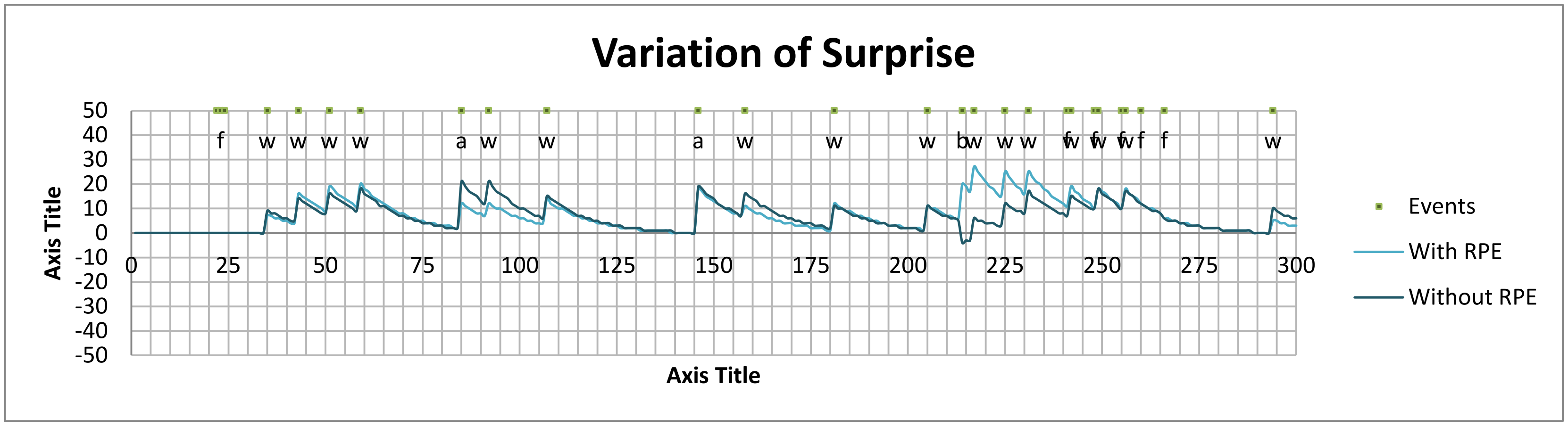


Figure 8-12 Comparison of the variation of surprise with and without the RPE module for a test scenario containing 2 safe areas, and 3 food and danger balls

3. Effect of Varying Temperaments

To compare how the emotional state differed based on temperaments we once again need to consider the exact same test run for all the temperaments studied. The same scenario as used for the RPE module comparison described in Chapter 8.1.C (scenario consisting of three food balls, three danger balls and two safe areas) was considered.

As discussed in Chapter 7.2 temperaments are defined by the weighting factors shown in Table 7-1. Four different temperaments of irate, timid, cheerful and anxious were considered. In each case, the weighting factors for only one emotion were changed keeping the rest constant.

Irate: The weighting factors used for anger were:

Emotion	w_0	w_1	w_2	γ
Anger	5	2.3	0.15	0.98

Timid: The weighting factors used for fear were:

Emotion	w_0	w_1	w_2	γ
Fear	5	2.3	0.15	0.98

Cheerful: The weighting factors used for happiness were:

Emotion	w_0	w_1	w_2	γ
Happiness	5	2.3	0.15	0.98

Anxious: The weighting factors used for surprise were

Emotion	w_0	w_1	w_2	γ
Surprise	2	2.0	0.15	0.97

The weighting factors for the anxious temperament differed as otherwise, the value of the emotion surprise was found to exceed the allowed range of -50 to +50.

Compared to the emotional state changes shown in Figure 8-8, only one emotion would be affected differently by the temperaments chosen to be simulated and so only the affected emotion is compared against that of the neutral temperament in the figures. Figure 8-14, Figure 8-15, Figure 8-16, and Figure 8-17 show the variation of anger, fear, happiness, and surprise for an irate, a timid, a cheerful, and an anxious temperament respectively. Figure 8-18 then compares the robot state variable speed for all four temperaments against the neutral temperament.

We observe that the steady state value of the affected emotion is offset from 0 by the value of the weighting factor w_0. The factor w_1 controls the instantaneous effect of an event on the emotions. As γ is increased we see that an emotion takes longer to return to its steady state value. Thus an irate temperament causes the robot to have a higher steady state value of anger and remain angry for longer once an event angers it. The same holds true for the other temperaments. In a practical application we would use a combination of these 'pure' temperaments to recreate the required behavior.

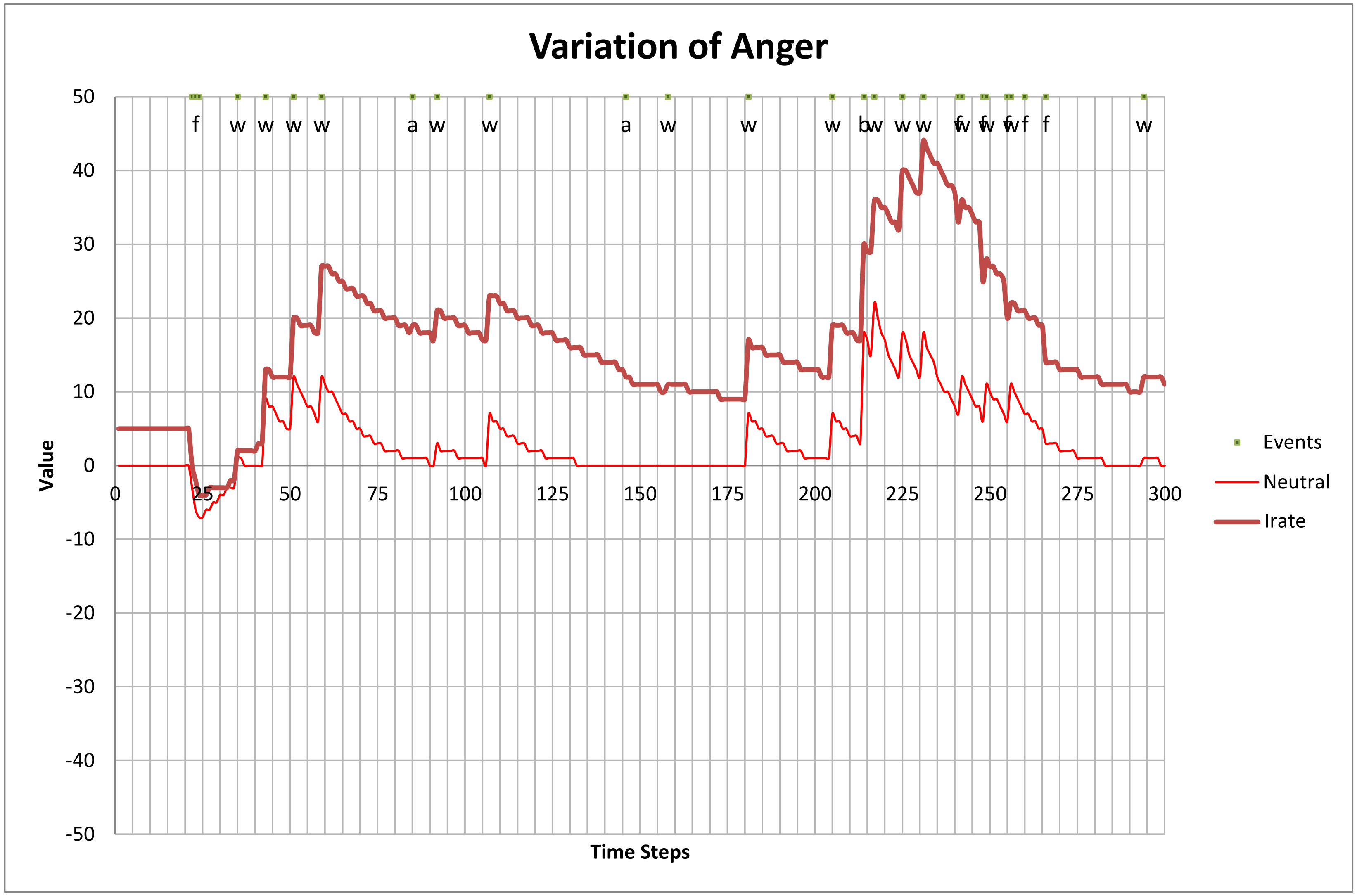


Figure 8-14 Comparison of the value of the emotion anger for an irate temperament with a neutral temperament

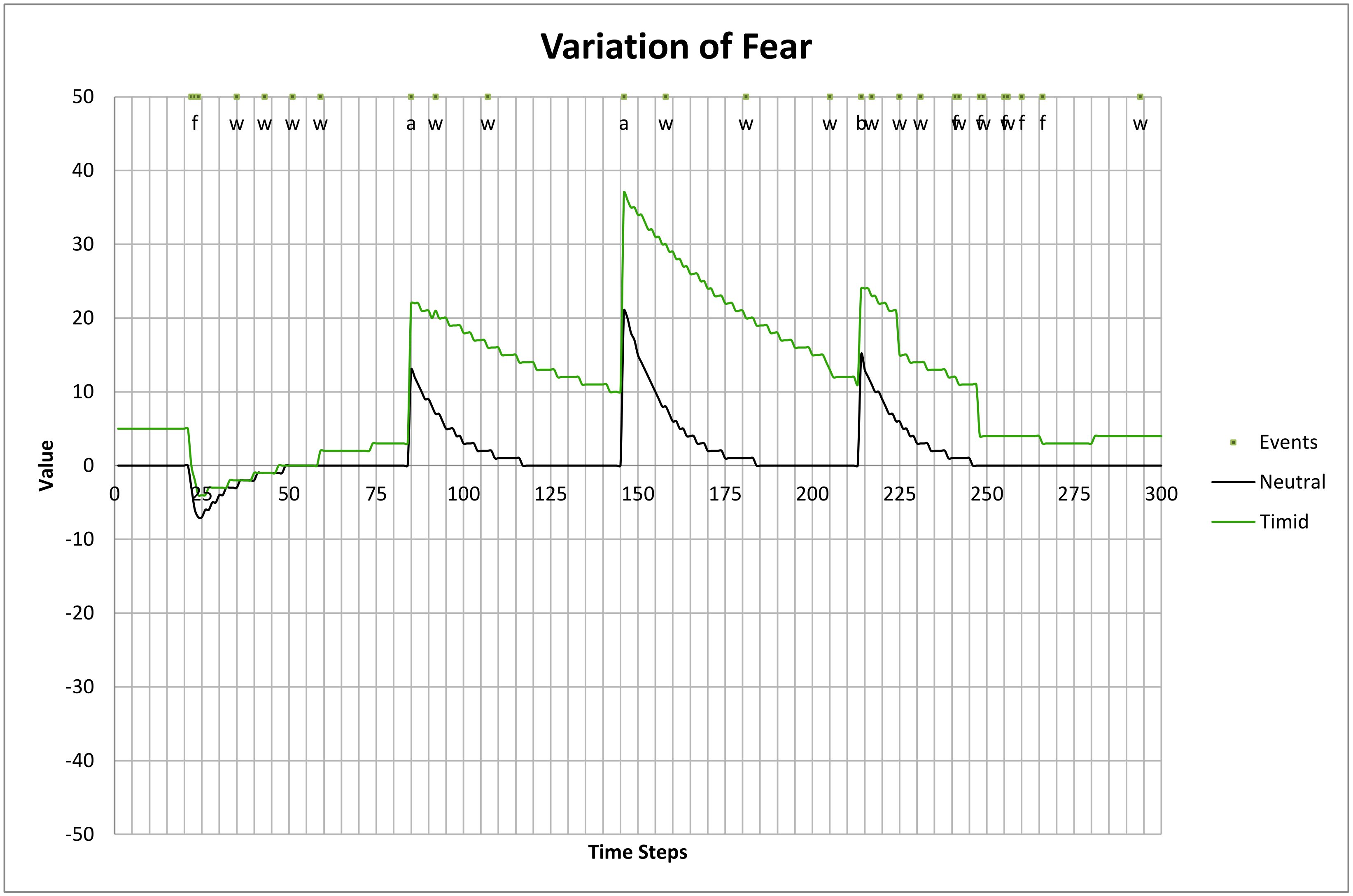


Figure 8-15 Comparison of the value of the emotion fear for a timid temperament with a neutral temperament

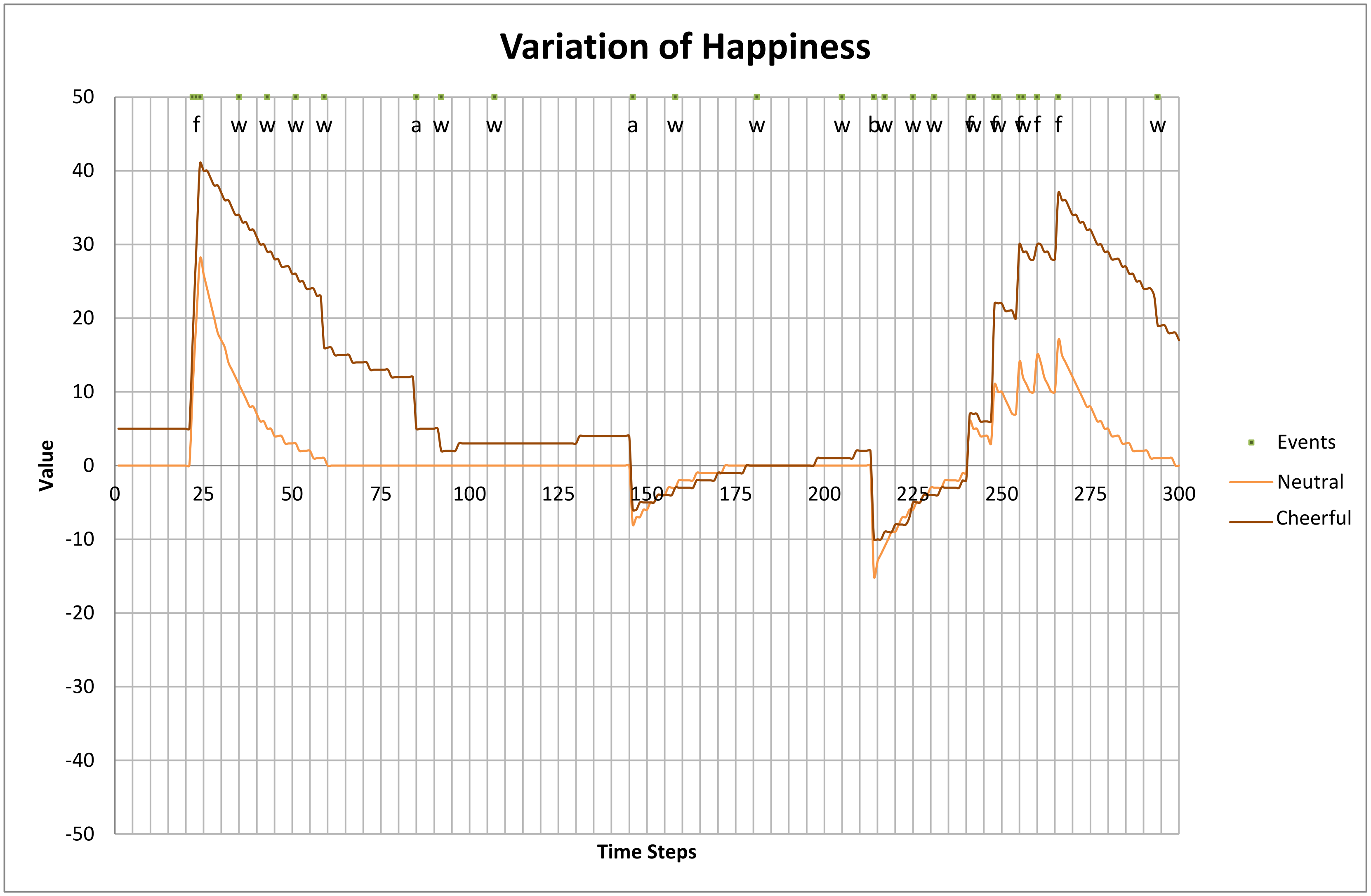


Figure 8-16 Comparison of the value of the emotion happiness for a cheerful temperament with a neutral temperament

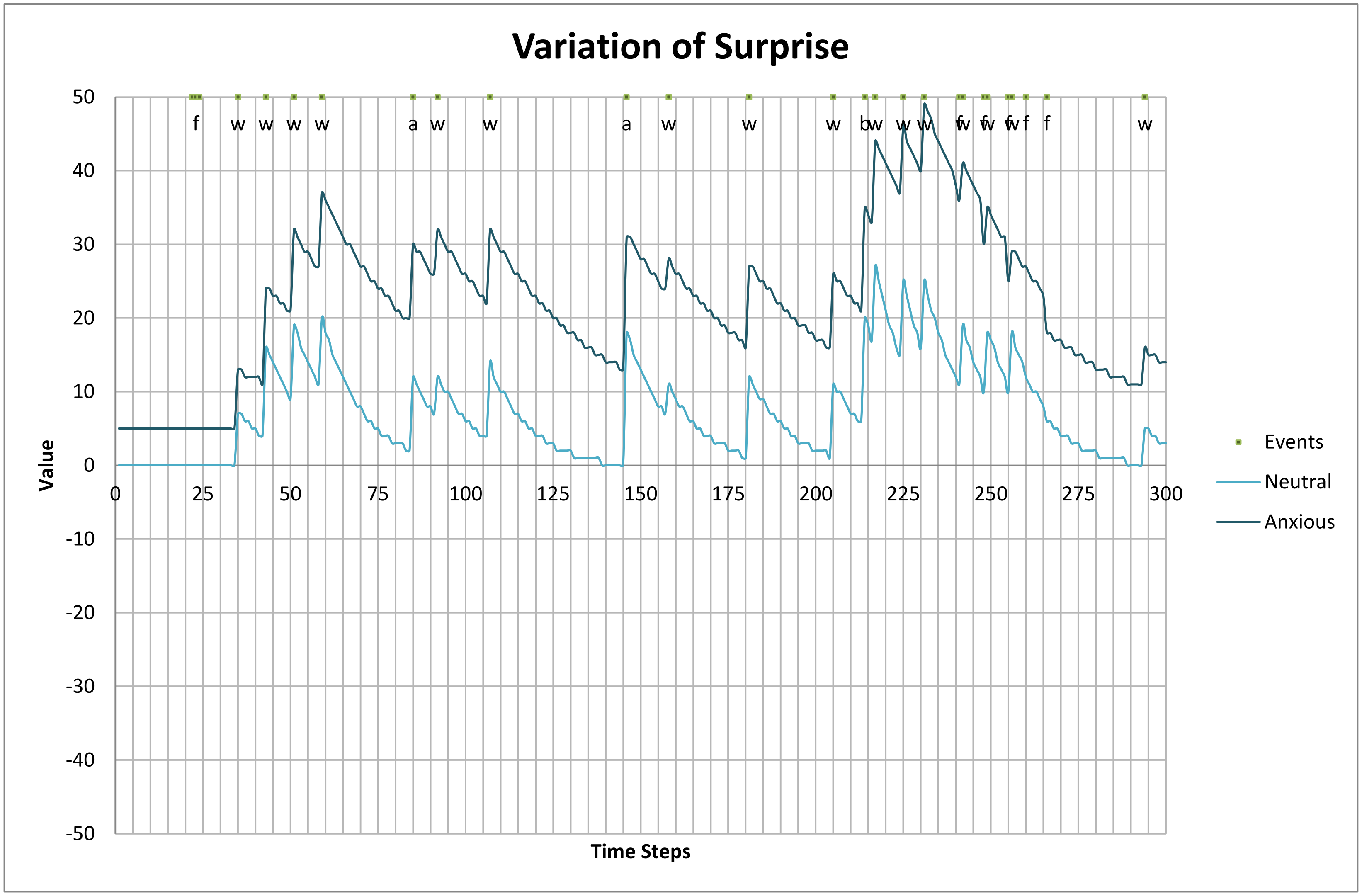


Figure 8-17 Comparison of the value of the emotion surprise for an anxious temperament with a neutral temperament

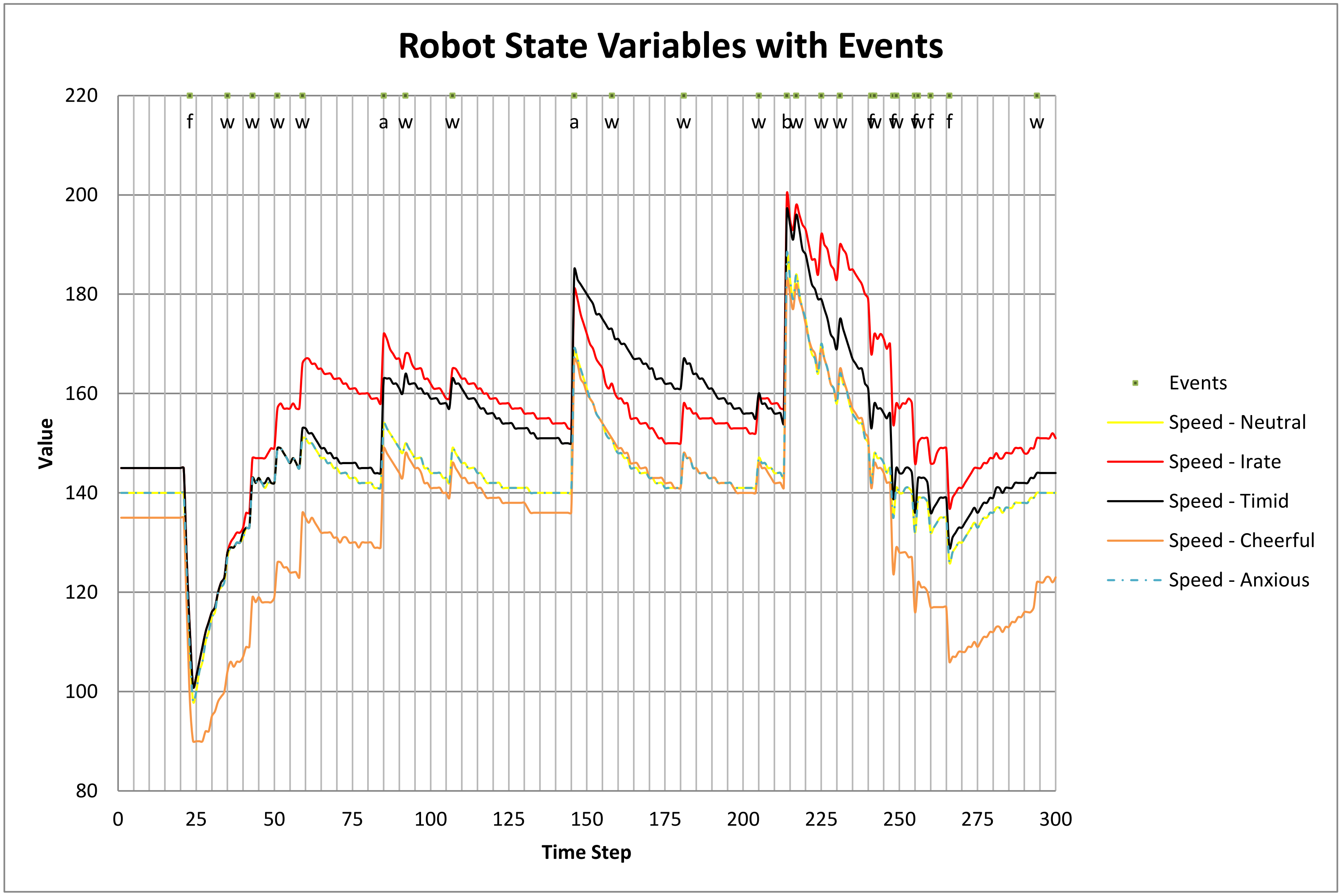


Figure 8-18 Comparison of the value of the robot state variable speed for the irate, timid, cheerful, anxious, and neutral temperaments

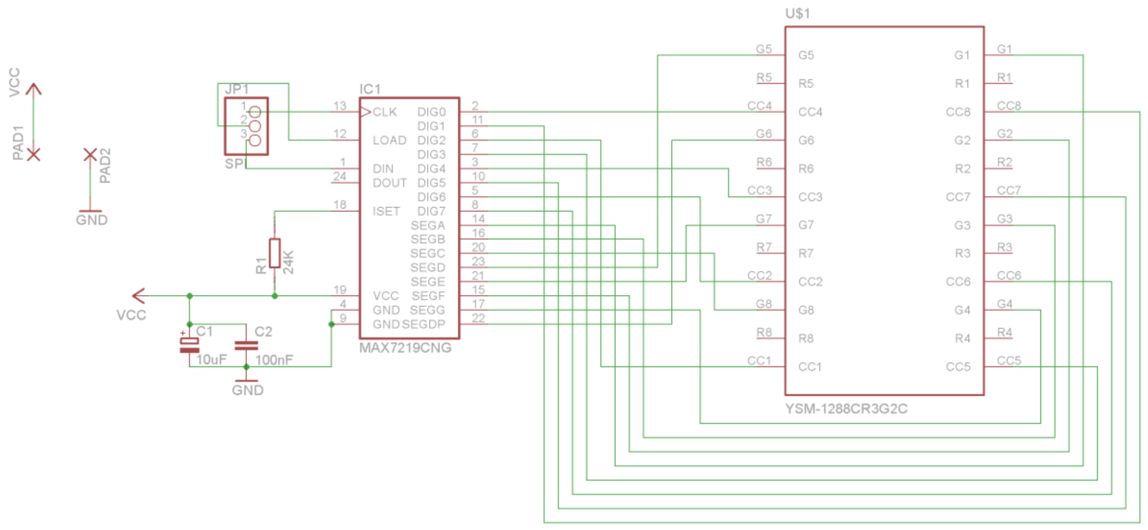
Chapter 9
Conclusions

By adapting a model for momentary well-being [7] as done by Long et al. [8], [9], and incorporating a reward prediction error, an action selection mechanism with an integrated emotions engine has been implemented. In addition, a low cost robot was developed simultaneously to be used to test this system. The ASM developed was a hybrid of a goal based and a dynamic planning action selection mechanism. Each goal was associated with subgoals, and each subgoal with a dynamic plan. Different events were defined that affected the individual emotions and the ASM provided means by which the emotional state could be used to modify the internal state of the robot through means of command modifiers. Command modifiers also allowed the modification of dynamic plans and the value of emotions to be coupled through suitable models. Temperaments and emotional variability was defined using a matrix of constants. Varying the temperaments was observed to result in a different emotional state over the time period of the experiment even when the scenario was kept constant. Finally, the importance of the reward prediction error was highlighted by showing that without it events affected the emotions by the same amount regardless of when they occurred. With the RPE, it was possible to mimic the emotional response observed in humans by Rutledge et al [7].

Additional tests should be conducted by specifying a more comprehensive list of command modifiers and fine tuning the temperament values based on the observation of an animal foraging for food in the wild. Since the temperament is specified by means of a constants matrix, this leads to the possibility of an agent with a time-variant temperament that can alter its emotional sensitivity during run time. The emotional model could also be modified to introduce a time lag between the occurrence of an event and it affecting the emotional state. This would allow the simulation of the four classic temperaments, namely, melancholic, phlegmatic, choleric, and sanguine theorized by Greek philosophers [36]. The low cost of the robot also permits the acquisition of a non-homogeneous robot swarm with varying temperaments to explore if emotions increase the effectiveness of the swarm.

Appendix

1. Schematic of Matrix Display



Bill of Materials	
Quantity	**Component**
1	8x8 Led Matrix
1	Max7219CNG
1	100nF ceramic capacitor
1	10uF, 50V electrolytic capacitor
1	24K resistor
1	3 pin female header

Bibliography

[1] R. W. Picard, “Affective Computing,” M.I.T, 321.
[2] Cynthia Breazeal and Rodney Brooks, “Robot Emotion: A Functional Perspective.”
[3] “Sociable machines - Overview.” [Online]. Available: http://www.ai.mit.edu/projects/sociable/overview.html. [Accessed: 15-Nov-2015].
[4] Valery Karpov, “Robot’s temperament,” *Biol. Inspired Cogn. Archit.*, vol. 7, pp. 76–86, 2014.
[5] P. Ekman, “Basic emotions,” in *Handbook of cognition and emotion*, 1st ed., Wiley & Sons Ltd., 1999, pp. 45–60.
[6] R. E. Jack, O. G. B. Garrod, and P. G. Schyns, “Dynamic Facial Expressions of Emotion Transmit an Evolving Hierarchy of Signals over Time,” *Curr. Biol.*, vol. 24, no. 2, pp. 187–192.
[7] R. B. Rutledge, N. Skandali, P. Dayan, and R. J. Dolan, “A computational and neural model of momentary subjective well-being,” *Proc. Natl. Acad. Sci.*, vol. 111, no. 33, pp. 12252–12257, Aug. 2014.
[8] Lyle N. Long, Troy D. Kelley, and Eric S. Avery, “An Emotion and Temperament Model for Cognitive Mobile Robots,” presented at the 24th Conference on Behavior Representation in Modeling and Simulation (BRIMS), Washington, DC, 2015.
[9] Lyle N. Long, “Modeling Emotion and Temperament on Cognitive Mobile Robots,” presented at the 22nd Annual ACT- R Workshop, Carnegie Mellon University, 2015.
[10] “AI Impacts – Trends in the cost of computing.” [Online]. Available: http://aiimpacts.org/trends-in-the-cost-of-computing/. [Accessed: 15-Nov-2015].
[11] “DFRobot 2WD Mobile Platform for Arduino - RobotShop.” [Online]. Available: http://www.robotshop.com/en/dfrobot-2wd-mobile-platform-arduino.html. [Accessed: 11-Nov-2015].
[12] “Arduino - ArduinoBoardUno.” [Online]. Available: https://www.arduino.cc/en/Main/arduinoBoardUno. [Accessed: 11-Nov-2015].
[13] “Arduino - ArduinoMotorShieldR3.” [Online]. Available: https://www.arduino.cc/en/Main/ArduinoMotorShieldR3. [Accessed: 11-Nov-2015].
[14] “BCM2835 - Raspberry Pi Documentation.” [Online]. Available: https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2835/README.md. [Accessed: 11-Nov-2015].
[15] “raspberrypirev2.jpg (640×426).” [Online]. Available: http://www.vesalia.de/pic/raspberrypirev2.jpg. [Accessed: 11-Nov-2015].
[16] “Pololu - MinIMU-9 v3 Gyro, Accelerometer, and Compass (L3GD20H and LSM303D Carrier).” [Online]. Available: https://www.pololu.com/product/2468. [Accessed: 11-Nov-2015].
[17] Gilad Lerman, “The Shannon Sampling Theorem and Its Implications.” [Online]. Available: http://www.math.umn.edu/~lerman/math5467/shannon_aliasing.pdf. [Accessed: 11-Nov-2015].
[18] Mircrochip, “MCP3004/3008 - 2.7V 4-Channel/8-Channel 10-Bit A/D Converters with SPI Serial Interface.” .
[19] “Pololu Step-Down Voltage Regulator D15V35F5S3.” [Online]. Available: https://www.pololu.com/product/2110. [Accessed: 11-Nov-2015].
[20] SHARP, “GP2Y0A21YK/GP2Y0D21YK General Purpose Type Distance Measuring Sensors.” .
[21] “Webcam C170 - Logitech Support.” [Online]. Available: http://support.logitech.com/en_us/product/webcam-c170. [Accessed: 11-Nov-2015].

[22] “WNR1000 | WiFi Routers | Networking | Home | NETGEAR.” [Online]. Available: http://www.netgear.com/home/products/networking/wifi-routers/WNR1000.aspx#tab-techspecs. [Accessed: 11-Nov-2015].
[23] “IBM ThinkPad T30 specs - Engadget.” [Online]. Available: http://www.engadget.com/products/ibm/thinkpad/t30/specs/. [Accessed: 12-Nov-2015].
[24] “Installing OpenCV on a Raspberry Pi - Robert Castle Consulting.” [Online]. Available: http://robertcastle.com/2014/02/installing-opencv-on-a-raspberry-pi/. [Accessed: 11-Nov-2015].
[25] “Basic Structures — OpenCV 2.4.12.0 documentation.” [Online]. Available: http://docs.opencv.org/2.4/modules/core/doc/basic_structures.html. [Accessed: 11-Nov-2015].
[26] “Reading and Writing Images and Video — OpenCV 2.4.12.0 documentation.” [Online]. Available: http://docs.opencv.org/2.4/modules/highgui/doc/reading_and_writing_images_and_video.html#imencode. [Accessed: 11-Nov-2015].
[27] “Image Filtering — OpenCV 2.4.12.0 documentation.” [Online]. Available: http://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html. [Accessed: 11-Nov-2015].
[28] “Structural Analysis and Shape Descriptors — OpenCV 2.4.12.0 documentation.” [Online]. Available: http://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html. [Accessed: 11-Nov-2015].
[29] “User Interface — OpenCV 2.4.12.0 documentation.” [Online]. Available: http://docs.opencv.org/2.4/modules/highgui/doc/user_interface.html. [Accessed: 11-Nov-2015].
[30] “Miscellaneous Image Transformations — OpenCV 2.4.12.0 documentation.” [Online]. Available: http://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html#cvtcolor. [Accessed: 11-Nov-2015].
[31] “Operations on Arrays — OpenCV 2.4.12.0 documentation.” [Online]. Available: http://docs.opencv.org/2.4/modules/core/doc/operations_on_arrays.html#inrange. [Accessed: 11-Nov-2015].
[32] William Premerlani and Paul Bizard, “Direction Cosine Matrix IMU: Theory,” 17-May-2009. [Online]. Available: https://gentlenav.googlecode.com/files/DCMDraft2.pdf. [Accessed: 22-Sep-2015].
[33] C. Brom and J. Bryson, “Action selection for Intelligent Systems.” [Online]. Available: http://www.vernon.eu/euCognition/asm-whitepaper-final-060804.pdf. [Accessed: 09-Nov-2015].
[34] Ross A. Thompson and Abby C. Winer, “The individual child: temperament, emotion, self, and personality,” in *Developmental science: An advanced textbook, Edition: 6*, 6th ed., Psychology Press / Taylor & Francis, pp. 427–468.
[35] Dennis Ford, “Lecture Three ~ Emotions,” in *Lectures on General Psychology*, vol. 2, iUniverse, 2015.
[36] “Four Humors - And there’s the humor of it: Shakespeare and the four humors.” [Online]. Available: https://www.nlm.nih.gov/exhibition/shakespeare/fourhumors.html. [Accessed: 22-Nov-2015].