The Development of a Low-Cost Robotic Visual Tracking System

A thesis presented in partial fulfilment of the requirements for the degree of

Master of Engineering

Mechatronics

at Massey University, Albany, New Zealand.

Shun Fan Chiang

2009

ACKNOWLEDGMENT

I wish to express my gratitude to my supervisor, Dr. Johan Potgieter, who was abundantly helpful and offered invaluable assistance, support and guidance throughout the duration of the project.

Special thanks also to all my graduate friends, especially Biqing Chen and Baden Rielly for sharing the literature and invaluable assistance. Not forgetting to my best friend Jennifer who always been there.

I wish to express my gratitude to my beloved families, for their understanding and encouragement through my studies, in particular I would like to thank my parents for their love and patience, my brother Vince and my cousin Kevin, without their assistance this project would not have been made possible.

By The Author
Shun Fan Chiang

ABSTRACT

This thesis describes a system which is able to track and imitate human motion. The system is divided into two major parts: computer vision system and robot arm motion control system. Through the use of two real-time video cameras, computer vision system identifies the moving object depending on the colour features, as the object colour is matched within the colour range in the current image frame, a method that employs two vectors is used to calculate the coordinates of the object. After the object is detected and tracked coordinates are saved to a pre-establish database in the purpose of further data processing, a mathematical algorithm is performed to the data in order to give a better robotic motion control. Robot arm manipulator responds with a move within its workspace which corresponds to a consequential human-type motion. Experimental outcomes have shown that the system is reliable and can successfully imitate a human hand motion in most cases.

Table of Contents

List of Figures

List of Tables

1. INTRODUCTION

As human beings, we communicate with our physical world through vision, touch, and sound. We perform our daily tasks through the use of our hands. However, in an effort to change this, people have been designing, building, and analysing ways of using modern technology to imitate human motion [Sturman].

Advances in technology and the use computation technologies have helped to develop object detection and imitating systems become more affordable.

Repetition visual tracking of objects and image features is a challenging task with many potential practical applications. Real-time three-dimensional tracking of a moving object is an important requirement for many growing industry applications. Visual tracking techniques can be employed in many areas from the military and surveillance, through to medicine and the film and special effects industries.

By using the most recent technologies, objects can be detected, tracked, recorded, imitated and applied to develop an object tracking imitation system. The objective of this project is to research and build a robust system which can locate and track a moving object, for example a human hand, in a three dimensional coordinate space system accurately.

1.1 Project background

This thesis introduces a camera base motion analysis approach to describe a visual tracking system that tracks a hand motion of human being. The goal of this project is to develop and operate an accurate and robust tracking system for use in an augmented reality application. Augmented reality combines computer graphics and virtual-reality displays with images of the real world. A hardware system and method of tracking are going to be discussed, and a coordinate system which uses to direct the hand movement is going to be developed for further robotic imitation application.

For the purpose of interpreting a motion movement, an object tracking system had been developed and introduced to the industry many years ago by using mechanic and electronic technologies.

In 1970, researchers in MIT developed a general-purpose computer system that input the interpretation of hand motion directly by using 3-space tracking sensors. The system is now improved and widely used, by emitting a pulsed magnetic field (from origin point), the cooperate sensors attached on the moving objects response their position and orientation relate to the origin, and the information were used for a further graphic application [Sturman].

1.2 Object detecting and tracking technologies

Object positioning is characterized by the location of the object in space and relate the coordinates to the orientation system. Three technologies are used predominantly to track the position of an object. Magnetically based, uses a magnetic field radiating source and sensors that reports the position and orientation with respect to the origin source; acoustically based, using triangulation of ultrasonic “pings” to locate the object; or optically based, using cameras to examine the object from a distance [Sturman].

1.3 Glove technologies

Tracking single object alone has received considerable attention, tracking multiple objects simultaneously can be more useful, however more problematic. It is more useful because objects we want to track often exist in close proximity to other similar objects. It is more problematic because the objects of interest can touch, occlude, and interact with each other; they can enter and leave the image; and we must be able to tell them apart. In addition, multiple object tracking deal with all the hard problems of single object tracking, including running at a reasonable rate and adapting to changing background conditions [Multi Object Tracking].

The development of electronic glove technologies has been tried to overcome problems of detecting hand motion and uses as interface to computer applications. Glove devices measure the shape of hands and uses electronic device to discriminate between the fingers and palm [Sturman].

Over the past decade, many researchers have built hand measuring devices for computer input. The differences between them are the makers and material of glove been used for tracking.

1.4 Robot arm manipulating system

There are many kinds of manipulation system in the industry, to choose a suitable robot arm to accomplish the goal is the primary task. Most robot arm manipulators are operating in two-dimension and three-dimension space, and it is determined by the degree-of-freedom.

2-D	Translation only	2-DOF
2-D	Translation and orientation	3-DOF
3-D	Orientation only	3-DOF
3-D	Translation and orientation	6-DOF

Table 1.1 2-D and 3-D robot arm manipulator

An example of 2-D robot is the mobile robot that moving on the floor, and a manipulator robot arm doing car painting is an example of 3-D robot.

1.5 Objective and contribution

The final task of this project is going to be the interaction of a computer vision system with an actuating robot arm, in order for the manipulator arm to move through the path direct by a human being and without harming the environment.

A complete system allows the users to control the robot through the path they intend to, in advance applications, in certain dangerous or complex industry, some manual jobs can be replaced by the operating system. With an additional on-line system which enables the robot to be controlled remotely in real-time or off-line.

2. LITERATURE REVIEW

2.1 Introduction

Before starting to design and build a computer vision - aid arm imitation robot system, related works and applications of human imitation system need to be investigated and understood. A successful system can be divided into two regions of study, capturing and moving. Therefore, detailed knowledge of computer vision system concepts in detecting and tracking such as image segmentation, colour space, colour transformation, and image processing are essential. Moreover, how to sense a moving human body and what algorithm and technique to be used must be described. In more detail, it is important that the computer vision system is operating under a consistent coordinate system with the environment, hence, an explanation of how the computer vision coordinate system relating to three dimension real world environment and the executing robot arm is going to be mentioned.

In addition to the above information, a basic concept of the robot and controlling the arm manipulator to act as a human arm in a steady state with the given trajectories is addressed in the second part of this chapter.

2.2 Computer vision system

Computer vision system has reached a level of maturity that allows us to not only delve into the field of computer vision, but also expand the research territory which leads to the building of a further robust integrated system of significant complexity.

Computer vision automates and integrates a wide range of processes used for vision perception. It includes many techniques and algorithms that are useful by themselves, such as image processing (transforming, encoding, and transmitting images) and statistical pattern classification (statistical decision theory applied to general patters, visual or otherwise). Moreover, it also includes techniques for geometric modelling and cognitive processing [Sebe]. These progresses provide mathematical information for the use of images to reconstruct and model the environment.

Computer vision is not like human vision. It is not affected by the visual band or electromagnetic spectrum. Many computer vision applications involve tasks that require either work in a hostile environment, a high rate of processing, access and use of large databases of information, or are tedious for people to perform. Computer vision systems are used in many and various types of environments, for example, in manufacturing systems, computer vision is often used for product quality control.

2.2.1 Image segmentation

Image and video segmentation is one of the most critical tasks of analysis which has the objective of extracting information from an image or a sequence of images. The field of image and video segmentation had great advancement and progress especially in recent years.

According to structure of image engineering (IE) framework, image techniques and algorithms can be organized into three levels: image processing (low level), image analysis (mid level) and image understanding (high level), as shown in figure 2.1. Low level image operation includes pre-processing works such as image noise reduction or contrast adjustment, yet, the inputs and outputs are both images. In the mid level, process of segmentation and classification of object are involved, in addition, usually the inputs are images, and outputs are the characteristic extract from the input image (edges, contours). A high level processing that employs the data from previous operations to perform a function of image which associate with computer vision, for example, object recognition [Zhang].

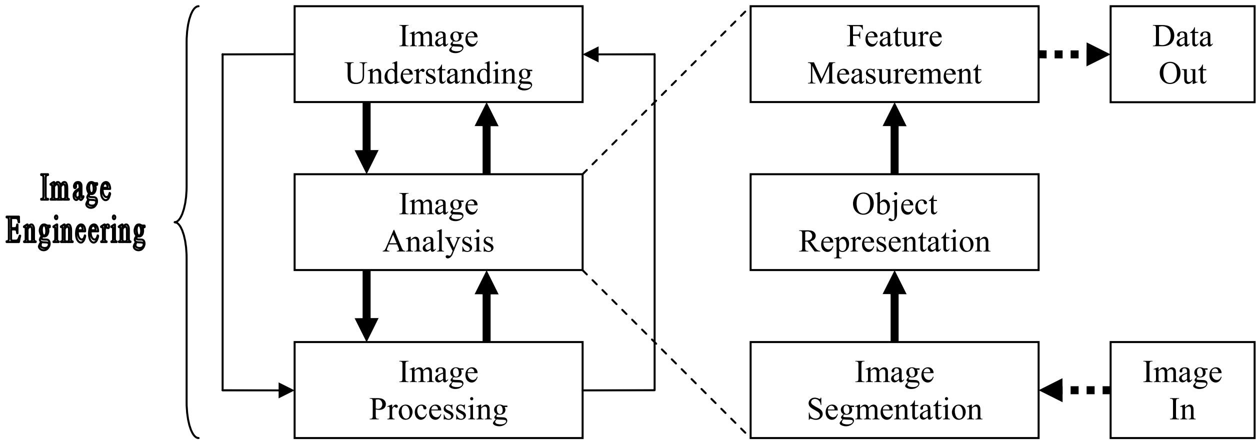


Figure 2.1 Image Engineering and image segmentation [Zhang]

As shown in figure 2.1, image segmentation is the primary and most important step of image analysis procedures. Purposing of image analysis is intended to retrieve the information which interesting (object) through the process of image segmentation, object representation and feature measurement. On the other hand, through the process of partition an image into regions that allows to perform a task such as objects recognition in post-processing. Since image segmentation is first step of analysis, this part of process become extremely important because it is affecting the accuracy of following procedure object representation and also the outcome of feature measurement.

Considering image segmentation as the partition of an image into a set of non-overlapping regions whose union is the entire image, some rules to be followed for regions resulting from the image segmentation can be stated as:

1. They should be uniform and homogeneous with respect to some characteristics;
2. Their interiors should be simple and without many small holes;
3. Adjacent regions should have significantly different values with respect to the characteristic on which they are uniform; and
4. Boundaries of each segment should be simple, not ragged, and must be spatially accurate.

The technique of evaluation of image segmentation can be categorized in to two types: characterization and comparison. A formal definition of image segmentation, supposing the whole image is represented by R and Ri, where $i = 1,2,\ldots,n$ are disjoint non-empty regions of R, consists of the following conditions:

1. $\bigcup_{i=1}^{n} Ri = R$;
2. for all i and j, $i \neq j$, there exits $Ri \bigcap Rj = \varnothing$;
3. for $i = 1, 2, ..., n$, it must have $P(Ri) = TRUE$;
4. for all $i \neq j$, there exits $P(Ri \bigcup Rj) = FALSE$;
5. for all $i = 1,2,\ldots,n$, Ri, is a connected component.

where $P(Ri)$ is a uniformity predicate for all elements in set Ri and $\varnothing$ represents an empty set [Zhang].

From the above, five conditions can be described in more detail as below: the region of segmentation could be all pixels from an image, but each region can not overlap with the others, pixels are connect with each other in within same segmented region, and they should contain similar properties or information, in other words, pixels in different region of segmentation should have different properties.

With several of segmentation algorithms been developed, a classification of techniques can be made in six groups:

1. Thresholding
2. Pixel classification (including relaxation, Markov random field based approaches and neural network based approaches)
3. Range image segmentation
4. Colour image segmentation
5. Edge detection
6. Methods based on fuzzy set theory (including fuzzy thresholding, fuzzy clustering and fuzzy edge detection) [Zhang].

Discontinuity and similarity are two properties to use while segmenting a grey level image. Two types of algorithms can be classified, edge-base use the property of boundaries to detect object, region–base use property of object areas. However, based on the process strategy, the algorithms can be partitioned into parallel and sequential process [Zhang]. Combining the properties from above, four groups of algorithm techniques are defined in table 2.1:

Classification	Edge-based (discontinuity)	Region-base (similarity)
Parallel process	G1: Edge-based parallel process	G3: Region-base parallel process
Sequential process	G2: Edge-based sequential process	G4: Region-base sequential process

Table 2.1 General classification of segmentation algorithms [Zhang]

2.2.1.1 Colour image segmentation

Segmentation is a process that partitions an image into regions of interest. Although there are many algorithms of doing colour image segmentation with different colour model, the one that produces the most accurate result is chosen. Two algorithms to segment basic colour models are going to introduce as follow.

The segmentation of an object from the image of RGB colour model is performed by specifying a colour range in an RGB image, in the other words, the pixel is satisfied to the colour criterion must carry the RGB values within the colour range.

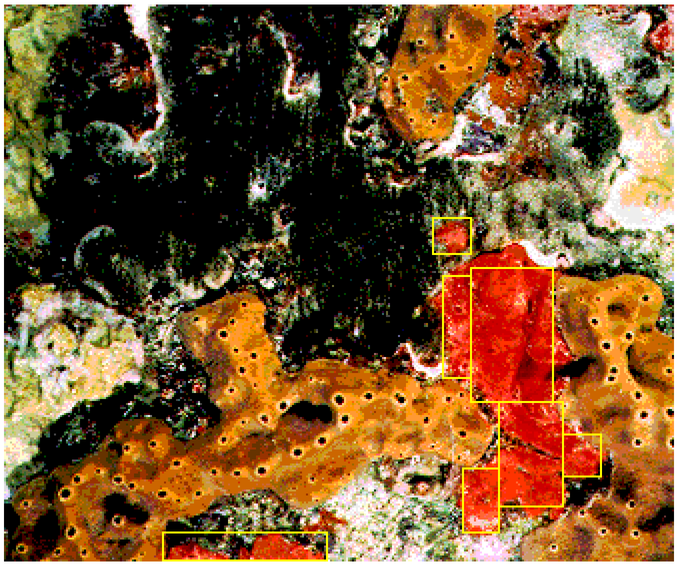


Figure 2.2 Image segmentation in RGB colour model with interest colour region

From above figure, a reddish colour sample is assigned, a colour range is determined base on the point, after that, a binary mask (black or white) is added, if the pixel containing reddish RGB values within the colour range, it shows white, if not, black colour is displayed, as shown in figure 2.3.

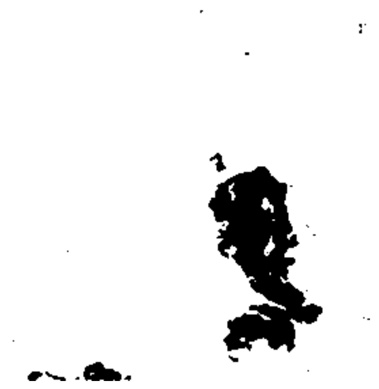


Figure 2.3 Result of colour segmentation

Colour image segmentation base on HSI model is convenient because of the hue and saturation properties. Hue property is used to identify colours and saturation is useful in masking process. After generating three components from the image (hue, saturation, intensity), a binary mask is completed by setting the threshold value according to the maximum value of saturation, for example, 10% of the maximum saturation value, any pixel carrying more than the threshold value will show white (1) in binary mask, on the other hand, the pixel carrying lower than the threshold value will display black (0). Performing a mathematical product of multiplication with hue image and binary mask, that produces the result of image segmentation.

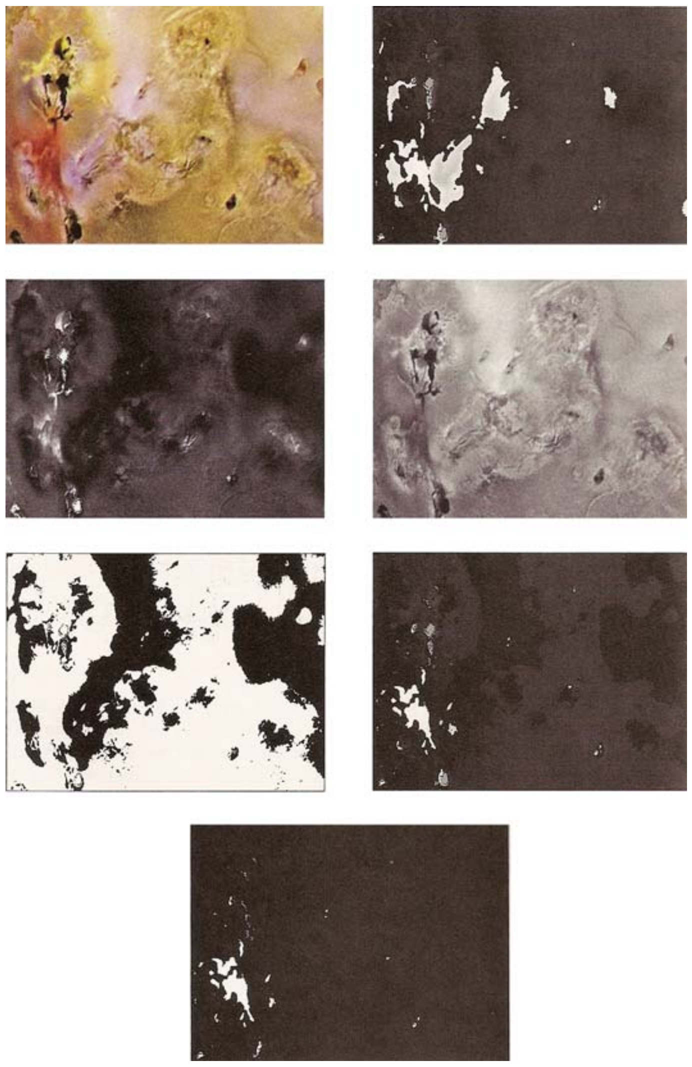


Figure 2.4 Image segmentation in HIS colour model [Gonzalez]

This is a basic algorithm which has been used to segment a colour image in HSI colour model. There are many other ways to perform the task by using the properties of colour model. Normally, using HSI colour model is more intuitive in colour image segmentation, but RGB colour vectors contain better result [Gonzalez].

2.2.2 Image processing

A two-dimension function $f(x, y)$ which represents the intensity of a digital image, where x and y are plane coordinates, and the value f is determined by the physical source of image. Illumination and reflectance are two physical components which forms the image of an element, in the other words, the production of the physical process that produces the image outcome.

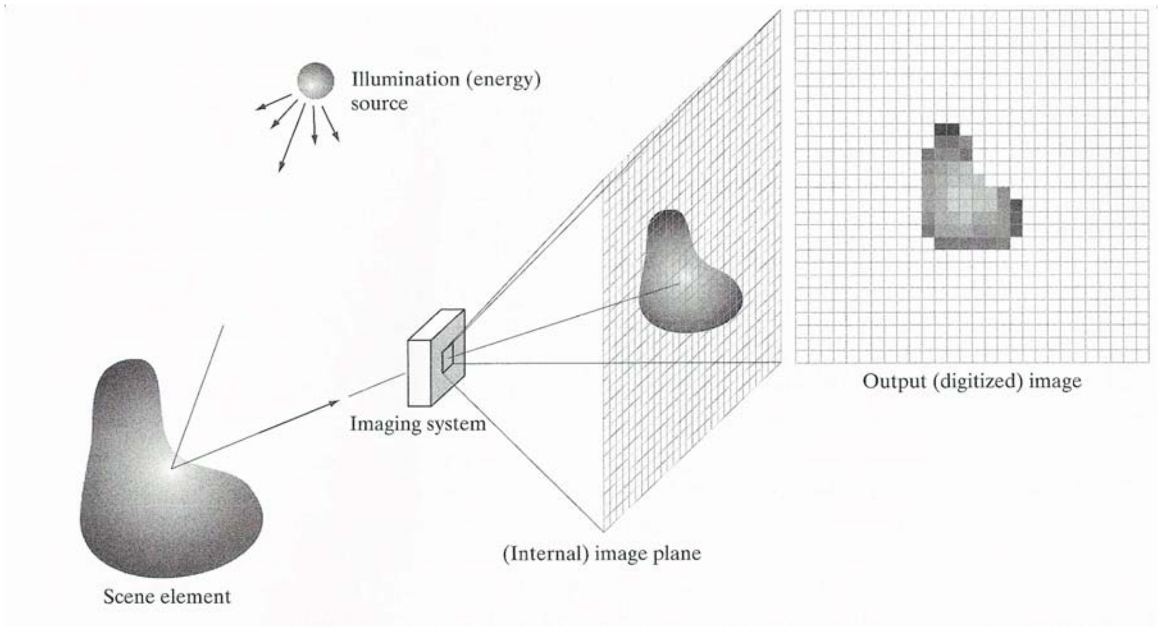


Figure 2.5 Image formation

From the result of a scene image, $f(x, y)$ must be non-zero and finite, therefore,

$$0 < f(x, y) < \infty$$

Since the image is generated by the physical component illumination and reflectance, thus,

$$f(x, y) = i(x, y) \times r(x, y)$$

Where

$$0 < i(x, y) < \infty$$

and

$$0 < r(x, y) < 1$$

in a gray level

$$l = f(x_0 y_0)$$

it is evident that l is between

$$L_{\min} \leq l \leq L_{\max}$$

$L_{\min}$ is > 0 and $L_{\max}$ is $< \infty$, hence

$$L_{\min} = i_{\min} r_{\min} \text{ and } L_{\max} = i_{\max} r_{\max}$$

[Gonzalez]

Above values are the boundaries in theoretical, through experiments, some average values of $i(x, y)$ and $r(x, y)$ are analyzed. For an indoor operation, the illumination level in an office is$1000 lm/m^2$, for reflectance, 0.01 for black velvet, 0.65 for stainless steel, 0.80 for white paint, and 0.93 for snow [Gonzalez].

Since the discovery of the imaging concept, a number of improvements have been made to the image formation technique. As a two-dimensional still grey level image, a combination of three-dimensional, moving, and colour image function can be expressed as follow:

2-D => 3-D	$f(x,y)$ => $f(x,y,z)$
2-D Still image => 3D Moving image	$f(x,y)$ => $f(x,y,z,t)$
2-D Grey still image => 3-D Colour moving image	$f(x,y)$ => $f(x,y,z,t)$

Table 2.2 Image extension

where z represents the vector in a plane coordinate, and t = temporal.

2.2.3 Pixel

A point is the smallest unit in our physical vision. By the same token, a digital image consists of thousands of points, and these points are called pixel in computer vision.

Assuming an image $f(x,y)$ is sensed with M rows long and N columns wide, thus, it can be represented as a $M \times N$ matrix of pixels array. Following matrix notation will help to ease of understanding:

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & \cdots & f(0,N-1) \\ f(1,0) & f(1,1) & \cdots & f(1,N-1) \\ \vdots & \vdots & & \vdots \\ f(M-1,0) & f(M-1,1) & \cdots & f(M-1,N-1) \end{bmatrix}$$

Figure 2.6 Matrix of pixels array

A pixel (x, y) has four neighbours in horizontal and vertical directions whose coordinates can be expressed as:

$$(x-1, y), (x, y+1), (x+1, y), (x, y-1)$$

above four pixels are called four-neighbours of pixel (x, y), and with a notation of $N_4(x, y)$. And four neighbours in diagonal direction, they are:

$$(x-1, y+1), (x+1, y+1), (x+1, y-1), (x-1, y-1)$$

above eight pixels are called eight neighbours of pixel (x, y), and with a notation of $N_8(x, y)$.

Two pixels are said to be connected if they are neighbours and carrying same value of similarity (binary image 0 or 1, gray level from 0 to 255, etc.). There are three ways to identify adjacency, assuming U contains the value of similarity, then:

1. 4-adjacency. Two pixels (x, y) and (s, t), (s, t) is in $N_4(x, y)$, and carrying same value of U.
2. 8-adjacency. Two pixels (x, y) and (s, t), (s, t) is in $N_8(x, y)$, and carrying same value of U.
3. m-adjacency. Two pixels (x, y) and (s, t), both carrying same value of U, and
    a) (s, t) is in $N_4(x, y)$, or
    b) (s, t) is in $N_{diagonal}(x, y)$ and $N_4(x, y) \bigcap N_4(s, t)$ is not carrying the same value as U.

m-adjacency has a advantage to remove the a ambiguities from 8-adjacency. It is showing in figure 2.7:

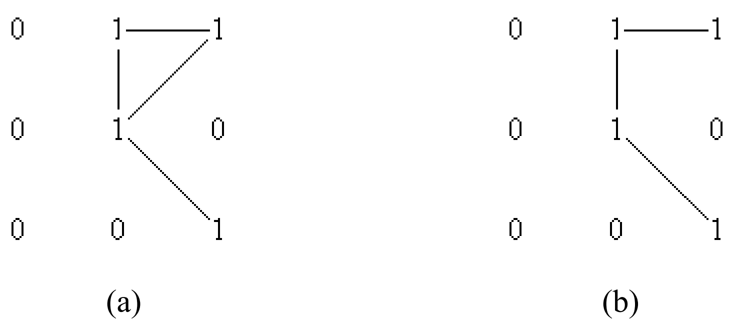


Figure 2.7 (a) 8-adjacent (b) m-adjacent [Gonzalez]

A binary representation is used in above figure, 0 and 1 are the values of similarity contained by U. From figure 2.7 (a), an ambiguity has discovered in top-right pixel, by following the m-adjacency rule, top-right pixel and central pixel has two coincident N_4 pixels which are top-central pixel and central- right pixel, but only central-right pixel not carrying the similarity value, and thus, it is not connect to the central pixel, hence, ambiguity has removed.

2.2.3.1 Distance measurement

Some simple mathematical function and algorithms can be employed to measure the distance between pixels:

let d be the function of pixels distance, assuming three pixels with coordinates of (x, y), (s,t), (v,w), then:

1. $d[(x, y),(s,t)] \geq 0$ ($d[(x, y),(s,t)] = 0$ if $p = q$)
2. $d[(x, y),(s,t)] = d[(s,t),(x, y)]$ and
3. $d[(x, y),(v,w)] \leq d[(x, y),(s,t)] + d[(s,t),(v,w)]$

Pythagorean theorem can be used to calculate geometry distance:

$$d[(x, y),(s,t)] = \sqrt{(x-s)^2 + (y-t)^2}$$

[Gonzalez]

2.2.4 Colour image

The use of colour in image processing increases the ability to track an object from the apparent characteristic and the process to identify an object is simplified, making the detection and extraction of captured image easier. Colour image are partitioned into two kinds of processes, full-colour and pseudocolour. A full colour image process needs associated equipment to sense the environment, such as colour camera or colour scanner and which are more expensive compare to the other process. Pseudocolour image process is done by assigning a particular colour and intensity to monochrome form, this is how most colour image process done in recent years.

2.2.4.1 Colour models (Colour space)

Red (R), green (G), and blue (B) are called primary colours of light, more additively, cyan (C), magenta (M), yellow(Y) are secondary colours, a combination of primary and secondary colours with different proportion that generates most of the environment colours been seen. An aid of CIE chromaticity diagram which describes the distribution of colours:

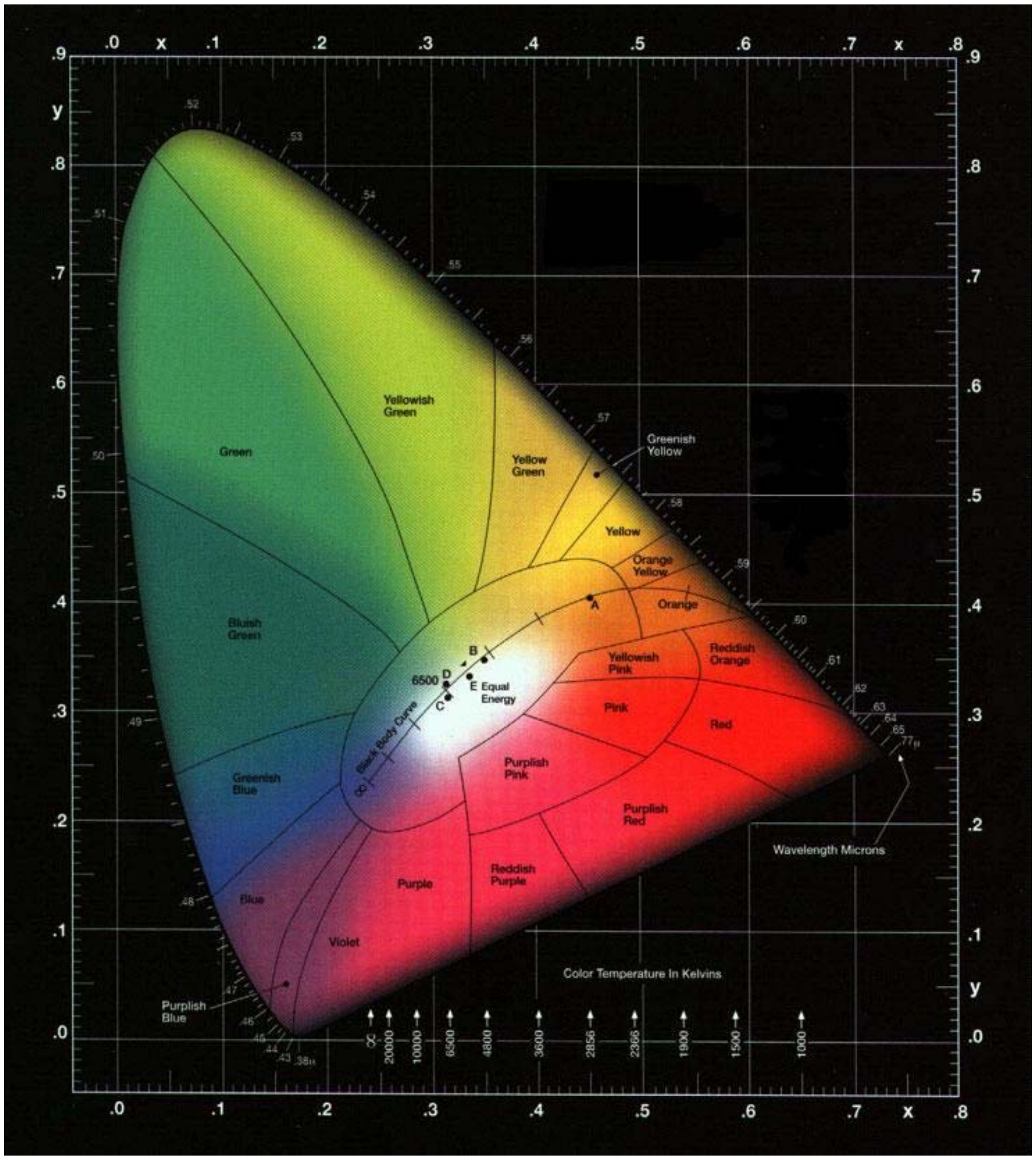


Figure 2.8 CIE chromaticity diagram

Different amounts of red, green, and blue been used to form a particular colour are called tristimulus values, and denoted as X , Y , Z , and the relationship among them are:

$$x = \frac{X}{X + Y + Z}$$

$$y = \frac{Y}{X + Y + Z}$$

$$z = \frac{Z}{X + Y + Z}$$

and it is note from above equations that

$$x + y + z = 1$$

[Gonzalez]

A colour model is a specification of a coordinate system and each colour is represented by a point within the system.

A RGB colour model is generated base on Cartesian coordinate system by assigning each primary and secondary colour to a corner in a cube. The black is at the origin, then, red, green, and blue are connected to black in x, y, z directions. According to the rule of mixtures of light,

$$green + blue = cyan$$

$$blue + red = magenta$$

$$red + green = yellow$$

$$red + green + blue = white$$

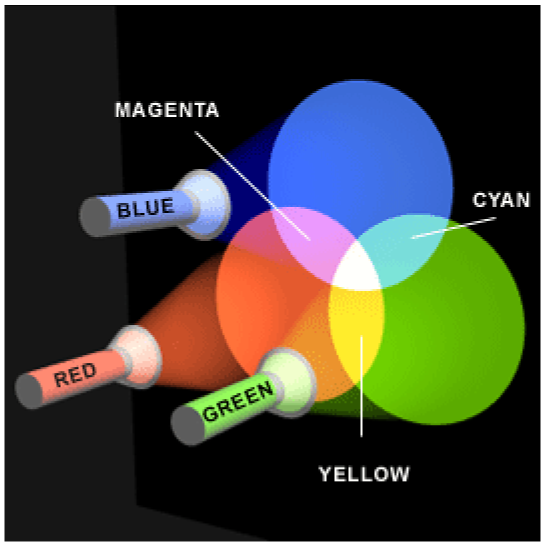


Figure 2.9 Colour mixture of light

hence, the secondary colours can be assigned. The green corner is connected to the cyan corner, and the blue corner is also connected to the cyan corner; the blue corner is connected to the magenta corner, and the red corner is also connected to the magenta corner; the red corner is connected to the yellow corner, and the green corner is also connected to the yellow corner. White is directly opposite to the black in the farthest corner. By connecting a line directly from black to white, this is called the line of grey scale. A RGB colour model coordinate system is shown in figure 2.10.

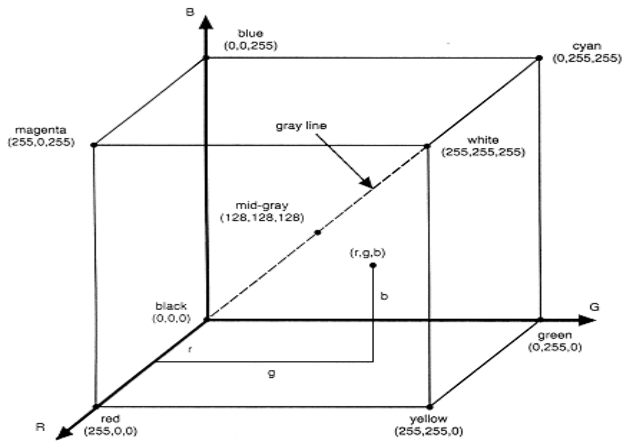


Figure 2.10 RGB colour model coordinate system

in 24-bit colour image, RGB colour model contains $(2^8)^3 = 16777216$ colours. Some RGB values of colours are shown in table 2.3.

R	G	B	Colour
0	0	0	Black
255	0	0	Red
0	255	0	Green
0	0	255	Blue
0	255	255	Cyan
255	0	255	Magenta
255	255	0	Yellow
255	128	128	Bright Red
128	255	128	Bright Green
128	128	255	Bright Blue
64	64	64	Dark Grey
128	128	128	Intermediate Grey
192	192	192	Bright Grey
255	255	255	White

Table 2.3 RGB value of colours

A colour model of CMY is used in most output devices (not a display device) such as colour printers, a data input in CMY form is required, and thus, a conversion equation that performs RGB to CMY is generated:

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

and rules for pigment mixture to produce colours are as follow:

$$cyan + magenta = blue$$

$$magenta + yellow = red$$

$$yellow + cyan = green$$

$$cyan + magenta + yellow = black$$

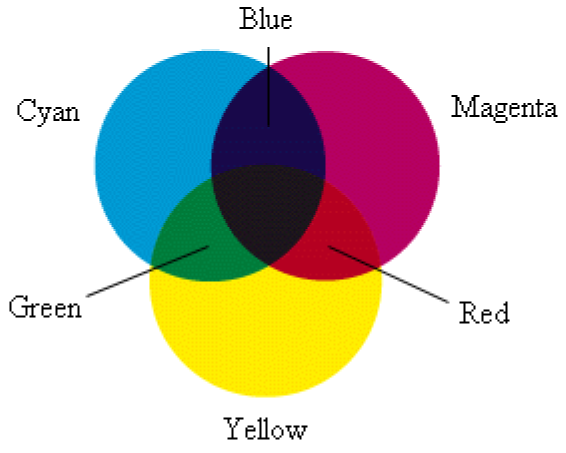


Figure 2.11 Colour mixture of pigment

According to the pigment mixture rule, $cyan + magenta + yellow$ that produces a colour in black, but in practice, a muddy looking black is produced, therefore, a fourth colour black is added to generate a true black, and this kind of colour model is called CMYK, where K represents black.

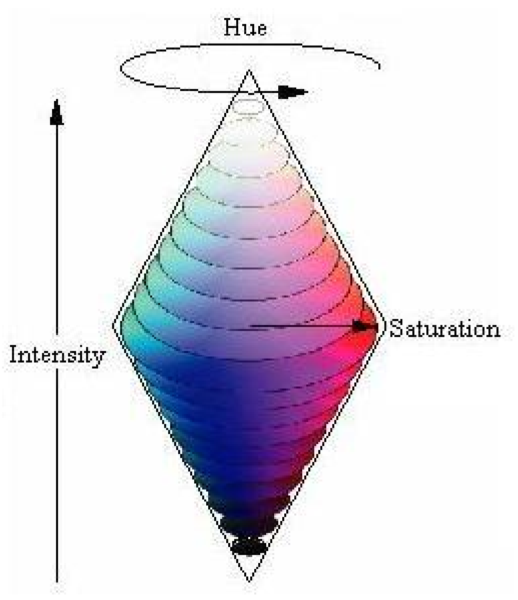
As human viewing an object, hue, saturation, and intensity are three properties used to describe an image:

1. Hue associates to a colour with some position in the colour spectrum – red, green, blue etc.

2. Saturation gives a measure of the degree to which a pure colour is diluted by white light.
3. Intensity corresponds to the brightness of a colour

[Sebe]

HSI colour model can be represented by following diagrams:



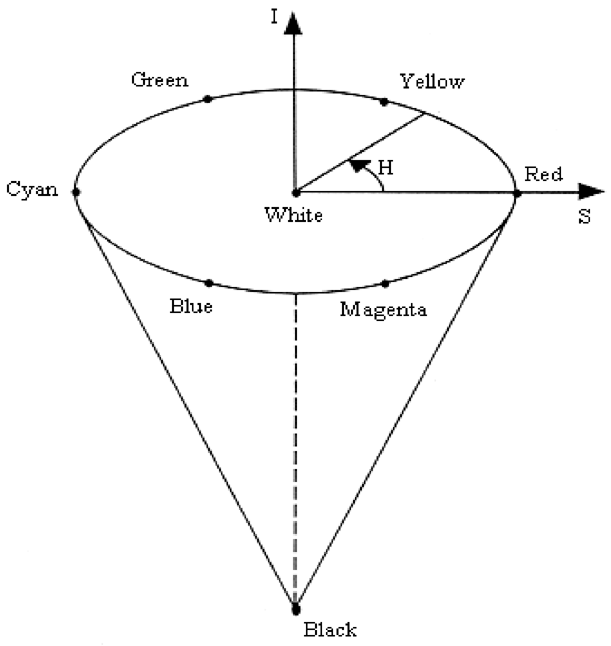


Figure 2.12 HSI colour model

In above diagram, primary and secondary colours divide the circumference into six sectors equally, any colour inside the cone contains information of hue, saturation, and intensity. *H* = hue, which is the angle between the red axis and the pixel; *S* = saturation, where the length of pixel determines the saturation; *I* = intensity, the depth of the pixel determines the value.

RGB and HSI colour models existing following relationship, in the situation of converting colours from RGB to HSI:

$$H = \theta \qquad \text{if } B \le G$$
$$H = 360 - \theta \qquad \text{if } B > G$$

and

$$\theta = \cos^{-1}\left\{ \frac{\frac{1}{2}[(R-G)+(R-B)]}{[(R-G)^2+(R-B)(G-B)]^{\frac{1}{2}}} \right\}$$

$$S = 1 - \frac{3}{R+G+B}\left[\min(R,G,B)\right]$$

$$I = \frac{1}{3}(R+G+B)$$

[Gonzalez]

in above equation, θ is the angle between the pixel and red axis in HIS model. In the situation of converting colours from HSI to RGB:

in RG sector, $0° \le H < 120°$

$$B = I(1-S)$$

$$R = I[1 + \frac{S\cos H}{\cos(60° - H)}$$

$$G = 3I - (R+B)$$

in GB sector, $120° \le H < 240°$

$$H = H - 120°$$

$$R = I(1 - S)$$

$$G = I\left[1 + \frac{S\cos H}{\cos(60^\circ - H)}\right]$$

$$B = 3I - (R + G)$$

similarly, in *BR* sector, $240^\circ \le H < 360^\circ$

$$H = H - 240^\circ$$

$$G = I(1 - S)$$

$$B = I\left[1 + \frac{S\cos H}{\cos(60^\circ - H)}\right]$$

$$R = 3I - (G + B)$$

[Gonzalez]

2.3 Robot arm

Humans are good at moving around in the real world, if human needs robot to operate like humankind, the robot needs to be taught how to perform actions like a human. In certain environment, robot has found to be better at execution than humans, so it can be imagined that a world with robotic technologies which improve humans life easier and better. As described in Webster’s Dictionary, a robot is an automatic apparatus or device that performs functions ordinarily ascribed to humans, or operates with what appears to be almost human intelligence [Lumelsky].

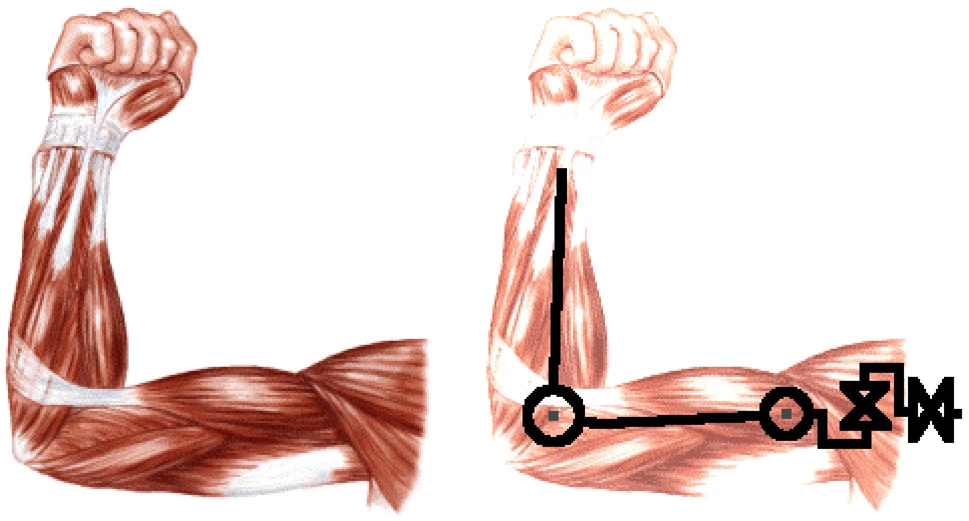


Figure 2.13 Human arm manipulator [Robot Arm Tutorial]

Basically a robot combines of three components, mechanics parts, computing algorithms, and sensing devices. A robot senses the environment information by sensing devices internally and externally, information is computed through a series of algorithms to generate useful inputs, consisting above progresses, mechanics parts are triggered to perform a motion as output. From above statements, to plan a motion with avoidance of object collision within certain workspace is main task of robot operation, without a clear definition of uncertain environment which is hazardous to both environment and robot, in other words, a robust robot manipulation system that accomplish tasks within its workspace without hitting or bumping into any unexpected object.

2.3.1 Workspace

The real world robot operates within a workspace either in two-dimensional space or three-dimensional space, for example an arm manipulator.

The physical structure of the robot can be considered as consisting of the arm and the hand. The arm generates the primary workspace which determines the regional structure and orientation structure, regional structure that describes the position of the arm, and orientation structure deals with the tool orientating. The hand generates the secondary workspace of a robot. The robot workspace is all places that the end effector can reach. The workspace is dependent on the angle limitations of robot degree of freedom, the arm link lengths, the angle between the link, etc. Some workspaces are spherical and some workspaces have very complicated shapes. While choosing a suitable robot arm to an industrial purpose, it is important that the workspace is large enough to cover all positions need to be reached by the robot arm [RRG].

The workspace of a robot is an important criterion in evaluating manipulator geometries. Manipulator workspace may be described in terms of the dexterous workspace and the accessible workspace. Dexterous workspace is the volume of space which the robot can reach with all orientations. The end-effector can be arbitrarily oriented in each point of dexterous workspace. The accessible workspace is the volume of space which the robot can reach in at least one orientation. In the dexterous workspace the robot has complete manipulative capability. However, in the accessible workspace, the manipulator's operational capacity is limited because the terminal device can only be placed in a restricted range of orientations [RRG].

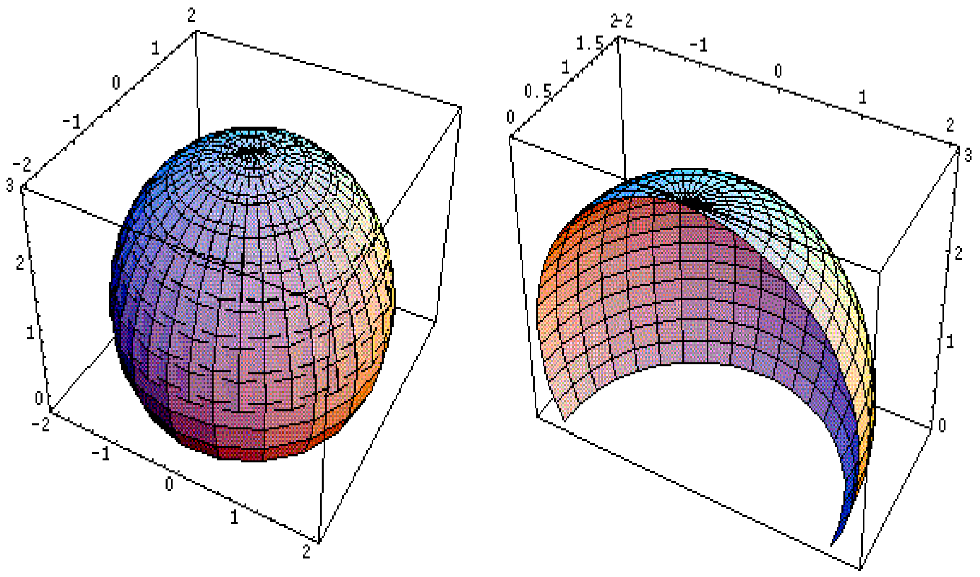


Figure 2.14 Workspace of an arm manipulator [CSS]

Figure 2.14 represents the workspace of the robot. The robot's workspace is the complete surface area of the sphere except the bottom which is the base of the robot, which the arm of the robot can not reach. Different sizes of workspace may generated by changing the length of link, but this would be the general shape. If there is any obstacle within the workspace of the arm movement, the shape can be more complicated.

2.3.2 Sensing

The robot moves depending on the information sensed. There are two situations may occur, complete information about the objects within the workspace is provided to the robot prior to move; there is incomplete and uncertainty about the environment information and may be obtained by the sensors in real time operation.

As the information of workspace is complete available to the robot, a method is employed to approach motion planning. Geometric approach is an example to solve motion planning, since complete information is obtained, this method relies on the principles of geometric space and the properties of the robot and obstacles in the workspace. Geometric approach can be summarized as follows:

1. It is applicable primarily to situations where complete information about the task is available.
2. It relies on geometric properties of objects.
3. It can, in principle, deliver the optimal solution.
4. It can, in principle, handle tasks of arbitrary dimensionality.
5. It is exceedingly complex computationally in more or less complex practical tasks [Lumelsky].

In the situation of incomplete information is provided, the environment information is fetched by the sensors, then the robot “think” what to do according to limited sensed data, thus, to solve motion planning based on topological properties of space is approached. A topological approach can be summarized as follows:

1. It is suited to unstructured tasks, where information about the robot surroundings appears in time, usually from sensors, and is never complete.
2. It relies on topological, rather than geometrical, properties of space.
3. It can not in principle deliver an optimal solution.
4. It can not in principle handle tasks of arbitrary dimensionality, and it requires specialized algorithms for each type of robot kinematics.
5. It is usually simple computationally: If a technique us applicable to the problem in hand, it will likely be computationally easy [Lumelsky].

Most robot arms only have internal sensors, such as encoders. Additional sensors such as visual, haptic, and tactile, etc, may be added to help retrieving information from the environment.

It is common that visual sensor has been used popular to sense surroundings, a robot arm can simply moves from one point to another without pre-programmed position by using a visual feedback algorithm, moreover, if the arm can locate a position in X-Y space of an image, it could then lead the end effector to go to that same X-Y location.

Haptic sensing involves human in the control loop. In this kind of application, human is capable of controlling the robot arm remotely. This is accomplished by wearing a special glove, or by operating a simulation model. In more advance, some robot arms carrying feed back sensors, such as touch sensors, which gets directed back to the human.

Tactile sensing usually involves force feedback sensors and current sensors. These sensors are physically touching the environment. Sensors detect unexpected force or current spike to inform the robot a collision has occurred. By measuring the force at the end effector, the robot will “know” the state of the griper, is it too tight or too lightly. In another way of sensing, collision may be detect by using current sensors, sudden current changes which means a collision has occurred [Robot Arm Tutorial].

To determine what type of sensing is suitable for a robot motion application, actually, all kinds of sensing is fine, as long as a proper motion planning algorithm is employed, even a simplest tactile sensing can guarantee to direct a robot to reach its target position [Lumelsky].

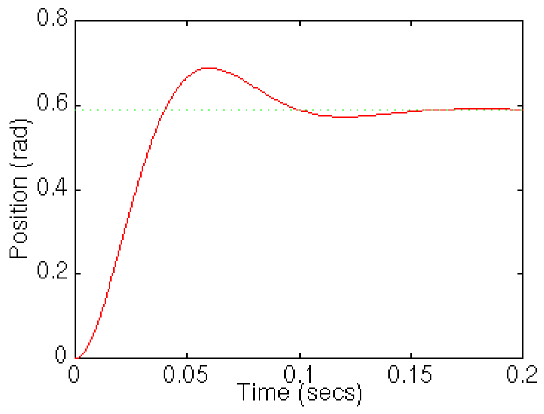
2.3.3 Robot motion control

A successful control of a robot motion is based on the algorithm been performed to direct the motors electrically and mechanically.

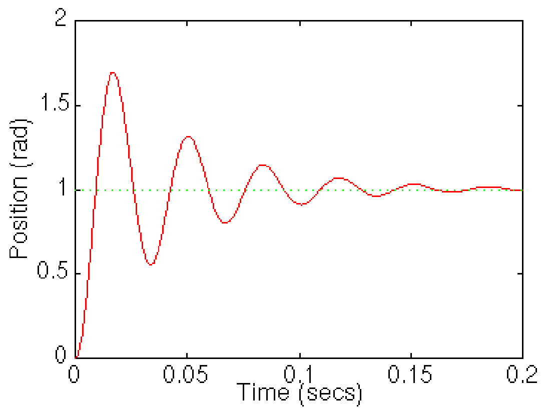
While performing a regulation of controlling the motion of arm motors, an appropriate relationship is established with the desire position and corresponding actual position. This relationship can be distinguished into three types:

1. Open loop control: the control system is regardless the system error.
2. Linear control: the control algorithm is a linear relationship.
3. Nonlinear control: the control algorithm is a nonlinear relationship.

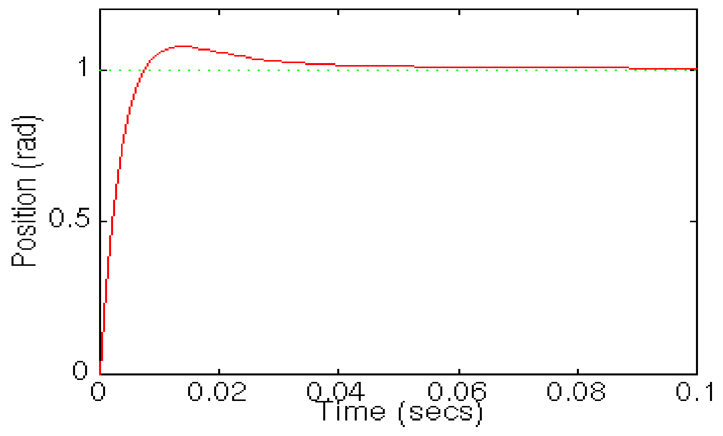
An open loop control is a control algorithm without error correction. Linear control and nonlinear control belong to close loop feedback control which performs error corrections. Closed loop control means a method in which a real-time measurement of the process being controlled is constantly fed back to the controlling device to ensure that the value which is desired. PID control is a common type of close loop feedback control algorithm that widely used in controlling robot motion. PID involves three mathematical control functions: Proportional, Integral and Derivative, these values are used in helping the robot moves smoothly to the desire position. Proportional parameter is the most important value in PID control. The proportional control gain has an overall influence on the performance of close loop control system, determining that gain of proportional control that makes the closed-loop control system stable, reasonable smooth and accurate. The proportional control gain determines the magnitude of the difference between the set-point and the error, and then applies appropriate changes to the control variable to eliminate error. A control action proportional to the integral of the error signal in a PID control law allows reducing accumulated errors over time. Integral control examines the offset of set-point and error over time and corrects it if necessary. In order to compensate for the loss of stiffness of the system, we need the derivative parameter. Derivative Control monitors the rate of change of the error and consequently makes changes to the output to accommodate unusual changes.



(a)



(b)



(c)

Figure 2.15 PID control

From figure 2.15 (a) above, the steady-state error looks good but the settling time is too long, as is the overshoot, and the steady-state error to a disturbance is large. By adding an integral term will eliminate the steady-state error and a derivative term will reduce the overshoot. In figure (b), the response is faster than previous figure by adjusting the P, I values, but the large I value has worsened the transient response. In (c), after adjusting the D value, derivative term will reduce the overshoot and get to the desire set-point smoothly.

In working with robotics motion control, account with for inertia, resistance, and other variables is needed. Careful selection of the PID gain values can minimize oscillation and overcompensation.

Most common application of a robot arm manipulator is to move from one position to another. People prefers to describe a position of an object in Cartesian coordinate system (x, y), similarly, robot is often commanded in this kind of manner.

There are two ways of transformation to calculate positions of robot arm, a two-dimensional arm manipulator is used to illustrate in this example:

1. Direct transformation, by knowing the properties of the robot arm such as length of the arm l_1, l_2, angles between the link θ_1, θ_2, a corresponding position of end-effector in Cartesian coordinate system (x, y) can be found.

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} l_1 \cos\theta_1 + l_2 \cos(\theta_1 + \theta_2) \\ l_1 \sin\theta_1 + l_2 \sin(\theta_1 + \theta_2) \end{bmatrix}$$

[Lumelsky]

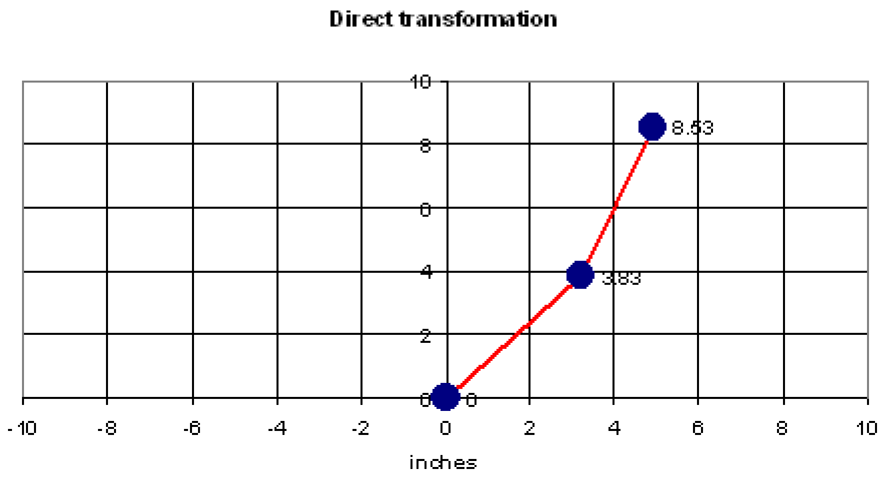


Figure 2.16 An example of direct transformation

From above diagram, a corresponding position of the robot arm end-effector is found $j_1 = (3.21, 3.83)$ and $j_2 = (4.92, 8.53)$ by calculating with $l_1 = 5$, $l_2 = 5$, $\theta_1 = 50°$, and $\theta_2 = 20°$.

2. Inverse transformation, this transformation is employed in an inverse situation. As the robot arm is arrived in certain position of Cartesian coordinate system by the sensed data, corresponding values of robot arm can be determined by the following calculation.

$$\theta_1 = \tan^{-1}\frac{y}{x} - \tan^{-1}\frac{l_2 \sin\theta_2}{l_1 + l_2 \cos\theta_2}$$

$$\cos\theta_2 = \frac{x^2 + y^2 - l_1^2 - l_2^2}{2 l_1 l_2}$$

[Lumelsky]

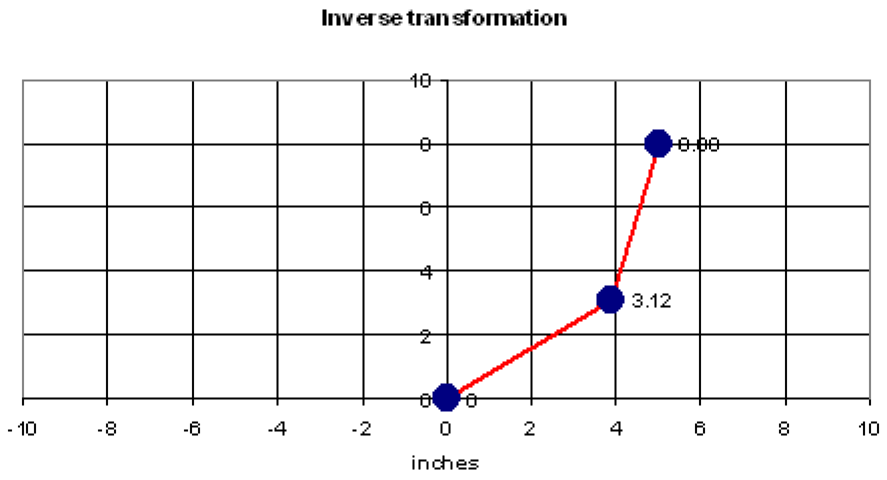


Figure 2.17 An example of inverse transformation

From figure 2.17, corresponding values of the robot arm end-effector in position $(5,8)$ is found $\theta_1 = 38.62490449$ and $\theta_2 = 38.7394246$ with $l_1 = 5$ and $l_2 = 5$.

2.3.4 Motion planning

Since a good robot is built up by many components' contribution, motion planning capability is essential for autonomous robot system, the task is to generate a collision-free path for a movable object among known and static obstacles. It is not hard to understand that the motion planning is kind of space problem, therefore, a success motion planning that drives the robot to operate in one or more of the following ways:

1. Execute the predefined path.
2. Find an optimal path.
3. Plan a “reasonable” path.
4. Plan a path that respects some constrains

[Lumelsky].

From previous chapters, again, motion planning is based on the purposed of collision avoidance, it brings the way to approach the problem into two groups:

1. Motion planning with complete information (off-line planning): all the information is available beforehand, and an optimal path can be found by computation before the robot start moving.
2. Motion planning with incomplete information (on-line motion planning): the optimal path is found by computing information collected by the real time sensing devices, and decision is made continuously with robot move.

Most of the current approaches to path planning are based on the concept of configuration space (C-space) introduced by Lozano-Pérez and Wesley. C-space is the set of all possible configurations of a robot. Once the problem has been formulated in the C-space, it becomes equivalent to finding a connected sequence of collision-free configurations from the start configuration to the final configuration. The number of independent parameters needed to fully specify a robot configuration is the dimension of the C-space [Motion Planning].

3. COMPUTER VISION SYSTEM

3.1 Introduction

As mentioned in the previous chapter, the review of current literature indicated that a complete arm imitation robot system needs a sensing technology such as computer vision device to provide the information to monitor and direct the motion of the object and manipulator. Three common tasks for computer vision processing are:

1. Detection. The detection of the presence of an object in an image.
2. Tracking. The spatial tracking of a once-acquired object over time.
3. Recognition. Recognition of one of many object types, hence, classification of the observation in the image into one of many classes.

This chapter outlines a computer vision model that been used to achieve the purpose of sensing environment. As a computer vision device, the system is able to detect, analyze and perform further process with the captured object image, for example, human hand, and base on the characteristic to track the object moves.

3.2 Why a computer vision system

There are few methods available to track an object in the industry, the difference between them are the sources and properties been used. Depending on the tracking environment and the need of the end user, following analyses is made.

3.2.1 Magnetic tracking system

Magnetic tracking is the most commonly used technology as an interface to a virtual world. The technology uses a magnetic field emitting source and relative sensors to capture the trajectory information of a moving object respect to the origin.

Advantages:

Do not restrict by the line-of-sight
Inexpensive and accurate
Long range detection is possible
High sample rate

However, interferences from the environment which may cause major problems with a tracker to report object motion. For example, metal in the environment will alter the magnetic field which gives inaccuracy readings; distance reduces the accuracy of tracking; filter processing introduces latency in the results which devastating in interactive visual realty applications.

Magnetic technology is popular because of its robustness and lack of constraints on user motion, and therefore the systems can track real world objects that have 6 DOF (Degrees of Freedom) and consequently it allows more realistic interaction with the virtual environment [Sturman].

3.2.2 Acoustical tracking system

Acoustic trackers use high-frequency sound to triangulate a source within the work area. System sends out pings from the source (mounted on the moving object) received by microphones in the environment. Precise placement of the microphones allows the system to locate the source in space to within a few millimetres.

Advantage:

Generates result accurately and precisely

These kinds of system rely on line-of-sight between the source and the microphones. They can suffer from acoustic reflections if surrounded by hard walls or other acoustically reflective surfaces. If multiple acoustic trackers are used together, they must operate at non-conflicting frequencies [Sturman].

3.2.3 Optical tracking system

Having witnessed the success of web camera applications and the appearance of high definition digital video cameras, we believe that digital video will soon become a part of everyday life. Video sequences provide more information about how objects and scenarios change over time [Wong].

Detection and tracking of a human body is a very complex problem due to the external factors such as irregular colour, shape, brightness etc. Tracking an object through an image sequence base typically involves feature of detection in one image to their new positions in the next. Performing object tracking by using image processing to find known objects which is low cost, portable and accurate, however, some physical problems such as occlusion, slow and lighting problem still exist.

Visual cameras can be used as a source to detect and track a moving object. The principle was using the features of moving object to trigger the sensitisation source of the camera, thus to achieve the purpose of tracking.

Advantages:

Real time tracking

Accurate

Possible to track multiple objects using multiple cameras

However, camera tracking system is affected with line-of-sight within the surveillance area, also the image processing time of camera is required, normally two or more cameras is required to cooperate in a wide area of detection.

3.3 Computer vision module

The camera module chosen for this project is the CMUcam2+ that produced by Carnage Mellon University. Its primary function is to track and monitor highly contrasting colour regions. It can also detect motion, provide colour statistics, and transmit image information to a computer for additional processing.

CMUcam2+ upgraded from previous generation with ROM and RAM, the master and slave mode feature which becomes the most important capability to increase the viable option for the complexity image process of the vision system required by the project. An inexpensive control board and camera module makes it worth to invest.

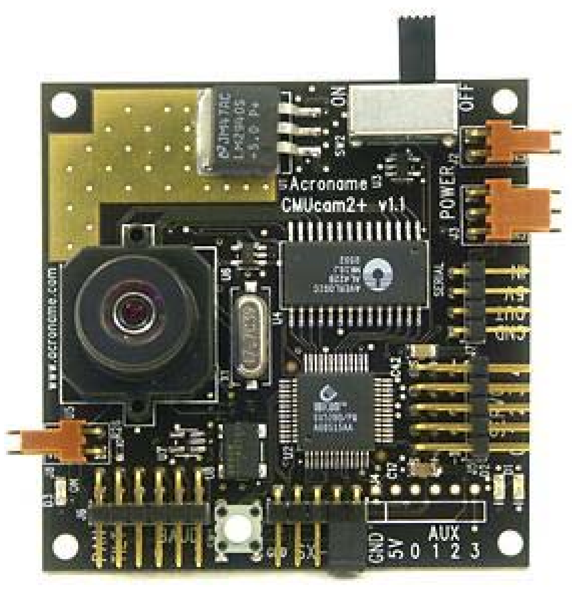


Figure 3.1 CMU2+ Camera module

The control board of CMUcam2+ utilises UBICOM SX52 microcontroller that operate at 75 MHz. This processor is a RISC processor that can operate at 765MIPS. It has 262 bytes of SRAM and a 4096 word flash programmable EEPROM. The control board can communicate with master devices through either TTL or RS232 serial port. The control board is able to control 5 servomotors and also has 4 auxiliary I/O ports.

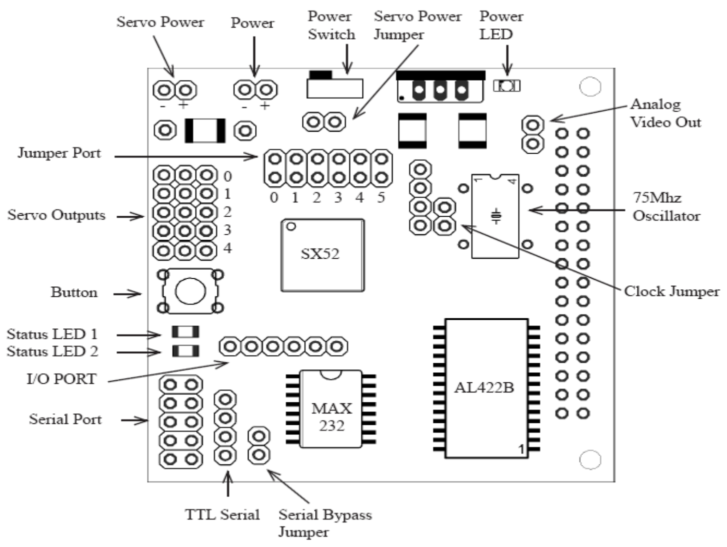


Figure 3.2 Board layout

OV6620 Omnivision CMOS camera comes along with the control board that allows simple high-level data to be extracted from the camera’s streaming video. The camera operating at a refresh rate 50 frames per second which allows the camera plentiful to achieve the requirement for the project. The module provides analogue video output at 352 x 288 resolution and maximum 176 x 255 onboard resolution. OV6620 also provides a function to operate under lower resolution 88 x 143 which reduces the pixels for image algorithms to process, thus allowing more frames per second to be operated. The logic chips and OV6620 camera module are mounted on a same board which saves a large amount of communication bandwidth usage, also, it is capable to transmit only the centre and bounding box (top, bottom, left and right) of the object image instead an entire image to be sent, hence, reduces the time required for transmitting in robotics projects, moreover, the master-slave feature will be useful to allow multi-cameras to be linked together, this setup let a parallel image processing and which improves the ability to track an object motion in a wide visual environment.

3.4 Colour tracking of computer vision module

The colour space is set default as RGB, but an optional setting YCrCb colour space is available, where Y is the intensity of the image, Cr is red chrominance and Cb is blue chrominance. Following are the transformation used by cameras to convert RGB to YCrCb:

$$Y = 0.59G + 0.31R + 0.11B$$
$$Cr = 0.713 \cdot (R - Y)$$
$$Cb = 0.564 \cdot (B - Y)$$

[Rosenberg]

The colour sensor has 356 columns and 292 rows of light sensitive cells arrange on a grid. Each location can detect a single colour: red, green or blue. A red, green and blue value is used to indicate how much of each channel is mixed into that final colour. Here is the sensors layout:

B(1,1) G(1,2) B(1,3) G(1,4) B(1,5) G(1,6)…….. B(1,355) G(1,356)

G(2,1) R(2,2) G(2,3) R(2,4) G(2,5) R(2,6)…….. G(2,355) R(2,356)

The module picks up the data from each two rows to generate the output.

B(1,1) G(2,1) R(2,2) G(1,2) B(1,3) G(2,3) R(2,4) G(1,4)…….

B(3,1) G(2,1) R(2,2) G(3,2) B(3,3) G(2,3) R(2,4) G(3,4)…….

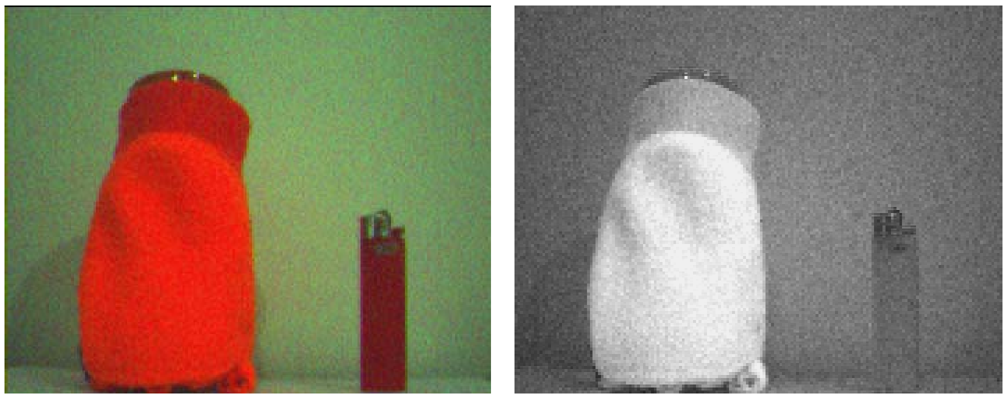
The extra green channel helps fill in the grid so that each pixel can be evenly distributed across the sensor, and that information is also closely to approximates the human eye which is more sensitive to the colour green. For the purpose of simplification, the CMUcam2+ ignores the second green value. CMUcam2+ generates the output data as following:

[R(2,2):G(1,2):B(1,1)] [R(2,4):G(1,4):B(1,3)]

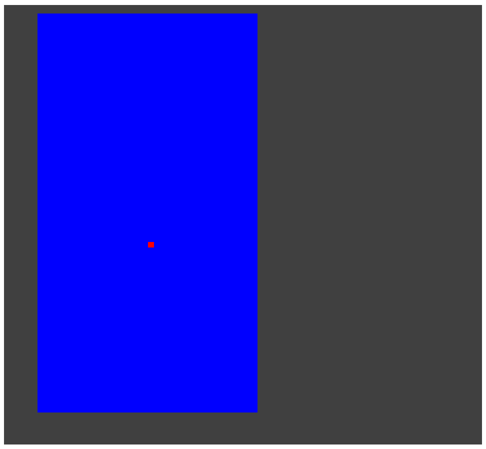
[R(2,2):G(3,2):B(3,1)] [R(2,4):G(3,4):B(3,3)]

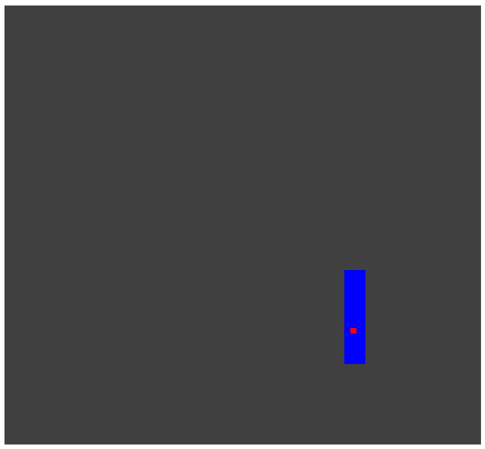
[Rosenberg]

However, a limitation of CMOS camera is that the colour channels are between 16 and 240 instead of the full 256 for each colour. This will be a problem when tracking objects with similar colour, for example, as the tracking value of red chrominance is defined to maximum, theoretically, the camera should track a red object, but as other values is set to minimum, that confuse the camera to track an red object and a orange object.



(a) (b)


(c)


(d)

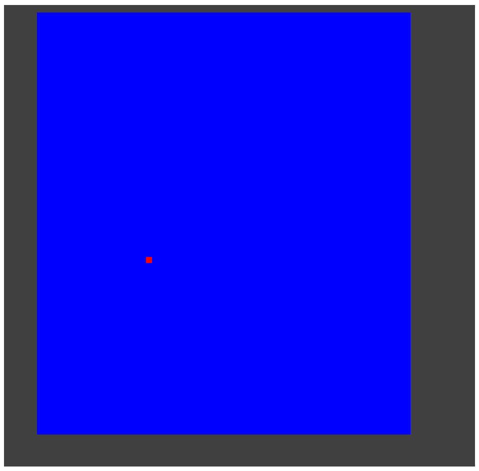

(e)

Figure 3.3 Colour tracking of orange and red by CMU2+ camera

From above figure, blue rectangle shows the bounding box of the object, and the red dot represents the centre. Objects in orange and red colour are shown in figure 3.3 (a), (b), and figure 3.3 (c) and (d) are the individual scene of each object as the red chrominance is set to maximum and other values in minimum. In figure 3.3 (e), it shows the camera recognizing two objects in one.

3.5 Image frame differencing

A moving object such as hand is required to be detected and tracked in this project. In the image point of view, the simplest way to detect the moves of an object is comparing scene of images. Performing an image differencing from the same or similar view point by feeding the camera a sequence of images which allows the camera to detect and isolate the moving object.

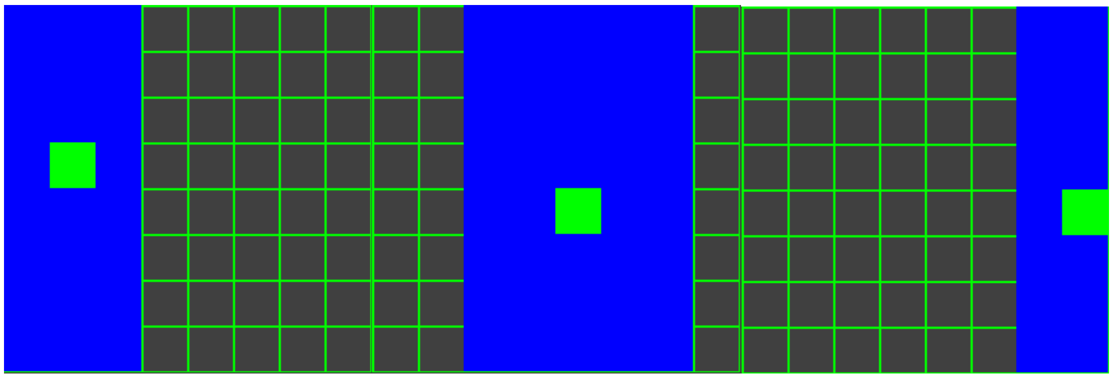


Figure 3.4 Image frame differencing

From above figure, it is not hard to see an object is moving from left hand side to right hand side, similar to previous section, the blue rectangle represents the boundary box of the object and green dot means the centre of the moving object.

3.6 Histogram analysis

A histogram graph is normally used to represents frequency and distribution of input data. For CMU2+ computer vision module, a histogram chart is used to indicate the frequency of red, green, or blue colour channel in an image. CMU2+ detects colour intensity value from 16 to 240, and divides them into 28 particular ranges, means that each particular range interval is $\frac{240-16}{28}=8$ intensity value. Each bar in the histogram graph shows the number of pixel been found in that particular colour range, hence, more values been found in a particular range means many colour were found in that range in an image.

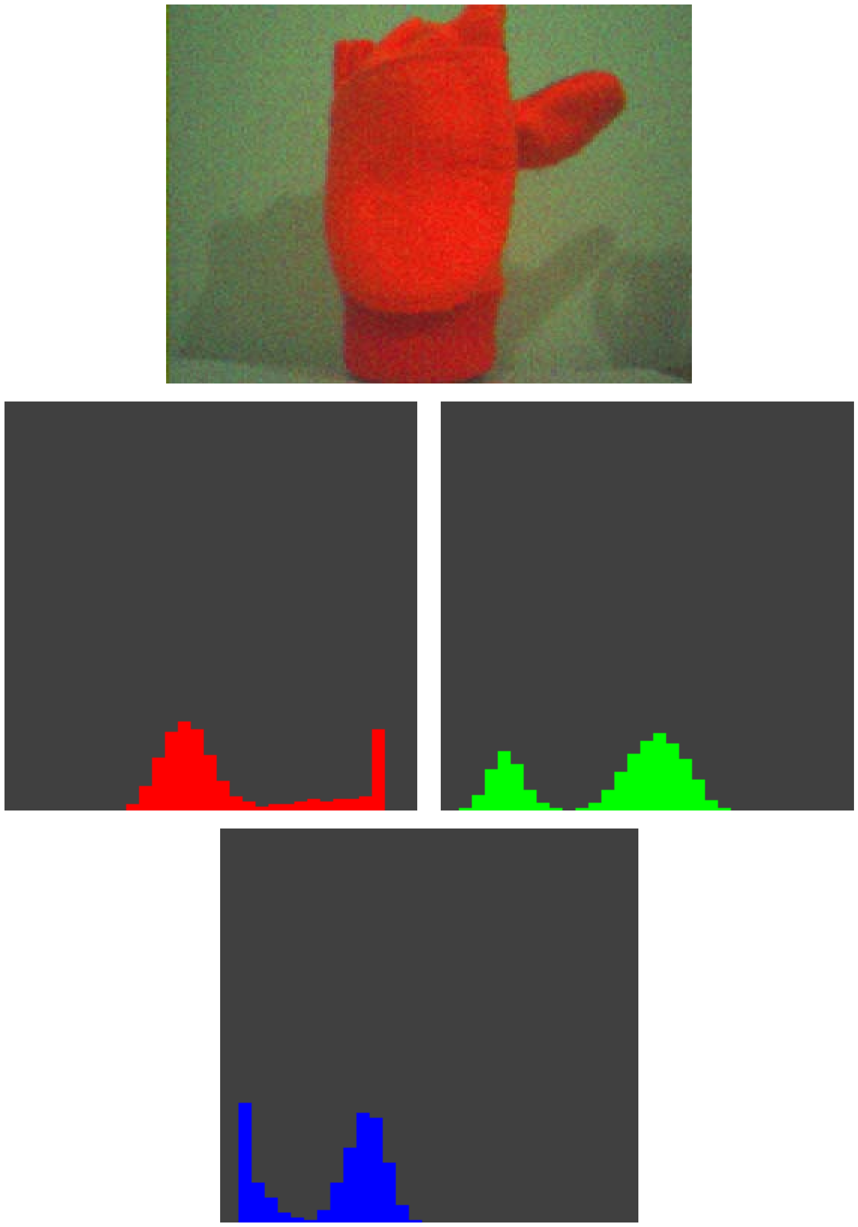


Figure 3.5 Histogram analysing

According to above figure 3.5, an orange (red) object is in the scene of image, thus, following conclusion can be made: colour with high intensity is distributed in right hand side of the histogram graph, object is detect by the camera with high intensity of red chrominance, green chrominance and blue chrominance are detected in lower intensity, although there is no actual object in green or blue colour, however, the colour intensity of orange (red) object is mixed by the primary colours, and some of them are caused by the light reflection.

3.7 Control commands

Serial commands are used to communicate with the CMU2+ module, the parameters format are:

• 1,200 to 115,200 Baud
• 8 Data bits
• 1 Stop bit
• No Parity
• No Flow Control

All commands are sent using visible ASCII characters [Rosenberg].

Control commands are classified into 11 groups according to the function they perform, that include camera module commands and system level commands, these may be employed for internal settings of camera module; colour tracking commands and image windowing commands, normally these groups of commands are used for monitoring and tracking moving objects; frame differencing commands, histogram commands and colour statistics commands are used to analyse images; furthermore, command groups such as buffer commands and data rate commands are controlling data transmission, and servo commands and auxiliary I/O commands are utilized to handle servo motors.

3.8 Space coordinates acquiring

Vector is important properties to state an object in a coordinate system, of course in a computer vision system. Establishing a vector of coordinates in the camera frame of its own is the primary task before further measuring application, once the vector of the camera is obtained a complete space system can be created from the location of the camera to the object.

Due to the nature of image formation is cause by light reflection and refraction on an object, light coming directly towards to the camera is not the only source being recorded into an image, other lighting sources from arbitrary angles are reflected or refracted, thus, perspective effect is created to visible by the camera. Since the angle of lighting source is an important factor to establish coordinate vectors and needs to be clear declared, there are a number of methods can be used to identify it.

Ray tracing is most commonly used in computer graphics. This is a process of simulating the path of light source travel. This method requires a camera with high accuracy of lens and photosensitive array, and that is done by measuring thousands of lighting rays, even though, a small error may cause a large angle error. In general ray tracing is preformed from the source of light and then terminating in the destination, however in this case it would begin from the photosensitive array until it had passed through the lens before terminating.

Look up table possibly is the most accurate and fastest method if doing it correctly. In order to generate the look up table, the centre pixel of an object is employed. According to the image viewed by the camera, the centre pixel is measured and assigned to the vector that created in the viewed image, as above procedure completed, next step is to repeat above procedure with slightly different location of the object. Since this might be the most accurate method, however, it needs a lot of time to generate the table with thousands of measurements which makes it unrealistic.

Simplified model, to create a simplified model of the lens some elements of a look up table are required however not to near the same extent. A constant ratio exists between pixel and angle in simplified model, pixel location and vector can be generated base on this linear relationship, and the ratio of pixel and angle can be measured in a minimal number. However, inaccuracy will be caused due to the manufacturing process of lenses which leaves some regions of the lens have no measurement been taken.

3.8.1 System method

Simplified model is the method chosen to obtain the space vector of the system. A constant ratio is used to multiply the distance of x and y axis to obtain yaw and pitch. The choice of a constant ratio for pixel to angle is based on the measurement of the slope of best line fit.

The system space vectors are generated as the position of the object is located. In the process of space vector creation, location of the camera, pitch and yaw angle of the camera is taken into account. This process is relatively simple to perform, the pitch and yaw angles is added together with the camera space. Once the vectors of apex and corner pixels have been assigned, a Cartesian coordinate vector is created through the sine and co-sine calculations.

3.9 Object positioning algorithm

3.9.1 Bi-Vector

Once two vectors have been established the position of the object within the view image can be found. To obtain the position of the object, a simple assumption that two vectors will cross at the object can be made. However, due to the extremely low chance that the vectors will actually meet mathematically, an intersection point which has the minimum distance to both vectors is prior to define, to find this point two points that have shortest distance to each vector are needed to predefine by using accustomed calculation of calculus.

Each vector can is represented in the form of:

$$[(X_0 + X_v \times t), (Y_0 + Y_v \times t), (Z_0 + Z_v \times t)]$$

Where X_0, Y_0 and Z_0 are the coordinates of the camera and X_v, Y_v and Z_v are the direction components of the vector, t is a variable that represents the distance of point from the camera and will be replaced by the variable s for another vector.

Following equation describes the distance between any points in the vectors:

$$D = \sqrt{(((X_{01} + X_{v1} \cdot t) - (X_{02} + X_{v2} \cdot s))^2 + ((Y_{01} + Y_{v1} \cdot t) - (Y_{02} + Y_{v2} \cdot s))^2 + ((Z_{01} + Z_{v1} \cdot t) - (Z_{02} + Z_{v2} \cdot s))^2)}$$

A simplify equation from above can be obtained in the purpose to find a minimum distance between two points in further progress by leaving out the square root, which gives that:

$$D = ((X_{01} + X_{v1} \cdot t) - (X_{02} + X_{v2} \cdot s))^2 + ((Y_{01} + Y_{v1} \cdot t) - (Y_{02} + Y_{v2} \cdot s))^2 + ((Z_{01} + Z_{v1} \cdot t) - (Z_{02} + Z_{v2} \cdot s))^2$$

Equations $\frac{dD}{dt}$ and $\frac{dD}{ds}$ can be obtained from above, that gives the slope of distance function between two points, depends on which vector is primary to look at.

Since there is only one closest point to two vectors, above equations will only have one single minimum value, hence, by equating both equations to zero with two variables *t* and *s* which allows solving both variables simultaneously.

Once *t* and *s* have been solved, a closest point of each vector can be found by substituting *t* and *s* into original vector equation, by averaging these two points dimensionally which gives the final point and that can be logically considered as the centre of the object.

Through a few of practical tests, above algorithm can be performed perfectly as two vectors are crossed with each other, but in fact two cases are excepted which may cause problems, if two vectors are exactly parallel or inaccuracies measurement are occurred.

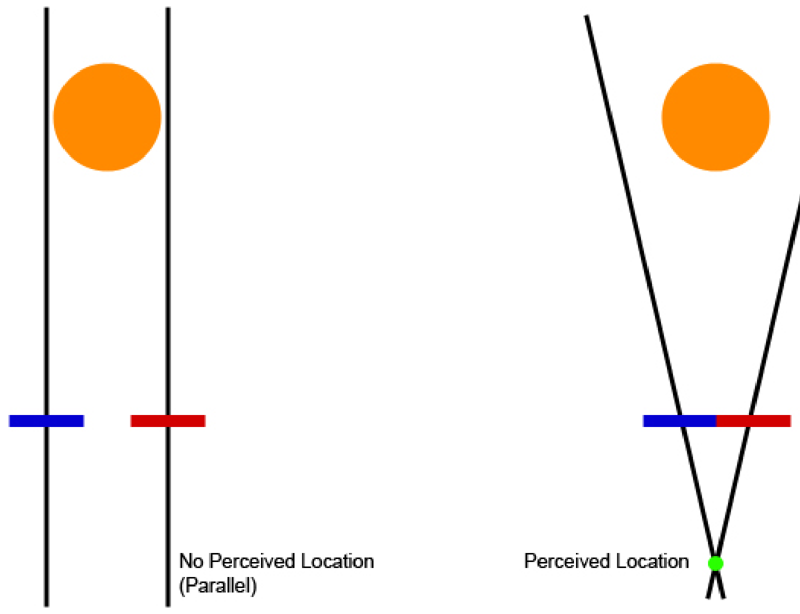


Figure 3.6 Exaggerated problem

However, the equations been used for vector calculation are fractions, that gives an opportunity to discover if the denominator equals to zero is occurred (vectors are parallel), then zero errors can be prevented in runtime division without the need for performing a dot product multiplication to find the angle between them.

3.9.2 Perspective

A simple thought can be made after previous discussion, bi-vector approach is more accurate than using a single camera and convert the object viewed into quantification values, these values are used in equations of vector calculation to produce a point which can be used as the centre of the object. In order to be more perceptive with changes of the viewed object the size of the object is required before a distance can be obtained.

Although there are inaccuracies existing in a single camera approach find the location of the object, however, it is worth to perform this algorithm in the purpose of making sure that the performance of bi-vector is correct.

The algorithm for obtaining the distance of an object from the viewed size is showing below:

$$D = RealRadius / \tan^{-1} (ViewedRadius \times C)$$

Where:
D is the distance from the camera.
RealDiameter is the size of the object.
ViewedRadius is the distance from the centre of the object to the top, bottom, left or right of the object in pixels.
C is the pixel to angle ratio for the either the X or Y axis depending on whether the axis being used for used for *ViewedRadius*.

It is possible to perform the algorithm a number of times with different radius measurements, an improvement of performance would be expected by taking an average calculation with these distances.

3.9.3 Methods combination

A cooperation of bi-vector and perspective methods is likely achievable, the perception method is only uses as when one camera has lost track of the object but the other one still has it. By creating a set of functions that are able to estimate the error of both methods, a confidence value can be assigned to each position of bi-vector and two perspective based locations, these values could then be normalized to create weighting values. The three locations could then be multiplied by the corresponding weighting value, before being added together to generate the final position. While this synergy is not yet in place it would be expected to be relatively straight forward to implement, however tuning of the confidence functions could be quite difficult to achieve such that results are reliable. The basis for confidence of the bi-vector method could be expected to be relative to the angle between.

The reacquisition algorithm can be done by taking the position of the camera which has lost the track of the object and the position of the object can be found by performing the perspective method of the other camera. Once a vector has been generated it is then transformed into pitch and yaw values by simply using ATan2 and Pythagoras' theorem.

3.10 Computer vision module interacting with human hands

Human visual system has the ability to detect hands in arbitrary environment and situation. It is crucial that a hand is acted as a mechanism input to a computer which is robust and reliably to detect an object in front of different backgrounds because all further stages and functionality depends on it.

Detection of a colour object such as colour glove which can be achieved in very high detection rates despite low false positive rates. Most of the hand detection method resort to less object-specific approaches and instead utilize colour information, and sometimes combine the technique of motion flow, background differencing and position priors. A reliable detection without constraints must employ as much information available in the image.

Tracking hands is extremely difficult because they can move very fast and the appearance can change within a few frames. Due to the difficulty of tracking with the appearance of hands, colour information or background differencing technique may be used to obtain colour segmentation and which can produce good result on tracking objects.

Recognising a hand configuration is very difficulty and largely unsolved problem due to different appearance, habits and physical conditions. Traditional methods based on edge detection can achieve fairly good result, but even though an extensive processing is required [Chen].

4. ARM MANIPULATION SYSTEM

4.1 Introduction

An actuator normally has an energy input and a mechanical output such as displacement, velocity or force, and controlled by different kinds of auxiliary energies, for example, electrical, pneumatic or hydraulic supplies. This chapter describes in detail of a robot arm manipulator that has been chosen to act like human being. The arm manipulator uses signals from the object detected by the camera, the position signal converts to the robot control signal which drives the DC servo motors that leads the end effector to the desire position.

4.2 Arm manipulator specification

The arm manipulator chosen for this project is the Movemaster RV-M1 industrial programmable micro-robot system produced by Mitsubishi. Its primary function is to move to the desire position according to the information or command received, furthermore, it is able to perform some simple tasks with the end effector that instructed by the user.

RV-M1 is a standard 5-joints or 5 degrees of freedom arm manipulator which is obvious to see in an industrial environment. It is consisting of waist pitch and wrist roll just like a human arm.

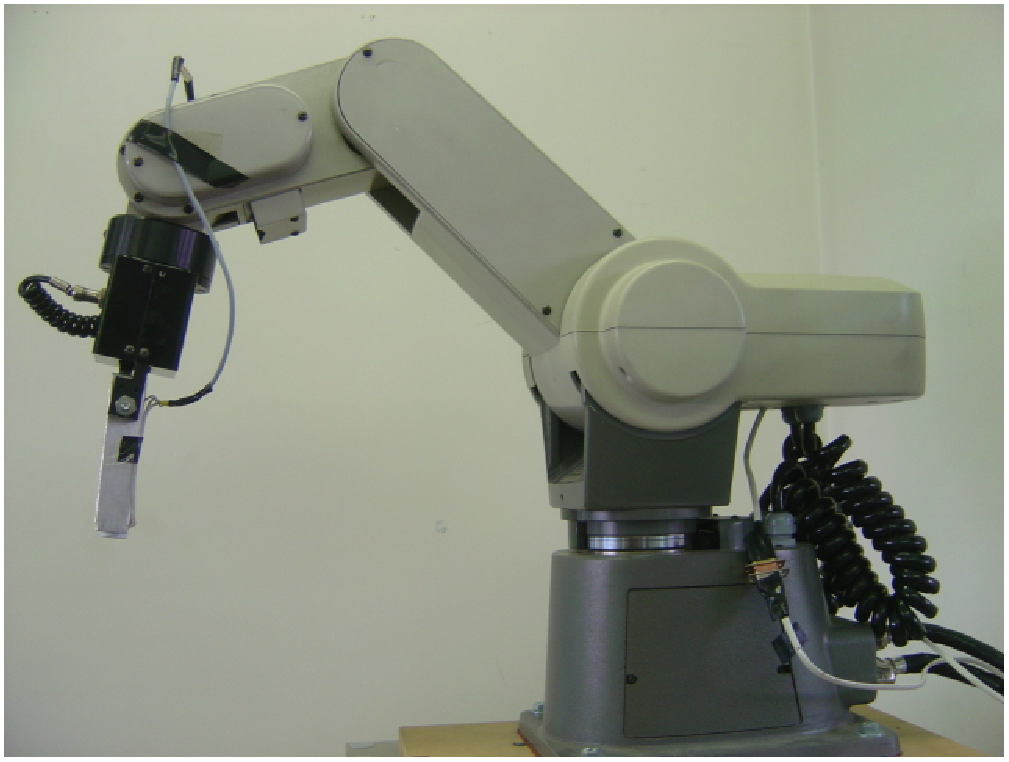
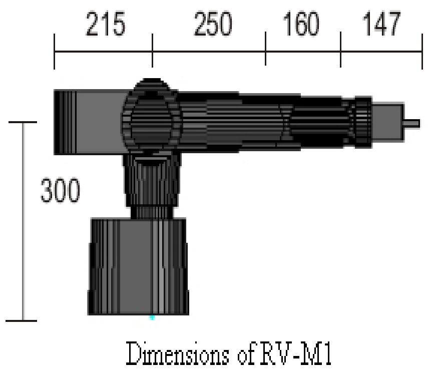


Figure 4.1 RV-M1 Movemaster arm manipulator

RV-M1 Movemaster is an arm manipulator that manufactured in 300mm height with upper arm 250mm and fore arm 160mm, which has similar length compares to the limb of human arm.



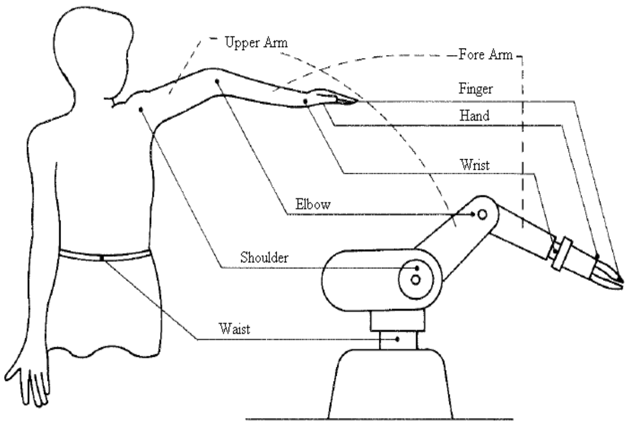


Figure 4.2 Dimensions of RV-M1

4.3 Drive unit and interfacing

The drive unit is the core of the robot system. Main CPU, RAM/EPROM memory and other electronic components are all located within drive unit, and also, it is a connection interface between robot to a PC and robot to a teaching box. In the front panel of the drive unit, there are switch buttons of emergency stop, program start and stop, and manual reset along with execute and error indicators which capable the users to observe the states of robot arm operation. Three sets of DIP switches are provided to set the baud rate and define positions in the Cartesian coordinate system. Heat sinks and ventilation openings are at the back of the drive unit, which allows protecting over heat that occurs to the circuitry.

Due to the operation requirement of the user, RV-M1 Movemaster offers a variety of interfacing options. It can be interfaced to a personal computer or a teaching box or to an external device such as microcontroller. According to the project requirement, an interface between Movemaster to a personal computer is considered. The drive unit makes two types of interfaces available for the link between Movemaster and a personal computer. These are the Centronics interface and the RS232 interface. The Centronics interface is a standard parallel transmission that establishes a very fast transmission speed, however, since Centronics interface is originally used in data transmission between a personal computer and a printer, according to that, the movemaster plays the same role as a printer, which means the communication is performed in one direction from personal computer to the robot only, and this kind of interface is restricted from the distances of 1 or 2 meters. RS232 is a common type of interface for data communication in the industry. Since data are sent through a single channel, it will cause a longer transmission time under a low transmission baud rate compare to parallel transmission. Instead, it is advantage in bidirectional data transfer that makes the personal computer capable of reading data from the robot, and moreover, serial communication also allows a longer transmission distance of 3 to 15 meters.

4.4 Operation space

RV-M1 has five joints driving by five DC servo motors individually, and each of them has their own operating range.

Joints	Motor	Range
Joint 1	Waist	300°
Joint 2	Shoulder	130°
Joint 3	Elbow	110°
Joint 4	Wrist Pitch	± 90°
Joint 5	Wrist Roll	±180°

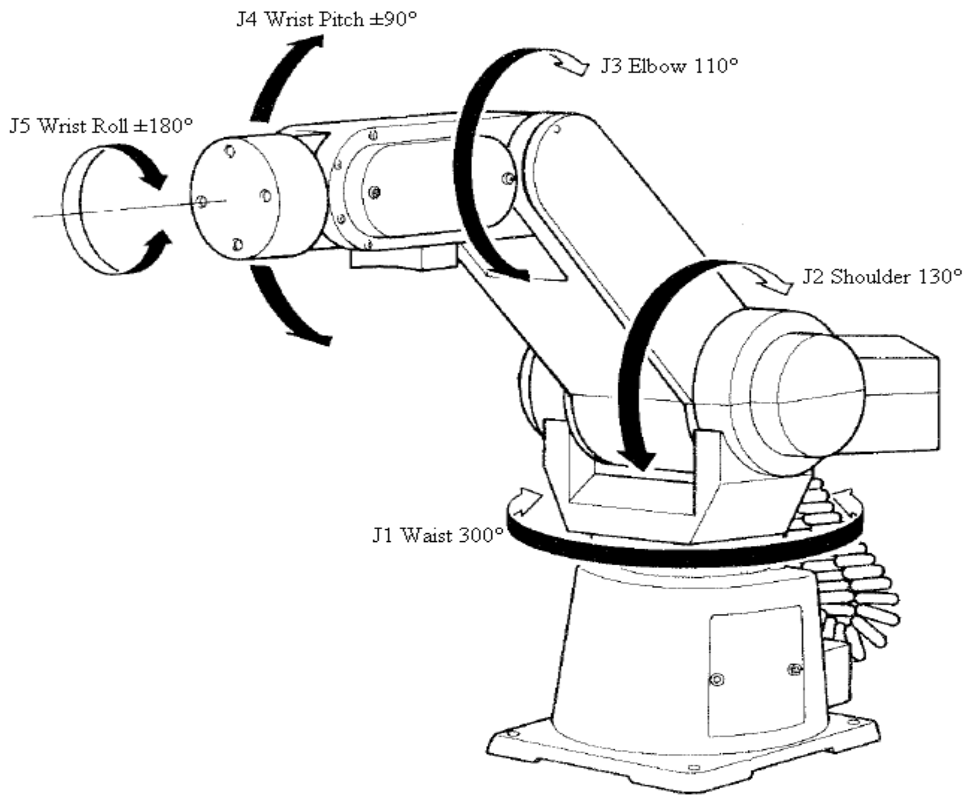


Figure 4.3 Operating range

According to above specification, an overall system operating space of RV-M1 Movemaster can be imagined.

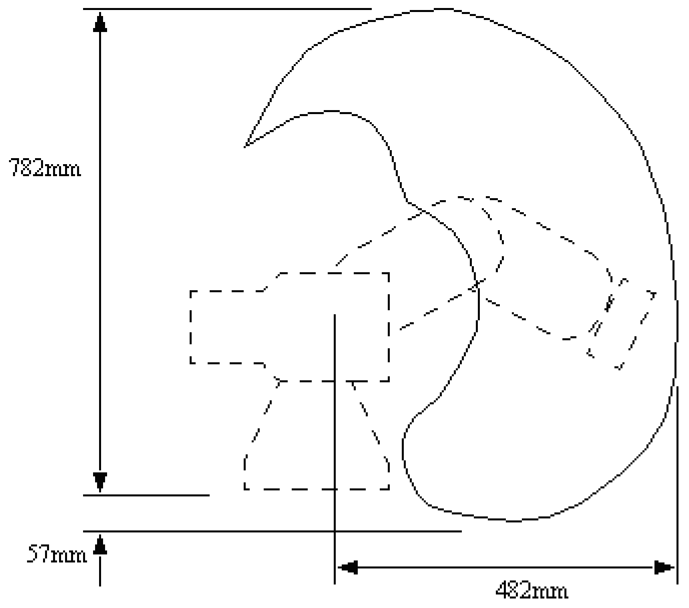


Figure 4.4 System operating space

The operating space is important in an arm manipulation system, not only because of the safety reason to the environment, but also, the drive unit will give an error message as the end effector moves to a position beyond the operating space, this will effect on the controlling and motion planning of the arm manipulator.

4.5 Control and commands

There are few ways to control the Movemaster arm actuator, according to how the commands been executed, control modes can be grouped as follow:

1. Direct Execution Mode:

   Control commands are used to move the robot directly. They are not executing in a form of program, instead, the commands are sent to the drive unit one at a time sequentially. An example below is showing how to use a simple command "MO" to control the Movamaster to a pre-defined position through RS232 interfacing.

```
MO 10
```

By executing above command which leads the Movemaster to the pre-defined position 10.

2. Program Execution Mode:

   Movemaster provides a space for users to program arm motion using EPROM memory locations. The program will transfer to drive unit for processing, as the program been transferred, the command "RN" starts the arm motion.

```
10 NT (Reset origin)
20 SP 9 (Set speed to 9)
30 MO 1 (Go to position 1)
40 MO 2 (Go to position 2)
50 TI 20 (Wait for 2 seconds)
60 GT 30 (Go to memory location 30)
70 ED (End program)
RN (Start the program)
```

3. Drive Unit Mode:

   This mode allows users to store program in the EPROM or RAM in the drive unit and execute by using the front panel start, stop and reset switches.

A handy tool teaching box is another option to control the Movemaster robot. Teaching box is mainly use for defining, checking and correcting positions of the robot arm. The teaching box can perform the following functions:

1. Reset the arm to home position

2. Define a position of the arm, particular in X,Y and Z coordinates
3. Move the arm to a pre-defined position.
4. Move the arm to any point in the Cartesian plane
5. Set the tool length
6. Open or Close the gripper
7. Emergency stop

The commands of Movemaster can be divided into 6 groups according to the functions they perform, they are:

1. Position and motion control instructions
2. Program control instructions
3. Hand control instructions
4. I/O control instructions
5. RS2232C read instructions
6. Miscellaneous

Position and motion control instructions are mainly used in this project, some descriptions and example of useful commands are given below:

1. HO (Home):

   A Cartesian coordinate system home position can be set by executing the HO command, and any consequent motion commands are executed under this reference coordinates, as the home reference position is set, it also can be accessed by executing OG command.

2. HE (Here):

   This command assigns a position number with current coordinates.

   10 MP 0, 350, 200, -60, 20 (Moves to the position of $X = 0$, $Y = 350$, $Z = 200$ with pitch angle $= -60°$ and roll angle$= 20°$)

20 HE 15 (Set above coordinate to position 15)

3. PD (Position Define):

   This command defines a Cartesian coordinate to a position number with pitch and roll angle.

   10 PD 15, 0, 350, 200, -60, 20 (Defines coordinate $X = 0$, $Y = 350$, $Z = 200$ with pitch angle $= -60°$ and roll angle$= 20°$ to position number 15)

   The format of the command is PD (position number), (*X* coordinate), (*Y* coordinate), (*Z* coordinate), (pitch angle), (roll angle).

4. MO (Move):

   This command lead the hand to the position defined by the command HE or PD.

   10 MO 15 (Moves the hand to defined position 15)

5. MP (Move Position):

   This command can move the hand to a specified coordinate position with pitch and roll angle directly.

   10 MP 0, 350, 200, -60, 20 (Moves to the position of $X = 0$, $Y = 350$, $Z = 200$ with pitch angle $= -60°$ and roll angle$= 20°$)

   The format of the command is MP (*X* coordinate), (*Y* coordinate), (*Z* coordinate), (pitch angle), (roll angle).

6. MJ (Move Joint):

   This command turns each joint of the arm to the specified angle from the current position.

   10 MJ +30, 0, 0, 0, 0 (Turns waist joint 30° in positive direction)

   20 MJ 0, -30, 0, 0, 0 (Turns shoulder joint 30° in negative direction)

The format of the command is MJ (waist angle), (shoulder angle), (elbow angle), (pitch angle), (roll angle).

7. OG (Origin):
   This command brings arm back to the origin reference position which sets by the HO command.

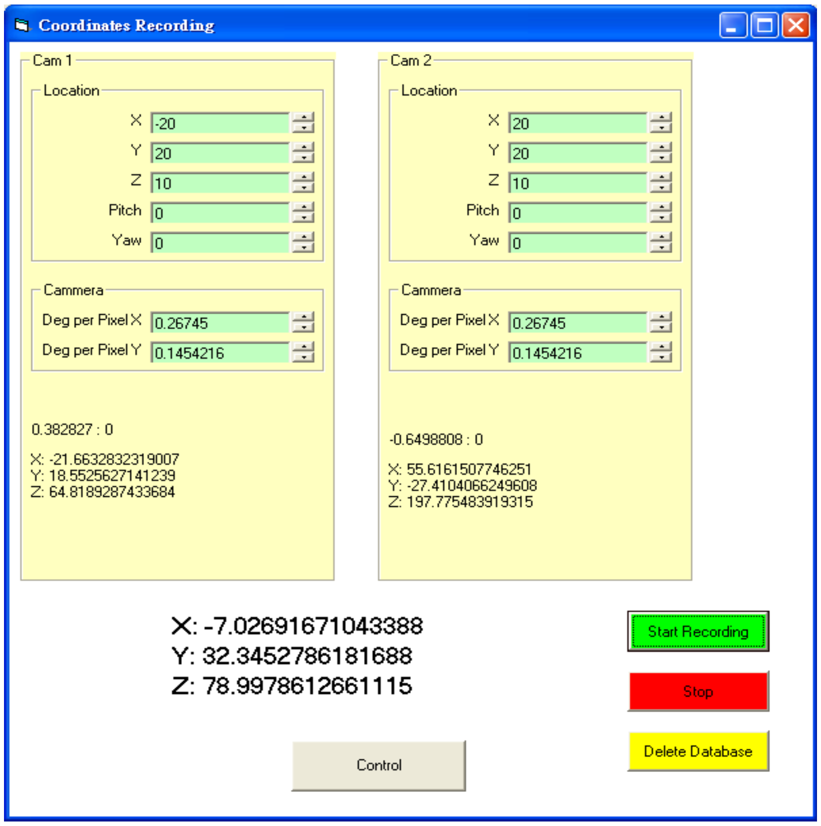
Above commands are most frequently used in the project, a proper use of other groups of commands may help to make the control motion more efficient, smoother and faster.

5. OVERALL OPERATING SYSTEM

5.1 Introduction

The details of computer vision and robot arm manipulator have been discussed in previous chapters. These two systems are now going to combine into an overall vision and motion system. Visual Basic 6 was chosen to develop the Graphic User Interfaces (GUIs) for the computer vision system and robot arm manipulator. Colour specification of the moving object (hand) was required for the computer vision system to start detecting, and an open space was necessary for the arm manipulator to avoid collision occurs.

5.2 GUI of computer vision system



Cam 1

Location

X -20
Y 20
Z 10
Pitch 0
Yaw 0

Cammera

Deg per Pixel X 0.26745
Deg per Pixel Y 0.1454216

0.382827 : 0

X: -21.6632832319007
Y: 18.5525627141239
Z: 64.8189287433684

Cam 2

Location

X 20
Y 20
Z 10
Pitch 0
Yaw 0

Cammera

Deg per Pixel X 0.26745
Deg per Pixel Y 0.1454216

-0.6498808 : 0

X: 55.6161507746251
Y: -27.4104066249608
Z: 197.775483919315

X: -7.02691671043388
Y: 32.3452786181688
Z: 78.9978612661115

Figure 5.1 GUI of computer vision system

Figure 5.1 shows the GUI of the computer vision system, it was established in the purpose to obtain the location of a specific object in Cartesian coordinate system. The first part of GUI was designed to set the position of the camera in X, Y, Z direction away from the static object. Boxes under camera location setting was used to adjust the resolution (how wide is a pixel contain) of the camera in X, Y direction. At the bottom the GUI shows the result of detection in Cartesian coordinate system. Three buttons on the right hand side were used to start or stop recoding coordinates into database, or the yellow button could be used to delete the coordinates which saved in database. The control button at the bottom allows the user to monitor the states of arm manipulator GUI.

A database was constructed using Microsoft Access and operates as a buffer which holds the data for further calculation process. Coordinates data are recorded to database in every 4000us approximately (can be adjusted in the computer vision program), since the camera system is very sensitive and the data recorded are in high accuracy relatively, an experiment shows that if data input directly into arm manipulator control which will cause an arm movement in a correct direction but low stability. Following data were recorded by the computer vision system as the user wearing a specific colour glove.

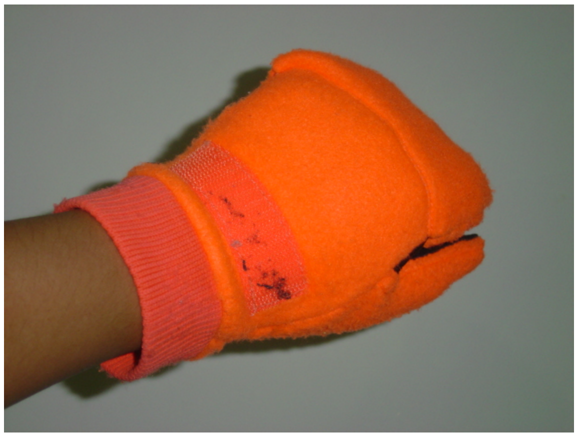


Figure 5.2 Specific colour glove

Traj_x (mm)	Traj_y (mm)	Traj_z (mm)		Traj_x (mm)	Traj_y (mm)	Traj_z (mm)
-196.6257912	353.1520975	79.6500964		16.0815434	348.0790454	77.4656587
-196.5464621	351.6753545	77.1484598		21.5274682	344.8231095	74.2654654
-196.7654191	356.5464327	79.5724236		29.3541087	346.6559354	77.0977462
-196.3246879	358.9765442	78.0982457		40.2644680	343.4565858	75.2564460
-214.1725411	356.0560064	82.6582689		54.3568652	345.9785139	78.6484339
-205.6451759	353.0968455	81.3551754		65.0785213	344.5668579	79.6553526
-206.7144747	350.2643654	82.0980894		77.7854234	344.1238603	79.8643544
-211.6654715	353.4721809	83.4644352		86.2548358	341.5617878	77.5476868
-203.9578540	357.4167678	84.5721465		94.1256707	344.2549873	79.8448234
-200.0986597	356.7549408	83.7699846		99.5676121	344.9786879	79.2408003
-186.5086513	356.4091501	82.2165489		104.6545124	344.1389755	79.8654430
-173.6250924	354.9176795	81.9546231		110.5478087	340.0659568	78.6571423
-163.6897652	354.4356457	79.3674915		121.0986548	340.4664254	78.8745625
-148.9815860	353.7653452	79.7531598		129.5407809	340.6970786	78.7645451
-142.5897918	355.9019874	81.2314658		143.2316549	340.8461232	78.8745245

-130.6986439	354.5379781	79.5642318		158.0678913	343.3278619	81.9645001
-124.7289953	356.3547648	82.6459872		168.5431098	343.4698757	81.0980398
-115.4468464	352.7409789	80.4861432		173.3217935	341.3426544	79.8907132
-102.8091239	351.5978543	77.3197870		182.7519598	340.5978709	77.8345340
-98.7626860	351.3568435	77.8965423		190.5982752	339.3757094	77.6970253
-89.1369813	351.1981597	78.4756645		196.8912567	343.7134207	80.9056754
-82.0981516	350.1898438	78.1237588		201.5845932	341.5089840	78.5978054
-72.3982358	349.8966556	78.3249908		209.7506843	340.1680065	77.9760534
-61.6076079	349.5315653	77.1236547		216.4265168	342.8396700	79.7874214
-52.1093850	348.2435656	76.8671284		218.0288215	338.3205082	77.8720342
-50.5123093	350.6543348	77.9807584		234.4523456	340.7060507	79.5056607
-43.8759764	351.2354640	77.3589845		241.7812876	338.2359781	76.9780636
-33.3572608	347.5408297	76.6546453		255.2350978	336.0604657	77.1325010
-24.2659477	347.7654253	75.8094509		255.5795498	334.3607808	78.0895065
-17.3652378	347.0273567	76.5378721		255.6757653	337.2456842	79.3469087
-7.6954243	346.3296708	75.5897545		260.4731275	340.3540528	80.7890460
-2.9765424	348.5615487	77.0854515		260.5443665	337.5568099	80.9803570
8.1609324	346.6526576	77.3654789				

Table 5.1 Trajectories recorded in database

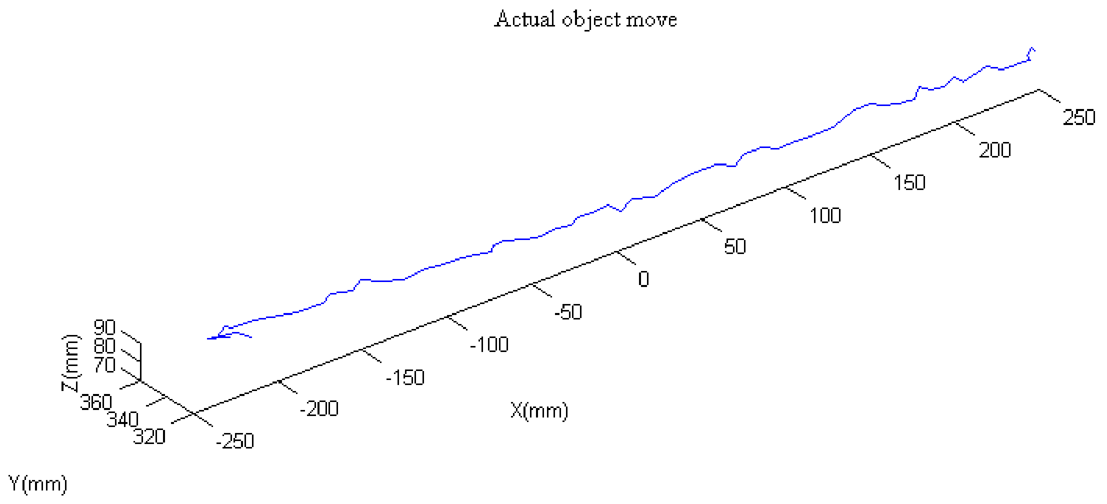


Figure 5.3 Trajectory of object move by inputting coordinates directly

Therefore, an advance data process is required to implement before input to the arm manipulator, this can be achieved by performing mathematic average algorithm to the data recorded. The average algorithm is calculated every three X, Y, Z coordinates repeatedly under following regulation:

first X, Y, Z averaged coordinate:

$$\frac{\left[(-196.6257912)+(-196.5464621)+(-196.7654191)\right]}{3}=-196.6$$

$$\frac{\left[(353.1520975)+(351.6753545)+(356.5464327)\right]}{3}=353.8$$

$$\frac{\left[(79.6500964)+(77.1484598)+(79.5724236)\right]}{3}=78.8$$

second X, Y, Z averaged coordinate:

$$\frac{\left[(-196.5464621)+(-196.7654191)+(-196.3246879)\right]}{3}=-196.5$$

$$\frac{\left[(351.6753545)+(356.5464327)+(358.9765442)\right]}{3}=355.7$$

$$\frac{\left[(77.1484598)+(79.5724236)+(78.0982457)\right]}{3}=78.3$$

third X, Y, Z averaged coordinate:

$$\frac{\left[(-196.7654191)+(-196.3246879)+(-214.1725411)\right]}{3}=-202.4$$

$$\frac{\left[(356.5464327)+(358.9765442)+(356.0560064)\right]}{3}=357.2$$

$$\frac{\left[(79.5724236)+(78.0982457)+(82.6582689)\right]}{3}=80.1$$

and so on…

Traj_x Avg (mm)	Traj_y Avg (mm)	Traj_z Avg (mm)		Traj_x Avg (mm)	Traj_y Avg (mm)	Traj_z Avg (mm)
-196.6	353.8	78.8		15.3	346.5	76.4
-196.5	355.7	78.3		22.3	346.5	76.3

-202.4	357.2	80.1
-205.4	356.0	80.7
-208.8	353.1	82.0
-208.0	352.3	82.3
-207.4	353.7	83.4
-205.2	355.9	83.9
-196.9	356.9	83.5
-186.7	356.0	82.6
-174.6	355.3	81.2
-162.1	354.4	80.4
-151.8	354.7	80.1
-140.8	354.7	80.2
-132.7	355.6	81.1
-123.6	354.5	80.9
-114.3	353.6	80.2
-105.7	351.9	78.6
-96.9	351.4	77.9
-90.0	350.9	78.2
-81.2	350.4	78.3
-72.0	349.9	77.9
-62.0	349.2	77.4
-54.7	349.5	77.3
-48.8	350.0	77.4
-42.6	349.8	77.3
-33.8	348.8	76.6
-25.0	347.4	76.3
-16.4	347.0	76.0
-9.3	347.3	76.4
-0.8	347.2	76.7
7.1	347.8	77.3
30.4	345.0	75.5
41.3	345.4	77.0
53.2	344.7	77.9
65.7	344.9	79.4
76.4	343.4	79.0
86.1	343.3	79.1
93.3	343.6	78.9
99.4	344.5	79.7
104.9	343.1	79.3
112.1	341.6	79.1
120.4	340.4	78.8
131.3	340.7	78.8
143.6	341.6	79.9
156.6	342.5	80.6
166.6	342.7	81.0
174.9	341.8	79.6
182.2	340.4	78.5
190.1	341.2	78.8
196.4	341.5	79.1
202.7	341.8	79.2
209.3	341.5	78.8
214.7	340.4	78.5
223.0	340.6	79.1
231.4	339.1	78.1
243.8	338.3	77.9
250.9	336.2	77.4
255.5	335.9	78.2
257.2	337.3	79.4
258.9	338.4	80.4

Table 5.2 Average of object trajectories

Average results are taken in one decimal place because the smallest input to the arm manipulator is 0.1mm.

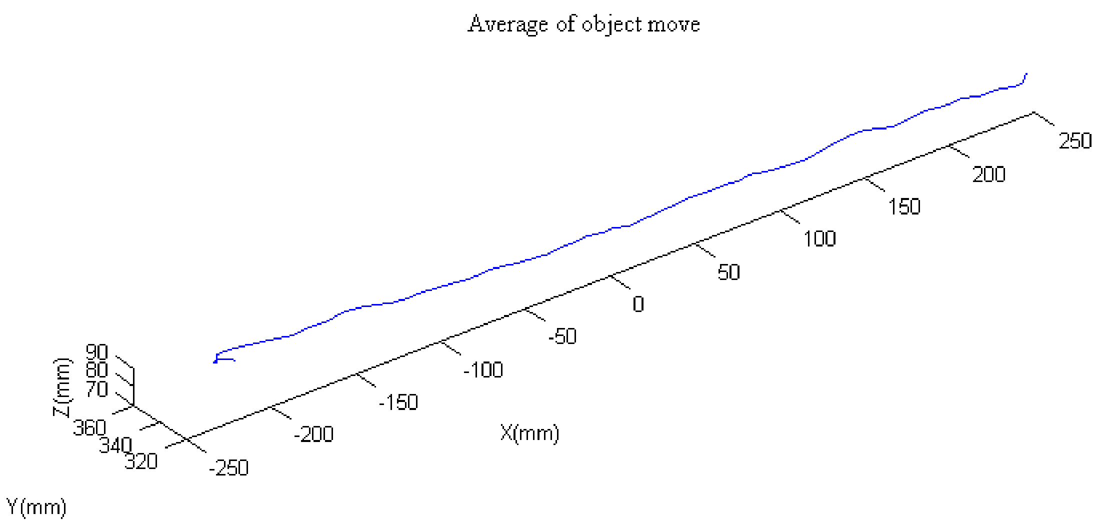


Figure 5.4 Object trajectory by inputting averaged coordinates

5.3 GUI of arm manipulator

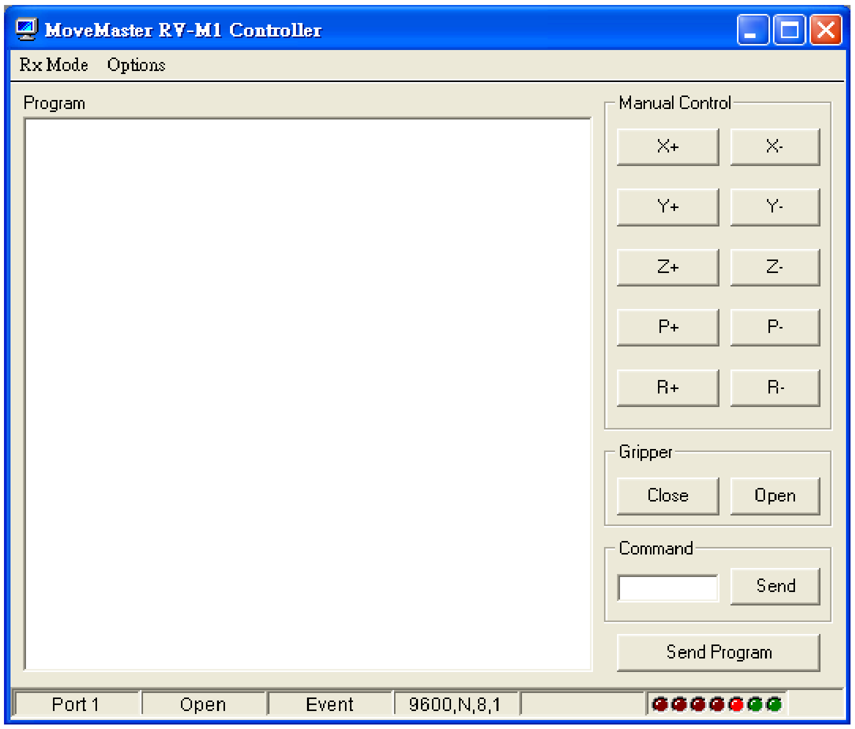


Figure 5.5 GUI of robot arm manipulator

The GUI was designed to communicate between user and arm manipulator. Control buttons at right hand side were used to control arm manipulator manually, each time the user press the X, Y, Z, Pitch or Roll button which cause the arm moves 10° in positive or negative direction. Gripper open or close button simply control the end-effector open or close. Command field allows the user to input command directly, that's the space where the averaged object coordinates to be entered for directing the arm manipulator to the same spot, and through the operation command MP (move to position) is employed.

5.4 System flowchart

There is a system flowchart underneath which helps to understand the overall operation of the catch and move system.

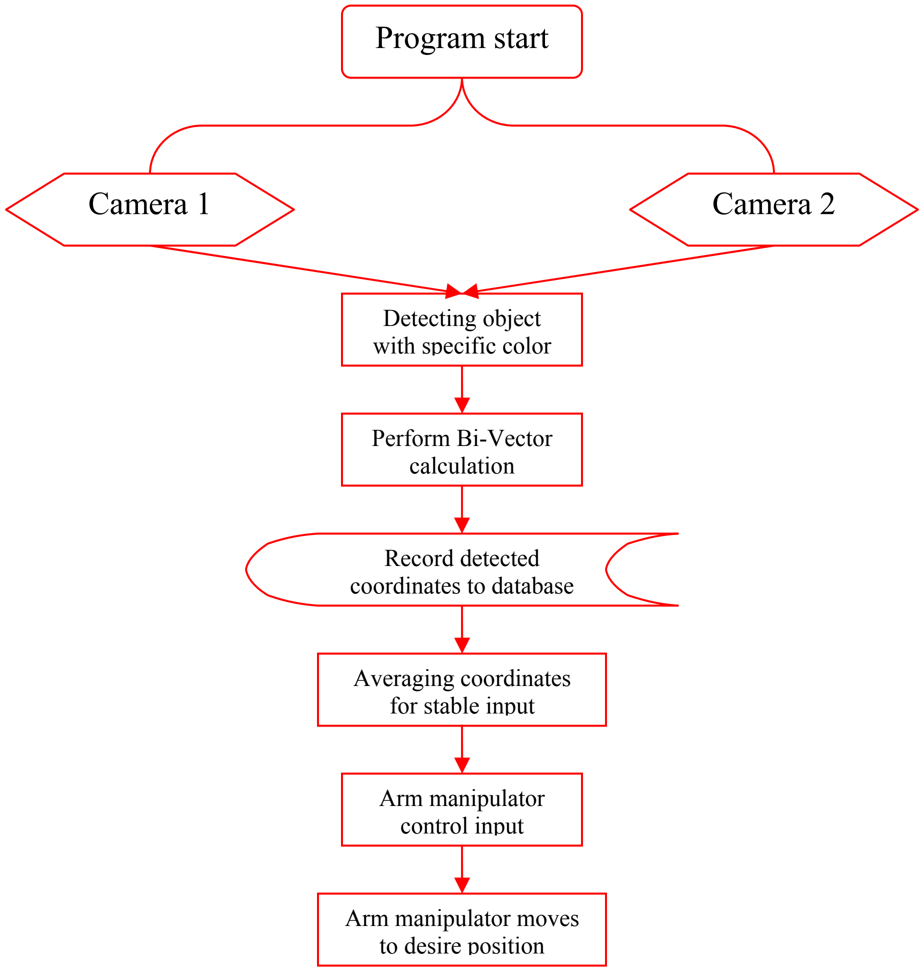
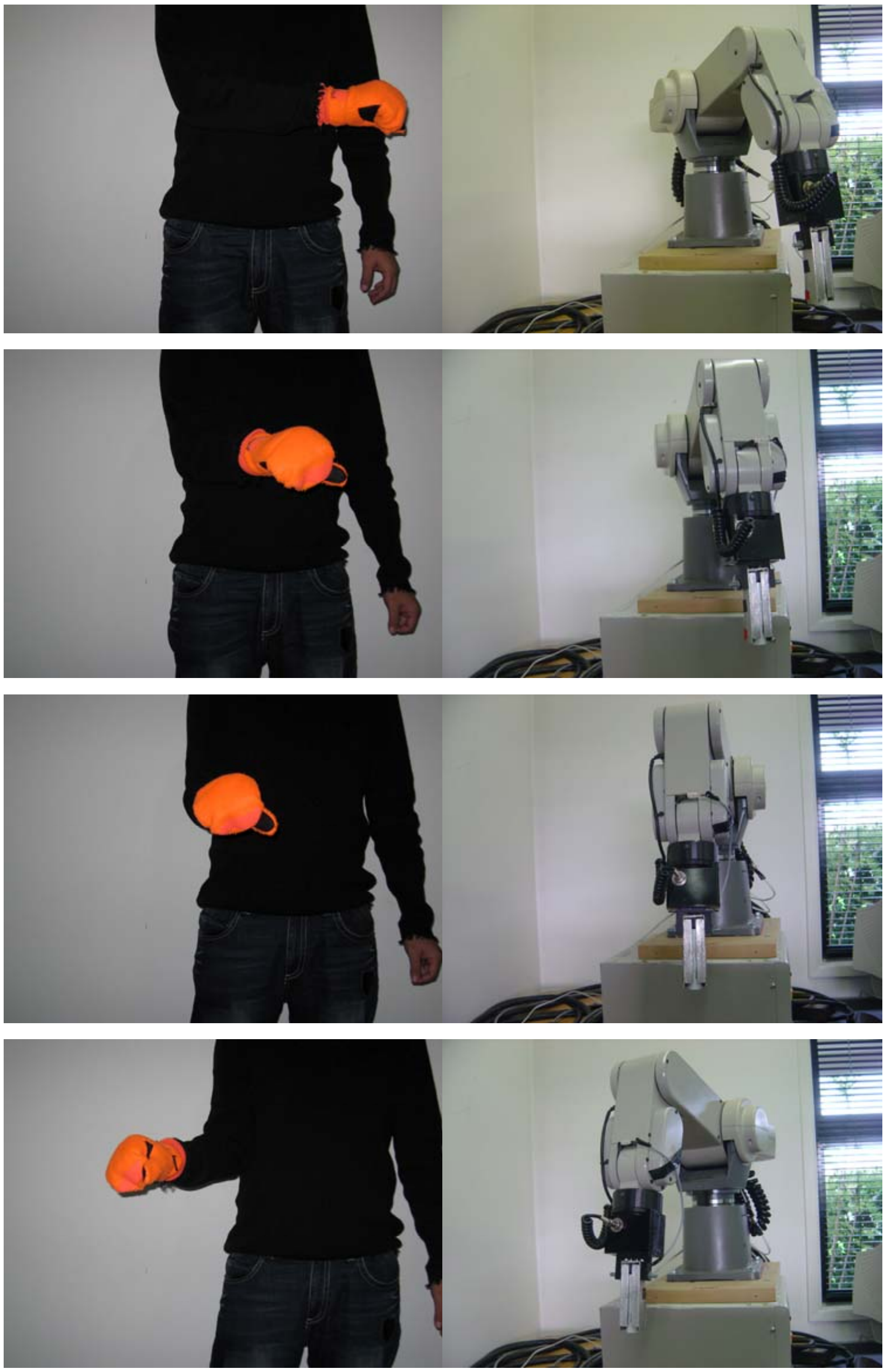

Program start
Camera 1
Camera 2
Detecting object with specific color
Perform Bi-Vector calculation
Record detected coordinates to database
Averaging coordinates for stable input
Arm manipulator control input
Arm manipulator moves to desire position

Figure 5.6 System flowchart

Based on the above system flowchart, a brief description can be made as follows, the computer vision system is used to detect and record the actual position of hand movement, the arm manipulator responds to the computer vision system data input by moving arm manipulator with desired motion trajectories.

5.5 Testing on the system

Based on section 5.2, averaged coordinates were entered the arm manipulator control system and drove to perform following movements.



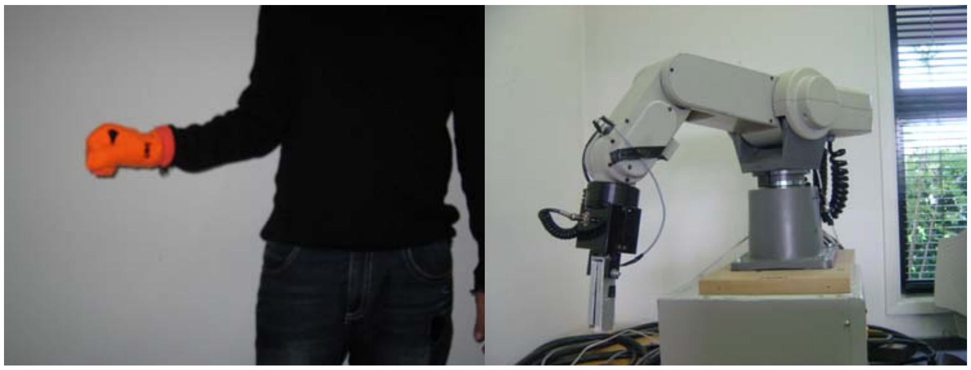
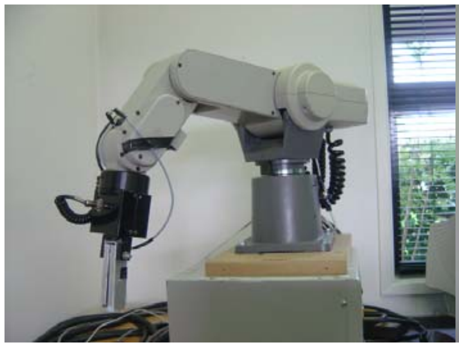
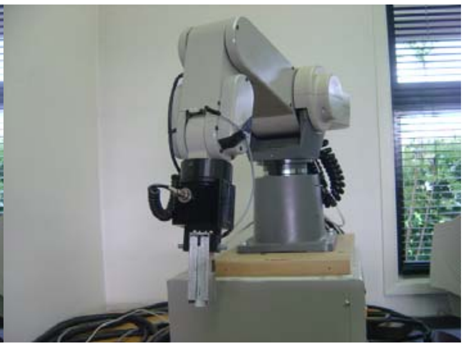


Figure 5.7 Actual robot arm movement



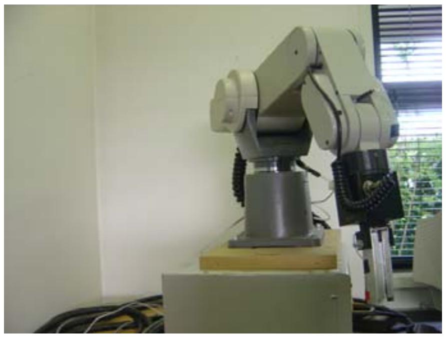
(a)



(b)

(c)

(d)



(e)

Figure 5.8 Arm manipulator movements with coordinates (a) X=258.9, Y=338.4, Z= 80.4 (b) X=104.9, Y=343.1, Z=79.3 (c) X=-0.8, Y=347.2, Z=76.7 (d) X=-105.7, Y=351.9, Z=78.6 (e) X=-196.6, Y=353.8, Z=78.8

In order to compare and proof the arm manipulator moved in the right motion planned, experimenter let the arm manipulator wear the specific glove as it moved and took record of the trajectories.

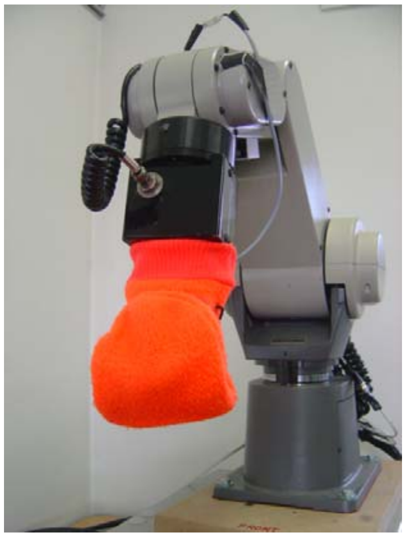


Figure 5.9 Arm manipulator with glove on

The results are as follow:

Traj_x (mm)	Traj_y (mm)	Traj_z (mm)	Traj_x (mm)	Traj_y (mm)	Traj_z (mm)
-270.7616	249.0490	105.6635	-20.4828	284.7370	113.0677
-270.6909	250.3724	105.5459	-2.9699	286.4318	113.1202
-270.4678	247.0006	105.4549	-2.9653	286.2430	113.1087
-268.3447	249.3378	106.1365	-2.7753	285.6685	113.2270
-244.5078	249.3089	103.6759	20.6108	275.5516	113.0337
-244.9202	248.9794	104.2172	18.9307	275.0720	112.0794
-244.3828	247.5810	103.9602	20.6686	277.7750	113.0483
-236.5441	231.8936	104.4806	39.7000	279.9718	113.0969
-236.2375	249.7596	104.5671	53.0302	278.0189	112.7403
-234.5927	250.9905	103.7885	53.5621	276.5252	112.1918
-209.8575	252.0206	103.8966	52.8685	280.6923	114.1719
-193.2059	254.7262	105.6232	101.7331	275.4364	110.9038
-194.5164	256.3792	106.5618	101.5173	274.2080	110.7106
-152.4859	261.9374	107.4397	99.6259	273.1102	109.9003
-152.7348	262.7147	107.0697	112.0322	273.1791	110.4724
-152.1442	264.0807	107.6630	112.8589	272.0702	110.1362
-146.5442	265.1998	106.4543	112.9854	273.8810	110.2790
-119.5384	273.4759	108.8540	144.3231	268.8512	108.5351
-119.6010	273.9056	109.1897	146.7945	267.7565	109.4971
-119.4031	274.4802	109.2142	144.6441	267.5958	108.7161
-67.6246	280.2641	112.4015	168.6161	265.5074	108.5207
-67.5883	279.4503	112.3171	167.5374	265.3242	108.2041
-67.6519	282.5188	112.4875	169.3055	264.9541	108.7997
-46.1506	275.1281	111.3524	198.0938	262.7926	108.2500
-46.4172	273.4610	111.1617	199.4611	264.1771	108.4673
-44.2473	276.8786	112.8289	201.5499	264.2985	109.0973
202.1156	266.5921	109.2716	273.7539	271.3067	111.8065
236.0436	267.6114	108.5183	307.5828	281.5151	118.9684
236.1523	266.6645	108.6023	306.6531	282.9948	118.7363
236.0895	267.5007	108.5413	307.0196	281.9377	118.8996
267.1439	270.4856	109.9614	306.7317	283.1106	118.7710
264.4425	270.5452	109.3802	307.0728	282.8774	118.9225
264.2668	270.0506	109.5328	355.2692	294.2481	129.3241
272.7039	270.8961	111.5561	355.7916	290.8316	129.5381
273.9088	271.2450	111.8521	355.2572	292.6641	129.3319

273.3027	270.5752	111.7367			

Table 5.3 Arm manipulator trajectories

plot of trajectories is shown below:

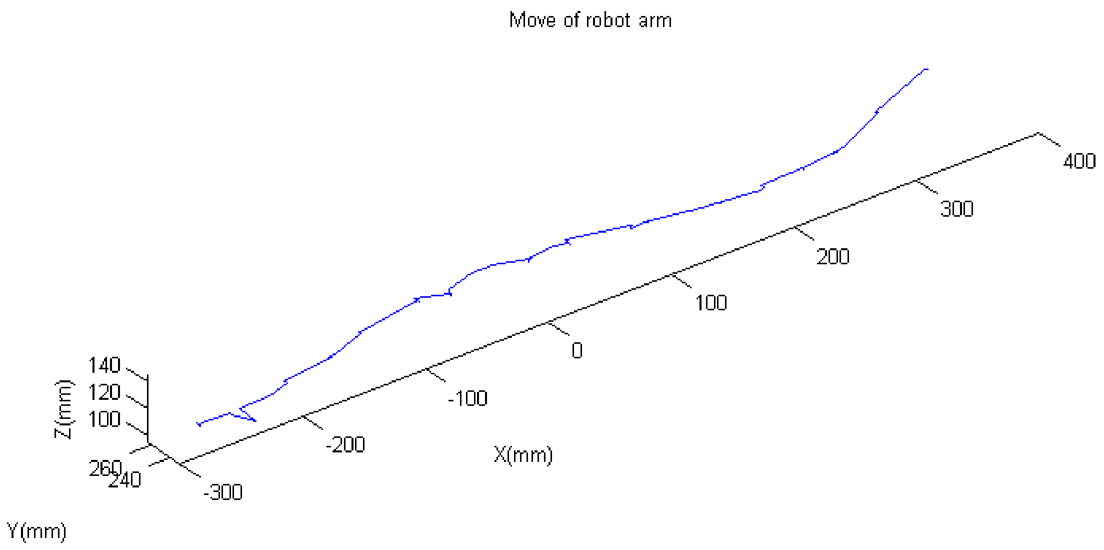


Figure 5.10 Trajectories of arm manipulator move

Spurs can be observed from the above figure, this means that there were vibrations while the arm was moving, and this was caused by the start and stop moving of arm manipulator. Furthermore, the figure above also showed different scale of Y-axis and Z-axis, this is because the arm manipulator was moving in different height and distance from the camera compare to human. As we adjusted the scale of the plot to the similar range of human hand, the plot can be obtained as follow, and spurs can be eliminated by averaging input trajectories:

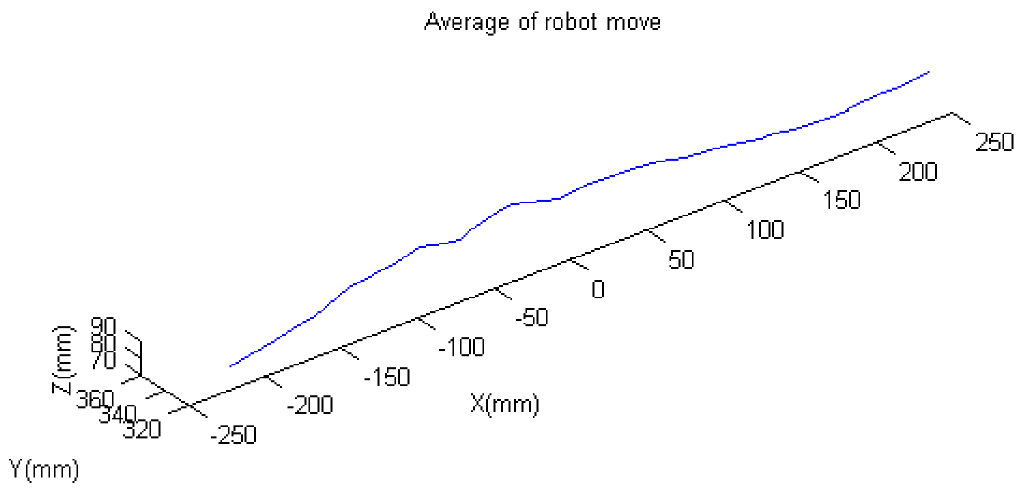


Figure 5.11 Trajectories of arm manipulator after averaging

Comparing two figures with each other, although they are not matching perfectly but an acceptable tolerance is considered.

6. CONCLUSION

The aim of project was to create a system which can locate and track an object (hand) in a three dimensional space, and using an arm manipulator to imitate hand motion base on a positioning vision system.

It has been shown that a computer vision system would be the best choice to achieve the goal due to cost, accuracy or complexity. Two cameras were used to locate the object in a wide field of view, image processing on frames were performed to fetch enough information to acquire accurate coordinates of a specific object. Few methods have been tested in order to make the object to be seen by the camera easily. Beacon lighting would be a reasonable option, but the wires connection may limit the moves of the object. Since the cameras are operating under colour base, a specific colour glove can be considered as wireless equipment and easy to distinguish from the background colour.

A database was established to record the trajectories of the object been through, these trajectories data were averaged by a mathematical algorithm in the purpose of giving a stable input to the arm manipulator, the more data average been taken the more smooth of the arm manipulator move.

A 3D 6-DOF Movemaster was employed to imitate the motion of the object. The physical structure and operating space are the two major properties as choosing a suitable manipulator. According to the data entered to the controller, Movemaster responded with a human-liked motion.

7. DISCUSSION AND FURTHER IMPROVEMENT

As expected a number of problems occurred during the project process, and some suggestion can be made for further system improvement.

The Movemaster is operating properly within its workspace, however the computer vision system only locates the object coordinates in X, Y, Z axes. Since the Movemaster is a 6-DOF arm manipulator, few coordinates input by computer vision system are unreachable due to the angle of pitch and roll, this makes the Movemaster showing an unrealistic wrist shape and returns an error message to the drive unit which forces the system stop executing commands afterwards. This problem can be solved by resetting the system manually to disregard the coordinate, or a limitation can be set to avoid the hand moves into that inaccessible area. Alternatively, extra sensors can be attached on the wrist to detect the pitch and roll angle, regrettably an additional cost for the device with complicated measuring algorithm is needed.

Hands are the primary apparatus for human to perform tasks. Most robot arms are not just designed to move but also get some tasks done, and Movemaster is one of them. It comes along with an end gripper which allows the arm manipulator to perform some task. At this stage, the vision system records the hand trajectories and makes the Movemaster to imitate human motion, if an additional click button is attached on the glove for gripper open/close control, the Movemaster will not just a human motion imitator and also a task finisher.

In 21st century, the internet has become part of everyday life. Internet service is not just a communication tool between human, it also can be used to transmit data from human to machine. In other words, remote control becomes feasible to the user and human can be replaced in some critical environment and get the task done.

REFERENCES

[Bhanu] Bhanu, B., & Pavlidis, I. (Eds.). (2005). *Computer vision beyond the visible spectrum.* London: Springer.

[Bicchi] Bicchi, A., & Christensen, H. I., & Prattichizzo, D. (Eds.). (2003). *Control problems in robotics.* New York: Springer.

[Chang] Chang, I. C., & Huang, C. L. (1996). IEEE proceedings of ICPR '96. *Ribbon-based motion analysis of human body movements,* 436-440.

[Chen] Chen, C. H., & Wang, P. S. P. (2005). *Handbook of pattern recognition and computer vision.* London: World scientific Publishing.

[Choset] Choset, H., & Lynch, K. M., & Hutchinson, S., & Kantor, G., & Burgard, W., & Kavraki, L. E., & Thrun, S. (2004). *Principles of robot motion.* England: MIT press.

[CSS] Computer System Support. (2008). *Robot Arm Workspace*. Retrieved from:

http://www.engineering.uiowa.edu/~sicei/public_html.ic41/rts.html

[Forsyth] Forsyth, D. A., & Mundy, J. L., & Gesu, V., & Cipolla, R. (Eds.). (1999). *Shape, contour and grouping in computer vision.* New York: Springer.

[Gonzalez] Gonzalez, R. C., & Woods, R. E. (2002). *Digital image processing.* New Jersey: Prentice Hall.

[Harris] Harris, L. R., & Jenkin, M. R. M. (2007). *Computational vision in neural and machine systems.* Cambridge: Cambridge University Press.

[Hartly] Hartley, R., & Zisserman, A. (2003). *Multiple view geometry in computer vision.* Cambridge: Cambridge University Press.

[Heijden] Heijden, F. (1994). *Image based measurement systems.* England: John Wiley & Sons.

[Hugh] Hugh, J. (2001). *Mitsubishi RV-M1 Manipulator*. Retrieved April 23, from: http://www.eod.gvsu.edu/eod/mechtron/mechtron-428.html

[Isler] Isler, V., & Khanna, S., & Spletzer, J., & Taylor, C. J. (2005). Computer vision and image understanding. *Target tracking with distributed sensors: the focus of attention problems,* 100, 225-247.

[Koller] Koller, D., & Klinker, G., & Rose, E., & Breen, D., &

Whitaker, R., & Tuceryan, M. (1997). In Proceedings of the Symposium on Virtual Reality Software and Technology. *Real-time Vision-Based Camera Tracking for Augmented Reality Applications,* Sep 15-17, 87-94.

[Liu] Liu, J. X. (2005). *Computer vision and robotics.* New York: Nova Science.

[Liu] Liu, J. X. (2005). *Mobile robots: new research.* New York: Nova Science.

[Lumelsky] Lumelsky, V. J. (2006). *Sensing, intelligence, motion: how robots and humans move in an unstructured world.* New Jersey: John Wiley & Sons.

[Macaire] Macaire, L., & Vandenbroucke, N., & Postaire, J. (2006). Computer vision and image understanding. *Color image segmentation by analysis of subset connectedness and color homogeneity properties,* 102, 105-116.

[Medioni] Medioni, G., & Kang, S. B. (2004). *Emerging topics in computer vision.* New Jersey: Prentice Hall.

[Motion Planning] Isto, P. (2003). *Motion planning.* Retrieved April 30, 2007, from: http://www.cs.hut.fi/~evp/Res/MP/motion_planning.html

[Multi Object Tracking] IEEE Workshop. (2001). *Multi-object tracking.* Retrieved June 7, 2007, from: http://research.microsoft.com/workshops/MultiObjectTracking/

[Parker]	Parker, J. R. (1997). *Algorithms for image processing and computer vision.* New York: Wiley Computer Publishing.
[Robot Arm Tutorial]	Society Of Robots. (2005). *Robot arm tutorial.* Retrieved May 7, 2007, from: http://www.societyofrobots.com/robot_arm_tutorial.shtml
[Rosenberg]	Rosenberg, C., & Nourbakhsh, I. (2003). *CMUCam2 manual.* U.S.A.: Anthony Rowe and Carnegie Mellon University.
[RRG]	Pickle, J. J. (2003). *Robot workspace.* Retrieved May 10, 2007, from: http://www.robotics.utexas.edu/rrg/learn_more/low_ed/workspaces/
[Sarfraz]	Sarfraz, M. (2005). *Computer-aided intelligent recognition techniques and applications.* England: John Wiley & Sons.
[Sebe]	Sebe, N., & Lew, M. S. (2003). *Robust computer vision: theory and applications.* Netherland: Kluwer Academic.
[Sebe]	Sebe, N., & Lew, M. S., & Huang, T. S. (Eds.). (2004). *Computer vision in human-computer interaction.* New York: Springer.
[Sonka]	Sonka, M., & Hlavac, V., & Boyle, R. (2008). *Image processing, analysis, and machine vision.* Toronto: Thomson Learning.

[Sturman]	Sturman, D. J., & Zeltzer, D. (1994). IEEE computer graphics and applications. *A survey of glove-based input,* 30-39.
[Wong]	Wong, K. Y., & Spetsakis, M. E. (2006). Computer vision and image understanding. *Tracking based motion segmentation under relaxed statistical assumptions,* 104, 45-64.
[Wechsler]	Wechsler, H. (2007). *Reliable face recognition methods: system design, implementation and evaluation.* New York: Springer.
[Zhang]	Zhang, Y. J. (2006). *Advances in image and video segmentation.* U.S.A.: IRM Press.

Appendix A
Object Tracking System
GUI

Main GUI

```
Public Cam1 As New frmCam
Public Cam2 As New frmCam
Dim interval_i As Integer

"*********Dim start**************
Dim dbs_cd As Database
Dim db_cd As String
Dim table_cd As String
Dim rst_cd As Recordset
Dim stop_f As Boolean
"*********Dim start**************

Private viewRadius As Double


Private Function GetBiVectorCoords() As D3DVECTOR

    Dim t As Double
    Dim t2 As Double
    Dim s As Double
    Dim s2 As Double

    Dim X As Double
    Dim Y As Double
    Dim z As Double


    t2 = (Cam2.TraceX ^ 2 * Cam1.TraceY ^ 2 + Cam2.TraceX ^ 2 * Cam1.TraceZ ^ 2 - 2
* Cam2.TraceX * Cam2.TraceY * Cam1.TraceY * Cam1.TraceX - 2 * Cam2.TraceX *
Cam2.TraceZ * Cam1.TraceZ * Cam1.TraceX + Cam2.TraceY ^ 2 * Cam1.TraceX ^ 2 +
Cam2.TraceY ^ 2 * Cam1.TraceZ ^ 2 - 2 * Cam2.TraceY * Cam2.TraceZ *
Cam1.TraceZ * Cam1.TraceY + Cam2.TraceZ ^ 2 * Cam1.TraceX ^ 2 + Cam2.TraceZ ^
2 * Cam1.TraceY ^ 2)
    s2 = (Cam1.TraceX ^ 2 + Cam1.TraceY ^ 2 + Cam1.TraceZ ^ 2)

    If (t2 = 0) Or (s2 = 0) Then _
        Exit Function

    t = (Cam2.TraceX * Cam1.LocationX * Cam1.TraceY ^ 2 + Cam2.TraceX *
Cam1.LocationX * Cam1.TraceZ ^ 2 - Cam2.TraceX * Cam1.TraceX * Cam1.TraceY *
Cam1.LocationY + Cam2.TraceX * Cam1.TraceX * Cam1.TraceY * Cam2.LocationY -
Cam2.TraceX * Cam1.TraceX * Cam1.TraceZ * Cam1.LocationZ + Cam2.TraceX *
Cam1.TraceX * Cam1.TraceZ * Cam2.LocationZ - Cam2.TraceX * Cam2.LocationX *
Cam1.TraceY ^ 2 - Cam2.TraceX * Cam2.LocationX * Cam1.TraceZ ^ 2 +
Cam2.TraceY * Cam1.LocationY * Cam1.TraceX ^ 2 + Cam2.TraceY *
Cam1.LocationY * Cam1.TraceZ ^ 2 - Cam2.TraceY * Cam1.TraceY * Cam1.TraceX *
Cam1.LocationX + Cam2.TraceY * Cam1.TraceY * Cam1.TraceX * Cam2.LocationX -
Cam2.TraceY * Cam1.TraceY * Cam1.TraceZ * Cam1.LocationZ + Cam2.TraceY *
Cam1.TraceY * Cam1.TraceZ * Cam2.LocationZ - Cam2.TraceY * Cam2.LocationY *
Cam1.TraceX ^ 2 _
   - Cam2.TraceY * Cam2.LocationY * Cam1.TraceZ ^ 2 + Cam2.TraceZ *
Cam1.LocationZ * Cam1.TraceX ^ 2 + Cam2.TraceZ * Cam1.LocationZ * Cam1.TraceY
^ 2 - Cam2.TraceZ * Cam1.TraceZ * Cam1.TraceX * Cam1.LocationX + Cam2.TraceZ *
Cam1.TraceZ * Cam1.TraceX * Cam2.LocationX - Cam2.TraceZ * Cam1.TraceZ *
Cam1.TraceY * Cam1.LocationY + Cam2.TraceZ * Cam1.TraceZ * Cam1.TraceY *
Cam2.LocationY - Cam2.TraceZ * Cam2.LocationZ * Cam1.TraceX ^ 2 - Cam2.TraceZ
* Cam2.LocationZ * Cam1.TraceY ^ 2)

   t = t / t2

   s = (-Cam1.TraceX * Cam1.LocationX + Cam1.TraceX * Cam2.LocationX +
Cam1.TraceX * Cam2.TraceX * t - Cam1.TraceY * Cam1.LocationY + Cam1.TraceY *
Cam2.LocationY + Cam1.TraceY * Cam2.TraceY * t - Cam1.TraceZ * Cam1.LocationZ
+ Cam1.TraceZ * Cam2.LocationZ + Cam1.TraceZ * Cam2.TraceZ * t)

   s = s / s2

   X = (Cam1.LocationX + Cam1.TraceX * s + Cam2.LocationX + Cam2.TraceX * t) / 2
   Y = (Cam1.LocationY + Cam1.TraceY * s + Cam2.LocationY + Cam2.TraceY * t) / 2
   z = (Cam1.LocationZ + Cam1.TraceZ * s + Cam2.LocationZ + Cam2.TraceZ * t) / 2


   GetBiVectorCoords = vec3(X, Y, z)

   ObjectPos.X = X

   ObjectPos.Y = Y
   ObjectPos.z = z

   Me.lblCombined = ("X: " & X & vbCr & "Y: " & Y & vbCr & "Z: " & z)
   "***********reccording to database***********
   If interval_i = 500 Then
      rst_cd.AddNew
      rst_cd.Fields("traj_x") = ObjectPos.X
      rst_cd.Fields("traj_y") = ObjectPos.Y
      rst_cd.Fields("traj_z") = ObjectPos.z
      rst_cd.Update
      interval_i = 0

"**************************Control*****************************
```

```
control_x = Int(ObjectPos.X * 10) / 10
control_y = ObjectPos.Y
control_z = ObjectPos.z
"1 Transfer unit



"2 call control bottom
frmMain.Text1 = "MP " & Chr(control_x) & ", " & Chr(control_y) & ", " &
Chr(control_z)
Click.frmMain.SendBottom

"**************************Control End****************************




   Else
      interval_i = interval_i + 1
   End If
   "***********reccording to database finish***********

End Function

Private Sub LostTrace()
   Dim tmpAngles As D3DVECTOR2
   tmpAngles = GetPitchYaw(vec3(Cam1.LocationX, Cam1.LocationY,
Cam1.LocationZ), vec3(Cam2.SingleVectorX, Cam2.SingleVectorY,
Cam2.SingleVectorZ))
   Me.lblCam1_2.Caption = tmpAngles.X & " : " & Cam1.ServoYaw.CurrentAngle
   If (Timer - Cam1.LastSeenTime) > 750 And (Timer - Cam2.LastSeenTime) < 300
Then
      Cam1.LastSeenTime = Timer - 300
      Cam1.ServoYaw.GoalAngle = tmpAngles.X
      Cam1.ServoPitch.GoalAngle = tmpAngles.Y
      Cam1.RequestServoUpdate = True
   End If


   tmpAngles = GetPitchYaw(vec3(Cam2.LocationX, Cam2.LocationY,
Cam2.LocationZ), vec3(Cam1.SingleVectorX, Cam1.SingleVectorY,
Cam1.SingleVectorZ))
   Me.lblCam2_2.Caption = tmpAngles.X & " : " & Cam2.ServoYaw.CurrentAngle
   If (Timer - Cam2.LastSeenTime) > 750 And (Timer - Cam1.LastSeenTime) < 300
Then
      Cam2.LastSeenTime = Timer - 300
      Cam2.ServoYaw.GoalAngle = tmpAngles.X
      Cam2.ServoPitch.GoalAngle = tmpAngles.Y
      Cam2.RequestServoUpdate = True
   End If

End Sub



Private Sub Command1_Click()

Delete_command_Click

stop_f = True
   Do While stop_f
      Timer = GetTickCount
      'GUI.Render
      DoEvents

      Cam1.Main
      Cam2.Main

      LostTrace

      ''Me.lblCam1_1.Caption = Cam1.ServoYaw.CurrentAngle
      ''Me.lblCam2_1.Caption = Cam2.ServoYaw.CurrentAngle
      Me.lblCam1_3.Caption = ("X: " & Cam1.SingleVectorX & vbCr & "Y: " &
Cam1.SingleVectorY & vbCr & "Z: " & Cam1.SingleVectorZ)
      Me.lblCam2_3.Caption = ("X: " & Cam2.SingleVectorX & vbCr & "Y: " &
Cam2.SingleVectorY & vbCr & "Z: " & Cam2.SingleVectorZ)


      GetBiVectorCoords

   Loop

End Sub

Private Sub Command3_Click()
   lblCombined.Caption = ""
   stop_f = False
   lblCombined.Caption = ""
End Sub
```

```
Private Sub Delete_command_Click()
Dim ci As Double
Dim cj As Integer
Dim rst_delall As Recordset

Set rst_delall = dbs_cd.OpenRecordset("select * from " & table_cd, dbOpenDynaset)

cj = rst_delall.RecordCount
   Do While rst_delall.RecordCount > 0
         rst_delall.MoveFirst
         rst_delall.Delete
         rst_delall.MoveNext
   Loop


End Sub

Private Sub Form_Load()
   Me.Show
   frmMain.Show
   frmMain.Visible = False

   Dim tmpAngles As D3DVECTOR2

   'GUI.Initialise (GUIBox.hWnd)

 ' *****************Connected to Database Start*****************

db_cd = "trajectorynew.mdb"
table_cd = "trajectory1_t"
Set dbs_cd = OpenDatabase(AppDir & db_cd)
Set rst_cd = dbs_cd.OpenRecordset("select * from " & table_cd, dbOpenDynaset)
interval_i = 0
' *****************Connected to Database Finish*****************


   Call Cam1.Init(1, -18, 20, 0, High)
   Call Cam2.Init(2, 19, 20, 0, High)

   LoadProperties


End Sub

Private Sub Form_Unload(Cancel As Integer)
```

```
   SaveProperties

End Sub


'----------------- Boring Variable Stuff ------------------

Private Sub SaveProperties()
   Open "Properties.dat" For Output As #1

      Write #1, NumboxCam1X.Value
      Write #1, NumboxCam1Y.Value
      Write #1, NumboxCam1Z.Value
      Write #1, NumboxCam1Pitch.Value
      Write #1, NumboxCam1Yaw.Value
''     Write #1, NumboxCam1ServoPitch.Value
''     Write #1, NumboxCam1ServoYaw.Value
      Write #1, NumboxCam1DegPixX.Value
      Write #1, NumboxCam1DegPixY.Value
''     Write #1, NumboxCam1AlphaX.Value
''     Write #1, NumboxCam1AlphaY.Value



      Write #1, NumboxCam2X.Value
      Write #1, NumboxCam2Y.Value
      Write #1, NumboxCam2Z.Value
      Write #1, NumboxCam2Pitch.Value
      Write #1, NumboxCam2Yaw.Value
''     Write #1, NumboxCam2ServoPitch.Value
''     Write #1, NumboxCam2ServoYaw.Value
      Write #1, NumboxCam2DegPixX.Value
      Write #1, NumboxCam2DegPixY.Value
''     Write #1, NumboxCam2AlphaX.Value
 ''    Write #1, NumboxCam2AlphaY.Value

   Close #1

End Sub

Private Sub LoadProperties()

   Dim tmp
```

```
  Open "Properties.dat" For Input As #1

      Line Input #1, tmp
      NumboxCam1X.Value = tmp

      Line Input #1, tmp
      NumboxCam1Y.Value = tmp

      Line Input #1, tmp
      NumboxCam1Z.Value = tmp

      Line Input #1, tmp
      NumboxCam1Pitch.Value = tmp

      Line Input #1, tmp
      NumboxCam1Yaw.Value = tmp

''      Line Input #1, tmp
''      NumboxCam1ServoPitch.Value = tmp

''      Line Input #1, tmp
''      NumboxCam1ServoYaw.Value = tmp

      Line Input #1, tmp
      NumboxCam1DegPixX.Value = tmp


      Line Input #1, tmp
      NumboxCam1DegPixY.Value = tmp

''      Line Input #1, tmp
''      NumboxCam1AlphaX.Value = tmp

''      Line Input #1, tmp
''      NumboxCam1AlphaY.Value = tmp


      Line Input #1, tmp
      NumboxCam2X.Value = tmp

      Line Input #1, tmp
      NumboxCam2Y.Value = tmp

      Line Input #1, tmp
      NumboxCam2Z.Value = tmp
```

```
        Line Input #1, tmp
        NumboxCam2Pitch.Value = tmp

        Line Input #1, tmp
        NumboxCam2Yaw.Value = tmp

''      Line Input #1, tmp
''      NumboxCam2ServoPitch.Value = tmp

''      Line Input #1, tmp
''      NumboxCam2ServoYaw.Value = tmp

        Line Input #1, tmp
        NumboxCam2DegPixX.Value = tmp

        Line Input #1, tmp
        NumboxCam2DegPixY.Value = tmp

 ''     Line Input #1, tmp
 ''     NumboxCam2AlphaX.Value = tmp

''      Line Input #1, tmp
''      NumboxCam2AlphaY.Value = tmp



    Close #1

End Sub


Private Sub GUIBox_MouseMove(Button As Integer, Shift As Integer, X As Single, Y
As Single)

If Button = 1 Then
    Call SetViewCam(vec3(Sin(X / 100) * viewRadius * Sin(Y / 100), Cos(Y / 100) *
viewRadius, Cos(X / 100) * viewRadius * Sin(Y / 100)), vec3(0, 0, 0))
ElseIf Button = 2 Then
    viewRadius = Y / 10

    Call SetViewCam(vec3(1 * viewRadius, 1 * viewRadius, 0 * viewRadius), vec3(0, 0,
0))

End If
End Sub


'----------Cam 1 -----------------

Private Sub NumboxCam1AlphaY_Change()
   If NumboxCam1AlphaY.Validate Then _
      Cam1.FilterAlphaY = NumboxCam1AlphaY.Value
End Sub

Private Sub NumboxCam1AlphaX_Change()
   If NumboxCam1AlphaX.Validate Then _
      Cam1.FilterAlphaX = NumboxCam1AlphaX.Value

End Sub

Private Sub NumboxCam1DegPixy_Change()
   If NumboxCam1DegPixY.Validate Then _
      Cam1.YPixelsPerRad = NumboxCam1DegPixY.Value

End Sub


Private Sub NumboxCam1DegPixX_Change()

   If NumboxCam1DegPixX.Validate Then _
      Cam1.XPixelsPerRad = NumboxCam1DegPixX.Value

End Sub

Private Sub NumboxCam1ServoYaw_Change()
   If NumboxCam1ServoYaw.Validate Then
      Cam1.ServoYaw.Offset = NumboxCam1ServoYaw.Value
      Cam1.RequestServoUpdate = True
   End If
End Sub

Private Sub NumboxCam1ServoPitch_Change()
   If NumboxCam1ServoPitch.Validate Then
      Cam1.ServoPitch.Offset = NumboxCam1ServoPitch.Value
      Cam1.RequestServoUpdate = True
   End If
End Sub
```

```
Private Sub NumboxCam1X_Change()
   If NumboxCam1X.Validate Then _
      Cam1.LocationX = NumboxCam1X.Value
End Sub

Private Sub NumboxCam1Y_Change()
   If NumboxCam1Y.Validate Then _
      Cam1.LocationY = NumboxCam1Y.Value
End Sub
Private Sub NumboxCam1Z_Change()
   If NumboxCam1Z.Validate Then _
      Cam1.LocationZ = NumboxCam1Z.Value
End Sub


'----------Cam 2 -----------------

Private Sub NumboxCam2AlphaY_Change()
   If NumboxCam2AlphaY.Validate Then _
      Cam2.FilterAlphaY = NumboxCam2AlphaY.Value
End Sub

Private Sub NumboxCam2AlphaX_Change()
   If NumboxCam2AlphaX.Validate Then _
      Cam2.FilterAlphaX = NumboxCam2AlphaX.Value

End Sub

Private Sub NumboxCam2DegPixy_Change()
   If NumboxCam2DegPixY.Validate Then _
      Cam2.YPixelsPerRad = NumboxCam2DegPixY.Value

End Sub


Private Sub NumboxCam2DegPixX_Change()
   If NumboxCam2DegPixX.Validate Then _
      Cam2.XPixelsPerRad = NumboxCam2DegPixX.Value

End Sub

Private Sub NumboxCam2ServoYaw_Change()
   If NumboxCam2ServoYaw.Validate Then
      Cam2.ServoYaw.Offset = NumboxCam2ServoYaw.Value
      Cam2.RequestServoUpdate = True
   End If
End Sub

Private Sub NumboxCam2ServoPitch_Change()
   If NumboxCam2ServoPitch.Validate Then
      Cam2.ServoPitch.Offset = NumboxCam2ServoPitch.Value
      Cam2.RequestServoUpdate = True
   End If
End Sub

Private Sub NumboxCam2X_Change()
   If NumboxCam2X.Validate Then _
      Cam2.LocationX = NumboxCam2X.Value
End Sub

Private Sub NumboxCam2Y_Change()
   If NumboxCam2Y.Validate Then _
      Cam2.LocationY = NumboxCam2Y.Value
End Sub

Private Sub NumboxCam2Z_Change()
   If NumboxCam2Z.Validate Then _
      Cam2.LocationZ = NumboxCam2Z.Value
End Sub

Private Sub RobFmDisplay_bottom_Click()
   frmMain.Show
   Me.Visible = False

End Sub
```

Robot Control GUI

```
Option Explicit

' program logic control
Private bLocalEcho            As Boolean
Private bMessageMode          As Boolean

' constants for setting the LED images,
' used as an index for imgLED()
Private Const RedOff          As Long = 0
Private Const RedOn           As Long = 1
Private Const GreenOff        As Long = 2
Private Const GreenOn         As Long = 3
```

```
' the sendmessage API is used to write
' to the textbox to reduce flicker, this
' not required for serial communications.

' Win32 API constants
Private Const EM_GETSEL         As Long = &HB0
Private Const EM_SETSEL         As Long = &HB1
Private Const EM_GETLINECOUNT   As Long = &HBA
Private Const EM_LINEINDEX      As Long = &HBB
Private Const EM_LINELENGTH     As Long = &HC1
Private Const EM_LINEFROMCHAR   As Long = &HC9
Private Const EM_SCROLLCARET    As Long = &HB7
Private Const WM_SETREDRAW      As Long = &HB
Private Const WM_GETTEXTLENGTH  As Long = &HE

' Win32 API declarations
Private Declare Function SendMessage Lib "user32" Alias "SendMessageA" _
   (ByVal hwnd As Long, ByVal wMsg As Long, ByVal wParam As Long, lParam As
Long) As Long


Private Sub cGripperClose_Click()
   If MSComm1.PortOpen Then MSComm1.Output = "GC" & vbCrLf
End Sub

Private Sub cGripperOpen_Click()
   If MSComm1.PortOpen Then MSComm1.Output = "GO" & vbCrLf
End Sub

Private Sub Command1_Click()

Dim command As String
command = Text1.Text

If MSComm1.PortOpen Then MSComm1.Output = command & vbCrLf

End Sub

Private Sub Command2_Click()

If MSComm1.PortOpen Then MSComm1.Output = "MJ +10,0,0,0,0" & vbCrLf

End Sub

Private Sub Command3_Click()
```

```
If MSComm1.PortOpen Then MSComm1.Output = "MJ -10,0,0,0,0" & vbCrLf

End Sub



Private Sub Command4_Click()
Dim a As String
If MSComm1.PortOpen Then MSComm1.Output = "wh" & vbCrLf
a = MSComm1.Input
Text2.Text = Text2.Text & a
If MSComm1.PortOpen Then MSComm1.Output = "rs" & vbCrLf

End Sub

Private Sub cSendProgram_Click()

Dim I As Integer
Dim Data As Integer
Dim programline As Variant
Dim IndividualLine As String

If tProgram.Text = "" Then
   MsgBox "No program to send!!!"
   Exit Sub
End If

programline = Split(tProgram.Text, vbCrLf)
   For I = LBound(programline) To UBound(programline)
      IndividualLine = programline(I)
      If MSComm1.PortOpen Then MSComm1.Output = IndividualLine & vbCrLf
   Next I


End Sub

Private Sub Form_Load()

   ' just in case the default port won't open
   On Local Error Resume Next

   ' disply the startup message in the terminal window
   txtTerminal.Text = "Reveived data will be displayed here." & vbCrLf & _
                "Keys will be transmitted as you press them." & vbCrLf & _
                "If you are connected to a modem it should echo" & vbCrLf & _
                "each key press." & vbCrLf & vbCrLf & _
```

```
                    "To change the comm settings use the OPTIONS|SETTINGS menu." &
vbCrLf

    ' move the cursor to the end of text
    txtTerminal.SelStart = Len(txtTerminal)

    ' set the startup color for the Rx & Tx LED's
    Set imgRx.Picture = imgLed(GreenOff).Picture
    Set imgTx.Picture = imgLed(GreenOff).Picture

    Me.Show
    Me.Refresh

    ' setup the default comm port settings
    MSComm1.CommPort = 1                ' comm port 1
    MSComm1.RThreshold = 1              ' use 'on comm' event processing
    MSComm1.Settings = "9600,n,8,1"     ' baud, parity, data bits, stop bits
    MSComm1.SThreshold = 1              ' allows us to track Tx LED
    MSComm1.InputMode = comInputModeBinary  ' binary mode, you can also use
                                        ' comInputModeText for text only use
    ' open the port
    MSComm1.PortOpen = True

    ' display status
    ShowInfo
    mnuMode_Click (0)
    mnuMode_Click (3)

End Sub

Private Sub Form_Resize()

    ' resize the display controls to match the form
    picInfo.Move 0, Abs(Me.Height - 1135), Abs(Width - 120), 315
    'txtTerminal.Move 0, 0, Abs(Width - 120), picInfo.Top

End Sub

Private Sub imgDTR_Click()

    ' DTR & RTS are output lines on the comm control
    ' clicking the DTR LED will toggle the DTR line
    MSComm1.DTREnable = MSComm1.DTREnable Xor &HFFFF
    SetLEDs

End Sub
```

```
Private Sub imgRTS_Click()

    ' DTR & RTS are output lines on the comm control
    ' clicking the RTS LED will toggle the RTS line
    MSComm1.RTSEnable = MSComm1.RTSEnable Xor &HFFFF
    SetLEDs

End Sub

Private Sub mnuClear_Click()

    ' clear the terminal window
    txtTerminal.Text = ""

End Sub

Private Sub mnuLocalEcho_Click()

    ' toggle local echo, if true, keypress's
    ' will be written to the terminal window
    mnuLocalEcho.Checked = mnuLocalEcho.Checked Xor &HFFFF
    If mnuLocalEcho.Checked Then
        bLocalEcho = True
    Else
        bLocalEcho = False
    End If

End Sub

Private Sub mnuLoopBack_Click()

    ' toggle on/off a timer that will send characters out the comm port for
    ' a simple loop back test. Connect pins 2 & 3 together to see the data

    mnuLoopBack.Checked = mnuLoopBack.Checked Xor &HFFFF
    If mnuLoopBack.Checked Then
        tmrLoopBack.Enabled = True
    Else
        tmrLoopBack.Enabled = False
    End If

End Sub

Private Sub mnuMode_Click(Index As Integer)
```

```
   ' switch character receive modes, OnComm Event and Polled mode
   ' switch character buffer modes, character, and message

   Select Case Index

      Case 0
         mnuMode(1).Checked = False
         tmrPolledMode.Enabled = False
         MSComm1.RThreshold = 1
         lblMode = "Event"
         mnuMode(3).Enabled = True
         mnuMode(4).Enabled = True
         If bMessageMode Then mnuMode(4).Checked = True Else
mnuMode(3).Checked = True

      Case 1
         mnuMode(0).Checked = False
         MSComm1.RThreshold = 0
         tmrPolledMode.Enabled = True
         lblMode = "Polled"
         mnuMode(4).Checked = False
         mnuMode(3).Checked = False
         mnuMode(3).Enabled = False
         mnuMode(4).Enabled = False

      Case 3
         mnuMode(4).Checked = False
         bMessageMode = False

      Case 4
         mnuMode(3).Checked = False
         bMessageMode = True

   End Select

   mnuMode(Index).Checked = True

End Sub

Private Sub mnuSettings_Click()

   Dim bLoaded     As Boolean
   Dim frm         As Form

   ' open the comm settings form
   frmSettings.CommSettings Me.MSComm1, "Communications Port Settings"
   ' wait for the settings form to unload
   ' modal is not used so multi port apps can
   ' continue while the settings form is visible.
   ' this is only required because we want to
   ' capture the new settings & display them.

   Do
      bLoaded = False
      For Each frm In Forms
         If frm.Name = "frmSettings" Then bLoaded = True
      Next
     DoEvents
   Loop While bLoaded

   ' display the new settings
   ShowInfo

End Sub

Private Sub MSComm1_OnComm()

'***********************************************************************
*******
' Synopsis:     Handle incoming characters, 'On Comm' Event
'
' Description:  By setting MSComm1.RThreshold = 1, this event will fire for
'               each character that arrives in the comm controls input buffer.
'               Set MSComm1.RThreshold = 0 if you want to poll the control
'               yourself, either via a TImer or within program execution loop.
'
'               In most cases, OnComm Event processing shown here is the prefered
'               method of processing incoming characters.
'
'***********************************************************************
*******


   Static sBuff    As String           ' buffer for holding incoming characters
   Const MTC       As String = vbCrLf  ' message terminator characters (ususally vbCrLf)
   Const LenMTC    As Long = 2         ' number of terminator characters, must match
MTC
   Dim iPtr        As Long             ' pointer to terminatior character

   ' OnComm fires for multiple Events
   ' so get the Event ID & process
```

```
Select Case MSComm1.CommEvent

  ' Received RThreshold # of chars, in our case 1.
  Case comEvReceive

    ' read all of the characters from the input buffer
    ' StrConv() is required when using MSComm in binary mode,
    ' if you set MSComm1.InputMode = comInputModeText, it's not required

    sBuff = sBuff & StrConv(MSComm1.Input, vbUnicode)

    ' a typical application would buffer characters here waiting for
    ' an end of message sequence like vbCrLf, that's why sBuff is declared
    ' as Static and the statement above sets sBuff = sBuff & MSComm1.Input
    ' When an end of message string is received the messages are passed
    ' through a parser routine. Here, we show processing a character at
    ' time and 'message parsing' options. MEssage parsing varies depending
    ' on what you're doing but would look something like this:

    If bMessageMode Then
      ' in message mode we wait for the message terminator
      ' before processing. This is typcal of a command & control
      ' program that interfaces with an external device and
      ' must decode data coming from the device. Most devices will
      ' use a start / end sequennce to ID each message.  You
      ' would process the messages by calling your message parser and
      ' passing the message just like the message is passed to the
      ' PosTerminal routine below. Some device's use character count
      ' to ID messages instead of start/end characters, this method is
      ' too machine specific to be shown here.

      ' look for message terminator
      iPtr = InStr(sBuff, MTC)
      ' process all queued messages
      Do While iPtr
        ' pass each message to the message parser
        ' in our case, it just gets displayed. To decode
        ' specific messages, you would pass the string
        ' Mid$(sBuff, 1, iPtr + LenMTC - 1)
        ' to a message decoder routime
        PostTerminal Mid$(sBuff, 1, iPtr + LenMTC - 1)
        ' remove from the message queue
        sBuff = Mid$(sBuff, iPtr + LenMTC)
        ' look for another message
        iPtr = InStr(sBuff, MTC)
      Loop
    Else
       ' in character mode we just pass each character to
       ' the parser as it comes in. The parser is responsibe
       ' for collecting the characters and assembling any messages.
       ' For our simple terminal example, character mode works fine.
       PostTerminal sBuff
       sBuff = vbNullString
    End If


    ' flash the Rx LED
    Set imgRx.Picture = imgLed(GreenOn).Picture
    tmrRxLED.Enabled = True


' Change in the CD line.
Case comEvCD
    SetLEDs

' Change in the CTS line.
Case comEvCTS
    SetLEDs

' Change in the DSR line.
Case comEvDSR
     SetLEDs

' Change in the Ring Indicator.
Case comEvRing

' An EOF charater was found in the input stream
Case comEvEOF

' There are SThreshold number of characters in the transmit  buffer.
Case comEvSend
    Set imgTx.Picture = imgLed(GreenOn).Picture
    tmrTxLED.Enabled = True

' A Break was received.
Case comEventBreak
    lblError = "Break"
    tmrClearError.Enabled = True

' Framing Error
Case comEventFrame
```

```
         lblError = "Framing"
         tmrClearError.Enabled = True

      ' Data Lost.
      Case comEventOverrun
         lblError = "Overrun"
         tmrClearError.Enabled = True

      ' Receive buffer overflow.
      Case comEventRxOver
         lblError = "Overflow"
         tmrClearError.Enabled = True

      ' Parity Error.
      Case comEventRxParity
         lblError = "Parity"
         tmrClearError.Enabled = True

      ' Transmit buffer full.
      Case comEventTxFull
         lblError = "Tx Full"
         tmrClearError.Enabled = True

      ' Unexpected error retrieving DCB]
      Case comEventDCB
         lblError = "DCB Error"
         tmrClearError.Enabled = True

   End Select


End Sub

Public Sub PostTerminal(ByVal sNewData As String)

   ' display incoming characters in the
   ' textbox 'terminal' window. API is
   ' used only to reduce flicker.

   Dim lPtr    As Long

   ' this is faster and has less flicker but requires use of the Win API
   With txtTerminal
      lPtr = SendMessage(.hwnd, EM_GETLINECOUNT, 0, ByVal 0&)
      If lPtr > 550 Then
         'LockWindowUpdate .hWnd
```

```
            Call SendMessage(.hwnd, WM_SETREDRAW, False, ByVal 0&)
            lPtr = SendMessage(.hwnd, EM_LINEINDEX, 100, ByVal 0&)
            .SelStart = 0
            .SelLength = IIf(lPtr > 0, lPtr, 1000)
            .SelText = vbNullString
            Call SendMessage(.hwnd, WM_SETREDRAW, True, ByVal 0&)
            ' LockWindowUpdate 0
        End If
        .SelStart = SendMessage(.hwnd, WM_GETTEXTLENGTH, True, ByVal 0&)
        .SelText = sNewData
        .SelStart = SendMessage(.hwnd, WM_GETTEXTLENGTH, True, ByVal 0&)
    End With

End Sub

Private Sub tmrClearError_Timer()

    lblError = ""
    tmrClearError.Enabled = False

End Sub

Private Sub tmrPolledMode_Timer()

    ' example of polled mode. This is an alternative to using
    ' the MSComm1 OnComm Event for receiving characters.
    ' collect characters here when the Timer fires. See
    ' the comments in MSComm1 OnComm Event for information
    ' on more complex processing. message mode has not been
    ' implimented here, see MSComm1 OnComm Event for example
    ' of message mode operation.

    If MSComm1.InBufferCount Then
        PostTerminal StrConv(MSComm1.Input, vbUnicode)
        Set imgRx.Picture = imgLed(GreenOn).Picture
        tmrRxLED.Enabled = True
    End If

End Sub

Private Sub tmrRxLED_Timer()

    Set imgRx.Picture = imgLed(GreenOff).Picture
    tmrRxLED.Enabled = False

End Sub
```

```
Private Sub tmrTxLED_Timer()

   Set imgTx.Picture = imgLed(GreenOff).Picture
   tmrTxLED.Enabled = False

End Sub

Private Sub tmrLoopBack_Timer()

   ' use this timer to send some characters out so we can test our receive code...
   ' this is only here for the loop back demo, it is not required for communications

   If MSComm1.PortOpen Then
      MSComm1.Output = Me.Caption & Format$(Timer, " ###,##0.000") & vbCrLf
   End If

End Sub

Private Sub txtTerminal_KeyPress(KeyAscii As Integer)

   ' send keys out the comm port, convert vbCr to vbCrLf
   Select Case KeyAscii
      Case 13
         If MSComm1.PortOpen Then MSComm1.Output = vbCrLf
      Case Else
         If MSComm1.PortOpen Then MSComm1.Output = Chr$(KeyAscii)
   End Select

   If Not bLocalEcho Then KeyAscii = 0

End Sub

Private Sub ShowInfo()

   ' display status info
   lblSettings = UCase$(MSComm1.Settings)
   lblPort = "Port " & MSComm1.CommPort
   lblOpen = IIf(MSComm1.PortOpen, "Open", "Closed")
   lblError = ""

   SetLEDs

End Sub

Private Sub SetLEDs()
```

```
    ' set the status LED's
    Set imgCD.Picture = IIf(MSComm1.CDHolding, imgLed(RedOn).Picture,
imgLed(RedOff).Picture)
    Set imgCTS.Picture = IIf(MSComm1.CTSHolding, imgLed(RedOn).Picture,
imgLed(RedOff).Picture)
    Set imgDSR.Picture = IIf(MSComm1.DSRHolding, imgLed(RedOn).Picture,
imgLed(RedOff).Picture)
    Set imgRTS.Picture = IIf(MSComm1.RTSEnable, imgLed(RedOn).Picture,
imgLed(RedOff).Picture)
    Set imgDTR.Picture = IIf(MSComm1.DTREnable, imgLed(RedOn).Picture,
imgLed(RedOff).Picture)

End Sub
```