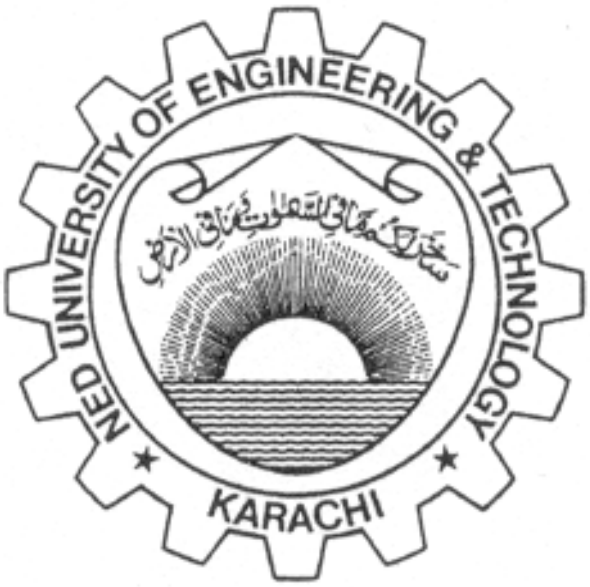
AUTONOMOUS ROVERBOT
USING SCENE ANALYSIS

B.E. (CIS) PROJECT REPORT

by

Syed Raza Abbas



Department of Computer and Information Systems Engineering
N.E.D. University of Engineering and Technology,
Karachi-75270

AUTONOMOUS ROVERBOT
USING SCENE ANALYSIS

B.E. (CIS) PROJECT REPORT

Project Group

Muhammad Usman Ghani	**CIS-022**
Faisal Nasim	**CIS-037**
Syed Raza Abbas	**CIS-105**

BATCH: 2002-2003

Project Adviser

Fahad Abdel Kader (Internal Adviser)

December 2006

Department of Computer and Information Systems Engineering

N.E.D. University of Engg. & Technology

Karachi-75270

ABSTRACT

Autonomous Roverbot using Scene Analysis covers all the major aspects of Computer Engineering from Software to Hardware and from Signaling to Control. The idea behind the project is to develop an autonomous vehicle which will be controlled through a remote station. The vehicle is fitted with a wireless video camera which transmits to a base-station where it processed through Matlab. The base-station, then submits controlling signals to the vehicle to navigate through its course. Such a robot could be used for surveillance, scanning pipes (through manual or limited autonomous control) and tracking moving objects.

ACKNOWLEDGEMENTS

We'd like to make special acknowledgement of two people who helped us in project selection and provided guidance through various parts of the project:

Our internal adviser **Mr. Fahad Abdel Kader** and **Prof. Dr. Uvais Qidwai**.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Scope of the Project

Autonomous intelligent vehicles are the one of the most widely researched areas in computer science and engineering. These vehicles capable of operating by themselves or at least with minimal human help, find applications in a lot of areas where sending humans is too dangerous or where human cognition skills are utilized only minimally. This project is an attempt to explore this active field by designing and implementing a small, simple bot that guides itself using visual information from its environment.

1.2 Application Areas

UAV – Modern powerful Unmanned Aerial Vehicles are being built for surveying of a landscape.

Precision Target Tracking – To track and follow a moving object through area or a terrain.

Self-navigating Vehicle – A vehicle that would avoid any obstacle.

Space Missions – Unmanned Rovers have been sent to Mars for planetary survey.

CHAPTER 2

AUTONOMOUS ROVERBOT USING SCENE ANALYSIS

2.1 Application Areas

Application areas for the roverbot include:

- Space exploration.
- Doemstic usage.
- Helping disabled people.
- Performing repetitive tasks in industry.
- Performing dangerous tasks.

2.2 Project Design Strategy

2.2.1 Main Objective

To design a mobile robotic vehicle controlled wirelessly through a base-station. The lightweight vehicle will transmit images back to the base-station and receive directions. The directions can be made specific to a task that it is currently performing.

2.2.2 Main Project Modules

- Matlab/OpenCV for Image Processing
- Hardware Design: Vehicle and and Wireless Solution
- Software Design

2.2.3 Matlab/OpenCV

Initially we planned to work with Matlab and later maybe port it over to OpenCV if Matlab performance is not satisfactory.

2.2.4 Hardware

The hardware consists of a regular toy car fitted with a remote camera and the remote control interfaced with computer. An intermediate software component links the software to the actual hardware.

2.2.5 Software Design

This segment actually depends upon what kind of approach we use. The general idea of the software design part is to design a communication interface between the robot and the base-station.

In design approach # 1:

- We don't need to send every frame of the stream. Limited patterns can be dealt directly on the mobile robot unit.
- Each time a new link is established a new set of intelligence should be downloaded on the robot.
- Image or video stream can be requested by the base-station for analysis or live tracking
- The synchronization could be done through compressed XML data (for Cow approach).

In design approach # 2 approach:

- There is very limited (and complex) software design on the Robochuriyan end which would take the data from the video devices and sensors, compress them and send them over to the wifi base-station.

2.3 Research Analysis

2.3.1 Tool Evaluation: RobotFlow

- http://robotflow.sourceforge.net/

This is a complete framework for robotics and includes video processing blocksets and AI, Fuzzy Logic frameworks. The major limitation is that it has limited vision algorithms which includes color tracking and training. A major set of algorithms will have to designed from scratch and will have to plugged in somehow into that framework. This work could very well encompass several FYPs in itself. Another option is to be able to connect this framework to Matlab and do the image processing there but it beats the purpose because this was being considered to replace the requirement of Matlab.

2.3.2 Bot Evaluation: Lego Mindstorm with IR

Interfacing and sending live commands to Robot through IR

- http://www.palmtoppaper.com/PTPHTML/43/43c00010.htm
- http://www.palmtoppaper.com/PTPHTML/43/43c00011.htm

Drawbacks: IR is not well suited for our application however it serves well to demo the system features and since Lego Mindstorm consists of discrete components which have

much better reliability compared to self-made circuits, hence this option is under consideration.

2.3.3 More Bot Evaluations

Showcase items

- New 2CH RTF SYMA DragonFly Radio Remote Control Helicopter, $36.95, ships next day: http://www.elitehobby.com/rc5-1012.html
- Desktop Rover: http://lego-mindstorms-roboticskits.stores.yahoo.net/desktoprover1.html
- Brink Rover: http://www.brink.com/brink_rover.php
- http://www.hobbytron.com/

Misc items

- **RC Cars**
  http://www.elitehobby.com/rc-toys-rc-cars.html
- **SYMA Dragonfly**
  http://www.egrandbuy.com/newsydrrarec.html
- **Digital Programmable Remote Copycat**
  http://www.hobbytron.com/G-603A.html
- **Silver fly**
  http://www.nitroplanes.com/sidr2rarecoe.html
- **LEGO Mindstorm**
  http://www.hobbytron.com/ProgrammableRobotKits.html

2.4 Design Approaches

Two design approaches have been devised by the team depending on how much functionality to incorporate into our roverbot. The two approaches are analogous to a Thin Client/Fat Client classification.

2.4.1 Design Approach # 1

To put an entire computer inside the remote vehicle, for example a laptop or a pentium motherboard which would be running a complete operating system. This approach will be necessary in order to start in any case as preliminary testing will be done over Matlab and a visual environment for rapid development.

Pros

- Can use wireless lan to transfer data instead of a customized solution.
- The robot can employ its own guidance mechanism if the base-station link terminates.

Cons

- Bulky vehicle.

Possible Solutions

- Use a laptop.
- Use a micro-ATX board with a frame-grabber card or USB.
- Run Linux/Windows.

2.4.2 Design Approach # 2

The vehicle will be fitted with only sensors and cameras. All the processing is done over the base-station. This kind of vehicle will have its own applications.

Pros

- Small vehicle.
- The product can showcase as: "Just install this card over your robot and bring it to life!". The card being our customized solution for sending video to the base-station.

Cons

- If the base-station connection breaks, the robot will not know what to do.
- If live video is to be transmitted, we'd need to find a suitable microcontroller and make an interface from the video camera and send the video over wire/air.

Possible Solutions

- Use some kind of PIC board with embedded video interface and optionally a Bluetooth/Wifi interface as well.
- Use a home wireless spy-camera solution (comes with receiver and free software).

2.5 Modified Ideas

2.5.1 Finalized Hardware

- Use a miniature wireless spy camera which will operate on RF.

- Process atmost 30 fps
- Use a ready-made toy car with modified remote control interfaced to PC.
- No video processing on the robot itself.

2.5.2 Finalized Software

- Matlab 2006a using the Image Acquisition and Image Processing toolboxes.
- Simulink Video and Image Processing Toolbox, for initial design and fast code generation for the final product.

2.5.3 Robot Characteristics

- Robot may be given two locations, source A and destination B and robot will move to that location by avoiding obstructions (Simulates Delivery).
- Robot may be locked onto a particular target and the robot will follow that target by avoiding obstacles (Simulates missile behaviour).
- The robot will navigate around itself freely by avoiding obstructions (Simulates surveillance).

CHAPTER 3

COMPUTER VISION

Computer vision is the study and application of methods which allow computers to "understand" image content or content of multidimensional data in general. The term "understand" means here that specific information is being extracted from the image data for a specific purpose: either for presenting it to a human operator (e. g., if cancerous cells have been detected in a microscopy image), or for controlling some process (e. g., an industry robot or an autonomous vehicle). The image data that is fed into a computer vision system is often a digital gray-scale or colour image, but can also be in the form of two or more such images (e. g., from a stereo camera pair), a video sequence, or a 3D volume (e. g., from a tomography device). In most practical computer vision applications, the computers are pre-programmed to solve a particular task, but methods based on learning are now becoming increasingly common.

Computer vision can also be described as the complement (but not necessary the opposite) of biological vision. In biological vision and visual perception real vision systems of humans and various animals are studied, resulting in models of how these systems are implemented in terms of neural processing at various levels. Computer vision, on the other hand, studies and describes technical vision system which are implemented in software or hardware, in computers or in digital signal processors. There is some interdisciplinary work between biological and computer vision but, in general, the field of computer vision studies processing of visual data as a purely technical problem.

The main reasons that computers are widely used for vision systems are:

- they are versatile and have fully experimentally capability.
- they are precise and efficient, ambiguities are not supported, unless specific programming.
- computers are able to give precise computer process and the amount of digital memory they use for certain task.

Computer vision and other research areas:

- computer vision research give new process and task to psychology, neurology, linguistic and philosophy.
- computers can be use to reproduce process of biological vision systems in order to understand them, and they can be used to implement new theories and experimental processes in order to achieve similar vision goals or others.

3.1 Related Fields

Computer vision, **Image processing**, **Image analysis**, **Robot vision** and **Machine vision** are closely related fields. If you look inside text books which have either of these names in the title there is a significant overlap in terms of what techniques and applications they cover. This implies that the basic techniques that are used and developed in these fields are more or less identical, something which can be interpreted as there is only one field with different names.

On the other hand, it appears to be necessary for research groups, scientific journals, conferences and companies to present or market themselves as belonging specifically to one of these fields and, hence, various characterizations which distinguish each of the

fields from the others have been presented. The following characterizations appear relevant but should not be taken as universally accepted.

Image processing and **Image analysis** tend to focus on 2D images, how to transform one image to another, e.g., by pixel-wise operations such as contrast enhancement, local operations such as edge extraction or noise removal, or geometrical transformations such as rotating the image. This characterization implies that image processing/analysis does not produce nor require assumptions about what a specific image is an image of.

Computer vision tends to focus on the 3D scene projected onto one or several images, e.g., how to reconstruct structure or other information about the 3D scene from one or several images. Computer vision often relies on more or less complex assumptions about the scene depicted in an image.

Machine vision tends to focus on applications, mainly in industry, e.g., vision based autonomous robots and systems for vision based inspection or measurement. This implies that image sensor technologies and control theory often are integrated with the processing of image data to control a robot and that real-time processing is emphasized by means of efficient implementations in hardware and software.

There is also a field called **Imaging** which primarily focus on the process of producing images, but sometimes also deals with processing and analysis of images. For example, Medical imaging contains lots of work on the analysis of image data in medical applications.

Finally, **pattern recognition** is a field which uses various methods to extract information from signals in general, mainly based on statistical approaches. A significant part of this field is devoted to applying these methods to image data.

A consequence of this state of affairs is that you can be working in a lab related to one of these fields, apply methods from a second field to solve a problem in a third field and present the result at a conference related to a fourth field!

3.2 Examples of Applications of Computer Vision

Another way to describe computer vision is in terms of applications areas. One of the most prominent application fields is medical computer vision or medical image processing. This area is characterized by the extraction of information from image data for the purpose of making a medical diagnosis of a patient. Typically image data is in the form of microscopy images, X-ray images, angiography images, ultrasonic images, and tomography images. An example of information which can be extracted from such image data is detection of tumours, arteriosclerosis or other malign changes. It can also be measurements of organ dimensions, blood flow, etc. This application area also supports medical research by providing new information, e.g., about the structure of the brain, or about the quality of medical treatments.

A second application area in computer vision is in industry. Here, information is extracted for the purpose of supporting a manufacturing process. One example is quality control where details or final products are being automatically inspected in order to find defects. Another example is measurement of position and orientation of details to be picked up by a robot arm. See the article on machine vision for more details on this area. Military applications are probably one of the largest areas for computer vision, even

though only a small part of this work is open to the public. The obvious examples are detection of enemy soldiers or vehicles and guidance of missiles to a designated target. More advanced systems for missile guidance send the missile to an area rather than a specific target, and target selection is made when the missile reaches the area based on locally acquired image data.

Modern military concepts, such as "battlefield awareness" imply that various sensors, including image sensors, provide a rich set of information about a combat scene which can be used to support strategic decisions. In this case, automatic processing of the data is used to reduce complexity and to fuse information from multiple sensors to increase reliability.

One of the newer application areas is autonomous vehicles, which include submersibles, land-based vehicles (small robots with wheels, cars or trucks), and aerial vehicles. An unmanned aerial vehicle is often denoted UAV. The level of autonomy ranges from fully autonomous (unmanned) vehicles to vehicles where computer vision based systems support a driver or a pilot in various situations. Fully autonomous vehicles typically use computer vision for navigation, i.e. for knowing where it is, or for producing a map of its environment (SLAM) and for detecting obstacles. It can also be used for detecting certain task specific events, e. g., a UAV looking for forest fires. Examples of supporting system are obstacle warning systems in cars and systems for autonomous landing of aircraft. Several car manufacturers have demonstrated systems for autonomous driving of cars, but this technology has still not reached a level where it can be put on the market. There are ample examples of military autonomous vehicles ranging from advanced missiles to UAVs for recon missions or missile guidance. Space

exploration is already being made with autonomous vehicles using computer vision, e. g., NASA's Mars Exploration Rover.

Other application areas include the creation of visual effects for cinema and broadcast, e.g., camera tracking or matchmoving, and surveillance.

3.3 Theory of Motion

3.3.1 Egomotion

Ego-motion means self motion or the motion of the observer. In computer vision it refers to the effects created by the motion of the camera itself. The goal of ego-motion computation is to describe the motion of an object with respect to an external reference system, by analyzing data acquired by sensors on-board on the object i.e. the camera itself.

Examples

- Given two images of a scene, determine the 3d rigid motion of the camera between the two views.

3.3.1.1 Problems

Ego-motion leads to problems in motion segmentation based scene analysis. When a scene (image) is being segmented on the basis of moving objects, the real motion (velocity, orientation etc.) of objects might appear to be different than the factual value.
a. Wrong determination of velocity/orientation. b. Static objects might be perceived as moving objects.

3.3.1.2 Solutions

A lot of solutions have been proposed to compensate for ego-motion, most of them being targeted to particular environments or situations, like urban traffic etc. The proposed solutions fall under a few broad categories like:

a. Using the knowledge of the environment to separate or remove background features. b. Using knowledge about the motion of the observer (camera) itself, like velocity of a car, if the camera is mounted on a car. c. Probabilistic methods to estimate background features. d. Techniques from stereoscopic vision to perceive depth, and separate background. e. Some combination of the aforementioned techniques.

3.3.1.3 Scope of ego-motion in our project

Ego-motion poses the greatest problem in roverbot navigation as it is almost impossible to avoid obstacles and track targets without identifying targets from obstacles. The problem emerges from the movement of the camera mounted on the roverbot. Unless the problem is solved, it will create major difficulties in the implementation of autonomous navigation.

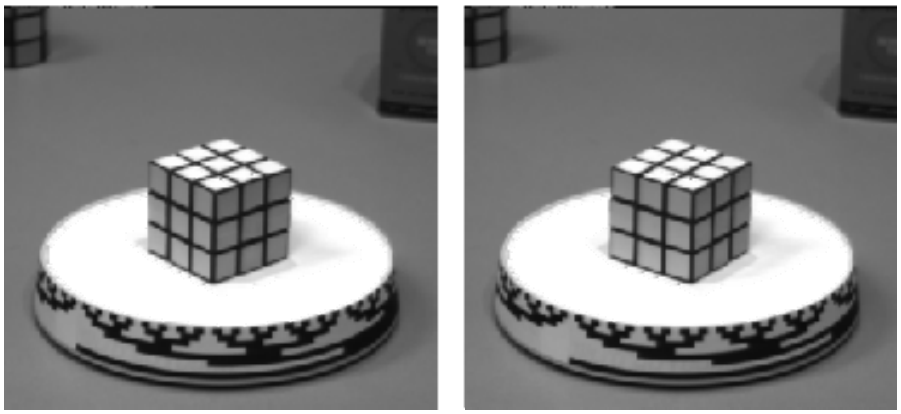
3.4 Optical Flow

Optical flow is a concept for estimating the motion of objects within a visual representation. Typically the motion is represented as vectors originating or terminating at pixels in a digital image sequence.

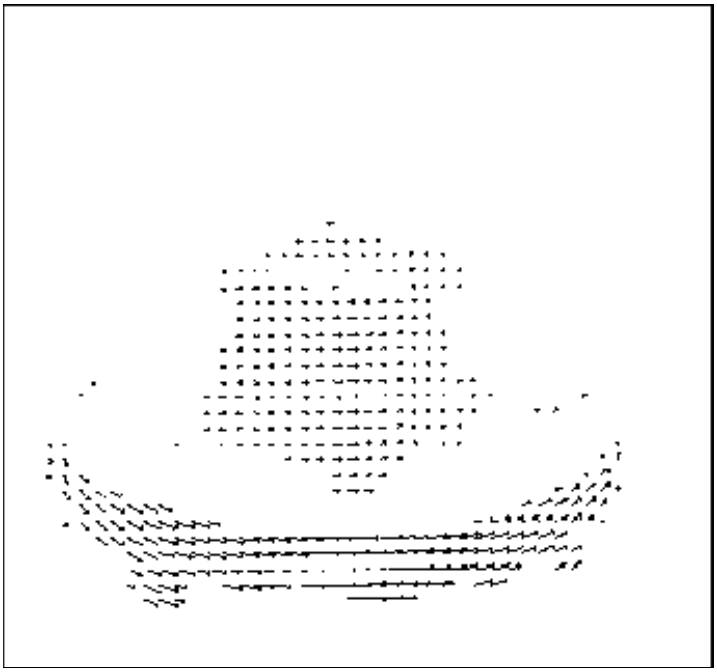
Optical flow is useful in pattern recognition, computer vision, and other image processing applications. It is closely related to motion estimation and motion compensation. Often the term optical flow is used to describe a dense motion field with vectors at each pixel, as opposed to motion estimation or compensation which uses

vectors for blocks of pixels, as in video compression methods such as MPEG. Some consider using optical flow for collision avoidance and altitude acquisition system for micro air vehicles (MAVs).

Two frames in an image sequence:



Optical Flow Vectors calculated from above images:



3.4.1 Horn Schunck Method

As described in [5], The Horn-Schunck method of estimating optical flow is a global method which introduces a global constraint of smoothness to solve the aperture problem.

3.4.2 Lucas Kanade Method

As described in [6], Lucas Kanade method of estimation of optical flow has sufficient degree of resistance towards the aperture problem. This is the method we have decided to use in our application.

3.5 Image Processing

In the broadest sense, **image processing** is any form of information processing for which both the input and output are images, such as photographs or frames of video. Most image processing techniques involve treating the image as a two-dimensional signal and applying standard signal processing techniques to it.

3.5.1 Solution Methods

A few decades ago, image processing was done largely in the analog domain, chiefly by optical devices. These optical methods are still essential to applications such as holography because they are inherently parallel; however, due to the significant increase in computer speed, these techniques are increasingly being replaced by digital image processing methods.

Digital image processing techniques are generally more versatile, reliable, and accurate; they have the additional benefit of being easier to implement than their analog

counterparts. Specialized hardware is still used for digital image processing: computer architectures based on pipelining have been the most commercially successful. There are also many massively parallel architectures that have been developed for the purpose. Today, hardware solutions are commonly used in video processing systems. However, commercial image processing tasks are more commonly done by software running on conventional personal computers.

3.5.2 Commonly Used Signal Processing Techniques

Most of the signal processing concepts that apply to one-dimensional signals also extend to the two-dimensional image signal. Some of these one-dimensional signal processing concepts become significantly more complicated in two-dimensional processing. Image processing brings some new concepts, such as connectivity and rotational invariance, that are meaningful only for two-dimensional signals.

The fast fourier transform is often used for image processing operations because it reduces the amount of data and the necessary processing time.

One-Dimensional Techniques

- Resolution
- Dynamic range
- Bandwidth
- Filtering
- Differential operators
- Edge detection
- Domain modulation
- Noise reduction

Two-Dimensional Techniques

- Connectivity
- Rotational invariance

Applications

- Photography and printing
- Satellite image processing
- Medical image processing
- Face detection, feature detection, face identification
- Microscope image processing
- Car barrier detection

3.5.3 Feature Extraction

Feature Extraction is the process of generating a set of descriptors or characteristic attributes from an image, like edge, curves, etc.

3.5.3.1 Low-level feature extraction

Overview

Low-level feature detection refers to those basic features that can be extracted automatically from an image without any shape information. As such, thresholding is actually a form of low-level feature extraction performed as a point operation. Other examples include edge detection and curvature estimation.

First-order edge detection operators

First-order edge detection is akin to first-order differentiation. In image processing, differentiation is implemented using finite differences.

Basic operators

Basic edge detection can be implemented using differencing between adjacent pixels. Differencing horizontally adjacent pixels leads to the detection of vertical edges, the differencing operator is called a horizontal edge detector, perhaps a misnomer. Similarly horizontal edges can be detected using a vertical edge detector. The vertical and horizontal edge detectors can be combined to form a general first-order edge detector.

A sample first-order edge detector might be like:

$$\begin{bmatrix} 2 & -1 \\ -1 & 0 \end{bmatrix}$$

Original Image

Edge Image

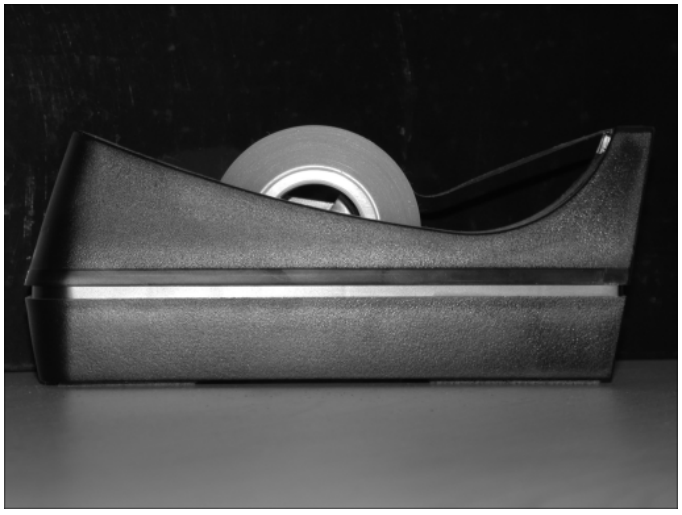
Prewitt edge detection operator

The Prewitt edge detector has the following mask:

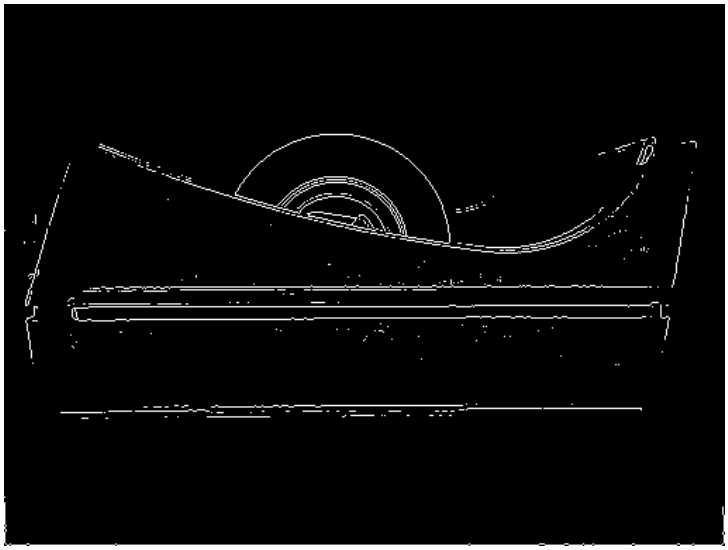
$$M_x = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

$$M_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

Grayscale Image



Edge Image



Sobel edge detection operator

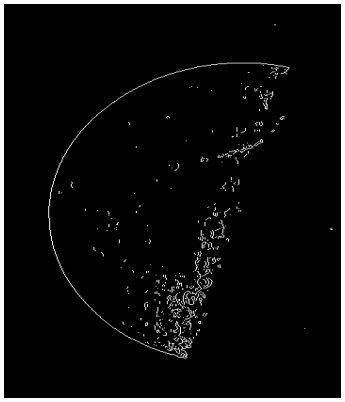
The sobel edge detector has the following mask :

$$M_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

$$M_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Original Image

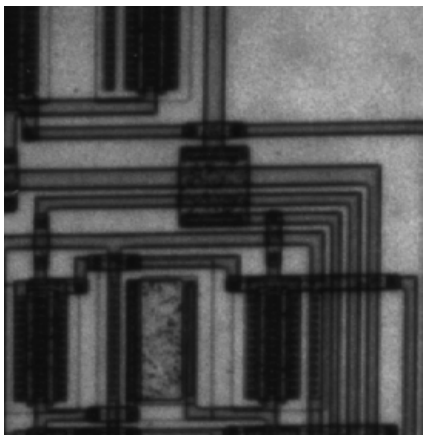
Edge Image



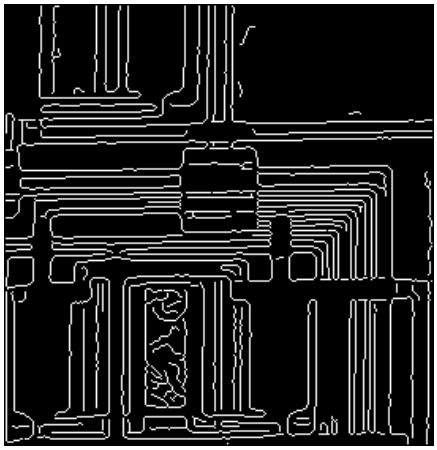
Canny edge detector

One of the most popular edge detectors of recent years, developed by Canny, uses the outputs of two Gaussian derivative masks. The two outputs are combined by squaring and adding. The peaks of ridges are then found. Ridges that contain a peak over a given threshold are retained as long as they stay above another, lower threshold.

Original Image



Edge Image



Second-order edge detection operators

Second-order edge detection is a form of second-order differentiation.

Laplacian operator

The Laplacian operator is a template which implements second-order differencing. The second-order differential can be approximated by the difference between two adjacent first-order differences :

$f''(x) = f'(x) - f'(x+1)$

which leads to :

$f''(x) = -f(x) + 2f(x+1) - f(x+2)$

This gives a horizontal second-order template :

$$\begin{bmatrix} -1 & 2 & -1 \end{bmatrix}$$

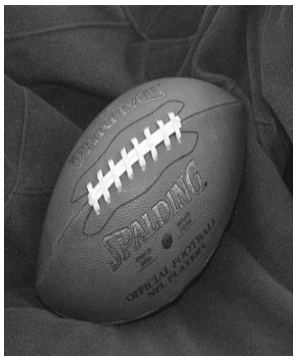
or a combined 2D template :

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

or

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Grayscale Image



Edge Image

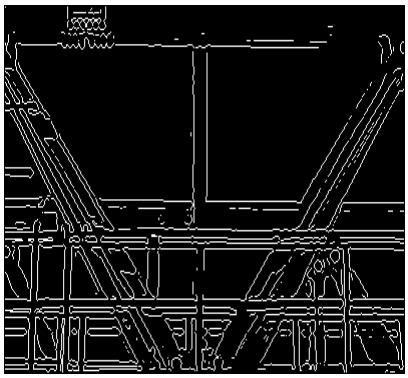
Marr-Hildreth operator

The Marr-Hildreth operator is a combination of the Gaussian smoothing filter (mask) and the Laplacian filter (mask). Combining them gives a LoG (Laplacian of Gaussian) or DoG (Difference of Gaussians) or the mexican-hat operator. A sample mask can be:

$$\begin{bmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & -2 & -1 & 0 \\ -1 & -2 & 16 & -2 & -1 \\ 0 & -1 & -2 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{bmatrix}$$

Grayscale Image

Edge Image



3.5.3.2 High-level feature extraction

Template Matching

Template matching is conceptually a simple process. We need to match a template to an image, where the template is a sub-image that contains the shape we are trying to find. Accordingly, we center the template on an image point and count up how many points in the template match those in the image. The procedure is repeated for the entire image and the point which led to the best match, the maximum count, is deemed to be the point where the shape (given by the template) lies within the image. The commonly methods used to match templates are:

- Sum of squared differences (minimization)

- Normalized sum of squared differences (minimization)
- Cross correlation (maximization)
- Normalized cross correlation (maximization)
- Correlation coefficient (maximization)
- Normalized correlation coefficient (maximization)

Hough Transform

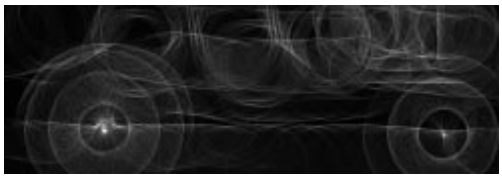
The Hough Transform is a technique that locates shapes in images. In particular, it has been used to extract lines, circles and ellipses (or conic sections). In the case of lines it mathematical definition is equivalent to the Radon transform. The HT implementation defines a mapping from the image points into an accumulator space (Hough space). The mapping is achieved in a computationally efficient manner, based on the function that describes the target shape. This mapping requires much less computational resources than template matching. However, it still requires significant storage and high computational requirements.

Though the GHT (Generalized Hough Transform) can be used to extract any shape from an image, specialised versions are used for lines, circles, ellipses and other commonly encountered shapes.

Grayscale Image

Canny edges (Edge orientation ignored)

Accumulator array - the transform itself



Most prominent circle

Flexible shape extraction (snakes)

Active contours or snakes are a completely different approach to feature extraction. An active contour is a set of points which aims to enclose a target feature, the feature to be extracted. It is a bit like using a balloon to find a shape: the balloon is placed outside the shape, enclosing it. Then by taking air out of the balloon, making it smaller, the shape is found when the balloon stops shrinking, when it fits the target shape. By this manner, active contours arrange a set of points so as to describe a target feature by enclosing it.

An initial contour is placed outside the target feature, and is then evolved so as to enclose it. Active contours are actually expressed as an *energy minimization* process. The target feature is a minimum of a suitably formulated energy functional. This energy functional includes more than just edge information: it includes properties that control the way the contours can stretch and curve. In this way, a snake represents a

compromise between its own properties (like its ability to bend and stretch) and image properties (like the edge magnitude). Accordingly, the energy functional is the addition of a function of the contour's internal energy, its constraint energy, and the image energy: these are denoted E_{int}, E_{image}, and E_{con}, respectively. These are functions of the set of points which make up a snake, v(s), which is the set of x and y coordinates of the points in the snake. The energy functional is the integral of these functions of the snake, given s E [0, 1] is the normalized length around the snake.

CHAPTER 4

INTRODUCTION TO ROBOTICS

4.1 Robot

A robot is an electro-mechanical device that can perform autonomous or preprogrammed tasks. A robot may act under the direct control of a human (eg. the robotic arm of the space shuttle) or autonomously under the control of a programmed computer. Robots may be used to perform tasks that are too dangerous or difficult for humans to implement directly (e.g. nuclear waste clean up) or may be used to automate repetitive tasks that can be performed with more precision by a robot than by the employment of a human (e.g. automobile production.)

Specifically, robot can be used to describe an intelligent mechanical device in the form of a human. This form of robot (culturally referred to as androids) is common in science fiction stories. However, such robots are yet to become common-place in reality. South Korea says it will have a robot in every home by 2015-2020.

4.2 Robotics

According to the Wikitionary, robotics is the science and technology of robots, their design, manufacture, and application. Robotics requires a working knowledge of electronics, mechanics, and software and a person working in the field has become

known as a roboticist. The word robotics was first used in print by Isaac Asimov, in his science fiction short story "Runaround" (1941).

Although the appearance and capabilities of robots vary vastly, all robots share the features of a mechanical, movable structure under some form of control. The structure of a robot is usually mostly mechanical and can be called kinematic chain (its functionality being akin to the skeleton of a body). The chain is formed of links (its bones), actuators (its muscles) and joints which can allow one or more degrees of freedom. Most contemporary robots use open serial chains in which each link connects the one before to the one after it. These robots are called serial robots and often resemble the human arm. Some robots, such as the Stewart platform, use closed parallel kinematic chains. Other structures, such as those that mimic the mechanical structure of humans, various animals and insects, are comparatively rare. However, the development and use of such structures in robots is an active area of research (e.g. biomechanics). Robots used as manipulators have an end effector mounted on the last link. This end effector can be anything from a welding device to a mechanical hand used to manipulate the environment.

The mechanical structure of a robot must be controlled to perform tasks. The control of a robot involves three distinct phases - perception, processing and action (robotic paradigms). Sensors give information about the environment or the robot itself (e.g. the position of its joints or its end effector). Using strategies from the field of control theory, this information is processed to calculate the appropriate signals to the actuators (motors) which move the mechanical structure. The control of a robot involves various aspects such as path planning, pattern recognition, obstacle avoidance, etc. More complex and adaptable control strategies can be referred to as artificial intelligence.

Any task involves the motion of the robot. The study of motion can be divided into kinematics and dynamics. Direct kinematics refers to the calculation of end effector position, orientation, velocity and acceleration when the corresponding joint values are known. Inverse kinematics refers to the opposite case in which required joint values are calculated for given end effector values, as done in path planning. Some special aspects of kinematics include handling of redundancy (different possibilities of performing the same movement), collision avoidance and singularity avoidance. Once all relevant positions, velocities and accelerations have been calculated using kinematics, methods from the field of dynamics are used to study the effect of forces upon these movements. Direct dynamics refers to the calculation of accelerations in the robot once the applied forces are known. Direct dynamics is used in computer simulations of the robot. Inverse dynamics refers to the calculation of the actuator forces necessary to create a prescribed end effector acceleration. This information can be used to improve the control algorithms of a robot.

In each area mentioned above, researchers strive to develop new concepts and strategies, improve existing ones and improve the interaction between these areas. To do this, criteria for "optimal" performance and ways to optimize design, structure and control of robots must be developed and implemented.

4.3 Robot Navigation

Systems that control the navigation of a mobile robot are based on several paradigms. Biologically motivated applications, for example, adopt the assumed behavior of animals. Geometric representations use geometrical elements like rectangles, polygons, and cylinders for the modeling of an environment. Also, systems for mobile robots exist

that do not use a representation of their environment. The behavior of the robot is determined by the sensor data actually taken. Further approaches were introduced which use icons to represent the environment.

4.3.1 Navigation

Systems that control the navigation of a mobile robot are based on several paradigms. Biologically motivated applications, for example, adopt the assumed behavior of animals. Geometric representations use geometrical elements like rectangles, polygons, and cylinders for the modeling of an environment. Also, systems for mobile robots exist that do not use a representation of their environment. The behavior of the robot is determined by the sensor data actually taken . Further approaches were introduced which use icons to represent the environment.

4.3.2 Coordinate Systems

This chapter explains the use of coordinate systems in the robotics and conversions between these systems. Movement in robotics is frequently considered as the local change of a rigid object in relation to another rigid object. Translation is the movement of all mass points of a rigid object with the same speed and direction on parallel tracks. If the mass points run along concentric tracks by revolving a rigid axis, it is a rotation. Every movement of an object can be described by declaration of the rotation and the translation. The Cartesian coordinate system is often used to measure the positions of the objects. The position of a coordinate system XC relative to a reference coordinate system XM is the origin O from XC written in coordinates from XM. For example, the origin of XM could be the base of a robot and the origin from XC could be a camera mounted on the robot. A vector of angles gives the orientation of a coordinate system

XC with respect to a coordinate system XM. By applying these angles to the coordinate system XM, it rotates so that it is commutated with the coordinate system XC. Angle aC deter-mines the rotation for the XM-axis of XM, angle bC the rotation for the YM-axis of XM, and angle cC the rotation for the ZM-axis of XM. These angles must be applied to the original coordinate system of XM. The location of a coordinate system XC comprises the position and the rotation in relation to a coordinate system XM. So the location is determined with vector lC that has six values

$$l_{\mathrm{C}} = (x_{\mathrm{M}}, y_{\mathrm{M}}, z_{\mathrm{M}}, \alpha_{\mathrm{C}}, \beta_{\mathrm{C}}, \gamma_{\mathrm{C}}).$$

The values xM, yM, and zM give the position in the reference coordinate system XM and the angles aC, bC, and cC the orientation. It is possible to write the orientation of a coordinate system XC in relation to a coordinate system XM with the aid of a 3x3 rotation matrix. Rotation matrices consist of orthogonal unit vectors. It holds that:

$$M^{-1} = M^{\mathrm{T}}.$$

The rotation matrix M can be processed from elemental 3x3 rotary matrices $X_{\mathrm{M}}(\alpha_{\mathrm{C}})$, $Y_{\mathrm{M}}(\beta_{\mathrm{C}})$, and $Z_{\mathrm{M}}(\gamma_{\mathrm{C}})$ of the three orientation angles aC, bC, and cC. The rotation with aC for the XM-axis is stated as $X_{\mathrm{M}}(\alpha_{\mathrm{C}})$, the rotation with b for the YM-axis as $Y_{\mathrm{M}}(\beta_{\mathrm{C}})$, and so forth.

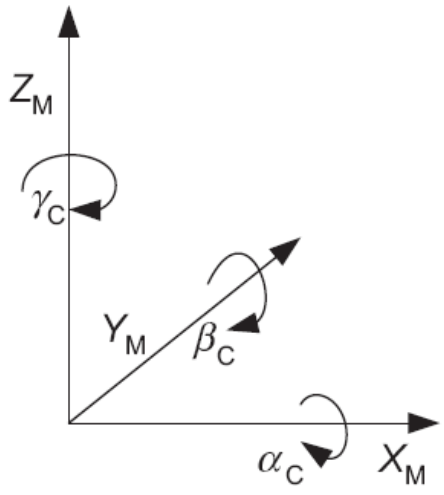


Figure 16 The six degrees of freedom

Figure 16 shows the three axes XM, YM, and ZM for the coordinate system XM. Rotation angles aC, bC, and cC are attached to the axes. The reference coordinate system XM can be moved in the direction of the three axes to obtain the coordinate system XC. It can also be rotated around the three axes. This means that six degrees of freedom are possible.

Homogeneous transformation uses a 4x4 matrix for rotation and translation. The transformation of the coordinate system XM into the coordinate system XC is written with the homogeneous matrix HM,C. Let $(x, y, z)_C$ and $(x, y, z)_M$ be the position of the same scene point in homogeneous coordinates, then the following formula holds:

$$(x, y, z)_C = H_{M,C} \cdot (x, y, z)_M$$

and always

$$H_{M,C} = (H_{C,M})^{-1}.$$

The location of a rigid object in the coordinate system XC and in the coordinate system XM can be represented with a homogeneous matrix HM,C:

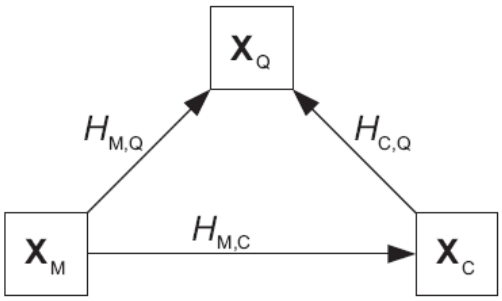


Figure 17 Conversion from locations

A further coordinate system XQ is introduced in Figure 17. If relations are given as in Figure 17, HM,Q can be processed:

$$H_{M,Q} = H_{M,C} \cdot H_{C,Q}.$$

Often several coordinate systems are necessary in robotics. For example, the view of a robot is represented with coordinate system XM. Therefore, the origin of XM is the base of the robot. If the robot is equipped with a sensor like a camera, it can be used as a second coordinate system XC, whereby the origin of XC represents the camera that is mounted on the robot, see Figure 18.

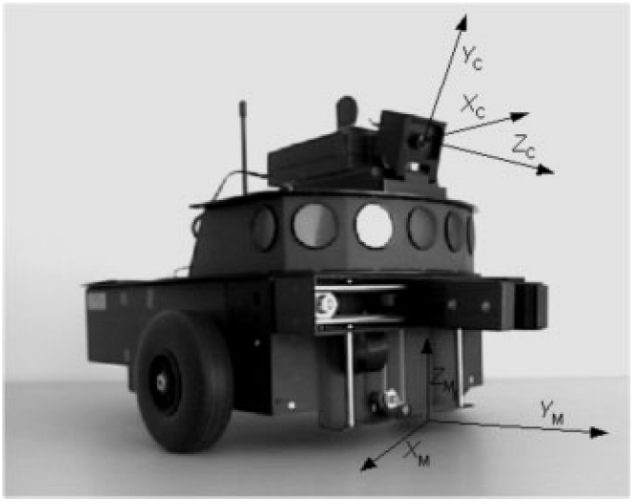


Figure 18 Coordinate systems for a mobile robot

For example, the mounted camera can be used for depth estimation. The taking of two images from different positions can perform this. It is possible to process the depth for the taken scenario with the coordinates of the camera's two different positions.

4.4 H Bridge

An **H-bridge** is an electronic circuit which enables DC electric motors to be run forwards or backwards. These circuits are often used in robotics. H-bridges are available as integrated circuits, or can be built from separate components.

The term "H-bridge" is derived from the typical graphical representation of such a circuit. An H-bridge is built with four switches (solid-state or mechanical). When the switches S1 and S4 (according to the first figure) are closed (and S2 and S3 are open) a positive voltage will be applied across the motor. By opening S1 and S4 switches and closing S2 and S3 switches, this voltage is reversed, allowing reverse operation of the motor.

Using the nomenclature above, the switches S1 and S2 should never be closed at the same time, as this would cause a short circuit on the input voltage source. The same applies to the switches S3 and S4. This condition is known as shoot-through.

A solid-state H-bridge is typically constructed using reverse polarity devices (i.e., PNP BJTs or P-channel MOSFETs connected to the high voltage bus and NPN BJTs or N-channel MOSFETs connected to the low voltage bus).

The most efficient MOSFET designs use N-channel MOSFETs on both the high side and low side because they typically have a third of the ON resistance of P-channel MOSFETs. This requires a more complex design since charge pump circuits must be

used to drive the gates of the high side MOSFETs. However, integrated circuit MOSFET drivers like the Harris Semiconductor HIP4081A make this easy.

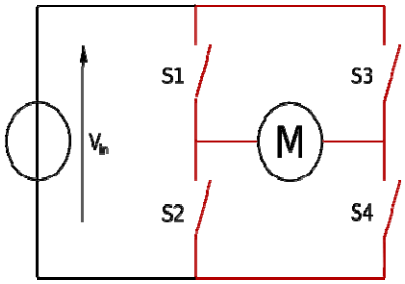


Figure: H-Bridge Circuit

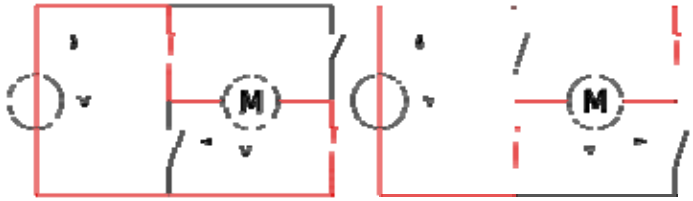


Figure: H-Bridge Circuit Operating

4.5 Servo Motors

A Servo is a small device that has an output shaft. This shaft can be positioned to specific angular positions by sending the servo a coded signal. As long as the coded signal exists on the input line, the servo will maintain the angular position of the shaft. As the coded signal changes, the angular position of the shaft changes. In practice, servos are used in radio controlled airplanes to position control surfaces like the elevators and rudders. They are also used in radio controlled cars, puppets, and of course, robots.

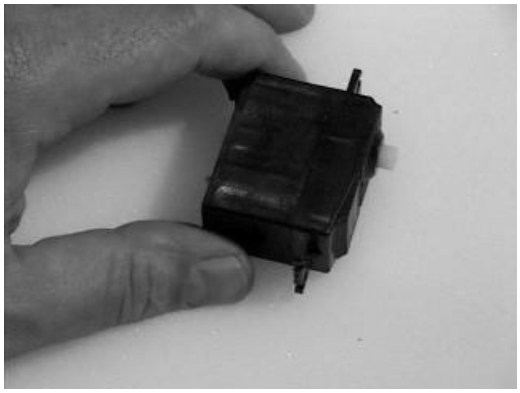


Figure: Servo Motor

Servos are extremely useful in robotics. The motors are small, as you can see by the picture above, have built in control circuitry, and are extremely powerful for thier size. A standard servo such as the Futaba S-148 has 42 oz/inches of torque, which is pretty strong for its size. It also draws power proportional to the mechanical load. A lightly loaded servo, therefore, doesn't consume much energy. A servo has 3 wires that connect to the outside world. One is for power (+5volts), ground, and the white wire is the control wire.

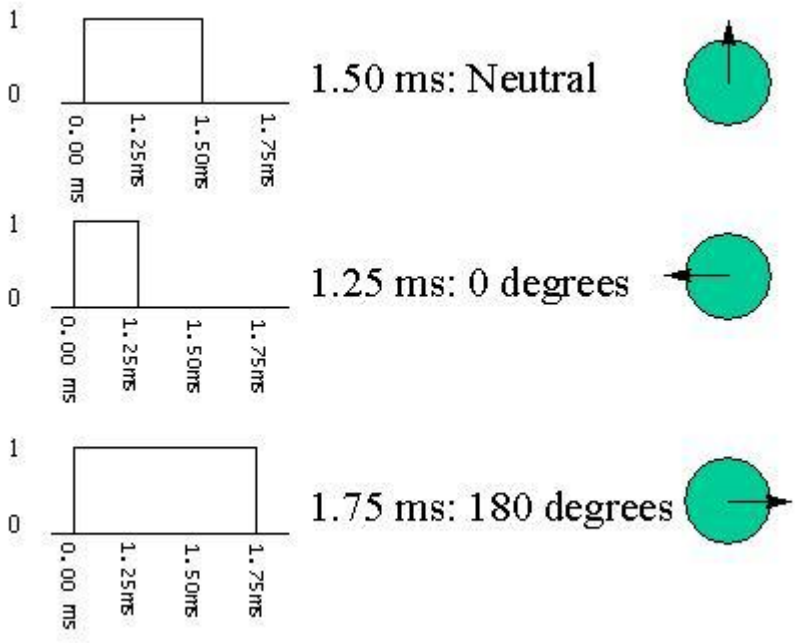
Figure: Servo Motor Components

The servo motor has some control circuits and a potentiometer (a variable resistor, aka pot) that is connected to the output shaft. In the picture above, the pot can be seen on the right side of the circuit board. This pot allows the control circuitry to monitor the current angle of the servo motor. If the shaft is at the correct angle, then the motor shuts off. If the circuit finds that the angle is not correct, it will turn the motor the correct

direction until the angle is correct. The output shaft of the servo is capable of travelling somewhere around 180 degrees. Usually, its somewhere in the 210 degree range, but it varies by manufacturer. A normal servo is used to control an angular motion of between 0 and 180 degrees. A normal servo is mechanically not capable of turning any farther due to a mechanical stop built on to the main output gear.

The amount of power applied to the motor is proportional to the distance it needs to travel. So, if the shaft needs to turn a large distance, the motor will run at full speed. If it needs to turn only a small amount, the motor will run at a slower speed. This is called proportional control.

The control wire is used to communicate the angle. The angle is determined by the duration of a pulse that is applied to the control wire. This is called Pulse Coded Modulation. The servo expects to see a pulse every 20 milliseconds (.02 seconds). The length of the pulse will determine how far the motor turns. A 1.5 millisecond pulse, for example, will make the motor turn to the 90 degree position (often called the neutral position). If the pulse is shorter than 1.5 ms, then the motor will turn the shaft to closer to 0 degress. If the pulse is longer than 1.5ms, the shaft turns closer to 180 degress.



The duration of the pulse dictates the angle of the output shaft (shown as the green circle with the arrow).

4.6 Robotics Future

Over the next few years, autonomous robots will become increasingly sophisticated and able to do more tasks other than simply entertain their owners.

Extrapolations have shown that a current PC has the computational equivalence of a low-order animal brain, and during the next decade it is likely that PC's will grow in speed to be equivalent to the brain of a higher animal such as a rat. If one considers the mobility and level of intelligence of these animals, it can be seen that there is enormous potential for converting a sophisticated entertainment device into something useful.

It is our contention that the robot designs that will succeed will be those that are adaptable to our environment – and one class of these are legged robots. The legged

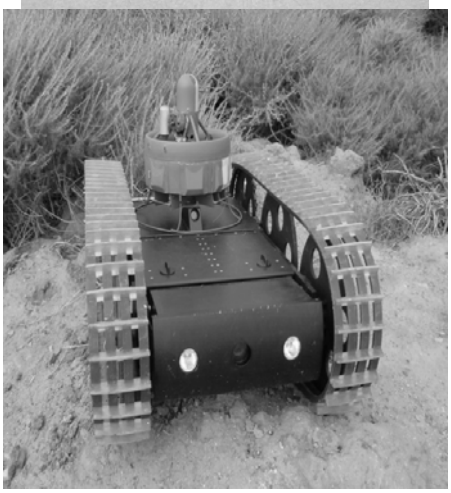
body form that is most suitable for our environment is a biped – identical in layout and size to us. Unfortunately, the biped is one of the most difficult body-forms to control, and therefore practical consumer autonomous legged robot design needs to evolve towards this goal.

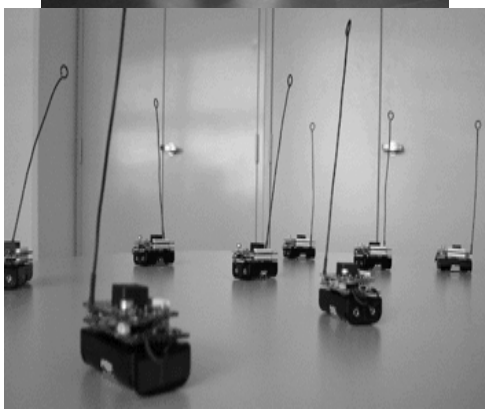
The steps needed to achieve a practical and affordable manifestation of the biped are complex and at the cutting edge of robotics – nevertheless it is our contention that these steps are tractable and achievable in the short term.

Such an autonomous biped would have the ability to interact with the physical world and integrate into society in a way that has never been seen before in a machine. The initial ability of movement and access to most areas of a domestic environment would quickly be matched by the machine's ability to move or carry objects around within the environment. Already, it is possible to imagine almost limitless uses for a device that can perform such a rudimentary task, in offices and factories as well as the home.

Beyond this point, with more sophisticated Artificial Intelligence and control, comes a vast number of tasks for which the machine can replace a person, including, in a domestic environment; cleaning, tidying, cooking, ironing, mowing, gardening, D.I.Y repair, building, child monitoring, playing, security – the list is limited only by the imagination. There are also countless commercial applications including high-risk environments, special effects and security or military applications.

4.7 Some Robots



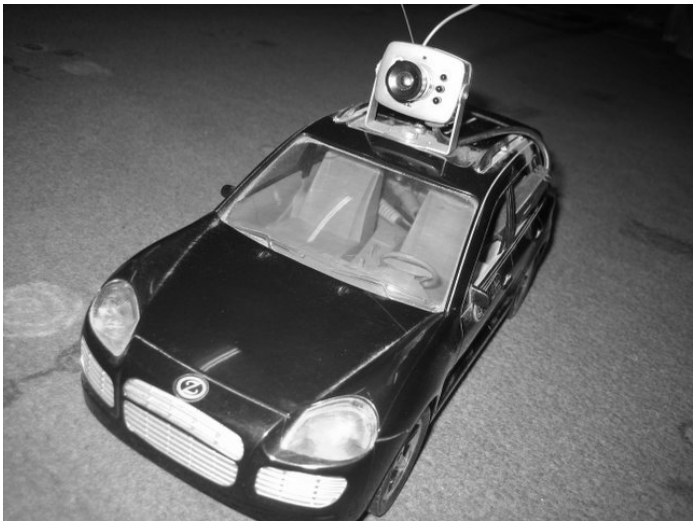


CHAPTER 5

THE BOT

5.1 General Design and Structure

The bot is a mobile unit controlled wirelessly through a computer. It consists of a remote controlled toy car with a camera mounted on top. The camera and the car's remote control both operate on RF.



5.2 Hardware Specifications

5.2.1 Camera specs

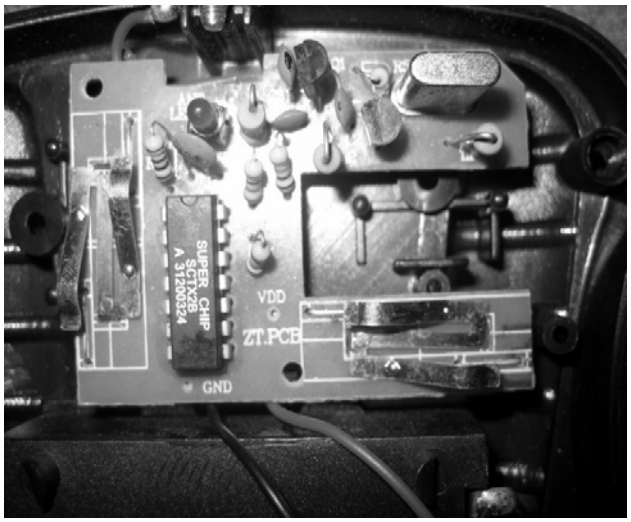
A wireless RF based camera sold by a JMK is used in the project to transmit live video.

Image pickup device 1/3 ¼ inch CMOS

TV System	PAL/CCIR NTSC/EIA
Definition	380 TV lines
Scan frequency	PAL/CCIR:50Hz NTSC/EIA:60Hz
Min illumination	3 LUX
Output Power	50mW - 200mW
Output Frequency	900MHz - 1200MHz
Power Supply	DC +6 ~ 12 V

5.2.2 Hardware modules used

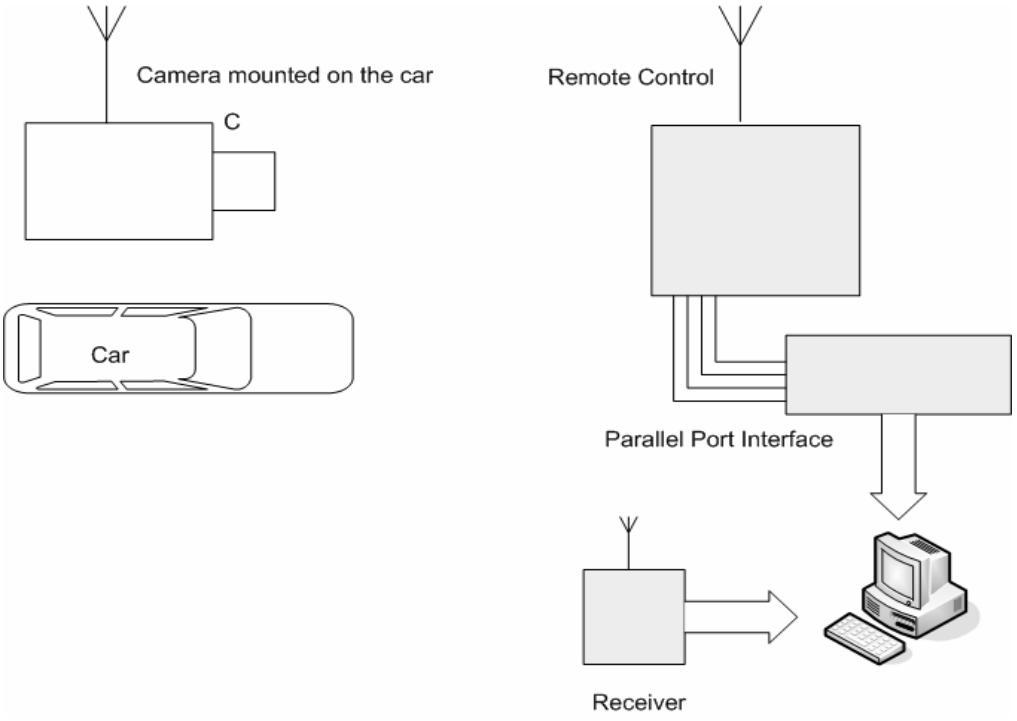
5.2.2.1 Remote Specs



The two circuits above are a part of remote control circuitry which operates over RF.

The real brain of the circuit is the SCTX2B IC. It works on active low and sending with separate pins for Forward, Reverse, Left and Right commands.

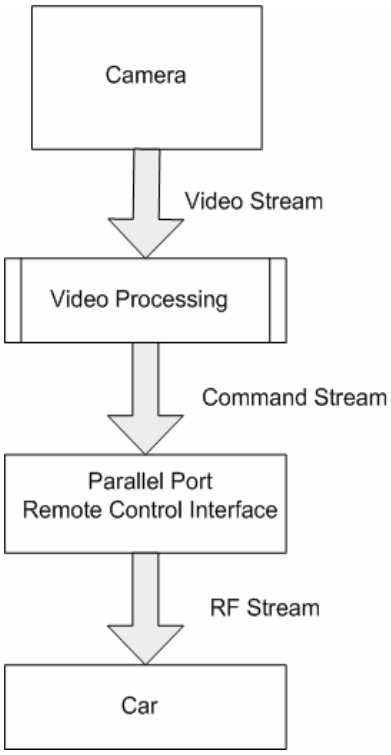
Integration



Control

- Camera sends video stream to the computer.
- Software processes the video stream through scene analysis.
- Software generates signals on the parallel port.
- Parallel port connects to an interface circuit
- Interface circuit connects to the remote control
- Remote controller communicates with the vehicle

Control flowchart for the bot



Running

- Wireless Camera transmits to the RF receiver on the terminal
- RF receiver connects to the frame grabber card
- Frame grabber card transmits data through PCI

Matlab acquires video through the **Image Acquisition Toolbox**.

CHAPTER 6

THE INTERFACE

All communication between the mobile robot and the base-station needed a simple yet efficient interfacing mechanism, one that would enable us to exercise adequate control over the robotic vehicle. Of the several choices, the **Parallel Printer Port (LPT1)** was chosen due its simplicity and adequate data-sending capability required to control our robot.

The interface was, therefore, a custom-made electronic circuit designed to communicate with both the Parallel Port and the robot's remote-control. The remote-control for the robot was modified to work in conjunction with the **interface circuit** that transmitted data between the two. A detailed explanation of the circuit follows.

6.1 The Parallel Port

The Parallel Port is the most commonly used port for interfacing home made projects. This port will allow the input of up to 9 bits or the output of 12 bits at any one given time, thus requiring minimal external circuitry to implement many simpler tasks. The port is composed of 4 control lines, 5 status lines and 8 data lines. It is found at the back of the PC as a DB-25 pin female connector. Newer parallel ports are made in accordance with the IEEE 1284 standard.

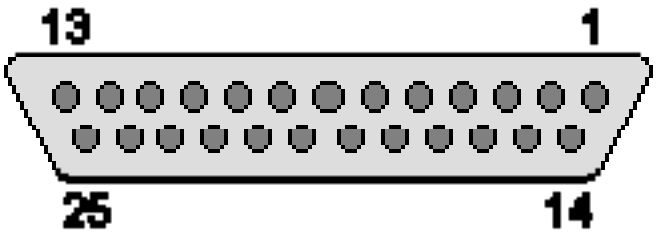


Figure: The DB-25 Female Parallel Port Connector

6.1.1 Hardware

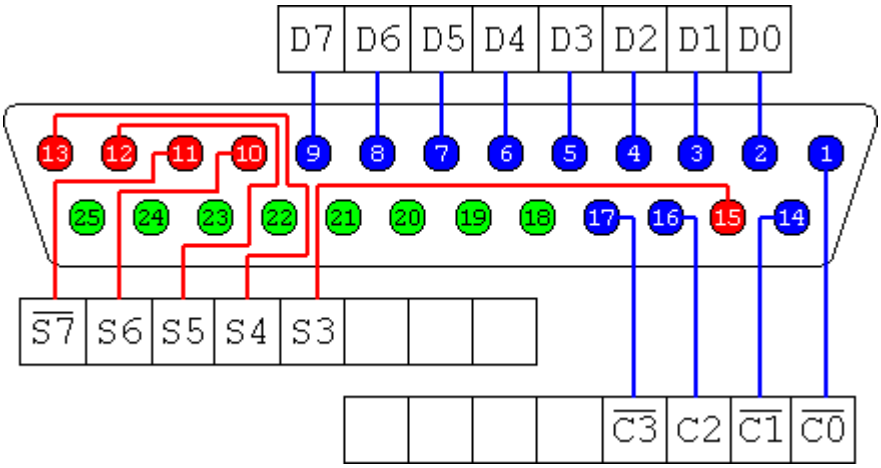


Figure: Parallel Port Pin Configuration

The Parallel Port pins can be further sub-divided into three ports namely:

Data Port or Port 0 (Pins 2-9): Used for data output.

Status Port or Port 1 (Pins 10-13 and Pin15): Used for status input.

Control Port or Port 2 (Pin 1, 14, 16, 17): Control port is a read/write port.

The Parallel Port (LPT1) address on most modern computers is **0x378**. A tabular representation of Parallel Port pins follows.

Pin	Description	Direction
1	Strobe	In/Out
2	Data 0	Output
3	Data 0	Output
4	Data 0	Output
5	Data 0	Output
6	Data 0	Output
7	Data 0	Output
8	Data 0	Output
9	Data 0	Output
10	Acknowledge	Input
11	Busy	Input
12	Paper Empty	Input
13	Select	Input
14	Autofeed	Output
15	Error	Input
16	Initialize Printer	Output
17	Select Input	Output
18-25	Ground	Gnd

6.1.2 Software

6.1.2.1 Parallel Port on Linux

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <asm/io.h>
```

```
#define base 0x378           /* printer port base address */
#define value 255            /* numeric value to send to printer port */

main(int argc, char **argv)
{
   if (ioperm(base,1,1))
    fprintf(stderr, "Couldn't get the port at %x\n", base), exit(1);

   outb(value, base);
}
```

6.1.2.2 Parallel Port on WindowsXP

Windows XP does not allow direct port access to user mode processes. It requires installation of a driver to allow port access to a process. The driver runs in kernel mode and provides a bridge which can be used by user mode processes to access the port.

Below is an example of how to do this using a driver called **PortTalk**.

```
#include <stdio.h>
#include <windows.h>
#include "pt_ioctl.c"

void __cdecl main(void)
{
    unsigned char value;
    printf("IoExample for PortTalk V2.0\nCopyright 2001 Craig
Peacock\nhttp://www.beyondlogic.org\n");
    OpenPortTalk();
    value = 0xFF;
    printf ( "Value sent: 0x%02X \n", value );
    outportb(0x378, value);
```

```
    value = inportb(0x378);
    printf("Value returned = 0x%02X \n",value);
    getch();
    puts ( "Press any key to continue..." );
    value = 0x00;
    printf ( "Value sent: 0x%02X \n", value );
    outp(0x378, 0x00);
    value = inp(0x378);
    printf("Value returned = 0x%02X \n",value);
    ClosePortTalk();
}
```

6.2 Interface Design

The interface provides the glue between the robot and the scene analysis software running on the base-station. It is used to control the robot's movements. The software uses the building blocks of this module to send appropriate motion commands to the mobile unit.

It consists of a parallel port interface coupled with the remote-control which operates the robotic vehicle.

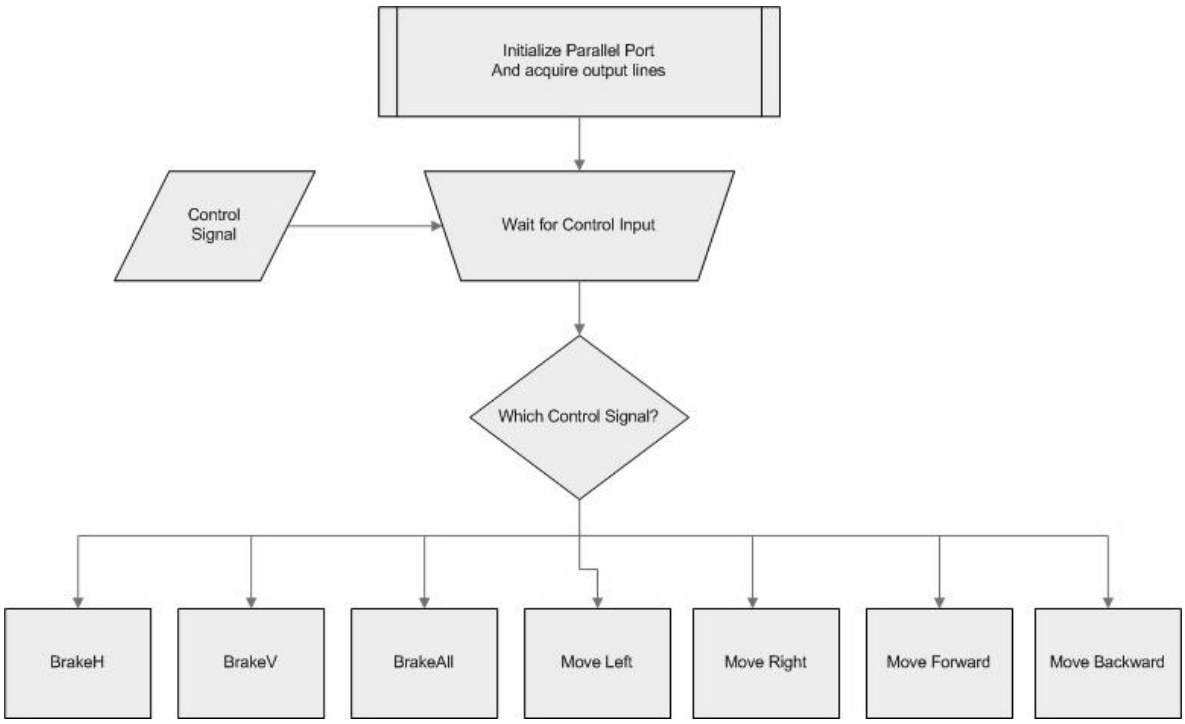


Figure: Interface Design Summary

6.2.1 Interface Hardware

In order to send commands pertaining to movement of the bot, a hardware interface has been developed between the remote-control of the vehicle and the scene analysis software using the Parallel Port (LPT1). The software reads/writes a byte of data to the Port0 of LPT1 using this interface.

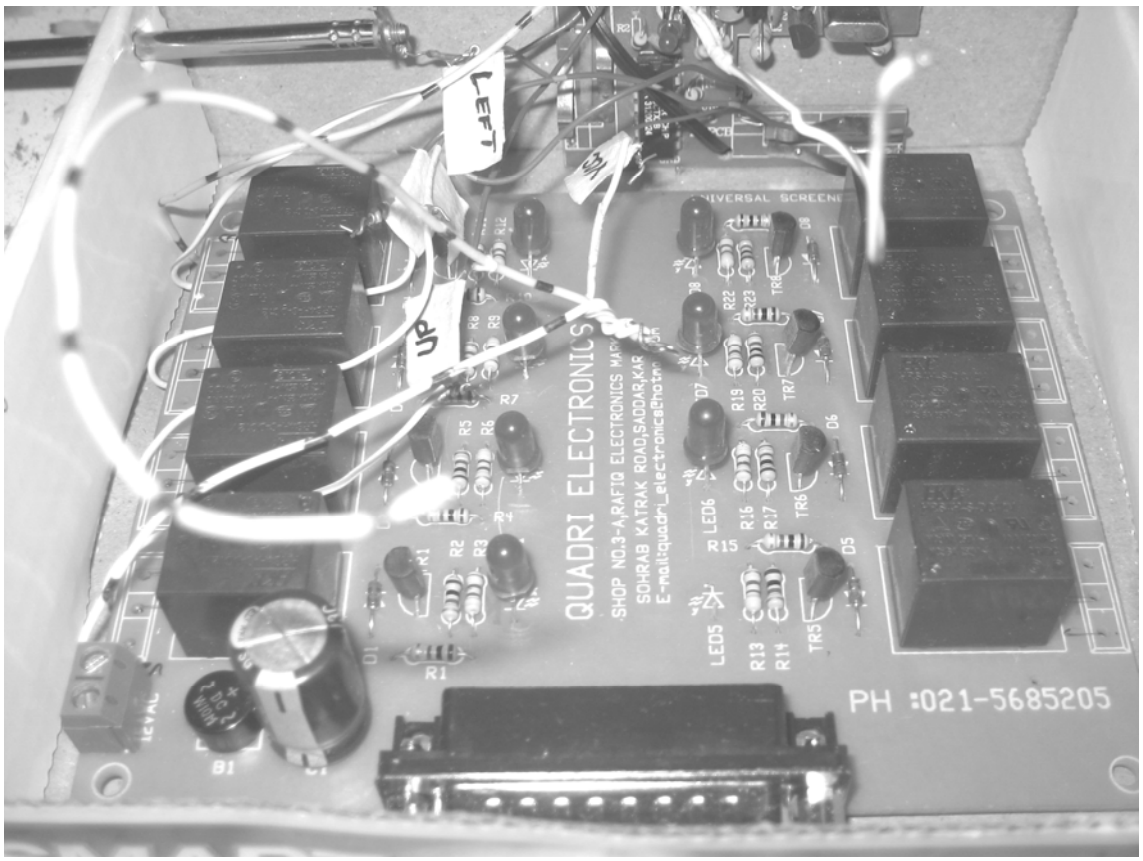


Figure: Interface Circuit Hardware

This interface consists of three stages:

- The first stage contains the female parallel port interface connector connected to a number of components that collectively act as a buffer.

- The next stage consists of transistors preceded with resistors to provide protection against reverse current which could damage the parallel port which is ensured using zener diodes.

- The third stage provides the connection between the interface circuitry and the remote control that drives the car. This connection is provided using Relays

followed by Normally-Open and Normally-Closed connections which help determine the current status of each bit of the port.

The complete schematic of the interface circuitry is provided as follows:

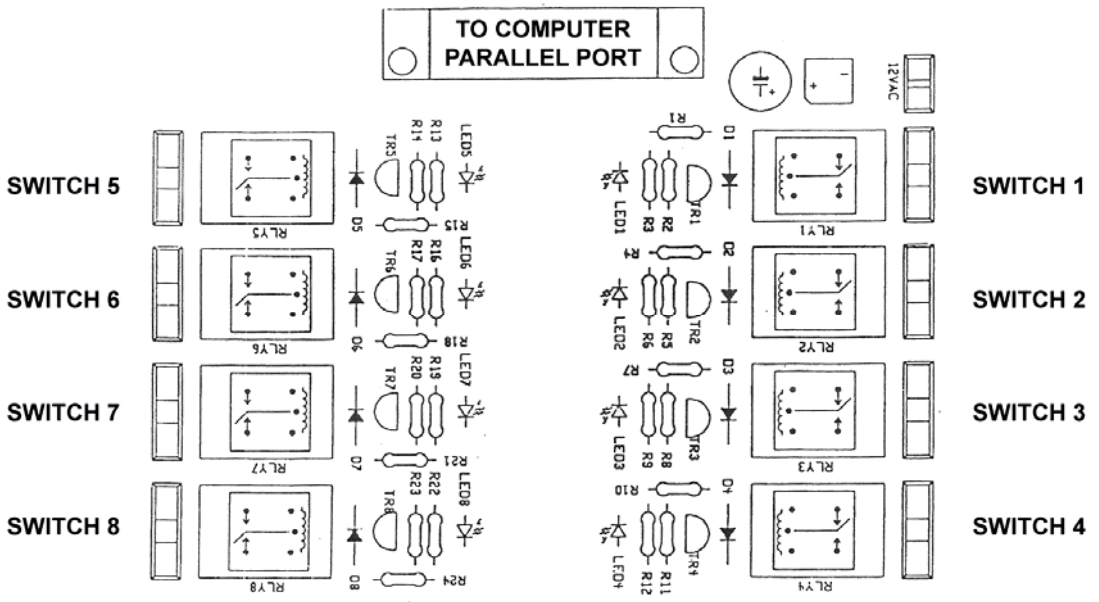


Figure: Interface Circuit Schematic

6.2.1.1 Port Bit Combinations Used

In order to convey the four commands to the bot, the last four bits of LPT1's Port0 are used. Reading from LSB towards MSB, each bit acts as a trigger for Forward, Backward, Right and Left commands with bit '1' signifying activation of motion and bit '0' signifying the de-activation.

The bit combinations used by the software are described as follows:

Command	On/Off	Bit Combination
Forward	On	X X X X X X 0 1
Forward	Off	X X X X X X 0 0
Backward	On	X X X X X X 1 0
Backward	Off	X X X X X X 0 0
Right	On	X X X X 0 1 X X
Right	Off	X X X X 0 0 X X
Left	On	X X X X 1 0 X X
Left	Off	X X X X 0 0 X X
BrakeAll	-	X X X X 0 0 0 0
BrakeH	-	X X X X X X 0 0
BrakeV	-	X X X X 0 0 X X

where **X** denotes don't care.

Note that for **Forward** motion, the bit signifying **Backward** motion is also set to zero to avoid conflicts and vice versa. The same is the case with **Left** and **Right** motion.

6.2.2 Interface Software

The software routines have been written in Matlab. An 8-bit buffer reads/writes pin status from Port0 of LPT1. This ensures that the same command is never sent twice to the port's pins by storing the last-sent command in a buffer variable.

The **Data Acquisition Toolbox** in Matlab provides access to digital I/O subsystems through a **Digital I/O** object. The Digital I/O object can be associated with a parallel port or with a Digital I/O subsystem on a data acquisition board.

The matlab routines controlling robot motion are named as:

1. MoveForward
2. MoveBackward
3. MoveLeft
4. MoveRight
5. BrakeH
6. BrakeV
7. BrakeAll

Before these routines can be used, the digital I/O object needs to be initialized and all pins on LPT1 Port 0 need to be reset. Two routines, therefore, instantiate and clear the digital I/O object from memory namely **ppo_start** and **ppo_stop**.

PPO_START Routine

```
%Declare our global variables
global ppo;
global LEFT_ON;
global LEFT_OFF;
global RIGHT_ON;
global RIGHT_OFF;
global UP_ON;
global UP_OFF;
global DOWN_ON;
global DOWN_OFF;
global Buffer;

%Set constants
%LEFT_ON = [0 0 0 0 1 0 0 0];
LEFT_ON=hex2dec('08');
%LEFT_OFF = [1 1 1 1 0 1 1 1];
LEFT_OFF=hex2dec('F7');
%RIGHT_ON = [0 0 0 0 0 1 0 0];
RIGHT_ON=hex2dec('04');
%RIGHT_OFF = [1 1 1 1  1 0 1 1];
RIGHT_OFF = hex2dec('FB');
%UP_ON = [0 0 0 0 0 0 0 1];
UP_ON = hex2dec('01');
%UP_OFF = [1 1 1 1 1 1 1 0];
UP_OFF=hex2dec('FE');
%DOWN_ON = [0 0 0 0 0 0 1 0];
DOWN_ON=hex2dec('02');
%DOWN_OFF = [1 1 1 1 1 1 0 1];
DOWN_OFF=hex2dec('FD');
Buffer=hex2dec('00');

%Instantiate our digitalio device object
```

```
ppo = digitalio('parallel','LPT1');


%Add output lines and start
ppolines = addline(ppo,0:7,0,'out');
start(ppo);


%Sending all zeros to the port at first
putvalue(ppo.Line(1:8),dec2binvec(Buffer,8));
```

PPO_STOP Routine

```
function ppo_stop
global ppo;
stop(ppo);
delete(ppo);
clear ppo
```

MoveForward Routine

```
function MoveForward(x)
%Declaring globals needed for this operation
global ppo;
global DOWN_OFF;
global UP_ON;
global UP_OFF;
global Buffer;


%First, we need to ensure the object is runnig
if(isrunning(ppo))
  %Get present port pin status
  ppoval=getvalue(ppo);


  %AND with DOWN_OFF to ensure that particular bit is 0
  Buffer = bitand(ppoval,dec2binvec(DOWN_OFF,8));
```

```
  %Depending on input param. 'x', use UP_ON or UP_OFF
  if(x==1)
  Buffer = bitor(Buffer,dec2binvec(UP_ON,8));
  else if(x==0)
  Buffer = bitand(Buffer,dec2binvec(UP_OFF,8));
      end
  end

  %Check to see whether the resulting bits are not the same
  %as the ones currently on the port's pins.
  if(binvec2dec(bitxor(Buffer,ppoval))~=0)
  putvalue(ppo,Buffer);
  end

end
```

MoveBackward Routine

```
function MoveBackward(x)
%Declaring globals needed for this operation
global ppo;
global UP_OFF;
global DOWN_ON;
global DOWN_OFF;
global Buffer;

%First, we need to ensure the object is runnig
if(isrunning(ppo))

  %Get present port pin status
  ppoval=getvalue(ppo);

  %AND with UP_OFF to ensure that particular bit is 0
```

```
  Buffer = bitand(ppoval,dec2binvec(UP_OFF,8));

  %Depending on input param. 'x', use DOWN_ON or DOWN_OFF
  if(x==1)
  Buffer = bitor(Buffer,dec2binvec(DOWN_ON,8));
  else if(x==0)
  Buffer = bitand(Buffer,dec2binvec(DOWN_OFF,8));
      end
  end

  %Check to see whether the resulting bits are not the same
  %as the ones currently on the port's pins.
  if(binvec2dec(bitxor(Buffer,ppoval))~=0)
  putvalue(ppo,Buffer);
  end

end
```

MoveLeft Routine

```
function MoveLeft(x)
%Declaring globals needed for this operation
global ppo;
global RIGHT_OFF;
global LEFT_ON;
global LEFT_OFF;
global Buffer;

%First, we need to ensure the object is runnig
if(isrunning(ppo))

  %Get present port pin status
  ppoval=getvalue(ppo);

  %AND with RIGHT_OFF to ensure that particular bit is 0
```

```
  Buffer = bitand(ppoval,dec2binvec(RIGHT_OFF,8));

  %Depending on input param. 'x', use LEFT_ON or LEFT_OFF
  if(x==1)
  Buffer = bitor(Buffer,dec2binvec(LEFT_ON,8));
  else if(x==0)
  Buffer = bitand(Buffer,dec2binvec(LEFT_OFF,8));
      end
  end

  %Check to see whether the resulting bits are not the same
  %as the ones currently on the port's pins.
  if(binvec2dec(bitxor(Buffer,ppoval))~=0)
  putvalue(ppo,Buffer);
  end

end
```

MoveRight Routine

```
function MoveRight(x)
%Declaring globals needed for this operation
global ppo;
global LEFT_OFF;
global RIGHT_ON;
global RIGHT_OFF;
global Buffer;

%First, we need to ensure the object is runnig
if(isrunning(ppo))

  %Get present port pin status
  ppoval=getvalue(ppo);

  %AND with LEFT_OFF to ensure that particular bit is 0
```

```
  Buffer = bitand(ppoval,dec2binvec(LEFT_OFF,8));

  %Depending on input param. 'x', use RIGHT_ON or RIGHT_OFF
  if(x==1)
  Buffer = bitor(Buffer,dec2binvec(RIGHT_ON,8));
  else if(x==0)
  Buffer = bitand(Buffer,dec2binvec(RIGHT_OFF,8));
      end
  end

  %Check to see whether the resulting bits are not the same
  %as the ones currently on the port's pins.
  if(binvec2dec(bitxor(Buffer,ppoval))~=0)
  putvalue(ppo,Buffer);
  end

end
```

BrakeAll Routine

```
function BrakeAll
%Declaring globals needed for this operation
global ppo;
global Buffer;

%First, we need to ensure the object is runnig
if(isrunning(ppo))
  %Get present port pin status
  ppoval=getvalue(ppo);
  %AND with all zeros
  Buffer = bitand(ppoval,dec2binvec(hex2dec('00'),8));
end

  %Check to see whether the resulting bits are not the same
  %as the ones currently on the port's pins.
```

```
  if(binvec2dec(bitxor(Buffer,ppoval))~=0)
  putvalue(ppo,Buffer);
  end
end
```

BrakeH Routine

```
function BrakeH
%Declaring globals needed for this operation
global ppo;
global UP_OFF;
global DOWN_OFF;
global Buffer;

%First, we need to ensure the object is runnig
if(isrunning(ppo))
  %Get present port pin status
  ppoval=getvalue(ppo);
  %AND with all zeros
  Buffer = bitand(ppoval,dec2binvec(UP_OFF,8));
  Buffer = bitand(Buffer,dec2binvec(DOWN_OFF,8));
end

  %Check to see whether the resulting bits are not the same
  %as the ones currently on the port's pins.
  if(binvec2dec(bitxor(Buffer,ppoval))~=0)
  putvalue(ppo,Buffer);
  end
end
```

BrakeV Routine

```
function BrakeV
%Declaring globals needed for this operation
global ppo;
```

```
global LEFT_OFF;
global RIGHT_OFF;
global Buffer;

%First, we need to ensure the object is runnig
if(isrunning(ppo))
  %Get present port pin status
  ppoval=getvalue(ppo);
  %AND with all zeros
  Buffer = bitand(ppoval,dec2binvec(RIGHT_OFF,8));
  Buffer = bitand(Buffer,dec2binvec(LEFT_OFF,8));
end

  %Check to see whether the resulting bits are not the same
  %as the ones currently on the port's pins.
  if(binvec2dec(bitxor(Buffer,ppoval))~=0)
  putvalue(ppo,Buffer);
  end
end
```

Roverbot Motion Controller Simulink Model

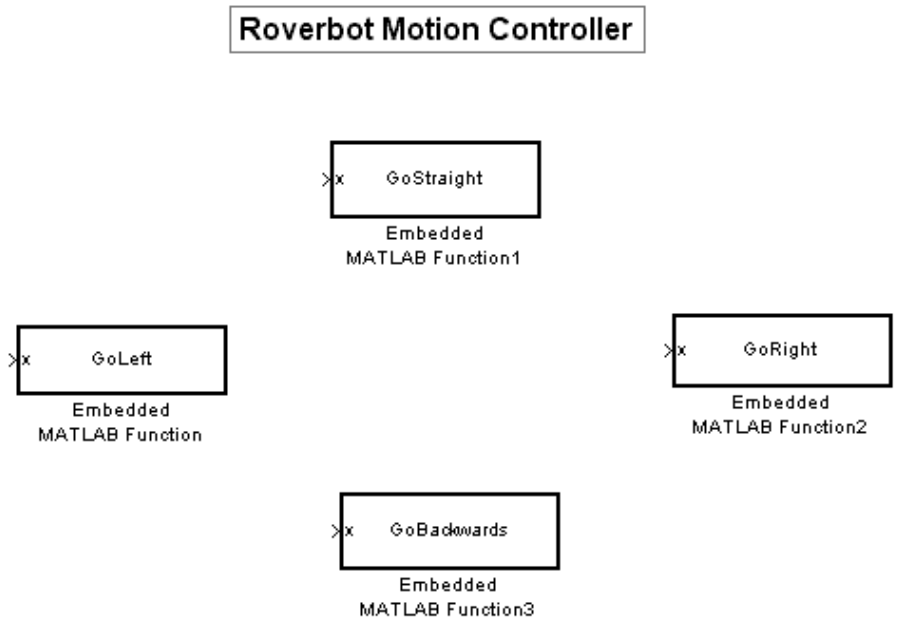


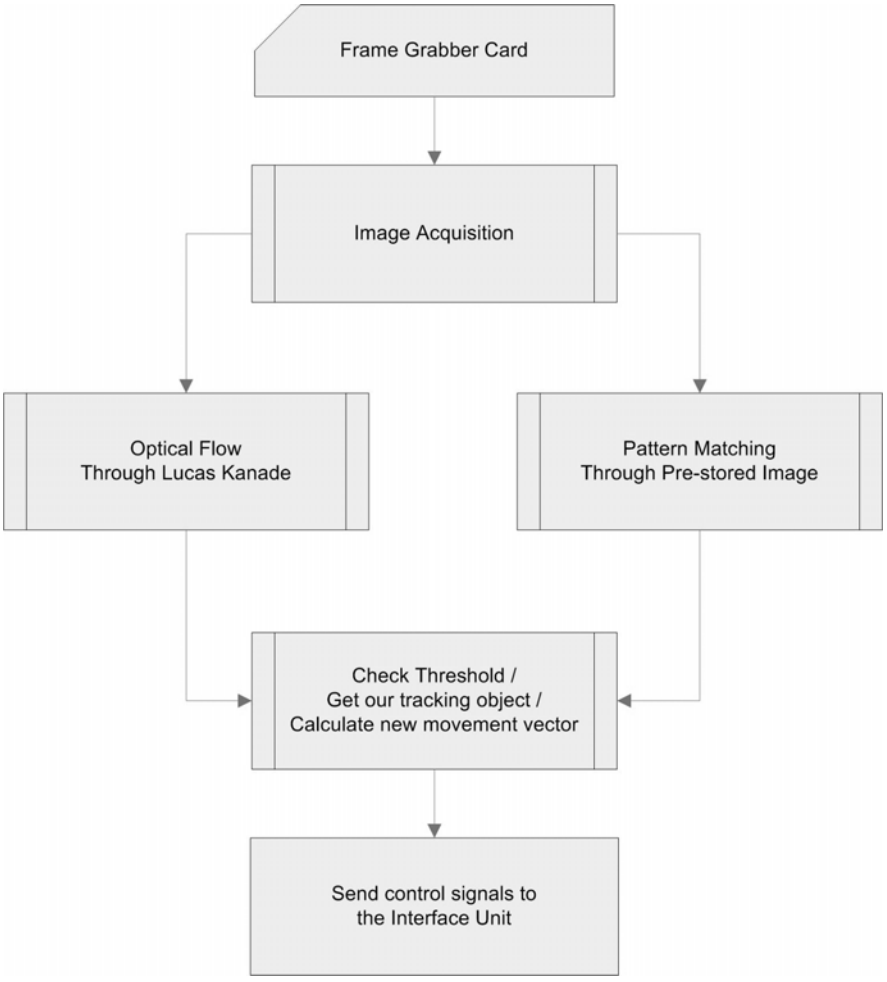
Figure: Simulink Model for Roverbot Motion Control

CHAPTER 7

SCENE ANALYSIS

7.1 Design Summary

This portion of the project is developed on Matlab using the **Video Image Processing blockset**.



Live video is grabbed through the frame grabber card which is then acquired in Matlab through the image acquisition toolbox. Two separate operations: Optical Flow and

Pattern matching are applied to track the object and new direction vector for the vehicle is calculated.

7.2 Matlab

MATLAB is a computing environment designed for engineers and scientists working in a variety of different fields. It has features for Numerical Computing, Image Analysis, Signal Processing, Artificial Intelligence and a lot of other development libraries.

7.2.1 Simulink

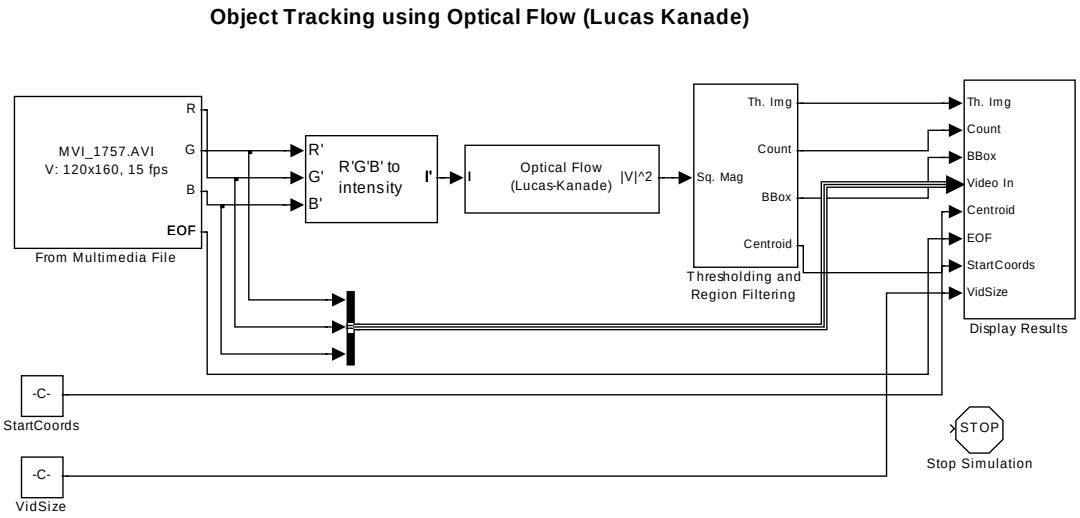
Simulink is a component of Matlab. It is a software package for modeling, simulating, and analyzing dynamic systems. It supports linear and nonlinear systems, modeled in continuous time, sampled time, or a hybrid of the two. Systems can also be multirate, i.e., have different parts that are sampled or updated at different rates.

7.2.2 VIP Blockset

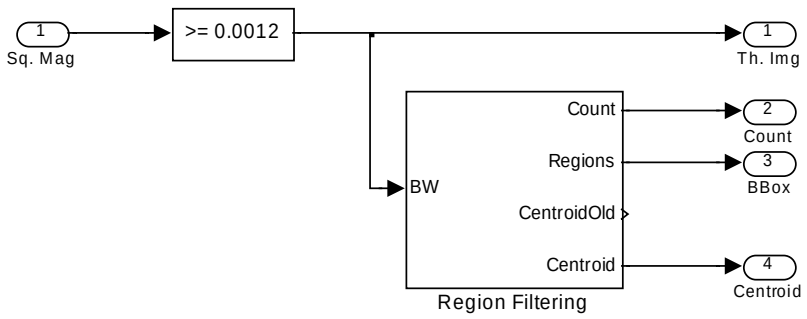
The Video and Image Processing Blockset is a tool used for the rapid design, prototyping, graphical simulation, and efficient code generation of video processing algorithms. The Video and Image Processing Blockset blocks can import streaming video into the Simulink environment and perform two-dimensional filtering, geometric and frequency transforms, block processing, motion estimation, edge detection and other signal processing algorithms.

7.3 Cell-Tracker using Lucas Kanade

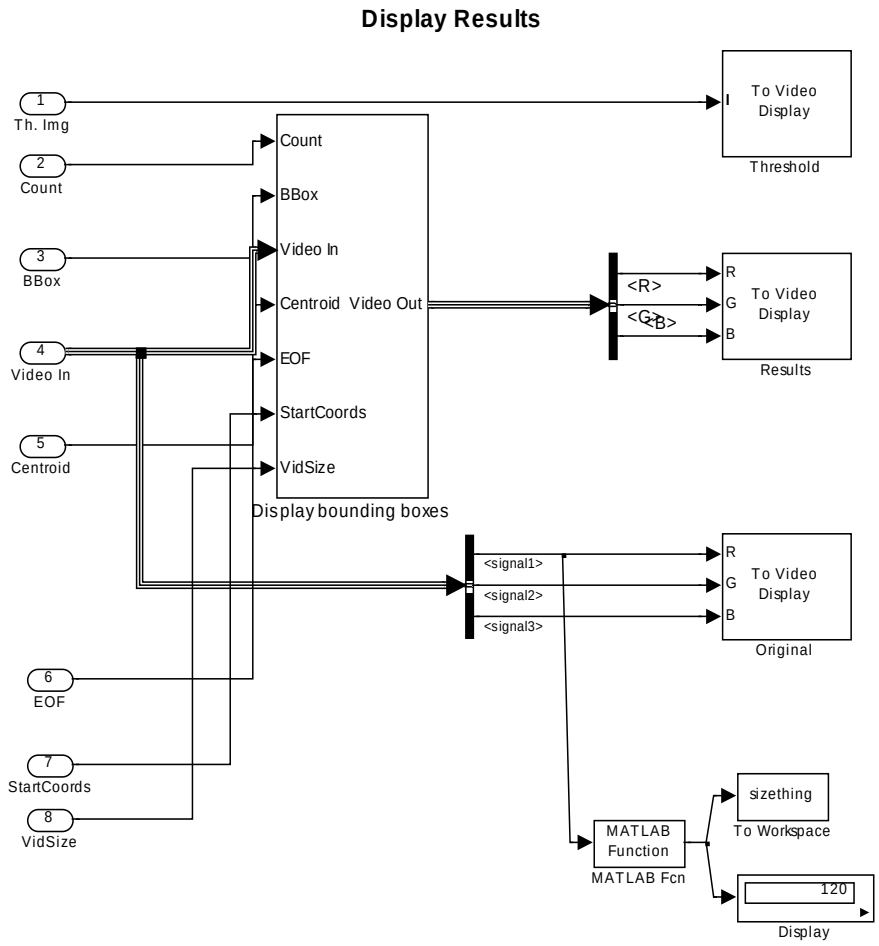
7.3.1 Code



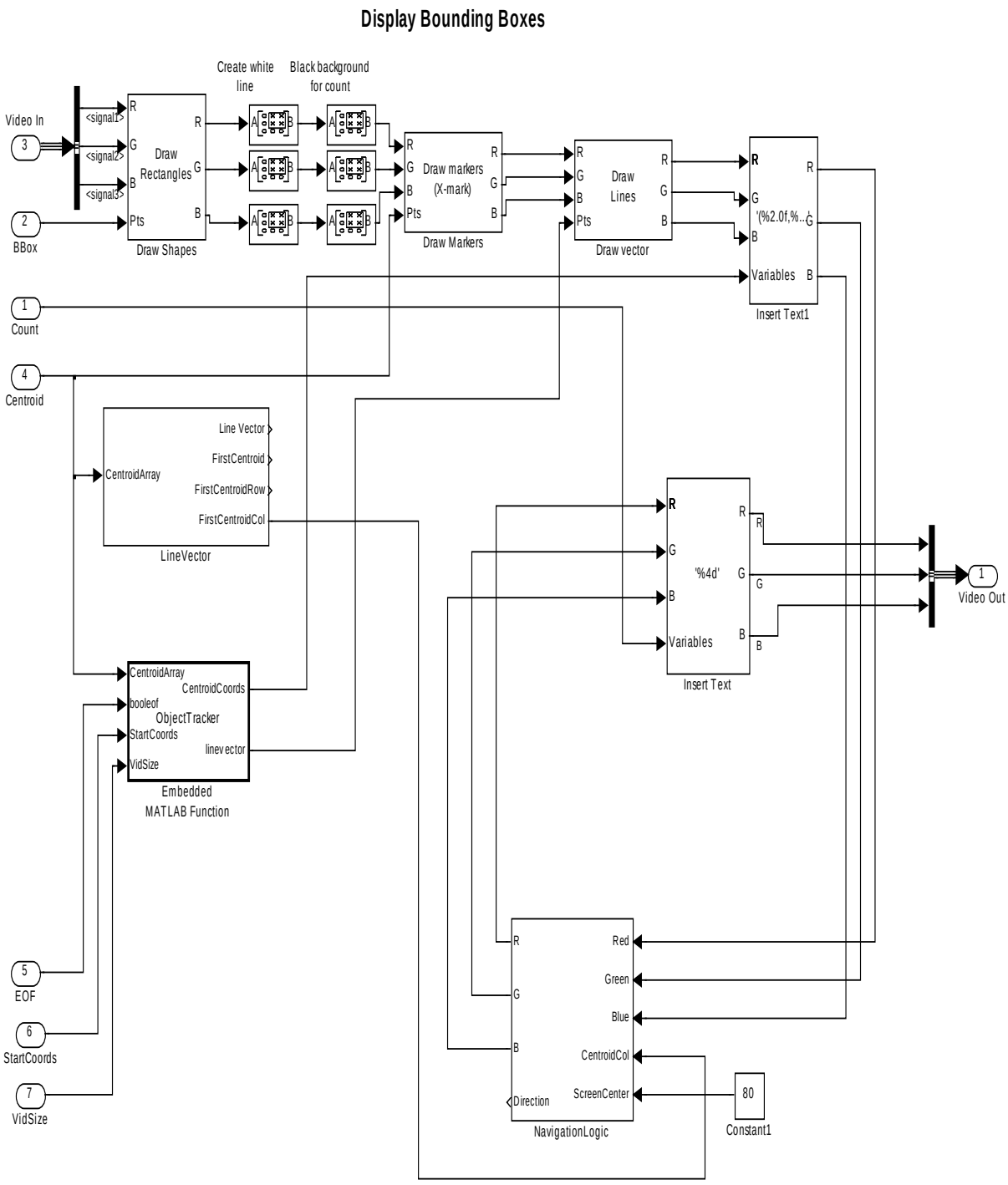
7.3.2 Threshold and Region Filtering



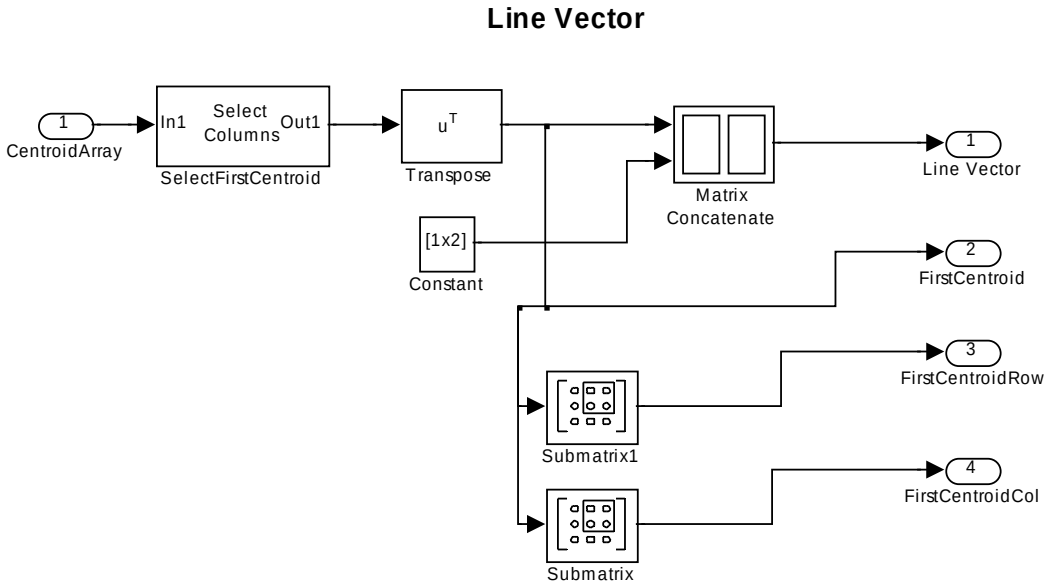
7.3.3 Results Box



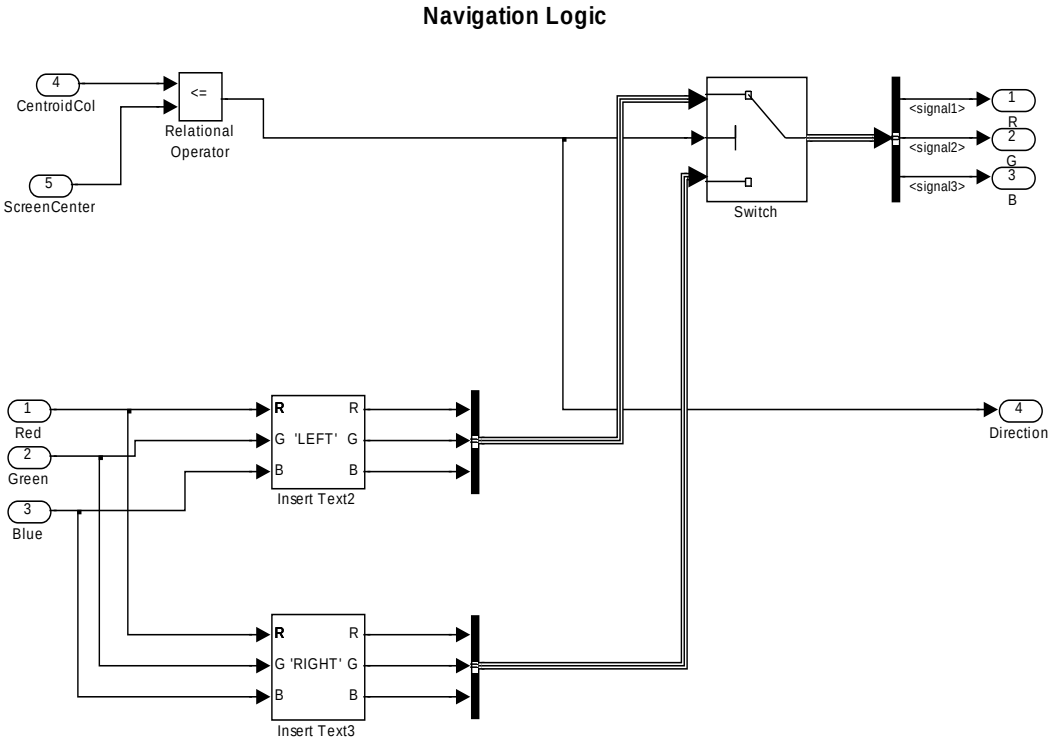
7.3.4 Display Bounding Boxes



7.3.5 Line Vector Calculation



7.3.6 Navigation Logic



7.3.7 Object Tracker Embedded Code

```
function [CentroidCoords,linevector] =
ObjectTracker(CentroidArray,booleof,StartCoords,VidSize)
persistent LastCentroid;

if isempty(LastCentroid) || booleof == true
    LastCentroid = StartCoords;
end

z = LastCentroid;

allx = CentroidArray(1,:) - z(1);
ally = CentroidArray(2,:) - z(2);
myvector = sqrt(allx.^2 + ally.^2);
[m i] = min (myvector);
centroid = CentroidArray(:,i(1));

if myvector(i(1)) < 30
    LastCentroid = centroid;
    CentroidCoords = centroid;
else
    CentroidCoords = z;
end

linevector = [ VidSize(1) , VidSize(2)/2 , CentroidCoords(1) ,
CentroidCoords(2) ];
```

Conclusion

Although the complete robot package is ready for deployment, there is still a lot of room for improvements on all fronts namely the Interface, the Robot itself and the Scene Analysis algorithms. As far as the interface goes, it could be upgraded from Parallel Port to Universal Serial Bus Port (USB), which has become the norm today. The interface could also be extended to provide greater accuracy of the mobile robot in terms of Speed Control and Degree-based Left/Right movements.

Appendix A: Algorithms

A.1 Lucas Kanade Method

Optical Flow methods try to calculate the motion between two image frames which are taken at times *t* and *t* + δ*t* at every pixel position. As a pixel at location *(x,y,z,t)* with intensity *I(x,y,z,t)* will have moved by δ*x*, δ*y*, δ*z* and δ*t* between the two frames, following image constraint equation can be given:

$$I(x,y,z,t) = I(x + \delta x, y + \delta y, z + \delta z, t + \delta t)$$

Assuming the movement to be small enough, we can develop the image constraint at *I*(*x*,*y*,*z*,*t*) with Taylor series to get:

$$I(x+\delta x, y+\delta y, z+\delta z, t+\delta t) = I(x, y, z, t)+\frac{\partial I}{\partial x}\delta x+\frac{\partial I}{\partial y}\delta y+\frac{\partial I}{\partial z}\delta z+\frac{\partial I}{\partial t}\delta t+H.O.T$$

where H.O.T. means higher order terms, which are small enough to be ignored. From these equations we achieve:

$$\frac{\partial I}{\partial x}\delta x + \frac{\partial I}{\partial y}\delta y + \frac{\partial I}{\partial z}\delta z + \frac{\partial I}{\partial t}\delta t = 0$$

or

$$\frac{\partial I}{\partial x}\frac{\delta x}{\delta t} + \frac{\partial I}{\partial y}\frac{\delta y}{\delta t} + \frac{\partial I}{\partial z}\frac{\delta z}{\delta t} + \frac{\partial I}{\partial t}\frac{\delta t}{\delta t} = 0$$

which results in

$$\frac{\partial I}{\partial x}V_x + \frac{\partial I}{\partial y}V_y + \frac{\partial I}{\partial z}V_z + \frac{\partial I}{\partial t} = 0$$

where V_x,V_y,V_z are the x,y and z components of the velocity or optical flow of $I(x,y,z,t)$ and $\frac{\partial I}{\partial x}$, $\frac{\partial I}{\partial y}$, $\frac{\partial I}{\partial z}$ and $\frac{\partial I}{\partial t}$ are the derivatives of the image at (x,y,z,t) in the corresponding directions. We will write I_x,I_y, I_z and I_t for the derivatives in the following.

Thus

$$I_x V_x + I_y V_y + I_z V_z = - I_t$$

or

$$\nabla I \cdot \vec{V} = -I_t$$

This is an equation in three unknowns and cannot be solved as such. This is known as the *aperture problem* of the optical flow algorithms. To find the optical flow we need another set of equations which is given by some additional constraint. The solution as given by Lucas and Kanade is a non-iterative method which assumes a locally constant flow.

Assuming that the flow (V_x,V_y,V_z) is constant in a small window of size $m \times m \times m$ with $m > 1$, which is centered at voxel x,y,z and numbering the pixels as $1...n$ we get a set of equations:

$$\begin{aligned}
I_{x_1} V_x + I_{y_1} V_y + I_{z_1} V_z &= -I_{t_1} \\
I_{x_2} V_x + I_{y_2} V_y + I_{z_2} V_z &= -I_{t_2} \\
&\vdots \\
I_{x_n} V_x + I_{y_n} V_y + I_{z_n} V_z &= -I_{t_n}
\end{aligned}$$

With this we get more than three equations for the three unknowns and thus an over-determined system. We get:

$$\begin{bmatrix} I_{x1} & I_{y1} & I_{z1} \\ I_{x2} & I_{y2} & I_{z2} \\ \vdots & \vdots & \vdots \\ I_{x_n} & I_{y_n} & I_{z_n} \end{bmatrix} \begin{bmatrix} V_x \\ V_y \\ V_z \end{bmatrix} = \begin{bmatrix} -I_{t1} \\ -I_{t2} \\ \vdots \\ -I_{t_n} \end{bmatrix}$$

or

$$A\vec{v} = -b$$

To solve the over-determined system of equations we use the least squares method:

$$A^T A\vec{v} = A^T(-b) \text{ or}$$

$$\vec{v} = (A^T A)^{-1} A^T(-b)$$

or

$$\begin{bmatrix} V_x \\ V_y \\ V_z \end{bmatrix} = \begin{bmatrix} I_x^2 & I_x I_y & I_x I_z \\ I_x I_y & I_y^2 & I_y I_z \\ I_x I_z & I_y I_z & I_z^2 \end{bmatrix}^{-1} (-A^T I_t)$$

This means that the optical flow can be found by calculating the derivatives of the image in all four dimensions. A weighting function *W(i,j,k)*, with $i, j, k \in [1, .., m]$ should be added to give more prominence to the center pixel of the window. Gaussian functions are preferred for this purpose. Other functions or weighting schemes are possible. Besides for computing local translations, the flow model can also be extended to affine image deformations.

When applied to image registration, such as stereo matching, the Lukas-Kanade method is usually carried out in a coarse-to-fine iterative manner, in such a way that the spatial derivatives are first computed at a coarse scale in scale-space (or a pyramid), one of the images is warped by the computed deformation, and iterative updates are then computed at successively finer scales.

One of the characteristics of the Lucas-Kanade algorithm, and that of other local optical flow algorithms, is that it does not yield a very high density of flow vectors, i.e. the flow information fades out quickly across motion boundaries and the inner parts of large homogenous areas show little motion. Its advantage is the comparative robustness in presence of noise.

A.2 Horn Schunck Method

The **Horn-Schunck method** of estimating optical flow is a global method which introduces a global constraint of *smoothness* to solve the *aperture problem* (see Lucas Kanade method for further description).

A global energy function is sought to be minimized, this function is given as:

$$f = \int ((\nabla I \cdot \vec{V} + I_t)^2 + \alpha(|\nabla V_x|^2 + |\nabla V_y|^2 + |\nabla V_z|^2))\mathrm{d}x\mathrm{d}y\mathrm{d}z$$

$$\nabla I = \begin{bmatrix} I_x \\ I_y \\ I_z \\ I_t \end{bmatrix}$$

where are the derivatives of the image intensity values along the x,y,z and t dimensions, $\vec{V}$ is the optical flow vector with the components V_x,V_y,V_z. The parameter α is a regularization constant. Larger values of α lead to a smoother flow. This function can be solved by calculating the Euler-Lagrange equations corresponding to the solution of the above equation. These are given as follows:

$$\Delta V_x - \frac{1}{\alpha} I_x (I_x V_x + I_y V_y + I_z V_z + I_t) = 0$$

$$\Delta V_y - \frac{1}{\alpha} I_y (I_x V_x + I_y V_y + I_z V_z + I_t) = 0$$

$$\Delta V_z - \frac{1}{\alpha} I_z (I_x V_x + I_y V_y + I_z V_z + I_t) = 0$$

where Δ denotes the laplace operator so that

$$\Delta V_x = \frac{\partial}{\partial x}\frac{\partial V_x}{\partial x},\ \Delta V_y = \frac{\partial}{\partial y}\frac{\partial V_y}{\partial y},\ \Delta V_z = \frac{\partial}{\partial z}\frac{\partial V_z}{\partial z}.$$

Solving these equations with Gauss-Seidel for the flow components V_x,V_y,V_z gives an iterative scheme:

$$V_x^{k+1} = \frac{\Delta V_x^k - \frac{1}{\alpha} I_x (I_y V_y^k + I_z V_z^k + I_t)}{\frac{1}{\alpha} I_x^2}$$

$$V_y^{k+1} = \frac{\Delta V_y^k - \frac{1}{\alpha} I_y (I_x V_x^k + I_z V_z^k + I_t)}{\frac{1}{\alpha} I_y^2}$$

$$V_z^{k+1} = \frac{\Delta V_z^k - \frac{1}{\alpha} I_z (I_x V_x^k + I_y V_y^k + I_t)}{\frac{1}{\alpha} I_z^2}$$

where the superscript *k+1* denotes the next iteration, which is to be calculated and *k* is the last calculated result. ΔV_i can be obtained as:

$$\Delta V_i = \sum_{N(p)} V_i(N(p)) - V_i(p)$$

where *N(p)* are the six neighbors of the pixel *p*.

An alternative algorithmic implementation based upon the Jacobi method is given as:

$$V_x^{k+1} = \overline{V_x^k} - \frac{I_x (I_x \overline{V_x^k} + I_y \overline{V_y^k} + I_z \overline{V_z^k} + I_t)}{\alpha^2 + I_x^2 + I_y^2 + I_z^2}$$

$$V_y^{k+1} = \overline{V_y^k} - \frac{I_y (I_x \overline{V_x^k} + I_y \overline{V_y^k} + I_z \overline{V_z^k} + I_t)}{\alpha^2 + I_x^2 + I_y^2 + I_z^2}$$

$$V_z^{k+1} = \overline{V_z^k} - \frac{I_z (I_x \overline{V_x^k} + I_y \overline{V_y^k} + I_z \overline{V_z^k} + I_t)}{\alpha^2 + I_x^2 + I_y^2 + I_z^2}$$

where $\overline{V_i^k}$ refers to the average of V_i^k in the neighborhood of the current pixel position. Advantages of the Horn-Schunck algorithm include that it yields a high density of flow vectors, i.e. the flow information missing in inner parts of homogeneous objects is *filled in* from the motion boundaries. On the negative side, it is more sensitive to noise than local methods.

Bibliography

[1] Robot Vision, *Stefan Florczyk*, WILEY

[2] Digital Image Processing - *Gonzalez and Woods*

[3] Digital Image Processing with MATLAB - *Gonzalez and Woods*

[4] Feature Extraction and Image Processing - *Mark Nixon & Alberto Aguado*

[5] Algorithms for Image Processing and Computer Vision - *JR Parker*

[6] Lucas B D and Kanade T 1981, An iterative image registration technique with an application to stereo vision. Proceedings of Imaging understanding workshop, pp 121--130.

[7] KLT: An Implementation of the Kanade-Lucas-Tomasi Feature Tracker

[8] Wikipedia

[9] Lecture series (from University of Sussex, UK), *David Young*

[10] Stanford Computer Vision Course CS-223b