Autonomous Control of a Humanoid Soccer Robot: Development of Tools and Strategies using Colour Vision

A thesis presented in partial fulfilment of the

requirements for a degree of

Master of Engineering

Mechatronics

at

Massey University,

Albany, New Zealand.

Baden Rielly

2007

Masters Abstract

Humanoid robots research has been an ongoing area of development for researchers due to the benefits that humanoid robots present, whether for entertainment or industrial purposes because of their ability to move around in a human environment, mimic human movement and being aesthetically pleasing. The RoboCup is a competition designed to further the development of robotics, with the humanoid league being the forefront of the competition.

A design for the robot platform to compete at an international level in the RoboCup competition will be developed. Along with the platform, tools are created to allow the robot to function autonomously, effectively and efficiently in this environment, primarily using colour vision as its main sensory input.

By using a 'point and follow' approach to the robot control a simplistic A.I. was formed which enables the robot to complete the basic functionality of a striker of the ball. Mathematical models are then presented for the comparison of stereoscopic versus monoscopic vision, with the expansion on why monoscopic vision was chosen, due to the environment of the competition being known. A monoscopic depth perception mathematical model and algorithm is then developed, along with a ball trajectory algorithm to allow the robot to calculate a moving balls trajectory and react according to its motion path.

Finally through analysis of the implementation of the constructed tools for the chosen platform, details on their effectiveness and their drawbacks are discussed.

Acknowledgments

I would like to thank all the people who have helped me throughout this project. I would like to thank my partner Dina for all her continual encouragement and assistance, which has seen me though all the good and bad times. Next I would like to thank my family for their support, and a big thanks to my mother, Rio for helping me proofread my thesis.

To the Massey University staff of the engineering department and my supervisors Prof. Olaf Diegel, Dr Johan Potgieter, who I would like to thank for their help and guidance throughout my masters, and for giving me this opportunity.

Finally I would like to thank all my friends and all the postgraduate students, Lim Kang for working on the project with me, Matthew Read, Courtney DeLautour and Jonathan Zyzalo for all your contributions, and all the others that have worked along side me, helped me to complete my project, and for all the good times had.

Contents

List of figures

List of tables

Chapter 1: Introduction

1.1: Project Background and Objectives

Robot soccer is a new area of research to further development in robotics. The humanoid leagues are a new area of advancement in the game, with the objective to have a fully autonomous humanoid robot team capable of competing with the world's best by 2050 [Behnke et.al. 3][Burkhard et.al.][Kitano et.al.]. The development of a humanoid soccer robot team is a goal of the Mechatronics department at Massey University.

The appeal of humanoid robotics is that a humanoid robot can be used to carry out tedious and dangerous tasks that a human would normally be required to perform. Industries like construction could use the robots to perform tasks that would be considered life threatening [Inoue et.al.]. Humanoids could also be used in the entertainment industry to serve humans with their close resemblance to humans making them more readily accepted [Bekey].

The first objective of this project is the development of a soccer robot platform using off the shelf components, which will be able to compete in a robot soccer competition. The robot platform, and external devices, will have to be researched, and the appropriate devices chosen.

The control of the robot through vision will be the main focus of this project. A set of tools will be developed using colour vision that will aid the A.I. (Artificial Intelligence) of the final autonomous robot design.

The first tool will be a basic A.I. based on a point and follow routine. The aim is to use the camera as a pointer for the body of the robot to follow. Through this routine's implementation it is the goal to have a robot that has the basic cognitive ability to perform the required tasks to score a goal in a game of soccer, therefore enabling the robot to function autonomously.

The second tool that will be developed is a monoscopic depth perception algorithm. This added depth perception will not only allow the robot to predict the distance from the robot that an object is located, but allow subsequent calculations to be performed which will enable the robot to determine the trajectory of the ball's travel when in motion. The new information received by calculating the ball's trajectory will enable improvements in the overall A.I. of the robot.

1.2: Research Publications

Refereed Conference Proceedings

- B.J. Rielly, O. Diegel, C.L. Kang, M.J. Read, J.R. Zyzalo, J. Potgieter, W.L. Xu, "A Mechatronics Approach to Autonomous Control of a Humanoid Robot", 13th ENZCon, University of Canterbury, Christchurch, New Zealand, November, 2006.

- C.L. Kang, O. Diegel, B.J. Rielly, M.J. Read, J.R. Zyzalo, J. Potgieter, W.L. Xu, "Humanoid Biped Robots: Walking and Balancing using Natural Dynamics, ZMP, and Gyroscopic Sensors", 13th ENZCon, University of Canterbury, Christchurch, New Zealand, November, 2006.

1.3: Thesis Layout

Chapter 2 will introduce robot soccer the RoboCup in particular, along with its requirements, and regulations. It will give provide background on humanoid robots that are being developed for the RoboCup and for other applications in the real world. Finally it will discuss the equipment that will be used for the robot platform being developed for this application.

Chapter 3 will go into the mechatronic systems produced and go over the choice between monoscopic and stereoscopic vision for this project. Chapter 4 outlines the distance and direction calculations that are used and leads on to trajectory calculations in chapter 5.

Chapter 6 discusses the results that were achieved through this project and finally chapter 7 outlines the future work presented by the project and the conclusions reached.

Chapter 2: Literature Review

2.1: Autonomy in Robotics

Autonomy refers to systems capable of operating in the real-world environment without any form of external control for extended periods of time [Bekey]. For this reason we can define autonomous robots as intelligent machines capable of performing tasks in the world *by themselves*, without explicit human control over their movements [Bekey]. Since living creatures are able to interact and survive in a dynamic environment without outside control is a good basis to build on for autonomous robots [Bekey].

When looking at robot autonomy, there are three main capabilities that a robot needs to be autonomous [Veloso], which include:

- Perception - of the environment and the surroundings
- Action - the ability to respond to events and change one's state
- Cognition - the ability to reason and select the required action needed

By including these three capabilities you have the building blocks for autonomous control.

2.2: Robot Localisation

It is possible to break localisation up into two categories, local and the absolute localisation. Localisation can be achieved through the use of odometry and inertial navigation. Since the robot is a humanoid robot and therefore has bipedal motion, odometry has to be measured by the distance travelled per step. Inertial navigation is done via the use of accelerometers and gyroscopes to give the robot's position through integration of their signals [Demetriou][Bekey].

Absolute localisation can be achieved via beacons, landmarks, map matching, and range finding, to name a few. This can be done through the use of equipment such as colour vision, compasses, GPS, and range finding sensors like ultrasonic sensors and laser range finders. As explained later the use of ultrasonic and laser range finders can not be used for this robot platform [Demetriou][Bekey].

2.3: RoboCup

The RoboCup [robocup.org], is a soccer competition that primarily uses colour vision as the means for robot navigation, and requires the robots to be fully autonomous, bipedal in motion, and have vision as their main form of navigation around the field. Soccer was chosen since it requires the robot to function in a dynamic environment and requires a team of robots to communicate with each other [Burkhard et.al.] [Kitano et.al.][Behnke et.al. 3].

Humanoid robots were chosen for this area of development due to their ability to move around uneven terrain, climb stairs and move around areas designed for a human to navigate. The soccer competition was chosen due to its requirement for multiple agent cooperation [Kitano et.al.].

The RoboCup has many rules but just several of the major rules will be covered in this section. The rules are as follows: Any camera or sound sensors used are to be attached to the head, and force sensors are permitted to be attached anywhere on their body. The field in the RoboCup consists of six colour posts for robot navigation, two colour goals (yellow and blue), and an orange ball. This competition has various leagues; the one of interest in this thesis is the kid size humanoid league. In this competition certain variables are known, and are shown in the table below. The size of the Ball is 8.4cm in diameter and weighs 26g.

	KidSize	
A	**Field length**	**450**
B	**Field width**	**300**
C	**Goal length**	**50**
D	**Goal width**	**150**
E	**Goal area length**	**50**
F	**Goal area width**	**190**
G	**Penalty kick distance**	**120**
H	**Restart marker width**	**75**
I	**Centre circle diameter**	**90**
J	**Border strip width (min.)**	**60**

Table 1: Soccer field in cm. [Behnke]

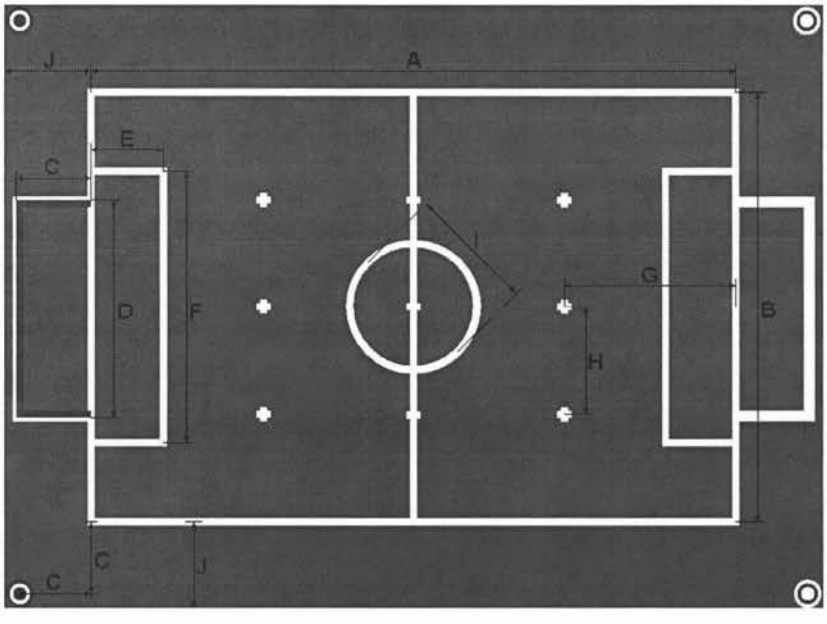


Figure 1: Soccer field [Behnke]

The design of the robot has to meet strict criteria. The robot's height, H must be between 30cm and 60cm tall and fit in the following formulas below. The body must be black in colour and only 10% of the robot's body can have "higher reflectance", e.g. grey or white. Furthermore less than 1% of the robot's body may be coloured

[Behnke]. Robots participating in the RoboCup Humanoid League must comply with the following restrictions:

- Each foot must fit into a rectangle of area $H^2/22$;
- The robot must fit into a cylinder of diameter H/2;
- If the arms are maximally stretched in horizontal direction, their extension must be less than 1.2 x H;
- The robot does not possess a configuration where it is extended longer than 1.5 x H;
- The length of the legs H_{leg}, including the feet, satisfies $0.4xH \leq H_{leg} \leq 0.6xH$;
- The height of the head H_{head}, including the neck, satisfies $0.1xH \leq H_{head} \leq 0.2xH$;
- Each robot must have an 8cm x 8cm team colour marker, either coloured magenta or cyan;
- The height of the robot must remain within $H = \min (H_{top}, 2.2 \times H_{com})$, where H_{com} is the height of the robots centre of mass [Behnke].

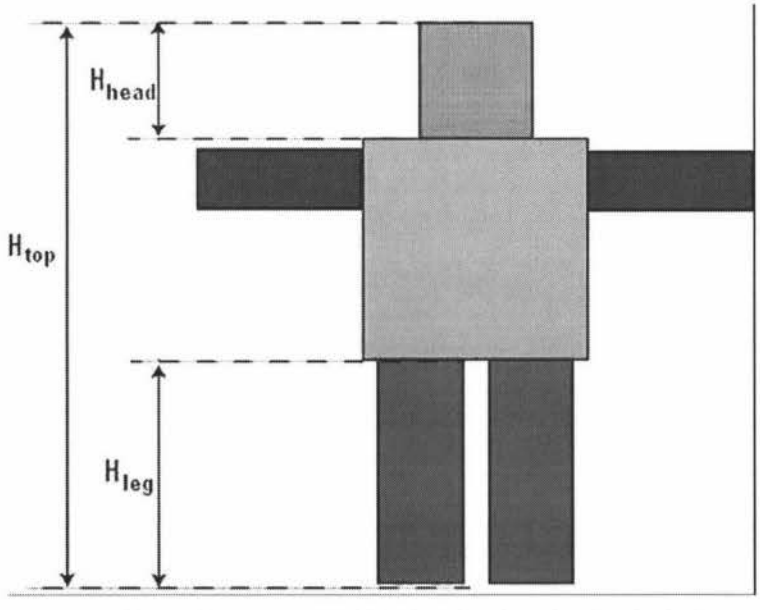


Figure 2: Humanoid robot body plan [Behnke]

2.4: Existing Humanoid Robots

ASIMO

Development of the Honda ASIMO started in 1986 with research into a set of bipedal legs. In 1996 Honda announced it was developing a two legged and two armed robot. The idea behind the need for a humanoid robot was that it would be able to move around an area designed for humans, e.g. where staircases, doors and furniture would exist, therefore being a suitable design for a "domestic robot". The P2 ASIMO robot stands 1820mm high and weighs 210kgs, and has a total of 30 degrees of freedom (DOF). To power this robot a NiZn battery pack weighing 20kgs is used and has a life of approximately 15 minutes [Hirai et.al.]. The P2 and P3 robots in the Honda robot series were not entirely autonomous but were partially controlled through a specially designed cockpit [Bekey].

HOAP - 2

HOAP-2 is a bipedal humanoid robot standing 50cm tall and has 26 DOF. It is able to walk while carrying objects through its use of dynamic walking using zero moment point (ZMP). The head of the robot has a two DOF neck and has two CCD cameras capable of capturing images with a resolution of 640 x 480 pixels [Xu et.al.] [Bekey]. A project using the HOAP-2 robot at the California Institute of Technology, involving depth perception of the robot gained accurate depth perception between the range of 20cm and 200cm by using the stereo vision onboard the robot, processed through an Intel Pentium 2.4GHz computer via a USB connection. At ranges greater than this the error increased dramatically [Stoica et.al.].

Robonaut

The Robonaut is a humanoid robot being developed by the Robotic Systems Technology Branch at NASA's Johnson space centre [Ambrose][Bekey]. This robot consists of 47 DOF despite only having a single leg, and has over 150 sensors per arm. Robonaut has 4 cameras for vision, 2 of which are placed approximately the

same distance apart as human eyes for stereo vision. Each image is filtered using Laplacian-of Gaussian filter and a bandpass filter to emphasise the edges and to smooth out small scale detail. Finally a Binary correlation is used to match patches from the left and right images [Bluethmann et.al.][Ambrose]. The purpose of this robot is for use in hazardous environments in low earth orbit and for planetary exploration. In the case of planetary exploration a Robonaut robot is in development that will be placed on a Segway RPM base [Drfter et.al.].

Sony's QRIO-SDR-4X

The QRIO-SDR-4X [Fujita et.al.] is the successor of the QRIO-SDR-3X [Ishida et.al.] which was developed by Sony in 2000. The QRIO-SDX-4X stands 54cm tall, and its four different types of Servo actuators, has 28 DOF, 6 of which are in each leg. It has 7 microphones located on its head and 2 CCD cameras for stereo vision, while the QRIO-SXR-3X only has one CCD camera and 2 microphones [Ishida_et.al.][Bekey]. It contains a 3-axis accelerometer and a gyroscopic sensor in its trunk, as well as 4 force sensors and 2-axis accelerometers in each foot, all of which are controlled through 3 CPU's. [Fujita et.al.] Even though the QRIO-SDR-4X has been available for purchase for the scientific community, the costs of QRIO-SDR-4X have been compared to the price of a luxury car [Behnke et.al. 4].

It has 4 main technical features, one of which is the "Real-time Integrated Adaptive motion control", [Fujita et.al.] allows the QRIO-SDR-4X to pass over rough and unlevelled terrain while maintaining its posture when external forces are placed on it. Another technical feature is "Real-time real-world Space Perception". [Fujita et.al.] This allows the QRIO-SDR-4X to walk in a real-world home environment with avoiding obstacles, [Fujita et.al.] and allows it to find humans and identify them using facial recognition and interact with its synthesised voice.

Robosapien, V2, Multimedia

Another humanoid robot with colour vision built in that is available on the market is the Robosapien V2 and Multimedia. The onboard camera has its colour image processing completed via its 55 MHz controller. This camera uses the RGB colour space and is only designed to see those colours individually. This basic colour vision allows the Robosapien to track a blue light, see a green ball and red bowling pins, and is able to distinguish skin colours [evosapien.com]. The Robosapien has been used as a platform for the RoboCup, for the humanoid league [Behnke et.al. 4].

Kondo KHR-1

The Kondo KHR-1 humanoid robot hobby kit has 17 DOF, 5 of which are in each leg and a 6^{th} DOF can be purchased as a separate kit if needed. The Kondo robot stands 34cm tall weighing 1.2kgs. The Kondo uses KRS-784ICS servo which are capable of 8.7kg/cm at 6V and have a response time of 0.17 seconds per 60 degrees. The firmware of the onboard controller is capable of storing 31 motions. The initial hobby kit does not come with gyroscopic sensors, but they too can be purchased separately [Kondo-Robot].

2.5: Existing RoboCup Systems

There are many humanoid robots in development for competition in robot soccer at present. This section will go over some of the robots that have been developed by giving an overview of the possible system design features that have been used and may be useful in the development of the robot's design.

Many cameras that are currently used by teams conducting research into humanoid robots for robot soccer competitions are low cost colour CCD or CMOS cameras, with a resolution ranging from 320 x 240 to 800 x 600 pixels [Behnke et.al. 1]. They often employ wide-angle lenses allowing a robot to see the ball, the goal, its own feet,

and any opponents in a single image, or an omni-directional lens to allow complete 360 degree view angle, thereby minimising localisation issues. The cameras are attached to a PDA (Personal desktop assistant) or PC104 board with approximately 500MHz processing power and 128MB of RAM. The PDA or PC104 board is required for the additional computational power needed to process the images. This has been shown to be effective with such existing systems as Abarenbou [Behnke et.al. 1], which is based on the Kondo KHR-1 robot; Toni [Behnke et.al. 1], which has similar functionality to the robot used in this research, including an accelerometer and 2 gyros; and the Robosapien soccer robots such as NIMBRO RS [Behnke et.al 1] [Behnke et.al. 2].

NIMBRO RS

The NIMBRO RS uses the WowWee Robotics Robosapien V1 robot as its platform, which is one of the cheapest off the shelf humanoid robots [Behnke et.al. 2]. Since the robosapien only has 3 motors to control its motion it does not pose the same amount of control as a servo based system which can have upwards of 6 DOF per leg compared to the total 9 DOF of the Robosapien [Behnke et.al. 2].

Along with the bump switch already on the front and rear of the Robosapien's feet, a 1.3M pixel camera is added, attached to a Pocket PC via its compact flash (CF) slot, giving the robot a view angle of about 55 degrees. The Pocket PC used comprise of an intel 400MHz processor, 64MB of RAM and 32MB of flash memory, and is capable of frame rates of 1fps at 1280 x 1024, 4fps at 320 x 240 and 15fps at 160 x 120. A sample of the Robosapien vision is shown in Figure 3.

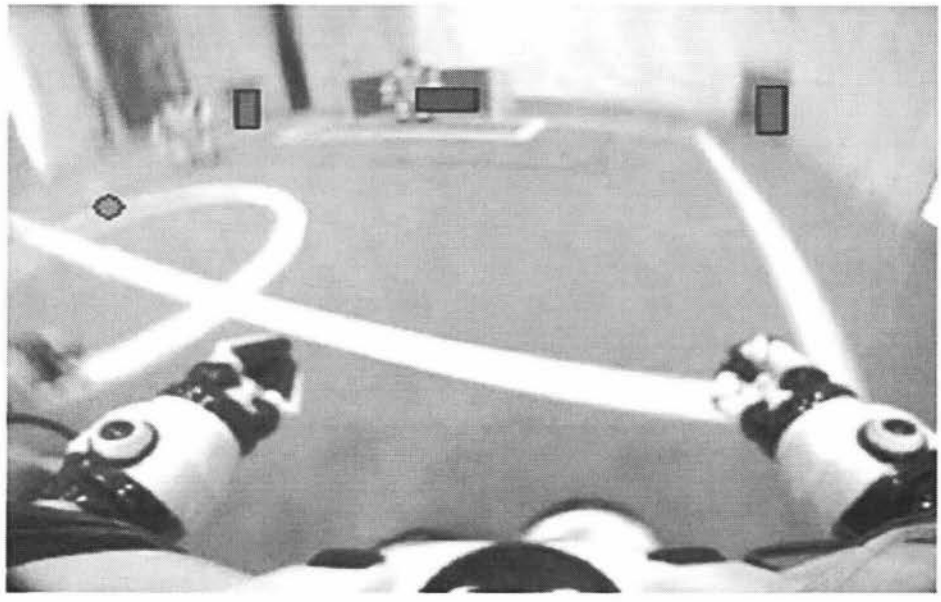


Figure 3: Robosapien vision through wide-angle lens [Behnke et.al. 1]

The colour detection algorithm used by the NIMBRO RS is accomplished by capturing single images when the robot is stationary (this is to avoid blurring). Colour segmentation of the individual RGB values and spatial information is used to find and estimate the largest blob of the tracked colour. It will then only focus on the largest blob. Complex behaviours are handled by a state machine, which converts the task into several subtasks, and are triggered by the presence of visual features

[Behnke et.al. 2].

Toni

Toni is classed as a teen-size robot since it stands 74cm tall and weighs 2.2kg. It has 18 DOF, with 6 DOF in each leg. The servos are controlled via 3 Motorola MC9S12C32 16Bit microcontrollers running at 24MHz with 2kB RAM and 32kB flash. The microcontrollers communicate to each other via a controller area network (CAN) bus at 1Mbps and to a computer via RS232 serial at 115Kbps. For control of balance Toni is fitted with a dual axis accelerometer and 2 gyroscopic sensors.

For vision, Toni has been fitted with two 1.3M pixel cameras with fisheye lenses, one facing forward and one backwards. These are fitted to a PDA attached to the torso, running a 520Mhz with 128MB RAM but has been tested with a 1.1Ghz Pentium M processor, with 512MB RAM and a 20GB hard drive. This system is capable of delivering frame rates of 30fps at 640 x 320 resolution [Behnke_et.al._5][Behnke_et.al._1].

Toni is capable of getting its altitude via the integration of its gyroscopic sensors allowing it to recreate a particular camera pose. The wide-angle CF colour camera allows seeing the ball at the robot's feet, the goal, and poles simultaneously [Behnke_et.al. 5]. The colour images are converted in YUV (Y = luminance, U = blue chrominance, V = red chrominance), colour space to decrease lighting effects from different lighting conditions. Localisation is done using a probabilistic Markov localisation method which integrates egocentric observations and motion commands over time [Behnke et.al. 5].

ABERENBOU

Abarenbou is designed on the Kondo KHR-1 humanoid robot kit, which has 17 DOF, [Behnke et.al. 1] with all the robot's control done through an attached PDA which communicates with the embedded robot through a shared 115200 baud serial connection.

The robot's vision is achieved via a CMOS camera which is directly attached to the PDA. The vision algorithm uses little colour information, instead it uses an approximate region segmentation algorithm, along with a flood fill algorithm, then finally checks to see if the corresponding object matches the object definition [McCann et.al.][Baltes et.al.].

ROPE

ROPE is a humanoid soccer robot initiative from the National University of Singapore. The initial robot was a set of legs which was used to develop the walking gates. The second robot was a height of 49.5cm, weighed 2.5kg, and has 18 DOF. For vision the CMUcam camera is used. For sensory, force sensors are attached to the soles of the feet. The use of a 3-axis accelerometer for velocities of the body and a gyro with its output integrated to gain position. These sensors are all connected to a DAQ board. The DAQ microcontroller for servo control and CMUcam camera are all connected to a PC – 104 board running RT – Linux [Nghia et.al.].

The third version of the robot utilises the CMUcam2 board to improve its colour vision. Another accumulation to the sensory of the robot was the addition of a digital compass for direction [Team RO-PE].

The forth version went from using the PC – 104 board to the use of 2 Atmel ATMega microprocessors to minimize the weight and size of the robot. The one problem with using purely microcontrollers is the decrease in processor power [Team RO-PE].

The Fifth and final ROPE robot uses 2 monoscopic cameras, one which focuses straight ahead, the second camera uses an omni-directional camera for increased localisation [Team RO-PE].

Vision

Omni-directional cameras have the benefit of allowing the robot to have a complete view of the entire area thereby giving it the ability to localise itself in that area with objects in its field of view. This idea has been used in robot soccer by TeamOsaka [Suzuki et.al.][Yamato et.al.] for their RoboCup humanoid entry called Vision, which has won them many titles in the competition. This robot consists of 23 DOF, 3-axis accelerometer, 3-axis gyro, voice communication, and wireless network

communication. 2 onboard CPUs control the robot, a 40Mhz processor for servo control and a 400Mhz processor with 4GB of ROM and 256MB of RAM for image processing and behaviour control [Yamato et.al.]

2.6: Hardware Requirements

2.6.1: Research Platform

From the previous sections some robot platforms have been described as being used by other competing teams. Although all these robots have made good platforms for development they were left lacking in particular areas. For this reason the following robot was the chosen development platform.

Pirkus – R Type 1 DX

The platform chosen for this research is the Pirkus R type 01 DX [IBee] as shown in Figure 4. This humanoid robot package contains 21 servos encased in an aluminium exoskeleton. The Pirkus stands approximately 29cm high and weighs approximately 1050g. For the RoboCup this would place the Pirkus in the kid size division of the competition which ranges from 30cm to 60cm tall.

The Pirkus' 21 servos allow 21 DOF; 6 DOF in each leg (including the waist joint); 4 DOF in each arm (including the shoulder joint); and 1 DOF for the head. The servos used for this robot are the Pirkus robotics PRS-DE07M digital servos which are pulse width modulation controlled and have 7.1kg/cm of torque, and a response time of 0.11 seconds.

Sensor feedback from the Pirkus consists of two gyroscopes for the pitch and roll axes and a 3-axis accelerometer for stability control. It is controlled via an Atmega 128 micro controller that runs at 16 MHz and is capable of 16MIPS. The ability to be

programmed in C allows for easy modification of the controller's code, allowing effective control of the motion and task handling to be developed for the robot. The Pirkus can be interfaced with a computer, using either bluetooth with speeds up to 115.2Kbps or serial RS232 at 57600Kbps.

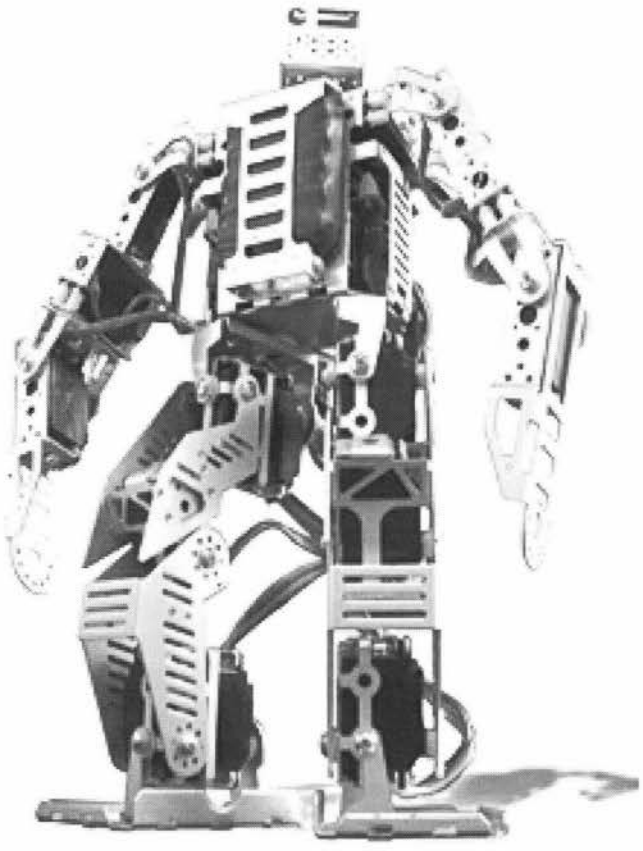


Figure 4: Pirkus – R Type 01 DX [IBee]

2.6.2: Digital Compass

To aid with the robot's localisation a digital compass was added. The compass chosen was the Devantech CMP03 magnetic compass module. This compass is fitted with the two Philips KMZ51 magnetic field sensors mounted at right angles to detect the horizontal component of the earth's magnetic field. The compass measures 32mm x

35mm in size and can be seen below in figure 5. This module can be interfaced by converting its outputted PWM values or through a 100 KHz I2C bus, which gives a reading in either degrees with a resolution of 0.1^0 by using 2 8bit registers, or by giving a value between 0 and 255 using just 1 of the 8bit registers. The compass requires 5VDC to run and draws approximately 25mA [CMPS03].

The I2C option was chosen since the conversion of PWM to the relative angle has already been done, allowing the value to be passed from the compass to be used in the control system being developed for the robot.

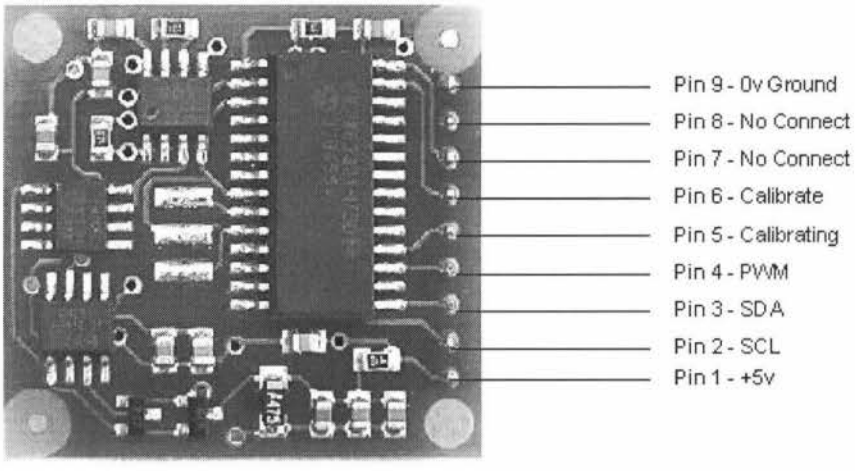


Figure 5: CMP03 electronic compass module [CMPS03]

2.7: Vision Systems

Vision systems have become an important factor in robotics, especially in the development of autonomous robots. Vision systems allow a robot to better interact with its environment, detect objects, and to avoid obstacles in their path. Competitions have been setup for development in this field especially for stereo and colour vision.

2.7.1: Colour Vision

For this project, colour image processing was chosen over grey scale for several reasons. The robot must be able to compete in the RoboCup competitions which are designed around colour vision; therefore it would need to be able to process colour images for localisation of the robot, and detection of the goals and ball. Another reason for using colour image processing is that a colour image contains information about an object's details that a grey scale image may not distinguish just from the intensity of a picture.

2.7.2: Omni-directional

Omni-directional cameras have been in development for localisation by Osaka and Padua Universities [Menegatti et.al.]. In this research project omni-directional cameras were mounted around the room in the form of a grid, with the same omni-directional camera was mounted onto a robots head. The robot would store several images from different locations (called reference images) and compare them to images that were captured on the robot using a Fourier signature, which gives a compact version of the complete picture. The reference images that most closely resembled the robot's image were deemed to be camera that is closest to the robot's position. This was achieved by the position being given by the magnitude coefficients

of the Fourier signature, and the orientation, calculated via the phase coefficients. This proved to be an effective way for a robot to localise itself to it surroundings, though this would not be feasible for our application.

The use of an omni-directional camera has the benefit of allowing the robot to have a complete view of the entire area and thereby gives it the ability to localise itself in that area using objects in its field of view as markers. However, this form of vision was not chosen because the robot would not significantly benefit by knowing what is behind it. An omni-directional vision system would give the robot a larger number of obstacles to focus on than required by our research, which intends to focus the robot's attention on a more specific task.

2.7.3: Wide Angle Lens

The wide angle lens, also known as the fisheye lens gives a much greater angle of view over your standard lens. This system has been chosen for many robotic applications since the robot is able to see most of the surrounding obstacles without the need to look around its environment. This broad line of site can be useful for some applications, especially where the robot is unable to quickly scan its environment. Many systems used in robot soccer competitions use a wide lens system [Behnke et.al 1].

2.7.4: Stereo Vision

Stereo vision allows robots to have a more advanced depth perception using the difference between the images captured via each camera. This system could be used to give the robot a more accurate account of distance and direction related to objects in the robot's path. In the case of robot soccer it would give a robot a coordinate of the ball's location in a XYZ plane. Stereo vision is not yet commonly used in the RoboCup competition, and will not be used for our vision system as explained later.

The use of stereo vision over monoscopic vision has been explored for the robot platform in question. A monoscopic vision system will allow distance to be judged via the number of pixels taken up by the object in question if the object size is known, and direction can be given via the location of the object in relation to the centre of the image, and by the direction the robot is facing. This system is currently being used by major competitors in the RoboCup [Behnke et.al 1][Nghia et.al.]. This has led to the conclusion that for a humanoid robot soccer environment the advantages gained from stereo are minimal compared to the complexity involved in integrating them effectively with minimal error.

2.7.5: Developed Vision Boards

The CMUcam [Rowe et.al. 3] and CMUcam2 [Rowe et.al. 2] have been used extensively in several projects, one being the ROPE robot as discussed previously. Its camera board has been used for target acquisition in unmanned vehicle (UMV) including helicopters [Mike] as well as other various tracking projects, including tracking the horizon for aircraft positioning [Cornall et.al.] and vehicle tracking [Humphries et.al.]. Through the above examples and the requirements of this project, the CMUcam2 was chosen to be used for the Pirkus' vision system.

CMUcam2 Camera

The camera module chosen for this project is the CMUcam2 (Figure 6) produced by Carnagie Mellon University [Rowe et.al. 1][Rowe et.al. 2][ovt.com][Bruce et.al.]. This camera is a direct upgrade from the CMUcam [Rowe et.al. 3]. The CMUcam2 has twice the ROM and RAM as its predecessor and can control 5 servos compared to the single servo control of the CMUcam. That makes it a more viable option for this

project considering the complexity required to implement the vision system, and the large number of servos that make up the Pirkus. Other features present in the CMUcam2 that are not present in the CMUcam may also prove useful. These include the master and slave mode capability, which will be described in later sections. One of the main advantages of this system is that the controller board and camera only cost $179US, making it very affordable. This is particularly important for robot soccer applications since multiple units will be purchased for a soccer team.

The camera obtained with this package is the Omnivision [ovt.com] OV6620 or OV7620 CMOS camera on a chip. Both cameras have an analogue video output with the OV6620 being PAL and the OV7620 being NTSC. The package chosen supplied was the OV6620 camera as New Zealand uses the PAL format. This camera has a resolution of 352 x 288 which is lower of that of the OV7620. This is unimportant since the unit only has a maximum resolution of 176 x 255 onboard independent of camera. The OV6620 allows a lower resolution mode of 88 x 143 though the OV7620 does not, which could prove useful for real-time applications since there will be less pixels for the vision algorithms to process allowing more frames per second, along with decreasing the amount of information to be sent via the serial port. The maximum refresh rate is 50fps (frames per second) which is ample for our robot's application when it is compared to the 24fps used by most cinema movies that look fluid to the human eye.

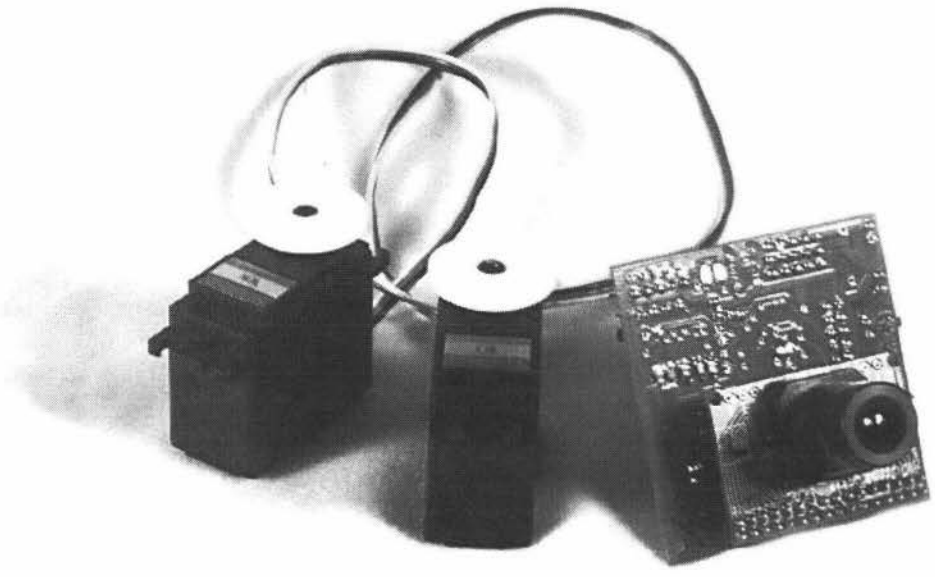


Figure 6: CMUcam2 [Rowe]

The control board of the CMUcam2 uses an Ubicom SX52 which operates at 75 Mhz. This processor is a RISC processor that can operate at 75MIPS. It has 262 bytes of SRAM and a 4096 word flash programmable EEPROM. The Control board can be communicated to via either TTL serial, or RS232 serial which will be more appropriate since the Pirkus has a RS232 connection. The Controller board in figure 7 is able to control 5 servo motors as well as having 4 I/O ports. The servo outputs can also be used for I/O ports.

The camera is initially set up for RGB colour space but can be set to The YUV colour space (Y = luminance, U = blue chrominance, V = red chrominance), also known as the YCbCr colour space where Y is the intensity of the image and UV are the red and blue chrominance of the colour. The YUV colour space will be more appropriate for this type of application because the lighting conditions will not affect the celebrated colour segmentations for the robot since the intensity component can be set to a threshold and the colour processing done on the U and V components.

A limitation of CMOS cameras is that the colour channels are between 16 and 240 instead of the full 256 for each colour.

Another impressive feature of the CMUcam2 that could be used for the project is that a single camera can have multiple controller boards linked together using a PC104 style connector in a master/slave setup. This allows parallel image processing to be done on the camera's images. This could prove very useful in the case of RoboCup where one controller can be responsible for ball location and the other for localisation for the robot. A problem was encountered with the master/slave mode in initial trials, in that the received images had significant levels of noise present. This will be overcome by using shielded cables to remove the noise.

The CMUcam2 has some useful built-in image processing tools including colour tracking, frame differencing and histogram analysis. The plan is to exploit these functions for the Pirkus, especially the colour tracking.

The CMUcam2 detects objects by using Run Length Encoding (RLE) Algorithms [9] on the image, which has been utilised by the Carnegie Mellon Universities legged soccer team, using the Sony AIBO platform. This process makes a horizontal run of the image, linking up the connected colours over two runs. Each colour grouping is then listed in order of size. Small neighbouring regions are finally grouped together to reduce noise.

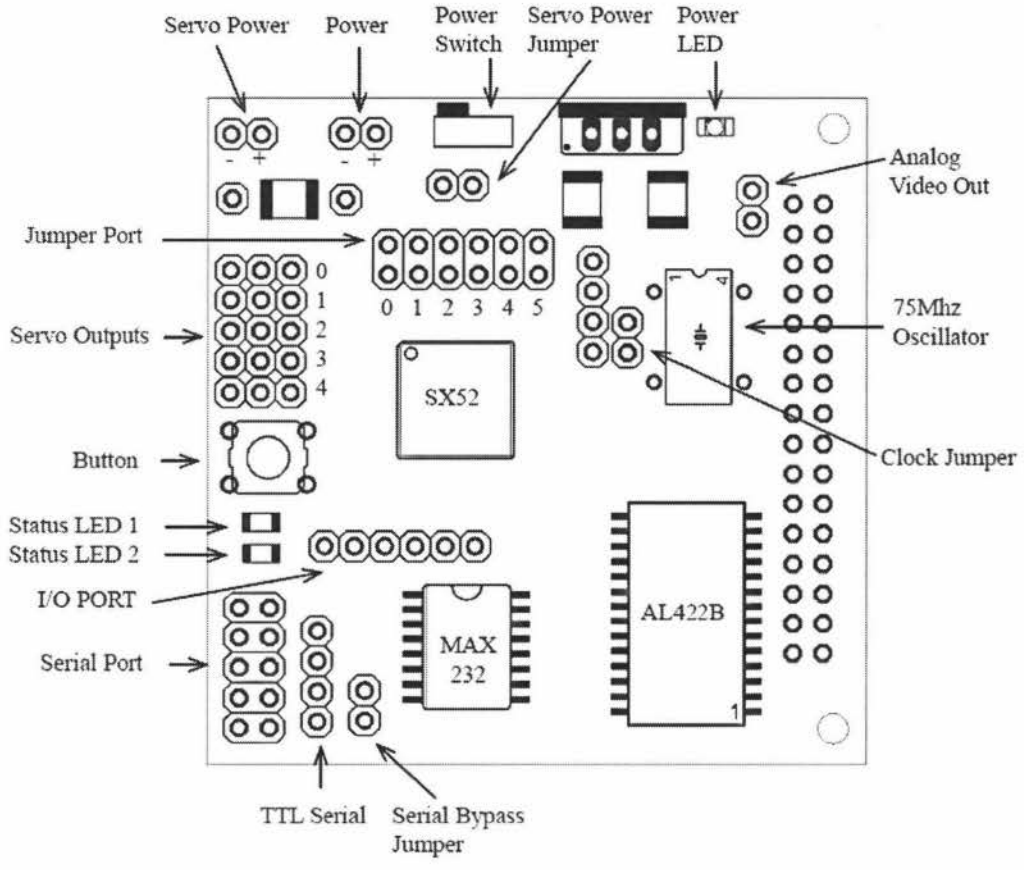


Figure 7: Controller Board [Rowe]

The camera will be mounted on the Pirkus' head servo. This will mean that the head casing will need modification, or a completely new casing developed with an additional servo added to vertically tilt the neck. This will allow full tilt and pan of the camera and will be discussed in detail in later sections.

2.8: Control Board

For integration and control of the systems required for autonomous control through colour vision and for the development of other forms of autonomy including balance, a central processing unit is required. This device will need to relay communication between devices while doing a majority of the processing of the received information and converting that information to required tasks.

There are three main types of controllers used for humanoid robots designed for the task of robot soccer, they consist of a PC-Based system such as a PC-104 board; a microcontroller based system; and a PDA based system [Zhou et.al.]. These 3 systems will be explored in this section.

2.8.1: Robot Controller board

The first option for this task would be to use a microcontroller with a simple circuit, which would be light weight and small in size. The chosen robot platform has an onboard 8 bit processor, the sole use of this processor for the autonomous control was not considered an option since it was already responsible for servo control, storing motions, wireless communication and the robots onboard gyro and accelerometer sensors. Since this robot is being developed as a platform, the need for extra sensory devices such as vision and a digital compass, as well as the need for future development of localisation and vision algorithms, extra processing would be required. The reason that a microcontroller was not chosen was because they lack sufficient user-programmable computing power required for development of an autonomous robot [Behnke, et.al. 1].

2.8.2: Microcontroller

Due to the current and future developments required, using a second micro processor was considered. Due to the robot and the camera needing to be communicated with via serial communication a microcontroller with multiple UART was required. The Atmel ATmega 128 chip as used by Singapore Universities ROPE V robot [Team RO-PE], and the same chip that is presently on the robot, was considered.

Since the aim of this project was to use off the shelf equipment to reduce costs and allow multiple robots to be developed a development board was investigated. The

development board [Futurlec.com] was only 60mm by 90mm in size and already had the chips UART converted to RS232 serial with two MAX232 chips. The limitation of this solution was the relatively slow processing speed of the processor at 16 MHz, along with the lack of expandability for future added sensory or algorithms. The same development board as above is available with an ARM Processor (LPC2103 microcontroller from Philips) running at 59 MHz, which even though is faster, is still not optimal for our design due to its low amount of processing power.

2.8.3: PDA

A PDA or pocket computer has been widely used as an onboard processor in various small autonomous humanoid robots. This is especially prevalent in robot soccer where they are used since they are lightweight, compact, robust, affordable and have many interfaces. [S.Behnke, et al]. In some current competitive robots a Pocket PC or Ultra-Portable PC is used with speeds ranging from 520 MHz to 1.1 GHz, with the camera connecting via a Compact Flash port or through USB and communication to the robot done via RS232 or a USB to RS232 converter. These usually have windows CE or windows mobile installed although Linux and Symbian operating systems are available.

2.8.4: PC - 104

A PC104 computer board is basically a fully functional small single board computer, measuring 3.5" by 3.75" by 0.6" in size. Speeds of processors range from 12 MHz to over 1 GHz. The software on these systems can range from DOS based OS to windows XP and Linux. Since they are so small and lightweight it makes them a good board for control of a small autonomous humanoid robot. In the case of robot soccer they have been used by the Singapore Universities ROPE team, in there ROPE 3 robot [Nghia et.al.]. A useful feature of PC-104 board is the ability to stack extra boards, this will allow extra functionality to be added if and when they are needed, the only problem with this is the added size, and therefore this needs to be kept to a

minimum. Even for the small size of this board it is still relatively big in relation to the robot, which will mean it will need to be attached to the back of the robot in the form of a backpack, as in the same form as the ASIMO robot [Hirai et.al.][Bekey]. The board chosen for this project was the SBC1495, which uses a 486DX processor running at 133 MHz, 64MB of SDRAM, and uses a CF card as an IDE hard drive, which is a good midrange system which is running DOS and Windows CE. For communication the board has 2 RS232 COM ports, 2 USB ports as well as a parallel port. The SBC1495 has the option of connecting a mouse, keyboard and CRT monitor. This will allow for easy setup of the robot without the need to have these connections when the robot is running allowing for the minimal size to be carried by the robot.

Even though the power requirements for the board are high with a 5V, 1.8A supply need, this is seen to not be a great issue. This conclusion is drawn from the fact that it is battery powered, and draws approximately 3A when in motion. This will be easily compensated with a second battery and voltage regulator is installed. The initial board purchased has an operating temperature of 45 degrees but high temperature board can be purchase for future robots which has an operating temperature up to 85 degrees (See Appendix D).

Chapter 3: Development of the Mechatronic System

3.1: Vision System Development

One of the dilemmas facing vision system designers is whether to use monoscopic or stereo vision. Besides the obvious questions of cost, little research has been done in determining which system performs more effectively under specific conditions. This section develops a mathematical model for determining the effectiveness of stereo and monoscopic vision. It discusses the benefits and downfalls of both systems, focusing on a small humanoid robot platform for robot soccer.

3.1.1: Monoscopic Vision

Since monoscopic vision uses only one camera, its depth perception is entirely based on knowing the size of the object being viewed. Therefore, for a monoscopic colour vision system, each colour would have to be assigned to an object of a known size to allow their distances from the camera to be calculated.

For an object size (O_S), resolution (R), and view angle of the camera (V), the distance from the camera to the object (X) can be calculated. The distance per pixel (D_P) can be calculated via the object size in millimetres (mm), and the number of pixels of the object (O_P) being examined. This is shown in equation 1. The value calculated can be multiplied by the respective resolution to give the horizontal and vertical field of view distances (F_D) in mm as in equation 2. By dividing the triangle contained by the field of view angle of the camera (V), and the horizontal or vertical distance F_D in mm by half, Pythagoras' theory can be used to calculate the distance of the object from the camera (X) with the equation 3. Since X is only the distance from the ball to the camera, the height from the ground to the camera (H) must be added. Figure 8 Shows

X is the hypotenuse of the triangle; therefore the distance of the robot from the object (O_D) is calculated from equation 4.

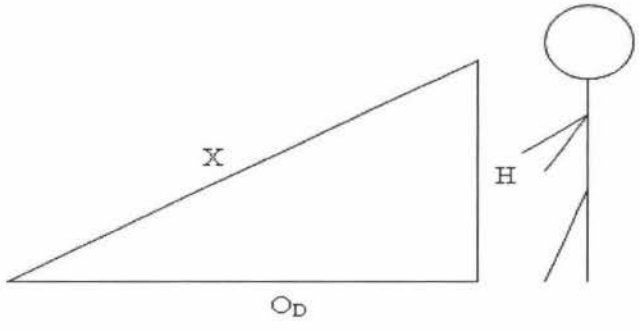


Figure 8: Calculating the robot's distance for the ball.

$$D_P = \frac{O_s}{O_P} \tag{1}$$

$$F_D = RD_P \tag{2}$$

$$X = \frac{F_D/2}{\tan(V/2)} \tag{3}$$

$$O_D = \sqrt{X^2 - H^2} \tag{4}$$

The next step is to include the systems resolution error. The two areas to look at are how far the distance will vary with the object being a pixel value (O_P), and how far the distance will vary for the image reaching the next pixel size. As seen in figure 9, as the object approaches the camera, the image size has to increase a specific pixel increment, therefore the Pirkus has to move a certain distance before the image increases to the next number of pixels.

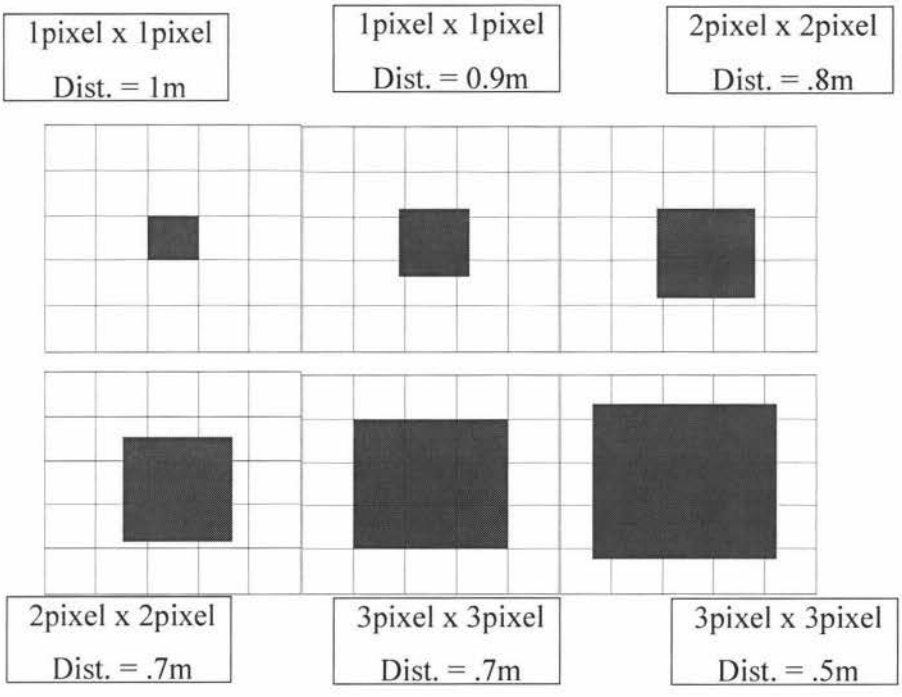


Figure 9: Resolution error of monoscopic vision

The description for the magnitude of change for the object's distance depending on the object's pixel value is as follows. The error is found by halving the difference between the calculated values for the current O_P value and the O_P before and after, as shown in equation 5 where X_0 is the distance from the object to the Robot at that particular number of pixels and X_1 and X_{-1} are the calculated distance to the robot at plus or minus one pixels in the objects size. From the formula the plus or minus distance from the object's actual distance is calculated, giving the range within which the object can be located (E_0). The horizontal and vertical values are then being used together to give greater accuracy by finding the average calculated O_P. By using the smallest maximum, and largest minimum E_0 of the horizontal and vertical axis, the range for a particular pixel value is decreased. Using the above calculations, a ball with an 80mm diameter (regulation RoboCup humanoid kid size soccer ball), and a 176x255 resolution camera 30cm off the ground (approximate height of the Pirkus);

the E_0 error can be seen in figure 9 above. From the figure it can be concluded that the higher the resolution of the camera, the more distance and accuracy of the vision system increase, therefore supporting the use of a smaller more focused lens for the fast vision system.

$$+E_0 = X_0 + \left|\frac{X_0 - X_{-1}}{2}\right| \qquad (5)$$

$$-E_0 = X_0 - \left|\frac{X_0 - X_1}{2}\right|$$

Another error occurs if the camera is out by a single pixel (E_1), as shown in figure 10, thus increasing the range of error that the ball will fall. Equation 6 shows the calculation of this increased error.

$$+E_1 = X_0 + \left(\left|X_0 - X_{-1}\right| + \left|\frac{X_0 - X_{-2}}{2}\right|\right) \qquad (6)$$

$$-E_1 = X_0 - \left(\left|X_0 - X_1\right| + \left|\frac{X_0 - X_2}{2}\right|\right)$$

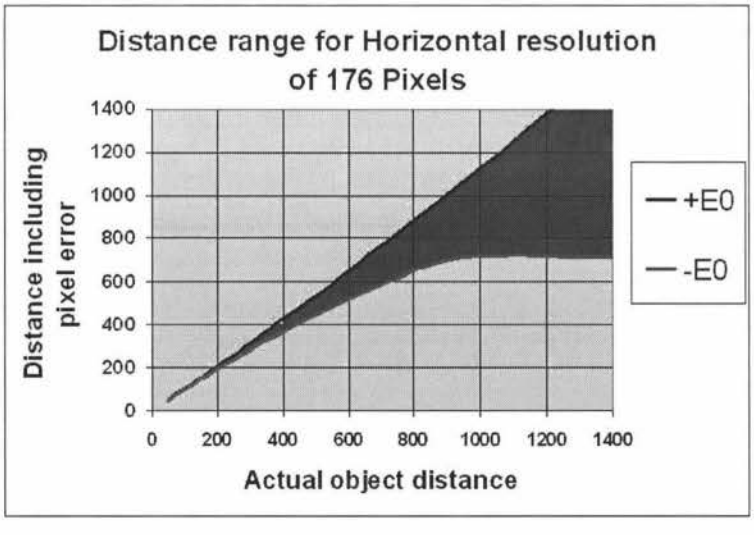


Figure 10: Distance error of monoscopic vision

An important factor in the monoscopic distance equation is the resolution of the camera, since the more pixels that an image has the greater the number of pixels in an object, therefore the smaller the error through the image being a pixel out. Figure 11 illustrates that the error range decreases with the resolution and the effective distance of the distance calculation increases. If the camera resolution is increased to approximately 1 mega pixel (1024 x 768), the calculation will have a much improved accuracy.

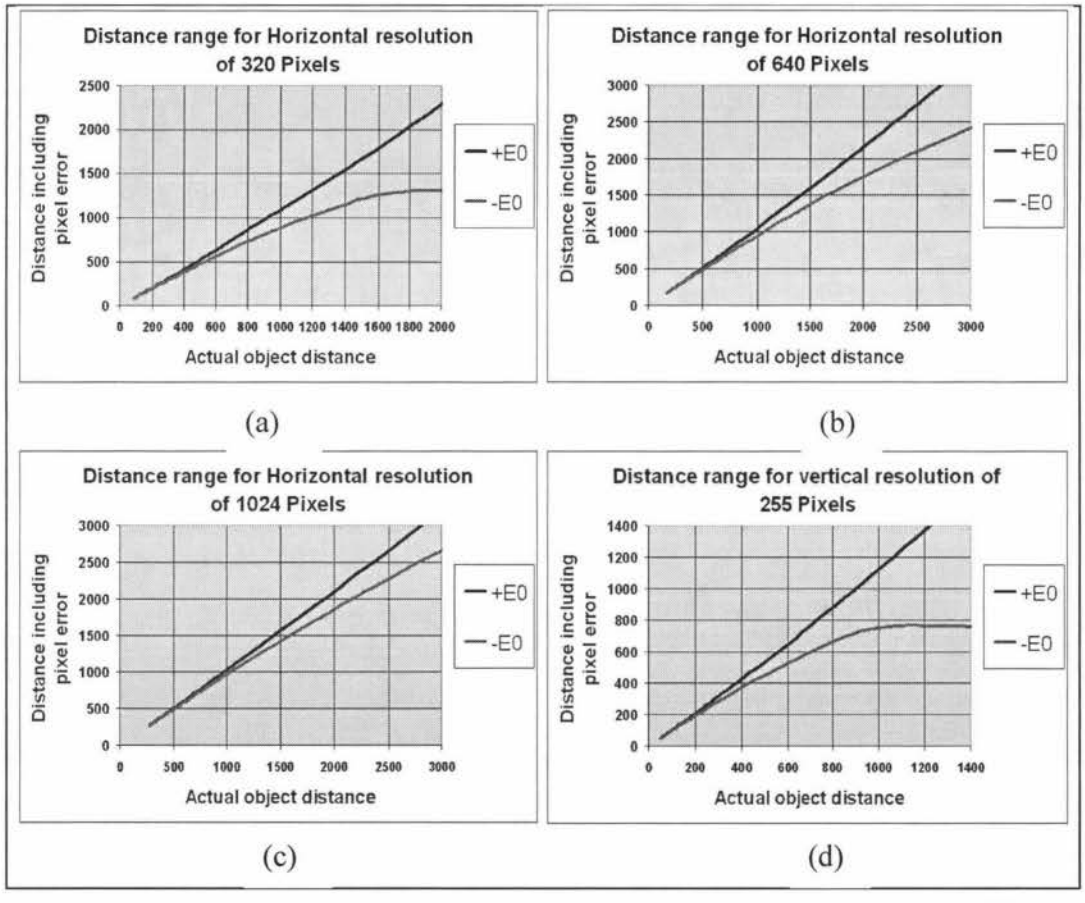


Figure 11: Graphs of different pixel errors for various resolutions (graph (d) is the vertical error of the CMUcam2 resolution with 33^0 view angle).

Another improvement that can be achieved with the large resolution is that the view angle of the lens can be increased without much degradation of the distance

calculations. This increase in view angle of the lens will allow the robot to view more of its surroundings allowing less need to search for objects, thereby speeding up the robots response time.

3.1.2: Stereo Vision

Human vision is based on stereo vision. It is a system that allows more accurate depth perception to be calculated [Florczyk] and, unlike monoscopic vision, the object size does not need to be known.

For the calculation of stereo vision a simple case is used in which one camera looks straight at the ball while the second camera looks at the ball at an angle. The above method has been chosen for ease of calculation of the accuracy. Both cameras find the midpoint of the ball, and along with the distance between the cameras (D_c), the Pythagoras theorem can be used to calculate the distance from the cameras to the ball as illustrated in figure 12. From this formula it is possible to see the accuracy of stereo vision is proportional to the distance between the cameras. As with monoscopic vision the distance from the robot to the ball has to be calculated using equation 4. The implementation of stereo vision above may not be the most capable system to employ but allows easy calculation of depth perception efficiency for stereo vision.

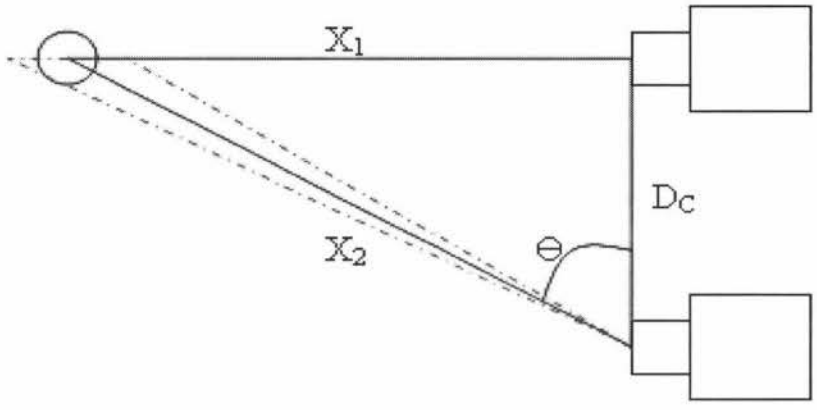


Figure 12: Stereo vision depth perception calculation

A drawback of this system is that the added size and weight of the required apparatus affects the centre of gravity of the robot, reducing its balance because of the robot's small size.

For this system the resolution error can be examined to see the effect of the objects central pixel in the cameras image having an error of 1 or 2 pixels. Unlike monoscopic vision this error relates to the object moving vertically or horizontally rather than its size increasing. For this calculation the vertical aspect has been removed for simplification. The case of the error being 2 pixels would occur if both cameras have an error of 1 pixel in opposite directions. An advantage that stereo vision has is that, if both cameras have an error in the same direction, the error is effectively cancelled out. The error can be calculated using the known field of view of the lens (V), and horizontal resolution of the camera I. From these values the degrees per pixel can be calculated and the result multiplied by the number of pixels of error, giving the difference from the original angle, which transforms the initial equation for (X) to equation 7. Equation 4 from the previous section must be subsequently used for calculation of the objects distance in relation to the robot. The dotted line in figure 12 shows the effect of the pixel error.

Even though a one to two pixel inaccuracy for stereo vision has been looked at, sub-pixel accuracies can be achieved, especially with circular objects, due to the centroid

producing a stable 2D position. This leads to their centre of gravity position to be a floating point number. Therefore a sub-pixel error is more likely than being a whole pixel out. [Lepetit]

$$X_1 = D_C \tan\left(\theta \pm \left(\frac{V}{R} E_1\right)\right) \quad (7)$$

3.1.3: Monoscopic versus Stereo Vision

Before a vision system can be chosen for a task, the surrounding environment must be explored. In robot soccer certain variables are known. These include the size and colour of the ball, goals, and the dimensions of the field. This allows the use of either monoscopic or stereo vision for depth perception.

The underlining question that needs to be addressed is whether the added accuracy of stereo vision for a soccer robot is worthwhile. Aspects that need to be considered include doubling the computational power required, and the need for a second camera thereby doubling the price and complexity of the system. Since a second camera would be required the added weight to the robots torso and head would raise the centre of mass of the robot affecting its ability to balance.

The next consideration that needs to be addressed is the robot's requirements for the task at hand. In the case of robot soccer, the robot needs to get to the ball faster than its competition, and then kicking it either at the opposition's goal or to a team mate. For this task the robot needs to know the location of the ball, and goal, in relation to itself, with the location of its team and the opposition also being useful to know. The accuracy of the ball location becomes more important as the robot approaches it. Therefore the greater the distance to the ball, the less the robot needs to know about its exact relative location. For the most basic artificial intelligence (AI), the robot just needs to know the general direction of the ball and to get to it as fast as possible. As it

approaches the ball, the errors in resolution gradually decrease until they become negligible.

Equation 7 in the previous section shows that the error ($^{+}/_{-}$(V/R x E_1), where E_1 equals the number of pixels out) presented by stereo vision has substantially greater accuracy at greater distances, and knowledge of the object size is not required. But, in the case of robot soccer, since sizes are known and the accuracy becomes less important at greater distances, a monoscopic system would give sufficient information.

A monoscopic vision system will allow distance to be judged via the number of pixels taken up by the object in question if the object size is known. The camera resolution therefore becomes important, especially in relation to the height of the robot being used. The direction can be determined by the location of the object in relation to the centre of the image and by the direction that the neck servos of the robot are facing. This system is currently being used by major competitors in the RoboCup [Behnke et.al 1].

This leads to the conclusion that, for a humanoid robot soccer environment, the advantages gained from stereo are minimal compared to the complexity involved in integrating them effectively with minimal error.

3.1.4: Camera

The camera module chosen for this project is the CMUcam2 produced by Carnegie Mellon University [Rowe], [Rowe et.al. 1], [Rowe et.al. 2], [Rowe et.al. 3]. The CMUcam2 can control 5 servos, therefore making it a viable option for this project because of the requirement of head control though the vision system. The resolution in this application will be set to the CMUcam2 high resolution mode, which only has a maximum resolution of 176 x 255 pixels. The OV6620 also allows a lower

resolution mode of 88 x 143 which could prove useful by allowing more frames per second, up to a maximum refresh rate of 50fps (frames per second) but since the distance implementation using the image size will be used, the larger resolution will be chosen, supporting a frame rate of 11-13fps.

The RGB colour space can have large colour value changes with intensity. Since the CMUcam2's YUV colour space will be used, a threshold for the intensity will be implemented to ignore anything that is too white or black allowing all colour processing to be done on the chrominance of the image. This dramatically reduces lighting effects caused by the intensity of an image.

Since CMOS cameras can only see colours between 16 and 240 instead of the full 256 for each colour, care must be taken that the limits for tracking the colour are set in the smallest possible range. For the application of robot soccer this will not cause significant problems since only several colours are to be tracked. In testing, results led to the orange of the soccer ball, for example, having the values V = 240 and U = 16, which were the same values as that of a red object. The camera would therefore have difficulty distinguishing between bright red and orange objects.

Another impressive feature of the CMUcam2 that could be used for the project is that a single camera can have multiple controller boards linked together in a master/slave setup. This could allow the camera to track multiple objects at a time, but will not be used in this project.

The CMUcam2 has some useful built-in image processing tools including colour tracking, frame differencing and histogram analysis. For this task the colour tracking of the CMUcam2 will be used, and will form the basis of all control in this project.

3.2: Implementation of Autonomy through Vision

The vision system being developed in this project includes real-time image processing, task trees and software to interface the camera and control the robot.

The initial goal of the project is to get the robot to track a ball and move towards it. The next objective is for the robot to locate the goal and position itself around the ball allowing it to take a shot at goal. The use of the colour tracking function of the CMUcam2 will be used for this task.

Since the CMUcam2 can perform colour tracking and has built in servo control it will be used for ball location and tracking. The control will be implemented via the Pirkus' newly developed neck turret for pan and tilt of the camera explained later. This neck will permit the Pirkus to track the ball at all times, allowing the body to follow the head's bearing using the value returned via the CMUcam2 control board.

The CAD model of the initial test rig design of the new neck can be seen in figure 13. The new neck design with its pan and tilt functionality will allow tracking of an object such as a ball in real-time. The real-time neck will allow the body to complete its current static motion before adjusting its trajectory to match that of the head. This will prove rather effective in the soccer application since the ball is not likely to remain in one location for an extensive period of time.

For this project control algorithms have to be developed to allow the robot to become autonomous using its sensory devices. This requires data acquisition, and decision tables to be developed.

The language chosen for this coding was Microsoft Visual Basic 6. This language is interrupt driven, which will prove useful when all the systems are implemented

together, allowing program control to be passed through external interrupts. Another advantage of using VB6 was that a GUI could be created for development of the robot, while also enabling an easy to use calibration system, that the end user could make use of.

The Pirkus' static motions are stored on its controllers EPROM and can be accessed through serial communication via its C program. The information processed via the Pirkus' controller boards will be communicated over serial to a PC. The programming on the PC will be programmed using visual basic, which will suit the task based system in hand. The visual basic program contains the intelligence of the robot and will have to be streamlined to run on a PC-104 Board for the robot to have all its processing completed onboard to comply with the RoboCup rules.

3.2.1: Robot Motions

The walking motion of the robot for the system in question requires several static walking motions. Static walking motions were chosen for the initial design of the robot to allow ease of A.I. development without the need to develop a dynamic walking model, although a dynamic walking system will need to be developed for future improvements of the robot platform.

Static motions for the Pirkus are made using the motion editor software provided with the robot. The home position of the robot must first be setup. This will be the reference position that all other motions will be based upon. The home position for the Pirkus robot will be set so it is standing straight up. All other motions will involve setting the servos around this home position. Once a motion has been created it is stored to the Pirkus' control boards EPROM at a location numbered from 0 to 99. Each of these motions can have 27 different frames, so each motion was designed to fit in one space on the EPROM.

The first motion that will be required is for the robot to walk in a straight line. Two different speeds would be an advantage, e.g. a slower walking speed for when the Pirkus gets near the ball to prepare for kicking. Along with the straight walk, a curved walking motion would be beneficial for slight corrections to the path required to be travelled but will not be developed for this system. Finally a stop and turn and a strafing motion are required for sharp turns and positioning of the Pirkus around the ball and goal. The motions created for this platform are:

- Walk straight ahead
- Turn left
- Turn right
- Strafe Left
- Strafe Right
- Kick
- Stop

The turn motions are designed so the robot turns around a circle which will allow the robot to more easily position itself around a ball. For ease of design the stop motion was set to halt the robot from its current motion and return to its home position.

The design of the code on the robot would allow a single character motion command to be sent via serial from the PC to the Pirkus controller board. Each motion command would be coupled with a motion stored on the Pirkus' EPROM. Once the motion is chosen, the series of key frames are played and the robot will perform the motion, followed by an acknowledgment sent back to the PC to let it know that the command has been received.

Once a motion has been started it has to be completed before another motion can be called. This will stop the robot changing its motion part way through, which could cause it to fall. The only motion that can interrupt other motions is the "stop" motion.

3.3: Device Integration and Interfacing

Since off the shelf systems are being integrated together for the final product, the communication and power requirements need to be developed to function as required. This is so the different devices can communicate between each other as well as getting the power that they require.

3.3.1: Device Communication

The CMUcam2 and the Pirkus both communicate through a RS232 serial connection, therefore the computer that is used to process the A.I. algorithms needs to be compatible with the serial communication. The compass board uses I^2C to send its values. Since the board in developement to control the balancing of the robot is using a PIC18F2520 microprocessor which has built in I^2C, it was chosen to take the compass data and pass it on to the robots main computer through RS232 serial. This means that the A.I. program needs to take in 3 serial communications, 2 at 115200bps and one at 57600bps. As well as receiving from the 3 sources, the A.I. program needs to send commands back to the robot and CMUcam2 control boards.

For communication with the CMUcam2 controller board a constant stream of values are taken from the camera. To test if a packet has been received, the input buffer is checked to see if it is greater than zero; if so the packet gets split at the carriage return which is added to the end of each packet sent. Since several packets may have been

received, only the last and most recent packet is chosen. The start of the packet is checked to see if a 'T' is at the front, which is always the first character for a tracked colour packet. Once the 'T' has been found the packet is split at every space, which separates the data in each packet. The variables received are placed into global variables related to that data type. For the case of a tracked colour packet the data received includes the middle x and y co-ordinates of the image; the x and y co-ordinates of the bounding box around the image; a ratio of the number of pixels found inside that bounding box; the confidence number, which relates to the ratio between pixels counted and the size of the bounding box; and finally the vertical and horizontal servo positions. All values that are received are between 0 and 255 so to fit in one byte.

The compass information received from the PIC microcontroller is handled in the same fashion as the CMUcam2 board explained above. In the case of the compass board the only information that is needed is the value of the compass in degrees, which is then stored in a global variable with each pass through the main program.

After commands are sent to the robot and CMUcam2 board, an acknowledgement is sent back (ACK) – this is to make sure that the packet has been received correctly. If an 'ACK' is not received the packet is sent again until the 'ACK' has been received.

3.3.2: Power Requirements

Since several devices have been chosen, their individual power requirements must be combined to choose the required batteries for the system. The Pirkus comes with a 6V battery made up of four 1.2V batteries with a 750mA hour; this is required to power the controller board, gyroscopes, and the 21 servos of the Pirkus. These can draw between 1A to 4A depending on the load on the servos, with the battery having approximately 15 minutes battery life, which is average for a walking robot

[Kitano_et.al.]. Through research into the servo motors they perform best at 7.2V, which means by adding another battery in series this can be achieved.

Since the CMUcam2 only draws 20mA and the PIC microcontroller circuit draws approximately 30mA and both have there own 5V regulator they can be run off the same battery as the Pirkus, which will not significantly lessen the batteries run time.

The next consideration required is for the PC104 computer board chosen, which uses 1.8A typically, and requires 5V $^{+}/_{.}5\%$. A battery identical to the one used by the Pirkus could be used with a 3A regulator circuit to power this board. The regulator circuitry could be included on the PIC microcontroller board and both these systems could run off the same battery.

The addition of a second battery will add extra weight to the robot. By placing both of the batteries in a newly developed foot, the robot would benefit from a raise in height and a lowering of its centre of gravity, therefore making it more stable in the same way as the Robosapien robots.

For the design and development of the robot, the batteries and the PC104 board will not be used. Instead the PC104 board will be replaced with a desktop computer and a 6A power supply set to 7.2V will be used in place of the batteries. This will allow more development to be focused on the A.I. of the robot.

3.4: Robot Mechanical Design

This section will go through the mechanical design of the new head and neck joint which will not only house the camera but allow the robot to have a neck that has the functionality as close as possible to a human.

As stated earlier the camera will be mounted on the Pirkus' head servo with an extra servo added to allow the neck to tilt on the vertical axis, allowing full tilt and pan of the camera as shown in figure 13. The control boards will be housed on the torso of the Pirkus and connected via a ribbon cable to the camera board. The housing for the control boards must be designed so it can be attached to the Pirkus while still keeping the robot within the required dimension ratios as stated in the RoboCup rules [robocup].

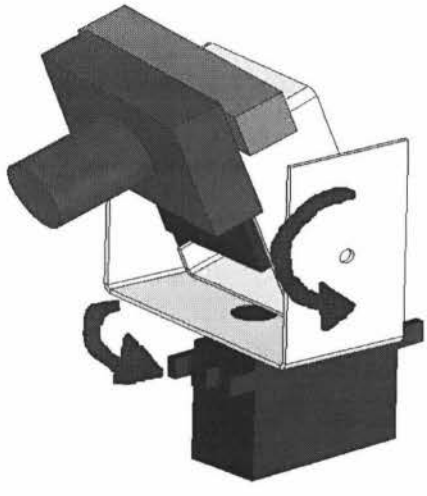


Figure 13: New neck joint initial design

The new head would be modelled on SolidWorks so that a prototype could be made using a FDM (fused deposition modelling) rapid prototyper which forms a 3D model out of ABS (Acrylonitrile butadiene styrene) plastic. This was chosen for the head

design not only for its availability and speed of manufacture, but also because it does not conduct electricity, therefore there is less risk of shorting out the camera and control boards.

To save redesigning the chassis of the entire robot to house the camera controller board, brackets would have to be added to the head to allow the controller board to be mounted without causing significant loss in movements, while keeping the head as small as possible to meet or be close to meeting the size restrictions in place. In the images in figure 14 the mounts are shown by the protrusions at the back of the main body of the head in the first image. The controller board is able to mount onto these two mounts. The curved pieces at the base of the head are for support of the head in case the robot was to fall; this would lessen the stress on the horizontal servo horn so it is less likely to fracture.

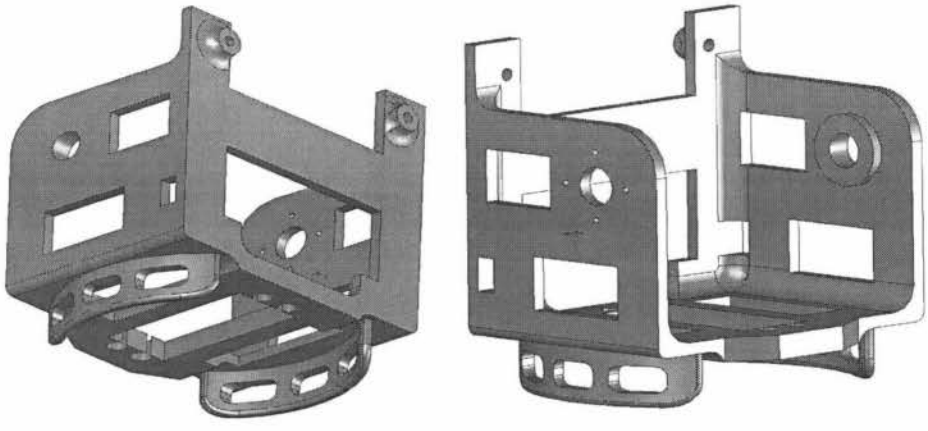


Figure 14: SolidWorks model of the main body of head joint

Since the head is going to be made from a layer prototype process, the joints need to have fillets added to strengthen them. The walls for the horizontal servo mount was thickened to add extra strength to the point where most force was going to be present. In Figure 15 the way in which the servos connect to the robot body and to the main head joint of the robot is illustrated.

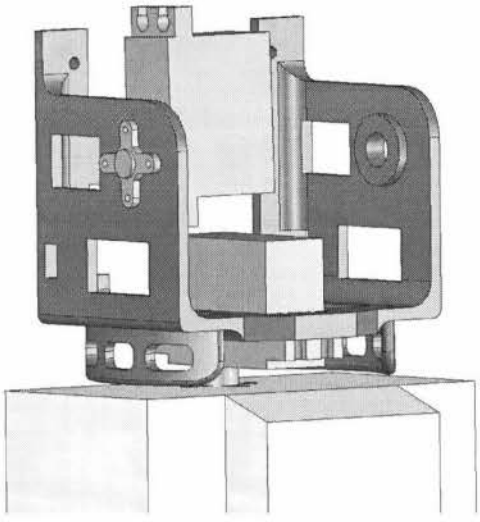


Figure 15: Servos attached to main body of head

To allow for the robot camera to be able to tilt up and down having a greater range of view, an inner head joint is made to which the camera board will be mounted. This joint needs to connect to the main head joint and the vertical servo while being free to rotate about its axis. In figure 16(a) the bracket on the right is for the vertical servo to mount to, as shown by images (c) and (d), and the servo horn will mount to the main head as in figure 15.

The side protrusion on the inner head (seen in figure 16(a)) of the inner head passes through the large whole on the main head. It is on this that the inner head joint will rotate. The large circle protrusions on each joint will act as washers to reduce the friction while the head is moving.

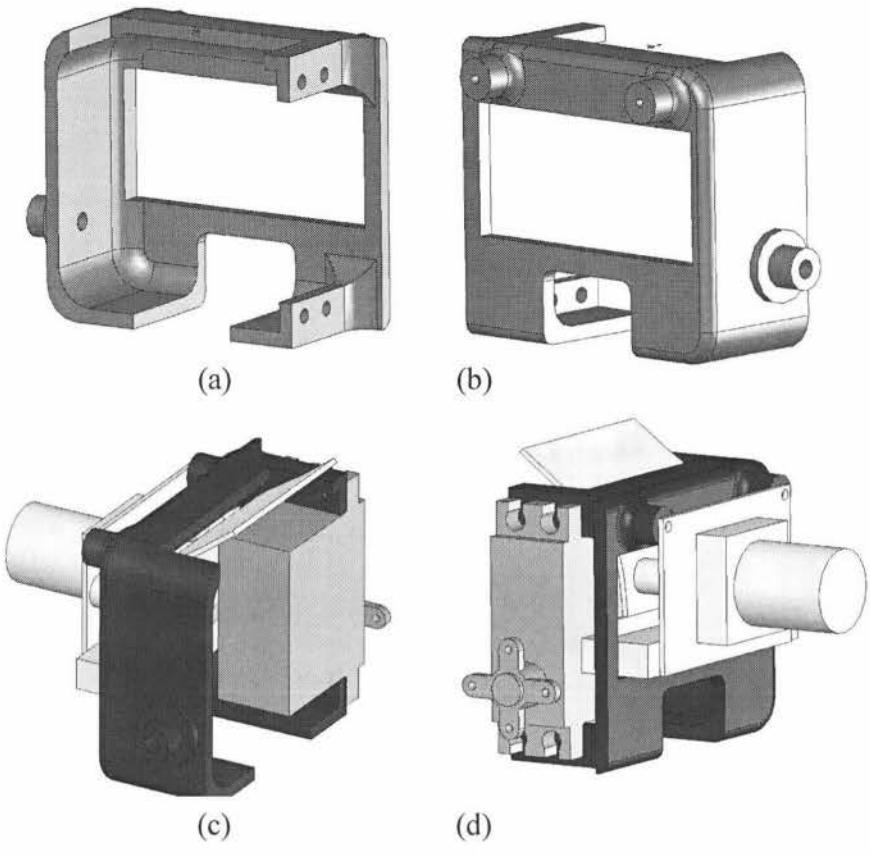


Figure 16: SolidWorks model of the inner head joint and assembly

The camera board would be fixed to the mounts at the top of the inner head; the orientation of the camera was chosen to place the least strain on the mounts of the camera board. This orientation would mean that the image captured would be upside down. By adding the jumper to invert the vertical servo direction while the object is being tracked, the tracking function operated correctly with the inverted image.

A series of holes were cut to allow for clearances and cables to pass through the inner head. The hole at the front of the joint is to allow the ribbon cable and connector to pass through. This is best illustrated by figure 16(c). The hole at the top of the inner head is for the ribbon cable to pass through. It is also to reduce the force effect that the stiff cable produces when the camera looks up. Instead the ribbon cable has more

area that is able to flex, therefore lessening the amount of deflection of the cable and thereby reducing the opposing force.

The hole on the base of the inner head allows it to be positioned lower without touching the horizontal servo while tilting (shown by the section view of the robot head design in figure 17), allowing the size of the head to be decreased. This decrease in size is important due to the head having to be within certain size criteria for the RoboCup rules as stated earlier. By decreasing the size of the head the required size of the body is reduced, which means less modification is required to the rest of the robots body to meet these requirements.

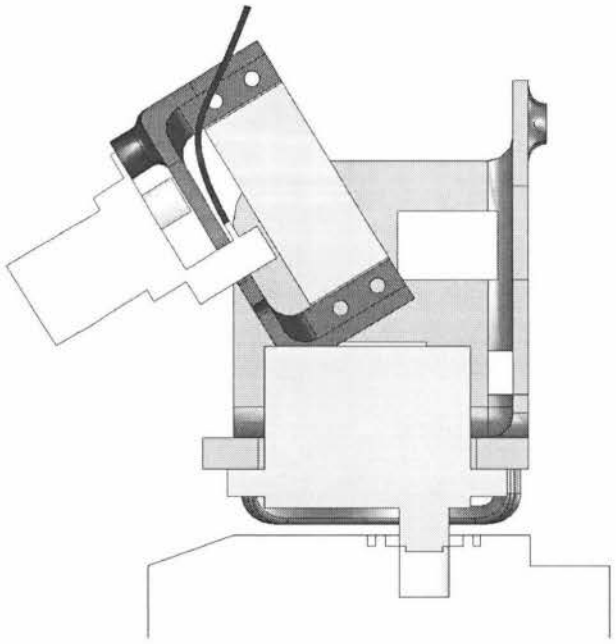


Figure 17: Section view of head assembly

Finally a series of rectangular slots were removed from the model not only to reduce weight without too much loss to the structural integrity, but to allow holes for cables to be fed through. The holes also decrease the amount of material needed for the prototype, therefore reducing costs.

The final design of the robot head and neck can be seen in figure 18 as well as the section view in the previous section. The SolidWorks model would be used to construct the prototype.

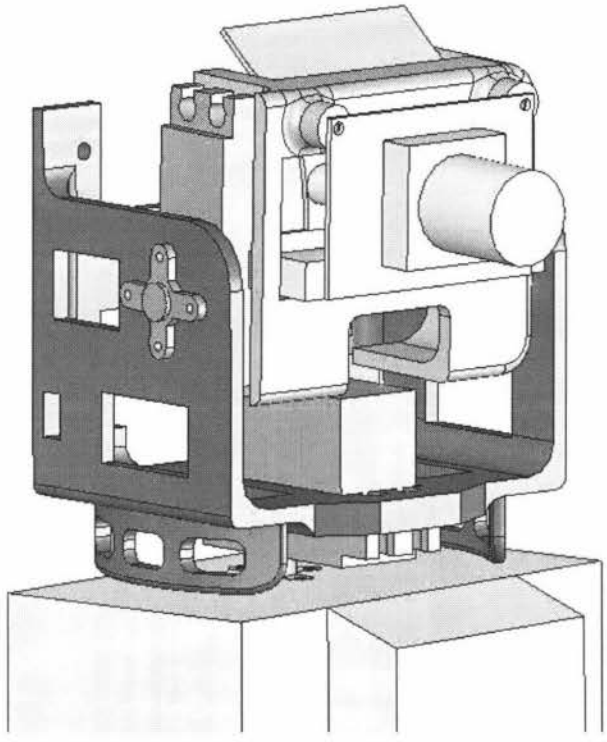


Figure 18: New neck joint final design

After the prototype was printed a ribbon cable was formed (shown in figure 19). In addition to the camera, the digital compass needed to be added. A housing for the compass board was designed to be attached to the back to the head (figure 19). The compass had to be added to the head due to the magnetic noise created by the servo motors which affected the compass readings.

Due to protruding screws and the addition of a backpack to house the microcontroller for the compass and balancing circuitry, the neck bracing of the robot had to be removed. The addition of the head and backpack also meant that the robot's ankle and hip servos struggled with the added weight and the rise in the centre of gravity, which needed to be taken into consideration when forming the robot's motions.

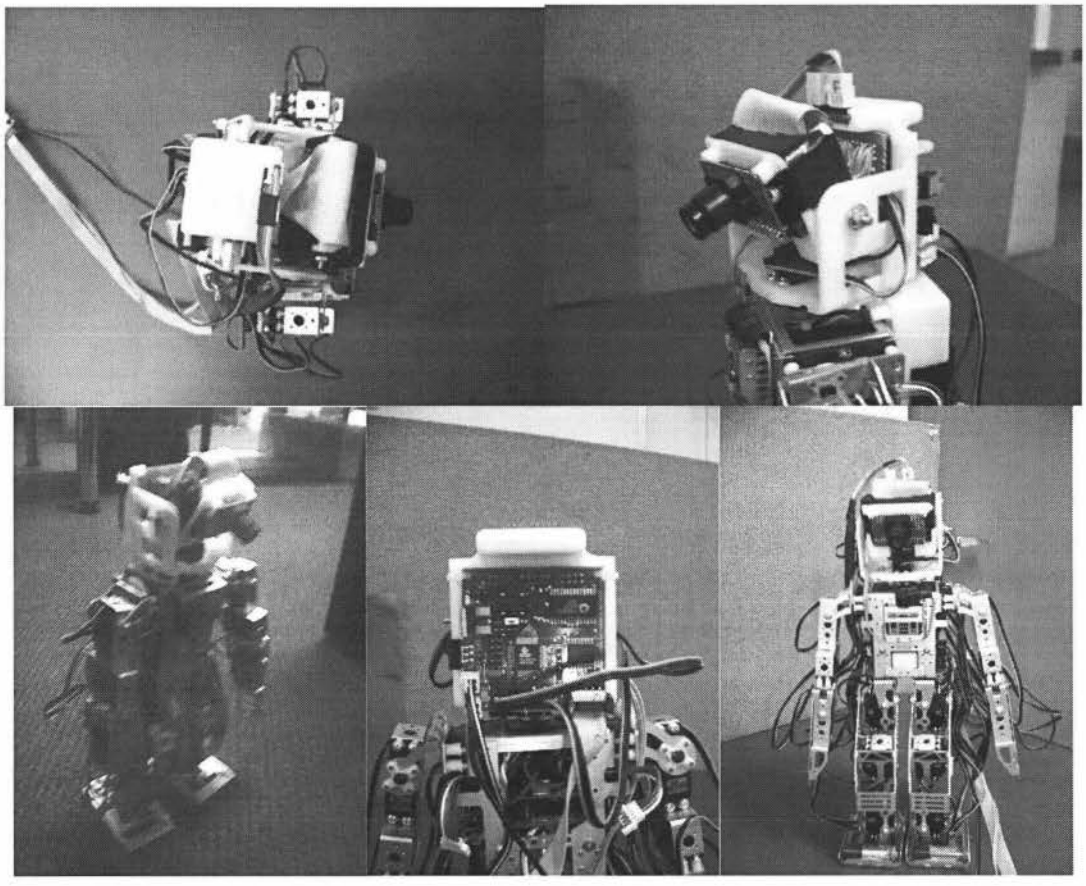


Figure 19: Final design of robot

Chapter 4: Humanoid Robot A.I.

For the Pirkus to be able to function in the RoboCup, the robot would have to be able to perform certain tasks. The main task focused on in this project is the goal shoot-out, in particular the offensive situation. On offence the robot will be required to find the ball, get to its location and shoot it at a coloured goal. The object colours and sizes are all known variables including the field dimensions.

Therefore an A.I. routine must be created to allow the robot to have the required autonomy to perform the necessary tasks.

4.1: Basic A.I Using Point and Follow Routine

The concept behind using a point and follow routine is as follows. The robot will run on basic instinctive control. This control involves the robot following its heads orientation to drive its motion. The chosen method of motion was to use a series of static predefined motions for initial control of the robot. The position of the two neck servo motors along with the data delivered from the camera would be used to select the appropriate motion. For this concept the tasks that the robot must preform need to be selected and a task coupled with a corresponding action.

4.2: Tasks

Before the robot can be told what to do with the information it is given, it must first know what is required of it. In this case the robot has been designed to be a striker (attacking player). This means the robot's fundamental required tasks are as follows:

- Find the ball
- Get to the ball
- Find the Goal
- Position around ball to line up with the goal
- Kick the ball at the goal
- Repeat

For these tasks a series of routines have been created that the robot is able to call upon depending on the values it receives from the CMUcam2 data packet.

4.3: Order of Events

From these tasks a truth table can be formed to check that each of these tasks has been completed in their required order. To develop an appropriate order of events a flow diagram, (figure 20) must first be formed. In figure 20 it can be seen that once started the robot, after performing its final task, will return to the start of the routine thereby staying in an infinite loop.

For this part of the A.I. a series of flags will be required which will allow the robot to check that the task has been completed and it can proceed to the next task. Along with

the task flags an error flag will be required in case something out of the ordinary occurs. This flag can then notify the robot to restart the routine. The flags that are included in this routine are:

- Ball found
- Goal found
- Ball in position
- Ball check
- Goal check
- Shot taken
- Error flag

From the order of events shown in figure 20, the robot starts with looking for the ball using the "search" routine. After the robot has found the location of the ball the robot starts to move towards the ball. While moving towards the ball the robot has the ability to call three separate motions – these include the walk motion and the two turning motions. If the ball is located outside a certain range of the horizontal servo axis then the robot turns in that direction then proceeds to walk, whereas if the ball is within the required range the robot will simply walk in a straight forward direction.

When the robot has reached the ball the "Ball found" flag is set letting the robot to run the position routine where it must first find the goal, again using the search routine. Once the goal has been found the robot repositions itself using a turning motion around the ball to face the goal.

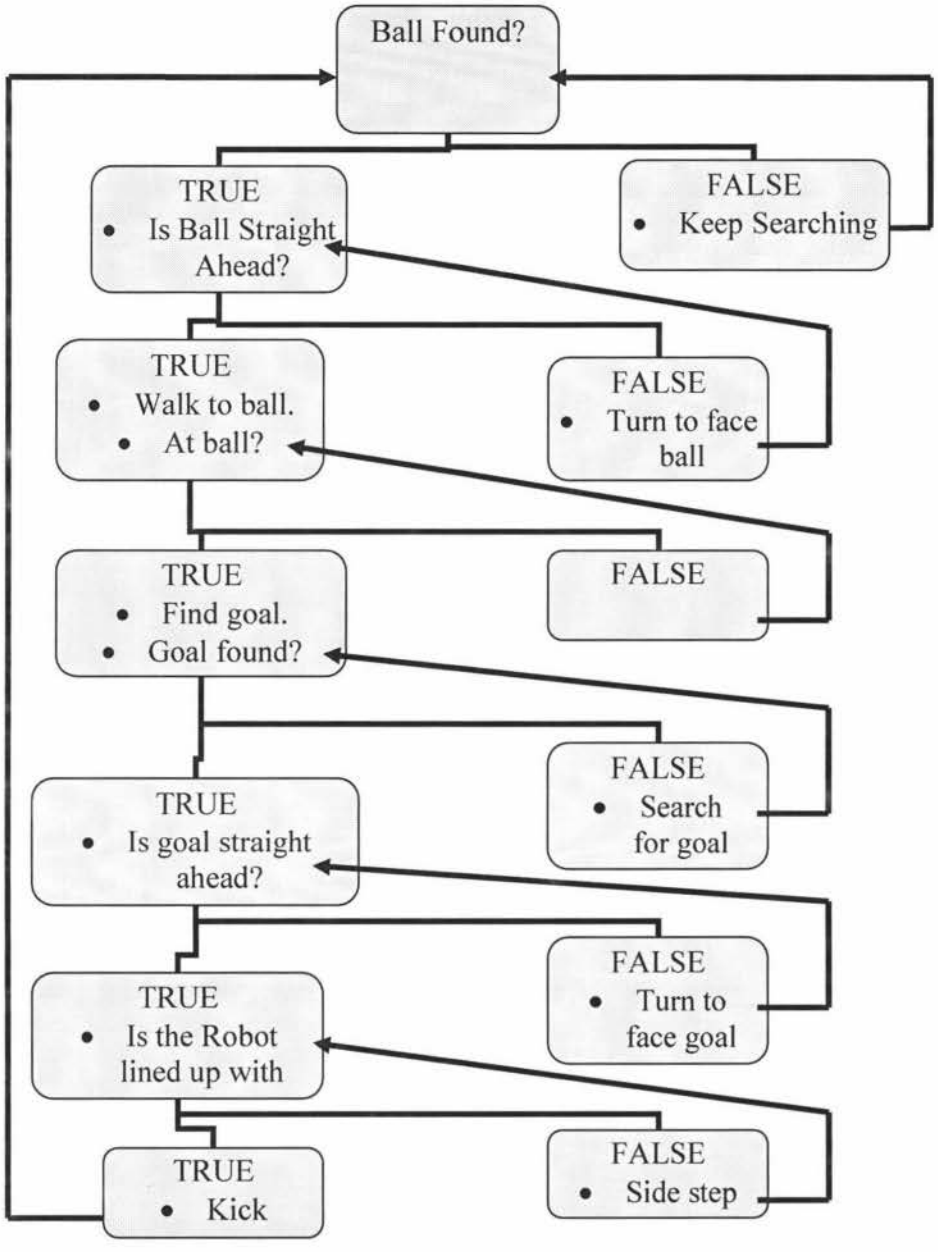


Figure 20: Initial Truth Table for Robot A.I

Once in a position where the goal is located in a chosen range of the horizontal servo the "Goal found" flag is set. Once this flag is set the robot is then able to side step in either direction till the ball is located in front of its kicking foot putting the robot in position to kick. It can then call its kick routine which tells the robot to initiate its

kicking motion to score a goal. Once the kick routine is performed the "Ball error" flag is set to restart the sequence in case a goal has not been scored.

4.4: Software Routines

For the required tasks to be performed several routines must be developed in the code and called in the appropriate order. The series of routines created are:

- Search
- Get to Ball
- Position to Ball
- Shoot Goal

Through these 4 main routines the majority of the autonomous control of the robot is performed.

4.4.1: Search

The search routine is called inside the "Get to Ball" and "Position to Ball" routines when the object that is being tracked is not found by the camera in its current location. The object is considered to be out of view if the number of pixels tracked and the confidence value of the camera packet are below a certain value. The main colours are orange for the ball and yellow or blue for the goals.

The search function is made up of a series of cases in a select case function. Each case has a specific area for the robot search. The function works by first finding the colour that is required to be tracked and passing its colour space values, along with the command to track the colour via the serial connection to the camera board. The camera board is then sent the command to move the horizontal and vertical servo

motors to the location at which they should be pointing. The function then passes back control to the main program to get the next packet from the camera to see if the ball is located in the area now in view. If the ball is not in this location the control is passed back to the search function which then moves onto the next case and a new location is searched. The case chosen depends on the object being tracked since different object are more likely to be found in different regions in relation to the robot. This routine continues until the object is located.

Since the goal is large and will always be found on the horizon the robot will be able to skip some of the cases that require the robot to look at the ground at its feet. This allows the 'search for the goal' task to be completed faster than the 'search for the ball' task.

In the case where the ball is not found the robot is then ordered to turn left and search again. This process is then continued until the object is found. A worse case scenario will require the robot to complete a full circle before the object is found.

4.4.1.1: Field of view

Due to the camera having a limitation of view angle from the focused lens, the field of view of the robot would have to be considered when constructing the search function.

The camera has a field of view of 44^0 in the horizontal and 33^0 in the vertical plane. To speed up the search routine the least number of points need to be searched that cover the largest area without having blind spots is required. To increase speed a 2 run system was chosen. By using trigonometry the best view of the camera can be calculated.

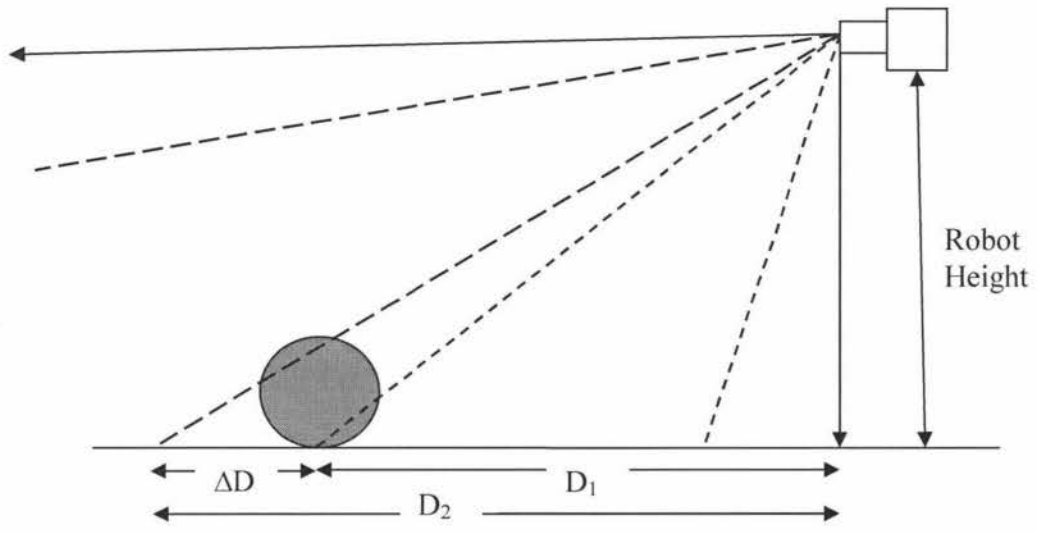


Figure 21: Vertical Search

For the vertical view there are only 66^0 out of the 90^0 needed for a complete view of the field, therefore there will be some areas that are lost. By using a top gap of 6.4^0 and a bottom gap of 9.4^0 to fit in with the real servo values, the limitations of this system can be calculated Due to this the view of the search function is limited to the following:

Variable	Distance in millimetres
Distance (Max.)	2800
Distance (Min.)	57
D_1	273
D_2	383
ΔD	100
D_2 (including ball height)	298
ΔD (including ball height)	25

Table 2: Limitations in vertical search

From Table 2, ΔD is the distance of the robots blind spot which is found between the maximum view distance of the close search, (D_1) and the minimum view distance of the far search, (D_2). The values "including ball height" take the height of the ball into the distance calculation which shortens the D_2 and ΔD distances.

Due to the limitations of the 2 run pass, the maximum distance that the robot will be able to see is approximately 2.8 meters. This will mean it is incapable of seeing from one side of the field to the other, but should allow the robot to see far enough to function as a striker of the ball.

By including the balls height into the D_2 calculation in the robot's search routine, a gap that would have been calculated to be presenting a blind spot for the robot, it would in reality actually be able to see the ball.

Since the horizontal field of view is larger at 44^0, the robot can cover 120^0 across the horizontal plane if the views are overlapped by 4^0 each, this will mean that there is no blind spots across its horizontal plane, which could lead to very large blind spots at larger distances.

4.4.2: Get to ball

The "Get to Ball" function is the first function to be called after a packet has been received from the robot. This function controls the robot's thought pattern from searching for the ball to getting to the balls location. The function is comprised of a checklist of "If" statements. The list of checks and actions performed are:

- If the ball is not located, then search for ball.

- If the ball is not at the robot's feet, but is straight ahead then walk forward.
- If the ball is not at the robot's feet, and is to the robots left then turn left.
- If the ball is not at the robot's feet, and if to the robots right then turn right.
- If the ball located at robot's feet, set "Ball Found" flag, then search for the goal.

Limits need to be set to define when the ball is straight ahead and when it has reached the ball. The chosen method for this is to use the horizontal and vertical servo values to define the limits, along with the objects pixel and confidence values to make sure that the object being tracked has not been lost. For the check to see if the ball is at the robot's feet the vertical servo value and the object size is used. Once the servo value has passed a required value and the object is above the required object size, the robot will stop and return to its home position. The horizontal servo value will be used to define the direction of the ball in relation to the robot. Once the direction of the ball is calculated the appropriate motion command can be selected and carried out. The advantage of using these values is their simplicity, which lowers the amount of code required to control these aspects of the robot's autonomy. At the end of the check, as with the search function, the control of the program is given back to the main function and the next packet is received from the camera.

4.4.3: Position to ball

Once the robot has reached the ball it is then required to move itself around the ball so it is in position to kick it at the goal. Although this function is slightly more complex than the "Get to ball" function it still follows the same basic concept of using a series of "If" statements to decide what is required to be done. Like the previous function this function also has a series of flags that if set allow access to different sets of "If" statements. As with all the functions once a single pass of the

function has been completed the control is passed back to the main "RunTrack_Click" sub-routine, so a new camera packet can be acquired and flags to be checked The list of flags, checks and actions performed are:

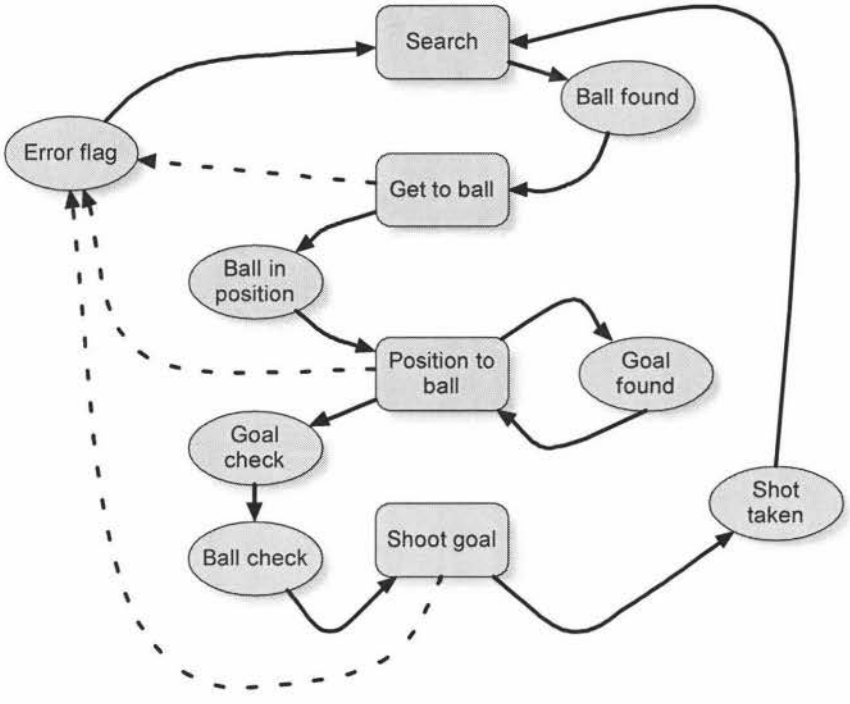


Figure 22: Flow diagram of flags

From the flow diagram above this function starts by finding the goal. Since the goal is bigger the cut-off value for the image size will need to be larger than the 'check for the ball', and like the "get to ball" function every check makes sure the object in question is still seen by checking the pixel and confidence values in the camera packet each time.

As like the "Get to ball" routine, if the goal is to the robot's right or left the motion command is set to the appropriate value to tell the robot to turn in the required value which is controlled via the horizontal servo value received in the camera packet.

Once the goal lies in the required range, the robot can set the "Goal position" flag. This will trigger the robot to locate the ball at its feet again and make sure that it is in the right position to kick. If the ball is outside the required range a command is sent to tell the robot to sidestep left or right to position the ball in front of its kicking foot. When the ball is in the appropriate range the robot then checks the goal, then ball once more to make sure that everything is still lined up. If the robot is not lined up the "ball error flag" is reset and the robot restarts the positioning checks. If everything is in position the "Ball position" flag is set which will allow the robot's main program to call the shoot goal routine.

4.4.4: Shoot Goal

This is the most simple of all the routines. This function sets the motion command to "k" which tells the robot to call its kicking motion then waits for a few seconds to allow the robot to kick the ball. Once the ball has been kicked the robot sets the "ball error flag" to reset the system to the start to track the ball again. The Shot flag is also set so that the first time through the code the robot does not get sent any motion commands that could interfere with the kicking of the ball and which may subsequently lead to the ball not being kicked or worse the robot falling over.

Since the camera points at the central pixel of the object, if an object such as an opposing player stands in the way of the goal then the central pixel will shift to the largest concentration of pixels. This in turn will turn the head to centre the object, therefore compensating for the blockade.

4.4.5: Special Cases

Several special cases can occur which will require the ball error flag to be set. The design of the error testing involves a final check of the ball and goals locations in relation to the robot to observe if the robot is positioned correctly to kick and score a goal. This test checks if the ball has moved to a position that can not be effectively kicked by the robot. This event could possibly occur due to the robot accidentally hitting the ball before it stops to look for the goal; the robot kicks the ball away when turning around it; the robot loses site of the ball or the goal; an opposing player interferes with the ball or blocks part of the goal. By restarting the routine the robot can quickly go though its check list of tasks and find which order of motions are required to get to position again and perform them.

These checks are done after the "goal found" and "ball position" flags have been sent since the robot has been moving around the ball at close quarters and has the highest chance of knocking the ball out of place. The first check involves the robot checking that the ball still remains within certain bounds of the horizontal and vertical servo positions. The second test involves the robot, after being in position, finding the ball and goal again using the search routine and making sure they still remain within their limits.

Chapter 5: Ball Path Planning

Once the robot is able to score a goal by using the point and follow method, path planning is implemented to improve its speed and abilities. The main aspect that will be investigated are an in-depth analysis of the ball trajectory planning for a moving ball.

5.1: Distance

For the robot to be able to track the direction and distance of objects a series of calculations and methods have to be used to first find these values. The routine to find the distance in a monoscopic system in a fixed environment has been covered earlier and will be built on here.

From the formulae in chapter 3 the distance can be calculated from the robot, using vertical and horizontal image sizes. Along with these two distances a third which was only just mentioned earlier is the distance calculated by via the vertical servo angle.

The camera focuses on the central pixel of the object, this central pixel and the height of the robot varies with the change of distance between the object and the robot, as shown in figure 23. Therefore equation 4 from chapter 3 needs to be modified to compensate for this change.

When looking at an average of the distances to get a more accurate and stable value, several characteristics of the individual measurements have to be explored. When the robot is calculating the distance from the object to itself, the error of each value has to be investigated. As explained in chapter 3 the error of the distance decreases with

resolution of the camera and size of the object, but increases as the distance of the object increases, shown by the graph in figure 11. The next step is to compare the error though using the vertical servo in finding the distance compared with the vertical and horizontal image size. As seen in figure 24 the servo value becomes less accurate with the increase in size of the object, but is not subject to the errors that the image size calculations have at close range especially if the ball gets partially covered by part of the robot being blocking the cameras line of sight. The servo distance is not as susceptible to "spikes" in its values which is found in the image size calculations when a foreign object acts as noise or when pixels are lost due to slight changes in the lighting of the area. Therefore it can be used to smooth out spikes from the system. The error in the servo distance calculation can be calculated by the servo being out by 1 value which would mean an error of 1.345^0 in either direction

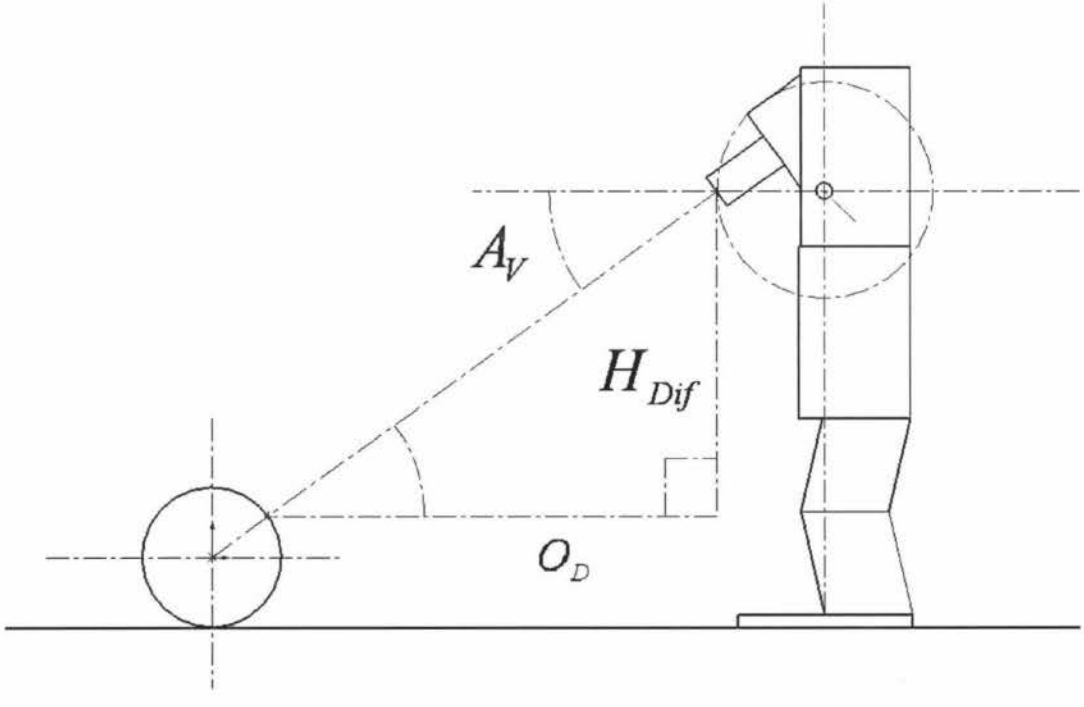


Figure 23: Actual distance of ball from robot

By finding the distance from the object to the camera two parts of the model need to be developed to reach a more accurate representation of the distance of the object from the robot in the horizontal plane, (O_D). Since the head is free to move up and down the height of the robot changes with the vertical servo angle and needs to be compensated for in the distance calculation. Along with the change in robot height, the distance the lens protrudes from the central horizontal axis also changes as the vertical servo moves about its axis.

Another factor that will effect the calculation is that the robot is not looking at the base of the object; it rather looks at the central pixel of the object, which lays half way up the image. This does not mean that the location looked at by the camera is half way up the object. The location looked at by the camera needs to be adjusted in the same way the height of the robot is adjusted.

If these adjustments are taken into account the object distance along the ground or horizontal plane from the robot shown in figure 23, can be calculated from the following formulae:

$$O_D = \frac{H_{Dif}}{\tan(A_V)} + Fix_Y \tag{8}$$

The angle of the camera can be found by taking the difference between the current vertical servo value and the servo value when the robot's camera is perpendicular to the front facing plane of the robot, then multiplying that difference by the angle per servo value.

$$A_V = 1.34 \cdot \left| S_{V\leftarrow} - S_{V_{Current}} \right| \tag{9}$$

From figure 23 the height of the robot and the height of the ball adjustments, need to be made to find the current height of the complete system. Therefore the remainder height, (H_{Dif}), can be found through equation 10.

$$H_{Dif} = H_{Real} - O_{H_{Real}} \tag{10}$$

The adjustments needed in the y, (horizontal) and z, (vertical) planes are found through the use of the sine and cosine trigonometric functions for the x and y distances around a circle, with the x axis of the trigonometric functions being given the value of the y plane and the y axis of the trigonometric functions given the value of the z plane from the robot's model. These adjustment calculations are illustrated in equations 11 and 12, where r is the radius of the circle. As seen in equation 10 before by adding the horizontal change, (Fix_Y) onto the distance to the ball, the horizontal distance accuracy is improved. These two equations need to be added to the original distance calculations using the horizontal and vertical image sizes.

$$Fix_Y = r \cdot \cos(A_V) \tag{11}$$

$$Fix_Z = r \cdot \sin(A_V) \tag{12}$$

The height that the camera lens from the ground, can be found by taking the height of the lens when perpendicular to the robot, and subtracting the vertical adjustment in the system, (Fix_Z).

$$H_{Real} = H_{R_{\leftarrow}} - Fix_Z \tag{13}$$

The actual height of the object that the camera is facing is found by taking the mid point of the object, which is half its actual height and adding the vertical adjustment to it.

$$O_{H_{Real}} = \left(\frac{O_H}{2}\right) + Fix_Z \tag{14}$$

After calculating the distance of an object from the robot the error in the system has to be calculated to give the range the ball will lie in if the servo is 1 servo value out. The positive and negative errors seen in figure 24 can be calculated using equations 15 and 16, and the error range can be found using found by using equation 17.

$$Error_+ = H_{Dif+1} \cdot \tan(\theta + 1.34°) \tag{15}$$

$$Error_- = H_{Dif-1} \cdot \tan(\theta - 1.34°) \tag{16}$$

$$Error_T = Error_+ - Error_- \tag{17}$$

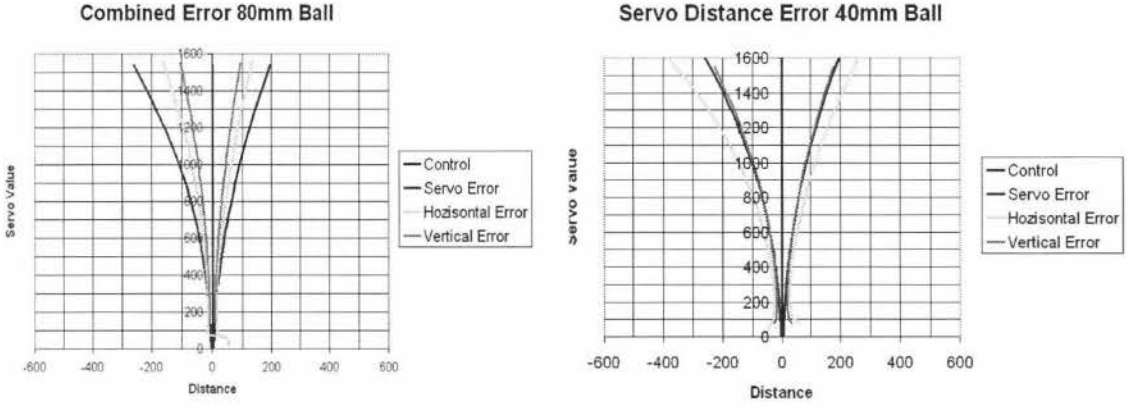


Figure 24: Comparison of distance calculation (a) 80mm ball (b) 40mm ball

Now the various ways of calculating the distances must be compared for their accuracies over a distance range and with various object sizes, for simplicity of the calculation H_{Dif} will remain constant with the height of the robot being the value of the robot's height when the camera is perpendicular and the object size remaining half the object height. Although the accuracy of the servo system will be slightly less

in the real case scenario this will give an overview of how well the servo system performs in comparison to the image size method.

5.2: Direction

The Direction in relation to the robot can be found through the horizontal servo value. This will inform the robot of the objects relative angle to itself. This system has some shortcomings, which are demonstrated when the robot moves in any direction. The relative direction of the object has been changed since the direction the robot is facing has also changed. Two solutions have been devised to counteract this problem.

The first solution requires the robot to stop moving when the ball is in motion to allow the distance and direction of its motion to be more accurately calculated. This now allows for the ball to move and the direction of the ball to be accurately calculated.

The second solution involves the inclusion of a digital compass the compass chosen earlier not only allows a more accurate angle to be found since it has a resolution of 0.1^0 compared to the servo resolution when controlled via the camera board of 1.34^0. The digital compass also gives the robot a global direction that the robot is facing. Since the compass has to be attached to the head of the robot to reduce its interference from the servo motors, the compass value can be used in conjunction with the horizontal servo to give the direction the robot is facing even if the robot has moved.

5.3: Distance implementation into software

The first implementation of the distance and direction into this system will be developed to allow the distance and direction calculations to be calculated and fine tuned to fit the robot's system. This first program will require the robot to remain in a stationary location and the distances from the robot to be calculated. The user interface of the program is shown in figure 25.

This program will allow the user to define the size of the object, the servo value of the robot when it is perpendicular to the robots front face, and the ability to adjust the actual viewed size of the object due to the outer pixels of the object getting lost through differing lighting conditions.

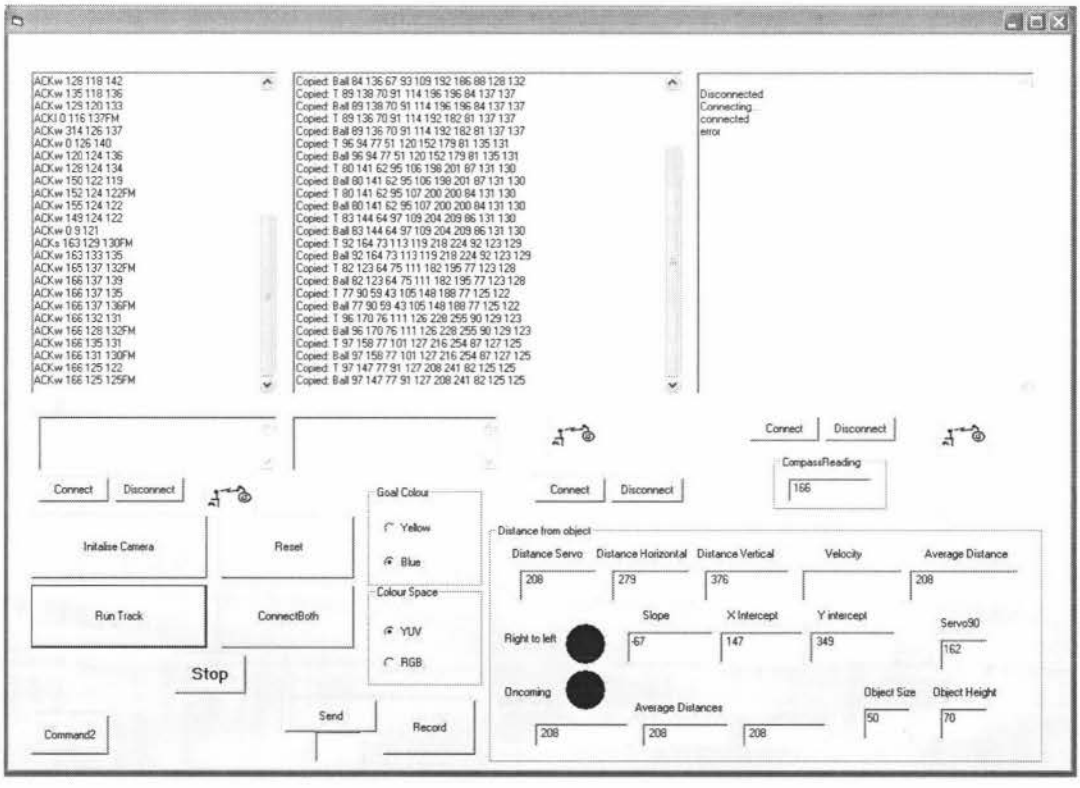


Figure 25: GUI for robot control

Boxes have been added to give the 3 distance readings, 3 different averaged distances using different methods, and the averaged distance that has been chosen for the later calculations. Along with these visual indicators, all the calculated and averaged distances will be saved to a CSV file to be later analysed.

The velocity of the ball and the vertical and horizontal cross, the direction indicators, as well as the travelled distance of the ball will all be setup to use the current distance and direction compared with the previous reading. This will be covered in the trajectory calculation section.

The goal of this program is to find how accurate the distance can be calculated in the real world environment using the 3 distance calculations and which averaged distance value gives the closest value to the actual distance.

5.4: Trajectory Calculation

Knowing if the ball is moving or stationary can be useful to the robot, but not without ability to work out where it is going. With the previous autonomy algorithms the robot would just run after the ball in its current location if the ball was moving it would not be able to judge if it was travelling away from the robot or towards it whether it was going left or right and where it would pass in front of the robot or cross the robot's path. By using the following trajectory calculations the robot could determine the velocity the ball is travelling and the time it has to reach a given location. Through finding the trajectory of the ball the robot would be able to utilise cognitive skills that a human soccer player would just take for granted.

Through finding the trajectory of the ball the robot will be able to try intercepting the ball. If interception is not possible, then the ability to turn and run in the direction of

its trajectory would be advantageous to help the robot get to its location before the opposition.

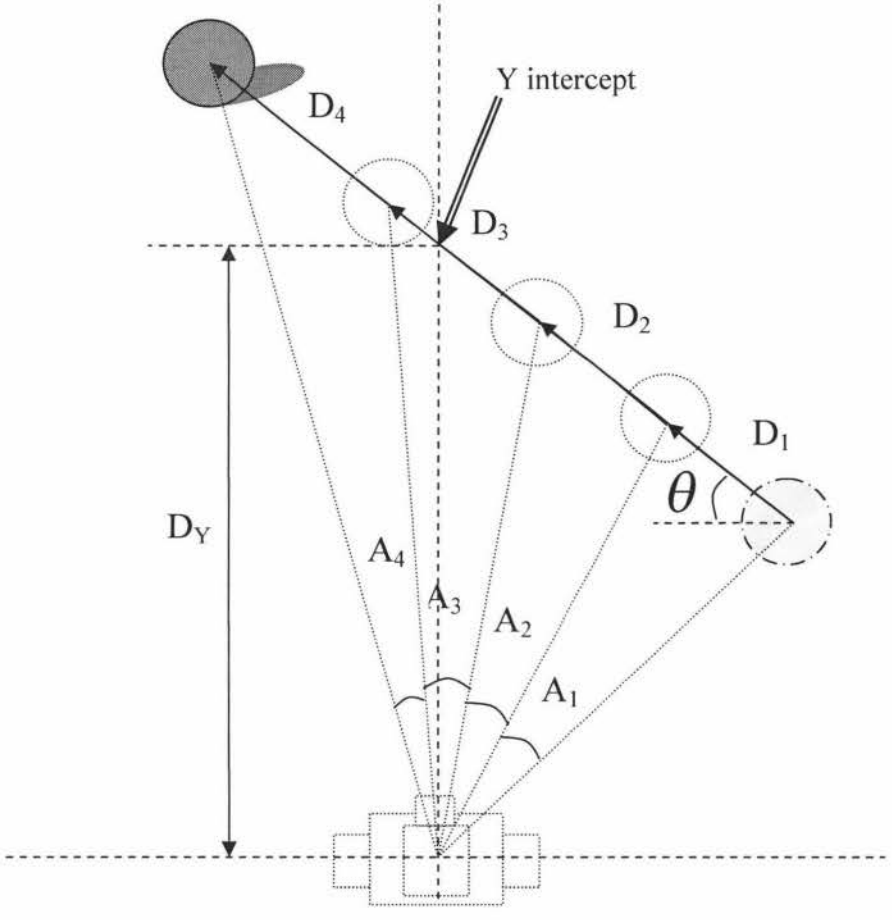


Figure 26: Ball travelling away from the robot

To calculate the trajectory of the ball, a series of points must first be taken. From those points the direction of the balls travel can be calculated, along with the angle of travel. One method of calculation involves getting two ball locations, as seen in figure 26. From these the two distances, D_1 and D_2 and the change of horizontal servo angle a vector can be formed to give the distance travelled D_T, by using the cosine rule as in equation 18. If the ball is heading towards the robot as in figure 27, then the horizontal intercept, (D_X), the point where the ball will pass in front of the robot

crossing its y axis, (D_Y), and the angle of the balls trajectory can be found using the law of sine's. If the distance travelled, (D_T) is measured over a time period, (t), then the velocity of the ball can be found. If multiple velocities are found, then the change in velocity can be calculated.

Therefore the total distance travelled by the ball from one point to the next is found by:

$$D_{T_1} = D_1^{\,2} + D_2^{\,2} - 2 \cdot D_1 \cdot D_2 \cdot \cos\left(A_1\right) \tag{18}$$

$$D_{T_2} = D_2^{\,2} + D_3^{\,2} - 2 \cdot D_2 \cdot D_3 \cdot \cos\left(A_2\right) \tag{19}$$

And the total distance travelled over a series of points are:

$$T_{D_T} = D_{T_1} + D_{T_2} + \ldots D_{T_n} \tag{20}$$

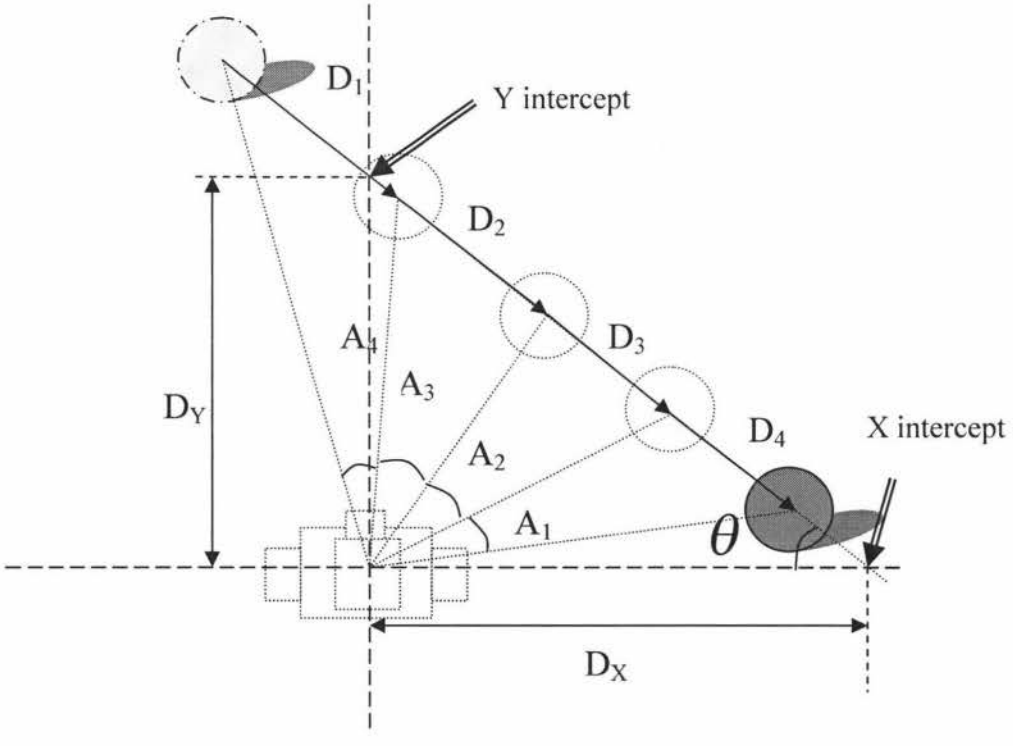


Figure 27: Ball travelling towards the robot

If the ball is oncoming then we can take the total distance travelled from the earlier calculation of equation 18 and applying the law of sines on it:

$$\frac{D_{T_1}}{\sin A_1} = \frac{D_2}{\sin \theta} \tag{21}$$

With the angle of the balls trajectory calculated using:

$$\theta = \frac{D_{T_1}}{D_2} \cdot \sin A_1 \tag{22}$$

Which then allows the distance from the robot to the intercept of the x axis (or the robots frontal plane) where the ball will pass, to be calculated:

$$D_X = D_1 \cdot \tan \theta \tag{23}$$

Also by finding the distance the ball has travelled, (D_T) in equation 18, this allows the velocity of the ball over that distance to be calculated, (V_1) as follows:

$$V_1 = \frac{D_{T_1}}{t} \tag{24}$$

This then allows the change of velocity to be found over 2 distances:

$$\Delta V = V_1 - V_n \tag{25}$$

These two velocity calculations can be used to determine where the ball will come to rest.

5.5: Implementation into A.I

Since the inaccuracies in the distance calculation of a monoscopic vision system, especially one with the low resolution of the CMUcam2 the trajectory model described earlier will not function as required in this robot platform. Figure 28 shows a moving ball passing in front of the robot's path. The dotted boxes surrounding the ball are the error in the ball location. As illustrated in this diagram the error decreases

in size as the ball becomes closer. The maximum and minimum trajectory angles that include this error are also shown. These illustrate, by using the previous formula the error in the balls trajectory can be very large and can vary greatly between measurements.

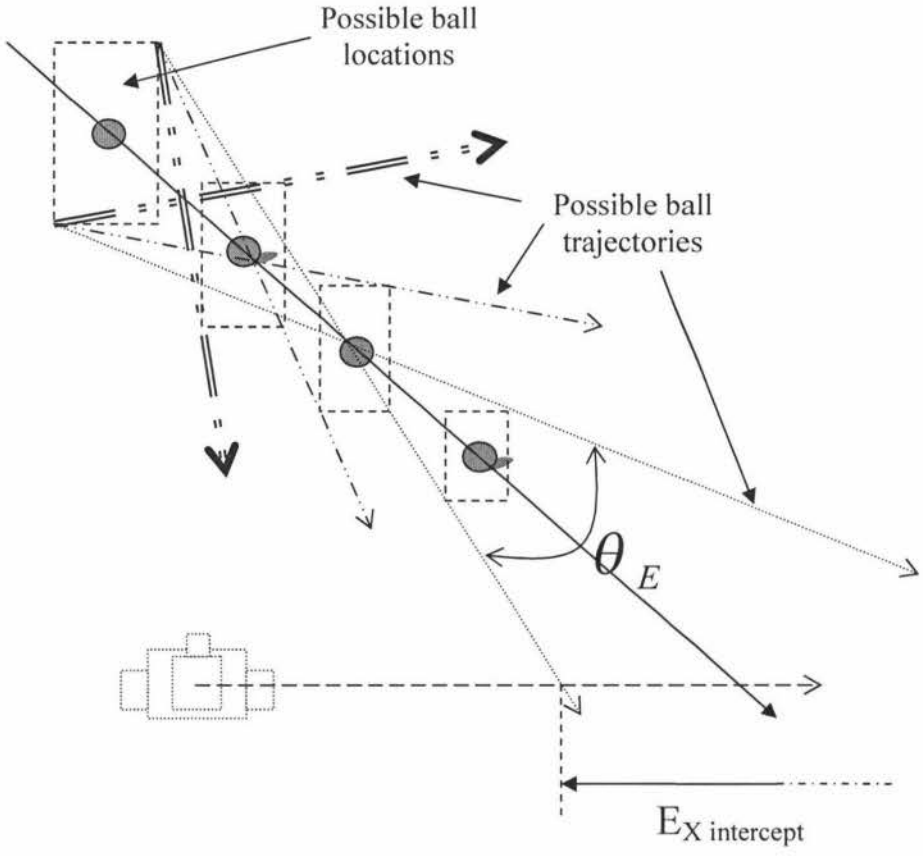


Figure 28: Error in trajectory calculations from monoscopic system

A solution devised to counter the problems that would be developed from this error would be to construct a line of best fit for the trajectory. One option for this would be to average the vectors developed from the above equations, to get a resultant vector for the trajectory, which would involve calculating a series of vectors then completing an averaging calculation on them. The Chosen idea for this project was a line of best fit.

By breaking the distance and direction into its x and y co-ordinates, the x and y co-ordinates of the ball in relation to the robot can be calculated through the use of trigonometric functions on the distance and direction. As demonstrated in figure 29, X is the direction perpendicular to the robot's front face on the horizontal plane and Y is the direction parallel to the front plane on the horizontal plane. The centre point of the calculation is located at the centre of rotation of the horizontal neck servo, which in turn will mean that when the ball is located at the robot's feet it will not be at a distance of 0.

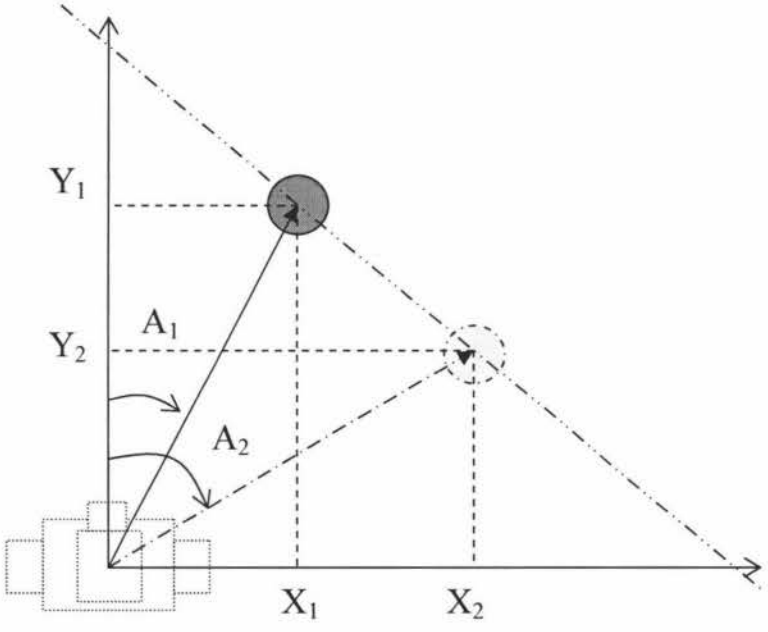


Figure 29: x and y co-ordinates of the ball

The angle that the ball is located in relation to the parallel horizontal plane of the robot can be calculated using the difference of the current horizontal neck servo to its parallel position and multiplying it by the angle of a servo step.

$$A = \left| S_{H\uparrow} - S_{H_{Current}} \right| \cdot 1.34^{\circ} \tag{26}$$

The distance of an object in the X and Y directions from the front plane of the robot can be found as follows:

$$X_n = D_n \cdot \sin\ (A_n) \tag{27}$$

$$Y_n = D_n \cdot \cos\ (A_n) \tag{28}$$

After the second X and Y values have been found the change in distance in both the X and Y directions can be calculated:

$$\Delta X = X_1 - X_n \tag{29}$$

$$\Delta Y = Y_1 - Y_n \tag{30}$$

By using the change in distance of the two co-ordinates, the distance of the X and Y intercepts where the ball will pass the robot's path can be calculated

$$D_X = \frac{Y_1}{\Delta Y} \cdot \Delta X \tag{31}$$

$$D_Y = \frac{X_1}{\Delta X} \cdot \Delta Y \tag{32}$$

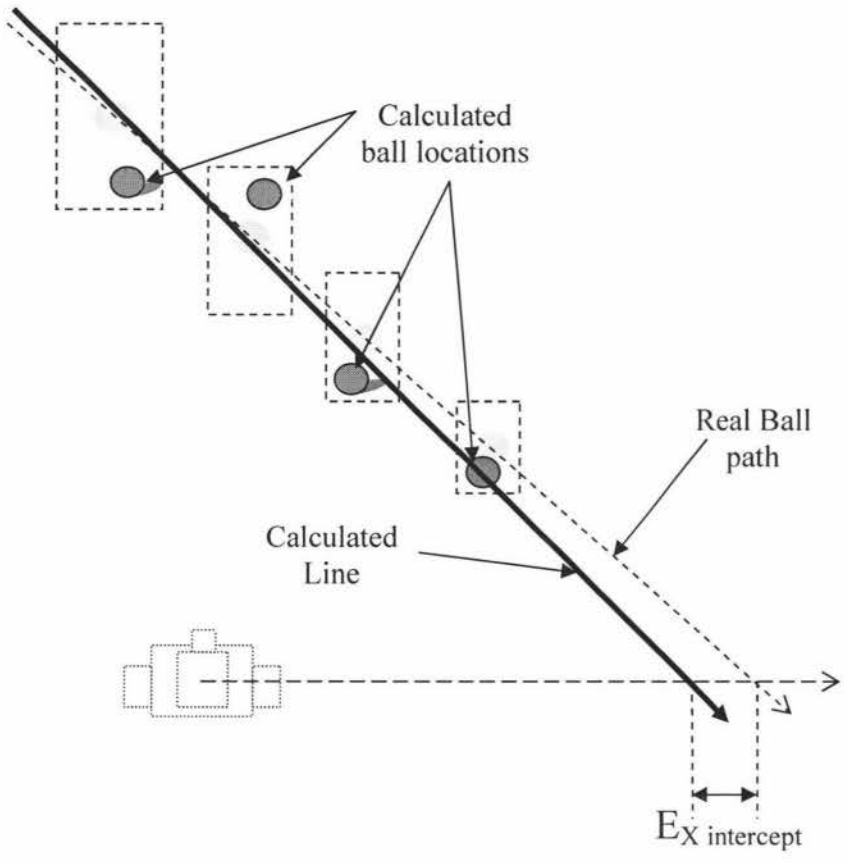


Figure 30: Line of best fit

By obtaining the x and y co-ordinates of the ball from the calculated distance and direction, a scatter plot of ball locations can be formed. After several points are taken a statistical regression on the points could be performed to get a line of best fit. This is demonstrated in figure 30 where the dotted line shows the actual balls path and the bold line shows the line of best fit from the recorded values. Since a series of points must be taken, the robot will need to remain stationary while calculating the required number of ball locations.

The y value of any point on the line can be found using the formula in equation 33, where m is the slope of the line and b is the y intercept. Line of best fit though regression can be calculated as follows:

$$y = m \cdot x + b \tag{33}$$

$$m = \frac{n \cdot \left(\sum x \cdot y\right) - \left(\sum x\right) \cdot \left(\sum y\right)}{n \cdot \left(\sum x^2\right) - \left(\sum x\right)^2} \tag{34}$$

$$b = \frac{\sum y - m \cdot \left(\sum x\right)}{n} \tag{35}$$

$$\sum x^2 = x_1^2 + x_2^2 + \ldots x_n^2 \tag{36}$$

$$\sum x \cdot y = x_1 \cdot y_1 + x_2 \cdot y_2 + \ldots x_n \cdot y_n \tag{37}$$

By re-arranging equation 33 the x intercept can be found using equation 38.

$$x = \frac{-b}{m} \tag{38}$$

Now the slope of the line has been calculated, which is the path of the balls trajectory, and the distance that the ball is going to cross the x and y intercepts, the direction the ball is travelling along the path still needs to be calculated. This can be done by comparing the first and last few ball locations to see if the ball is heading towards or away from the robot and whether it is moving left or right, which in turn will give the direction the ball is travelling along the slope.

As the ball moves along the slope it will pass a point where the ball ceases to be an oncoming ball and starts to travel away from the robot as is drawn in figure 31. At this point the robot needs to stop trying to intercept the ball if it is doing so, and start to head in the direction of the balls travel. The robot needs to check that its horizontal

servo motor is in an expectable range for the ball to be intercepted. If the ball is outside of this range then the ball is judged to be not intercepted and the robot needs to turn in the balls trajectory heading.

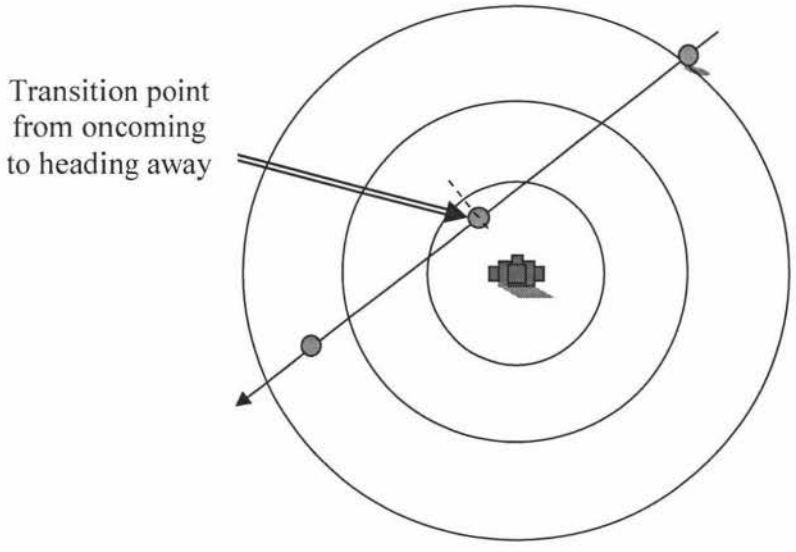


Figure 31: Transition from ball oncoming to travelling away from the robot

For the ball to be tracked several new functions must be constructed to calculate the balls trajectory and to make the appropriate decisions to allow the required motions to be called.

Now that a system has been devised to calculate the balls trajectory effectively a new flow diagram of required checks and tasks the robot needs to perform must be created. The new decision tree must include the original point and follow method as well as the new path planning algorithms using the trajectory of the ball.

The flow of the program will start the same way as the point and follow method with the robot searching for the ball. Once the ball is found the robot will need to check if the ball is stationary. If the ball is stationary then the robot can proceed onto the point

and follow routine which has been explained earlier. If the ball is not stationary then the robot will need to stop and get the balls trajectory.

After the trajectory has been calculated the robot needs to find out if the ball is heading in its direction and if so can the ball be intercepted? If this is true the robot must sidestep in the direction of the ball to intercept it. A side stepping motion was chosen due to the slow movement of the robot which limited the choice of motions the robot could perform to intercept the ball. If the ball could not be intercepted the robot would need to turn in the direction of the balls trajectory and head in that direction in the hope to be the first player to reach the ball. Once either of these actions is taken the robot must return to the start of the cycle.

Although the point and follow routine remains almost the same a additional check would be performed to see if the ball is moving while the robot is walking towards the ball. This would tell the robot to stop and calculated the balls trajectory. This is added for the case where a ball has been reached by the opposition first and has been kicked. This check involves checking the ball has not moved more than is expected in while the robot is in motion.

For the routine displayed in figure 32 most time intervals would be governed by the speed the robot could perform its motions, which would vary due to many external factors, as well as processing time. Along with these time factors, delays were also used between several steps, ranging from 100 to 300 milliseconds to allow for communication latency and the robot's motions to be performed. Along with a several second delay for the kicking motion to allow the robot some time to reposition itself before restarting the process.

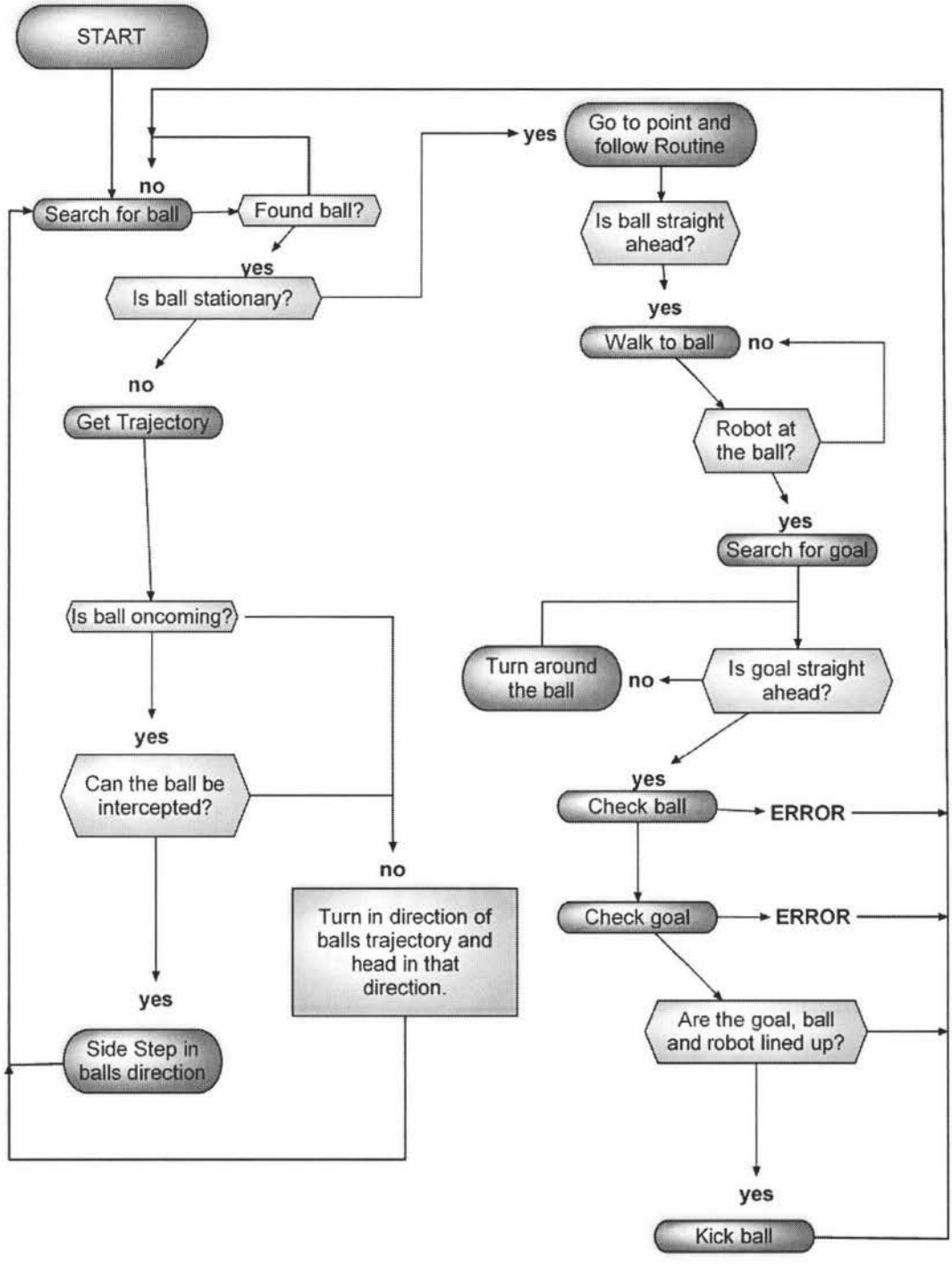


Figure 32: Updated flow diagram of robot A.I.

5.6: Simple Goal Localization

To improve the speed and ability of the robot, an implementation of a simple localisation method could be used. By initially calibrating the robot with the goal headings using the compass, this would mean that the robot would be able to turn to the direction of the goal after reaching the ball without the need to go through the search routine.

The global direction of the compass would be used as a reference. Once the robot had turned around till the compass value is within 20^0 of the calibrated value, the robot could begin to search for the goal.

This would dramatically improve the speed of the point and follow routine, especially in the case where the robot is facing away from the goal.

Chapter 6: Testing & Results

A series of test were performed to setup and test the effectiveness of the point and follow A.I. routine, along with tests for the distance and trajectory calculations which will aid the path planning routine.

6.1: Testing and analysis of the point and follow routine

To test the point and follow implementation all the functional routines need to be tested, this means placing the ball & robot in various locations and positions and running the A.I. program. Since the program functions in an infinite loop, if left long enough the robot would be able to score a goal, no matter what position it or the ball was in. For the test a 70mm diameter ball weighting 170g was used since it was the closest that could be found to the RoboCup specifications. The goal size chosen for the test was 300mm by 300mm, the smaller goal size was used to test and improve the accuracy of the shot taken.

On initialisation of the camera the camera register 18 was set to 32 to turn the auto white balance off. The camera was set to high resolution mode and there servo parameters were set to:

Pan range far = 30	30
Pan range near = 10	10
Pan long step = 5	5
Tilt range far = 30	30
Tilt range near = 12	12
Tilt long step = 3	3

Table 3: Servo Parameters

To reduce noise from foreign objects, the noise filter would be turned up to 3 if tracking the ball and 12 if tracking the goal. This would mean the number of tracked pixels joined together was increased. This increase would mean that random objects of the same colour in the background would have to be large to affect the tracking of the object.

The range of the colours chosen depended on the environments lighting, before each test the CMUcam2 GUI was used to find the colour values that the object in its current environment. These values would then be update into the code.

If the pixel value was less than 1 and the confidence was less than 10 the ball would be classed as not being found and the search routine would kick in. The goal not found values consisted of pixels less than 3, and confidence less than 5. The goal value was chosen to be larger since the goal is much larger object and the higher values would reduce the effect of noise.

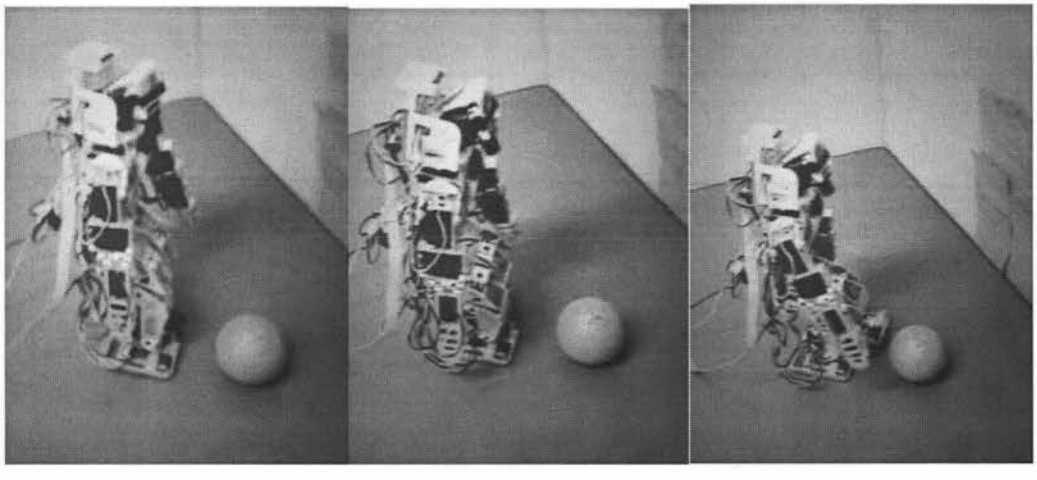


Figure 33: Robot Approaching the ball

As the robot approaches the ball the robot needs to know when the ball is at its feet. The vertical servo value was chosen for this task. If the vertical servo value received

from the camera was less than 107 units, the pixel value number was greater than 40, and the confidence value was above 20, the robot would stop walking. While approaching the ball if the horizontal servo would move out of the range of 118 to 142 the robot would turn in the appropriate direction (128 is centred).

After the ball is reached and the search function had found the goal the robot starts to position around the ball. The value range of the horizontal servo was used to define if the goal was in position, if the goal was between the value of 120 and 140 the goal was in position.

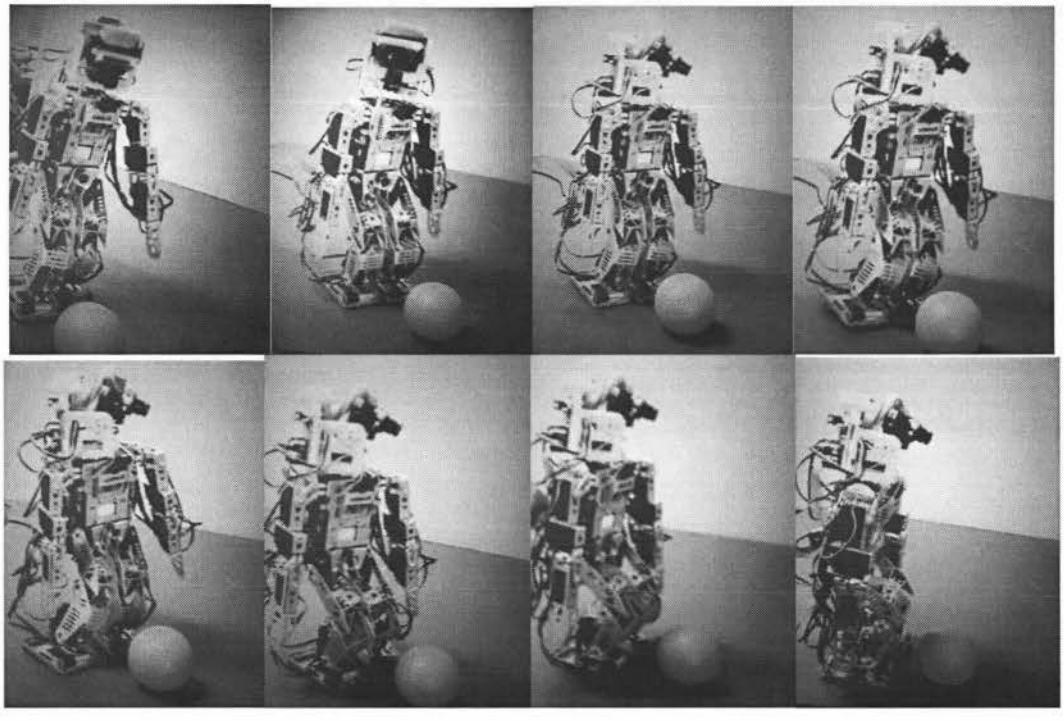


Figure 34: Searching for goal and positioning around ball

After the robot has positioned around the ball to face the goal the robot was then required to position the ball in front of its kicking foot using its strafing motion. Again a horizontal servo range is used. If the ball is within 138 to 144 the ball is in position to kick.

Figure 35: Final positioning for shot and kick at goal

After the ball is in position the kicking motion was called with the robot scoring a goal with 80% accuracy over a 30 run test with the chosen goal size at a maximum range of 0.8m. After the kick a delay of 5 seconds was chosen so the robot would not be interrupted while kicking.

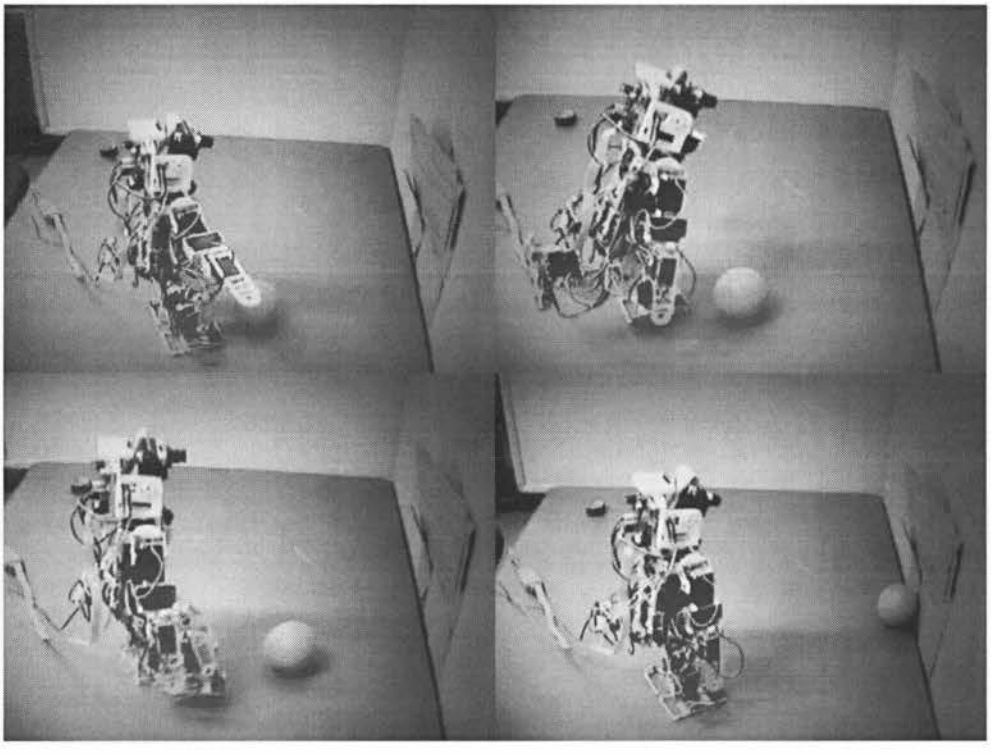


Figure 36: Robot shooting goal

6.2: Analysis of distance and direction calculations

For the distance and direction calculations there are two cases of interest. They are the distance from the object when the robot is stationary and when in motion. These cases should produce different results and the variables chosen for these values will be case specific.

6.2.1: Stationary Robot

While the robot is stationary there will be the least amount of noise in the system. This means that the majority of tweaking of the formulas will need to be done through the stationary testing to make them fit this system.

From the initial testing of the distance values, the distance error increased by a growing rate as the objects distance got further from the robot. Since the value of FixZ in equation 39 gets smaller as the distance decreases a variation of this was chosen to be used.

$$Fix_{E} = \frac{O_{H}}{2} \cdot \cos\left(A_{V}\right) \tag{39}$$

The vertical servo's 90^{0} value set to 162, along with the object height set to 70mm, and object size set to 50, gave the most accurate results for the 70mm test ball. The error present in the system was then dramatically reduced by multiplying the distance value by Fix_{E}, then multiplying the result by 5 for the horizontal distance and 7 for the vertical distance. This left the system approximately 60mm out for all distances, this was thought to be due to mechanical loses not included in the mathematic equations. By adding 60 to all distances this error was removed.

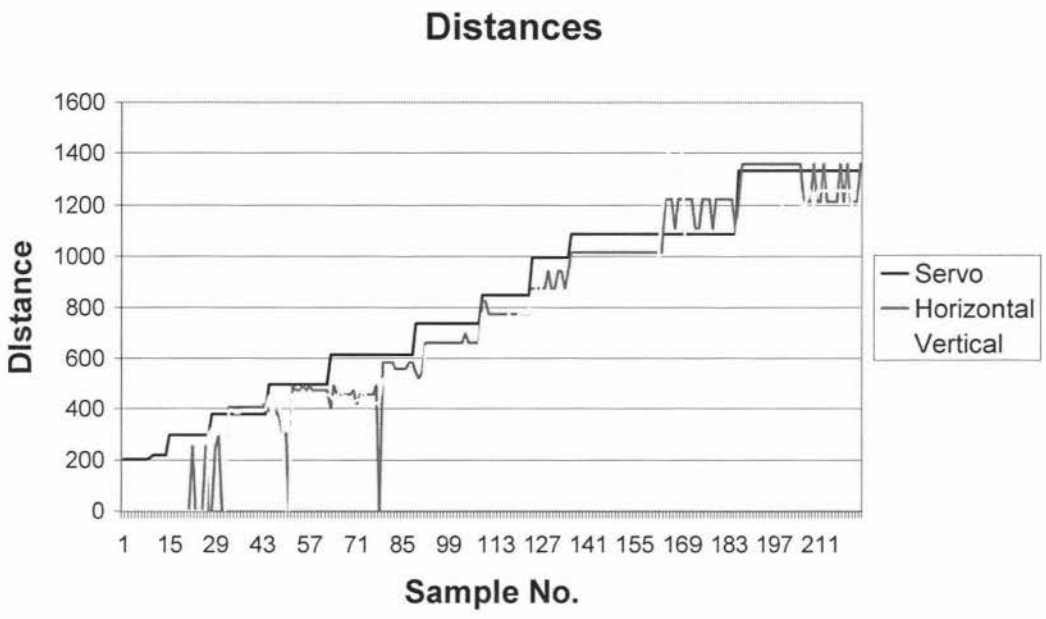


Figure 37: Comparison of distance calculations while robot stationary

Figure 37 is a graph of distances recorded from the robot in steps of 100mm starting at 200mm from the robot to 1300mm. this illustrates the stableness of the servo value when the robot is stationary and the closeness of the horizontal and vertical values. It also shows the errors of each value, as seen by the spikes in the graph at lower distances from the 'image size' distances which have huge errors when the distance is less than 400mm from the robot. Alternatively the servo distance is more accurate which can be seen in the graph. This is also the opposite for the greater distances where the servo value starts to hold its current value even when the distance increases.

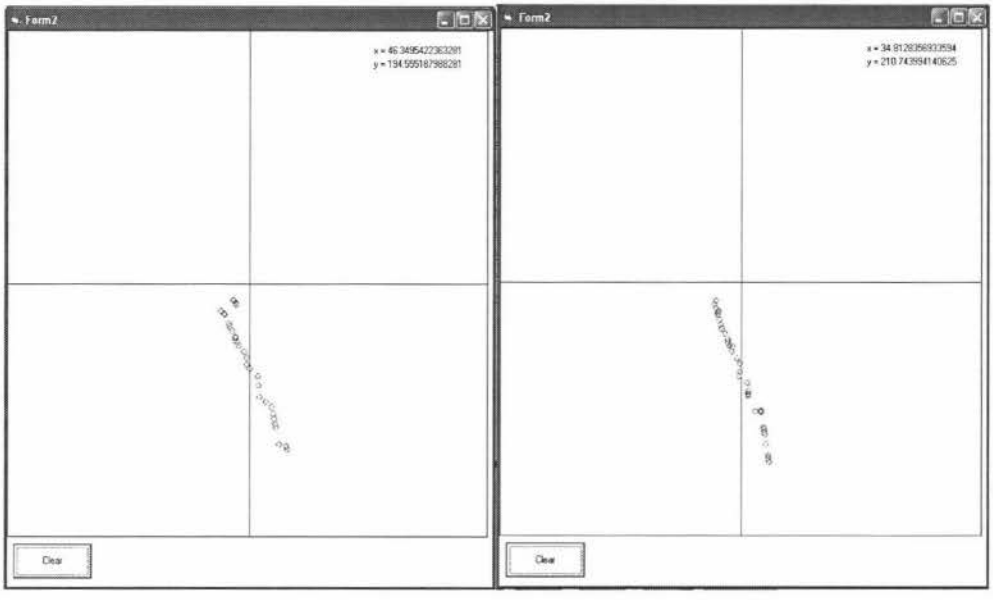


Figure 38: (a) Real-time plot of average distance,
(b) Real-time plot of corrected average distance

From this it is devised that the servo distance should be used in place of the image size distances for the calculation of objects within 400mm from the robot. By using only the servo distance value within this range the incorrect values calculated from the image size would be removed along with any spiking caused by a distance value of zero being erroneously calculated. From the second picture in figure 38 it can be seen that after the servo value is the only value used within 400mm in image (b), the large jumping error in the system has been removed. As explained earlier the distance calculated from the image size should be more heavily weighted for objects at greater distances, therefore removing the larger variation cause by the servo distance at this range.

6.2.2: Moving robot

The next aspect of the distance and direction calculations is to look at when the robot is in motion, which presents more noise in the system. The stance of the robot will also be different since it leans forward and lowers its centre of gravity while in the

walking motion. This will not only affect the robots height but its 90^0 vertical servo position.

To test change of stance the robot was held at the starting point of its continuous walking motion. At this point it was found that the robot's height lowered by 15mm, and its 90^0 servo position increased by 6 due to its new tilted stance. Therefore a new check was added to the code to see if the robot was in its home position or walking and the distance code was adjusted to the right values.

To test the distance while the robot is in motion, the robot will perform a walking motion. The ball will be place 1000mm (1m) away from the robot and the robot will walk straight forward to the ball.

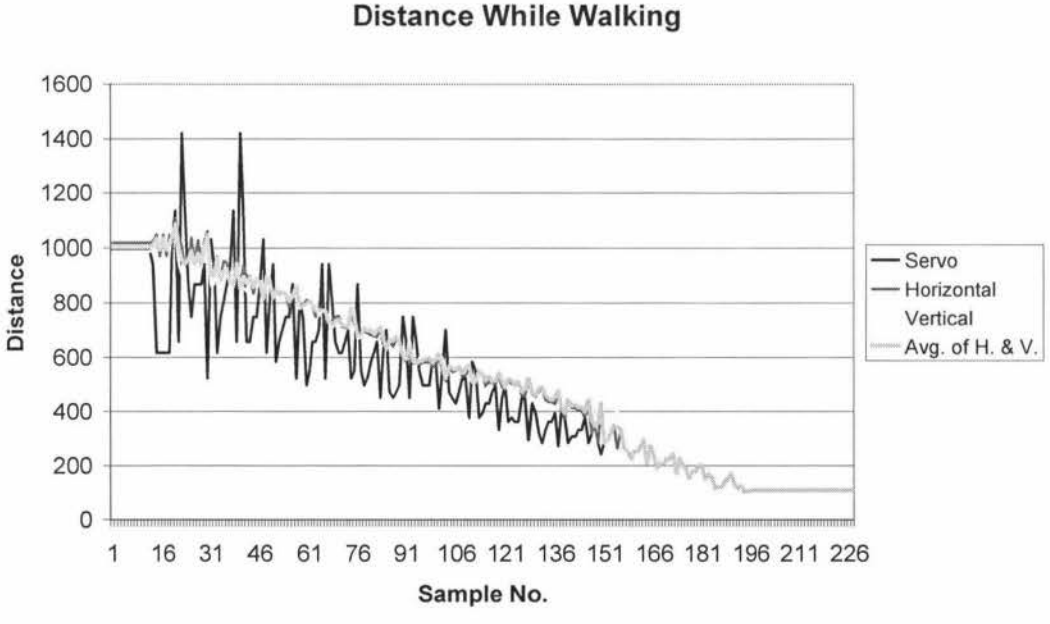


Figure 39: Comparison of distance calculations while robot walking

From the graph in figure 39 it can be seen that while the robot is in motion, the servo distance varies greatly when the ball distance is large. But as the distance gets

smaller, the error lessens considerably. The error in the distances calculated from the image sizes stay rather constant all the way through but start to get larger as the object gets closer. This is then compensated for when the servo value replaces the image size values at the 400mm mark. The last line is the average of the two 'image size' distances. As displayed by this line, the average error is approximately $^{+}/_{-}$ 100mm; therefore if a check for if the ball has moved more than $^{+}/_{-}$150mm would mean that the robot could decide if the ball is not stationary while approaching and act if it is moving.

As well as checking if the ball distance has changed the direction of the ball while walking also has to be monitored, since the distance may not change the required amount if the ball is moving along an angled path to the robot. To test the direction the same test was performed as the distance. The real-time plot of the balls position shows the cluster of points of the ball as the robot approaches and the error present in the direction of the ball.

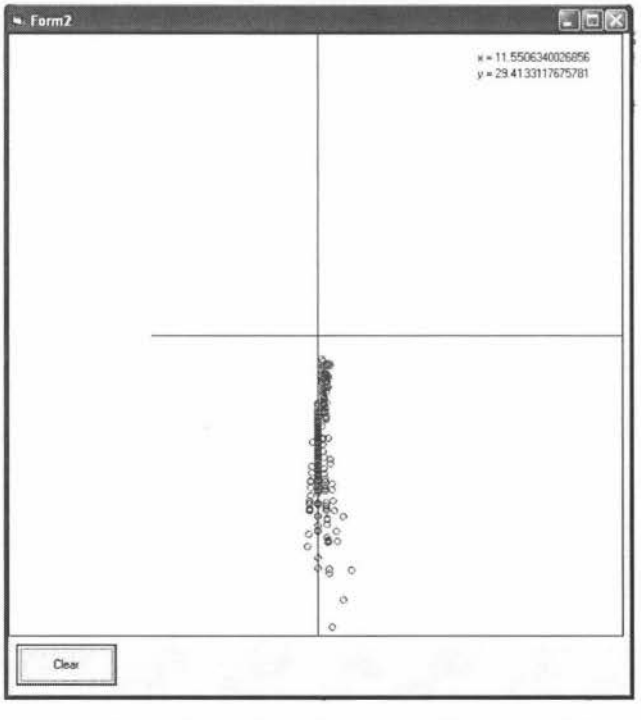


Figure 40: Real-time plot of average distance while walking

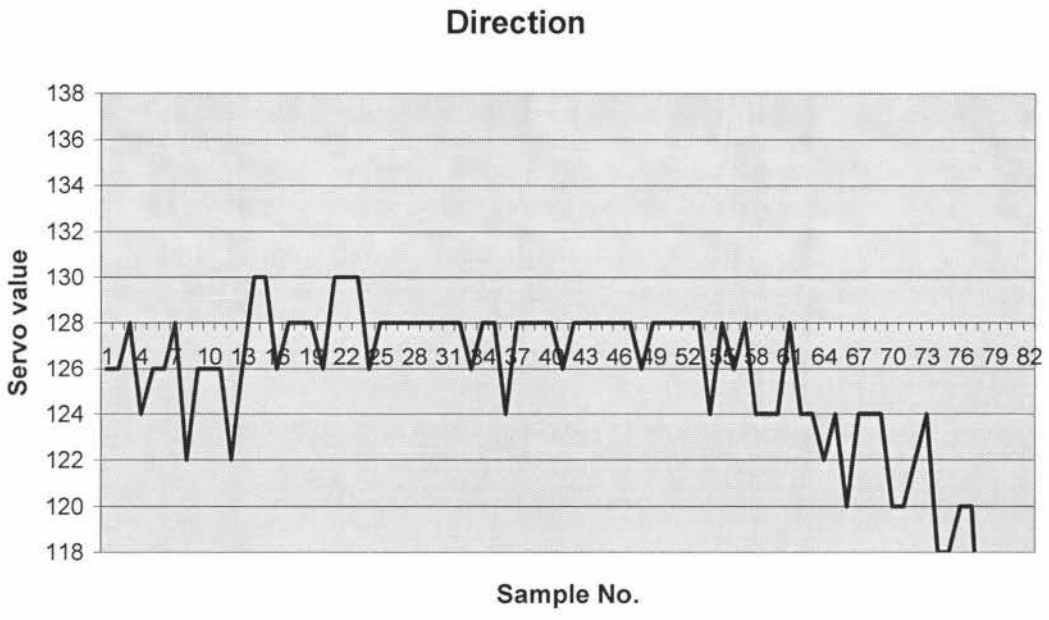


Figure 41: Graph of direction of head while robot walking

For this graph the horizontal servo value over time can be seen. The centre point of the horizontal axis has a value of 128. This graph shows for the sample taken, the direction of the ball was approximately at the value 125 at the start but moved to approximately 127 mid ways through travel. As the robot reaches the ball, it ends up slightly to the robot's left. This is also illustrated by the real time plot of the balls location in figure 40. This is due to the walking motion not producing a perfectly straight walking pattern but slowly drifting to the right instead.

The largest jump made between samples was 4 units, which is slightly over 5^0. by adding a an extra buffer of 2 servo value for any possible extra error that could occur that would mean if the value changed by more than 6 units the ball would be considered to be moving.

The distance from the robot when the graph starts to drift is when the ball is less than 400mm away from the robot. If the ball is within this amount then direction can be ignored as an indicator because it starts to move more rapid due to the balls image taking up more of the overall image and making the head bounce.

6.3: Examination of the estimated ball trajectory

When looking at trajectory, there are 2 main factors of interest. The first is when the ball is travelling straight across the X or the Y axis of the robot in either direction. The other trajectories of interest are when the ball is travelling on an angle to the robot where its path will cross the x axis, close to the robot's position and could be intercepted, along with a distance from the robot where it would not be able to be intercepted.

Too determine the error in the system a series of runs need to be performed across a known path. From a number of these runs the error in the system can be calculated. Limits were put in place where the ball had to have moved more than 50mm in the X and Y direction before it was deemed to be no longer stationary. The number of points that gave the best trajectory results in testing was 4 points, this allowed enough points to be taken to reduce the error in the system as well as few enough points so that the robot was able to have time to react.

One problem that was occurring was the robot could not get results fast enough when calculating the trajectory of the ball. By calling the "Distance" and "GetPacket" subroutines inside the "Direct" subroutine the points could be gained faster since the program control was not given back to the main "RunTrack" subroutine until the trajectory was calculated.

Trajectory along the axis

To test the trajectory calculations when the ball is moving across the axis, a series of runs in both directions are performed. The initial test was across the Y axis, with the ball was run straight away and towards the robot. This was done by making a runway for the ball to follow while in motion. The robot was then told to record a trajectory while the ball was rolled.

From the series of runs recorded the balls direction along the slope was always correct. While the ball was oncoming the x and y intercepts were always zero and the slopes range was from -87^0 to 90^0 which indicate that if the balls slope is between 87 and 90 degrees or -87 to -90 degrees the ball is heading on a straight path. While the ball was travelling away there were a few errors which place the ball travelling on an angle from the Y axis. The largest error placed the trajectory going from -235mm on the X axis and 711 on the Y axis. Both errors that were present were due to an erroneous reading from the last X co-ordinate recorded as the ball was at its furthest distance. Since this error was only present when the ball was heading away from the robot this lessens the concern about the error since the robot would go to its point and follow routine if the ball was heading away.

The second test is along the X axis with the ball heading in both directions. For this experiment the robot is set up approximately 400mm away from the balls path. A series of runs are performed in the same way as the Y axis. First heading left to right and then right to left. Through all the runs the flag for the direction travelled gives the correct value.

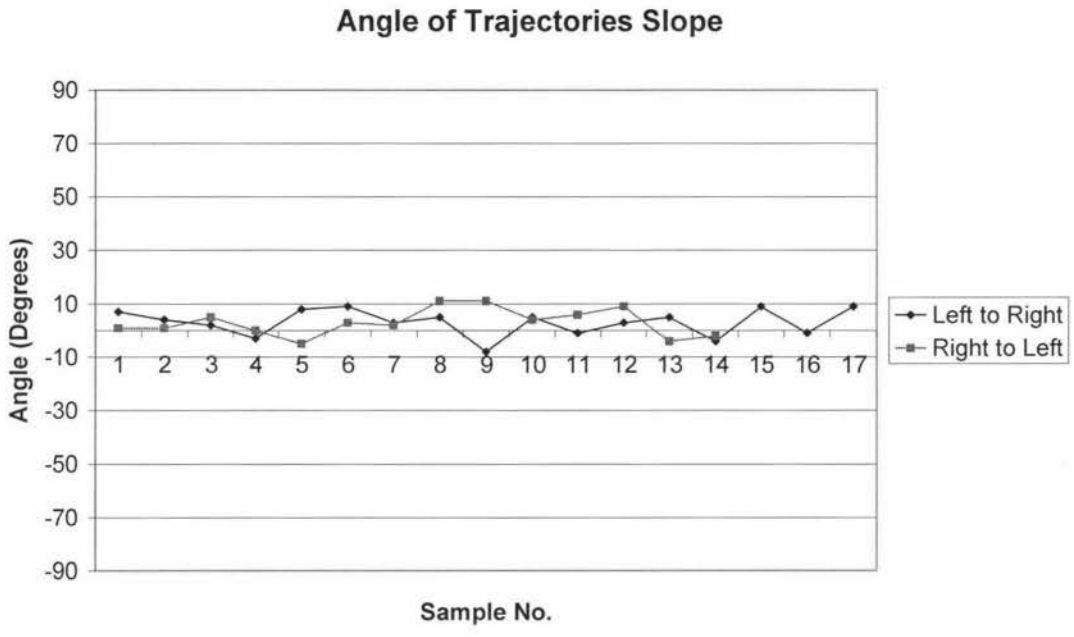


Figure 42: Slope of balls trajectory while ball travelling along y axis

From the graph in figure 42 of the angle of the slope that the ball travels, the slope remains within $^{+}/_{-}10^{0}$ of the actual slope, with the maximum slop being 11^{0} and the minimum slope equalling -9^{0}.

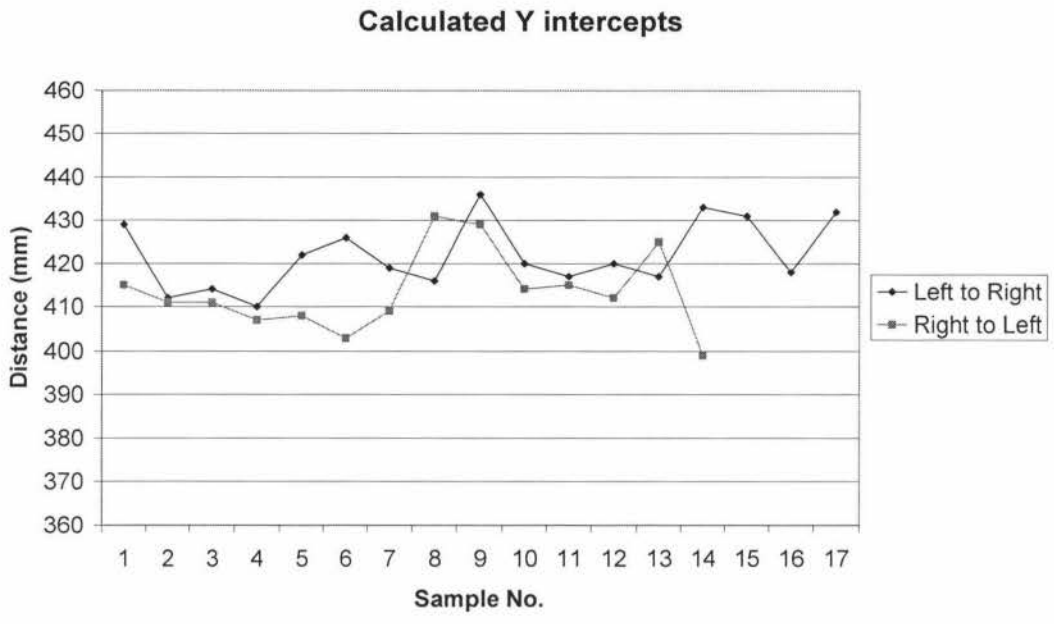


Figure 43: Graph of y intercepts while ball travelling along the x axis

The graph in figure 43 shows the calculated Y intercept of the ball. The points average around 422mm while heading left to right and 414mm from right to left. The points calculated stay within $^{+}/_{-}$16mm of that average. The x intercepts calculated are all greater than 2500mm in either direction, therefore if the x intercept is greater than that value the ball can be considered to be travelling straight across the front of the robot along the x axis. This can also be defined as true if the slope is within 11^{0} of zero.

Trajectories crossing the axes

As stated earlier one of the two trajectories of interest, was when the ball is crossing both axes while it is oncoming. The 2 trajectories to be used were; when the ball is crossing the x axis where it can be intercepted and the second will be slightly too far away for the robot to be able to intercept the ball. These two trajectories will be chosen since they will be the most influential instances on the robot's A.I algorithm. The values chosen for the first test are 500x and 380y for the 2 intercepts and the

second set of values are 200x and 170y. These values where chosen since the ball goes across that robot at approximately 39^0 from its x axis with the angle from the robot staying the same in both tests.

From the graph of the calculated slopes in figure 44 it can be seen that the slope has and accuracy of $^+/_-10^0$ which is the same as found when the ball was travelling along its axis. This graph also shows that the average slope, which was between 34 and 36 degrees is approximately equal to the actual slope, which is between 38 and 40 degrees.

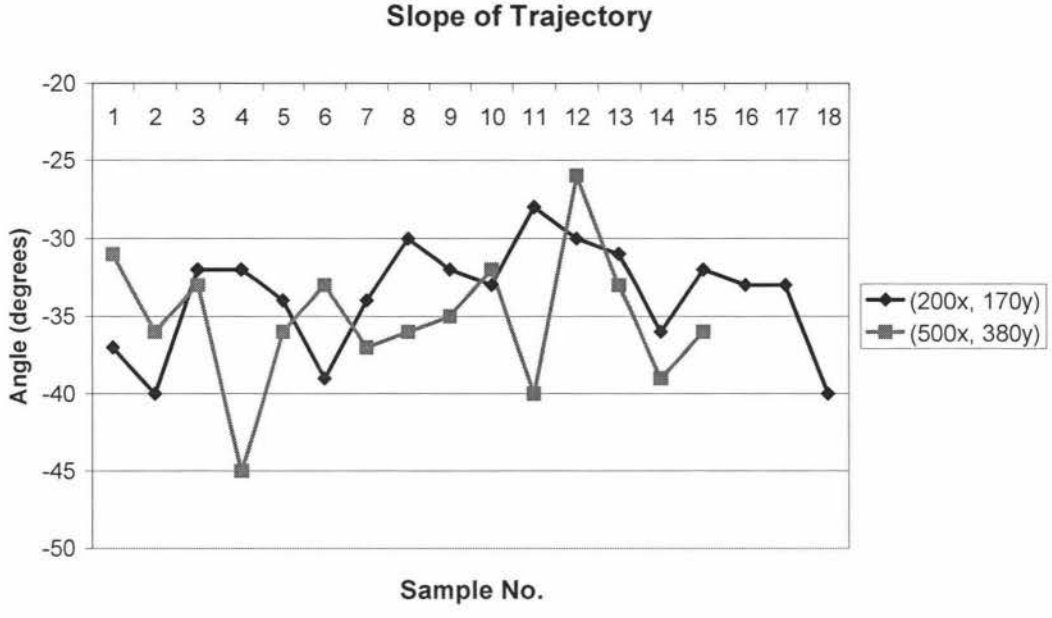


Figure 44: Slope of ball when travelling across the 2 axes

The x intercept results show that the further away from the robot the x intercept is located, the greater error present. The ratio of the error in the x axis compared to the added distance is close to 1:1, where the error in the y intercept remains very much the same. Although the larger error is present at greater distances, the average of the results over a series of samples is extremely close to the measured values, proving that the calculation has been performed is correct.

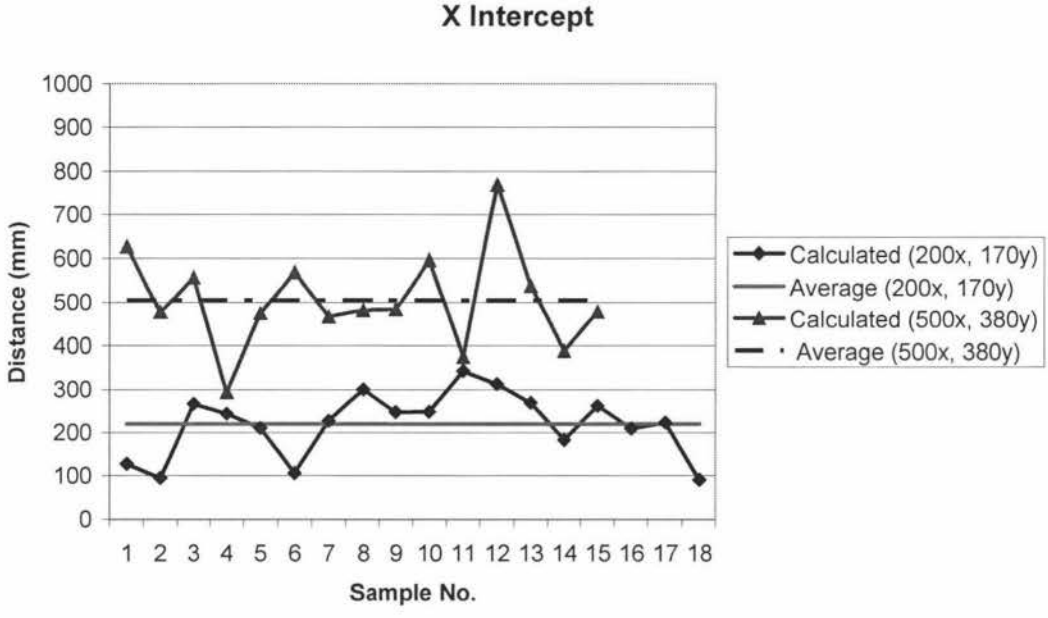


Figure 45: Graph of x intercepts of balls trajectories

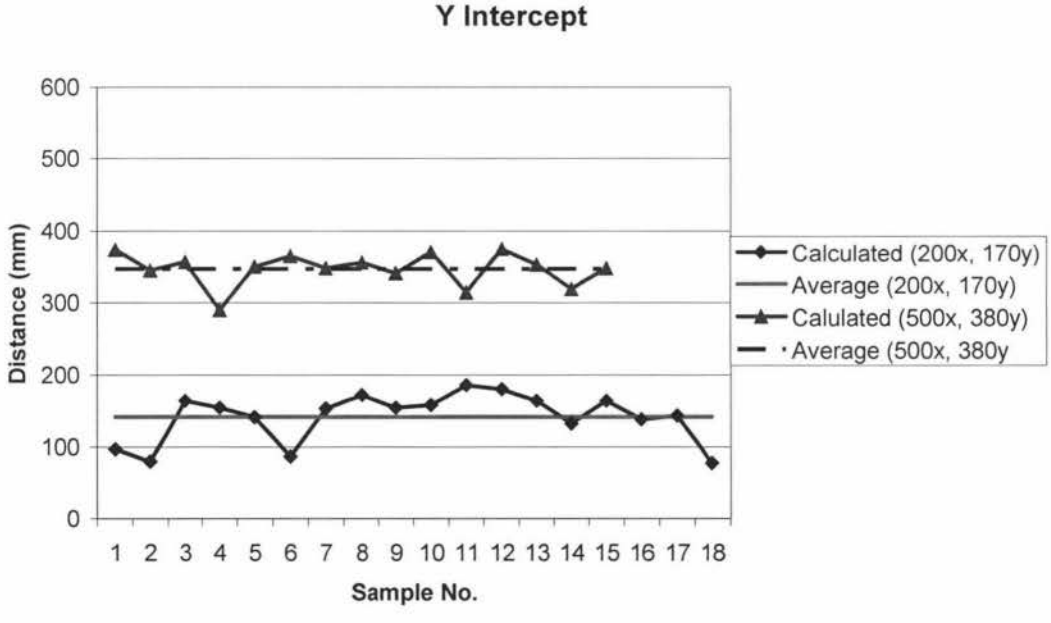


Figure 46: Graph of y intercepts of the balls trajectories

From these results, any value below 300mm in the x direction can be considered to be able to be intercepted through side stepping in the direction of the ball and values below 300mm in the y direction can be regard as able to be intercepted by the robot through its walking motion.

6.4: Implementation of the Path Planning Algorithms

To test the effectiveness of the ball path planning algorithms in relation to A.I. of the robot, the code for the distance and direction calculations were combined with the point and follow routine. The main testing would be to observe the effectiveness of the robot to act on the trajectory data and which values would best suit the routine.

The main function to be under scrutiny would be the 'CorrectedCourse' function. The decision was made to keep the robot from walking by changing the walking motion command to 'o' which causes the robot to not produce any motion. This would imitate the robot already having stopped to get the results of the 'corrected course' functionally.

The compass readings had to be removed from this function due to the compass periodical failing to produce readings resulting in the robot continuously turning, or the program crashing. This would mean if the robot needed to turn in the direction of the balls trajectory a form of control would be needed to restrict the number of turning motions called. It was decided that the slope of the trajectory would be an effective way to control the number of turning motions. Through testing this slope proved to be too large. It was found that by dividing the slope by 20 the number of turn motions called would be the appropriate amount.

For the robot to distinguish if the ball was in motion or stationary, the following values chosen were:

If Robot in Motion

Change in value	*Value*
Average Distance	150mm
Horizontal Direction	10 servo units

If Robot Stationary

Change in Value	*Value*
X Distance	70mm
Y Distance	80mm

Table 4: Ball Transition values

If the difference of the current and previous values of above was greater than these values the trajectory of the ball would be calculated. If the first and second to last values of the trajectory calculation were less than 80mm in the X direction and 90mm in the Y direction then the ball would be considered to be stationary and the 'corrected course' would not need to be performed. The values were increased from previous trajectory tests due to the robot's increased range in motions producing false results.

Since the slow speed of the robot if the X intercept was less 300mm and less than that of Y the robot would sidestep to intercept. If the Y intercept was the smallest and under 300 then the robot would use the walk function to intercept as stated earlier. In the case the robot was too slow the horizontal servo was checked to make sure that it remained between 110 and 150, if this range was broken then the robot would go to the turn function.

For the robot to try and intercept the ball, the ball had to travel at a slow speed. If the speed of the ball increases the robot could not try to intercept due to its limited range and speed of motions; the limitation through communication; plus the time required for the processing of the distance and trajectory calculations. Therefore there were minimal trials where the ball was travelling fast enough to be classed as moving, and slow enough so the robot could call the intercept routine, and no trials where an interception was achieved. Due to these limitations the result was the passing of the intercept task to the robot's turn function.

The turn function had to be limited to functioning only if the ball was oncoming due to the point and follow routine proving more effective if the ball was travelling away. This meant that only slopes of above 45^0 or below -45^0 would be used. As stated earlier the ratio of slope/20 was used for the number of turns which would result in the robot facing in the general direction of the ball and begin the tracking of the ball without the need to call the search function.

To check that the robot was turning in the right direction, an exit clause was added to each turn routine. If the horizontal servo, while turning and still tracking the ball passed a limit it would stop and return to the point and follow routine, which would automatically tell the robot to turn in the correct direction. The values chosen for this were 110 if the robot was turning left and 150 if the robot was turning right.

After the 'CorrectedCourse' function was complete the program would pass back control to the main 'RunTrack' subroutine. Since the robot had just calculated for a moving ball the calculation did not need to be performed again for a few runs, more so for the case where the ball was heading away, since it would cause the robot to keep on pausing while the ball was in motion. A halt of trajectory calculations for the next 10 runs of the main code proved long enough while testing.

Through this implementation it was found that there was no case where the ball could be intercepted with the limitations present in this system. However through a controlled test where the ball was stopped after the robot had calculated its trajectory, the robot would still define the ball to be able to be intercepted and would proceed to move to the appropriate location to perform the ball interception. Though the interception of the ball proved to be ineffective, the turn function proved an invaluable addition to the A.I. code since if the ball was to pass the robot with only the point and follow routine, the robot would definitely lose sight of the ball and have to start a long search process for the ball.

Chapter 7: Conclusion & Future Work

Humanoid robots have a promising future, whether for entertainment or industrial purposes due to there ability to move around in a human environment, mimic human movement and by being aesthetically pleasing. The RoboCup is a competition designed to further the development of robotics, with the humanoid league being the forefront of the competition.

For this thesis a design for the robot platform to compete in the RoboCup competition was developed using an off the shelf robot platform, camera board, compass, and processors. Colour vision was used as the primary sensor for all the developed tools and algorithms. Through analysis of the environment and the requirements of the RoboCup, monoscopic vision was chosen for this platform. A head and neck mechanism were developed for the robot to extend its vision for its focused camera setup.

Along with the platform, various tools were created to allow the robot to function autonomously, effectively and efficiently in its environment. The first of the developed tools was a simplistic A.I. using a 'point and follow' approach to the robot control, which enabled the robot with enough cognitive ability to complete the basic functionality of a striker of the ball. This included reaching the ball and kicking the it in the appropriate direction, which in the testing conditions proved to be 80% effective from the first kick.

Mathematical models were used to show the comparison of stereoscopic verses monoscopic vision and detailing the selection of monoscopic vision due to the known environment of the RoboCup competition. A monoscopic depth perception mathematical model and algorithm were then developed. Through analysis of the

distance implementation on the robot platform, the depth perception was effective up to 1300mm with the current camera resolution. With the use of the three distance calculations a stable average distance could be found, both for when the robot remained stationary and while in motion.

An effective trajectory calculation model was developed and through testing, provided $^{+}/_{-}10^{0}$ accuracy. This ball trajectory algorithm was designed to allow the robot to calculate a moving balls trajectory and react according to its motion path. Through testing it was found that due to the restricted motion of the robot and limited speed of the current algorithm, the robot was unable to intercept the ball but could turn in the balls direction of motion. With streamlining of the code, an increased camera resolution, and the implementation of dynamic motion the speed of the robots abilities and reactions could be improved and these tool prove a very effective addition to the robots A.I.

Future Work

Future vision algorithms to implement could include a background mask [Ulrich et.al.] which will require the robot to be calibrated to determine the difference between the floor and obstacles. This could be done by a binary image being formed to define the floor as black and everything else as there original colouring. This will allow the robot to navigate around unknown areas without crashing into obstacles in its path. A problem that has occurred with implementations of this system is that the robot will have to be calibrated for every different surface it walks on. For the application of soccer the robot is told to ignore white so it does not think that the markings on the field are obstacles. With this mask being active the robot will distinguish everything that is off the field as an obstacle as well as other robots in its path. This will prove very useful for the robot distinguishing the outer boundaries of the field.

Fast and slow systems, as commonly used by the armed forces, could be utilised. The implementation will include 2 vision systems. The first is the slow system which will

require a wide field of view and will act like the robot's peripheral vision. A monoscopic vision system utilising a wide angle or omni directional lens will supply the wider field of view required. Since the slow system has a wide view it can pick up all the objects surrounding the robot and feedback any objects of importance to the fast system. The fast system is a more focused system that will have a more narrow view angle and focus on only a single item at a time. By focusing on a single item at a time the response time and accuracy of the robot for a required task will be improved since the amount of data required for processing will be decreased, therefore more processing can be used for depth perception calculations.

The addition of dynamic walking and balancing will be invaluable to the improvement of not only the tools created but all future development of the robot. It will allow the robot to have a more complete and free range of motions that will allow the more functionality and adaptability in any environment. The ability to perform complex and precise movements at a rapid speed will allow more accurate and efficient A.I. algorithms and tools. This along will improvements to the mechanics and design of the robot will help create an ideal robot platform.

Glossary

- ABS - Acrylonitrile Butadiene styrene
- A.I. - Artificial Intelligence
- Autonomous/Autonomy - the ability to operate within an environment and be capable of independent decision and action in pursuit of an objective.
- Bipedal - Moves on two legs
- Bit (b) - Binary digit (either 1 or 0)
- Bite (B) - Eight bits
- bps - Baud Rate (bits per second)
- CAD - Computer Aided Design
- CAN - Serial bus (Controller Area Network)
- CCD - Charge-Coupled Device
- CMOS - Complementary Metal Oxide Semiconductor
- CF - Compact Flash (Flash storage device)
- COM - Serial port interface
- CPU - Central Processing Unit
- CRT - Cathode Ray Tube (display)
- CSV - Comma-Separated Values
- DAQ - Data Acquisition
- DC - Direct current
- DOF - Degrees of Freedom
- DOS - Disk Operating System
- EEPROM - Electrically Erasable Programmable Read-Only Memory
- FDM - Fused Deposition Modelling
- fps - Frames per second
- GPS - Global Positioning System
- GUI - Graphical User Interface

- I^2C - Multi-master serial computer bus (Inter-Integrated Circuit)
- I/O - Input/Output
- IDE - Integrated Drive Electronics
- MIPS - Million Instructions per Second
- NTSC - Analogue television system (National Television Standards Committee)
- OS - Operating System
- PAL - Colour encoding system (Phase Alternating Line)
- PC - Personal Computer
- PDA - Portable Digital Assistant
- PWM - Pulse-Width Modulation
- RAM - Random Access Memory
- RGB - A colour space (Red, Green, Blue)
- RISC - Reduced Instruction Set Computer
- RLE - Run Length Encoding
- ROM - Read Only Memory
- SDRAM - Synchronous Dynamic Random Access Memory
- SRAM - Static Random Access Memory
- TTL - Transistor-transistor Logic
- UART - Universal Asynchronous Receiver/Transmitter
- USB - Universal Serial Bus
- YUV - A colour space ()
- ZMP - Zero Moment Point

References

[Ambrose] Ambrose R., NASA's Johnson S pace Centre Robonaut Web Page: http://robonaut.jsc.nasa.gov/

[Arbter et.al.] Arbter K., Hirzinger G., Langwald J., Wei G.Q., Wunsch P., *Robust Vision for Vision-Based Control of Motion,* M Vincze, GD Hager Wiley-IEEE Press. Chapter 9, 1999.

[Baltes et.al.] Baltes J., McCann S., Anderson J., Humanoid Robots: Abarenbou and DaoDan, http://www.informatik.uni-freiburg.de/~rc06hl/qualification/AkDong_TDP.pdf, 2006.

[Behnke] Behnke S., *RoboCup Soccer Humanoid League Rules and Setup for the 2006 competition in Bremen, Germany*, http://www.humanoidsoccer.org/rules.html, June 2006.

[Behnke et.al 1] Behnke S., Muller J., Schreiber M., "*Using Handheld Computers to Control Humanoid Robots*", Proceedings of 1st International Conference on Dextrous Autonomous Robots and Humanoids. Freiburg, Germany, 2004.

[Behnke et.al. 2] Behnke S., Langner T., Muller J., Neub H., Schreiber M., *NimbRo RS: A Low-Cost Autonomous Humanoid Robot for Multi-Agent Research.* Proceedings of Workshop on Methods and Technology. Freiburg, Germany, 2004.

[Behnke et.al. 3] Behnke S., Schreiber M., Bennewitz M., Stuckler J., Strasdat H., Schwenk J., "*Designing a Team of Soccer-Playing Humanoid Robots*", http://www.informatik.uni - freiburg.de/~nimbro/papers/ROBOTIK06_Behnke.pdf, 2006.

[Behnke et.al. 4] Behnke S., Muller J., Schreiber M., *Playing Soccer with RoboSapien*, Lecture Notes in Artificial Intelligence, Springer, 2006.

[Behnke et.al. 5] Behnke S., Muller J., Schreiber M., *Toni: A Soccer Playing Humanoid Robot*, Lecture Notes in Artificial Intelligence, Springer, 2006.

[Bekey] Bekey G.A., "*Autonomous Robots: From Biological Inspiration to Implementation and Control*", The MIT Press, MA, USA, 2005.

[Bluethmann et.al.] Bluethmann W., Ambrose R., Diftler M., Askew S., Huber E., Goza M., Rehnmark F., Lovchic C., Magruder D., "*Robonaut: A Robot Designed to Work with Humans in Space*", Springer Autonomous Robots, Vol. 14, pp 179-197, 2003.

[Bruce et.al.] Bruce J., Balch T., Veloso M., "*Fast and Inexpensive Color Image Segmentation for Interactive Robots*", Proceedings of IROS 2000, 2000.

[Burkhard et.al.] Burkhard H.D., Dominique D., Fujita M., Lima P., Murphy R., Rojas R., "*The Road to RoboCup 2050*", Robotics & Automation Magazine, IEEE, June 2002.

[CMPS03] CMPS03 - Compass Module website, http://www.robot-electronics.co.uk/htm/cmps3tech.htm, visited on 13/3/2007.

[Cornall et.al.] Cornall T., Egan G., Calculating Attitude from Horizon Vision, Proceedings of the First Australasian Unmanned Air Vehicles Conference, 2006.

[Demetriou] Demetriou G.A., *A Survey of Sensors for Localization of Unmanned Ground Vehicles (UGVs)*, http://ww1.ucmss .com/books/LFS/CSREA2006/ICA8122.pdf, 2006.

[Difter et.al.] Diftler M.A., Ambrose R.O., Goza S.M., Tyree K.S., Huber E.L., "*Robonaut Mobile Autonomy: Initial Experiments*", Proceedings of the IEEE International Conference on Robotics and Automation, Barcelona, Spain, April 2005.

[evosapien.com] Robosapien Website, "Home Page", http://www.evosapien.com, visited on 20/7/2006.

[Florczyk] Florczyk S., "*Robot Vision: Video-based Indoor Exploration with Autonomous and Mobile Robots*", Weinheim, John Wiley & Sons, 2005.

[Fujita et.al.] Fujita M., Kuroki Y., Ishida T., Doi T.T., *A Small Humanoid Robot SDR-4X for Entertainment Applications*, Proceedings of the IEEE/ASME International Conference on Advanced Intelligent Mechatronics, AIM, 2003.

[Futurlec.com] Futurlec website, http://www.futurlec.com /DevelopmentBoards.shtml, visited on 15/3/2007.

[Hirai] Hirai K., Hirose M., Haikawa Y., Takenaka T., "*The Development of Honda Humanoid Robot*", Proceedings of the IEEE International Conference on Robotics & Automation, Leuven, Belgium, May 1998.

[Humphries et.al.] Humphries M., Radev P., Shirvaikar M., *A realtime vehicle tracking system*, Proceedings of the Thirty-Seventh Southeastern Symposium on System Theory, SSST, March 2005.

[IBee] Pirkus robot website, http://www.robot-labs.jp/top.html, visited on 14/3/2006

[Inoue et.al.] Inoue H., Tachi S., Nakamura Y., Hirai K., Ohyu N., Hirai S.H., Tanie K., Yokoi K., Hirukawa H., *Overview of Humanoid Robotics Project of METI*, Proceedings of the 32nd ISR(International Symposium on Robotics), April 2001.

[Ishida et.al.] Ishida T., Kuroki Y., Yamaguchi J., Fujita M., Doi T.T., *Motion Entertainment by a Small Humanoid Robot Based on OPEN-R*, Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Maui, Hawaii, USA, October 2001.

[Kitano et.al.] Kitano H., Asada M., "*RoboCup Humanoid Challenge: That's One Small Step for A Robot, One Giant Leap for Mankind*", Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Victoria, B.C., Canada, October 1998.

[Kondo-Robot] Kondo Kagaku Co., Ltd. KHR-1."Home page" http://www.kondo-robot.com, visited on 14/3/2006.

[Lepetit] Lepetit V., Fua P., Monocular Model-Based 3D tracking of Rigid objects, Foundations and Trends in Computer Graphics and Vision Vol. 1, No. 1, 2005

[McCann et.al.] McCann S., Baltes J., *Abarenbou - A Small Vision-Based Humanoid Robotic Research Platform*, Department of Computer Science, University of Manitoba, Winnipeg, Canada, 2004.

[Menegatti et.al.] E. Menegatti E., Gatto G., Pagello E., Minato T., Ishiguro H., *DISTRIBUTED VISION SYSTEM FOR ROBOT LOCALISATION IN INDOOR ENVIRONMENT*, The University of Padua, Italy, 2005.

[Mike] Mike L.C.M., *Automatic Targeting System for Unmanned Air Vehicle Operations*, National Uniersity of Singapore, 2006.

[Nghia et.al.] Nghia H.H., Sateesh T., Meng C.C., Soon H.G., *Team "ROPE"*, Team descriptions for RoboCup, 2004.

[ovt.com] Omnivision Technologies Incorporated, "*OV6620 Single-Chip CMOS CIF Color Digital Camera Technical Documentation*", "Home Page" http://www.ovt.com/ visited on 20/4/2006.

[robocup.org] The Robocup Federation Website, "Home Page", http://www.robocup.org/, visited on 6/5/2006.

[Rowe] Rowe A., *CMUcam2 Vision Sensor User Guide*, "home page" http://www.cs.cmu.edu/~cmucam2/, visited on 28/5/2006.

[Rowe et.al. 1] Rowe A., Rosenberg C., Nourbakhsh I., CMUcam Website, "Home Page", http://www.cs.cmu.edu/~cmucam/, visited on 20/8/2006.

[Rowe et.al. 2] Rowe A., Rosenberg C., Nourbakhsh I., "*A Second Generation Low Cost Embedded Color Vision System*", The Proceedings of IROS, 2005.

[Rowe et.al. 3] Rowe A., Rosenberg C., Nourbakhsh I., "*A Low Cost Embedded Color Vision System*," The Proceedings of IROS, 2002.

[Stoica et.al.] Stoica A., Keymeulen D., "*Humanoids in support of lunar and planetary surface operations*", Proceeding of the IEEE Aerospace Conference, CA, USA, January 2006.

[Suzuki et.al.] Suzuki S., Kato T., Ishizuka H., Takahashi Y., Uchibe E., Asada M., "*An Application of Vision-Based Learning in RoboCup for a Real Robot with an Omnidirectional Vision System and the Team Description of Osaka University "Trackies"*", Lecture Notes in Computer Science, Vol. 1604, pages 316-325, 1999.

[Team RO-PE] Team RO-PE Project Website, "Home Page", http://guppy.mpe .nus.edu.sg/ ~legged_group/, visited on 7/4/2007.

[Ulrich et.al.] [13] Ulrich I., Nourbakhsh I., *Appearance-Based Obstacle Detection with Monocular Color Vision*. AAAI National Conference on Artificial Intelligence, Austin, TX, 2000.

[Veloso] Veloso M.M., "*Entertainment Robotics*", Communications of the ACM, Vol. 45, Issue 3, pp 59-63, NY, USA, March 2002.

[Xu et.al.] Xu J., Stoica A., Keymeulen D., "*Integrating Vision Capabilities into HOAP-2 Humanoid Robot*", CA Institute of Technology, USA, September 2004.

[Yamato et.al.] Yamato N., Akazawa Y., Ishiguro H., Takahashi T., Maeda T., Imagawa T., Takayama H., Mitsunaga N., Miyashita T., "*VisiON NEXTA: Fully autonomous humanoid robot*", RoboCup 2005 Humanoid League Team Descriptions, Osaka, Japan, 2005.

[Zhou et.al.] Zhou C., Yue P.K., "*Robo-Erectus: a low-cost autonomous humanoid soccer robot*", Springer Advanced Robotics, Vol.18, No. 7, pp 717-720, August 2004.

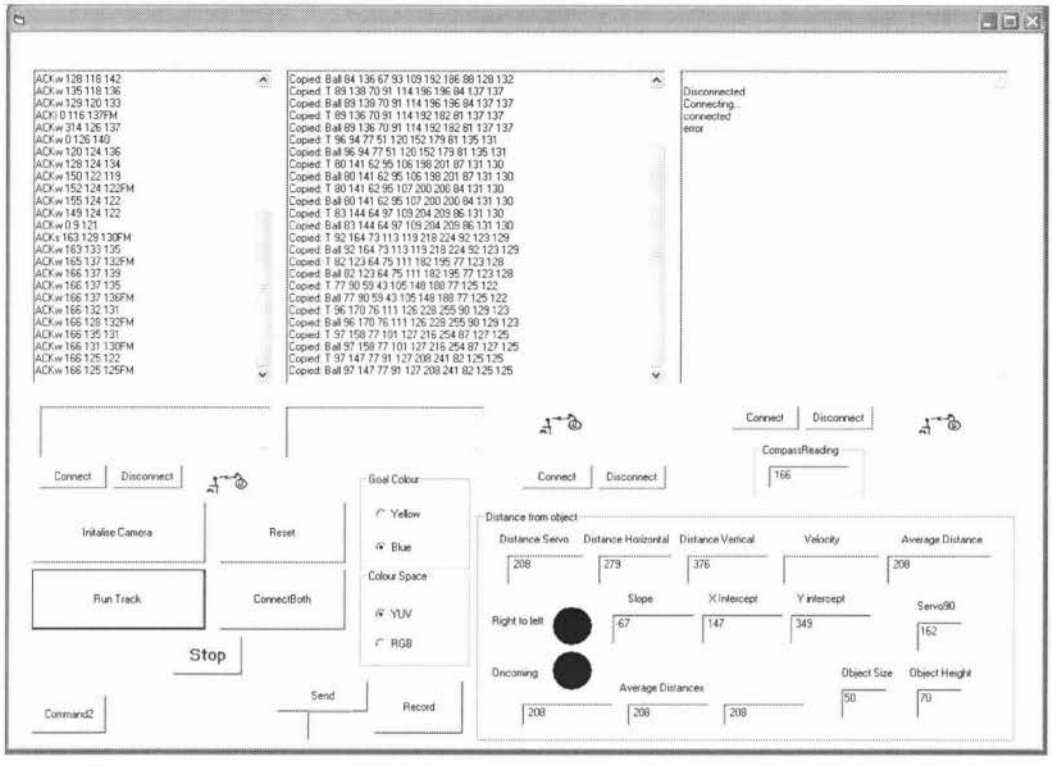
Appendix A (How to operate the robot)

1. Calibrate camera

If the camera during a run produces erroneous results, calibration of the colour space is needed. By using the CMUcam2 GUI (with the below settings) to grab a frame of the objects you wish to track in the current environment. From the images capture find the colour boundaries for the object and update the Pirkus' VB6 code with the correct colour space.

2. Run the VB6 code

If the code has been modified then create a new .exe and start-up the A.I. program.



3. **Power up the all boards, including the Pirkus servos.**

(If an error occurs then restart the program and re-power up all devices.)

4. **Insert the appropriate vales into the Servo90, Object Size and Object Height boxes high-lighted in yellow.**

5. **Select the RGB or YUV colour space and the colour of the goal in the check boxes.**

6. **Click in order**

   - Reset
   - Connect Both

- Initialise Camera
- Run Track

This sequence will start the robot operating. If "connected" does not appear then the serial is not connected appropriately. If "ACK" does not appear on the first two windows then the program and the power sequence have to be restarted.

7. **Click on the "command2" button to open the real-time tracking window. Of the balls trajectory.**

8. **Click the "Stop" button to stop the program if you which to restart then return to step 6.**

** (For Further Details please refer to comments in the code.)

Appendix B (Result Spreadsheets)

Trajectory Calculation, Intercepting 200X 170Y, Oncoming, Heading Left

Slope	Y Intercept	X Intercept	Oncoming	Heading Left	Xco(1)	Xco(2)	Xco(3)	Xco(4)	Xco(N)
-37	97	127	#TRUE#	#TRUE#	-502.644	-438.386	-346.282	-274.965	-274.965
-40	80	95	#TRUE#	#TRUE#	-463.781	-423.331	-380.091	-297.088	-297.088
-32	164	266	#TRUE#	#TRUE#	-506.732	-434.489	-357.774	-274.81	-274.81
-32	154	243	#TRUE#	#TRUE#	-476.941	-424.106	-373.413	-307.238	-307.238
-34	141	210	#TRUE#	#TRUE#	-521.336	-428.213	-341.036	-261.634	-261.634
-39	87	106	#TRUE#	#TRUE#	-457.504	-410.777	-366.143	-313.666	-313.666
-34	153	227	#TRUE#	#TRUE#	-471.452	-400.316	-307.698	-241.442	-241.442
-30	172	300	#TRUE#	#TRUE#	-517.683	-449.096	-340.913	-278.838	-278.838
-32	154	247	#TRUE#	#TRUE#	-449.096	-391.977	-304.01	-260.765	-260.765
-33	158	248	#TRUE#	#TRUE#	-463.375	-330.025	-272.24	-221.081	-221.081
-28	186	343	#TRUE#	#TRUE#	-515.445	-381.06	-318.458	-247.269	-247.269
-30	180	313	#TRUE#	#TRUE#	-523.526	-437.367	-354.781	-315.656	-315.656
-31	164	269	#TRUE#	#TRUE#	-472.657	-416.966	-332.747	-249.097	-249.097
-36	132	183	#TRUE#	#TRUE#	-434.489	-307.698	-267.281	-249.713	-249.713
-32	164	262	#TRUE#	#TRUE#	-492.859	-373.349	-327.592	-248.458	-248.458
-33	138	209	#TRUE#	#TRUE#	-522.796	-453.431	-391.947	-295.098	-295.098
-33	143	223	#TRUE#	#TRUE#	-506.732	-459.272	-325.603	-276.53	-276.53
-40	78	91	#TRUE#	#TRUE#	-484.006	-430.305	-382.881	-309.024	-309.024

Yco(1)	Yco(2)	Yco(3)	Yco(4)	Yco(N)
492.9144	429.8997	339.5788	325.3773	325.3773
476.584	435.0173	390.5839	335.325	335.325
474.1924	446.4839	367.6505	341.0626	341.0626
467.7085	415.8965	366.185	363.5671	363.5671
487.8578	440.0339	350.4505	324.7103	324.7103
470.134	422.1172	376.2505	354.0374	354.0374
484.4673	411.3672	347.3009	330.1312	330.1312
462.2126	440.4022	367.1232	329.9601	329.9601
440.4022	384.3892	359.7481	308.5738	308.5738
454.4054	355.3987	354.5341	287.9102	287.9102
460.2146	410.3573	342.9415	322.0153	322.0153
489.9077	409.2813	400.4439	356.2828	356.2828
463.5075	408.8949	358.3299	324.3947	324.3947
446.4839	347.3009	331.7184	309.9153	309.9153
461.2101	421.4018	369.7557	308.358	308.358
489.2244	424.3133	402.7672	333.0795	333.0795
474.1924	429.7795	367.5102	312.1217	312.1217
497.3674	442.184	393.4506	348.7979	348.7979

Trajectory Calculation, Intercepting 500X 380Y, Oncoming, Heading Left

Slope	Y Intercept	X Intercept	Oncoming	Heading Left	Xco(1)	Xco(2)	Xco(3)	Xco(4)	Xco(N)
-31	374	627	#TRUE#	#TRUE#	-214.505	-138.909	-69.1041	-27.7614	-27.7614
-36	345	478	#TRUE#	#TRUE#	-292.168	-259.705	-182.032	-102.664	-102.664
-33	357	558	#TRUE#	#TRUE#	-262.861	-188.612	-132.673	-96.1753	-96.1753
-45	290	294	#TRUE#	#TRUE#	-271.685	-199.792	-154.99	-110.17	-110.17
-36	350	475	#TRUE#	#TRUE#	-327.309	-285.806	-210.246	-146.094	-146.094
-33	365	570	#TRUE#	#TRUE#	-308.916	-285.806	-205.213	-146.094	-146.094
-37	348	467	#TRUE#	#TRUE#	-221.088	-142.502	-130.228	-57.6241	-57.6241
-36	356	482	#TRUE#	#TRUE#	-263.087	-179.734	-110.933	-47.0183	-47.0183
-35	341	484	#TRUE#	#TRUE#	-309.388	-271.657	-170.111	-138.61	-138.61
-32	371	597	#TRUE#	#TRUE#	-257.928	-202.502	-143.101	-93.1853	-93.1853
-40	314	376	#TRUE#	#TRUE#	-266.956	-225.257	-161.52	-110.17	-110.17
-26	375	770	#TRUE#	#TRUE#	-271.657	-197.082	-139.508	-67.3113	-67.3113
-33	353	539	#TRUE#	#TRUE#	-221.079	-150.189	-38.7665	36.05123	36.05123
-39	319	389	#TRUE#	#TRUE#	-438.238	-341.458	-289.579	-164.613	-164.613
-36	348	479	#TRUE#	#TRUE#	-214.505	-138.011	-86.2905	8.13807	8.13807

Yco(1)	Yco(2)	Yco(3)	Yco(4)	Yco(N)
510.7872	442.7191	418.3307	395.0257	395.0257
578.3958	514.1295	463.5389	430.9397	430.9397
520.3776	480.2933	460.2596	403.702	403.702
570.6238	475.7512	423.5316	418.7502	418.7502
611.9678	534.3696	500.6452	465.6183	465.6183
577.5777	534.3696	488.6592	465.6183	465.6183
526.4611	454.1687	415.0491	407.9503	407.9503
552.5661	491.1464	421.6514	400.2478	400.2478
578.4595	507.9156	464.8517	441.765	441.765
541.7314	482.2052	456.077	436.1565	436.1565
560.6921	473.1122	441.3744	418.7502	418.7502
507.9156	469.2972	444.6273	407.4778	407.4778
493.7662	442.1902	413.1854	306.8897	306.8897
662.1055	638.4218	541.424	449.8262	449.8262
510.7872	439.8567	403.8848	347.9048	347.9048

Trajectory Calculation, Away, Straight

Slope	Y Intercept	X Intercept	Oncoming	Heading Left	Xco(1)	Xco(2)	Xco(3)	Xco(4)	Xco(N)	Yco(1)	Yco(2)	Yco(3)	Yco(4)	Yco(N)
90	0	0	#FALSE#	#FALSE#	0	0	0	0	0	334	363	398	424	424
90	0	0	#FALSE#	#FALSE#	0	0	0	0	0	704	717	786	842	842
90	0	0	#FALSE#	#FALSE#	0	0	0	0	0	466	501	554	588	588
90	0	0	#FALSE#	#FALSE#	0	0	0	0	0	384	440	509	533	533
90	0	0	#FALSE#	#FALSE#	0	0	0	0	0	576	622	677	679	679
90	0	0	#FALSE#	#FALSE#	0	0	0	0	0	753	785	867	906	906
90	0	0	#FALSE#	#FALSE#	0	0	0	0	0	768	768	785	785	785
72	711	-235	#FALSE#	#FALSE#	0	0	0	38.76216	38.76216	640	699	794	828.0933	828.0933
90	0	0	#FALSE#	#FALSE#	0	0	0	0	0	891	906	922	1068	1068
87	0	0	#FALSE#	#FALSE#	45.86933	46.75773	49.93726	56.20279	56.20279	979.9271	998.9063	1066.832	1200.685	1200.685
87	0	0	#FALSE#	#FALSE#	21.36828	26.09081	28.4287	37.35942	37.35942	456.5002	557.3897	607.335	798.1261	798.1261
90	0	0	#FALSE#	#FALSE#	0	0	0	0	0	666	765	786	786	786
90	0	0	#FALSE#	#FALSE#	0	0	0	0	0	640	746	891	981	981
87	0	0	#FALSE#	#FALSE#	23.42562	29.50413	33.99287	39.69731	39.69731	500.452	630.3099	726.2048	848.0714	848.0714
87	0	0	#FALSE#	#FALSE#	18.37579	22.95805	33.24475	36.47103	36.47103	392.5702	490.463	710.2224	779.1469	779.1469
90	0	0	#FALSE#	#FALSE#	0	0	0	0	0	386	541	748	808	808
87	0	0	#FALSE#	#FALSE#	22.63074	28.89628	46.47718	47.55261	47.55261	483.4706	617.3241	992.9128	1015.888	1015.888
90	0	0	#FALSE#	#FALSE#	0	0	0	0	0	446	562	711	840	840
79	499	-96	#FALSE#	#FALSE#	0	0	0	30.90686	30.90686	363	515	619	660.277	660.277
90	0	0	#FALSE#	#FALSE#	0	0	0	0	0	379	487	576	660	660

Trajectory Calculation, Oncoming, Straight

Slope	Y Intercept	X Intercept	Oncoming	Heading Left	Xco(1)	Xco(2)	Xco(3)	Xco(4)	Xco(N)	Yco(1)	Yco(2)	Yco(3)	Yco(4)	Yco(N)
-87	0	0	#TRUE#	#FALSE#	-45.8693	-39.183	-34.5072	-32.0291	-32.0291	979.9271	837.0834	737.1928	684.2508	684.2508
90	0	0	#TRUE#	#FALSE#	0	0	0	0	0	739	656	574	438	438
90	0	0	#TRUE#	#FALSE#	0	0	0	0	0	731	574	449	309	309
-89	0	0	#TRUE#	#FALSE#	-19.363	-17.2583	-14.569	-12.0434	-12.0434	827.7736	737.7982	622.8297	514.8592	514.8592
-89	0	0	#TRUE#	#FALSE#	-15.2238	-13.4231	-10.2427	-8.48885	-8.48885	650.822	573.843	437.8802	362.9007	362.9007
90	0	0	#TRUE#	#FALSE#	0	0	0	0	0	768	649	513	430	430
90	0	0	#TRUE#	#FALSE#	0	0	0	0	0	805	739	662	594	594
-87	0	0	#TRUE#	#FALSE#	-55.969	-55.969	-49.5632	-41.9884	-41.9884	1195.691	1195.691	1058.841	897.0178	897.0178
-87	0	0	#TRUE#	#FALSE#	-16.2717	-12.8116	-10.1464	-7.99557	-7.99557	347.6194	273.7003	216.7627	170.813	170.813
90	0	0	#TRUE#	#FALSE#	0	0	0	0	0	855	756	692	672	672
-87	0	0	#TRUE#	#FALSE#	-34.554	-30.299	-23.5659	-18.3758	-18.3758	738.1917	647.2913	503.4488	392.5702	392.5702

Trajectory Calculation, Sideways, Right

Slope	Y Intercept	X Intercept	Oncoming	Heading Left	Xco(1)	Xco(2)	Xco(3)	Xco(4)	Xco(N)	Yco(1)	Yco(2)	Yco(3)	Yco(4)	Yco(N)
7	429	-3616	#TRUE#	#FALSE#	61.12069	20.33961	-20.1058	-119.101	-119.101	432.7046	434.5242	429.5297	413.1767	413.1767
4	412	-6293	#TRUE#	#FALSE#	28.88311	-9.56457	-163.756	-229.529	-229.529	410.9863	408.8882	416.9988	384.734	384.734
2	414	-14884	#TRUE#	#FALSE#	58.60314	-39.2335	-136.039	-208.756	-208.756	414.8815	418.1635	400.5277	413.2673	413.2673
-3	410	7087	#FALSE#	#FALSE#	-9.75165	-107.625	-149.492	-205.913	-205.913	416.886	409.0793	408.506	432.4823	432.4823
8	422	-3138	#TRUE#	#FALSE#	-97.334	-137.004	-245.383	-292.947	-292.947	408.5659	403.3684	390.2666	381.5009	381.5009
9	426	-2838	#TRUE#	#FALSE#	-49.0017	-118.27	-224.406	-266.219	-266.219	417.1317	410.294	396.9607	382.4699	382.4699
3	419	-8395	#TRUE#	#FALSE#	100.115	9.891965	-59.1626	-160.101	-160.101	420.2392	422.8843	418.8422	407.6908	407.6908
5	416	-4449	#TRUE#	#FALSE#	89.63342	49.35168	9.962121	-136.039	-136.039	419.5317	420.1112	425.8835	400.5277	400.5277
-8	436	3119	#FALSE#	#FALSE#	140.8628	100.8103	60.42137	0	0	414.7309	423.1575	427.7537	435	435
5	420	-4684	#TRUE#	#FALSE#	29.65426	-9.82181	-48.6517	-117.993	-117.993	421.9593	419.8851	414.1522	409.3332	409.3332
-1	417	19377	#FALSE#	#FALSE#	171.9139	69.91904	0	-38.8599	-38.8599	409.3673	423.2639	417	414.181	414.181
3	420	-8295	#TRUE#	#FALSE#	69.91904	0	-39.2335	-78.682	-78.682	423.2639	420	418.1635	415.6178	415.6178
5	417	-4974	#TRUE#	#FALSE#	-38.9533	-136.039	-183.075	-223.914	-223.914	415.1766	400.5277	408.8859	396.0901	396.0901
-4	433	6287	#FALSE#	#FALSE#	177.3346	132.6225	70.73394	30.3553	30.3553	422.2753	422.6822	428.1971	431.9347	431.9347
9	431	-2686	#TRUE#	#FALSE#	-370.558	-378.411	-425.932	-469.253	-469.253	363.3843	371.0861	380.2931	345.9636	345.9636
-1	418	33445	#FALSE#	#FALSE#	49.00167	9.798424	-89.2156	-132.024	-132.024	417.1317	418.8854	417.5758	420.774	420.774
9	432	-2671	#TRUE#	#FALSE#	-69.4301	-109.152	-214.119	-257.203	-257.203	420.304	414.8818	400.3363	388.5909	388.5909

Trajectory Calculation, Sideways, Left

Slope	Y Intercept	X Intercept	Oncoming	Heading Left	Xco(1)	Xco(2)	Xco(3)	Xco(4)	Xco(N)	Yco(1)	Yco(2)	Yco(3)	Yco(4)	Yco(N)
1	415	-18777	#FALSE#	#TRUE#	-138.933	-78.868	-38.9533	0	0	409.0496	416.6003	415.1766	412	412
1	411	-16811	#FALSE#	#TRUE#	-147.774	-48.4183	68.94115	138.9331	138.9331	403.8106	412.1658	417.3441	409.0496	409.0496
5	411	-4490	#FALSE#	#TRUE#	-48.0683	48.0683	122.9787	191.6563	191.6563	409.1863	409.1863	426.6289	428.0524	428.0524
0	407	-50734	#FALSE#	#TRUE#	-147.774	-48.4183	47.36828	117.9931	117.9931	403.8106	412.1658	403.2273	409.3332	409.3332
-5	408	4844	#TRUE#	#TRUE#	-123.256	-77.0079	-37.4587	0	0	427.5898	406.7748	399.2466	420	420
3	403	-7207	#FALSE#	#TRUE#	-37.4587	57.62408	161.1971	209.207	209.207	399.2466	407.9503	410.4832	414.1599	414.1599
2	409	-14996	#FALSE#	#TRUE#	-262.17	-149.836	-79.426	-9.30733	-9.30733	396.0959	409.4451	419.548	397.8911	397.8911
11	431	-2142	#FALSE#	#TRUE#	-230.042	-186.138	-89.2156	-49.1183	-49.1183	385.5928	390.9496	417.5758	418.1248	418.1248
11	429	-2158	#FALSE#	#TRUE#	-248.923	-208.931	-108.643	-86.7083	-86.7083	376.0826	390.6365	412.9476	405.8407	405.8407
4	414	-6463	#FALSE#	#TRUE#	-237.215	-191.297	-127.833	-28.8831	-28.8831	397.6158	401.7842	407.416	410.9863	410.9863
6	415	-3821	#FALSE#	#TRUE#	-226.455	-211.761	-172.301	-123.342	-123.342	379.5814	395.9273	410.2893	393.104	393.104
9	412	-2562	#FALSE#	#TRUE#	-104.826	-9.44765	60.14164	102.2007	102.2007	398.4413	403.8895	425.7734	428.9942	428.9942
-4	425	5617	#TRUE#	#TRUE#	69.43009	171.9139	246.949	264.3776	264.3776	420.304	409.3673	413.9326	399.4314	399.4314
-2	399	9851	#TRUE#	#TRUE#	-219.778	-166.493	-107.88	-37.3653	-37.3653	410.9179	396.4594	410.0463	398.251	398.251

Distance Calculation with Zero Readings

Servo	Horizontal	Verical	Average1	Average2	Average3
201.0339	0	0	120.6203	0	67.0113
201.0339	0	0	120.6203	0	67.0113
201.0339	0	0	120.6203	0	67.0113
201.0339	0	0	120.6203	0	67.0113
201.0339	0	0	120.6203	0	67.0113
201.0339	0	0	120.6203	0	67.0113
201.0339	0	0	120.6203	0	67.0113
201.0339	0	0	120.6203	0	67.0113
209.1143	0	0	125.4686	0	69.70477
217.4544	0	0	130.4726	0	72.48479
217.4544	0	0	130.4726	0	72.48479
217.4544	0	0	130.4726	0	72.48479
217.4544	0	0	130.4726	0	72.48479
217.4544	0	0	130.4726	0	72.48479
296.5838	0	0	177.9503	0	98.86125
296.5838	0	0	177.9503	0	98.86125
296.5838	0	0	177.9503	0	98.86125
296.5838	0	0	177.9503	0	98.86125
296.5838	0	0	177.9503	0	98.86125
296.5838	0	0	177.9503	0	98.86125
296.5838	0	0	177.9503	0	98.86125
296.5838	253.5708	0	285.8305	275.0773	183.3848
296.5838	0	0	177.9503	0	98.86125
296.5838	0	0	177.9503	0	98.86125
296.5838	0	0	177.9503	0	98.86125
296.5838	253.5708	0	285.8305	275.0773	183.3848
296.5838	0	301.1095	297.7152	298.8466	199.2311
378.5117	0	323.2677	364.7007	350.8897	233.9265
378.5117	250.3215	330.4816	366.5042	354.4966	319.7716
378.5117	292.6969	337.4347	353.1334	357.9732	336.2145
378.5117	0	307.7056	360.8102	343.1087	228.7391
378.5117	0	0	227.107	0	126.1706
378.5117	405.729	394.7887	387.2106	400.2589	393.0098
378.5117	405.729	394.7887	387.2106	400.2589	393.0098
378.5117	405.729	382.3957	384.732	380.4537	388.8788
378.5117	405.729	382.3957	384.732	380.4537	388.8788
378.5117	405.729	394.7887	387.2106	400.2589	393.0098
378.5117	405.729	394.7887	387.2106	400.2589	393.0098
378.5117	405.729	394.7887	387.2106	400.2589	393.0098
378.5117	405.729	394.7887	387.2106	400.2589	393.0098
378.5117	405.729	394.7887	387.2106	400.2589	393.0098
378.5117	405.729	394.7887	387.2106	400.2589	393.0098
378.5117	419.7499	394.7887	390.0148	386.6502	397.6835
497.1772	392.5486	450.0935	485.4063	473.6354	446.6065
497.1772	392.5486	360.1114	448.8383	376.33	416.6124
497.1772	392.5486	457.5395	487.2678	477.3584	449.0885
497.1772	362.996	421.3864	478.2295	392.1912	427.1865

497.1772	302.5715	317.5894	422.3385	310.0804	372.446
497.1772	302.5715	0	448.5258	399.8743	266.5829
497.1772	0	0	298.3063	0	165.7257
497.1772	491.4387	457.5395	488.102	494.3079	482.0518
497.1772	473.2314	457.5395	484.4605	465.3855	475.9827
497.1772	473.2314	457.5395	484.4605	465.3855	475.9827
497.1772	491.4387	457.5395	488.102	494.3079	482.0518
497.1772	473.2314	457.5395	484.4605	465.3855	475.9827
497.1772	491.4387	457.5395	488.102	494.3079	482.0518
497.1772	473.2314	457.5395	484.4605	465.3855	475.9827
497.1772	473.2314	457.5395	484.4605	465.3855	475.9827
497.1772	473.2314	457.5395	484.4605	465.3855	475.9827
497.1772	473.2314	457.5395	484.4605	465.3855	475.9827
497.1772	473.2314	457.5395	484.4605	465.3855	475.9827
614.0095	403.5276	451.1715	539.3455	427.3496	489.5695
614.0095	493.0357	435.4663	583.766	464.251	514.1705
614.0095	454.9045	451.1715	549.6209	453.038	506.6952
614.0095	454.9045	475.835	554.5536	465.3698	514.9163
614.0095	454.9045	451.1715	549.6209	453.038	506.6952
614.0095	454.9045	451.1715	549.6209	453.038	506.6952
614.0095	454.9045	420.2371	543.434	437.5708	496.3837
614.0095	473.4899	420.2371	578.8796	446.8635	502.5788
614.0095	420.0516	435.4663	539.5093	427.759	489.8425
614.0095	454.9045	435.4663	546.4798	445.1854	501.4601
614.0095	454.9045	475.835	554.5536	465.3698	514.9163
614.0095	454.9045	451.1715	549.6209	453.038	506.6952
614.0095	454.9045	435.4663	546.4798	445.1854	501.4601
614.0095	454.9045	420.2371	543.434	437.5708	496.3837
614.0095	493.0357	420.2371	583.766	456.6364	509.0941
614.0095	0	427.7981	567.4566	520.9038	347.2692
614.0095	584.394	520.7841	606.6056	599.2018	573.0625
614.0095	584.394	520.7841	606.6056	599.2018	573.0625
614.0095	584.394	520.7841	606.6056	599.2018	573.0625
614.0095	584.394	520.7841	606.6056	599.2018	573.0625
614.0095	559.1543	520.7841	584.3934	539.9692	564.6493
614.0095	559.1543	520.7841	584.3934	539.9692	564.6493
614.0095	559.1543	520.7841	584.3934	539.9692	564.6493
614.0095	559.1543	520.7841	584.3934	539.9692	564.6493
614.0095	584.394	520.7841	606.6056	599.2018	573.0625
614.0095	584.394	520.7841	606.6056	599.2018	573.0625
737.5349	545.68	450.285	689.5712	497.9825	577.8333
737.5349	521.9584	476.442	642.201	499.2002	578.6451
737.5349	545.68	433.6808	689.5712	489.6804	572.2986
737.5349	661.2296	645.7471	703.9163	653.4884	681.5039
737.5349	661.2296	645.7471	703.9163	653.4884	681.5039
737.5349	661.2296	645.7471	703.9163	653.4884	681.5039
737.5349	661.2296	631.5026	701.0674	646.3661	676.7557
737.5349	661.2296	645.7471	703.9163	653.4884	681.5039
737.5349	661.2296	645.7471	703.9163	653.4884	681.5039
737.5349	661.2296	645.7471	703.9163	653.4884	681.5039

737.5349	661.2296	617.8204	698.331	639.525	672.195
737.5349	661.2296	645.7471	703.9163	653.4884	681.5039
737.5349	661.2296	617.8204	698.331	639.525	672.195
737.5349	661.2296	645.7471	703.9163	653.4884	681.5039
737.5349	661.2296	645.7471	703.9163	653.4884	681.5039
737.5349	697.2525	617.8204	727.4643	717.3937	684.2026
737.5349	661.2296	645.7471	703.9163	653.4884	681.5039
737.5349	661.2296	617.8204	698.331	639.525	672.195
737.5349	661.2296	645.7471	703.9163	653.4884	681.5039
737.5349	661.2296	645.7471	703.9163	653.4884	681.5039
847.959	824.3356	776.3845	828.9194	836.1473	816.2264
847.959	824.3356	735.7449	842.0532	836.1473	802.6798
847.959	773.5958	735.7449	810.6436	754.6703	785.7666
847.959	773.5958	735.7449	810.6436	754.6703	785.7666
847.959	773.5958	735.7449	810.6436	754.6703	785.7666
847.959	773.5958	735.7449	810.6436	754.6703	785.7666
847.959	773.5958	735.7449	810.6436	754.6703	785.7666
847.959	773.5958	735.7449	810.6436	754.6703	785.7666
847.959	773.5958	776.3845	818.7715	774.9901	799.3131
847.959	773.5958	735.7449	810.6436	754.6703	785.7666
847.959	773.5958	755.5236	814.5993	764.5597	792.3595
847.959	773.5958	776.3845	818.7715	774.9901	799.3131
847.959	773.5958	776.3845	818.7715	774.9901	799.3131
847.959	773.5958	776.3845	818.7715	774.9901	799.3131
847.959	773.5958	776.3845	818.7715	774.9901	799.3131
993.9274	873.7934	862.8435	943.6838	868.3184	910.1881
993.9274	873.7934	862.8435	943.6838	868.3184	910.1881
993.9274	873.7934	921.151	955.3453	897.4722	929.6239
993.9274	873.7934	862.8435	943.6838	868.3184	910.1881
993.9274	873.7934	862.8435	943.6838	868.3184	910.1881
993.9274	940.8627	811.5827	980.6612	967.3951	915.4576
993.9274	873.7934	811.5827	963.8939	842.688	893.1012
993.9274	873.7934	862.8435	943.6838	868.3184	910.1881
993.9274	940.8627	811.5827	980.6612	967.3951	915.4576
993.9274	940.8627	862.8435	980.6612	967.3951	932.5446
993.9274	873.7934	862.8435	943.6838	868.3184	910.1881
993.9274	940.8627	836.4212	980.6612	967.3951	923.7371
1086.293	1015.204	916.057	1068.521	1050.748	1005.851
1086.293	1015.204	983.1715	1051.451	999.1875	1028.223
1086.293	1015.204	983.1715	1051.451	999.1875	1028.223
1086.293	1015.204	983.1715	1051.451	999.1875	1028.223
1086.293	1015.204	916.057	1068.521	1050.748	1005.851
1086.293	1015.204	916.057	1068.521	1050.748	1005.851
1086.293	1015.204	983.1715	1051.451	999.1875	1028.223
1086.293	1015.204	983.1715	1051.451	999.1875	1028.223
1086.293	1015.204	916.057	1068.521	1050.748	1005.851
1086.293	1015.204	916.057	1068.521	1050.748	1005.851
1086.293	1015.204	916.057	1068.521	1050.748	1005.851
1086.293	1015.204	916.057	1068.521	1050.748	1005.851
1086.293	1015.204	916.057	1068.521	1050.748	1005.851

1086.293	1015.204	983.1715	1051.451	999.1875	1028.223
1086.293	1015.204	916.057	1068.521	1050.748	1005.851
1086.293	1015.204	916.057	1068.521	1050.748	1005.851
1086.293	1015.204	983.1715	1051.451	999.1875	1028.223
1086.293	1015.204	916.057	1068.521	1050.748	1005.851
1086.293	1015.204	983.1715	1051.451	999.1875	1028.223
1086.293	1015.204	983.1715	1051.451	999.1875	1028.223
1086.293	1015.204	983.1715	1051.451	999.1875	1028.223
1086.293	1015.204	916.057	1068.521	1050.748	1005.851
1086.293	1015.204	983.1715	1051.451	999.1875	1028.223
1086.293	1015.204	983.1715	1051.451	999.1875	1028.223
1086.293	1015.204	916.057	1068.521	1050.748	1005.851
1086.293	1015.204	983.1715	1051.451	999.1875	1028.223
1086.293	1015.204	983.1715	1051.451	999.1875	1028.223
1086.293	1015.204	1153.726	1103.151	1120.009	1085.074
1086.293	1222.316	1264.844	1149.208	1243.58	1191.151
1086.293	1222.316	1401.318	1120.299	1154.305	1236.643
1086.293	1222.316	1264.844	1149.208	1243.58	1191.151
1086.293	1108.779	1264.844	1091.915	1097.536	1153.306
1086.293	1222.316	1264.844	1149.208	1243.58	1191.151
1086.293	1222.316	1401.318	1120.299	1154.305	1236.643
1086.293	1222.316	1061.337	1080.054	1073.815	1123.315
1086.293	1222.316	1061.337	1080.054	1073.815	1123.315
1086.293	1222.316	1061.337	1080.054	1073.815	1123.315
1086.293	1108.779	1061.337	1085.799	1097.536	1085.47
1086.293	1108.779	1061.337	1085.799	1097.536	1085.47
1086.293	1222.316	1061.337	1080.054	1073.815	1123.315
1086.293	1222.316	1061.337	1080.054	1073.815	1123.315
1086.293	1222.316	1061.337	1080.054	1073.815	1123.315
1086.293	1108.779	1061.337	1085.799	1097.536	1085.47
1086.293	1222.316	1061.337	1080.054	1073.815	1123.315
1086.293	1222.316	1061.337	1080.054	1073.815	1123.315
1086.293	1222.316	1061.337	1080.054	1073.815	1123.315
1086.293	1222.316	1061.337	1080.054	1073.815	1123.315
1086.293	1222.316	1061.337	1080.054	1073.815	1123.315
1086.293	1222.316	1061.337	1080.054	1073.815	1123.315
1086.293	1108.779	1061.337	1085.799	1097.536	1085.47
1331.307	1213.927	1143.665	1301.962	1178.796	1229.633
1331.307	1355.041	1143.665	1337.24	1343.174	1276.671
1331.307	1355.041	1143.665	1337.24	1343.174	1276.671
1331.307	1355.041	1143.665	1337.24	1343.174	1276.671
1331.307	1355.041	1143.665	1337.24	1343.174	1276.671
1331.307	1355.041	1143.665	1337.24	1343.174	1276.671
1331.307	1355.041	1143.665	1337.24	1343.174	1276.671
1331.307	1355.041	1143.665	1337.24	1343.174	1276.671
1331.307	1355.041	1143.665	1337.24	1343.174	1276.671
1331.307	1355.041	1143.665	1337.24	1343.174	1276.671
1331.307	1355.041	1143.665	1337.24	1343.174	1276.671
1331.307	1355.041	1143.665	1337.24	1343.174	1276.671
1331.307	1355.041	1143.665	1337.24	1343.174	1276.671
1331.307	1355.041	1143.665	1337.24	1343.174	1276.671

1331.307	1355.041	1254.886	1337.24	1343.174	1313.745
1331.307	1355.041	1254.886	1337.24	1343.174	1313.745
1331.307	1355.041	1254.886	1337.24	1343.174	1313.745
1331.307	1355.041	1254.886	1337.24	1343.174	1313.745
1331.307	1355.041	1254.886	1337.24	1343.174	1313.745
1331.307	1355.041	1254.886	1337.24	1343.174	1313.745
1331.307	1213.927	1143.665	1301.962	1178.796	1229.633
1331.307	1213.927	1254.886	1292.547	1234.407	1266.707
1331.307	1213.927	1254.886	1292.547	1234.407	1266.707
1331.307	1355.041	1196.582	1337.24	1343.174	1294.31
1331.307	1213.927	1254.886	1292.547	1234.407	1266.707
1331.307	1213.927	1254.886	1292.547	1234.407	1266.707
1331.307	1355.041	1254.886	1337.24	1343.174	1313.745
1331.307	1213.927	1254.886	1292.547	1234.407	1266.707
1331.307	1213.927	1254.886	1292.547	1234.407	1266.707
1331.307	1213.927	1254.886	1292.547	1234.407	1266.707
1331.307	1213.927	1254.886	1292.547	1234.407	1266.707
1331.307	1355.041	1254.886	1337.24	1343.174	1313.745
1331.307	1213.927	1254.886	1292.547	1234.407	1266.707
1331.307	1355.041	1254.886	1337.24	1343.174	1313.745
1331.307	1213.927	1254.886	1292.547	1234.407	1266.707
1331.307	1213.927	1143.665	1301.962	1178.796	1229.633
1331.307	1213.927	1143.665	1301.962	1178.796	1229.633
1331.307	1355.041	1254.886	1337.24	1343.174	1313.745

Distance calculation, Small Distance Modification

Servo	Horizontal	Verical	Average1	Average2	Average3	AVG AVGS
201.0339			120.6203		67.0113	62.54388262
201.0339			120.6203		67.0113	62.54388262
201.0339			120.6203		67.0113	62.54388262
201.0339			120.6203		67.0113	62.54388262
201.0339			120.6203		67.0113	62.54388262
201.0339			120.6203		67.0113	62.54388262
201.0339			120.6203		67.0113	62.54388262
201.0339			120.6203		67.0113	62.54388262
217.4544			130.4726		72.48479	67.65246887
217.4544			130.4726		72.48479	67.65246887
217.4544			130.4726		72.48479	67.65246887
217.4544			130.4726		72.48479	67.65246887
217.4544			130.4726		72.48479	67.65246887
217.4544			130.4726		72.48479	67.65246887
217.4544			130.4726		72.48479	67.65246887
296.5838			177.9503		98.86125	92.27050001
296.5838			177.9503		98.86125	92.27050001
296.5838			177.9503		98.86125	92.27050001
296.5838			177.9503		98.86125	92.27050001
296.5838			177.9503		98.86125	92.27050001
296.5838			177.9503		98.86125	92.27050001
296.5838			177.9503		98.86125	92.27050001
296.5838			177.9503		98.86125	92.27050001
296.5838			177.9503		98.86125	92.27050001
296.5838			177.9503		98.86125	92.27050001
296.5838			177.9503		98.86125	92.27050001
296.5838			177.9503		98.86125	92.27050001
378.5117	337.744	382.3957	371.135	380.4537	366.2171	372.6019356
378.5117	308.7846	382.3957	379.4827	380.4537	356.564	372.1668123
378.5117	250.3215	330.4816	366.5042	354.4966	319.7716	346.9241319
378.5117	292.6969	337.4347	353.1334	357.9732	336.2145	349.1070065
378.5117	405.729	394.7887	387.2106	400.2589	393.0098	393.4930809
378.5117	405.729	394.7887	387.2106	400.2589	393.0098	393.4930809
378.5117	405.729	382.3957	384.732	380.4537	388.8788	384.688159
378.5117	405.729	382.3957	384.732	380.4537	388.8788	384.688159
378.5117	405.729	394.7887	387.2106	400.2589	393.0098	393.4930809
378.5117	405.729	394.7887	387.2106	400.2589	393.0098	393.4930809
378.5117	405.729	394.7887	387.2106	400.2589	393.0098	393.4930809
378.5117	405.729	394.7887	387.2106	400.2589	393.0098	393.4930809
378.5117	405.729	394.7887	387.2106	400.2589	393.0098	393.4930809
378.5117	405.729	394.7887	387.2106	400.2589	393.0098	393.4930809
378.5117	405.729	394.7887	387.2106	400.2589	393.0098	393.4930809
378.5117	419.7499	394.7887	390.0148	386.6502	397.6835	391.4494764
378.5117	419.7499	394.7887	390.0148	386.6502	397.6835	391.4494764
378.5117	405.729	394.7887	387.2106	400.2589	393.0098	393.4930809
497.1772	392.5486	450.0935	485.4063	473.6354	446.6065	468.5493697

497.1772	392.5486	360.1114	448.8383	376.33	416.6124	413.9269218
497.1772	392.5486	457.5395	487.2678	477.3584	449.0885	471.2381964
497.1772	362.996	421.3864	478.2295	392.1912	427.1865	432.535736
497.1772	302.5715	317.5894	422.3385	310.0804	372.446	368.288319
497.1772	491.4387	457.5395	488.102	494.3079	482.0518	488.1539004
497.1772	473.2314	457.5395	484.4605	465.3855	475.9827	475.2762176
497.1772	473.2314	457.5395	484.4605	465.3855	475.9827	475.2762176
497.1772	491.4387	457.5395	488.102	494.3079	482.0518	488.1539004
497.1772	473.2314	457.5395	484.4605	465.3855	475.9827	475.2762176
497.1772	491.4387	457.5395	488.102	494.3079	482.0518	488.1539004
497.1772	473.2314	457.5395	484.4605	465.3855	475.9827	475.2762176
497.1772	473.2314	457.5395	484.4605	465.3855	475.9827	475.2762176
497.1772	473.2314	457.5395	484.4605	465.3855	475.9827	475.2762176
497.1772	473.2314	457.5395	484.4605	465.3855	475.9827	475.2762176
497.1772	473.2314	457.5395	484.4605	465.3855	475.9827	475.2762176
497.1772	473.2314	442.7658	481.5058	485.2043	471.0581	479.2560556
497.1772	473.2314	442.7658	481.5058	485.2043	471.0581	479.2560556
497.1772	473.2314	457.5395	484.4605	465.3855	475.9827	475.2762176
614.0095	454.9045	451.1715	549.6209	453.038	506.6952	503.1180207
614.0095	454.9045	475.835	554.5536	465.3698	514.9163	511.6132228
614.0095	454.9045	451.1715	549.6209	453.038	506.6952	503.1180207
614.0095	454.9045	451.1715	549.6209	453.038	506.6952	503.1180207
614.0095	454.9045	420.2371	543.434	437.5708	496.3837	492.462817
614.0095	473.4899	420.2371	578.8796	446.8635	502.5788	509.4406051
614.0095	420.0516	435.4663	539.5093	427.759	489.8425	485.7035635
614.0095	454.9045	435.4663	546.4798	445.1854	501.4601	497.7084368
614.0095	454.9045	475.835	554.5536	465.3698	514.9163	511.6132228
614.0095	454.9045	451.1715	549.6209	453.038	506.6952	503.1180207
614.0095	454.9045	435.4663	546.4798	445.1854	501.4601	497.7084368
614.0095	454.9045	420.2371	543.434	437.5708	496.3837	492.462817
614.0095	493.0357	420.2371	583.766	456.6364	509.0941	516.4988434
614.0095	584.394	520.7841	606.6056	599.2018	573.0625	592.9566346
614.0095	584.394	520.7841	606.6056	599.2018	573.0625	592.9566346
614.0095	584.394	520.7841	606.6056	599.2018	573.0625	592.9566346
614.0095	584.394	520.7841	606.6056	599.2018	573.0625	592.9566346
614.0095	559.1543	520.7841	584.3934	539.9692	564.6493	563.0039356
614.0095	559.1543	520.7841	584.3934	539.9692	564.6493	563.0039356
614.0095	559.1543	520.7841	584.3934	539.9692	564.6493	563.0039356
614.0095	559.1543	520.7841	584.3934	539.9692	564.6493	563.0039356
614.0095	584.394	520.7841	606.6056	599.2018	573.0625	592.9566346
614.0095	584.394	520.7841	606.6056	599.2018	573.0625	592.9566346
614.0095	584.394	520.7841	606.6056	599.2018	573.0625	592.9566346
614.0095	584.394	520.7841	606.6056	599.2018	573.0625	592.9566346
614.0095	584.394	520.7841	606.6056	599.2018	573.0625	592.9566346
737.5349	521.9584	450.285	683.6408	486.1217	569.9261	579.8961961
737.5349	545.68	450.285	689.5712	497.9825	577.8333	588.4623432
737.5349	545.68	450.285	689.5712	497.9825	577.8333	588.4623432
737.5349	521.9584	476.442	642.201	499.2002	578.6451	573.348787
737.5349	545.68	433.6808	689.5712	489.6804	572.2986	583.8500682
737.5349	661.2296	645.7471	703.9163	653.4884	681.5039	679.6361877

737.5349	661.2296	645.7471	703.9163	653.4884	681.5039	679.6361877
737.5349	661.2296	645.7471	703.9163	653.4884	681.5039	679.6361877
737.5349	661.2296	631.5026	701.0674	646.3661	676.7557	674.7297351
737.5349	661.2296	645.7471	703.9163	653.4884	681.5039	679.6361877
737.5349	661.2296	645.7471	703.9163	653.4884	681.5039	679.6361877
737.5349	661.2296	645.7471	703.9163	653.4884	681.5039	679.6361877
737.5349	661.2296	617.8204	698.331	639.525	672.195	670.0169935
737.5349	661.2296	645.7471	703.9163	653.4884	681.5039	679.6361877
737.5349	661.2296	617.8204	698.331	639.525	672.195	670.0169935
737.5349	661.2296	645.7471	703.9163	653.4884	681.5039	679.6361877
737.5349	661.2296	645.7471	703.9163	653.4884	681.5039	679.6361877
737.5349	697.2525	617.8204	727.4643	717.3937	684.2026	709.686891
737.5349	661.2296	645.7471	703.9163	653.4884	681.5039	679.6361877
737.5349	661.2296	617.8204	698.331	639.525	672.195	670.0169935
737.5349	661.2296	645.7471	703.9163	653.4884	681.5039	679.6361877
737.5349	661.2296	645.7471	703.9163	653.4884	681.5039	679.6361877
737.5349	661.2296	631.5026	701.0674	646.3661	676.7557	674.7297351
737.5349	697.2525	645.7471	727.4643	717.3937	693.5115	712.7898568
847.959	773.5958	776.3845	818.7715	774.9901	799.3131	797.6915788
847.959	773.5958	776.3845	818.7715	774.9901	799.3131	797.6915788
847.959	824.3356	821.7798	837.9985	823.0577	831.3581	830.8047822
847.959	824.3356	776.3845	828.9194	836.1473	816.2264	827.097724
847.959	824.3356	735.7449	842.0532	836.1473	802.6798	826.9601259
847.959	773.5958	735.7449	810.6436	754.6703	785.7666	783.6934896
847.959	773.5958	735.7449	810.6436	754.6703	785.7666	783.6934896
847.959	773.5958	735.7449	810.6436	754.6703	785.7666	783.6934896
847.959	773.5958	735.7449	810.6436	754.6703	785.7666	783.6934896
847.959	773.5958	735.7449	810.6436	754.6703	785.7666	783.6934896
847.959	773.5958	735.7449	810.6436	754.6703	785.7666	783.6934896
847.959	773.5958	776.3845	818.7715	774.9901	799.3131	797.6915788
847.959	773.5958	735.7449	810.6436	754.6703	785.7666	783.6934896
847.959	773.5958	755.5236	814.5993	764.5597	792.3595	790.5061551
847.959	773.5958	776.3845	818.7715	774.9901	799.3131	797.6915788
847.959	773.5958	776.3845	818.7715	774.9901	799.3131	797.6915788
847.959	773.5958	776.3845	818.7715	774.9901	799.3131	797.6915788
847.959	773.5958	776.3845	818.7715	774.9901	799.3131	797.6915788
993.9274	873.7934	862.8435	943.6838	868.3184	910.1881	907.3967912
993.9274	873.7934	862.8435	943.6838	868.3184	910.1881	907.3967912
993.9274	873.7934	862.8435	943.6838	868.3184	910.1881	907.3967912
993.9274	873.7934	921.151	955.3453	897.4722	929.6239	927.4804678
993.9274	873.7934	862.8435	943.6838	868.3184	910.1881	907.3967912
993.9274	873.7934	862.8435	943.6838	868.3184	910.1881	907.3967912
993.9274	940.8627	811.5827	980.6612	967.3951	915.4576	954.504645
993.9274	873.7934	811.5827	963.8939	842.688	893.1012	899.8943597
993.9274	873.7934	862.8435	943.6838	868.3184	910.1881	907.3967912
993.9274	940.8627	811.5827	980.6612	967.3951	915.4576	954.504645
993.9274	940.8627	862.8435	980.6612	967.3951	932.5446	960.2002948
993.9274	873.7934	862.8435	943.6838	868.3184	910.1881	907.3967912
993.9274	940.8627	836.4212	980.6612	967.3951	923.7371	957.2644848
993.9274	940.8627	862.8435	980.6612	967.3951	932.5446	960.2002948

1086.293	936.5853	983.1715	1035.727	959.8784	1002.017	999.2074861
1086.293	1015.204	1061.337	1067.084	1073.815	1054.278	1065.059038
1086.293	1015.204	1061.337	1067.084	1073.815	1054.278	1065.059038
1086.293	1015.204	916.057	1068.521	1050.748	1005.851	1041.70691
1086.293	1015.204	983.1715	1051.451	999.1875	1028.223	1026.287122
1086.293	1015.204	983.1715	1051.451	999.1875	1028.223	1026.287122
1086.293	1015.204	983.1715	1051.451	999.1875	1028.223	1026.287122
1086.293	1015.204	916.057	1068.521	1050.748	1005.851	1041.70691
1086.293	1015.204	916.057	1068.521	1050.748	1005.851	1041.70691
1086.293	1015.204	983.1715	1051.451	999.1875	1028.223	1026.287122
1086.293	1015.204	983.1715	1051.451	999.1875	1028.223	1026.287122
1086.293	1015.204	916.057	1068.521	1050.748	1005.851	1041.70691
1086.293	1015.204	916.057	1068.521	1050.748	1005.851	1041.70691
1086.293	1015.204	916.057	1068.521	1050.748	1005.851	1041.70691
1086.293	1015.204	916.057	1068.521	1050.748	1005.851	1041.70691
1086.293	1015.204	916.057	1068.521	1050.748	1005.851	1041.70691
1086.293	1015.204	983.1715	1051.451	999.1875	1028.223	1026.287122
1086.293	1015.204	916.057	1068.521	1050.748	1005.851	1041.70691
1086.293	1015.204	916.057	1068.521	1050.748	1005.851	1041.70691
1086.293	1015.204	983.1715	1051.451	999.1875	1028.223	1026.287122
1086.293	1015.204	916.057	1068.521	1050.748	1005.851	1041.70691
1086.293	1015.204	983.1715	1051.451	999.1875	1028.223	1026.287122
1086.293	1015.204	983.1715	1051.451	999.1875	1028.223	1026.287122
1086.293	1015.204	983.1715	1051.451	999.1875	1028.223	1026.287122
1086.293	1015.204	916.057	1068.521	1050.748	1005.851	1041.70691
1086.293	1015.204	983.1715	1051.451	999.1875	1028.223	1026.287122
1086.293	1015.204	983.1715	1051.451	999.1875	1028.223	1026.287122
1086.293	1015.204	916.057	1068.521	1050.748	1005.851	1041.70691
1086.293	1015.204	983.1715	1051.451	999.1875	1028.223	1026.287122
1086.293	1015.204	983.1715	1051.451	999.1875	1028.223	1026.287122
1086.293	1015.204	1153.726	1103.151	1120.009	1085.074	1102.745043
1086.293	1222.316	1264.844	1149.208	1243.58	1191.151	1194.646489
1086.293	1222.316	1401.318	1120.299	1154.305	1236.643	1170.415405
1086.293	1222.316	1264.844	1149.208	1243.58	1191.151	1194.646489
1086.293	1108.779	1264.844	1091.915	1097.536	1153.306	1114.252093
1086.293	1222.316	1264.844	1149.208	1243.58	1191.151	1194.646489
1086.293	1222.316	1401.318	1120.299	1154.305	1236.643	1170.415405
1086.293	1222.316	1061.337	1080.054	1073.815	1123.315	1092.394895
1086.293	1222.316	1061.337	1080.054	1073.815	1123.315	1092.394895
1086.293	1222.316	1061.337	1080.054	1073.815	1123.315	1092.394895
1086.293	1108.779	1061.337	1085.799	1097.536	1085.47	1089.601607
1086.293	1108.779	1061.337	1085.799	1097.536	1085.47	1089.601607
1086.293	1222.316	1061.337	1080.054	1073.815	1123.315	1092.394895
1086.293	1222.316	1061.337	1080.054	1073.815	1123.315	1092.394895
1086.293	1222.316	1061.337	1080.054	1073.815	1123.315	1092.394895
1086.293	1108.779	1061.337	1085.799	1097.536	1085.47	1089.601607
1086.293	1222.316	1061.337	1080.054	1073.815	1123.315	1092.394895
1086.293	1222.316	1061.337	1080.054	1073.815	1123.315	1092.394895
1086.293	1222.316	1061.337	1080.054	1073.815	1123.315	1092.394895
1086.293	1222.316	1061.337	1080.054	1073.815	1123.315	1092.394895

1086.293	1222.316	1061.337	1080.054	1073.815	1123.315	1092.394895
1086.293	1222.316	1061.337	1080.054	1073.815	1123.315	1092.394895
1086.293	1108.779	1061.337	1085.799	1097.536	1085.47	1089.601607
1331.307	1213.927	1391.461	1346.345	1361.384	1312.232	1339.987003
1331.307	1213.927	1391.461	1346.345	1361.384	1312.232	1339.987003
1331.307	1213.927	1391.461	1346.345	1361.384	1312.232	1339.987003
1331.307	1213.927	1254.886	1292.547	1234.407	1266.707	1264.553285
1331.307	1213.927	1254.886	1292.547	1234.407	1266.707	1264.553285
1331.307	1355.041	1196.582	1337.24	1343.174	1294.31	1324.90816
1331.307	1213.927	1254.886	1292.547	1234.407	1266.707	1264.553285
1331.307	1213.927	1254.886	1292.547	1234.407	1266.707	1264.553285
1331.307	1355.041	1254.886	1337.24	1343.174	1313.745	1331.386359
1331.307	1213.927	1254.886	1292.547	1234.407	1266.707	1264.553285
1331.307	1213.927	1254.886	1292.547	1234.407	1266.707	1264.553285
1331.307	1213.927	1254.886	1292.547	1234.407	1266.707	1264.553285
1331.307	1213.927	1254.886	1292.547	1234.407	1266.707	1264.553285
1331.307	1355.041	1254.886	1337.24	1343.174	1313.745	1331.386359
1331.307	1213.927	1254.886	1292.547	1234.407	1266.707	1264.553285
1331.307	1355.041	1254.886	1337.24	1343.174	1313.745	1331.386359
1331.307	1213.927	1254.886	1292.547	1234.407	1266.707	1264.553285
1331.307	1213.927	1143.665	1301.962	1178.796	1229.633	1236.796784
1331.307	1213.927	1143.665	1301.962	1178.796	1229.633	1236.796784
1331.307	1355.041	1254.886	1337.24	1343.174	1313.745	1331.386359

Distance Calculation, Robot Walking

Servo	Horizontal	Verical	Average1	Average2	Average3	H&V AVG
994	1019	988	998	991	1000	1003.5
994	1019	988	998	991	1000	1003.5
994	1019	988	998	991	1000	1003.5
994	1019	988	998	991	1000	1003.5
994	1019	988	998	991	1000	1003.5
994	1019	988	998	991	1000	1003.5
994	1019	988	998	991	1000	1003.5
994	1019	988	998	991	1000	1003.5
994	1019	988	998	991	1000	1003.5
994	1019	988	998	991	1000	1003.5
994	1019	988	998	991	1000	1003.5
994	1019	988	998	991	1000	1003.5
942	1024	993	969	1008	986	1008.5
617	1047	1021	784	1034	895	1034
617	970	1021	705	996	869	995.5
617	1047	1021	784	1034	895	1034
617	970	1021	705	996	869	995.5
617	1047	987	710	1017	884	1017
1030	1020	1066	1035	1025	1039	1043
1134	1109	1060	1114	1122	1101	1084.5
656	1044	950	730	997	883	997
1420	1007	905	1317	956	1111	956
1134	937	948	1057	942	1006	942.5
868	950	963	903	956	927	956.5
748	1036	941	796	988	908	988.5
868	950	931	897	940	916	940.5
868	1028	931	884	900	942	979.5
868	950	931	897	940	916	940.5
942	1024	993	969	1008	986	1008.5
522	1058	1034	732	1046	871	1046
1030	941	863	1008	902	945	902
942	946	868	943	944	919	907
617	970	955	755	962	847	962.5
748	891	883	804	887	841	887
804	954	878	822	841	879	916
868	950	901	891	884	906	925.5
942	879	868	915	874	896	873.5
1134	870	858	1026	864	954	864
656	966	920	771	943	847	943
1420	861	847	1194	854	1043	854
1134	937	858	1085	898	976	897.5
656	899	842	702	870	799	870.5
656	899	893	752	896	816	896
748	834	883	792	858	822	858.5
748	891	883	804	887	841	887
868	883	873	872	870	875	878
1030	817	812	944	814	886	814.5

617	903	897	730	900	806	900
804	887	878	835	882	856	882.5
942	821	817	893	819	860	819
583	850	851	690	850	761	850.5
656	842	819	726	830	772	830.5
699	838	837	754	838	791	837.5
748	834	832	782	833	805	833
748	784	832	772	766	788	808
868	826	822	850	824	839	824
522	857	860	657	858	746	858.5
804	780	782	795	781	789	781
748	784	787	763	786	773	785.5
496	811	780	616	796	696	795.5
551	804	811	654	808	722	807.5
656	792	797	711	794	748	794.5
656	748	797	703	772	734	772.5
699	788	752	727	770	746	770
942	771	772	874	772	828	771.5
522	763	776	621	770	687	769.5
942	726	732	857	729	800	729
804	735	706	771	720	748	720.5
656	748	721	687	734	708	734.5
617	752	726	666	739	698	739
617	712	726	658	719	685	719
656	708	721	679	714	695	714.5
699	704	716	703	702	706	710
522	763	795	625	779	693	779
551	720	771	593	746	681	745.5
868	691	668	793	680	742	679.5
551	685	673	602	679	636	679
496	693	712	579	702	634	702.5
522	689	707	592	698	639	698
583	681	714	629	698	659	697.5
617	677	693	644	685	662	685
656	673	721	672	664	683	697
450	700	721	554	710	624	710.5
617	645	693	638	631	652	669
699	636	626	672	631	654	631
472	665	660	548	662	599	662.5
450	639	677	533	658	589	658
472	665	687	554	676	608	676
496	661	655	561	658	604	658
748	603	608	691	606	653	605.5
656	584	606	632	595	615	595
450	613	664	491	638	576	638.5
748	576	596	683	586	640	586
656	584	571	625	578	604	577.5
551	573	598	565	562	574	585.5
496	581	596	533	588	558	588.5
496	581	607	535	594	561	594

496	581	607	535	594	561	594
583	568	593	582	588	581	580.5
583	568	616	587	576	589	592
410	596	625	490	610	544	610.5
551	550	598	560	550	566	574
699	511	535	629	523	582	523
472	562	570	510	566	535	566
450	545	575	494	560	523	560
429	549	561	479	555	513	555
472	562	570	510	566	535	566
522	533	551	530	528	535	542
522	533	581	536	528	545	557
376	560	574	452	567	503	567
583	504	513	553	508	533	508.5
551	509	510	534	510	523	509.5
376	540	556	445	548	491	548
392	537	535	450	536	488	536
429	493	534	463	514	485	513.5
429	511	543	468	527	494	527
472	503	517	487	510	497	510
496	499	529	503	498	508	514
331	532	552	415	542	472	542
450	489	538	475	470	492	513.5
496	481	496	493	496	491	488.5
360	509	528	423	518	466	518.5
376	505	524	431	514	468	514.5
360	492	499	414	496	450	495.5
360	509	513	420	511	461	511
450	457	491	460	454	466	474
450	457	506	463	454	471	481.5
294	509	533	385	521	445	521
429	461	481	446	471	457	471
392	455	464	419	460	437	459.5
318	473	498	385	486	430	485.5
283	483	491	365	487	419	487
331	442	481	383	462	418	461.5
360	435	460	395	448	418	447.5
360	435	448	393	442	414	441.5
392	427	477	401	410	432	452
272	472	477	353	474	407	474.5
392	401	427	401	396	407	414
376	376	409	383	376	387	392.5
283	431	451	346	441	388	441
306	412	427	351	420	382	419.5
306	412	437	353	424	385	424.5
331	405	423	364	414	386	414
331	405	423	364	414	386	414
376	376	420	385	376	391	398
283	431	451	346	441	388	441
318	318	417	318	318	351	367.5

345	345	345	345	345	345	345
283	283	430	283	283	332	356.5
242	410	445	316	428	366	427.5
283	283	283	283	283	283	283
294	294	294	294	294	294	294
318	318	318	318	318	318	318
345	345	345	345	345	345	345
262	262	418	262	262	314	340
331	331	331	331	331	331	331
262	262	262	262	262	262	262
252	252	252	252	252	252	252
225	225	225	225	225	225	225
252	252	252	252	252	252	252
252	252	252	252	252	252	252
272	272	272	272	272	272	272
294	294	294	294	294	294	294
208	208	208	208	208	208	208
272	272	272	272	272	272	272
242	242	242	242	242	242	242
193	193	193	193	193	193	193
208	208	208	208	208	208	208
208	208	208	208	208	208	208
225	225	225	225	225	225	225
225	225	225	225	225	225	225
242	242	242	242	242	242	242
171	171	171	171	171	171	171
225	225	225	225	225	225	225
200	200	200	200	200	200	200
193	193	193	193	193	193	193
151	151	151	151	151	151	151
178	178	178	178	178	178	178
178	178	178	178	178	178	178
200	200	200	200	200	200	200
200	200	200	200	200	200	200
151	151	151	151	151	151	151
171	171	171	171	171	171	171
158	158	158	158	158	158	158
115	115	115	115	115	115	115
121	121	121	121	121	121	121
121	121	121	121	121	121	121
139	139	139	139	139	139	139
151	151	151	151	151	151	151
171	171	171	171	171	171	171
132	132	132	132	132	132	132
115	115	115	115	115	115	115
126	126	126	126	126	126	126
103	103	103	103	103	103	103
103	103	103	103	103	103	103
109	109	109	109	109	109	109
109	109	109	109	109	109	109

109	109	109	109	109	109	109
109	109	109	109	109	109	109
109	109	109	109	109	109	109
109	109	109	109	109	109	109
109	109	109	109	109	109	109
109	109	109	109	109	109	109
109	109	109	109	109	109	109
109	109	109	109	109	109	109
109	109	109	109	109	109	109
109	109	109	109	109	109	109
109	109	109	109	109	109	109
109	109	109	109	109	109	109
109	109	109	109	109	109	109
109	109	109	109	109	109	109
109	109	109	109	109	109	109
109	109	109	109	109	109	109
109	109	109	109	109	109	109
109	109	109	109	109	109	109
109	109	109	109	109	109	109
109	109	109	109	109	109	109
109	109	109	109	109	109	109
109	109	109	109	109	109	109
109	109	109	109	109	109	109
109	109	109	109	109	109	109
109	109	109	109	109	109	109
109	109	109	109	109	109	109
109	109	109	109	109	109	109
109	109	109	109	109	109	109
109	109	109	109	109	109	109
109	109	109	109	109	109	109

Appendix C (CODE)

VB6 CODE

FORM 1

```
'--------------------------------------------------------------------------
'--------------------------------------------------------------------------

Private Declare Sub Sleep Lib "Kernel32.dll" (ByVal dwMilliseconds As Long)

Private Declare Function timeGetTime Lib "WinMM.dll" () As Long

Public Sub Wait(ByVal inMilliseconds As Long)
'Delay function
   Dim SleepTime As Long, TimeNow As Long
   Dim SleepTo As Long, SleepEnd As Long

   Const MaxSleep As Long = 100

   TimeNow = timeGetTime()
   SleepTime = inMilliseconds \ 10
   If (SleepTime > MaxSleep) Then SleepTime = MaxSleep
   SleepTo = TimeNow + inMilliseconds

   Do
      DoEvents
      TimeNow = timeGetTime()
      SleepEnd = SleepTo - TimeNow
      If (SleepEnd <= SleepTime) Then Exit Do
      Call Sleep(SleepTime)
   Loop

   If (SleepEnd > 0) Then Call Sleep(SleepEnd)
End Sub

Private Sub Command2_Click()
Form2.Show
End Sub

Private Sub Form_Load()
   ' Use COM value from text input.
   MSComm1.CommPort = 3
   MSComm2.CommPort = 1
   MSComm3.CommPort = 7
   ' Baud rate, no parity, 8 data, and 1 stop bit.
   MSComm1.Settings = "115200,N,8,1"
   MSComm2.Settings = "57600,N,8,1"
   MSComm3.Settings = "57600,N,8,1"
   ' Tell the control to read entire buffer when Input
   ' is used.
```

```
Image1.Visible = True
Image2.Visible = False
Image3.Visible = True
Image4.Visible = False

ComsConnected = False
InitCom = False
Position = 1
Counter1 = 1
Counter2 = 1
YellowGoal = False
RGB = False

BallFound = False
GoalFound = False
Shot = False
BallPosition = False
GoalCheck = False
BallCheck = False
SearchTurn = False

Option1.value = False      'Yellow
Option2.value = True       'Blue
YUV.value = True
RGB.value = False
StopProgram = False
GetPacketCount = 0
SearchCount = 0
Search2ndRun = False

Dist1 = 0
Dist2 = 0
Angle1 = 0
Angle2 = 0
pi = 3.14159265
VariableCount = 1

compass = 0
BallStationary = False

Xco = Array(0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
Yco = Array(0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
Xcompare = Array(0, 0, 0)
Ycompare = Array(0, 0, 0)
Xcompareb = Array(0, 0, 0)
Ycompareb = Array(0, 0, 0)
```

```
    Shape1.FillColor = &H8000000F
    Shape2.FillColor = &H8000000F

    MotionCommand = "s"
    MotionFinished = False
    MadeCorrection = False

    Call OpenTextFile

End Sub

Sub OpenTextFile()

Open "c:\DistCalc.csv" For Output As #1

Write #1, "Servo", "Horizontal", "Verical", "Average1", "Average2", "Average3"

Close #1

End Sub

Private Sub ConnectBoth_Click()
    Call Connect1_Click
    Call Connect2_Click
    Call Connect3_Click
    ComsConnected = True
End Sub

'This function sends commands to the camera and waits for an acknowledge.
Public Sub Camera_Out(Output)

    Ack = False
    Do While Ack = False
        MSComm1.InBufferCount = 0
        MSComm1.Output = Output & Chr(13)
        Call GotAck(1)
    Loop
End Sub

'Checks if an ACK has been received from the camera

Private Sub GotAck(com)
    Dim com1 As String
    Dim com2 As String
    Dim com3 As String
```

```
  Dim fm As String
  Ack = False

  If com = 1 Then
    Do
      DoEvents
    Loop Until MSComm1.InBufferCount > 0

    com1 = MSComm1.Input
    Call printf(Text2, com1, True)
    If InStr(com1, "ACK") > 0 Then
      Ack = True
    Else
      Ack = False
    End If
  End If

'Checks for ACK from the robot,
'places the input into a public variable
'& prints what was received in textboxLHS
  If com = 2 Then
    StartTime = Timer()
    Call Wait(50)
    EndTime = Timer()
    If MSComm2.InBufferCount > 0 Then
      com2 = MSComm2.Input
      If InStr(com2, "ACK") > 0 Then
        Ack = True
      Else
        Ack = False
      End If
      RobotIn = com2
      Text1.Text = Text1.Text & vbCrLf & RobotIn
      Text1.SelStart = Len(Text1.Text)
      Text1Count = Text1Count + 1
    End If
  End If

  If com = 3 Then
      com3 = RobotIn
      If InStr(com3, "FM") > 0 Then
        MotionFinished = True
        fm = "true"
      Else:
        MotionFinished = False
        fm = "false"
      End If
```

```
        Text1.Text = Text1.Text & vbCrLf & fm
        Text1.SelStart = Len(Text1.Text)
        Text1Count = Text1Count + 1
      End If

End Sub

' Initalisation of the camera.
' sets servo parameters, colour space and HiRes mode.
' Sets it to track orange.

Private Sub InitCam_Click()

   If InitCom = True Then
      InitCom = False
   End If

'Initalise the camera.
   Call Camera_Out("CR 18 32")
   Call Camera_Out("HR 1")
   Call Camera_Out("SP 30 10 5 30 12 3")
   Call Camera_Out("SM 15")
   Call Camera_Out("ST 240 240 20 230 16 16")
   Call Camera_Out("NF 5")

'Start robot program.
   If RobotInitalised = False Then
      Ack = False
      Do While Ack = False
         MSComm2.InBufferCount = 0
         MSComm2.Output = "m 000 000 000" & Chr(13)
         Call GotAck(2)
      Loop
      RobotInitalised = True
   End If
   InitCom = True

End Sub

Private Sub Option1_Click()
   Option2.value = False
   YellowGoal = True
End Sub

Private Sub Option2_Click()
   Option1.value = False
```

```
    YellowGoal = False
End Sub

Private Sub Reset_Click()
    Call Disconnect2_Click
    Call Disconnect1_Click
    Call Disconnect3_Click
    Call Form_Load

End Sub

'Starts the tracking of the camera and com communication.

Private Sub RunTrack_Click()

    Dim temp As String
    Dim e As String
    Dim OldServo1 As String
    Dim OldServo2 As String
    Dim temp1 As Integer
    Dim temp2 As Integer
    temp1 = 0
    temp2 = 0

    If InitCom = True Then
        InitCom = False
    End If
    Call Colour2Track(orange)
    InitCom = True
    Servo90 = Val(Text12.Text)
    ObjectHeight = Val(Text13.Text)
    ObjectSize = Val(ObjSize.Text)

    Do
        If Text1Count > 100 Then
            Text1.Text = ""
            Text1Count = 0
        End If
        If Text2Count > 100 Then
            Text2.Text = ""
            Text2Count = 0
        End If
        If Text3Count > 100 Then
            CompassBoard.Text = ""
            Text3Count = 0
        End If
'Get Reading from the compass.
```

```
      Call CompassPacket
'Get reading from the camera board,
'then make sure it's complete.
      If GetPacket = True Then
         Call DataEx(Object)
         Call Direct(1, False)

'If the robot is not turning around then
'check to see if the ball is moving.
'Make sure ball not to close to robot.
'Ball has to have moved more than 150mm
'or head must have moved more than
'10 servo positions in horizontal direction.
         If MadeCorrection = False Then
            If (MotionCommand = "w" Or MotionCommand = "a" _
            Or MotionCommand = "d" Or MotionCommand = "o") Then
               If (Abs(Xcompare(0) - Xcompare(1)) > 150) _
               Or (Abs(Ycompare(0) - Ycompare(1)) > 10) Then
                  BallStationary = False
               End If
            End If

            If (MotionCommand = "s") Then
               If (Abs(Xcompareb(0) - Xcompareb(1)) > 70) Or _
               (Abs(Ycompareb(0) - Ycompareb(1)) > 70) Then
                  BallStationary = False
               End If
            End If

            If BallStationary = False Then
               Call BallMoving
            End If

            If BallStationary = False Then
               Call CorrectedCourse
            End If

         Else:
            temp2 = temp2 + 1
            If temp2 > 10 Then
               temp2 = 0
               MadeCorrection = False
            End If
         End If

         If BallStationary = True Then
            If BallFound = True And BallPosition = False Then
```

```
            Call Position2Goal
          End If

          If BallFound = False Or Shot = True Then
            Call Get2Ball
          End If

          If BallPosition = True And Shot = False Then
            Call ShootGoal
          End If

        End If
      End If

      If StopProgram = True Then
        Exit Sub
      End If
'Error flag then reset to look for ball
      If BallErrorFlag = True Then
        Call BallError
      End If
    Loop

End Sub

'stores camera information into relevant
'field so it can be utilised later by other
'functions.
Private Sub DataEx(Object)

    If Object = 1 Then
      Ball_mx = mx
      Ball_my = my
      Ball_x1 = x1
      Ball_y1 = y1
      Ball_x2 = x2
      Ball_y2 = y2
      Ball_pixels = pixels
      Ball_conf = conf
      Ball_servo1 = servo1
      Ball_servo2 = servo2
'Print result.
      Text2.Text = Text2.Text & vbCrLf & "Copied: Ball " _
      & Ball_mx & " " & Ball_my & " " & Ball_x1 & " " & Ball_y1 _
      & " " & Ball_x2 & " " & Ball_y2 & " " & Ball_pixels & " " _
      & Ball_conf & " " & Ball_servo1 & " " & Ball_servo2
      Text2.SelStart = Len(Text2.Text)
```

```
    Text2Count = Text2Count + 1
  End If

  If Object = 2 Then
    BGoal_mx = mx
    BGoal_my = my
    BGoal_x1 = x1
    BGoal_y1 = y1
    BGoal_x2 = x2
    BGoal_y2 = y2
    BGoal_pixels = pixels
    BGoal_conf = conf
    BGoal_servo1 = servo1
    BGoal_servo2 = servo2
'Print result.
    Text2.Text = Text2.Text & vbCrLf & "Copied: Blue " _
    & BGoal_mx & " " & BGoal_my & " " & BGoal_x1 & " " _
    & BGoal_y1 & " " & BGoal_x2 & " " & BGoal_y2 & " " _
    & BGoal_pixels & " " & BGoal_conf & " " & BGoal_servo1 _
    & " " & BGoal_servo2
    Text2.SelStart = Len(Text2.Text)
    Text2Count = Text2Count + 1
  End If

  If Object = 3 Then
    YGoal_mx = mx
    YGoal_my = my
    YGoal_x1 = x1
    YGoal_y1 = y1
    YGoal_x2 = x2
    YGoal_y2 = y2
    YGoal_pixels = pixels
    YGoal_conf = conf
    YGoal_servo1 = servo1
    YGoal_servo2 = servo2
'Print results.
'   Text2.Text = Text2.Text & vbCrLf & "Copied: Yelw " _
'   & YGoal_mx & " " & YGoal_my & " " & YGoal_x1 & " " _
'   & YGoal_y1 & " " & YGoal_x2 & " " & YGoal_y2 & " " _
'   & YGoal_pixels & " " & YGoal_conf & " "_
'   & YGoal_servo1 & " " & YGoal_servo2
'   Text2.SelStart = Len(Text2.Text)
'   Text2Count = Text2Count + 1
  End If

  If Object = 4 Then
    Obsticle_mx = mx
```

```
        Obsticle_my = my
        Obsticle_x1 = x1
        Obsticle_y1 = y1
        Obsticle_x2 = x2
        Obsticle_y2 = y2
        Obsticle_pixels = pixels
        Obsticle_conf = conf
        Obsticle_servo1 = servo1
        Obsticle_servo2 = servo2
'Print results.
'    Text2.Text = Text2.Text & vbCrLf & "Copied: Objt " & _
'    Obsticle_mx & " " & Obsticle_my & " " & Obsticle_x1 & " " & Obsticle_y1 _
'    & " " & Obsticle_x2 & " " & Obsticle_y2 & " " & Obsticle_pixels _
'    & " " & Obsticle_conf & " " & Obsticle_servo1 & " " & Obsticle_servo2
'    Text2.SelStart = Len(Text2.Text)
'    Text2Count = Text2Count + 1
    End If

End Sub

'This function sends the command to the
'camera to track the required colour.
Public Sub Colour2Track(Colour As ObjectColour)

    If RGB = True Then
        Ack = False
        Do While Ack = False

            If Colour = orange Then
                Call Camera_Out("TC 220 240 16 240 16 16")
                '"TC 240 240 20 120 16 16"     'Orange
                '"TC 220 240 16 240 16 16"     'Red
            End If

            If Colour = Blue Then
                Call Camera_Out("TC 16 30 90 110 133 153")
                '"TC 16 30 90 110 133 153"     'Blue
                '"TC 71 89 30 50 153 165"      'Blue2
            End If

            If Colour = Yellow Then
                Call Camera_Out("TC 193 233 240 240 16 40")
                '"TC 193 233 240 240 16 40"    'Yellow
            End If
        Loop

    Else:
```

```
Ack = False
Do While Ack = False

  If Colour = orange Then
    Object = 1
    Call Camera_Out("NF 3")
    Call Camera_Out("TC 239 240 20 230 16 30")
    '"TC 240 240 20 230 16 16"     'Orange
    '"TC 239 240 20 230 16 30"     'Orange1
    '"TC 235 240 20 70 16 40"      'Red
  End If

  If Colour = Blue Then
    Object = 2
    Call Camera_Out("NF 12")
    Call Camera_Out("TC 16 60 50 110 120 150")
    '"TC 180 220 20 180 110 140"   'Purple
    '"TC 90 105 20 50 132 147"     'Blue
    '"TC 16 60 50 110 120 150"     'Blue
    '"TC 61 90 30 65 135 199"      'Blue2
  End If

  If Colour = Yellow Then
    Object = 3
    Call Camera_Out("NF 12")
    Call Camera_Out("TC 130 140 60 120 16 16")
    '"TC 130 140 60 120 16 16"     'Yellow1
    '"TC 174 176 20 120 16 16"     'Yellow2
    '"TC 180 190 80 100 16 16"     'Yellow3
  End If

  If Colour = Magenta Then
    Object = 4
    Call Camera_Out("NF 5")
    Call Camera_Out("TC 180 220 20 180 110 140")
    '"TC 180 220 20 180 110 140"   'Purple
  End If

  If Colour = Cyan Then
    Object = 4
    Call Camera_Out("NF 5")
    Call Camera_Out("TC 180 220 20 180 110 140")
    '"TC 180 220 20 180 110 140"   'Purple
  End If
 Loop
End If
```

```
End Sub

'Gets T packet from the camera and
'places the relevent information into
'public variables for calculations to be performed.

Private Function GetPacket()

   Dim Count As Integer
   Dim e As String
   Dim value As String
   Dim temp As Variant
   Dim SplitText As Variant
   Dim i As Integer
   Dim a As Boolean
   Dim Robot As String
   Dim Happy As Boolean

   On Error GoTo ErrHandler
   Count = 0
   Happy = False

   If InitCom = True Then

      Do
         StartTime = Timer()
         Call Wait(50)
         EndTime = Timer()
         Count = Count + 1
         If Count > 10 Then
            Happy = True
         End If

      Loop Until MSComm1.InBufferCount > 0 Or Happy = True


      e = MSComm1.Input
      temp_string = e
      MSComm1.InBufferCount = 0
      If InStr(temp_string, Chr(13)) > 0 Then
         ' Split on /r
         temp = Split(temp_string, Chr(13))
         value = temp(UBound(temp) - 1)
         ' break up info in packet
         SplitText = Split(value, " ")
         ' use first packet
         i = LBound(SplitText)
```

```
a = False

Do
   ' look for the start of a T packet
   If SplitText(i) = "T" Then
      a = True
   Else
      i = i + 1
   End If
' Place packet info into variables
Loop Until a = True

If (UBound(SplitText) - LBound(SplitText)) = 10 Then

   T = SplitText(i)
   i = i + 1
   mx = SplitText(i)
   i = i + 1
   my = SplitText(i)
   i = i + 1
   x1 = SplitText(i)
   i = i + 1
   y1 = SplitText(i)
   i = i + 1
   x2 = SplitText(i)
   i = i + 1
   y2 = SplitText(i)
   i = i + 1
   pixels = SplitText(i)
   i = i + 1
   conf = SplitText(i)
   i = i + 1
   servo1 = SplitText(i)
   i = i + 1
   servo2 = SplitText(i)
   i = i + 1

   Text2.Text = Text2.Text & vbCrLf & "Copied: " _
   & T & " " & mx & " " & my & " " & x1 & " " & y1 _
   & " " & x2 & " " & y2 & " " & pixels & " " & conf _
   & " " & servo1 & " " & servo2

   Text2.SelStart = Len(Text2.Text)
   ' carry part of second packet
   temp_string = temp(LBound(temp) + 1)
   Text2Count = Text2Count + 1
   GetPacket = True
```

```
        Else:
            Text2.Text = Text2.Text & vbCrLf & "error"
            Text2.SelStart = Len(Text2.Text)
            temp_string = temp(LBound(temp) + 1)
            Text2Count = Text2Count + 1
ErrHandler:
            GetPacket = False

        End If
      Else:
        If Happy = True Then
            Call BallError
        End If
        GetPacket = False

      End If
   End If

End Function
Private Function CompassPacket() As Boolean

   Dim Count As Integer
   Dim e As String
   Dim value As String
   Dim temp As Variant
   Dim SplitText As Variant
   Dim i As Integer
   Dim a As Boolean
   Dim Robot As String
   Dim Happy As Boolean
   Dim tempnumber As Double
   Dim length As Integer

   On Error GoTo ErrHandler5
   Count = 0
   Happy = False

   If InitCom = True Then

      Do
        StartTime = Timer()
        Call Wait(100) ' Wait
        EndTime = Timer()
        Count = Count + 1
        If Count > 10 Then
            Happy = True
```

```
        End If
        DoEvents
    Loop Until MSComm3.InBufferCount > 0 Or Happy = True


    e = MSComm3.Input
    temp_string = e
    MSComm3.InBufferCount = 0
    If InStr(temp_string, Chr(13)) > 0 Then
        ' Split on /r
        temp = Split(temp_string, Chr(13))
        ' use last full packet (most recent)
        value = temp(UBound(temp) - 1)
        ' break up info in packet
        SplitText = Split(value, "~")
        a = False

        If UBound(SplitText) = 5 Then

            i = 4

        End If

        ' Place packet info into variables
        If (UBound(SplitText) - LBound(SplitText)) > 1 Then
            compass = Val(SplitText(i))
            compass = compass / 10
            CompassReading.Text = Str(compass)
            CompassPacket = True

        Else:
            CompassBoard.Text = CompassBoard.Text & vbCrLf & "error"
            CompassBoard.SelStart = Len(CompassBoard.Text)
            temp_string = temp(LBound(temp) + 1)
            Text3Count = Text3Count + 1
ErrHandler5:
            CompassPacket = False

        End If
    End If
  End If

End Function

'This function organises the information
'for the robot into the appropriate
'format and sends it via serial to the robot.
```

```
Private Sub SendRobotInstructions()

   Robot = Str(compass) & Str(servo1) & Str(servo2)
   Robot = Format(Robot, " ###000 ###000 ###000")
   Ack = False
   Do While Ack = False
      MSComm2.InBufferCount = 0
      MSComm2.Output = MotionCommand & Robot & Chr(13)
      Call GotAck(2)
   Loop
End Sub


Private Sub Form_Unload(Cancel As Integer)
   End
End Sub


Private Sub Text3_Change()

   If Text3.Text = "" Then Exit Sub
   Call printf(Text1, Text3.Text, True)
   If MSComm2.PortOpen = True Then
      MSComm2.Output = Text3.Text
   Else
      Call printf(Text1, "ERROR: No connection present", True)
   End If
   Text3.Text = ""
End Sub

Private Sub Text4_Change()

   If Text4.Text = "" Then Exit Sub
   Call printf(Text2, Text4.Text, True)
   End If
   If MSComm1.PortOpen = True Then
      MSComm1.Output = Text4.Text
   Else
      Call printf(Text2, "ERROR: No connection present", True)
   End If
   Text4.Text = ""
End Sub

Private Sub Connect1_Click()

   ' Open the port.
   Dim x As Integer
```

```
    Dim temp1 As String
    Dim temp2 As String
    x = 1
    ' Enable error-handling routine.
    On Error GoTo ErrorHandler1

    Call printf(Text1, "Connecting...", True)

    If Not (MSComm2.CommPort = MSComm1.CommPort) Then
       If MSComm2.PortOpen = False Then MSComm2.PortOpen = True
       Call printf(Text1, "connected", True)
       Image1.Visible = False
       Image2.Visible = True
    Else
       Call printf(Text1, "Comm Port" & "'" & Val(MSComm2.CommPort) _
       & "'" & "is aready in use.", True)
    End If
xit1:
'Turn off error trapping.
    On Error GoTo 0
    'Defer error trapping.
    'object, then test for
    'Check for likely Automation errors.
    On Error Resume Next

    If Err.Number = 440 Or Err.Number = 432 Then
       ' Tell user what happened. Then clear the Err object.
       Msg = "There was an error attempting to open the Automation object!"
       MsgBox Msg, , "Deferred Error Test"
       Err.Clear   ' Clear Err object fields
    End If
Exit Sub     'Exit to avoid handler.

ErrorHandler1:  'Error-handling routine.
    'Evaluate error number.
    Select Case Err.Number
       Case 68   '"File already open" error.
          x = 0

       Case Else
       ' Handle other situations here...
    End Select
    Call printf(Text1, "Comm Port " _
    & MSComm2.CommPort & " does not exist!", True)
    GoTo xit1  ' Resume execution at same line
          ' that caused the error.
End Sub
```

```
Private Sub Connect2_Click()

   ' Open the port
   Dim x As Integer
   Dim temp1 As String
   Dim temp2 As String
   x = 1
   ' Enable error-handling routine.
   On Error GoTo ErrorHandler2

   Call printf(Text2, "Connecting...", True)

   If Not (MSComm1.CommPort = MSComm2.CommPort) Then
      If MSComm1.PortOpen = False Then MSComm1.PortOpen = True
         Call printf(Text2, "connected", True)
         Image3.Visible = False
         Image4.Visible = True
   Else
      Call printf(Text2, "Comm Port" & "" & Val(MSComm1.CommPort) _
      & "" & "is aready in use.", True)
   End If
xit2:
   On Error GoTo 0   ' Turn off error trapping.
   On Error Resume Next   ' Defer error trapping.
         ' object, then test for
         'Check for likely Automation errors.
   If Err.Number = 440 Or Err.Number = 432 Then
      ' Tell user what happened. Then clear the Err object.
      Msg = "There was an error attempting to open the Automation object!"
      MsgBox Msg, , "Deferred Error Test"
      Err.Clear   ' Clear Err object fields
   End If
Exit Sub      ' Exit to avoid handler.

ErrorHandler2:   ' Error-handling routine.
   Select Case Err.Number   ' Evaluate error number.
      Case 68   ' "File already open" error.
         x = 0

      Case Else
      ' Handle other situations here...
   End Select
   Call printf(Text2, "Comm Port " & Str(MSComm1.CommPort) _
   & " does not exist!", True)
   GoTo xit2   ' Resume execution at same line
         ' that caused the error.
```

```
End Sub
Private Sub Connect3_Click()
   ' Open the port.
   Dim x As Integer
   Dim temp1 As String
   Dim temp2 As String
   x = 1
   ' Enable error-handling routine.
   On Error GoTo ErrorHandler3

   Call printf(CompassBoard, "Connecting...", True)

   If Not (MSComm3.CommPort = MSComm2.CommPort) _
   Or (MSComm3.CommPort = MSComm1.CommPort) Then

      If MSComm3.PortOpen = False Then MSComm3.PortOpen = True
      Call printf(CompassBoard, "connected", True)
      Image5.Visible = False
      Image6.Visible = True
   Else

      Call printf(CompassBoard, "Comm Port" & "'" & _
      Val(MSComm3.CommPort) & "'" & "is aready in use.", True)

   End If
xit1:
   On Error GoTo 0   ' Turn off error trapping.
   On Error Resume Next   ' Defer error trapping.
         ' object, then test for
         'Check for likely Automation errors.
   If Err.Number = 440 Or Err.Number = 432 Then
      ' Tell user what happened. Then clear the Err object.
      Msg = "There was an error attempting to open the Automation object!"
      MsgBox Msg, , "Deferred Error Test"
      Err.Clear   ' Clear Err object fields
   End If
Exit Sub     ' Exit to avoid handler.

ErrorHandler3:   ' Error-handling routine.
   Select Case Err.Number   ' Evaluate error number.
      Case 68   ' "File already open" error.
         x = 0

      Case Else
      ' Handle other situations here...
   End Select
```

```
    Call printf(CompassBoard, "Comm Port " & MSComm3.CommPort _
    & " does not exist!", True)
    GoTo xit1  ' Resume execution at same line
          ' that caused the error.
End Sub

Private Sub Disconnect3_Click()
    Dim temp1 As String
    Dim temp2 As String
    If MSComm3.PortOpen = True Then MSComm3.PortOpen = False
    Call printf(CompassBoard, "Disconnected", True)
    Image5.Visible = True
    Image6.Visible = False
End Sub

Private Sub Disconnect1_Click()

    Dim temp1 As String
    Dim temp2 As String
    If MSComm2.PortOpen = True Then MSComm2.PortOpen = False
    Call printf(Text1, "Disconnected", True)
    Image1.Visible = True
    Image2.Visible = False
End Sub

Private Sub Disconnect2_Click()

    Dim temp1 As String
    Dim temp2 As String
    If MSComm1.PortOpen = True Then MSComm1.PortOpen = False
    Call printf(Text2, "Disconnected", True)
    Image3.Visible = True
    Image4.Visible = False
End Sub

Public Sub printf(txtbox As Object, s As String, newline As Boolean)

    Dim tmp1 As String
    Dim tmp2 As String
    tmp1 = txtbox.Text
    tmp2 = s
    If newline = True Then
       txtbox.Text = tmp1 & Chr(13) & Chr(10) & tmp2
    Else
       txtbox.Text = tmp1 & tmp2
    End If
    txtbox.SelStart = Len(txtbox.Text)
```

```
End Sub

Public Sub printf1(txtbox As Object, s As String)

   txtbox.Text = s
End Sub

Private Sub YUV_Click()
   RGB.value = False
   RGB = False
End Sub

Private Sub RGB_Click()
   YUV.value = False
   RGB = True
End Sub

Private Sub Stop_Click()
   StopProgram = True
End Sub

'------------------------------------------------------------------------
'------------------------------------------------------------------------
'------------------------------------------------------------------------
'------------------------------------------------------------------------
'This function allows the distance from
'the ball to be calculated using the objects
'horizontal and vertical image size along
'with the distance calculated from the
'angle of the cameras vertical servo.
'
'These values are then averaged so to give
'a more stable distance that can be used
'for subsequent calculations

Public Sub Distance()

   Dim DpH As Double
   Dim DpV As Double
   Dim OpH As Double
   Dim OpV As Double
   Dim ResH As Double
   Dim ResV As Double
   Dim ViewAngleH As Double
   Dim ViewAngleV As Double
   Dim FOVdH As Double
   Dim FOVdV As Double
```

```
    Dim DistanceCamH As Double
    Dim DistanceCamV As Double
    Dim HeightRobot As Double
    Dim DistanceH As Long
    Dim DistanceV As Long
    Dim DistanceObj As Long
    Dim AngleV As Double
    Dim ServoCurrentV As Integer
    Dim ServoCurrentH As Integer
    Dim CameraAngle As Double
    Dim FixZ As Double
    Dim FixY As Double
    Dim FixE As Double
    Dim temp1 As Double
    Dim temp2 As Double
    Dim temp3 As Double
    Dim XA As Integer
    Dim XB As Integer
    Dim YA As Integer
    Dim YB As Integer
    Dim ObjectImageHeight As Double

'find the distance from the object in question
    Select Case Object
        Case 1
            ServoCurrentH = Ball_servo1
            ServoCurrentV = Ball_servo2
            XA = Ball_x1
            XB = Ball_x2
            YA = Ball_y1
            YB = Ball_y2

        Case 2
            ServoCurrentH = BGoal_servo1
            ServoCurrentV = BGoal_servo2
            XA = BGoal_x1
            XB = BGoal_x2
            YA = BGoal_y1
            YB = BGoal_y2

        Case 3
            ServoCurrentH = YGoal_servo1
            ServoCurrentV = YGoal_servo2
            XA = YGoal_x1
            XB = YGoal_x2
            YA = YGoal_y1
            YB = YGoal_y2
```

```
        Case 4
            ServoCurrentH = Obsticle_servo1
            ServoCurrentV = Obsticle_servo2
            XA = Obsticle_x1
            XB = Obsticle_x2
            YA = Obsticle_y1
            YB = Obsticle_y2

    End Select

    ResH = 160
    ResV = 255
    ViewAngleH = 44 * pi / 180
    ViewAngleV = 33 * pi / 180
    AngleV = (1.345 * (Abs(Servo90 - ServoCurrentV))) * pi / 180
    FixY = (50 * Cos(AngleV))
    FixZ = (50 * Sin(AngleV))
    FixE = ((ObjectHeight / 2) * Sin(AngleV))
    If Not MotionCommand = "s" Then
        HeightRobot = 300 - FixZ
        Servo90 = (Val(Text12.Text)) + 6
    Else:
        HeightRobot = 315 - FixZ
        Servo90 = Val(Text12.Text)
    End If
    ObjectImageHeight = (ObjectHeight / 2) + ((ObjectHeight / 2) * Sin(AngleV))


    robotballratio = Abs(HeightRobot - ObjectImageHeight)


'servo
    If Not AngleV = 0 Then
        DistanceObj = (robotballratio / Tan(AngleV)) + FixY + 60
        Distservo.Text = Str(DistanceObj)
    End If
'horizontal
    OpH = XB - XA
    If Not OpH = 0 Then
        DpH = ObjectSize / OpH
        FOVdH = ResH * DpH
        DistanceCamH = (FOVdH / 2) / (Tan(ViewAngleH / 2))
        If robotballratio < DistanceCamH Then

            DistanceH = (Sqr((DistanceCamH * DistanceCamH) _
            - (robotballratio * robotballratio))) + FixY + (5 * FixE) + 60
```

```
        DistH.Text = Str(DistanceH)
      End If
    End If

    If DistanceH < 400 Then
      DistanceH = DistanceObj
    End If

'vertical
    OpV = YB - YA
    If Not OpV = 0 Then
      DpV = ObjectSize / OpV
      FOVdV = ResV * DpV
      DistanceCamV = (FOVdV / 2) / (Tan(ViewAngleV / 2))
      If robotballratio < DistanceCamV Then

        DistanceV = (Sqr((DistanceCamV * DistanceCamV) _
        - (robotballratio * robotballratio))) + FixY + (7 * FixE) + 60

        DistV.Text = Str(DistanceV)
      End If
    End If

    If DistanceV < 400 Then
      DistanceV = DistanceObj
    End If

    temp1 = Abs(DistanceObj - DistanceH)
    temp2 = Abs(DistanceH - DistanceV)
    temp3 = Abs(DistanceV - DistanceObj)
'--------------------------------------------------------------------------
If (MotionCommand = "s" Or MotionCommand = "o") Then
'3*S & (V or H or (V & H))
    If temp2 < 50 Then
      AvgDistance = (3 * DistanceObj + DistanceV + DistanceH) / 5
    Else:
      If temp1 > temp3 Then
        AvgDistance = (3 * DistanceObj + DistanceV) / 4
      Else:
        AvgDistance = (3 * DistanceObj + DistanceH) / 4
      End If
    End If

Else:
    AvgDistance = (DistanceV + DistanceH) / 2
End If
```

```
    Avg.Text = Str(AvgDistance)
    Text11.Text = Str(AvgDistance)
'-------------------------------------------------------------------------
'test for (s&h) or (s&v) or (h&v)

    If temp1 > temp2 Then
        If temp2 > temp3 Then
            AvgDistance1 = (DistanceV + DistanceObj) / 2
        Else:
            AvgDistance1 = (DistanceH + DistanceV) / 2
        End If
    Else:
        If temp1 > temp3 Then
            AvgDistance1 = (DistanceObj + DistanceV) / 2
        Else:
            AvgDistance1 = (DistanceObj + DistanceH) / 2
        End If
    End If
        Text9.Text = Str(AvgDistance1)
'-------------------------------------------------------------------------
'Avg of all 3 values
    AvgDistance2 = (DistanceH + DistanceV + DistanceObj) / 3
    Text10.Text = Str(AvgDistance2)

'-------------------------------------------------------------------------
'write distance values to excel file for graphing
    'Open "c:\DistCalc.csv" For Append As #2
    'Write #2, DistanceObj, DistanceH, DistanceV, _
    'AvgDistance, AvgDistance1, AvgDistance2
    'Close #2

End Sub

'works out the x and y co-ordinates
'from the distance of the object
'and the angle of the camera.

Private Function XYco() As Vector2
    Dim Distance As Single
    Dim ServoAngle As Double
    ServoAngle = (128 - Ball_servo1) * 1.34 * pi / 180
    Distance = AvgDistance
    XYco.x = Distance * Sin(ServoAngle)
    XYco.y = Distance * Cos(ServoAngle)
End Function

'This function works out the direction that
```

```
'the ball is traveling in relation to the
'robot, and the distance travelled over that period.
'Y = MX + B

Private Sub Direct(N As Integer, GetTraj As Boolean)

    Dim sumXY As Double
    Dim sumX As Double
    Dim sumY As Double
    Dim sumX2 As Double
    Dim M As Double
    Dim b As Double
    Dim d As Single
    Dim e As Single
    Dim plotx As Double, ploty As Double
    Dim BallLocation As Vector2
    Dim tempcount As Integer
    tempcount = 0
    VariableCount = 1

    Do While tempcount <= N
        StartTime = Timer()
        Call Wait(100) ' Wait
        EndTime = Timer()
        If GetPacket = True Then
        Call DataEx(Object)
        Call Distance

        BallLocation = XYco


        d = BallLocation.x
        e = BallLocation.y

        plotx = 300 + d * (300 / 1500)
        ploty = 300 + e * (300 / 1500)

        Form2.setpoint plotx, ploty

        If GetTraj = False Then
            Xcompare(0) = Xcompare(1)
            Xcompare(1) = AvgDistance
            Ycompare(0) = Ycompare(1)
            Ycompare(1) = Ball_servo1

            Xcompareb(0) = Xcompareb(1)
            Xcompareb(1) = d
```

```
    Ycompareb(0) = Ycompareb(1)
    Ycompareb(1) = e

    tempcount = tempcount + 1
End If

If GetTraj = True Then

'save variables into array
    If VariableCount <= N Then
        Xco(VariableCount) = d
        Yco(VariableCount) = e
        VariableCount = VariableCount + 1
        'Exit Sub
    Else:

        VariableCount = 1

    'line of best fit (Regression)
        sumX = Xco(0) + Xco(1) + Xco(2) + Xco(3) + Xco(4) _
        + Xco(5) + Xco(6) + Xco(6) + Xco(7) + Xco(8) + Xco(9)

        sumY = Yco(0) + Yco(1) + Yco(2) + Yco(3) + Yco(4) _
        + Yco(5) + Yco(6) + Yco(6) + Yco(7) + Yco(8) + Yco(9)

        sumXY = (Xco(0) * Yco(0)) + (Xco(1) * Yco(1)) + (Xco(2) * Yco(2)) _
        + (Xco(3) * Yco(3)) + (Xco(4) * Yco(4)) + (Xco(5) * Yco(5)) _
        + (Xco(6) * Yco(6)) + (Xco(7) * Yco(7)) + (Xco(8) * Yco(8)) _
        + (Xco(9) * Yco(9))

        sumX2 = (Xco(0) * Xco(0)) + (Xco(1) * Xco(1)) + (Xco(2) * Xco(2)) _
        + (Xco(3) * Xco(3)) + (Xco(4) * Xco(4)) + (Xco(5) * Xco(5)) _
        + (Xco(6) * Xco(6)) + (Xco(7) * Xco(7)) + (Xco(8) * Xco(8)) _
        + (Xco(9) * Xco(9))

        If Not sumX = 0 Then
            M = ((N * sumXY) - (sumX * sumY)) / ((N * sumX2) - (sumX * sumX))
        Else:
            M = 0
        End If
        If M = 0 Then
            b = 0
        Else:
            b = (sumY - (M * sumX)) / N
        End If
        Yintercept = b
        If M = 0 Then
```

```
            TSlope = 90
          Else:
            TSlope = Atn(M) * 180 / pi
          End If
          If Not b = 0 And Not M = 0 Then
            Xintercept = (0 - b) / M
          Else:
            Xintercept = 0
          End If
          If Yco(N) < Yco(1) Then
            OnComing = True
            Shape1.FillColor = &HFF00&
          ElseIf Yco(N) > Yco(1) Then
            OnComing = False
            Shape1.FillColor = &HFF&
          Else:
            Shape1.FillColor = &H8000000F
          End If

          If Xco(N) > Xco(1) Then
            TravelLeft = True
            Shape2.FillColor = &HFF00&
          End If
          If Xco(N) < Xco(1) Then
            TravelLeft = False
            Shape2.FillColor = &HFF&
          End If
'Checks to see what direction ball is headed.
'Print answer.
          Slope.Text = Str(TSlope)
          Yint.Text = Str(Yintercept)
          Xint.Text = Str(Xintercept)
'Write trajectory values to excel file for graphing
          'Open "c:\TrajCalc.csv" For Append As #2
          'Write #2, TSlope, Yintercept, Xintercept, OnComing, _
          'TravelLeft, Xco(1), Xco(2), Xco(3), Xco(4), Xco(5), _
          'Xco(N), Yco(1), Yco(2), Yco(3), Yco(4), Yco(5), Yco(N)
          'Close #2
        End If
      tempcount = tempcount + 1
      End If
      End If
  Loop

'check to see if ball is stationary
  If (GetTraj = True) Then
```

```
    If (((Abs(Xco(1) - Xco(N - 1)) > 80) Or (Abs(Xco(2) - Xco(N)) > 80)) _
    Or ((Abs(Yco(1) - Yco(N - 1)) > 90) Or (Abs(Yco(2) - Yco(N)) > 90))) Then

    BallStationary = False
    Shape1.FillColor = &H80000008
    Shape2.FillColor = &H80000008
    End If
  Else:
    BallStationary = True

  End If

  compassOld = compass

End Sub

'If ball moving and want to track ball,
'stop robot and get trajectory

Private Sub BallMoving()

  MotionCommand = "s"
  SendRobotInstructions
  StartTime = Timer()
  Call Wait(300) ' Wait
  EndTime = Timer()
  Call Direct(4, True)


End Sub

'-------------------------------------------------------------------
'-------------------------------------------------------------------
'-------------------------------------------------------------------
'-------------------------------------------------------------------
'This function is called if the ball is in motion
'and the trajectory has been calculated. then tries
'to intercept the ball or chase after it is at the
'required angle to its trajectories slope/path.

Private Sub CorrectedCourse()

  Dim StartTime As Double
  Dim EndTime As Double
  Dim temp As Integer, value As Integer
  Dim WalkCount As Integer
  Dim temp2 As Integer, temptime As Integer
```

```
    Dim incr As Integer, FixCompass As Integer
    incr = 0
    WalkCount = 0
    temp2 = 0
    temptime = 0
    value = 0

    If Xintercept = 0 And Yintercept = 0 And OnComing = True Then
       Call Direct(4, True)
    End If

'Intercept ball if possible
    If AvgDistance > 200 And Xintercept < 300 _
    And Xintercept < Yintercept And OnComing = True Then
'If ball heading left, sidestep left
       temp = 2
       If TravelLeft = True Then
          Do
'Strafe left
             MotionCommand = "a"
             Position = 1
             Counter1 = 0
             Call SendRobotInstructions
             StartTime = Timer()
             Call Wait(100)
             EndTime = Timer()
             Call GotAck(3)
             If MotionFinished = True Then
                incr = incr + 1
             End If
             Call GetPacket
             If servo1 > 142 Or servo1 < 118 Then GoTo MustTurn
          Loop Until incr > temp

       End If
'If ball heading right, sidestep right
       If TravelLeft = False Then
          Do
'Strafe right
             MotionCommand = "d"
             Position = 1
             Counter1 = 0
             Call SendRobotInstructions
             StartTime = Timer()
             Call Wait(100)
             EndTime = Timer()
             Call GotAck(3)
```

```
            If MotionFinished = True Then
               incr = incr + 1
            End If
            Call GetPacket
            If servo1 > 142 Or servo1 < 118 Then GoTo MustTurn
         Loop Until incr > temp

      End If
   End If

   If Yintercept < 300 And Xintercept > Yintercept And OnComing = True Then
      temp = 3
      Do
         MotionCommand = "w"
         Position = 1
         Counter1 = 0
         Call GotAck(3)
         Call SendRobotInstructions
         StartTime = Timer()
         Call Wait(100)
         EndTime = Timer()
         If MotionFinished = True Then
            incr = incr + 1
         End If
         Call GetPacket
         If servo1 > 142 Or servo1 < 118 Then GoTo MustTurn
      Loop Until incr > temp
   End If

'If can't intercept, turn in balls direction.
   If Xintercept >= 300 And Yintercept >= 300 _
   And OnComing = True Then
MustTurn:
      If TSlope >= 45 Then  'Heading left.
         temp = 10 + (TSlope / 2)

         Do
'Turn left.
            MotionCommand = "r"
            Position = 1
            Counter1 = 0
            Call SendRobotInstructions
            StartTime = Timer()
            Call Wait(100)
            EndTime = Timer()
            Call GotAck(3)
            If MotionFinished = True Then
```

```
            value = value + 10
          End If
          Call GetPacket
          If servo1 < 110 Then GoTo StopTurn
        Loop Until value > temp
      End If

      If TSlope <= -45 Then  'Heading right.
        temp = 10 + (Abs(TSlope / 2))

        Do
'Turn right.
          MotionCommand = "l"
          Position = 1
          Counter1 = 0
          Call SendRobotInstructions
          Call GotAck(3)
          If MotionFinished = True Then
            value = value + 10
          End If

          Call GetPacket
          If servo1 > 150 Then GoTo StopTurn
        Loop Until value > temp
      End If

    End If


StopTurn:
      BallStationary = True
      MadeCorrection = True

End Sub
' If something goes wrong then reset the routine.
Private Sub BallError()
    BallFound = False
    GoalFound = False
    Shot = False
    BallPosition = False
    GoalCheck = False
    BallCheck = False
    BallErrorFlag = False
    Call Colour2Track(orange)
End Sub

'This function lets the robot find the ball, walk up to it
```

```
'and stop when the robot gets close to the ball.
Private Sub Get2Ball()

   Dim StartTime As Double
   Dim EndTime As Double


   If Shot = True Then
      MotionCommand = "o"
      Call SendRobotInstructions
      StartTime = Timer()
      Call Wait(5000) ' Wait for 1/2 seconds
      EndTime = Timer()
      BallErrorFlag = True
      Call Search(orange)
'shot has been taken so ball must be checked for movement
      BallStationary = False
      Exit Sub
   End If

'ball is at the robots foot, set BallFound flag to true
'& call search for goal
   If (pixels > 60) And (conf > 20) And (servo2 <= 106) Then
         MotionCommand = "s"
         BallFound = True
         Position = 1
         Counter1 = 0
         If YellowGoal = False Then
            Call Search(Blue)
         Else:
            Call Search(Yellow)
         End If
         Call SendRobotInstructions
      StartTime = Timer()
      Call Wait(100) ' Wait
      EndTime = Timer()
      Exit Sub

   End If

'ball is not at robot's feet and is to the
'robot's right, tell robot to turn right
   If (conf > 10) And (servo2 > 106) And (servo1 > 142) Then
         MotionCommand = "r"
         Position = 1
         Counter1 = 0
         Call SendRobotInstructions
```

```
      StartTime = Timer()
      Call Wait(100) ' Wait
      EndTime = Timer()
      Exit Sub
   End If

'ball not at robot's feet and to the left
'of the robot, tells robot to turn left.
   If (conf > 10) And (servo2 > 106) And (servo1 < 118) Then
         MotionCommand = "l"
         Position = 1
         Counter1 = 0
      Call SendRobotInstructions
      StartTime = Timer()
      Call Wait(100)
      EndTime = Timer()
      Exit Sub
   End If

'ball not at robot's feet and is straight ahead,
'tells robot to walk ahead.
   If (conf > 10) And (servo2 > 106) And (servo1 >= 118) And (servo1 <= 142) Then
         MotionCommand = "w"
         Position = 1
         Counter1 = 0
      Call SendRobotInstructions
      StartTime = Timer()
      Call Wait(100)
      EndTime = Timer()
      Exit Sub
   End If

'ball not located, search for ball.
   If (pixels < 1) And (conf < 10) Then
         Counter1 = Counter1 + 1
         If Counter1 > 3 Then
            MotionCommand = "s"
            Call Search(orange)
            Counter1 = 0
         End If
      Call SendRobotInstructions
      StartTime = Timer()
      Call Wait(100) ' Wait
      EndTime = Timer()
      Exit Sub

   End If
```

```
End Sub

'This function allows the robot to search for the goal,
'reposition itself around the ball,
'and finally kick the ball at the goal.

Private Sub Position2Goal()

   Dim StartTime As Double
   Dim EndTime As Double

    If Shot = True Then
      MotionCommand = "o"
      Call SendRobotInstructions
      StartTime = Timer()
      Call Wait(5000) ' Wait for 1/2 seconds
      EndTime = Timer()
      BallErrorFlag = True
      Call Search(orange)
      Exit Sub
   End If

   If GoalFound = False Or BallCheck = True Then

'can't see goal and ball is not in position, call search for goal
      If (pixels < 3) And (conf < 5) Then
         Counter2 = Counter2 + 1
         If Counter2 = 5 Then
            MotionCommand = "s"
            Counter2 = 0
            If YellowGoal = False Then
               Call Search(Blue)
            Else:
               Call Search(Yellow)
            End If
         End If
         Call SendRobotInstructions
         StartTime = Timer()
         Call Wait(100) ' Wait
         EndTime = Timer()
         Exit Sub
      End If

'goal to robots left, tells robot to turn left
      If (pixels >= 5) And (conf >= 5) And (servo1 < 120) Then
         MotionCommand = "l"
```

```
        Direction = 130 - servo1
        Position = 1
        Counter2 = 0
        Call SendRobotInstructions
        StartTime = Timer()
        Call Wait(100) ' Wait
        EndTime = Timer()
        Exit Sub
      End If

'Goal to robots right, tells robot to turn right.
      If (pixels >= 5) And (conf >= 5) And (servo1 > 140) Then
        MotionCommand = "r"
        Direction = servo1 - 130
        Position = 1
        Counter2 = 0
        Call SendRobotInstructions
        StartTime = Timer()
        Call Wait(100) ' Wait
        EndTime = Timer()
        Exit Sub
      End If

'found goal, if goal in position set GoalFound flag to true
      If (pixels >= 5) And (conf >= 5) And (servo1 >= 120) And (servo1 <= 140)
Then

        If BallCheck = True Then
          StartTime = Timer()
          Call Wait(500) ' Wait
          EndTime = Timer()
          BallPosition = True
          Exit Sub
        Else:
          GoalFound = True
          StartTime = Timer()
          Call Wait(500) ' Wait
          EndTime = Timer()
          Call Search(orange)
          Exit Sub
        End If
      End If
    End If

'check for ball routine.
    If (pixels >= 40) And (conf >= 20) And (servo2 < 111) _
    And (GoalFound = True) And (BallCheck = False) Then
```

```
'if ball at robots feet but to the right, tells robot to sidestep right.
      If (servo1 > 144) Then
         MotionCommand = "d"
         Position = 1
         Counter2 = 0
      End If

'Ball at feet but to the robots left, tells robot to sidestep left.
      If (servo1 < 138) Then
         MotionCommand = "a"
         Position = 1
         Counter2 = 0
      End If

'ball in position to be kicked, robot told to kick ball.
      If (servo1 >= 137) And (Ball_servo1 <= 144) Then
         BallCheck = True
         If YellowGoal = False Then
            Call Search(Blue)
         Else:
            Call Search(Yellow)
         End If
         Exit Sub
      End If

   ElseIf (GoalFound = True) And (BallCheck = False) And (pixels < 40) _
   And (conf < 20) Then
         Counter1 = Counter1 + 1
         If Counter1 = 5 Then
            StartTime = Timer()
            Call Wait(500) ' Wait
            EndTime = Timer()
            MotionCommand = "s"
            Call Search(orange)
            Counter1 = 0
         End If

   Else:
      MotionCommand = "o"

   End If

      Call SendRobotInstructions
      StartTime = Timer()
      Call Wait(100) ' Wait
      EndTime = Timer()
```

```
    If (GoalFound = True) And (BallCheck = False) And (servo2 >= 111) _
    And (pixels >= 2) And (conf >= 20) Then
        BallErrorFlag = True
    End If

End Sub

Private Sub ShootGoal()

    Dim StartTime As Double
    Dim EndTime As Double


        MotionCommand = "k"
        Position = 1
        Counter2 = 0
        Call SendRobotInstructions
        StartTime = Timer()
        Call Wait(5000) ' Wait
        EndTime = Timer()
        Shot = True

End Sub

'This function allows the robot to go
'through a seach routine to find the ball.
'The robot will search in 8 locations for
'the ball covering 160 degrees,
'if the ball can not be found then the
'robot will turn around and start again.

Private Sub Search(SearchColour As ObjectColour)

    Dim StartTime As Double
    Dim EndTime As Double

    If SearchTurn = True Then
        MotionCommand = "l"
        Call SendRobotInstructions
        StartTime = Timer()
        Call Wait(3000) ' Wait
        EndTime = Timer()
        SearchTurn = False
        Exit Sub
    End If

    Select Case Position
```

```
'Straight and down, or goal
Case 1
   Call Camera_Out("SV 0 130")
   Ball_servo1 = 130
   If SearchColour = orange Then
      Call Camera_Out("SV 1 108")
      Ball_servo2 = 108
      Call Colour2Track(orange)
      SearchCount = SearchCount + 1
      If SearchCount > 1 Then
         Position = 2
         SearchCount = 0
      End If
   Else:
      Call Camera_Out("SV 1 145")
      Ball_servo2 = 145
      If YellowGoal = False Then
         Call Colour2Track(Blue)
      Else:
         Call Colour2Track(Yellow)
      End If
      SearchCount = SearchCount + 1
      If SearchCount > 1 Then
         Position = 3
         SearchCount = 0
      End If
   End If

'Straight and up
Case 2
   Call Camera_Out("SV 1 139")
   Ball_servo2 = 139
   Call Camera_Out("SV 0 130")
   Ball_servo1 = 130
   If SearchColour = orange Then
      Call Colour2Track(orange)
   End If
   SearchCount = SearchCount + 1
   If SearchCount > 1 Then
      Position = Position + 1
      SearchCount = 0
   End If



'Right and up, or Goal
```

```
Case 3
   Call Camera_Out("SV 0 161")
   Ball_servo1 = 161
   If SearchColour = orange Then
      Call Colour2Track(orange)
      SearchCount = SearchCount + 1
      If SearchCount > 1 Then
         Position = Position + 1
      End If
   Else:
      If YellowGoal = False Then
         Call Colour2Track(Blue)
      Else:
         Call Colour2Track(Yellow)
      End If
      SearchCount = SearchCount + 1
      If SearchCount > 1 Then
         Position = 4
         SearchCount = 0
      End If
   End If

'Far Right and up, or goal
Case 4
   Call Camera_Out("SV 0 192")
   Ball_servo1 = 192
   If SearchColour = orange Then
      Call Colour2Track(orange)
      SearchCount = SearchCount + 1
      If SearchCount > 1 Then
         Position = Position + 1
      End If
   Else:
      If YellowGoal = False Then
         Call Colour2Track(Blue)
      Else:
         Call Colour2Track(Yellow)
      End If
      SearchCount = SearchCount + 1
      If SearchCount > 1 Then
            Position = 7
      End If
   End If

'Far Right and down
Case 5
   Call Camera_Out("SV 1 108")
```

```
    Ball_servo2 = 108
    Call Camera_Out("SV 0 192")
    Ball_servo1 = 192
    If SearchColour = orange Then
       Call Colour2Track(orange)
    End If
    SearchCount = SearchCount + 1
    If SearchCount > 1 Then
          Position = Position + 1
    End If


'Right and down,
Case 6
    Call Camera_Out("SV 0 161")
    Ball_servo1 = 161
    If SearchColour = orange Then
       Call Colour2Track(orange)
       SearchCount = SearchCount + 1
       If SearchCount > 1 Then
          Position = Position + 1
       End If
    End If

'Left and down, or goal
Case 7
    Call Camera_Out("SV 0 99")
    Ball_servo1 = 99
    If SearchColour = orange Then
       Call Camera_Out("SV 1 108")
       servo2 = 108
       Call Colour2Track(orange)
       SearchCount = SearchCount + 1
       If SearchCount > 1 Then
          Position = 8
          SearchCount = 0
       End If
    Else:
       Call Camera_Out("SV 1 145")
       Ball_servo2 = 145
       If YellowGoal = False Then
          Call Colour2Track(Blue)
       Else:
          Call Colour2Track(Yellow)
       End If
       SearchCount = SearchCount + 1
       If SearchCount > 1 Then
```

```
            Position = 9
            SearchCount = 0
        End If
    End If

'left and up
Case 8
    Call Camera_Out("SV 1 139")
    Ball_servo2 = 139
    Call Camera_Out("SV 0 99")
    Ball_servo1 = 99
    If SearchColour = orange Then
        Call Colour2Track(orange)
    End If
    SearchCount = SearchCount + 1
    If SearchCount > 1 Then
        Position = Position + 1
        SearchCount = 0
    End If


'Ball Not Found on inital scan,
'therefore have to turn and scan
Case 9
    MotionCommand = "l"
    Call SendRobotInstructions
    StartTime = Timer()
    Call Wait(300) ' Wait
    EndTime = Timer()
    MotionCommand = "l"
    Call SendRobotInstructions
    StartTime = Timer()
    Call Wait(300) ' Wait
    EndTime = Timer()
    MotionCommand = "l"
    Call SendRobotInstructions
    StartTime = Timer()
    Call Wait(300) ' Wait
    EndTime = Timer()
    MotionCommand = "l"
    Call SendRobotInstructions
    StartTime = Timer()
    Call Wait(300) ' Wait
    EndTime = Timer()
    SearchTurn = True
    Position = 10
```

```
'Left and down, or Goal
'For goal go back to turn
Case 10
   Call Camera_Out("SV 0 99")
   Ball_servo1 = 99
   If SearchColour = orange Then
      Call Camera_Out("SV 1 108")
      Ball_servo2 = 108
      Call Colour2Track(orange)
      SearchCount = SearchCount + 1
      If SearchCount > 1 Then
         Position = 11
         SearchCount = 0
      End If
   Else:
      Call Camera_Out("SV 1 145")
      Ball_servo2 = 145
      If YellowGoal = False Then
         Call Colour2Track(Blue)
      Else:
         Call Colour2Track(Yellow)
      End If
      SearchCount = SearchCount + 1
      If SearchCount > 1 Then
         Position = 9
         SearchCount = 0
      End If
   End If

'left and up
'go back to turn
Case 11
   Call Camera_Out("SV 1 139")
   Ball_servo2 = 139
   If SearchColour = orange Then
      Call Colour2Track(orange)
   Else:
      If YellowGoal = False Then
         Call Colour2Track(Blue)
      Else:
         Call Colour2Track(Yellow)
      End If
   End If
   SearchCount = SearchCount + 1
   If SearchCount > 1 Then
      Position = 9
```

```
            SearchCount = 0
        End If

    End Select

End Sub

'-------------------------------------------------------------------------------
'-------------------------------------------------------------------------------
'-------------------------------------------------------------------------------
'-------------------------------------------------------------------------------
'this function retrieves the compass heading of the ball and goal so
'the ball to goal heading can be calculated to help speed up the robots
'shooting time once it reaches the ball.

Private Sub Headingbutton_Click()
    Call Heading(70, 300)
End Sub

Private Function Heading(BallH As Double, GoalH As Double) As Heading2

    Dim temp As Integer
    Dim HeadingCount As Integer
    HeadingCount = 0
    HeadingError = False
    BallHeading = False
    GoalHeading = False
    HeadingCount = 0
    Position = 1
Do
    DoEvents
    If HeadingCount < 3 Then
        Call Search(orange)
        If (pixels > 3) And (conf > 20) Then
            Heading.BallH = compass
            BallHeading = True
        End If
        HeadingCount = HeadingCount + 1
    End If

    temp = Position
    Position = Position + 1

    If HeadingCount < 3 Then
        If postion = 1 Or postion = 3 Or postion = 4 Or postion = 7 Then
            Search (Blue)
            If (pixels > 10) And (conf > 30) Then
```

```
            Heading.GoalH = compass
          End If
        End If
        HeadingCount = HeadingCount + 1
      End If

      If BallHeading = False Then
        Position = temp
      End If

'if it gets to position 9 then jst goto GetToBall routine,
'but if BallHeading = true then search at ball heading first.

      If Position = 9 Then
        HeadingError = True
      End If

Loop Until (BallHeading = True And GoalHeading = True) Or _
(BallHeading = True And HeadingError = True)

End Function

Private Function BallToGoal() As Double

    Dim value As Heading2
    Dim temp As Double

    temp = value.BallH - value.GoalH

End Function
```

FORM 2

```
'plots points on a picture to show trajectory
Private Sub Command1_Click()
Picture1.Cls
Picture1.Line (0, 300)-(600, 300)
Picture1.Line (300, 0)-(300, 600)
End Sub

Public Sub setpoint(x As Double, y As Double)

Label1.Caption = "x = " & x - 300
Label2.Caption = "y = " & y - 300

Picture1.Circle (x, y), 3, vbRed

End Sub
```

ModPublics

```
Public Type Vector2
   x As Double
   y As Double
End Type

Public Type Heading2
   BallH As Double
   GoalH As Double
End Type

Public compass As Long
Public compassOld As Long

Public MotionFinished As Boolean
Public MadeCorrection As Boolean

Public Ack As Boolean
Public InitCom As Boolean
Public YellowGoal As Boolean
Public RGB As Boolean
Public StopProgram As Boolean
Public ComsConnected As Boolean

Public GetPacketCount As Integer
Public SearchCount As Integer   'counts the no. of times the robot
                                'has search a particular position.
Public Counter1 As Integer  'Counter for loss of contact with the ball.
Public Counter2 As Integer
Public Position As Integer  'position no. for the search function.
Public Search2ndRun As Boolean    'Search twise before turning around
Public Text1Count As Integer
Public Text2Count As Integer
Public Text3Count As Integer
Public Object As Integer
Public SearchTurn As Boolean

'Distance Variables
Public VariableCount As Integer
Public AvgDistance As Long
Public AvgDistance1 As Long
Public AvgDistance2 As Long
Public DistH As Double
Public DistV As Double
Public TravelLeft As Boolean
Public OnComing As Boolean
Public pi As Double
Public Xco As Variant
```

```
Public Yco As Variant
Public Xcompare As Variant
Public Ycompare As Variant
Public Xcompareb As Variant
Public Ycompareb As Variant
Public BallStationary As Boolean
Public ObjectSize As Double
Public ObjectHeight As Double
Public Servo90 As Integer
Public Xintercept As Long
Public Yintercept As Long
Public TSlope As Long
Public RobotInitalised As Boolean
Public RobotIn As String
Public MotionCommand As String  'single letter command that
                    'relates to a motion on the robot.
Public Direction As Integer     'no of horizontal servo movements
                    'from the centre.
Public BallFound As Boolean     'true if ball has been found and is at robots feet
Public GoalFound As Boolean     'true if goal has been found
Public Shot As Boolean          'true if robot has taken a shot at goal.
Public BallPosition As Boolean  'true if robot is positioning itself
                    'around the ball to shoot a goal.
Public GoalCheck As Boolean     'true if robot has checked where the ball is located
                    '& goal is still in right location.
Public BallCheck As Boolean
Public BallErrorFlag As Boolean 'true if ball or goal fall out of limits &
                    'sets the robot back to finding the ball.
Public HeadingError As Boolean  'error trying to get heading so abort
Public BallHeading As Boolean
Public GoalHeading As Boolean
Public Face As Double           'compass value to face to see goal

Public Enum ObjectColour
   orange
   Blue
   Yellow
   Magenta
   Cyan
End Enum

Public T As Variant
Public mx As Long
Public my As Long
Public x1 As Long
Public y1 As Long
Public x2 As Long
```

```
Public y2 As Long
Public pixels As Long   'No. of pixels in bounding box
Public conf As Long
Public servo1 As Long   '(Horizontal) servo 0 value
Public servo2 As Long   '(vertical) servo 1 value
Public temp_string As Variant

Public Ball_mx As Long
Public Ball_my As Long
Public Ball_x1 As Long
Public Ball_y1 As Long
Public Ball_x2 As Long
Public Ball_y2 As Long
Public Ball_pixels As Long   'No. of pixels in bounding box
Public Ball_conf As Long
Public Ball_servo1 As Long   '(Horizontal) servo 0 value
Public Ball_servo2 As Long   '(vertical) servo 1 value

Public BGoal_mx As Long
Public BGoal_my As Long
Public BGoal_x1 As Long
Public BGoal_y1 As Long
Public BGoal_x2 As Long
Public BGoal_y2 As Long
Public BGoal_pixels As Long   'No. of pixels in bounding box
Public BGoal_conf As Long
Public BGoal_servo1 As Long   '(Horizontal) servo 0 value
Public BGoal_servo2 As Long   '(vertical) servo 1 value

Public YGoal_mx As Long
Public YGoal_my As Long
Public YGoal_x1 As Long
Public YGoal_y1 As Long
Public YGoal_x2 As Long
Public YGoal_y2 As Long
Public YGoal_pixels As Long   'No. of pixels in bounding box
Public YGoal_conf As Long
Public YGoal_servo1 As Long   '(Horizontal) servo 0 value
Public YGoal_servo2 As Long   '(vertical) servo 1 value

Public Obsticle_mx As Long
Public Obsticle_my As Long
Public Obsticle_x1 As Long
Public Obsticle_y1 As Long
Public Obsticle_x2 As Long
Public Obsticle_y2 As Long
Public Obsticle_pixels As Long   'No. of pixels in bounding box
```

```
Public Obsticle_conf As Long
Public Obsticle_servo1 As Long   '(Horizontal) servo 0 value
Public Obsticle_servo2 As Long   '(vertical) servo 1 value

Public Function Vec2(x As Double, y As Double) As Vector2
Vec2.x = x
Vec2.y = y
End Function
```

PIRKUS CODE

```
//
//      Pirkus static motion code.
//
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <pirkus-r-sdk/pirkus_r_sdk_re001.h>

#define UART_BAUD_RATE        (u32)57600
//
u08             txBuffer[10];                   // Buffer for transmission
u08             rxBuffer[10];                   // Buffer for reception
u16             walk = 0;

main() {

MOTION_LIST_HEADER mhd;
u16    motion_id;
u08    ch, command;
u08 cam_in[80];
u16 s1 = 0, s2 = 0, dist = 0;
s16    err;
BIOS_Init( UART_BAUD_RATE , txBuffer, sizeof( txBuffer), rxBuffer,
sizeof( rxBuffer ) );
I2C_Init( 200 );                // Initializing I2C where EEPROM is connected
sei();                          // The interruption is made effective
Servo_Init();                   // It initializes servo
Servo_RTableLoad(); // Load before doing, it is necessary to finish I2C_Init.
KeyFrame_Init();                // KeyFrame is initialized
MotionTask_Init();              // MotionTask is initialized
MotionTask_MoveToOrigin(10);    // Moves servo to starting point.
motion_id = 0;
ch = BIOS_CharGet();            // Waits for 1 letter input
if ( ch == '!' ) {
PCLINK_Init();      // Initialization of PCLINK.
BIOS_StringPut("PCLINK Initialized!");
while(1){       // The command from of Motion Edit is processed
PCLINK_CommandWait();
}
}
if ( ch == 'm' ) {
BIOS_StringPut("ACK\r");
while (1){
Servo_Wait2500us( 100 );
BIOS_StringGet(cam_in, 20);
```

```
if (strcmp (cam_in, "") != 0 ){
BIOS_StringPut("ACK\r");
 sscanf(cam_in, "%s %d %d %d" , &command, &dist, &s1, &s2);
BIOS_Printf("%s %d %d %d\r", &command, dist, s1, s2);
switch ( command ){
case 'w':
if ( walk = 0 ){
if ( MotionTask_IsRunning() == 0){
// It was input, from letter it converts to numerical value
motion_id = 16;
// For verifying whether motion is effective, loading the header
if ( MotionList_HeaderLoad( motion_id, &mhd ) == NO_ERROR ) {
mhd.m_aName[ MOTION_NAME_MAX_LENGTH -1] = 0;
// executes
MotionTask_Start_EEPROM( motion_id );
walk = 1;
}
}
} else {
if ( MotionTask_IsRunning() == 0 ){
// It was input, from letter it converts to numerical value
motion_id = 17;
// For verifying whether motion is effective, loading the header
if ( MotionList_HeaderLoad( motion_id, &mhd ) == NO_ERROR ) {
mhd.m_aName[ MOTION_NAME_MAX_LENGTH -1] = 0;
// executes
MotionTask_Start_EEPROM( motion_id );
}
}
}
break;
case 'k':
if ( MotionTask_IsRunning() == 0){
// It was input, from letter it converts to numerical value
motion_id = 8;
// For verifying whether motion is effective, loading the header
if ( MotionList_HeaderLoad( motion_id, &mhd ) == NO_ERROR ) {
mhd.m_aName[ MOTION_NAME_MAX_LENGTH -1] = 0;
// executes
MotionTask_Start_EEPROM( motion_id );
}
}
break;
case 'a':
if ( MotionTask_IsRunning() == 0){
// It was input, from letter it converts to numerical value
motion_id = 31;
```

```
// For verifying whether motion is effective, loading the header
if ( MotionList_HeaderLoad( motion_id, &mhd ) == NO_ERROR ) {
mhd.m_aName[ MOTION_NAME_MAX_LENGTH -1] = 0;
// executes
MotionTask_Start_EEPROM( motion_id );
}
}
break;
case 'd':
if ( MotionTask_IsRunning() == 0){
// It was input, from letter it converts to numerical value
motion_id = 30;
// For verifying whether motion is effective, loading the header
if ( MotionList_HeaderLoad( motion_id, &mhd ) == NO_ERROR ) {
mhd.m_aName[ MOTION_NAME_MAX_LENGTH -1] = 0;
// executes
MotionTask_Start_EEPROM( motion_id );
}
}
break;
case 'r':
if ( MotionTask_IsRunning() == 0){
// It was input, from letter it converts to numerical value
motion_id = 24;
// For verifying whether motion is effective, loading the header
if ( MotionList_HeaderLoad( motion_id, &mhd ) == NO_ERROR ) {
mhd.m_aName[ MOTION_NAME_MAX_LENGTH -1] = 0;
// executes
MotionTask_Start_EEPROM( motion_id );
}
}
break;
case 'l':
if ( MotionTask_IsRunning() == 0){
// It was input, from letter it converts to numerical value
motion_id = 23;
// For verifying whether motion is effective, loading the header
if ( MotionList_HeaderLoad( motion_id, &mhd ) == NO_ERROR ) {
mhd.m_aName[ MOTION_NAME_MAX_LENGTH -1] = 0;
// executes
MotionTask_Start_EEPROM( motion_id );
}
}
break;
case 's':
MotionTask_MoveToOrigin( 100 );
walk = 0;
```

```
break;
}
//u16 dist = (3000 + (10*(sv_h - 128)));
//if (60 < sv_h < 210){
//Servo_PWMSet( 19, dist );
//}
}
}
}
}
```

Appendix D (Data Sheets & Publications)

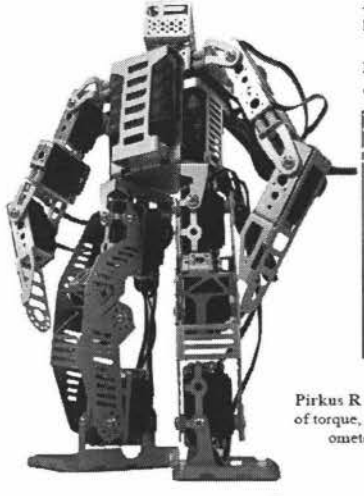
A MECHATRONICS APPROACH TO AUTONOMOUS CONTROL OF A BIPEDAL HUMANOID SOCCER ROBOT USING COLOUR VISION

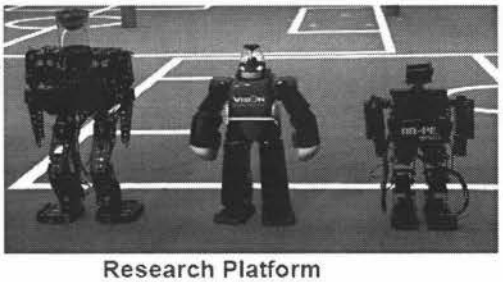
B.J. Rielly, O. Diegel, C.L. Kang, M.J. Read, J.R. Zyzalo, J. Potgieter, W.L. Xu
Institute of Technology and Engineering, Albany Campus, Massey University
Email: B.J.Rielly@massey.ac.nz

Introduction

A Mechatronics project in which a vision system is being developed for autonomous control of a bipedal humanoid robot to compete in the Robocup robot soccer competition.

Includes the implementation of the system including a set of mathematical models showing the effect of any error caused by camera resolution and to effectively compare monoscopic and stereo vision systems.



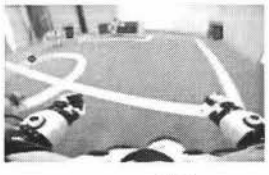


Existing Systems

Colour CCD or CMOS cameras: resolution ranging from 320 x 240 to 800 x 600 pixels using wide-angle lenses, omni-directional camera's, and stereo vision.

Controllers: PDA, DSP, or Microcontrollers.

Competition Robots: Kondo KHR-1; Tomi; Vision; RO-PE; Robosapien.



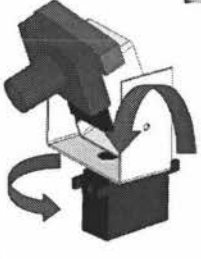
Research Platform

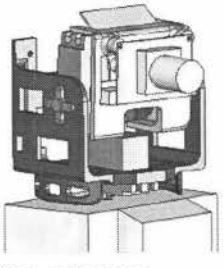
Pirkus R type 01 DX; 21 PRS-DE07M digital servos, PWM controlled, 7.1kg/cm of torque, 0.11s. response time; 21 DOF; 2 gyros for pitch and roll; 3 axis accelerometer; Atmel Atmega 128 micro controller; Bluetooth and RS232 serial; Height = 29cm, Weight = 1050g.

Camera

CMUcam2: produced by Carnegie Mellon University, $179US, 176 x 255 resolution, 75MHz Ubicom SX52, RS232 serial, control up to 5 servo, Omnivision OV6620 CMOS camera, master and slave mode , built-in image processing tools including colour tracking.

YUV colour space used (Y = luminance, U = blue chrominance, V = red chrominance), colour processing done on the U,V components and threshold Y.





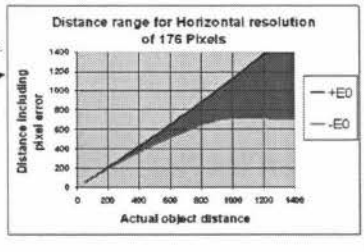
Monoscopic Vision

Depth perception is entirely based on each colour being assigned to an object of known size.

Resolution error, two areas :

How far the distance will vary with the object being a pixel value (Op).

How far the distance will vary for the image reaching the next pixel size.

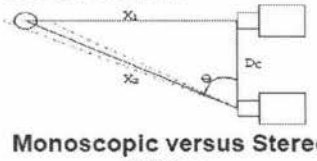


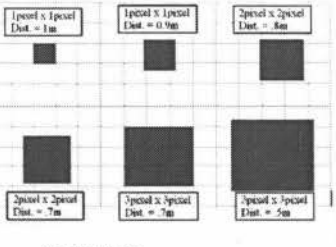
Stereo Vision

Allows more accurate depth perception, and object size does not need to be known.

Resolution error:

Objects central pixel in the cameras image deviating by 1 or 2 pixels.





Monoscopic versus Stereo Vision

Is the added accuracy of stereo vision for a soccer robot is worthwhile?

Cost and complexity of a stereo vision system is twice that of monoscopic vision, therefore the informational gain must be over twice as valuable. Accuracy becomes less important at greater distances.

Implementation of Vision

Fast and slow systems, as commonly used by the armed forces, will be utilised.

The fast system is a more focused system that will have a more narrow view angle and focus on only a single item at a time. An extra servo will be added to allow the neck to tilt in the vertical axis. This will allow full tilt and pan of the camera in real-time using colour tracking, allowing the body to follow the head's bearing.

Finally the addition of the slow system will be implemented, which has a wide view. It can pick up all the objects surrounding the robot and feedback any objects of importance to the fast system.

References

1. The Robocup Federation Website, "Home Page", http://www.robocup.org, visited on 6/5/2006.
2. S. Behnke, J. Müller, M. Schreiber, "Using Handheld Computers to Control Humanoid Robots", *Proceedings of 1st International Conference on Electronic Autonomous Robots and Humanoids*, Freiburg, Germany, 2004.
3. A. Rowe, C. Rosenberg, I. Nourbakhsh, CMUcam Website, "Home Page", http://www.cs.cmu.edu/~cmucam, visited on 20/8/2006.

Massey University

Humanoid Biped Robots: Walking and balancing using Natural Dynamics, ZMP, and Gyroscopic Sensors

L.C. Kang, O. Diegel, B.J. Rielly, J.R. Zyzalo, M.J. Read, J. Potgieter, W.L. Xu

Massey University, Albany, New Zealand

Email: l.kang@massey.ac.nz

Abstract

This paper describes research being conducted on humanoid bipedal robot walking and balancing using Zero Moment Point, gyroscopic sensors, and exploiting natural dynamics in these systems. A robot is first controlled using natural dynamics, static walking and gyroscopes. A method is proposed to combine the quick accessibility of static walking with the flexibility and adaptability of dynamic walking using Zero Moment Point. Sub-motions are used to generate complete static walking gait motions. Gyroscopic sensors and switches are used to help detect stability and balance. A method is described on how these are used, and a method is described on how the static motion selection is implemented and on when both static and dynamic stability will be used.

Keywords: Gyroscope, Biped, Humanoid, Robot, ZMP

1 Introduction

Bipedal humanoid robots can be useful for a large range of applications ranging from entertainment to helping out in the work force [1,2]. Many researchers have been working towards emulating anthropoids in robotics. One of the main areas of focus in this research has been in achieving human-like walking gaits. Gait is used to describe the manner or style of walking, although the words gait and walking are sometimes used interchangeably [3].

There are two main ways a bipedal gait can be implemented: one is through static walking and the other through dynamic walking. Static walking is mainly used by small servo actuated humanoid robots [5], whereas larger DC actuated robots commonly use dynamic walking for gait generation.

Static walking is achieved by creating and recording key frames of servo speeds and positions. The key frames are then replayed, in sequence, to achieve the desired motion.

Dynamic walking is implemented using Zero Moment Points (ZMP), Center of Pressure (COP) or Center of Mass (COM), which will be further discussed in some detail in this paper. To achieve dynamic walking, the robot requires feedback to know things are happening to it. Gyroscopic sensors, for example, are used to help the robot to balance and the technique of ZMP is used to maintain dynamic balance of the robot. ZMP is currently used in humanoid biped robots such as the Honda ASIMO [4].

This paper describes a method that combines dynamic and static walking techniques in such a way that various pre-recorded key-frame based motions are recalled based on the requirements of dynamic walking algorithms. This allows the robot to use preprogrammed motions, which do not require any extra computing power to generate, but with intelligent data from the various sensors used to decide whether to use these preprogrammed motions or to dynamically react to the environment, which requires heavy computing power, until the preprogrammed motions can be used.

2 Basics of Humanoid Bipedal Walking

2.1 Humans as Models

From everyday human experience, it is apparent that the human gait is a flexible and effective means of exploiting bipedal locomotion for perambulatory purposes. Therefore, understanding how the human gait works can greatly help with understanding how to implement gait in a bipedal robot. Human gait can be broken up into two phases: single phase (when one foot is on the ground) and double phase (when both feet are on the ground). In normal human gait about 20% of the gait cycle is in double phase. This means that, if we model robot gait based on human gait, 80% of the time will be spent on trying to balance, with only one foot on the ground [1,4,6]. Also, at some point during the walking cycle, the robot has to allow itself to fall forward to allow for a smooth gait pattern.

2.2 Types of Balance

The balance of a bipedal robot can be broken down into two separate sub-problems: one is that of static stability and the second is that of walking and moving.

Static stability relates to being able to maintain the robots balance while standing still, and possibly carrying an object that may change the robots center of mass (COM). The robot should be able to counter the applied force automatically and adjust its body position accordingly; the same goes for uneven surfaces. The robot should also be able to maintain its balance when an external force is applied to it. It should, ideally, react by pushing back in the opposite direction or by stumbling in the same direction in which the force was applied – acting as an inverted pendulum on two legs.

Besides being able to keep its balance while stationary, the robot must also be able to walk and maneuver around an area while avoiding obstacles and maintaining balance. It must be able to maintain its balance while adapting to any changes in the environment and any external forces that may be applied to it. Hence it should be able to carry objects while walking; bump into objects and still maintain balance; and even kick a ball or push a button without falling over.

2.2.1 Gait Generation

2.2.1.1 Static Walking

There are two main types of bipedal gaits for humanoid robotics: static walking and dynamic walking. Static walking is generally very slow, sometimes taking up to a few seconds to complete a single step. During static walking the robot controls its center of mass (COM) in such a way that the COM is located in the middle of the points that are in contact with the ground. Static walking is commonly generated manually by recording sequences of key frames (references of servo positions and speed) which can then be played back. An advantage of the static bipedal gait is that it is easier to implement than dynamic walking and can be quicker to implement in real time. While static walking has its advantages it compromises on adaptability.

2.2.1.2 Dynamic Walking

In dynamic walking, posture control based on dynamic generalizations of the concept of center of mass, such as the zero-moment point (ZMP) [7] (originally proposed by Vukobratovic and Juricic (1969)) and Center of Pressure (COP) [8] are used for generating stable bipedal gaits [1]. The ZMP is the point on the ground where the torques around the (horizontal) x and y axes, generated by reaction forces and torques, are equal to zero. If the ZMP is enclosed within the zone of the support region defined by the foot, the gait is dynamically balanced, thus the robot will not fall [1,7,8]. Dynamic walking allows the robot to be more adaptable to changing environments without having to setup all possible solutions or different situations.

3 Sensors and Switches used in Robot Balance Systems

Feedback sensors can be used to detect certain states and situations that the robot is in. Some examples of sensors that could be used are: gyroscopic sensors, accelerometers, and switches or pressure pads. Gyroscopic sensors detect the rate of change of angular displacement. Two gyroscopic sensors can be used to determine if the robot is falling and in which direction it is falling in. The two gyroscopic sensors can thus be used to sense the roll and pitch of the robot in the frontal and sagittal plane. An accelerometer can be used to detect the movement in the transverse, sagittal and frontal planes of the robot. This helps to determine the direction that the robot is moving, and the speed it is moving at can also be determined. Switches or pressure pads on the soles of the feet can help determine whether the robot is flat footed or not, and can help detect when the robot is on uneven surfaces. By being able to determine when the robot has a flat foot on the ground, the Center-of-Pressure and the ZMP can be determined, and from this data various torques can be applied to the motors to maintain balance.

Another less common option for balancing is to use a camera for vision sensing. Vision can help a robot judge if it is coming into contact with an external force. This can allow it to make adjustments prior to the moment the force is applied. For example it may be able to detect if a hand is coming in to push it and then apply force in the opposite direction of the incoming force to counteract it before it is too late.

4 Humanoid Bipedal Research

In 1986 Honda started development on a humanoid robot. The desired goal was to develop a robot able to coexist and collaborate with humans, and to perform tasks that humans cannot. In other words, to create a mobile robot that brings additional value to human society. ASIMO (Figure 1) is a bipedal robot that is able to negotiate stairs. The ASIMO uses ZMP and ground reaction forces to control the balancing of the robot. The latest ASIMO weighs in at about 43Kg and stands 130cm tall and has 34 DOF. It can walk at speeds of up to 2.7 km/h and run at 6 km/h. Battery life for the Honda biped is about 1 hour.

Sony and Toyota are two other multinational corporations that have invested resources into humanoid robotic research. Sony's development is called the QRIO (Quest for Curiosity). It stands 0.6 meters tall and weighs 7.3Kg, it has superior balancing and walking abilities to the ASIMO

possibly due to its smaller frame. The QRIO [9] is the first biped to be able to run, which is defined as having both feet off the ground at some point during the gait cycle. Toyota [10] made their announcement of joining robotics research by introducing two trumpet playing robots; one a bipedal robot and the other on wheels similar to that of the Segway [11].

Figure 1: Honda ASIMO.

4.1 Natural Dynamics

Pratt and Pratt [6] from the Massachusetts Institute of Technology (MIT) Leg Laboratory proposed a method to exploit natural dynamics in the control of a planar bipedal walking robot. The main idea behind this research is to use the natural swinging motion of the leg to help produce a natural gait pattern. In order to achieve this, the team from MIT Leg Lab uses three main functions: an added knee cap, a compliant ankle and a passive swing leg.

The added knee cap simplified the control and made the resultant motion smoother and more efficient [6]. During the down swing the knee locks up with the knee cap without any actuator torque. The knee cap allows for better control and a more natural swinging leg. It also eliminates the problem where the actuator vibrates while trying to locate its exact position.

Pratt and Pratt also added a compliant ankle. The compliant ankle incorporated of a spring that acted as a dampener. Velocity fluctuations were reduced since the center of pressure on the foot can travel forward, staying below the center of mass of the body. The compliant ankle also helps control speed and assists motion at the end of the stride through toe off [6].

A passive swing leg is employed to attain a natural swinging motion. This is achieved by moving the hip joint to the desired angel and then allowing the knee to swing freely. Damping is added to the knee to prevent it from banging into the knee cap.

The planar bipedal robot walks using four main phases. The robot starts with phase one (support) where both feet are on the ground. The transition from phase one to phase two (toe off) is when the heel of the back foot lifts off the ground. During phase two the compliant ankle is pushing the feet off the ground. In phase three (swing) the knee swings forward and transitions into phase four (straighten). Phase four is when the leg is straight and the heel hits the ground on the transition back to phase one. The phases are then repeated for the opposite foot.

4.2 Small Humanoid Bipedal Robot

Researchers at the University of Manitoba describe methods on stabilizing the walking gait of a 30cm tall fully autonomous humanoid bipedal robot [5]. The robot utilizes static walking. The phases for the robot are similar to that used by Pratt and Pratt [6]. The only difference is that the gait pattern was divided into three phases for each step. The robot uses center of mass (COM) for balancing. In single phase the COM is located in the middle of the foot that is in contact with the ground. In double phase the COM is located between the two feet that are in contact with the ground.

The sensor feedback for robot is acquired from two gyroscopes: one for pitch and one for roll. The raw data from the gyroscopes is very noisy so the data was averaged over five samples to smooth out the noise. The data is used to produce a "safe zone" for the robot, so whenever the robot falls out of the safe zone an alarm sounds. With this function the robot is able to indicate that it is about to fall over. The end result of this research shows that the robot is able to detect a fall with 95% accuracy.

5 Humanoid Bipedal Research Platform

The robot used for the research described in this paper is the Pirkus – R Type 01 DX (Figure 2). This platform was chosen because the package came with most of the parts needed for the initial setup of the robot. The Pirkus is approximately 29cm in height and weighs 1050g.

The Pirkus comes with 21 servos for 21 Degrees of Freedom (DOF). Six DOF are allocated for each leg, 4 for each arm and 1 for the head. This give the Pirkus enough DOF to imitate basic human movement without being overly complicated. The servos supplied with the Pirkus package are the PRS-DE07M servos which provide 7.1kg/cm of torque, and a response time of 0.11 seconds. The servos are PWM controlled. The servos are relatively light and powerful.

The onboard controller that comes with the Pirkus package is the Atmel Atmega 128 microcontroller. The controller possesses an 8 bit bus, operating at 16 Million Instructions Per Second (MIPS), and can hold 128 KB in the flash memory. Communication to the Pirkus can be through Bluetooth or Serial RS232 communication. Initial trials of the robot are being performed via the RS232 link and all processing will be done on an external computer to provide the power needed to control the walking and balancing.

Figure 2: Pirkus – R Type 01 DX

6 Combined Static and Dynamic Motion

This research project will design a control system using feedback from various sensors together with generated gait patterns to allow the Pirkus robot to walk and maintain balance. A pre-generated gait pattern along with ZMP will be used to produce a reliable gait for the robot. To help with balancing, sensors will to be added to the feet of the Pirkus. Natural dynamics will be exploited for gait generation. The Pirkus already has a knee cap but a compliant ankle may have to be added for this combined method to work more effectively. The Gyroscopic sensors will provide emergency feedback to indicate that the robot is about to fall over. This allows for a set of procedures, or reflexes, to be set out to maintain the balance of the robot.

6.1 Static Walking Implementation

Static walking is implemented by the use of a set of sub-motions. Each sub-motion of the gait pattern is set up manually and then loaded onto the robot as predefined sub-motions. The sub-motions can be combined together to make the complete motions that the robot needs for locomotion purposes. The walking pattern, for example, that is generated for the Pirkus is made up of three sub-motions as shown in figure 3. The first is the start of the walk. The next sub-motion moves from one step to the next step and finally the last sub-motion is to stop walking. The second sub-motion in the walking pattern can be repeated over and over until the robot is told to stop. This technique of using sub-motions allows for the robot's controller to call the routines quickly without having to generate the motions each time. Turning sub-motions and other necessary sub-motions can be combined to make a complete motion set for the robot.

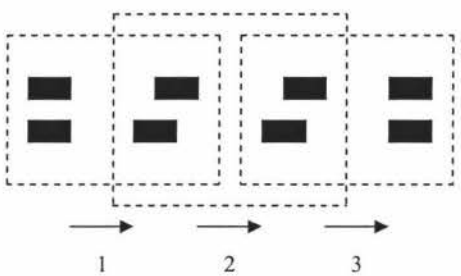


Figure 3: The foot prints of the robot showing the three sub-motions for walking from start of walk to stop of walk.

The sub-motions are generated using a software program called Motion Editor. Motion Editor is a program provided by the developers of the Pirkus. The Motion Editor allows the programmer of the robot to position all the servos of the robot and record the position and the servo speed that is used to get to the desired position. These recorded positions are called key-frames. A set of generated key-frames are used as sub-motions.

From observations of the robot using the static walking technique, accurate walking is only achieved by operating in a controlled environment i.e. a flat surface. When tested on a rough surface, such as carpet, the robot failed to walk. This was due to initial programming being done on a smooth table top surface. To correct this problem, a combination of static walking and dynamic walking could be beneficial as dynamic walking is more adaptable to environmental changes.

6.2 Static Walking Incorporating ZMP

Static walking can be combined with ZMP, which results in synergistic combination of static and dynamic walking. The static walking from the predefined gait patterns are combined with the ZMP calculations to be able to cross reference the progress of the static walk and the dynamic walking ability of the ZMP calculations. By doing this the robot will have quick access to various motion routines while still being actively adaptable at balancing. This will allow the Pirkus to maintain balance in various environments as well as being able to react to external forces that may be applied to the robot.

Of the two methods of dynamic walking, ZMP is preferred over COP. The advantage of ZMP is that it is cheaper and easier to implement than COP. Simple switches can be used that are cheaper than more expensive pressure sensor.

Switches will be added to the feet. A micro switch will be mounted to the corners of each foot, for a total eight switches. The switches enable the robot to know when the foot is flat on the ground. Once the foot is flat on the floor, the robot can use it as a reference point to calculate the position of the robot. The robot can then adjust itself so that the reaction forces and torques at the reference point equal zero.

6.3 Natural Dynamics

The design of the Pirkus is such that the knees already contain a knee cap. This kneecap stops the knees from bending too far and helps incorporate natural dynamics into the gait pattern. Another characteristic that will be added to the robot to help with generating a gait pattern using natural dynamics will be to add toes to the robot's feet. This can be done by adding an extra piece of plate to act as the 'toe' and then hinge it onto each foot with a spring. The spring eliminates the need for extra actuators and acts in a way similar to the compliant ankle developed by the team at the MIT Leg Lab. The toe will help during 'toe off' in the gait.

Applying a natural swing motion to the gait is difficult due to the resistance caused by the gearing of the servo motors used in the Pirkus. Rather than switching off the motors to produce a natural down-swing, a positive actuation can help overcome the resistance of the gears. If a positive actuation is to be used to help produce a natural swing motion, a constant force will have to be found to balance the resistance of the gearing of the servo motors.

6.4 Gyroscopic Sensors

Gyroscopic sensors are used to detect the angular velocity of the torso relative to the ground. The Gyroscopic sensors used are the dual rate micro piezo CN2022A, produced by Century Helicopter Products. The angular velocities received from the gyroscopes are integrated to get absolute angular displacement. Two sensors are used, one to detect the angle relative to the ground in the frontal plane and one to detect the angle relative to the ground in the sagittal plane.

Tests were conducted on the Pirkus using the method of finding the angular displacement to determine the state of balance that the robot is in. The results showed that, though the accuracy of the gyroscopes is unreliable, the accuracy is good enough to detect a fall. Once a fall is detected, the robot calls a motion to stand up on its feet using appropriate sub-motions stored in memory.

6.5 Algorithms

The static walking gait is the main method for implementing locomotive actions. The robot calls the required motions to execute from the motion task library that has been set up. This allows quick and easy execution of tasks as opposed to dynamic walking, which requires more computational processing as the motions are created in real time.

It is preferred to use dynamic walking when the robot cannot navigate its way because of changes in the environment such as uneven surfaces, steps and gradients. Because dynamic walking requires more processing power than static walking, the algorithms for the robot will be designed to use static walking motions as much as possible.

The gyroscopes will govern the logic of when to use dynamic walking over static walking. Static walking is used until an interrupt signal is sent to the controller when the gyroscopes have detected that the robot is becoming unbalanced, or if it has fallen. The robot can then switch to dynamic walking using ZMP.

6.5.1 Motion Selection

For static walking the robots controller decides on motion selection using a rule based algorithm. The rules are based on several factors. The final goal that the robot is trying to achieve is the main aspect that determines the routine the robot will employ. For example, if the robot needs to get to a position one metre in front and to the left of itself, the robot needs to call the turn left motion and then the walk function until it reaches the destination. Motions for standing up are used when the gyroscopes sense the robot has fallen over.

For dynamic walking and balancing the robot needs to be able to move itself in such a way that a point within the ZMP safe zone is balanced. To achieve this, the robot has to know the position of all the servos and move the joints in such a way that the point with no forces acting on it laterally is in the zone.

7 Conclusion and Future Research

As bipedal humanoid robots are used in a wide variety of applications, better methods of locomotion need to be developed. This paper describes several research projects that have been undertaken on gait generation and balancing of humanoid bipedal robots. There are two main ways a bipedal gait can be implemented, one is through static walking and the other through dynamic walking.

This paper describes how static walking is implemented through the use of key frames, whereas the gait generated for a robot using dynamic walking is achieved by employing the methods of ZMP, COP

and COM. Details on the techniques of applying dynamic walking to static walking are examined.

The paper then proposes a method of gait generation that combines static and dynamic motion. With this method, pre-programmed static motions are stored in memory, and dynamic motion principles are used to determine which static motions to employ for any particular circumstances. Gyroscopic sensors and additional switches are used to generate the data required by the dynamic motion algorithms.

Future work on this project include the addition of a flexible toe section to the robot, as well as the possible addition of a compliant ankle to help generate more natural gait motions. In addition to this, much of the future work on this project will be focused on generating the intelligence needed for the dynamic motion to intelligently call up the static motions. This is currently implemented through a simple rule based system, but future methods may include neural networks, or genetic algorithms that will allow the robot to more flexibly adapt and learn from its surroundings

8 References

[1] M. Wahde, and J. Petterson, "A Brief review of Bipedal Robotics Research", *Proceedings of the 8th UK Mechatronics Forum International Conference.* Pp. 480-488, 2002.

[2] R. Brooks, (1996). "Prospects for human level intelligence for humanoid robots", *Proeedings. of the First International Conference on Humanoid Robots (HURO-96)*, 1996.

[3] M. Whittle, *"Gait Analysis an Introduction"*, Chapter 2. Great Britain, Reed Educational and Professional Publishing Ltd. 2002.

[4] K. Hirai, Hirose, M., Haikawa, Y. Takenaka, T. "The Development of Honda Humanoid Robot", *Proceedings of the 1998 IEEE International Conference on Robotics & Automation,* Leuven: Belgium, 1998.

[5] J. Baltes, S. McGrath, and J. Anderson, "The use of gyroscope feedback in the control of the walking gaits for a small humanoid robot", *The Seventh RoboCup Competitions and Conferences*, Berlin: Springer, 2005.

[6] J. Pratt, and G. Pratt, "Exploiting natural dynamics in the control of a planar bipedal walking robot", *Proceedings of the Thirty-Sixth Annual Allerton Conference on Communication, Control, and Computing*, pages 739-748, 1998.

[7] T. Arakawa, and T. Fukuda, "Natural motion trajectory generation of bipedlocomotion robot using genetic algorithm through energy optimization", *Proeedings .of the 1996 IEEE International Conference on Systems, Man and Cybernetics*,pp. 1495–1500, 1996.

[8] P. Sardain, and G. Bessonnet, "Forces Acting on a Biped Robot. Center of Pressure – Zero Moment Point", *IEEE transaction on systems, man, and cybernetics – Part A,* vol. 34, no. 5, pp 630 – 637, 2004.

[9] Sony Dream Robot QRIO, "Home Page", http://www.sony.net/SonyInfo/QRIO/, visited on 01/09/2006.

[10] Toyota Partner Robot, "Home Page", http://www.toyota.co.jp/en/special/robot/, visited on 01/09/2006.

[11] Segway, "Home Page", www.segway.com, visited on 01/09/2006.

A Mechatronics Approach to Autonomous Control of a Bipedal Humanoid Soccer Robot using Colour Vision

B.J. Rielly, O. Diegel, C.L. Kang, M.J. Read, J.R. Zyzalo, J. Potgieter, W.L. Xu

Institute of Technology and Engineering, Albany Campus, Massey University

Email: B.J.Rielly@massey.ac.nz

Abstract

This paper describes a mechatronics project in which a vision system is being developed for autonomous control of a Bipedal humanoid robot to compete in the Robocup robot soccer competition. The implementation of the system is discussed and a set of mathematical models are developed to effectively compare monoscopic and stereo vision systems.

Keywords: Humanoid, Colour vision, Robotics, Autonomous Control

1 Introduction

Vision systems have become an important factor in robotics, especially in the development of autonomous robots. Vision systems allow a robot to interact better with its environment, detect objects, and to avoid obstacles in their path. Competitions have been created to stimulate development of this field especially in stereo and colour vision.

One example of such an event is the Robocup soccer competition [1], which primarily uses colour vision as the means for robot navigation, and requires the robots playing to be fully autonomous, bipedal in motion, and have vision as their main form of navigation. The field in the Robocup consists of 6 colour posts for robot navigation, two colour goals (yellow and blue), and an orange ball.

This paper gives an overview of a humanoid biped robot platform being developed for the Robocup soccer competition. It discusses the hardware and software systems in use and the mechatronics approach used in the development of new systems.

It then discusses and compares monoscopic and stereo vision systems and develops a set of mathematical models to determine the effect of any error caused by camera resolution. These errors occur because the image size can only increase to the next pixel size when it has moved within enough of a range to be detected by the resolution of the camera.

In discussing the pros and cons of monoscopic and stereo vision systems in the context of robot soccer, it is concluded the monoscopic systems are well suited for the task. Their large error at long distances is not critical because, at that stage, the robot only needs a general direction to head in. As the robot approaches the ball, the error reduces to a level that is negligible.

2 Existing Systems

Many cameras that are currently used by teams conducting research into humanoid robots for robot soccer competitions are low cost colour CCD or CMOS cameras, with a resolution ranging from 320 x 240 to 800 x 600 pixels [2]. They often employ wide-angle lenses allowing a robot to see the ball, the goal, its own feet, and any opponents in a single image, thereby minimising Localization issues. The cameras are attached to a PDA (Personal desktop assistant) with approximately 500MHz processing power and 128MB of RAM. The PDA is required for the additional computational power needed to process the images. This has been shown to be effective with such existing systems as Abarenbou [2], which is based on the Kondo KHR-1 robot; Toni [2], which has similar functionality to the robot used in this research, including a accelerometer and 2 gyros; and the Robosapien soccer robots [2][3]. A sample of the Robosapien vision is shown in Figure 1.

Another humanoid robot with built-in colour vision is the Robosapien V2. The onboard camera has its colour image processing done via its 55 MHz controller. This camera uses the RGB colour space and is only designed to see individual red, green and blue colours. This system allows the Robosapien to track a blue light, see a green ball and red bowling pins. Yet none of the research community has managed to interface with this system [4].

Omni-directional camera's have the benefit of allowing the robot to have a complete view of the entire area and thereby giving it the ability to localise itself in that area with objects in its field of view. This idea has been used in robot soccer by team Osaka [5] for their Robocup humanoid entry Vision that has won them many titles in the competition.

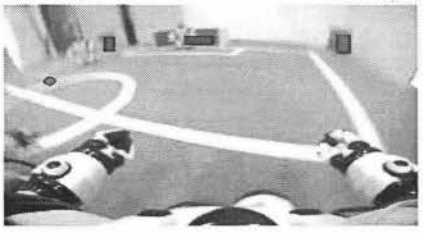


Figure 1: Robosapien vision through wide-angle lens

2.1 Camera Selection Considerations

For this project colour image processing was chosen over grey scale for several reasons: The robot must be able to compete in the Robocup competitions which are designed around colour vision; therefore it would need to be able to process colour images for localization of the robot, and detection of the goals and ball. Another reason for using colour image processing is that a colour image contains information about an object's details that a grey scale image may not distinguish just from the intensity of a picture.

The use of an omni-directional camera has the benefit of allowing the robot to have a complete view of the entire area and thereby gives it the ability to localise itself in that area using objects in its field of view as markers. This idea has been used in robot soccer by team Osaka [5] for their Robocup humanoid entry and has won them many titles in the competition. However, this form of vision was not chosen because the robot would not significantly benefit by knowing what is behind it. An omni-directional vision system would give the robot a larger number of obstacles to focus on than required by our research, which intends to focus the robots attention on a more specific task.

Stereo vision allows robots to have more advanced depth perception using the difference between the images captured via each camera. Such a system can be used to give the robot a more accurate account of distance and direction related to objects in the robots path. In the case of robot soccer it would give a robot a co-ordinate of the ball's location in an XYZ plane.

One of the key focuses of this paper is a detailed analysis of stereo vision in comparison to monoscopic vision and the development of mathematical models to determine which system should be used under specific conditions.

2.2 Research Platform

The platform chosen for this research is the Pirkus R type 01 DX as shown in Figure 2. This humanoid robot package contains 21 servos encased in an aluminium exoskeleton. The Pirkus stands approximately 29cm high and weighs approximately 1050g. For the Robocup this would place the Pirkus in the kid size division of the competition which ranges from 30cm to 60cm tall.

The Pirkus' 21 servos allow 21 Degrees of Freedom (DOF); 6 DOF in each leg (including the waist joint); 4 DOF in each arm (including the shoulder joint); and 1 DOF for the head. The servos used for this robot are the Pirkus robotics PRS-DE07M digital servos which are pulse width modulation controlled and have 7.1kg/cm of torque, and a response time of 0.11 seconds.

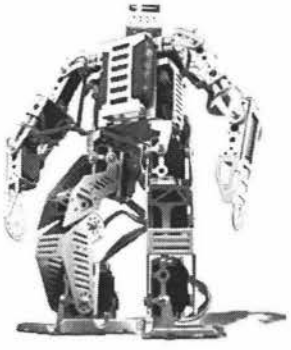


Figure 2: Pirkus – R Type 01 DX

Sensor feedback from the Pirkus consists of two gyroscopes for the pitch and roll axes and a 3 axis accelerometer for stability control. It is controlled via an Atmega 128 micro controller that runs at 16 MHz and is capable of 16MIPS. The ability to be programmed in C allows for easy modification of the controller's code, allowing effective control of the motion and task handling to be developed for the robot. The Pirkus can be interfaced with a Computer, using either Bluetooth with speeds up to 115.2Kbps or serial RS232 at 57600Kbps.

2.3 Camera

The camera module chosen for this project is the CMUcam2 produced by Carnegie Mellon University [6],[7],[8]. The CMUcam2 can control 5 servos, therefore making it a viable option for this project because of the complexity required of the vision system, and the large number of servos that make up the Pirkus. Other features present in the CMUcam2 include a master and slave mode capability, which will be described later in the paper. One of the main advantages of this system is that the controller board and camera cost $179US, making it affordable. The camera module contained in this package is the Omnivision OV6620. The camera has an analogue video output and a resolution of 352 x 288. The resolution is unimportant in this application since the CMUcam2 only has a maximum of 176 x 255 onboard resolution. The OV6620 also allows a lower resolution mode of 88 x 143 which could prove useful for real-time applications since there will be less pixels for the vision algorithms to process, allowing more frames per second, along with decreasing the amount of information to be sent via the serial port. The maximum refresh rate is 50fps (frames per second) which is ample for our application.

The control board of the CMUcam2 uses an Ubicom SX52, which operates at 75 Mhz. It has 262 bytes of SRAM and a 4096 word flash programmable EEPROM. The Control board can be communicated to via RS232 serial, The Controller board has 5 servo outputs and 4 I/O ports.

The camera is initially set up for RGB colour but can also be set to YUV colour space (Y = luminance, U = blue chrominance, V = red chrominance). This is also known as the YCbCr colour space where Y is the intensity of the image and UV represents the red and blue chrominance of the colour. The YUV colour space is more appropriate for this application because the lighting conditions will not affect the colour segmentations for the robot as the intensity component can be set to a threshold and the colour processing done only on the U and V components.

A limitation of CMOS cameras is that the colour channels are between 16 and 240 instead of the full 256 for each colour. For the application of robot soccer this will not cause significant problems since the colour space will most likely be reduced to less than 28 colours. The major limitation of this is that it will restrict the colour band of high and low colours. In testing, results led to the orange of the soccer ball, for example, having the values V = 240 and U = 16, which were the same values as that of a red object. The camera would therefore have difficulty distinguishing between red and orange objects.

Another impressive feature of the CMUcam2 that will be used for the project is that a single camera can have multiple controller boards linked together in a master/slave setup. This allows parallel image processing to be done on the cameras' images. This will prove useful in the case of Robocup where one controller can be responsible for ball location and the other for localization of the robot.

The CMUcam2 has some useful built-in image processing tools including colour tracking, frame differencing and histogram analysis.

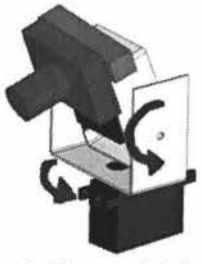


Figure 3: New neck joint design

The camera will be mounted on the Pirkus' head servo. An extra servo will also be added to allow the neck to tilt in the vertical axis. This will allow full tilt and pan of the camera as shown in figure 3. The control boards will be housed on the torso of the Pirkus and connected via a ribbon cable to the camera board. The housing for the control boards must be designed so it can be attached to the Pirkus while still keeping the Pirkus within the required dimension ratios as stated in the Robocup rules [1].

3 Implementation of Autonomy through Vision

The vision system being developed in this project includes real-time image processing, task trees and software to interface the camera and control the robot.

The initial goal of this project is to get the robot to track a ball and move towards it. This can be taken care of via the CMUcam2 built-in firmware. The next objective is for the robot to locate the goal and position itself around the ball allowing it to take a shot at goal. The use of the colour tracking function of the CMUcam2 will be used for this task.

Since the CMUcam2 can perform colour tracking and control up to five servos this functionality will be used for ball location and tracking. This control will be implemented via the Pirkus' newly developed neck servos for pan and tilt of the camera. This neck will permit the Pirkus to track the ball at all times, allowing the body to follow the head's bearing using feedback from the digital servos. The CAD model of the initial test rig design of the new head can be seen in figure 3 above. The new neck design with its pan and tilt functionality will allow tracking of an object such as a ball in real-time. The real-time neck will allow the body to complete its current static motion before adjusting its trajectory to match that of the head. This will prove rather effective in the soccer application since the ball is not likely to remain in one location for an extensive period of time.

The Pirkus' static motions are stored on its controllers EPROM and can be accessed through serial communication via its C program. The information processed via the Pirkus' controller boards will be communicated over serial to a PC. The programming on the PC will be programmed using visual basic, which is an interrupt based language and will suit the task based system in hand. The visual basic program contains the intelligence of the robot and will have to be streamlined to run on a PDA for the robot to have all its processing completed onboard to comply with the Robocup rules.

The RGB colour space can have large colour value changes with intensity. Since the CMUcam2's YUV colour space will be used, a threshold for the intensity will be implemented to ignore anything that is too white or black allowing all colour processing to be done on the chrominance of the image. This dramatically reduces lighting effects caused by the intensity of an image.

The CMUcam2 detects objects by using RLE (Run Length Encoding) Algorithms [9] on the image, which has been utilized by the Carnegie Mellon Universities legged soccer team, using the Sony

AIBO platform. This process makes a horizontal run of the image, linking up the connected colours over two runs. Each colour grouping is then listed in order of size. Small neighbouring regions are finally grouped together to reduce noise.

Fast and slow systems, as commonly used by the armed forces, will be utilised. The implementation will include 2 vision systems. The first is the slow system which will require a wide field of view and will act like the robots peripheral vision. A monoscopic vision system utilising a wide angle or omni directional lens will supply the wider field of view required. Since the slow system has a wide view it can pick up all the objects surrounding the robot and feedback any objects of importance to the fast system. The fast system is a more focused system that will have a more narrow view angle and focus on only a single item at a time. By focusing on a single item at a time the response time and accuracy of the robot for a required task will be improved since the amount of data required for processing will be decreased.

The walking motion of the robot for the system in question requires several static walking motions. The first will require the robot to walk in a straight line at 2 different speeds. A slower speed will be required for when the Pirkus gets near the ball for its kicking motion. Along with the straight walk, a curved walking motion is necessary for slight corrections to the path required to be travelled. Finally a stop and turn and a strafing motion are required for sharp turns and positioning of the Pirkus around the ball and goal.

3.1 Vision Types

One of the dilemmas facing vision system designers is whether to use monoscopic or stereo vision. Besides the obvious questions of cost, little research has been done in determining which system performs more effectively under specific conditions. This section develops a mathematical model for determining the effectiveness of stereo and monoscopic vision. It discusses the benefits and downfalls of both systems, focusing on a small humanoid robot platform for robot soccer.

3.1.2 Monoscopic Vision

Since monoscopic vision uses only one camera, its depth perception is entirely based on knowing the size of the object being viewed. Therefore, for a monoscopic colour vision system, each colour would have to be assigned to an object of a known size to allow their distances from the camera to be calculated.

For an object size (O_S), resolution (R), and view angle of the camera (V), the distance from the camera to the object (X) can be calculated. The distance per pixel (D_P) can be calculated via the object size in millimetres (mm), and the number of pixels of the object (O_P) being examined. This is shown in equation 1. The value calculated can be multiplied by the respective resolution to give the horizontal and vertical field of view distances (F_D) in mm as in equation 2. By dividing the triangle contained by the field of view angle of the camera (V), and the horizontal or vertical distance Fd in mm by half, Pythagoras' theory can be used to calculate the distance of the object from the camera (X) with the equation 3. Since X is only the distance from the ball to the camera, the height from the ground to the camera (H) must be added. Figure 4 Shows X is the hypotenuse of the triangle; therefore the distance of the robot from the object (O_D) is calculated from equation 4.

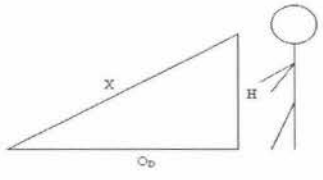


Figure 4: Calculating the robot's distance for the ball.

$$D_P = \frac{O_S}{O_P} \quad (1)$$

$$F_D = RD_P \quad (2)$$

$$X = \frac{F_D 2}{\tan(V 2)} \quad (3)$$

$$O_D = \sqrt{X^2 - H^2} \quad (4)$$

The next step is to include the systems resolution error. The two areas to look at are how far the distance will vary with the object being a pixel value (O_P), and how far the distance will vary for the image reaching the next pixel size. As seen in figure 5, as the object approaches the camera, the image size has to increase a specific pixel increment, therefore the Pirkus has to move a certain distance before the image increases to the next number of pixels.

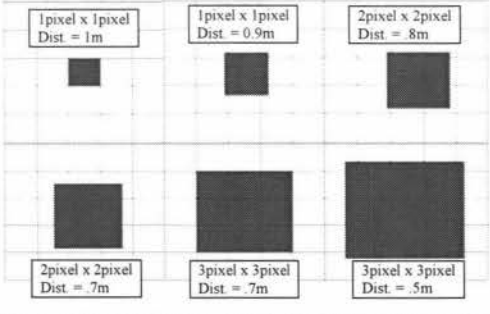


Figure 5: Resolution error of monoscopic vision

The description for the magnitude of change for the object's distance depending on the object's pixel value is as follows. The error is found by halving the difference between the calculated values for the current O_P value and the O_P before and after, as shown in equation 5 where X_0 is the distance from the object to the Robot at that particular number of pixels

and X_1 and X_{-1} are the calculated distance to the robot at plus or minus one pixels in the objects size. From the formula the plus or minus distance from the object's actual distance is calculated, giving the range within which the object can be located (E_0). The horizontal and vertical values are then being used together to give greater accuracy by finding the average calculated O_P. By using the smallest maximum, and largest minimum E_0 of the horizontal and vertical axis, the range for a particular pixel value is decreased. Using the above calculations, a ball with an 80mm diameter (regulation Robocup humanoid kid size soccer ball), and a 176x255 resolution camera 30cm off the ground (approximate height of the Pirkus); the E_0 error can be seen in figure 4 above. From the figure it can be concluded that the higher the resolution of the camera, the more distance and accuracy of the vision system increase, therefore supporting the use of a smaller more focused lens for the fast vision system.

$$+E_0 = X_0 + \left|\frac{X_0 - X_{-1}}{2}\right| \qquad (5)$$

$$-E_0 = X_0 - \left|\frac{X_0 - X_1}{2}\right|$$

Another error occurs if the camera is out by a single pixel (E_1), as shown in figure 6, thus increasing the range of error that the ball will fall. Equation 6 shows the calculation of this increased error.

$$+E_1 = X_0 + \left(|X_0 - X_{-1}| + \left|\frac{X_0 - X_{-2}}{2}\right|\right) \qquad (6)$$

$$-E_1 = X_0 - \left(|X_0 - X_1| + \left|\frac{X_0 - X_2}{2}\right|\right)$$

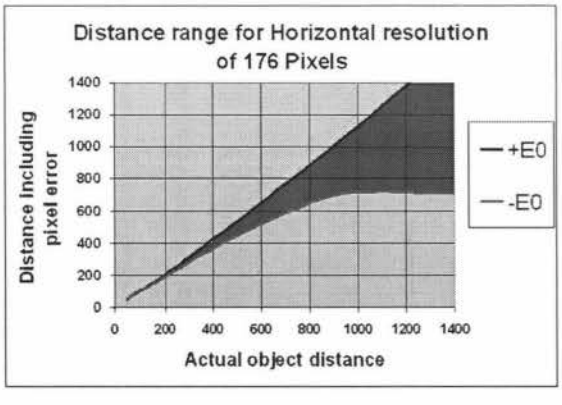


Figure 6: Distance error of monoscopic vision

3.1.3 Stereo Vision

Human vision is based on stereo vision. It is a system that allows more accurate depth perception to be calculated [10] and, unlike monoscopic vision, the object size does not need to be known.

For the calculation of stereo vision a simple case is used in which one camera looks straight at the ball while the second camera looks at the ball at an angle. The above method has been chosen for ease of calculation of the accuracy. Both cameras find the midpoint of the ball, and along with the distance between the cameras (D_c), the Pythagoras theorem can be used to calculate the distance from the cameras to the ball as illustrated in figure 7. From this formula it is possible to see the accuracy of stereo vision is proportional to the distance between the cameras. As with monoscopic vision the distance from the robot to the ball has to be calculated using equation 4. The implementation of stereo vision above may not be the most capable system to employ but allows easy calculation of depth perception efficiency for stereo vision.

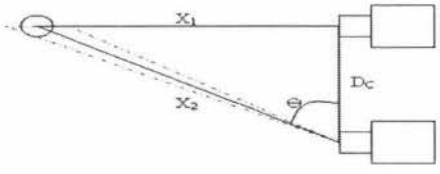


Figure 7: Stereo vision depth perception calculation.

A drawback of this system is that the added size and weight of the required apparatus affects the centre of gravity of the robot, reducing its balance because of the robot's small size.

For this system the resolution error can be examined to see the effect of the objects central pixel in the cameras image having an error of 1 or 2 pixels. Unlike monoscopic vision this error relates to the object moving vertically or horizontally rather than its size increasing. For this calculation the vertical aspect has been removed for simplification. The case of the error being 2 pixels would occur if both cameras have an error of 1 pixel in opposite directions. An advantage that stereo vision has is that, if both cameras have an error in the same direction, the error is effectively cancelled out. The error can be calculated using the known field of view of the lens (V), and horizontal resolution of the camera (R). From these values the degrees per pixel can be calculated and the result multiplied by the number of pixels of error, giving the difference from the original angle, which transforms the initial equation for (X) to equation 7. Equation 4 from the previous section must be subsequently used for calculation of the objects distance in relation to the robot. The dotted line in figure 7 shows the effect of the pixel error.

$$X_1 = D_C \tan\left(\theta \pm \left(\frac{V}{R} E_1\right)\right) \qquad (7)$$

3.2 Monoscopic versus Stereo Vision

Before a vision system can be chosen for a task, the surrounding environment must be explored. In robot soccer certain variables are known. These include the size and colour of the ball, goals, and the dimensions of the field. This allows the use of either monoscopic or stereo vision for depth perception.

The underlining question that needs to be addressed is whether the added accuracy of stereo vision for a soccer robot is worthwhile. Aspects that need to be considered include doubling the computational power required, and the need for a second camera thereby doubling the price and complexity of the system. Since a second camera would be required the added weight to the robots torso and head would raise the centre of mass of the robot affecting its ability to balance.

The next consideration that needs to be addressed is the robot's requirements for the task at hand. In the case of robot soccer, the robot needs to get to the ball faster than its competition, and then kicking it either at the opposition's goal or to a team mate. For this task the robot needs to know the location of the ball, and goal, in relation to itself, with the location of its team and the opposition also being useful to know. The accuracy of the ball location becomes more important as the robot approaches it. Therefore the greater the distance to the ball, the less the robot needs to know about its exact relative location. For the most basic artificial intelligence (AI), the robot just needs to know the general direction of the ball and to get to it as fast as possible. As it approaches the ball, the errors in resolution gradually decrease until they become negligible.

Since the cost and complexity of a stereo vision system is twice that of monoscopic vision, the informational gain for the task has to be over twice as valuable for it to be practical. From the results described in the previous section, stereo vision has substantially greater accuracy at greater distances, and knowledge of the object size is not required. But, in the case of robot soccer, since sizes are known and the accuracy becomes less important at greater distances, a monoscopic system would give sufficient information.

A monoscopic vision system will allow distance to be judged via the number of pixels taken up by the object in question if the object size is known. The camera resolution therefore becomes important, especially in relation to the height of the robot being used. The direction can be determined by the location of the object in relation to the centre of the image and by the direction that the neck servos of the robot are facing. This system is currently being used by major competitors in the Robocup [2].

This leads to the conclusion that, for a humanoid robot soccer environment, the advantages gained from stereo are minimal compared to the complexity involved in integrating them effectively with minimal error.

4 Conclusion

This paper gives an overview of a humanoid biped robot platform being developed for the Robocup soccer competition, discussing the hardware and software systems in use and the development of new systems that are being implemented.

It then discusses and compares monoscopic and stereo vision systems and develops a set of mathematical models to determine the effect of any error caused by camera resolution. These errors occur because the image size can only increase to the next pixel size when it has moved within enough of a range to be detected by the resolution of the camera.

In discussing monoscopic and stereo vision systems in the context of robot soccer, it is concluded the monoscopic systems are usually well suited for the task. Their large error at long distances is not critical because, at that stage, the robot only needs a general direction to head in. As the robot approaches the ball, the error reduces to a level that makes it negligible.

5 References

[1] The Robocup Federation Website, "Home Page", http://www.robocup.org/, visited on 6/5/2006.

[2] S. Behnke, J. M¨uller, M. Schreiber. "Using Handheld Computers to Control Humanoid Robots", *Proceedings of 1st International Conference on Dextrous Autonomous Robots and Humanoids.* Freiburg, Germany, 2004.

[3] S Behnke, T Langner, J Muller, H Neub, M Schreiber. NimbRo RS: A Low-Cost Autonomous Humanoid Robot for Multi-Agent Research. *Proceedings of Workshop on Methods and Technology.* Freiburg, Germany, 2004.

[4] Robosapien Website, "Home Page", http://www.evosapien.com, visited on 20/7/2006.

[5] S.S. Kato, T. Ishizuka, H. Takahashi, Y. Uchibe, E. Asada, M. "An Application of Vision-Based Learning in RoboCup for a Real Robot with an Omnidirectional Vision System and the Team Description of Osaka University "Trackies"", *Lecture Notes in Computer Science*, Vol. 1604, pages 316-325, 1999.

[6] A. Rowe, C. Rosenberg, I. Nourbakhsh, CMUcam Website, "Home Page", http://www. cs.cmu.edu/~cmucam/, visited on 20/8/2006.

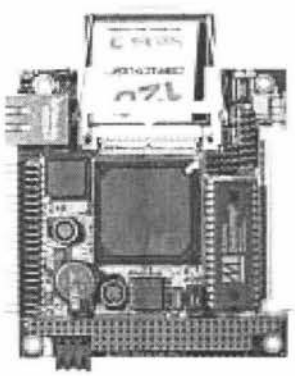
[7] A. Rowe, C. Rosenberg, I. Nourbakhsh, "A Second Generation Low Cost Embedded Color Vision System", *The Proceedings of IROS*, 2005.

[8] Omnivision Technologies Incorporated, "OV6620 Single-Chip CMOS CIF Color Digital Camera Technical Documentation", "Home Page" http://www.ovt.com/ visited on 20/4/2006.

[9] J. Bruce, T. Balch, M. Veloso, "Fast and Inexpensive Color Image Segmentation for Interactive Robots", *Proceedings of IROS 2000*, 2000.

[10] S. Florczyk, "*Robot Vision: Video-based Indoor Exploration with Autonomous and Mobile Robots*", Weinheim, John Wiley & Sons, 2005.

486/586 PC/104 Computer with CompactFlash and Flat Panel SBC1495



Features

- ✓ Ready to run 486/586 computer
- ✓ 120 or 133MHz
- ✓ CRT and flat panel output
- ✓ 64MB SDRAM
- ✓ CompactFlash connector
- ✓ 10/100BASE-T Ethernet
- ✓ Two serial ports
- ✓ Extended temperature available

The SBC1495 packs a fast 486DX processor with plenty of memory, and copious amounts of storage into a PC/104-sized board. Both CRTs and color TFT flat panels are supported by the built-in VGA interface. Additional I/O includes six LVTTL digital I/O lines, dual serial ports, USB, EIDE, LPT, keyboard, and mouse.

In its stackthrough version, the SBC1495 is an ideal computer to plug into a custom OEM I/O card. Immediately and easily, an advanced engine is available for software development.

With 1MB of on-board flash, accessible as a read/write disk, and 64MB of SDRAM, many large programs can be run. However, if additional storage capacity is required, the CompactFlash connector allows hundreds of megabytes of removable program and data storage.

If additional capabilities are needed, PC/104 expansion allows a wide variety of I/O cards to be stacked on the SBC1495.

Software Support

DOS emulation
MSDOS 5.0
Linux
Windows CE
RTOS
Comm Library, CommBLOK™
PID loop library, PidBLOK™
C, compilers
[Items above in Section 6]

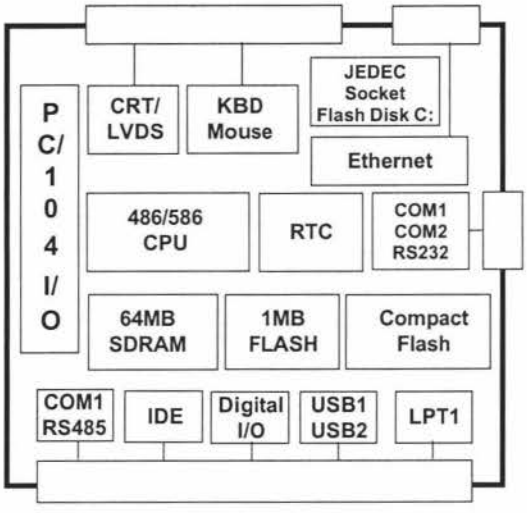
Compatible Hardware

PC/104 expansion cards
[Items above in Section 4]
RS232/RS485 devices
Custom

Mounting/Packaging

Standoffs, STDOFF01
[Items above in Section 5]
Custom

Micro/sys 3730 Park Place, Montrose, CA 91020 (818)244-4600 Fax (818)244-4246
email: info@embeddedsys.com



Technical Details:

The SBC1495 core is an ST Microelectronics STPC Atlas processor running at 120 or 133 MHz. The STPC 486DX processor core is clocked at a rate of 133MHz, and includes hardware floating point math. While other 486DX systems access RAM with a 32-bit data bus, the Atlas accesses RAM with a 64-bit wide data bus, offering performance similar to low-end Pentium-based designs.

The Atlas allows compatibility with both real mode and 32-bit protected mode programs. The Atlas also integrates many PC-compatible peripherals. Dual USB ports, a keyboard and mouse controller, an EIDE controller, two cascaded 82C59A interrupt controllers, dual 16C550 UARTs, three timer/counters (82C54 compatible), and a dual DMA controller are all present. A hardware accelerated VGA controller, with support for both CRTs and TFT panels, is also implemented.

The memory subsystem on the SBC1495 allows many programs to be run without any external storage. 64 Mbytes of synchronous DRAM (SDRAM) is more than sufficient for many complex, protected-mode programs and operating systems.

The 1-Mbyte Flash memory chip contains both the BIOS and a user application code space. The user space can be configured as a 768k read/write flash disk.

If a larger program or data storage space is required, or if removability is needed, the CompactFlash interface can provide hundreds of megabytes of storage. CompactFlash is used in the True IDE mode, where it is register compatible with an EIDE hard drive. Thus, it does not require any special drivers for most operating systems.

Micro/sys 3730 Park Place, Montrose, CA 91020 (818)244-4600 Fax (818)244-4246
email: info@embeddedsys.com

The user byte-wide socket can accept a number of different devices. EPROM, 5v Flash, DiskOnChip®, or SRAM can all be plugged in. The SRAM can be battery-backed, which makes for fast storage for data that is updated often.

The VGA controller supports resolutions up to 1280 x 1024 (CRT) or 1024 x 1024 (programmable panel). It includes hardware acceleration for fast graphic updates. The output can drive a standard RGB CRT monitor, and an LCD flat panel display. Active matrix (TFT) LCD panels are supported, in 18-bit color. The LVDS interface is compatible with many displays and ensures that the signal integrity is maintained.

Two serial ports allow communication with many different devices. COM1 and COM2 are 16C550-compatible UARTs (with transmit and receive FIFOs). These serial ports are capable of speeds up to 115200 baud, have RS-232 transceivers, and have RTS and CTS modem control lines. Additionally, COM1 is configurable for half-duplex RS-485 communication with jumperable termination resistors.

The PC/104 connector provides support for both 8-bit and 16-bit expansion boards and operates with standard PC/104 bus protocol and timing. The default configuration is non-stackthrough connectors, allowing the SBC1495 to be the bottom card in a stack. The stackthrough option (SBCOPT16ST) allows the SBC1495 to be plugged into a custom-designed OEM I/O board as an automation component.

The SBC1495 can support application development under numerous strategies. If 16-bit DOS or DOS-extended software is sufficient, Micro/sys offers a free DOS-compatible operating system preinstalled on the SBC1495. For a small royalty fee, true MSDOS 5.0 can be preinstalled. Powerful, cost-effective remote debug capabilities are provided through Borland's Turbo Debugger.

For true 32-bit application development, the SBC1495 supports a number of alternatives. Due to its PC compatibility, 32-bit real time operating systems (RTOS) such as PharLap® ETS, and VxWorks® can be booted on the SBC1495. All support 32-bit linear protected mode operation, and have full tool suites available, including compilers and debuggers.

The firmware suite that is preinstalled in flash on the SBC1495 includes an industrial BIOS that allows configuration of many of its features. In addition to allowing configuration of the normal PC-compatible peripherals such as floppy drives and hard drives, it allows 768k of the system flash to be used as a read/write wear-leveled flash drive. Another feature of the BIOS is its ability to redirect the console out COM1, COM2, or the VGA/keyboard so that even "headless" systems can have a user console when needed for configuration or debug.

For pre-configured sets of options, Micro/sys can provide OEMs with a single part number for ordering. In addition, custom versions of the SBC1495 are available. Please call Micro/sys Technical Sales for details.

Specifications:

Mechanical:

- PC/104 standard
- 3.55" (plus I/O region) x 3.775" x .6"
- Installed CompactFlash card extends past edge of board opposite the PC/104 connector
- If installed, Ethernet connector on top side has height of .535"

Power Requirements:

- +5v ±5% at 1.3A typical, 1.8A max.
- +12v required only if used by PC/104 modules

Micro/sys 3730 Park Place, Montrose, CA 91020 (818)244-4600 Fax (818)244-4246
email: info@embeddedsys.com

Environmental:

Part number	*Board Airflow**	*Operating Temp*
SBC1495-1	0 cfm	0° to +48°C
SBC1495-1	17 cfm	0° to +70°C
SBC1495-ET	0 cfm	-40° to +85°C
SBC1495-1-ET	0 cfm	-40° to +85°C

*Using 80mm fan

- ❑ -40° to +85°C storage
- ❑ 5%-95% relative humidity, non-condensing

Processor Core Section:
- ❑ STPC Atlas CPU
- ❑ 120 or 133 MHz clock rate
- ❑ Hardware floating point math
- ❑ AT-compatible timers, interrupts, DMA

On-board Memory:
- ❑ 64MB Synchronous DRAM based at 0
- ❑ 1M of Flash at top of memory map with BIOS and operating system installed; 768k available for user application
- ❑ JEDEC 32-pin socket for 128k/512k SRAM for battery-backed RAM, or DiskOnChip®

Watchdog Timer:
- ❑ Program must refresh watchdog timer periodically, or system will be reset
- ❑ Enabled through software

Keyboard, Mouse, and Speaker:
- ❑ PS/2-compatible keyboard port
- ❑ PS/2-stype mouse port
- ❑ AT-compatible TTL speaker output

SVGA Video Output:
- ❑ CRT and color LCD outputs
- ❑ Resolutions to 1280 x 1024 (CRT) or 1024 x 1024 (programmable panel)
- ❑ Direct connect to TFT flat panels
- ❑ 3.3V 18-bit panel color support
- ❑ LVDS (PanelLink/FPD-Link) drivers

COM1-COM2 Serial Ports:
- ❑ Two async serial ports, PC compatible
- ❑ 16550-compatible
- ❑ RTS and CTS modem controls
- ❑ RS232 on both channels
- ❑ COM1 RS485 half duplex

Serial Port Connector			
Pin	Signal	Signal	Pin
1	RX COM1	RTS COM1	2
3	TX COM1	CTS COM1	4
5	-	-	6
7	GND	RX COM2	8
9	RTS COM2	TX COM2	10
11	CTS COM2	-	12
13	-	GND	14

Digital I/O:
- ❑ Six LVTTL bi-directional signals
- ❑ 5v-tolerant

User Interface Connector			
Pin	Signal	Signal	Pin
1	GND	TXCLK+	2
3	TXCLK-	GND	4
5	TXOUT2+	TXOUT2-	6
7	GND	TXOUT1+	8
9	TXOUT1-	GND	10
11	TXOUT0+	TXOUT0-	12
13	GND	GND	14
15	TFT VCC	TFT VCC	16
17	TFT PWM	TFT EN5V	18
19	GND	GND	20
21	MOUSE CLK	MOUSE DTA	22
23	+5V	+5V	24
25	KBD DTA	KBD CLK	26
27	SPKR	-	28
29	-	I2C CLK	30
31	I2C DTA	HSYNC	32
33	GND	VSYNC	34
35	GND	BLUE	36
37	GND	GREEN	38
39	GND	RED	40

Parallel Port:
- ❑ Bi-directional LPT standard

Micro/sys 3730 Park Place, Montrose, CA 91020 (818)244-4600 Fax (818)244-4246
email: info@embeddedsys.com

Main I/O Connector			
Pin	Signal	Signal	Pin
A1	GND	IDE RESET#	B1
A2	USB D0-	GND	B2
A3	USB D0+	IDE D7	B3
A4	USB VCC	IDE D8	B4
A5	GND	IDE D6	B5
A6	USB D1-	IDE D9	B6
A7	USB D1+	IDE D5	B7
A8	USB VCC	IDE D10	B8
A9	GND	IDE D4	B9
A10	GPIO0	IDE D11	B10
A11	GPIO1	IDE D3	B11
A12	GPIO2	IDE D12	B12
A13	GPIO3	IDE D2	B13
A14	GPIO4	IDE D13	B14
A15	GPIO5	IDE D1	B15
A16	GND	IDE D14	B16
A17	-	IDE D0	B17
A18	GND	IDE D15	B18
A19	RS485+	GND	B19
A20	RS485-	-	B20
A21	+5V	IDE DRQ	B21
A22	LPT STB#	GND	B22
A23	LPT AFD#	IDE IOW#	B23
A24	LPT D0	GND	B24
A25	LPT ERR#	IDE IOR#	B25
A26	LPT D1	GND	B26
A27	LPT INIT#	IDE IORDY	B27
A28	LPT D2	GND	B28
A29	LPT SLIN#	IDE DACK#	B29
A30	LPT D3	GND	B30
A31	GND	IDE IRQ	B31
A32	LPT D4	IDE IO16#	B32
A33	LPT D5	IDE DA1	B33
A34	LPT D6	IDE PDIAG#	B34
A35	LPT D7	IDE DA0	B35
A36	LPT ACK#	IDE DA2	B36

Real Time Clock:
- ❑ RTC with on-board battery
- ❑ Driver software in BIOS

PC/104 Interface:
- ❑ Non-stackthrough PC/104 connectors
- ❑ Standard mounting holes
- ❑ 8-bit and 16-bit PC/104 module support
- ❑ Full IRQ and DRQ support
- ❑ Stackthrough option available (SBCOPT16ST)

CompactFlash Interface:
- ❑ Supports Type I CompactFlash
- ❑ Operates in True IDE mode
- ❑ CF+ cards not supported
- ❑ Not hot-swappable

Power Connector	
Pin	Signal
1	+5V
2	+12V
3	GND

DK1495 Development Kit:
- ❑ Free with first SBC1495 purchase
- ❑ Breakout cable to COM1-COM2
- ❑ Breakout cable to IDE, USB, LPT, digital I/O
- ❑ Breakout cable to CRT, keyboard, mouse, speaker
- ❑ Download cable and utilities
- ❑ Documentation, schematics, sample software

External Connections:
- ❑ 80-pin connector for IDE, USB, LPT, and digital I/O
- ❑ 14-pin header for COM1-COM2
- ❑ 40-pin header for CRT, flat panel, keyboard, mouse, speaker
- ❑ 3-pin removable terminal strip for power input

Ordering Information:

Single Board Computer:

SBC1495	486/586 CPU, 133MHz, 64MB RAM, 1M Flash
SBC1495-1	486/586 CPU, 133MHz, 64MB RAM, 1M Flash, 10/100BASE-T Ethernet

Micro/sys 3730 Park Place, Montrose, CA 91020 (818)244-4600 Fax (818)244-4246
email: info@embeddedsys.com

SBC1495-ET	486/586 CPU, 120MHz, 64MB RAM, 1M Flash, -40 to +85C operating temperature
SBC1495-1-ET	486/586 CPU, 120MHz, 64MB RAM, 1M Flash, 10/100BASE-T Ethernet, -40 to +85C operating temperature
DK1495	No charge development kit, available with first order only
SDK Linux	LInux Kit (requires Ethernet and 1495OPT50)
1495OPT25	MSDOS 5.0 in Bootable A: Flash Disk
1495OPT28	Color TFT (LVDS) panel support
1495 OPT50	Linux startup kernel installed in flash

Related Products:

CA4089	Breakout cable to two DB9 COM port connectors
CA4097	Breakout cable for EIDE, USB, LPT, Digital I/O
CA4098	Breakout cable for CRT, Kbd, mouse, speaker, TFT panel
RAM128	128k RAM device
RAM512	512k RAM device
SBCOPT16ST	Stackthrough PC/104
CF-FL128	128MB CompactFlash Card
CF-FL256	256MB CompactFlash Card
CF-FL512	512MB CompactFlash Card

Cables nominally 15", other lengths available

Micro/sys 3730 Park Place, Montrose, CA 91020 (818)244-4600 Fax (818)244-4246
email: info@embeddedsys.com