Perception Based Obstacle Detection and Avoidance System for Autonomous Vehicles with Self-Localization Capability

Developed by

Muhammad Farooq Azam Khan

Supervised by

Dr. Sikander Hayat Khiyal

Department of Computer Science
International Islamic University, Islamabad
(2004)

In the name of

ALLAH

The most Compassionate

The most Merciful

Department of Computer Science,
International Islamic University, Islamabad.

Dated: 30-06-2004

Final Approval

It is certified that we have read the thesis, titled "**Perception Based Obstacle Detection and Avoidance System for Autonomous Vehicles with Self-Localization Capability**" submitted by **Muhammad Farooq Azam Khan** under University Reg. No. **20-CS/MS/01**. It is our judgment that this thesis is of sufficient standard to warrant its acceptance by the International Islamic University, Islamabad, for the Degree of **Master of Science**.

Committee

External Examiner
Dr. Syed Afaq Husain
Head,
Computer Sciences Department,
Shaheed Zulfikar Ali Bhutto
Institute of Science & Technology,
Islamabad.

[illegible]

Internal Examiner
Asim Munir
Lecturer,
Department of Computer Science,
Faculty of Applied Sciences,
International Islamic University,
Islamabad.

[illegible]

Supervisor
Dr. Sikander Hayat Khiyal
Head,
Department of Computer Science,
Faculty of Applied Sciences,
International Islamic University,
Islamabad.

S. Hayat. [illegible]

A dissertation submitted to the
Department of Computer Science,
International Islamic University, Islamabad
as a partial fulfillment of the requirements
for the award of the degree of
Master of Science

Declaration

I hereby declare that this software, neither as a whole nor as a part thereof has been copied out from any source. It is further declared that I have developed this software entirely on the basis of my personal efforts made under the sincere guidance of my supervisor. No portion of the work presented in this report has been submitted in support of any application for any other degree or qualification of this or any other university or institute of learning.

Muhammad Farooq Azam Khan
20-CS/MS/01

Dedication

Dedicated to my family.

Acknowledgements

Praise be to the Almighty Allah, the most Merciful, the most Gracious, the Creator, the Savior, One who ever was and One who forever will be. Without His blessings I would not have been able to complete this project.

Thanks to my Parents and sisters who were always there whenever I needed them.

I must accept that this project would never have been completed without the help of my supervisor Dr. Sikandar Hayat Khiyal. His far seeing vision and clarity of thought steered me clear of countless disastrous situations and decisions. I can honestly say, I am honored to have worked under him.

I also acknowledge my teachers for their help. Most prominently Dr. Syed Afaq Husain, for his sincere guidance and help.

They say a true friend is hard to find. I found two. I am glad to have Raheel Siddiqui and Muneeb Ahmad as friends, they always helped me out whenever I was in a tight spot.

Muhammad Farooq Azam Khan

Project in Brief

Project Title:	Perception Based Obstacle Detection and Avoidance System for Autonomous Vehicles with Self-Localization Capability
Objective:	To develop a system capable of autonomous vehicle navigation and hurdle avoidance with application of perception.
Undertaken By:	**Muhammad Farooq Azam Khan**
Supervised By:	**Dr. Sikander Hayat Khiyal** Head, Department of Computer Science, Faculty of Applied Sciences, International Islamic University, Islamabad.
Technologies Used:	Microsoft® Visual C++ 6.0, OpenCV®
System Used:	Pentium® III
Operating System Used:	Microsoft® Windows® 2000 Professional
Date Started:	1st December, 2002
Date Completed:	30th July, 2003

Abstract

The challenge for every autonomous navigation vision system is the bottle neck of high amount of data processing required in real time. Along with that, the systems capable of perceiving the environment and navigate autonomously to destination are presented with a lot of uncertainties due to complexity and unpredictability of the environment.

Against this background, this report presents a vision-based path understanding and navigation technique which aims at developing a system that would give the driving seat of a vehicle to the computer. Allowing it to do operations such as detection of path, hurdles in that path, using perception to understand the path and find destination at run time.

Suggested solution gives us satisfactory results in term of speed, accuracy and reliability and proves to be competitive software. The technique applied in this software can also be used in many other computer vision tasks such as intruder detection and motion detection.

TABLE OF CONTENTS

Chapter 1
Introduction

1. INTRODUCTION

Computer scientists all around the world are working hard to achieve one goal: an intelligent computer. In a constant quest for improvement, all possible roads are trodden to make computer understand. Numerous fields have emerged within the field of computer science. Computer vision is just one such example. The essence of computer vision is simple: "to make computer see things", and consequently, to make it "understand" the world around it. For if computer starts seeing, it's formidable ability to work on and on without making a single mistake could be utilized in a variety of places and number of ways. Computers that see could be used where humans fail. This project is yet another step in that direction.

The project in context is an effort towards taking the human out of the driving seat and place computer behind the wheel. The reason for computer driven vehicles are numerous, most examples are, computers can not fall asleep while driving, they donot get tired after a prolonged drive, they are precise and accurate, they don't get tense and so on. All in all, this project has potential to be taken up seriously.

In that respect we were successfully able to differentiate the ground plane from the rest of the image and navigate the vehicle around different hurdles while maintaining a database of all the hurdles encountered in a particular path.

The future enhancements include further reduction of computational load by making the ground and hurdle detection algorithm even lighter and increasing the ability to work on different types of ground textures.

1.1 Computer Vision

Computer vision is a field that runs parallel to Image processing. In fact, both Computer Vision and Image Processing are descendent of a single field called "Computer Imaging". The difference between Computer Vision and Image Processing is that the end user of resultant of Image Processing operations is human, while the end user of resultant of Computer Vision operations is computer. These two fields share between them a variety of definitions, processes, algorithms and methods.

Computer Vision is a field that deals with images. Just like we see with our eyes and unconsciously store important aspects in our memory, the images in computer can be captured using cameras and stored on some media such as hard disk. But the computer can do much more with the images other than just storing them. In fact, the images for computer can be of variety of formats for example: color images, gray (intensity) images, infra red images etc and they can be obtained from variety of sources for example: digital camera, web cam, internet, satellite etc. These images can have many different spectrums. These spectrums may range from simple black and white images to multi-spectrum images such as RGB images, that have three spectrums. But the list does not end here; there are satellite images that have up to 24 spectrums and others that have even more spectrums. These spectrums are of great use for storing significant data. This enormous capacity and power could be used where human endurance runs out.

1.2 Computational Cost of Processing Images

The images used in Computer Imaging are digital instead of analogue. A digital image $a[m, n]$ described in a 2D discrete space is derived from an analog image $a(x, y)$ in a 2D continuous space through a *sampling* process that is frequently referred to as "digitization". The 2D continuous image $a(x, y)$ is divided into *N rows* and *M columns*. The intersection of a row and a column is termed a *pixel*. The value assigned to the integer coordinates $[m,n]$ with $\{m=0,1,2,\ldots,M-1\}$ and $\{n=0,1,2,\ldots,N-1\}$ is $a[m,n]$. The value assigned to every pixel is the average brightness in the pixel rounded to the nearest integer value.

As every digital image is made up of elements called "pixels", we can say that operations in Computer Vision (and Image processing) are actually operations done on pixels.

The images obtained from web cams or cameras are generally of pre-specified standard sizes. For example an image captured from a web cam can be of 640 by 480 pixels or 320 by 240 pixels. Thus an image of 320 by 240 means 76,800 pixels. This means if we have a single spectrum image of dimension 320 by 240, the computer will have to work with 76,800 different pixels and pixel values. If we have a single spectrum image of dimension 640 by 480, the computer will have to work with 307200 different pixels and pixel values. In the same way, if we have a commonly used RGB image (i.e. three spectrums) image of dimension 320 by 240, the computer will have to work with 76, 800 different pixels and 2,30,400 pixel values. In the same way, if we have a RGB image (i.e. three spectrums) image of dimension 640 by 480, the computer will have to work with 2,30,400 different pixels and 6,91,200 pixel values. So a single RGB image of 640 by 480 pixels dimensions require 6, 91,200 computations for reading the image. If you need to do some processing with the image, that would naturally require extra computations since 6, 91,200 computations are just for reading the image (lets say) into memory. Now if you have a filter that makes two passes on the image, then it would require 13, 82,400 extra computations (i.e. apart from loading the image). If you make an application that process real time images from web cam at a rate of 10 frames per second then the number of required computations would be at least 2,07,36,000 computations per second. That is quite a load, since generally you don't just require only two passes, you have to access pixel locations, change pixel values, recompute, save and so on.

This computation load is a great bottle neck for all real time vision systems to date. The challenge for every autonomous navigation vision system is this bottle neck of high amount of data processing required in real time. The vision scientists are constantly trying to tackle this issue through the introduction of newer techniques.

1.3 Autonomous Navigation

Autonomous navigation means navigation not requiring human help, in other words "automatic navigation". This navigation is not limited to streets and roads, rather it encapsulates on-ground, under-ground, underwater and even space navigation.

Autonomous navigation has great field of applications in our daily life. Autonomous machines are already helping us build cars, rockets and doing many other useful tasks in many cases better than any human can do. As already mentioned, the use of autonomous navigation range from indoor environment to outer space exploration. One such example could be the baseline 2003/2005 Mars Sample Return missions. The baseline 2003/2005 Mars Sample Return missions require the return of a science rover to the lander for the transfer of sample cache containers to a Mars Ascent Vehicle or MAV. Along these lines, the newest mission guidelines for the Mars Sample Return call for a science rover to descend from the lander using ramps, acquire core samples from as far away as hundreds of meters from the lander, return to the lander, and then ascend the ramps to deposit these samples in the MAV. The return operation requires tracking and docking techniques for the development of necessary integrated rover capabilities key to the lander rendezvous operation. The science rover must autonomously recognize, track, and precisely rendezvous with the lander from distances as far away as hundreds of meters. The Sample Return Rover, or the SRR, is a rover prototype that was originally developed for the rapid retrieval of samples collected by longer ranging mobile science systems, and the return of these samples to an Earth ascent vehicle [1].

"Systems-of-systems will, in the 21st Century, replace every major combat system on the battlefield with distributed robots —in the air and on the ground, autonomous, netcentric, and integrated." —Unknown DARPA Source [2]

This vision is driving much of the current robotics research in government and defense laboratories around the globe. Its realization will require a demonstrable capability in Intelligent Autonomy (IA), i.e., "The capability to operate effectively, singly or in groups, with reduced, remote (geographically or temporally) or no human command and interaction, and the ability to adapt independently to a changing, uncertain, unpredictable and hostile external environment."[2]

Thus the importance of autonomous navigation cannot be over emphasized.

1.4 Perception

Literally perception is defined as the "ability to understand" (CHAMBERS Dictionary for Learners, BRITISH NATIONAL CORPUS). That is, to understand anything, for example: the world around us, what one is saying and so on. However the word "Perception" has a special meaning in computer vision and is related to an outstanding quality of every human being. Lately, the term "Machine Perception" was introduced. For the past few years, a lot of research has been done in the field of machine perception. *In the early years the development was mainly driven by space, underwater, and automation applications for hazardous areas, but especially in the recent five years, different factors have led to an increasing number of applications. The first important factor was the exponentially growing computational power, enhanced control algorithms, and new mechatronic sensors and actuators [3].*

1.4.1 Perception and Autonomous Navigation

Perception to humans comes easy. We are adept at undetstanding different things and situations. Scintists attribute this skill to millions of years of evolution. Further more this ability to understand and adapt to cahnging environment has been one of the biggest factors of human survival through the centuries. But computers on the other hand are relatively "newborn" as compared to humans and they do not have that "instinct" to survive or understand. If somehow we can make computers see the world around it, the next possible step is to make them understand the world around them. These two goals, if achieved can bring computers closer to the humans. Scientifically, it would increase the already dominant usablilty of the computers in all fields of life.

Perception has many applications in the field of computer vision. It can allow computer to take premptive actions. Specially in the field of navigation, perception can be of very much use in deciding which path to take, where to turn, slow down or speed up, recognize landmarks and so on.

1.5 Literature Survey

In order to judge our work it is important to bring into perspective the existing work in this field.

a - A Path Following System for Autonomous Robots with Minimal Computing Power

This work by Andrew Thomson and Jacky Baltes [4] deals with following a pre-specified path. The path to follow is marked by illuminating it. The robot contains of single camera facing the ground directly in front of the vehicle. The illuminated region within each frame is searched from the gray scale image. The system tries to stay at the centre of this illuminated strip.

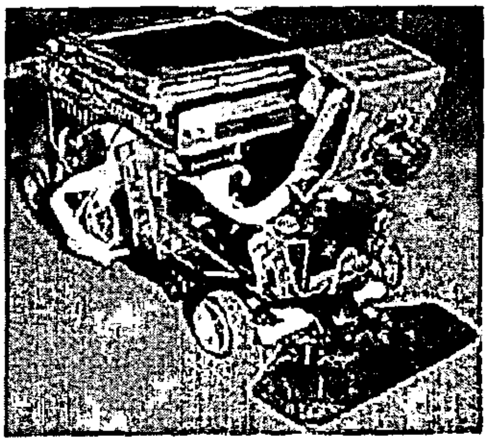


Fig. 1-1: CITR Autonomous robot

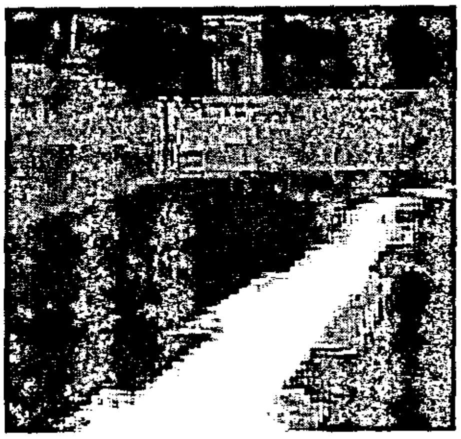


Fig. 1-2: Gray scale image of the path

b - Obstacle Avoidance of Autonomous Mobile Robot using Stereo Vision Sensor

This work by Masako Kumano and Akihisa Ohya [5] use two monochrome CCD cameras equipped with about 90 degrees wide-angle lenses, which are fixed on the left and right side with the same height at the top of the robot (See Figure 1-3). Two images are captured synchronously on an image processing board. One image is estimated from the other by the matrix calculated with relative position. Then each point on the real image is compared with the corresponding one on the estimated image. If there is a certain difference of brightness, around there any 3D objects are detected.

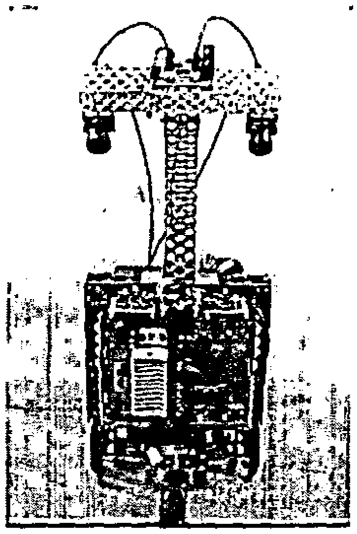


Figure 1-3: Mobile robot equipped with stereo vision sensor

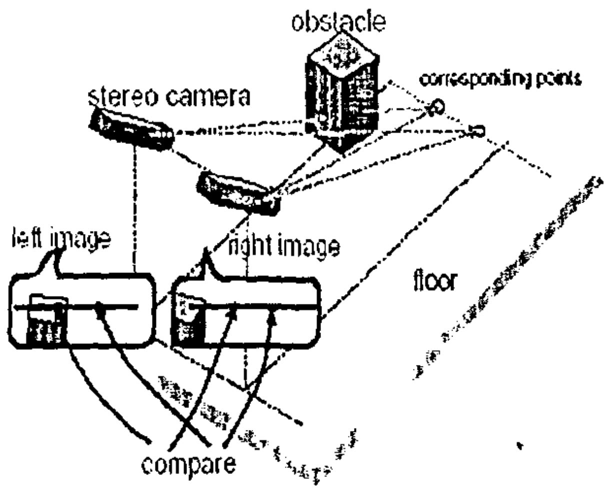


Figure 1-4: The principle of obstacle detection in this research is that if both right and left of the brightnesses of corresponding points are almost equal, there is not any obstacles there. The difference of brightnesses means there is something around the point.

c - Ground Plane Detection using Visual and Inertial Data Fusion

This work by Jorge Lobo and Jorge Dias [6] uses inertial information for navigation. In humans this inertial information is obtained from a sensorial system which is located in the inner ear and it is crucial for several visual tasks and head stabilization. This work only deal with stereo vision based ground plane detection and does not take into account hurdles in the path. The inertial unit is placed at the middle of two stereo cameras. Each camera position has its own referential, *{R}* and *{L}* being for the right and left positions.

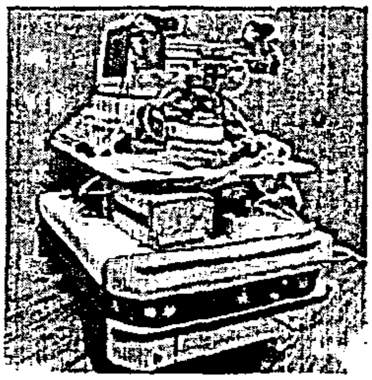


Figure 1-5: The mobile system with the active vision system

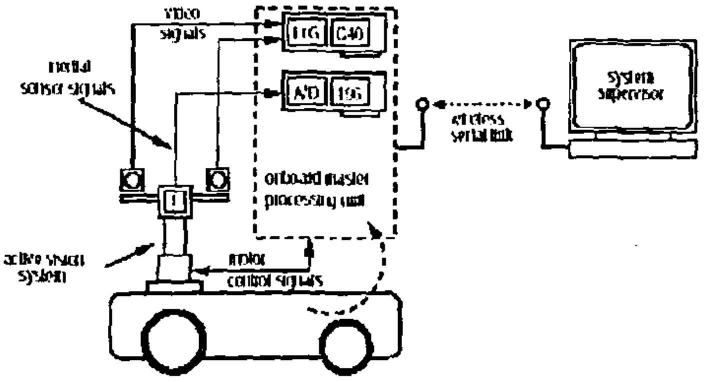


Figure 1-6: System Architecture. The inertial system processing board uses the Master processing unit as host computer.

d - Obstacle Detection and Self-Localization without Camera Calibration Using Projective Invariants

In this work Kyoung Sig Roh, Wang Heon Lee and In So Kweon [7] detect obstacles by comparing the pre-stored risk zone with a current risk zone. The positions of the detected obstacles are also determined by relative positioning. Their system makes use of the assumption that an environmental map database is available for matching between the scene and the model. Intersection points between floor and the vertical lines of door frames are used as point features to compute cross ratios. As an off-line process, the system construct a database consisting of the cross ratios of point features. Using the cross ratios in the constructed database, the correspondences between the model and scene features can be found. The corresponding point features in the database of a real environment and in the image are used to compute the positions of the mobile robot and obstacles inside the risk zone.

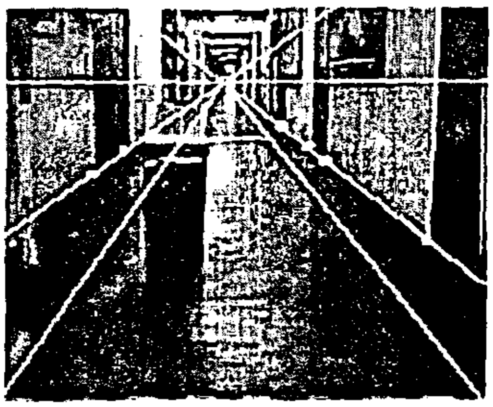


Fig 1-7: Risk Zone and point features

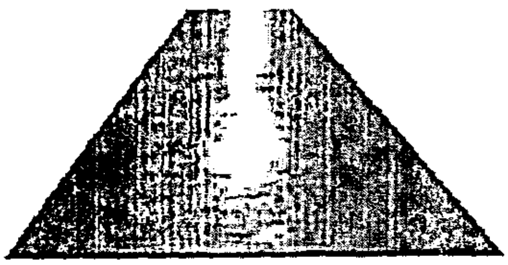


Fig 1-8: Reference risk zone

e - Deriving and Matching Image Fingerprint Sequences for Mobile Root Localization

This work by Lamon Pierre [8] deals with localization of a vehicle. Mr. Pierre proposed a method for mobile vehicle localization called Finger Print Sequencing. According to this method, as the fingerprints of a person are unique, so are at each location the unique visual characteristics (save in pathological circumstances). So a unique virtual *fingerprint* of a location can be created. If locations are denoted by unique fingerprints in this manner, then the actual location of a mobile robot/vehicle may be recovered by constructing a Fingerprint from its current view and comparing to a database of known fingerprints.

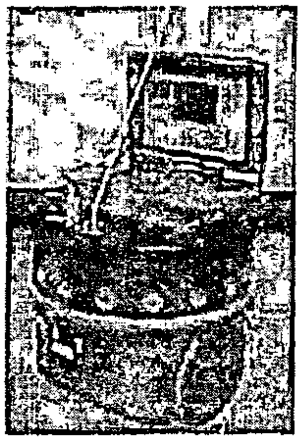


Fig 1-9: Views of the system

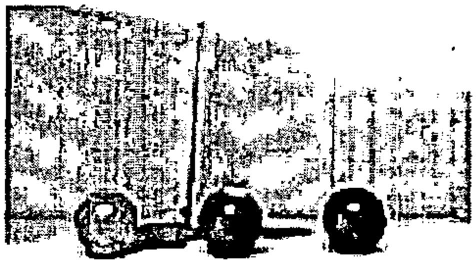


Fig 1-10: Example image for sequence encoding

Fig 1-11: String extracted from image of Fig. 1-10

1.5.1 Conclusions Drawn from Literature Survey

Self-localization, obstacle detection and avoidance are the basic requirements for successful navigation of any autonomous vehicle. Most of the vision based navigation systems concentrate on only one aspect of autonomous navigation. Some systems try to detect obstacles in their path while others try to find their relative position with respect to environment without the facility of obstacle avoidance.

The work by Andrew Thomson and Jacky Baltes [4] uses illumination underneath the floor for ground detection. This approach does not take into account obstacles detection or avoidance. That makes this approach a little far from natural environment where a path can not always be illuminated from underneath. The work by Kumano, et al [5] uses stereo vision for obstacle detection. They use two monochrome CCD cameras equipped with about 90 degrees wide-angle lenses, which are fixed on the left and right side with the same height at the top of a robot. Two images are captured synchronously on an image processing board. One image is estimated from the other by the matrix calculated with relative positioning. If there is a certain difference of brightness, around there any 3D objects are detected. The handicap of this approach is that if both of the cameras are focused on the same obstacle, that obstacle would not be detected. In their work Roh, et al [7] detect obstacles by comparing a pre-stored risk zone with a current risk zone. The positions of the detected obstacles are also determined by relative positioning.

Although a lot of work has been done in exploring the possibility of vision in autonomous navigation, much is left to be done to make it a part of our indoor/outdoor every day life.

1.6 The Project

The project *Perception Based Obstacle Detection and Avoidance System for Autonomous Vehicles with Self-Localization Capability* is an effort towards taking the human out of the driving seat and place computer behind the wheel.

The project could be divided into two major parts.

1- Navigation:

The navigation part deals with operations such as detection of path and hurdles in that path on and taking actions to avoid the hurdles on run time

2- Perception

The word "Perception" in our project has a special and slightly different meaning. Since perception means understanding, with reference to autonomous navigation, we have defined perception as *"understanding some natural/artificial aspect of the environment and taking preemptive and reactive actions accordingly in order to facilitate navigation"*.

1.7 Project Scope

Previous work in this field has mostly concentrated on any one aspect of navigation. That is why you would come across an application that would be capable of detecting a hurdle and navigating around it while another that could only reach destination by following some artificial property (such as specially colored floor) without any hurdles in the way (or stop on finding one). Another observation is the heavy dependence on specialized equipment in most of the existing systems. The reason being the need of improving accuracy and reducing computational load. Apart from that, some things are inevitable without the use of external hardware.

The scope of this project is to construct a system that would be capable of navigation in a controlled environment. In other words, the project requires development of a system that would be able to work in natural environment with reliable accuracy. In practice, the system should be capable of detecting the ground plane, separate hurdles from the ground plane, maintain a track record of encountered hurdles.

Another important aspect of the project is that no specialized equipment such as frame grabber or laser range finder are used to keep the development cost of the project as low as possible.

1.8 Objectives

The objectives of this project are given below:

- Ground plane detection
- Hurdle detection
- Hurdle avoidance
- Localization

In practice, we would require two tasks to meet the objectives of the project:

1- Real time algorithm for ground plane detection, hurdle detection and hurdle avoidance.

2- Application of perception for localization

1.9 Conclusion

In this chapter we gave an introduction to our field of work with brief definitions and explanations of basic concepts that are important in our system. Along with that we gave examples of work that has already been done in this field, what were their shortfalls and the need of what more has to be done. At the end we defined our scope of work and objectives.

Chapter 2
System Analysis

2. ANALYSIS

Software engineering, at a technical level, begins with a series of modeling tasks that lead to a complete specification of requirements and a comprehensive design representation for the software to be built. The Analysis model, actually a set of models, is the first technical representation of a system. Over the years many methods have been proposed for analysis modeling. However just two models now dominate the analysis modeling landscape. The first, *structured analysis* is a classical modeling method and the other approach is *object oriented* method. We have used the both modeling techniques for the analysis of our project, *Perception Based Obstacle Detection and Avoidance System for Autonomous Vehicles with Self-Localization Capability.*

2.1 Structured Analysis

It is a model building activity. Using a notation that satisfies the operational analysis principles, we create models that depict information (data and control) contents and flow, we partition the system functionally and behaviorally, and we depict the essence of what must be built.

2.2 Analysis Model

The Analysis Model must achieve three primary objectives.

1. Describe what the customer requires.
2. Establish a basis for the creation of a software design.
3. Define a set of requirements that can be validated once the software is built.

To accomplish these objectives, the analysis model derived during the structured analysis takes the form illustrated in Figure 2-1.

At the core of the model lies the data dictionary — a repository that contains description of all data objects consumed or produced by the software. Three different diagrams surround the core. The entity-relationship diagram (ERD) depicts relationships between data objects. The ERD is the notation that is used to conduct the data modeling activity. The attributes of each data object noted in the ERD can be described using a data object description.

The Data flow Diagram (DFD) serves two purposes:

1. Provide an indication of how data are transformed as they move through the system.

2. Depict the functions and subfunctions that transform the data flow.

The DFD provides additional information that is used during the analysis of the information domain and serves as a basis for the modeling of function. A description of each function presented in the DFD is contained in a process specification (PSPEC).

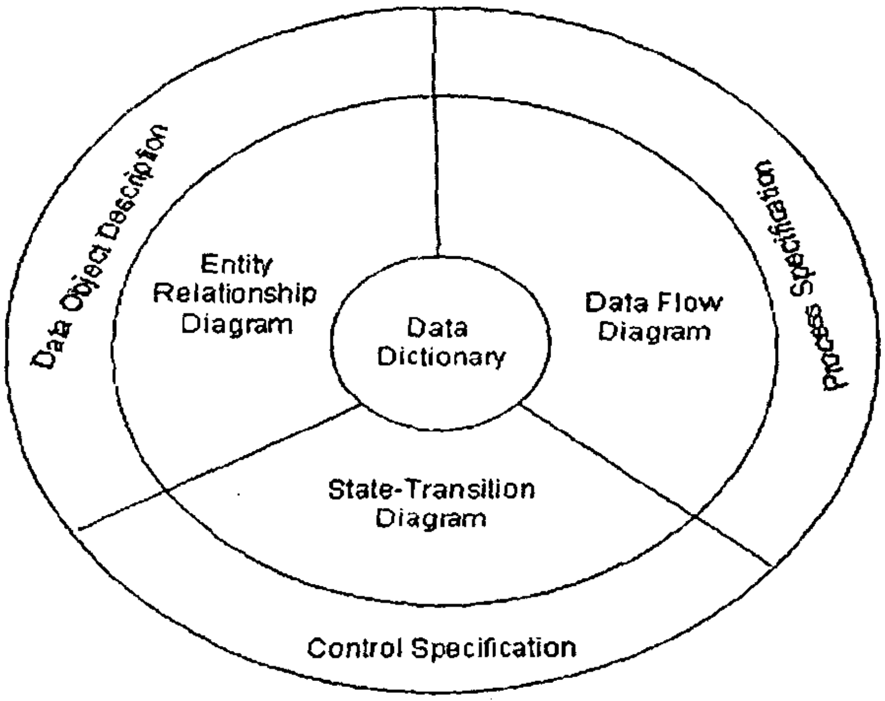


Figure 2-1: Analysis Model.

The State-transition diagram (STD) indicates how the system behaves as a consequence of external events. To accomplish this, the STD represents the various modes of behavioral modeling. Additional information about control aspects of the software is contained in the control specification (CSPEC).

2.2.1 Object Description

Object Description is used to describe the Objects. Object is a representation of almost any composite information that must be understood by the software. By composite, we mean something has a number of different attributes or properties. For example, in *Perception Based Obstacle Detection and Avoidance System for Autonomous Vehicles with Self-Localization Capability*, an object can be hurdle. Whenever the object "hurdle" is encountered, the system shifts from path navigation to hurdle tracking and navigation mode. It is used as a means of identifying the path and is compared with other objects (hurdles) for recognition purposes.

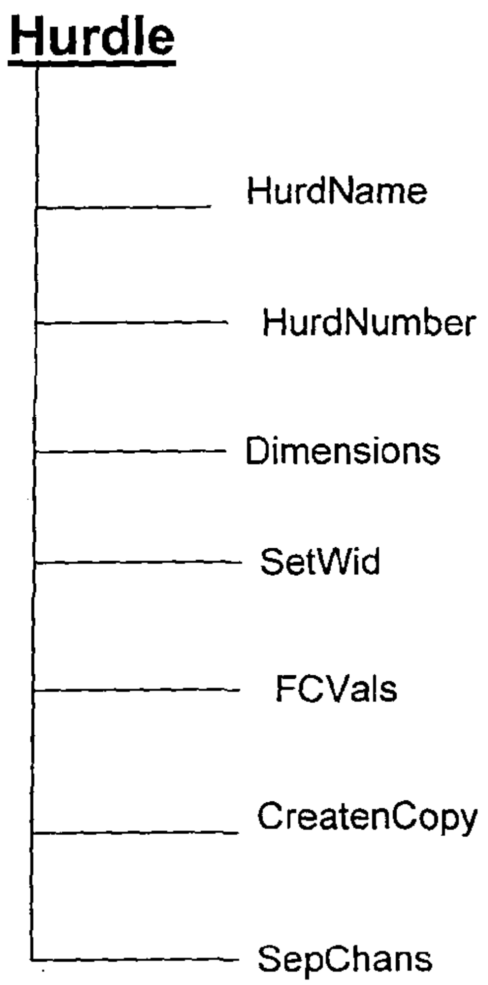


Figure 2-2: The Hurdle Object.

Let us discuss the fields of the Hurdle object:

- **HurdName** specifies the name of the hurdle. The initial name assigned to the hurdle is "Un-Named Hurdle". The name may be changed later on finding a successful match in the database.

- **HurdNumber** is an internal index of each object called "Hurdle" to separate and distinguish it from the rest of the hurdles.

- **Dimensions** is a three dimensional array used to store the location and size of the hurdle in a particular frame.

- **SetWid** is a member function of Hurdle object which is used to fill the Dimension array one element at a time.

- **FCVals** is the member function of Hurdle object which actually calls another function (CheckIt()) in order to find out whether hurdle has been encountered for the first time or if it is already present in the hurdle database.

- **CreatenCopy** is actually master-mind behind each Hurdle object, it receives the information about a hurdle and creates the hurdle, calling all the subsequent functions.

- **SepChans** separates the three channels of the hurdle and stores them into array in order to facilitate recognition process.

- **ShowHurdle** used for displaying the Hurdle object in a window named after that hurdle.

2.2.2 Entity-Relationship Diagram (ERD)

ERD is used to define the relationship between different entities or objects. These objects are joined with the other based on the relationship they have. ERD focuses solely on data (and therefore satisfies the first operational analysis principle), representing a "data network" that exist for a given system. ERD is especially useful for applications in which data and the relationships that govern data are complex. ERD of the system is given in figure 2-3.

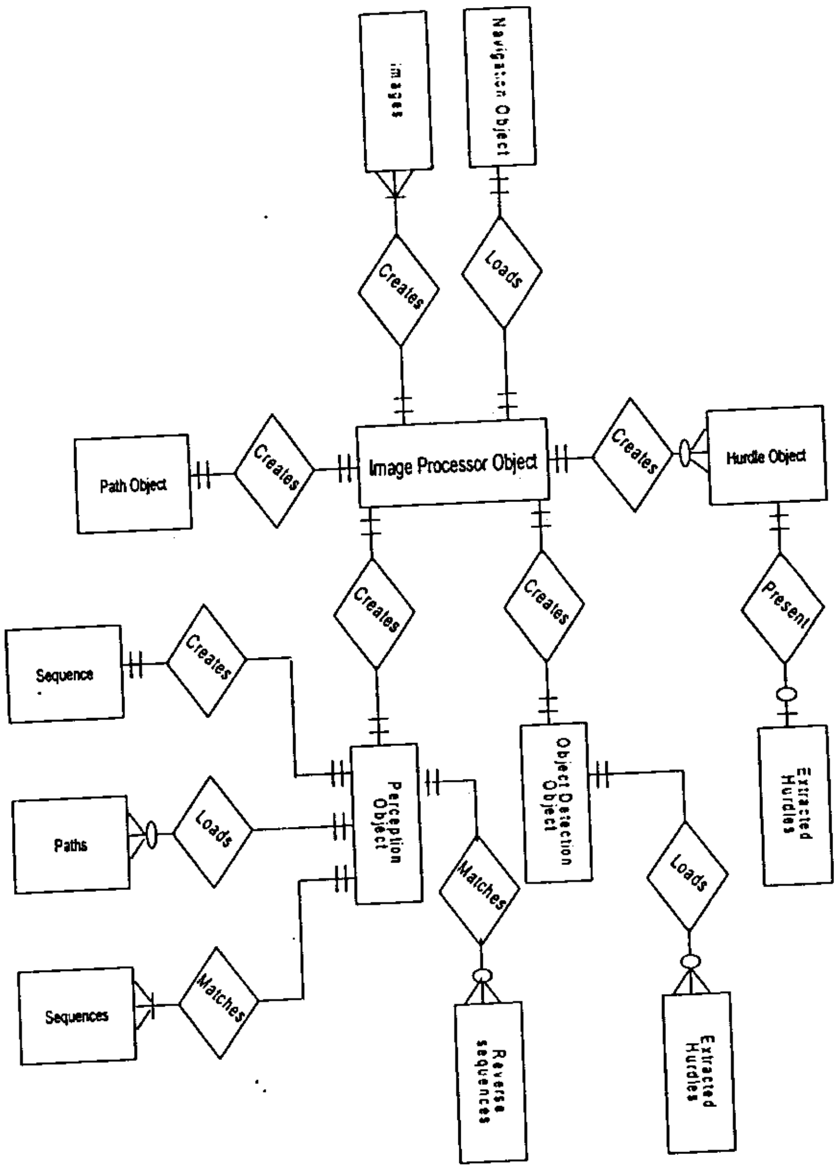


Figure 2-3: Entity-Relation Diagram (ER Diagram) for Vision One - *Perception Based Obstacle Detection and Avoidance System for Autonomous Vehicles with Self-Localization Capability.*

2.2.3 Data Flow Diagrams (DFD)

As information moves through the software, it is modified by a series of transformations. A DFD is a graphical technique that depicts information flow and the transforms that are applied as data moves from input to output. The DFD is also known as *Data flow graph* or a *bubble chart*.

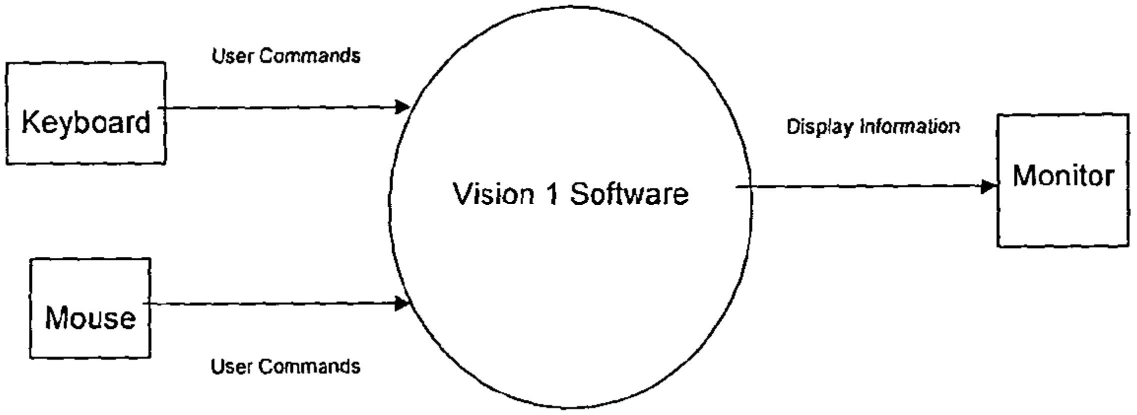


Figure 2-4: Context Level DFD for Vision 1 - *Perception Based Obstacle Detection and Avoidance System for Autonomous Vehicles with Self-Localization Capability*

Figure 2-4 shows the Context Level DFD for the software Vision 1 - *Perception Based Obstacle Detection and Avoidance System for Autonomous Vehicles with Self-Localization Capability*. This level is the highest level of abstraction where no details are shown; only the input to the software and output from software is shown. There is only one bubble, which is the software and reveals no function of the software.

Now the DFD is expanded and level one shows the detail of the process or functions of the software.

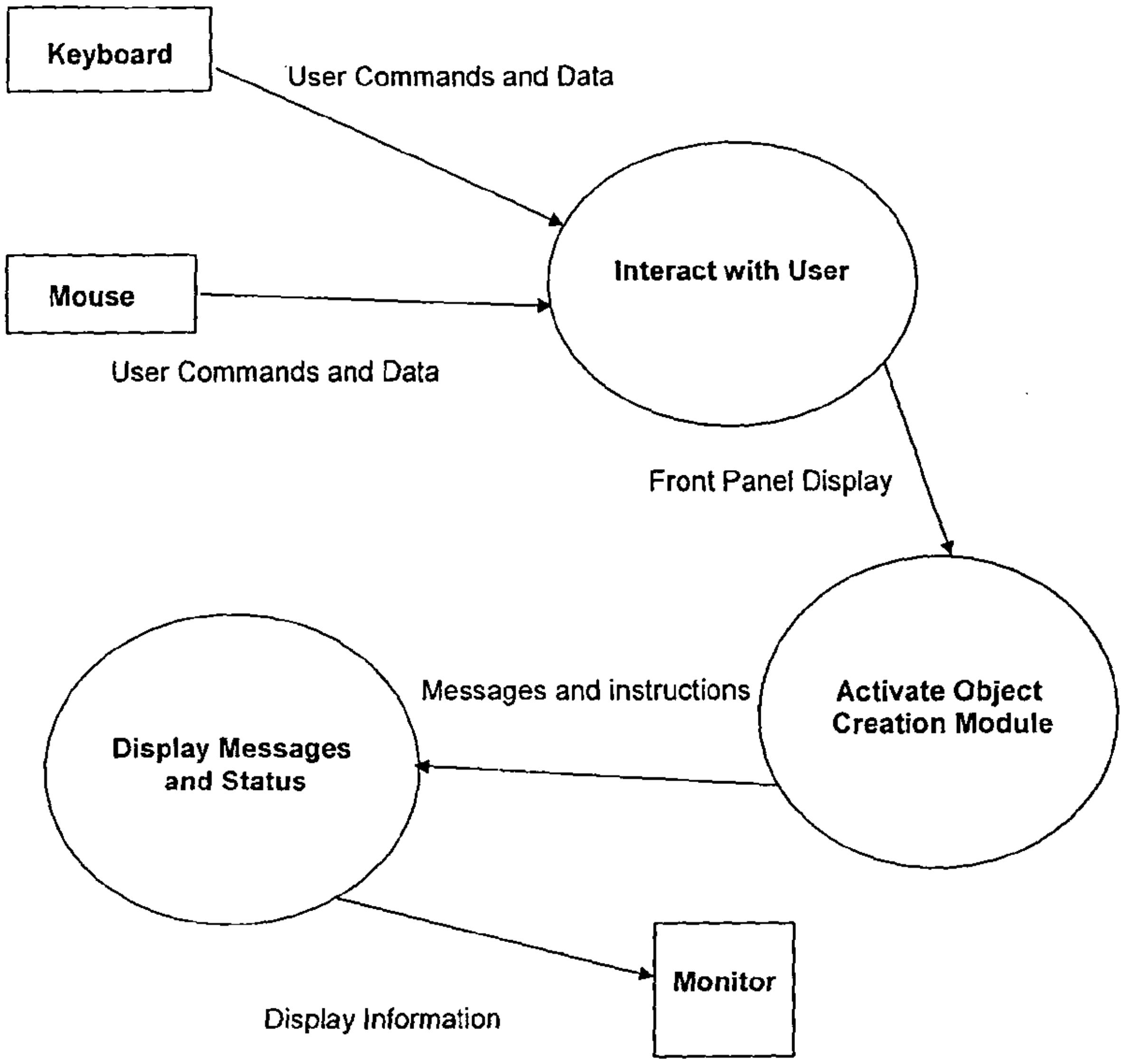


Figure 2-5: Level 1 DFD for Vision 1

Figure 2-5 expands the bubble in Level 1 DFD and here the level of abstraction decreases but only up to the functions still the sub functions are not reveled. It also shows the data storage and the arrow, to show which process stores the data and which process use the stored data.

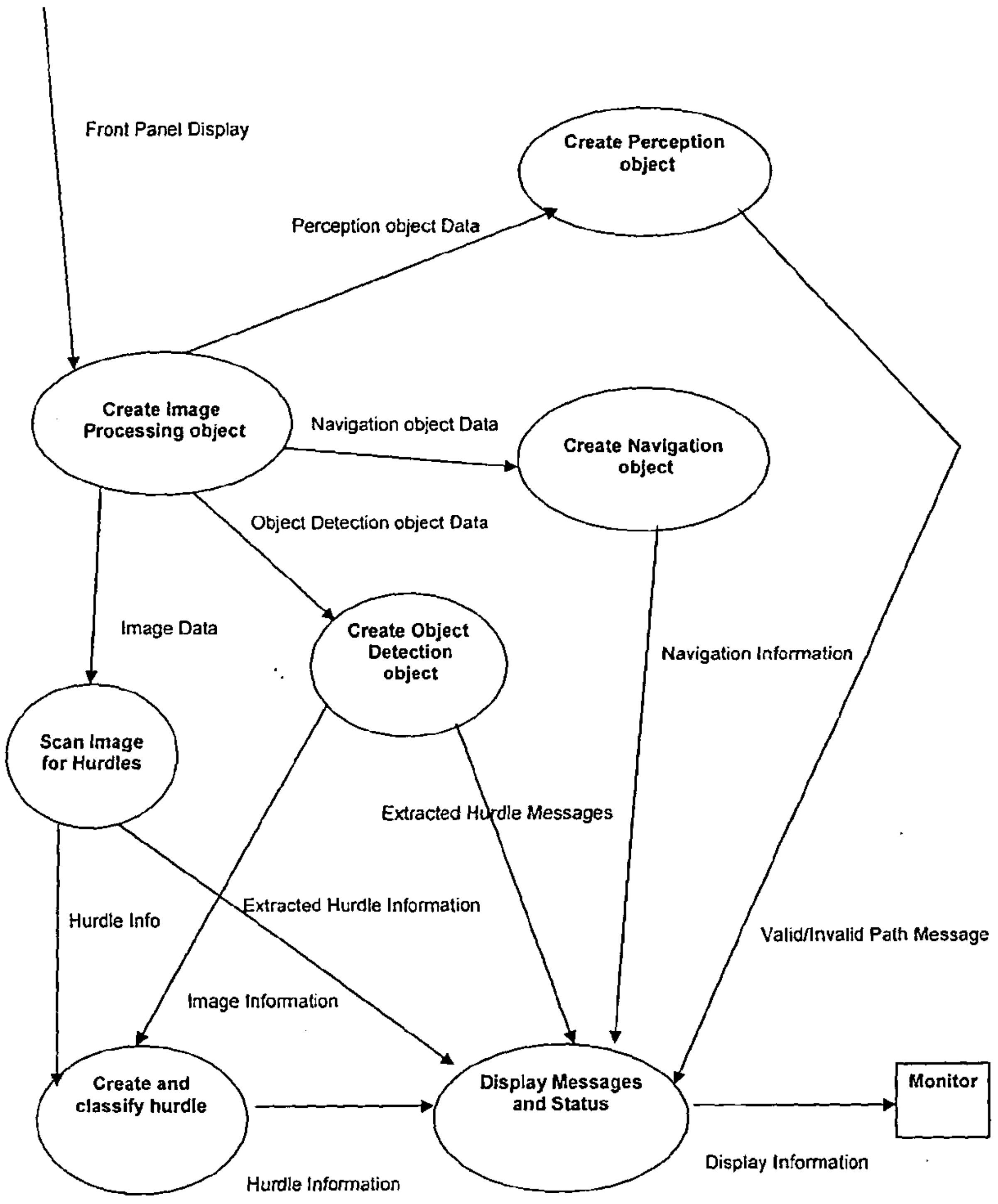


Figure 2-6: Level 2 DFD for Process Activate Object Creation Module.

Level 2 DFD process Activate Object Creation Module is shown in Figure 2-6. This level shows the sub functions of the process Activate Object Creation Module and describes almost all the data flow in the process Activate Object Creation Module.

Figures 2-7, 2-8, 2-9 and 2-10 show Level 3 DFD of processes "Scan Image for Hurdle", "Match Sequence", "Create Perception Object" and "Create object Detection Object" respectively.

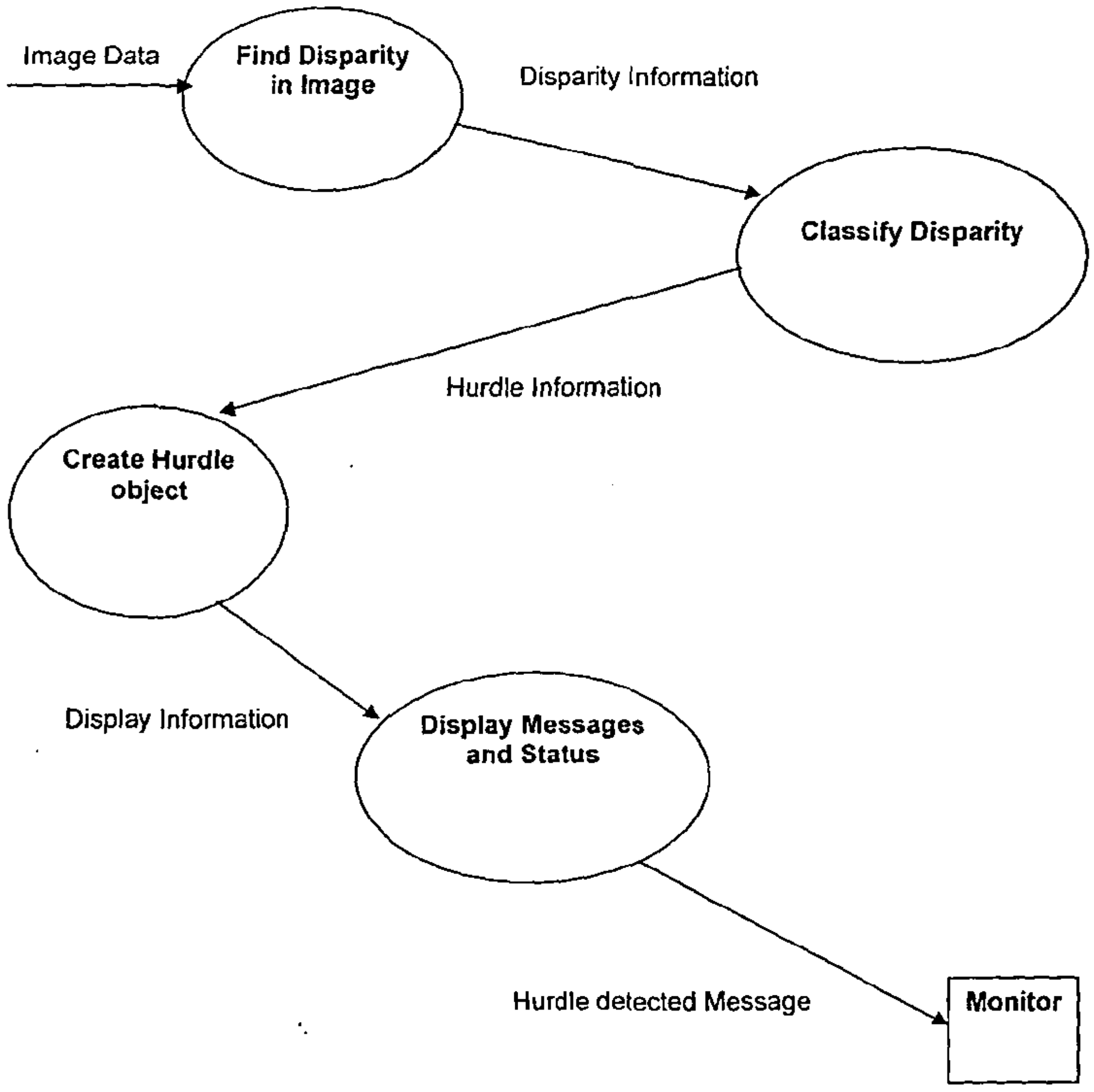


Figure 2-7: Level 3 DFD for Process Scan Image for Hurdle.

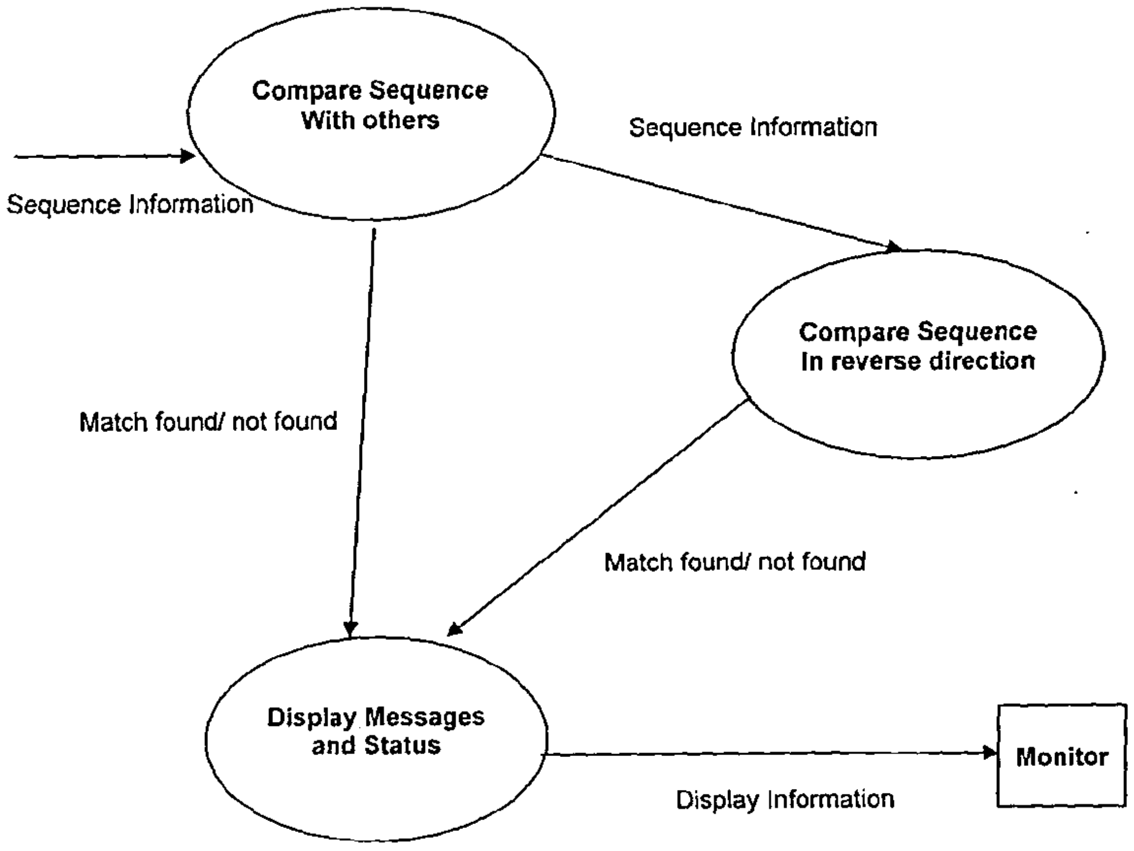


Figure 2-8: Level 3 DFD for Process Match Sequence

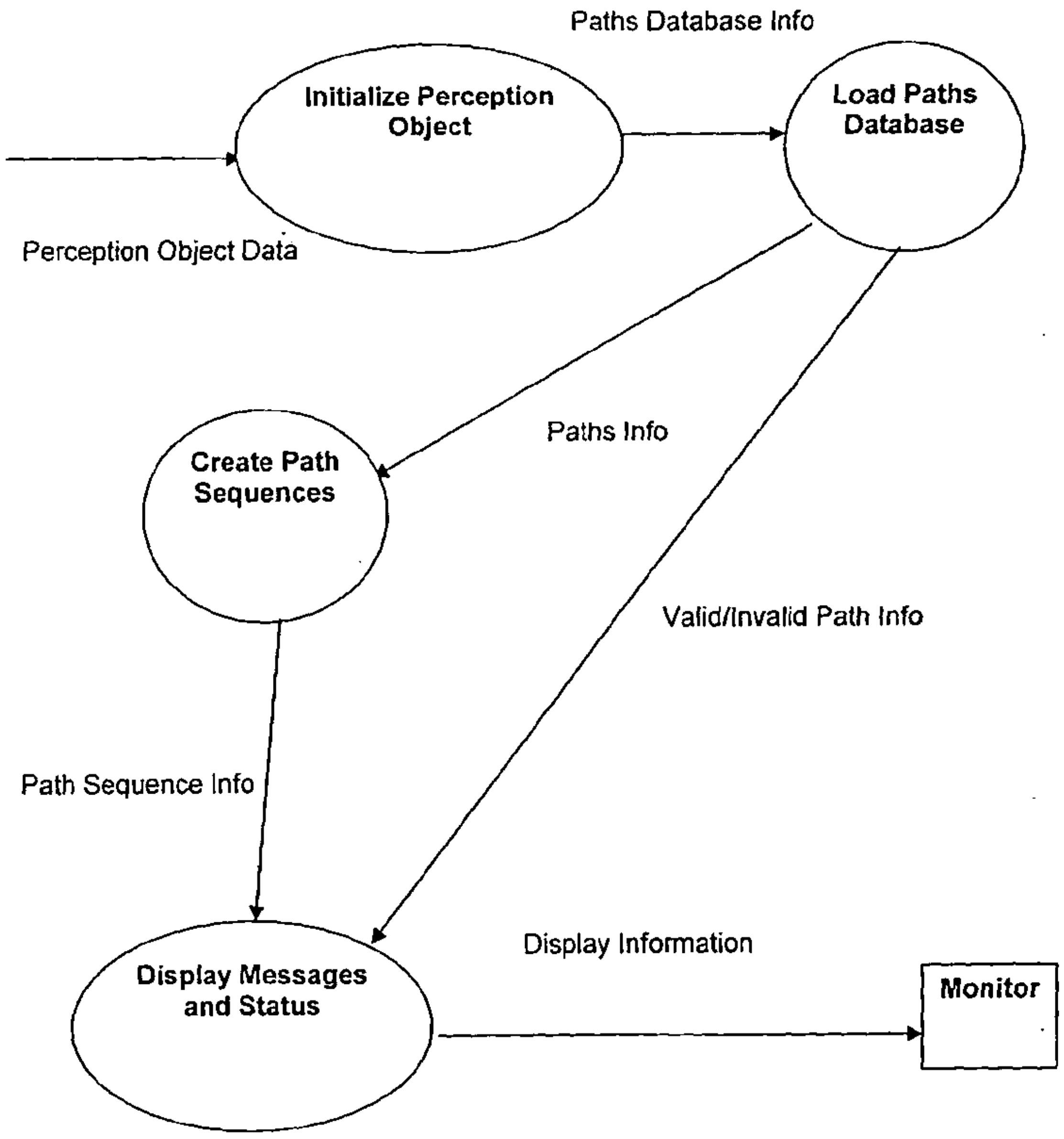


Figure 2-9: Level 3 DFD for Process Create Perception Object

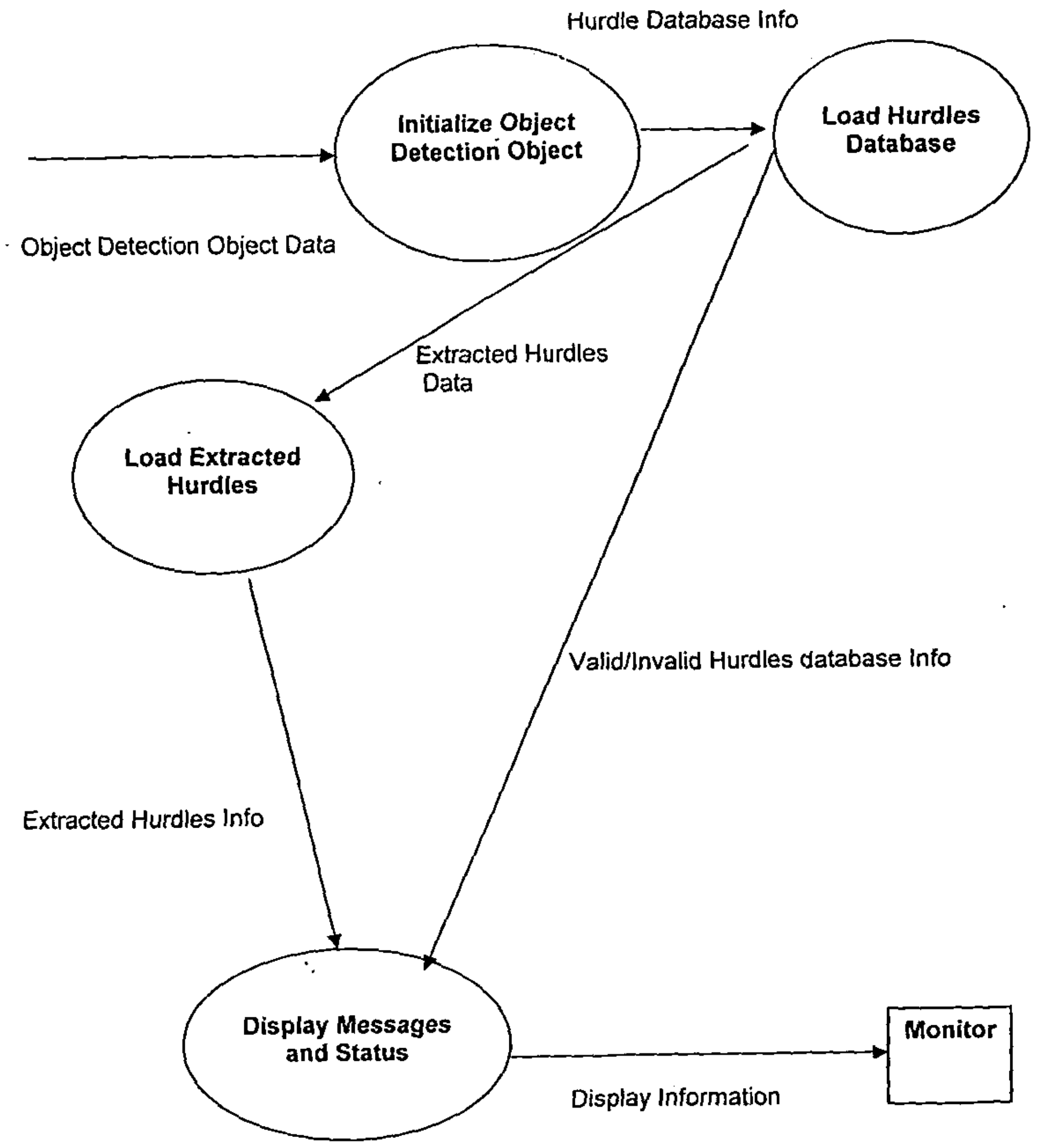


Figure 2-10: Level 3 DFD for Process Create Object Detection Object

2.2.4 Process Specification (PSPEC)

The Process specification contains the detail information about the processes defined in the DFD. These details either can be in simple English or in Program Design Language (PDL) format. In simple English, the process is defined in simple words while in PDL, the process is written in the format similar to the algorithms but they are not complex as algorithms are. We will define the process in PDL.

2.2.4.1 Scan Image for Hurdle

```
Procedure Scan Image for Hurdle;

    Get the image data;

    Scan image data to find any disparity larger than pre-specified values in
    dimensions;

    If disparity larger than pre-specified values in dimensions;

      Then begin

        Classify disparity as hurdle;
        Create an object of class Hurdle;
        Assign properties of disparity to the created hurdle object;
        Display information of hurdle on screen;

      End;

    Else begin

      Classify disparity as noise;
      Replace the disparity color to its original color;

    End if;

End proc
```

2.2.4.2 Match Sequence

```
Procedure Match Sequence

    Get the sequence information of current path;

    Compare sequence information of current path with sequence information
    of previous paths in database;

    If match exists
        Display information on screen regarding path existence;

    Else begin

        Compare sequence information of current path with reversed
        sequence information of previous paths in database;

        If match exists
            Display information on screen regarding path existence;

        End if

End proc
```

2.2.5 State Transition Diagram (STD)

The State Transition Diagram indicates how the system behaves as consequence of external events. The labeled transition arrows indicate how the system reacts to the external events as it traverses the defined states. By studying STD, a software engineer can determine the behavior of the system and can ascertain whether there are "holes" in the specified behavior.

Figure 2-11 shows the State Transition Diagram for the software Vision 1 - *Perception Based Obstacle Detection and Avoidance System for Autonomous Vehicles with Self-Localization Capability.*

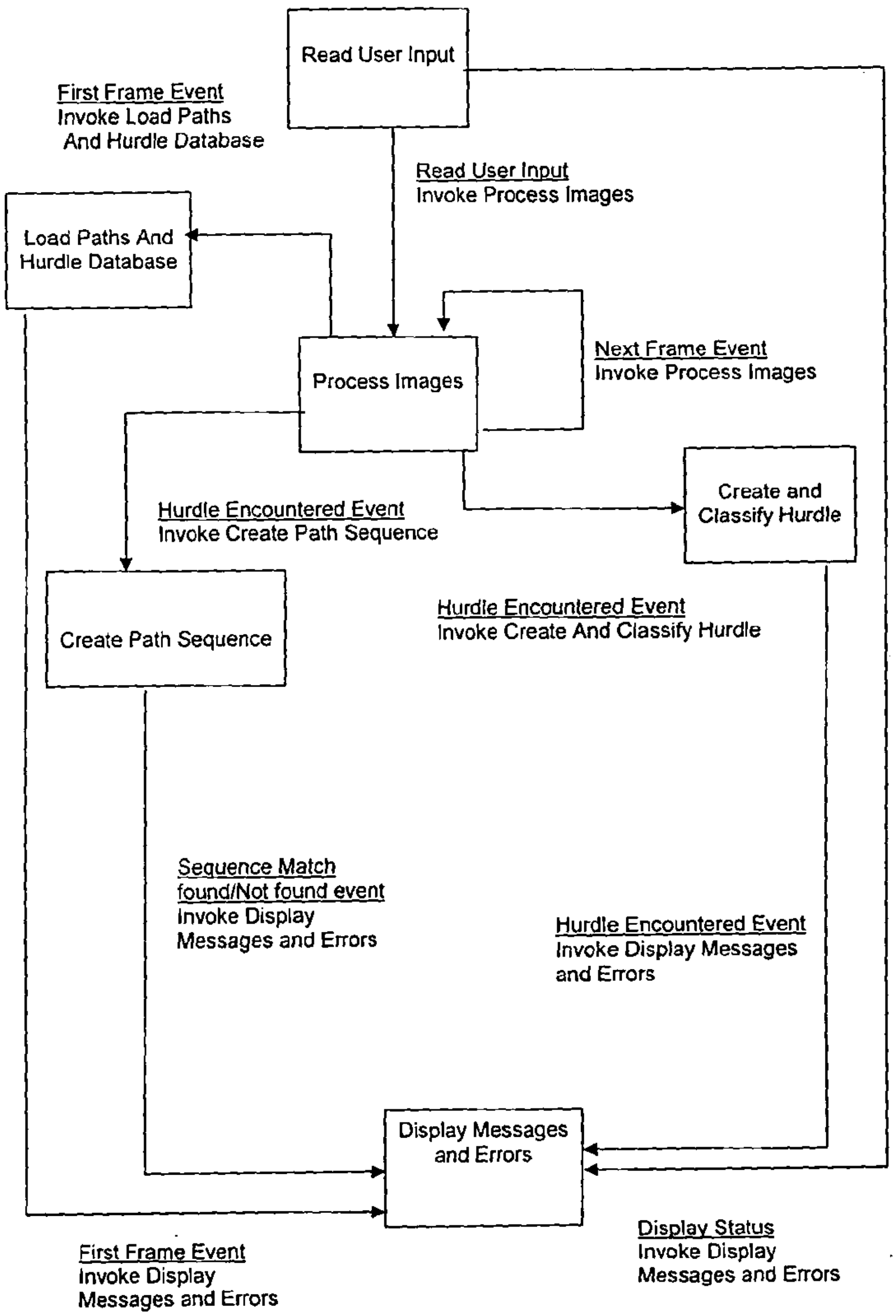


Fig 2-11: The State Transition Diagram for the software Vision One

2.2.6 Data Dictionary

The analysis model encompasses representations of data objects, function and control. In each representation, data objects and/or control items play a role. Therefore, it is necessary to provide and organized approach for representing the characteristics of each data objects and control item. This is accomplished with the data dictionary.

The Data Dictionary is an organized listing of all data elements that are pertinent to the system, with precise, rigorous definitions so that both user and system analyst will have a common understanding of inputs, outputs, components of store and intermediate calculations.

Name:	ip
Aliases:	None
Where used/how used:	Object of class ImageProcessor. Used for processing of images of video stream.
Description:	Only a single instance of this object is created, it virtually acts as a master mind, calling and controlling all the other objects.
Name:	pOD
Aliases:	None
Where used/how used:	Used in Image processing class
Description:	ObjectDetection class object. Only a single instance of this object is created. It is used to detect different hurdles.
Name:	pNav
Aliases:	None
Where used/how used:	Pointer to the object of class Navigation. Used in ImageProcessor class.
Description:	Only a single instance of this object is created. It is used for navigation of vehicle on a path and around different hurdles.
Name:	pPrep

Aliases:	None
Where used/how used:	In ImageProcessor class.
Description:	Pointer to the object of class Perception. Only a single instance of this object is created. It is used for perception purposes.
Name:	pP1
Aliases:	None
Where used/how used:	Used in ImageProcessor class.
Description:	Pointer to the object of class Path. Only a single instance of this object is created. This object is kept for adding extra features to the project. Currently, its contribution to project is almost non, although it contains full fledge functional capability.
Name:	wid
Aliases:	None
Where used/how used:	In ImageProcessor class.
Description:	It is a three dimensional integer array. It is used for storing dimensions of possible hurdle dimensions. Its use through out the project is very extensive.
Name:	wi
Aliases:	None
Where used/how used:	In ImageProcessor class.
Description:	Used as an index into wid array.
Name:	Lpt1
Aliases:	None
Where used/how used:	In ImageProcessor class.

Description:	Used as bottom left starting point for second pass of ground detection filter.
Name:	Lpt2
Aliases:	None
Where used/how used:	In ImageProcessor class.
Description:	Used as top left ending point for second pass of ground detection filter.
Name:	Rpt1
Aliases:	None
Where used/how used:	In ImageProcessor class.
Description:	Used as bottom right starting point for second pass of ground detection filter.
Name:	Rpt2
Aliases:	None
Where used/how used:	In ImageProcessor class.
Description:	Used as top right starting point for second pass of ground detection filter.

2.3 Object-Oriented Analysis

It is a method of analysis that examines the requirements of end-user from the perspective of objects and classes found in the vocabulary of problem domain.

2.4 A Unified Approach to Object-Oriented Analysis

Over the past decade, Grady Booch, James Rumbaugh, and Ivar Jacobson have collaborated to combine the best features of their individual object-oriented analysis and design methods into a unified method. The result, called the *Unified Modeling Language* (UML), has become widely used throughout the industry.

The following views are presented in UML:

- **User Model View.** This view represents the system (product) from the user's (called *actors* in UML) perspective.
- **Structural Model View.** Data and functionality are viewed from inside the system. That is, static structure (classes, objects, and relationships) is modeled.
- **Behavioral Model View.** This part of the analysis model represents the dynamic or behavioral aspects of the system. It also depicts the interactions or collaborations between various structural elements described in the user model and structural model views.
- **Implementation Model View.** The structural and behavioral aspects of the system are represented as they are to be built.
- **Environment Model View.** The structural and behavior aspects of the environment in which the system is to be implemented are represented

2.5 Domain Analysis

This activity, called *domain analysis*, is performed when an organization wants to create a library of reusable classes (components) that will be broadly applicable to an entire category of applications.

2.5.1 Reuse and Domain Analysis

Object-technologies are leveraged through reuse. The benefits derived from reuse are consistency and familiarity. Patterns within the software will become more consistent, leading to better maintainability. Be certain to establish a set of reuse "design rules" so that these benefits are achieved.

2.5.2 The Domain Analysis Process

The goal of domain analysis is straightforward: to find or create those classes that are broadly applicable, so that they may be *reused*.

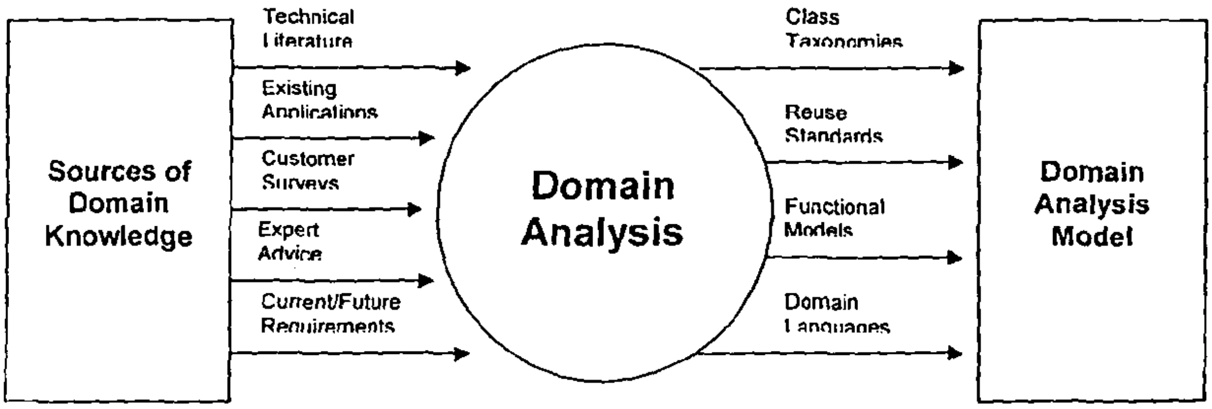


Figure 2-12: Input and Output for Domain Analysis.

Figure 2-12 illustrates key inputs and outputs for the domain analysis process. Sources of domain knowledge are surveyed in an attempt to identify objects that can be reused across the domain. In essence domain analysis is quite similar to knowledge engineering. The knowledge engineer investigates a specific area of interest in an attempt to extract key facts that may be of use in creating an expert system of artificial neural network. During domain analysis, *object* (and class) *extraction* occurs.

2.6 The Object-Oriented Analysis Process

The OOA process doesn't begin with a concern for objects. Rather, it begins with an understanding of the manner in which the system will be used—by people, if the system is human-interactive; by machines, if the system is involved in process control; or by other programs, if the system coordinates and controls applications. Once the scenario of usage has been defined, the modeling of the software begins.

A series of techniques is used to gather basic customer requirements and then define an analysis model for an object-oriented system.

2.6.1 Use-Cases

Use-cases model the system from the end-user's point of view. Created during requirements elicitation, use-cases should define the functional and operational requirements of the system, provide a clear and unambiguous description of how the end-user and the system interact with one another and provide a basis for validation testing.

2.6.1.1 Use-Cases in the System

A *use-case* is a high level piece of functionality that the system provides.

- **Activate Application**
- **Initialize Dialog**
- **Perform Post Processing**
- **Perform Real - Time Processing**
- **View Gray Scale Image Space**
- **View RGB Image Space**
- **Choose Windows to be displayed**
- **Choose Hurdle Seek Area**
- **Choose Display Messages Option**
- **Choose Hurdle Dimensions**
- ***Load Saved Settings***
- **Save New Settings**
- **Load Paths And Hurdle Database**
- **Process Frames**
- **Display Resultant Images**
- **Display Messages**
- **Initialize Modules**

2.6.1.2 Actors in the System

An actor is anyone or anything that interacts with the system being built. The actor in our system are:

- User

2.6.1.3 Expanded Use-Case Format

- **Use-Case: Activate Application**

  Actors: User

  Subject: Activates the software application.

  Summary: The user gives the software execution command, which sends an activation message to the application; the application responds the activating and performing the initialization.

  Type: Essential, Primary

  Typical Course of Actions:

  1. This Use-Case begins when the user gives the software execution command.
  2. The application responds by activating.
  3. It performs initialization.
  4. Starts Message Loop

  Alternative Courses of Actions:

  6a. Application initialization failed, Exit the application.

- **Use-Case: Initialize Dialog**

  Actors: None

  Subject: Initializes the Dialog.

  Summary: This use-case is used by the Activate Application use-case. It responds by performing the initialization, and display steps afterwards.

  Type: Essential, Primary

Typical Course of Actions:

1. This Use-Case begins when Activate Application use-case uses this use case.
2. It responds by dialog initialization.
3. Loads Saved Settings.
4. Initialize Lists.
5. Create Controls and Icons.

Alternative Courses of Actions:

5a. Dialog initialization failed, EXIT the application.

- **Use-Case: Perform Post Processing**

Actors: User

Subject: Performs Post Processing services

Summary: The use-case is initiated by the user selecting the post processing option. It performs post processing function.

Type: Essential, Primary

Typical Course of Actions:

1. This Use-Case begins when the user selects Post-Processing option
2. The Use-Case responds by displaying "Choose .avi file" dialog box
3. The user selects the .avi file
4. The Use-Case sends the specified .avi file information to process frames Use-Case

Alternative Courses of Actions:

3a. The user does not select any .avi file, close "Choose .avi file" dialog box

3b. The user selects an invalid .avi file, close "Choose .avi file" dialog box and display "Invalid .avi file" message

- **Use-Case: Perform Real-Time Processing**

  Actors: User

  Subject: Performs Real - Time Processing services

  Summary: The use-case is initiated by the user selecting the Real-Time processing option. It performs Real-Time processing function.

  Type: Essential, Primary

  Typical Course of Actions:

  1. This Use-Case begins when the user selects Real-Time Processing option
  2. The Use-Case responds by searching for available camera
  3. The camera is initialized
  4. The Use-Case sends the specified camera information to process frames Use-Case

  Alternative Courses of Actions:

  2a. There is no camera connected, display "Camera not found" message and Exit

  2b. There are more than one camera connected to the system, choose the first available camera

  3a. Camera does not respond, display "Camera not responding" message and Exit

- **Use-Case: View Gray Scale Image Space**

  Actors: User

  Subject: View the Gray Scale Image Space of a specified image

  Summary: The Use-Case displays gray scale - intensity vs frequency graph

  Type: Essential, Secondary

  Typical Course of Actions:

  1. This Use-Case begins when user selects the "Gray Scale Space" option

2. It responds by displaying "Choose .bmp image" dialog box
3. The user selects a particular .bmp image
4. The Use-Case creates a Gray Scale format of the specified image
5. The Use-Case creates Intensity vs. frequency graph
6. The Use-Case call "Display Resultant Images" to display the graph

Alternative Courses of Actions:

3a. User does not select a .bmp file, display "Invalid File Format" message and Exit

4a. The selected image is already in gray scale, go to step number 5

- **Use-Case: View RGB Image Space**

Actors: User

Subject: View the RGB Image Space of a specified image

Summary: The Use-Case displays RGB - intensity vs frequency graph

Type: *Essential, Secondary*

Typical Course of Actions:

1. This Use-Case begins when user selects the "RGB Space" option
2. It responds by displaying "Choose .bmp image" dialog box
3. The user selects a particular .bmp image
4. The Use-Case inspects RGB channel values of the specified image
5. The Use-Case creates Intensity vs. frequency graph of the image
6. The Use-Case calls "Display Resultant Images" Use-case to display the graph

Alternative Courses of Actions:

3a. User does not select a .bmp file, display "Invalid File Format" message and Exit

- **Use-Case: Choose Windows to be displayed**

  Actors: User

  Subject: Selecting Windows appearing on the screen

  Summary: The Use-Case lets the user select the particular windows he wants to view and those which he does not

  Type: Essential, Secondary

  Typical Course of Actions:

  1. The Use-Case begin when the user selects "Windows to be Displayed Option"
  2. The user selects windows to be displayed
  3. The user selects windows not to be displayed
  4. The Use-Case calls "Save New Settings" to save the user selection

  Alternative Courses of Actions:

  2a. The user makes no changes, Exit

  3a. The user makes no changes, Exit

- **Use-Case: Choose Hurdle Seek Area**

  Actors: User

  Subject: To choose the area in which the system would attempt to locate hurdles

  Summary: This Use-Case lets the user set the area in which the system would attempt to locate hurdles

  Type: Essential, Secondary

  Typical Course of Actions:

  1. The Use-Case begins when the user selects "Hurdle Seek Area" option
  2. The user selects one of the pre-specified set of values
  3. The Use-Case calls "Save New Settings" to save the user selection

Alternative Courses of Actions:

2a. The user makes no changes, Exit

- **Use-Case: Choose Display Messages Option**

Actors: User

Subject: Selecting whether or not user wants to view important decision oriented messages generated during execution

Summary: This Use-Case lets the user select whether or not he wants to view important decision oriented messages generated during execution

Type: Essential, Secondary

Typical Course of Actions:

1. The Use-Case begins when the user selects "Display Messages" option
2. The user selects/deselects messages to be displayed
3. The Use-Case calls "Save New Settings" to save the user selection

Alternative Courses of Actions:

2a. The user makes no changes, Exit

- **Use-Case: Choose Hurdle Dimensions**

Actors: User

Subject: Choosing the dimensions of the hurdle to seek (in pixels)

Summary: This Use-Case lets the user select the dimensions of the hurdle to seek (in pixels)

Type: Essential, Secondary

Typical Course of Actions:

1. The Use-Case begins when the user selects "Hurdle Dimensions" option
2. The user selects one of the pre-specified set of values

3. The Use-Case calls "Save New Settings" to save the user selection

Alternative Courses of Actions:

2a. The user makes no changes, Exit

- **Use-Case: Load Saved Settings**

  Actors: None

  Subject: Load the selections made by the user for appropriate course of action

  Summary: This Use-Case loads the selections made by the user for appropriate course of action to be taken on the basis of the choices made

  Type: Essential, Primary

  Typical Course of Actions:

  1. The Use-Case starts when the user has made all the selections and is ready to proceed
  2. The Use-Case calls Use-Case "Initialize Modules" with the selections made by the user

  Alternative Courses of Actions:

  1a. The user does not make any changes to the settings, call Use-Case "Initialize Modules" with pre-set values

- **Use-Case: Save New Settings**

  Actors: None

  Subject: Save the changes to the settings made by the user

  Summary: This Use-Case saves the changes to the settings made by the user

  Type: Essential, Primary

  Typical Course of Actions:

  1. The Use-Case starts when the user makes a change to the setting and commits
  2. The Use-Case updates and stores changes in the values made by the user

Alternative Courses of Actions:

1a. The user does not make any changes to the settings, call Use-Case "Initialize Modules" with pre-set values

- **Use-Case: Load Paths And Hurdle Database**

Actors: None

Subject: Loads the Database of Paths and Hurdles encountered and stored on the previous runs

Summary: This Use-Case loads paths and hurdle encountered and stored on the previous runs

Type: Essential, Primary

Typical Course of Actions:

1. This Use-Case begins when Use-Case "Initialize Modules" calls it to load previous paths and hurdles

2. The Use-Case loads Paths from the paths database

3. The Use-Case loads hurdles from the hurdle database

Alternative Courses of Actions:

2a. "Current Path" folder is not empty; call Use-Case "Display Messages" with "Please Empty the Current Path Folder" message

2b. The format of Paths database is invalid, call Use-Case "Display Messages" with "Paths Database format Invalid" message

3a. The format of Hurdles database is invalid, call Use-Case "Display Messages" with "Hurdles Database format Invalid" message

- **Use-Case: Process Frames**

Actors: None

Subject: Process the frames from input stream

Summary: Processing the frames of input stream (Real-Time or Post-Processing)

Type: Essential, Primary

Typical Course of Actions:

1. This Use-Case is called by "Initialize Modules" use case
2. The Use-Case retrieves the next frame form the input stream
3. It applies processing on the frame
4. It calls Use-Cases "Display Resultant Images" and "Display Messages" based on the result of processing on that frame

Alternative Courses of Actions:

2a. The frame cannot be retrieved, "Display Messages" with "Frame Cannot be retrieved" message

2b. Previous frame was the last frame, "Display Messages" with "Processing Complete" message

- **Use-Case: Display Resultant Images**

Actors: None

Subject: Displays Resultant Images based on the processing performed

Summary: This Use-Case is used for displaying resultant images based on the processing performed

Type: Essential, Primary

Typical Course of Actions:

1. The Use-Case begins when any other use-case requests it to display a particular image
2. The Use-Case responds by creating a window in which to display the image
3. The Use-Case loads the specified image into that window
4. The Use-Case displays the window with the specified image in it

Alternative Courses of Actions:

2a. Window cannot be created, call "Display Messages" Use-Case with "Cannot create Window" message

3a. The image cannot be loaded in the create window, call "Display Messages" Use-Case with "Cannot load image in the window" message

4a. Window cannot be displayed, call "Display Messages" Use-Case with "Cannot display Window" message

- **Use-Case: Display Messages**

Actors: None

Subject: Displaying messages

Summary: This Use-Case is used for displaying messages generated during execution

Type: Essential, Primary

Typical Course of Actions:

1. The Use-Case begins when any other use-case requests it to display a particular message
2. The Use-Case responds by receiving the sent message
3. The Use-Case creates a message box to display the message
4. The Use-Case displays the message in the message box

Alternative Courses of Actions:

2a. Message is not in proper format; display a message "Improper format of received message"

3a. Message box with specified properties cannot be created, display a simple message with "Message box with specified properties cannot be created"

4a. Message cannot be displayed, display a simple message with "Message cannot be displayed properly"

- **Use-Case: Initialize Modules**

  Actors: None

  Subject: Initialization of all the modules

  Summary: This Use-Case performs initialization of all the modules according to the requirements of the user's choice

  Type: Essential, Primary

  Typical Course of Actions:

  1. The Use-Case begins when the Use-Case "Load Saved Settings" calls it with specification about what course of action to take.

  2. The Use-Case initializes all the internal modules that would be required

  3. The Use-Case calls Use-Case "Process frames"

  Alternative Courses of Actions:

  2a. Module(s) cannot be initialized, call Use-Case "Display Messages" with "Unable to initialize Modules" message and Exit.

2.6.1.4 Use-Cases Diagram

Use-Case diagram in Figure 2-13 shows some of the use-cases in the system, some of the actors in the system, and the relationships between them.

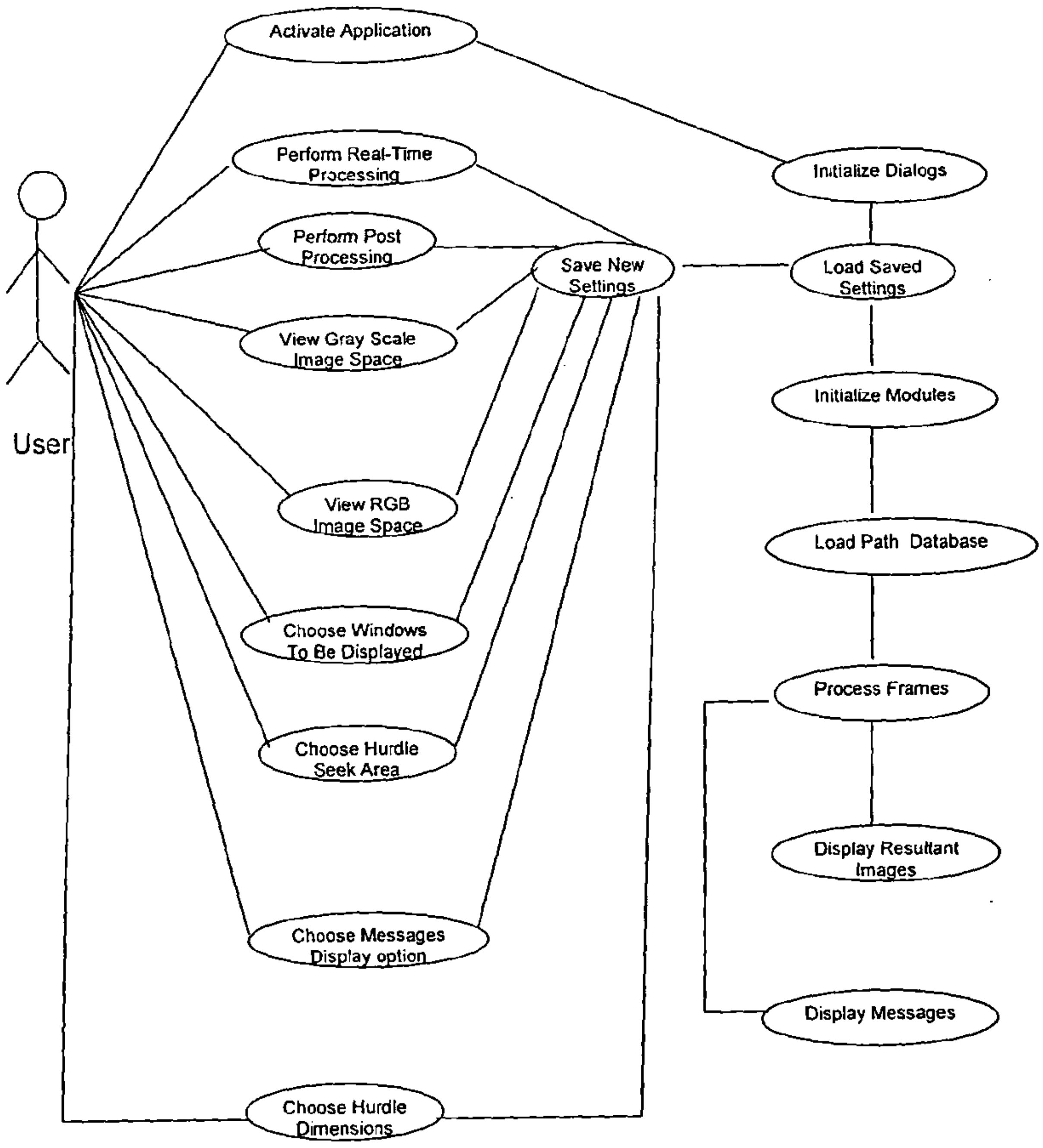


Figure 2-13: Use-Case Diagram for Vision One - *Perception Based Obstacle Detection and Avoidance System for Autonomous Vehicles with Self-Localization Capability*

2.6.2 Conceptual Model

Conceptual Model in Figure 2-14 depicts the concepts found in the domain of the system. In conceptual model we identify the conceptual (as opposed to physical) objects in the application.

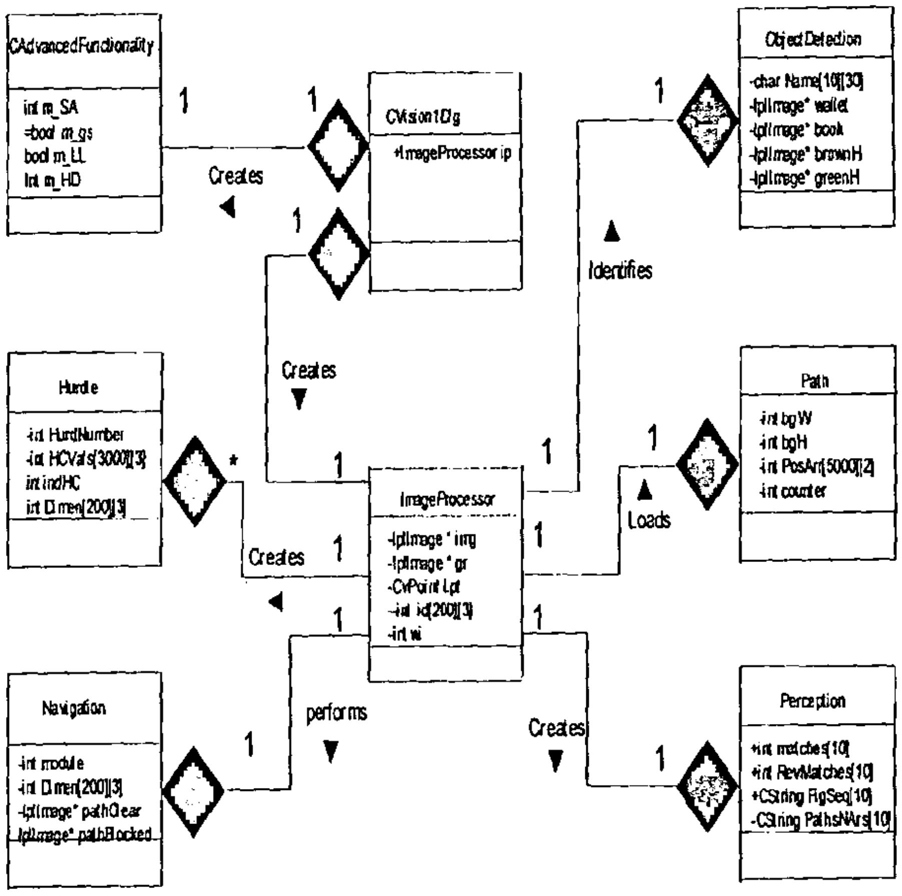


Figure 2-14: Conceptual Model for Vision One - *Perception Based Obstacle Detection and Avoidance System for Autonomous Vehicles with Self-Localization Capability*

2.7 Conclusion

In this chapter we have given detailed description of our analysis phase of the system development. In the beginning of this chapter, we have given an introduction to the basic concepts of analysis and different techniques to carry out analysis phase. Since at the beginning of analysis phase, our approach was structured, which later on evolved into object oriented, we have presented our work in both structured and object oriented approaches to analysis.

Chapter 3
Design

3. DESIGN

Design is an iterative process transforming requirements into a "blueprint" for constructing the software. It is the first step in the development phase for any engineered product or system. It can also be defined as "the process of applying various techniques and principles for the purpose of defining a device, a process or a system in sufficient detail to permit its physical realization."

The designer's goal is to produce a model or representation of an entity that will later be built. The process by which the model is developed combines intuition and judgment based on experience in building similar entities, a set of principles and/or heuristics that guide the way in which the model evolves, a ultimately leads to a final design representation.

3.1 Object-Oriented Design

Object-Oriented Design is a process of object-oriented decomposition and a notation for representing logical and physical as well as static and dynamic models of the system under design.

The four layers of object-oriented design pyramid are:

- **The Subsystem Layer** contains a representation of each of the subsystems that enable the software to achieve its customer-defined requirements and to implement the technical infrastructure that supports customer requirements.
- **The Class and Object Layer** contains the class hierarchies that enable the system to be created using generalizations and increasingly more targeted specializations.
- **The Message Layer** contains the design details that enable each object to communicate with its collaborators.
- **The Responsibilities Layer** contains the data structure and algorithmic design for all attributes and operations for each object.

3.2 Design Patterns

The best engineers in any field have an uncanny ability to see patterns that characterize a problem and corresponding patterns that can be combined to create a solution. Throughout the OOD process, a software engineer should look for every opportunity to reuse existing design patterns (when they meet the needs of the design) rather than creating new ones.

3.2.1 Describing a Design Patterns

All design patterns can be described by specifying the following information:

- The name of the pattern
- The intent of the pattern
- The "design forces" that motivate the pattern
- The solution that mitigates these forces
- The classes that are required to implement the solution
- The responsibilities and collaboration among solution classes
- Guidance that leads to effective implementation
- Example source code or source code templates
- Cross-references to related design patterns

The design pattern name is itself an abstraction that conveys significant meaning once the applicability and intent are understood.

3.2.2 Using Patterns in Design

In an object-oriented system, design patterns can be used by applying two different mechanisms: inheritance and composition. Using *inheritance*, an existing design pattern becomes a template for new subclass. The attributes and operations that exist in the pattern become part of the subclass.

3.3 Object-Oriented Design Process

UML design modeling addresses the structural model, behavioral model, implementation model, and environmental model views.

3.3.1 Structural Model

Data and functionality are viewed from inside the system. That is, static structure (classes, objects, and relationships) is modeled.

3.3.1.1 What is a Class?

A *class* is something that encapsulates *information* and behavior. We take a little bit of information and a little bit of behavior, and encapsulate them into something called a class.

3.3.1.2 Finding a Class

A good place to start when finding classes is the flow of events for the use-cases. Looking at the nouns in the flow of events will let us know what some of the classes are. When looking at the nouns, they will be one of four things:

- An actor
- A class
- An attribute of a class
- An expression that is not an actor , class, or attribute

By filtering out all of the nouns except for the classes, we have found classes identified for our system.

3.3.1.3 Class Diagram

Since the project "*Perception Based Obstacle Detection and Avoidance System for Autonomous Vehicles with Self-Localization Capability*" has quite a number of classes, each having a number of members, the class diagram is represented first by a diagram showing the relationship between different classes in the system, while the later class diagrams show each class with detailed listing of its members.

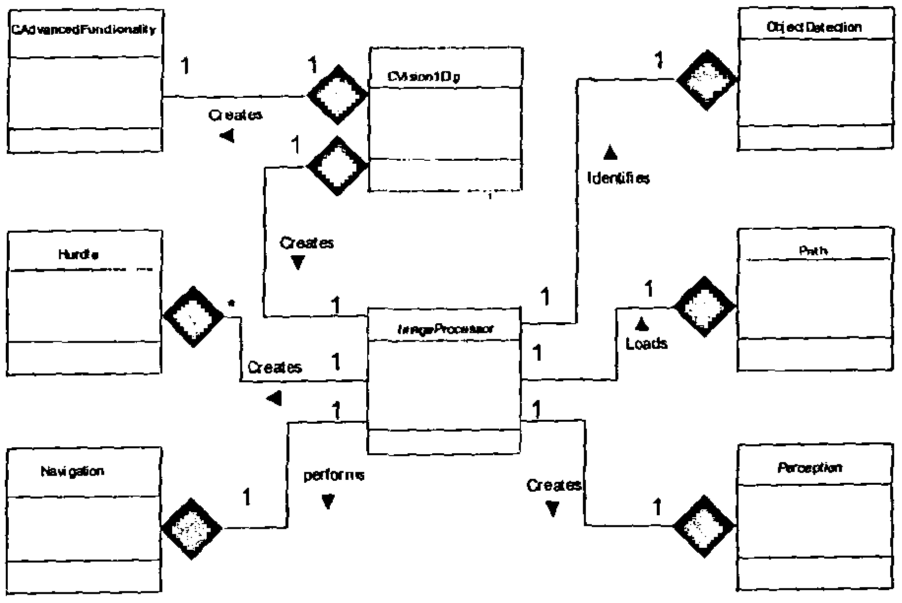
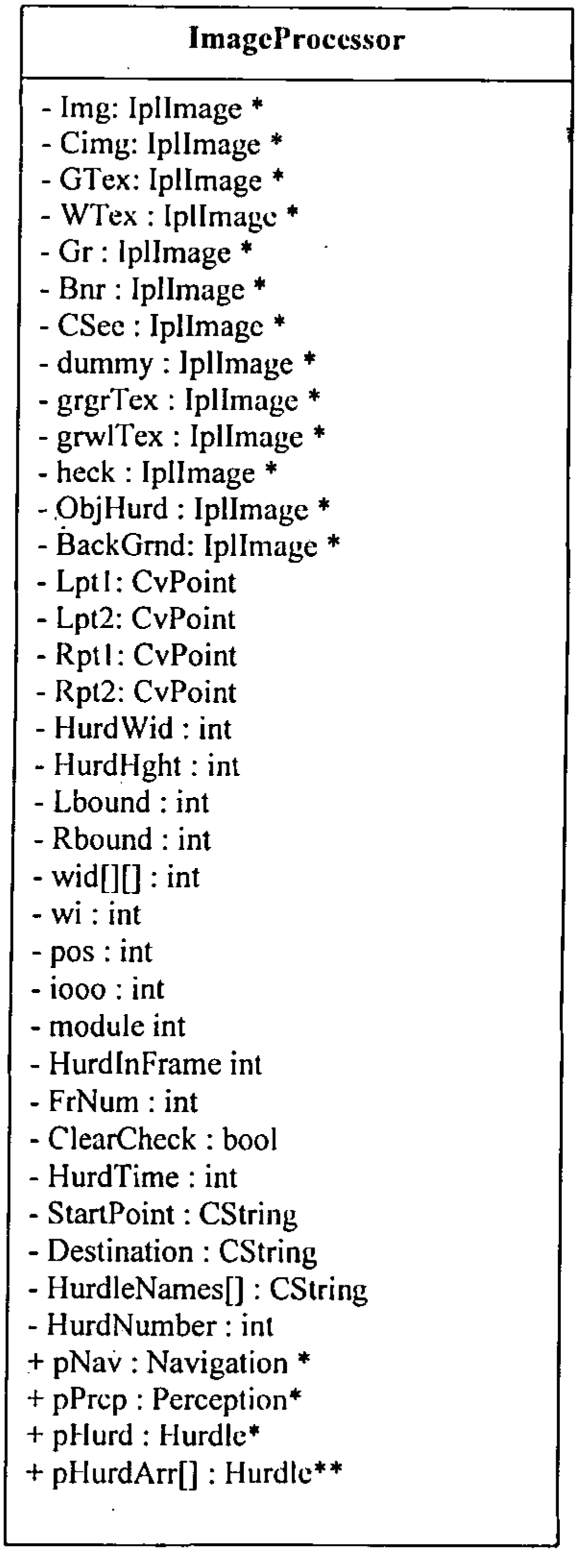


Fig 3-1: Class Diagram showing different classes and their inter-relationship for the project "*Perception Based Obstacle Detection and Avoidance System for Autonomous Vehicles with Self-Localization Capability*"



ImageProcessor

- Img: IplImage *
- Cimg: IplImage *
- GTex: IplImage *
- WTex : IplImage *
- Gr : IplImage *
- Bnr : IplImage *
- CSec : IplImage *
- dummy : IplImage *
- grgrTex : IplImage *
- grwlTex : IplImage *
- heck : IplImage *
- ObjHurd : IplImage *
- BackGrnd: IplImage *
- Lpt1: CvPoint
- Lpt2: CvPoint
- Rpt1: CvPoint
- Rpt2: CvPoint
- HurdWid : int
- HurdHght : int
- Lbound : int
- Rbound : int
- wid[][] : int
- wi : int
- pos : int
- iooo : int
- module int
- HurdInFrame int
- FrNum : int
- ClearCheck : bool
- HurdTime : int
- StartPoint : CString
- Destination : CString
- HurdleNames[] : CString
- HurdNumber : int

+ pNav : Navigation *
+ pPrcp : Perception*
+ pHurd : Hurdle*
+ pHurdArr[] : Hurdle**

```
+ indHurd : int
+ pP1 : Path*
+ pOD : ObjectDetection*
+ im_gs : bool
+ im_LL : bool
+ im_win1 : bool
+ im_win2 : bool
+ im_AIS : bool
+ im_SA : int
+ im_HD : int
+ im_ImgSp : int
```

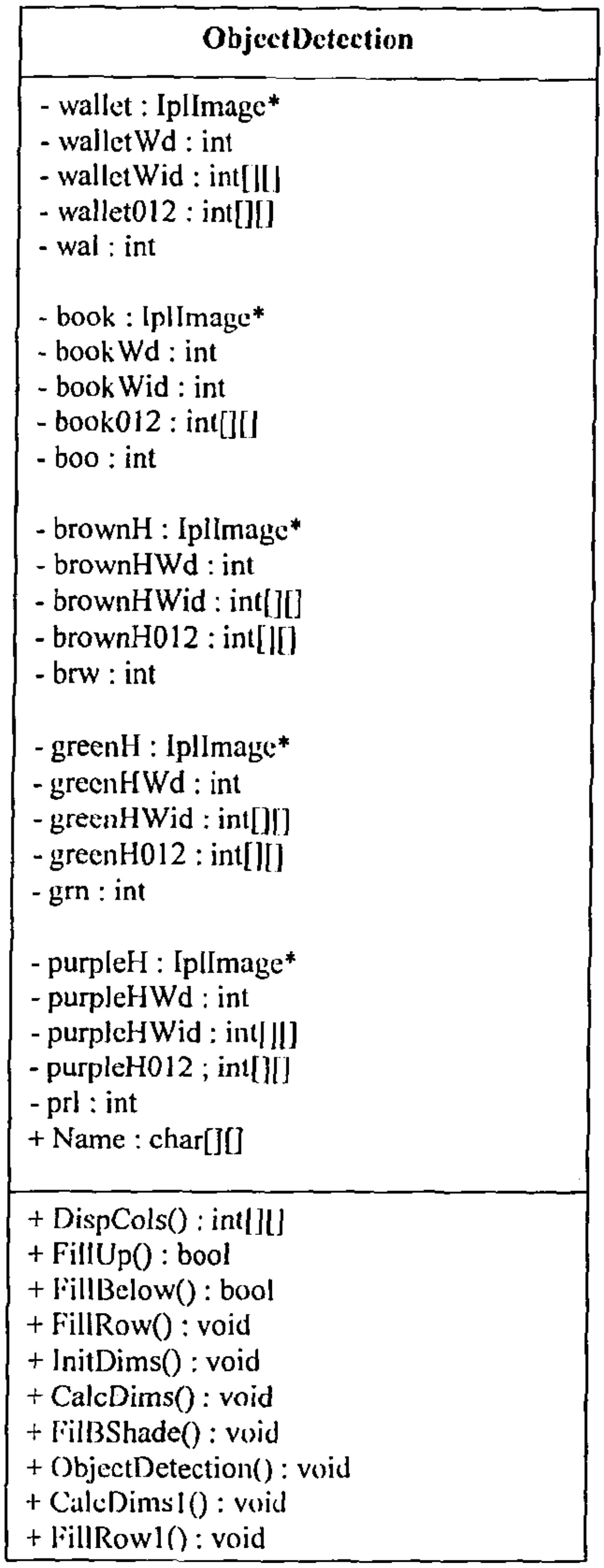
```
- SetUpThings() : int
- SearchItUp() : void
- MatchItPal() : IplImage*
- FillChs() : void
- DoMore( ): void
- it() : void
- CopletePic() : void
- QuickSort() : void
- SortIt1() : void
- SepChans() : void
- FindRope() : void
- LoadnProcessOrigImg() : void
- AllProcessing() : void
- TexCompareGround() : void
- TexCompareWall() : void
- LiveCapture() : void
-  Display() : void
- DrawPath() : void
- Filling() : bool
- MyFloodFill() : void
- MasterMind(): void
- SetBounds(int) : void
- SearchUp() : bool
- InitWid() : void
- fillUp() : void
- InocGuilty() : bool;
- AddIt() : void
```

- SeeDummy();int) : void - DestroyEm() : void - SetHurdDims(): void - Eper(): void - HurdSearchUp() : bool - HurdfillUp() : void - UpsideDownChk() : void - InUDC(): void - DisplayForCam(): void - void InitEvForCam() : void - InitOnEvFrame() : void

Fig 3-2: Class Diagram for Class ImageProcessor

Path
- bgW : int - bgH : int - counter : int - BGI : IplImage* - pos : CvPoint - StartPoint : CvPoint - EndingPoint : CvPoint - FrNum : long int - PosArr[][] : int[][] - indPA : int
+ SaveIt() : void + SetPosArr() : void + ByForceInit() : void + ShowAgain() : void + DrawThePath() : void + CreatenFill() : void + Path() : void

Fig 3-3: Class Diagram for Class Path



ObjectDetection

- wallet : IplImage*
- walletWd : int
- walletWid : int[][]
- wallet012 : int[][]
- wal : int

- book : IplImage*
- bookWd : int
- bookWid : int
- book012 : int[][]
- boo : int

- brownH : IplImage*
- brownHWd : int
- brownHWid : int[][]
- brownH012 : int[][]
- brw : int

- greenH : IplImage*
- greenHWd : int
- greenHWid : int[][]
- greenH012 : int[][]
- grn : int

- purpleH : IplImage*
- purpleHWd : int
- purpleHWid : int[][]
- purpleH012 ; int[][]
- prl : int

+ Name : char[][]

+ DispCols() : int[][]
+ FillUp() : bool
+ FillBelow() : bool
+ FillRow() : void
+ InitDims() : void
+ CalcDims() : void
+ FillBShade() : void
+ ObjectDetection() : void
+ CalcDims1() : void
+ FillRow1() : void

Fig 3-4: Class Diagram for Class ObjectDetection

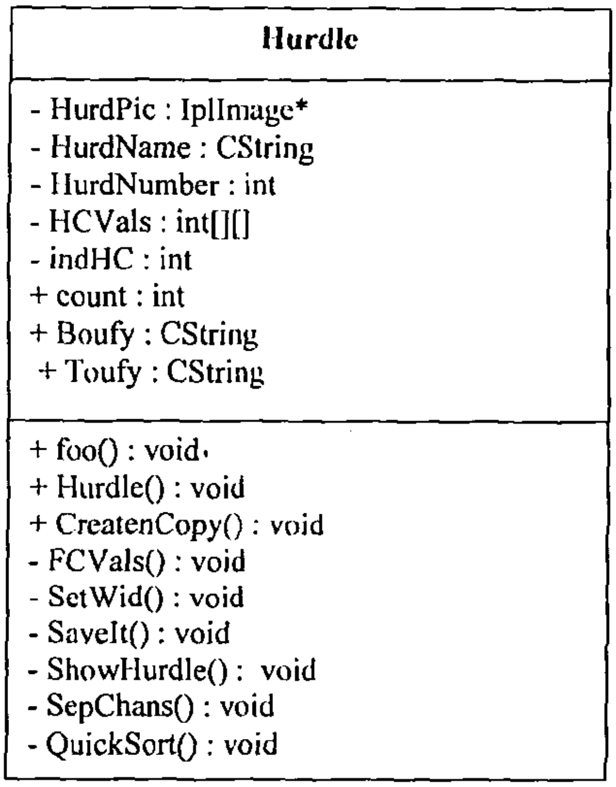
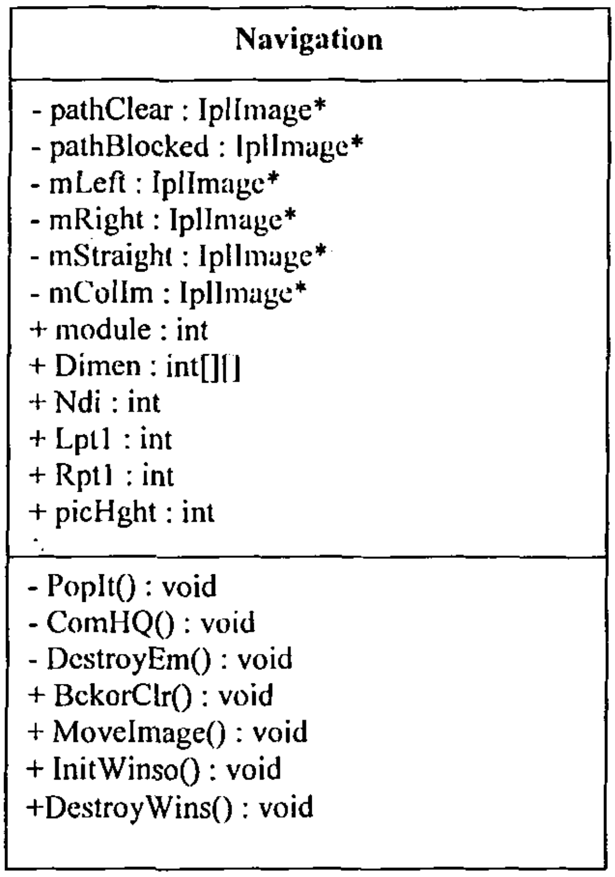


Fig 3-5: Class Diagram for Class Hurdle



```
Navigation
----------------------------
- pathClear : IplImage*
- pathBlocked : IplImage*
- mLeft : IplImage*
- mRight : IplImage*
- mStraight : IplImage*
- mColIm : IplImage*
+ module : int
+ Dimen : int[][]
+ Ndi : int
+ Lpt1 : int
+ Rpt1 : int
+ picHght : int
----------------------------
- PopIt() : void
- ComHQ() : void
- DestroyEm() : void
+ BckorClr() : void
+ MoveImage() : void
+ InitWinso() : void
+DestroyWins() : void
```

Fig 3-6: Class Diagram for Class Navigation

3.4 Object Diagram

An Object diagram captures the instances and links of the system. It is built during analysis and design phase. Object diagram illustrates data/object structures.

Following are different *Object diagrams* identified in the project "*Perception Based Obstacle Detection and Avoidance System for Autonomous Vehicles with Self-Localization Capability*".

Since the project "*Perception Based Obstacle Detection and Avoidance System for Autonomous Vehicles with Self-Localization Capability*" has quite a number of classes, each having a number of members, the object diagram in some cases can not be represented on one page, hence the diagram is extended from one page to the next with a "**continued...**" at the end of the page representing that this diagram is a continuation of the diagram at the respective previous page.

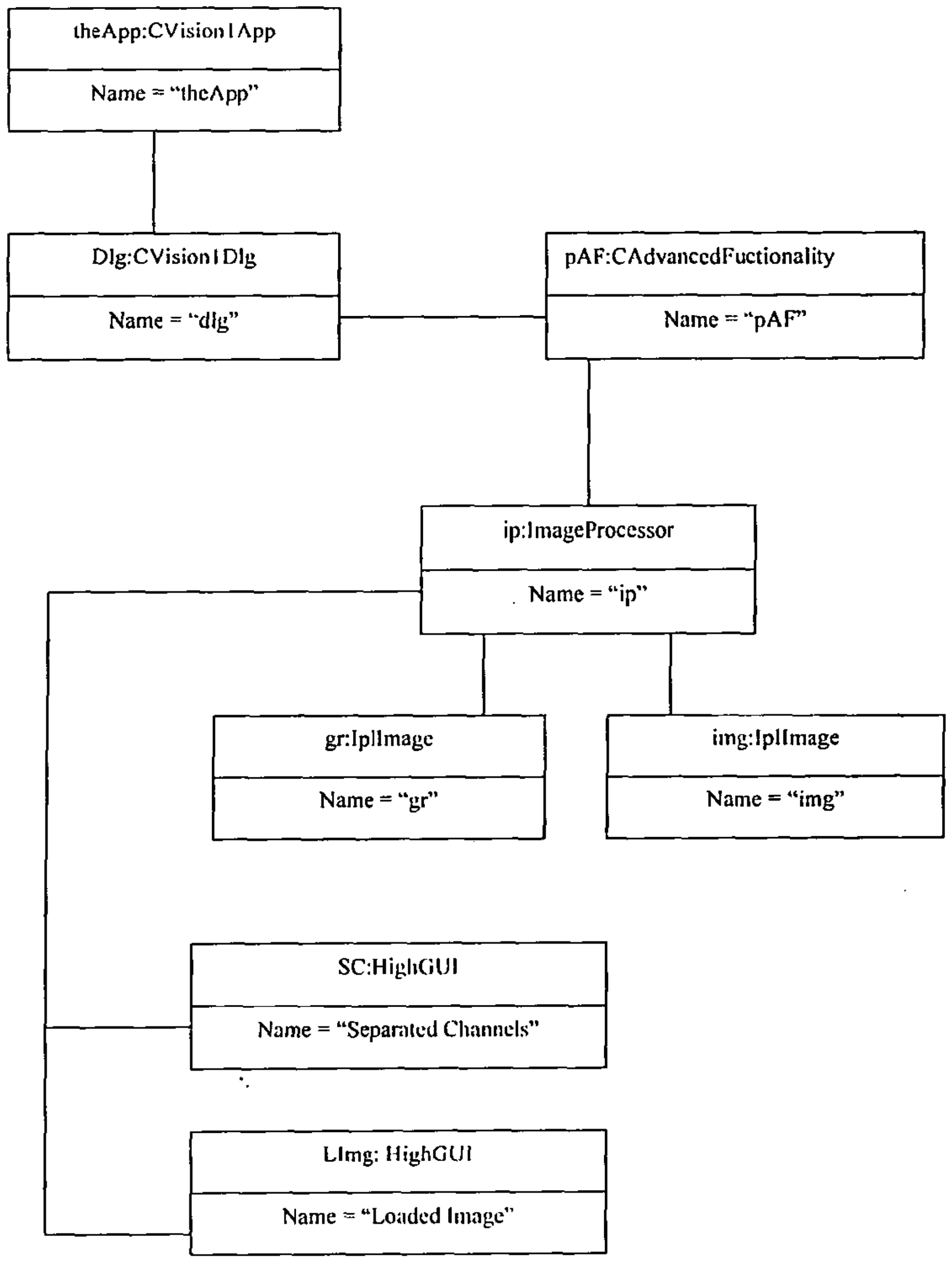
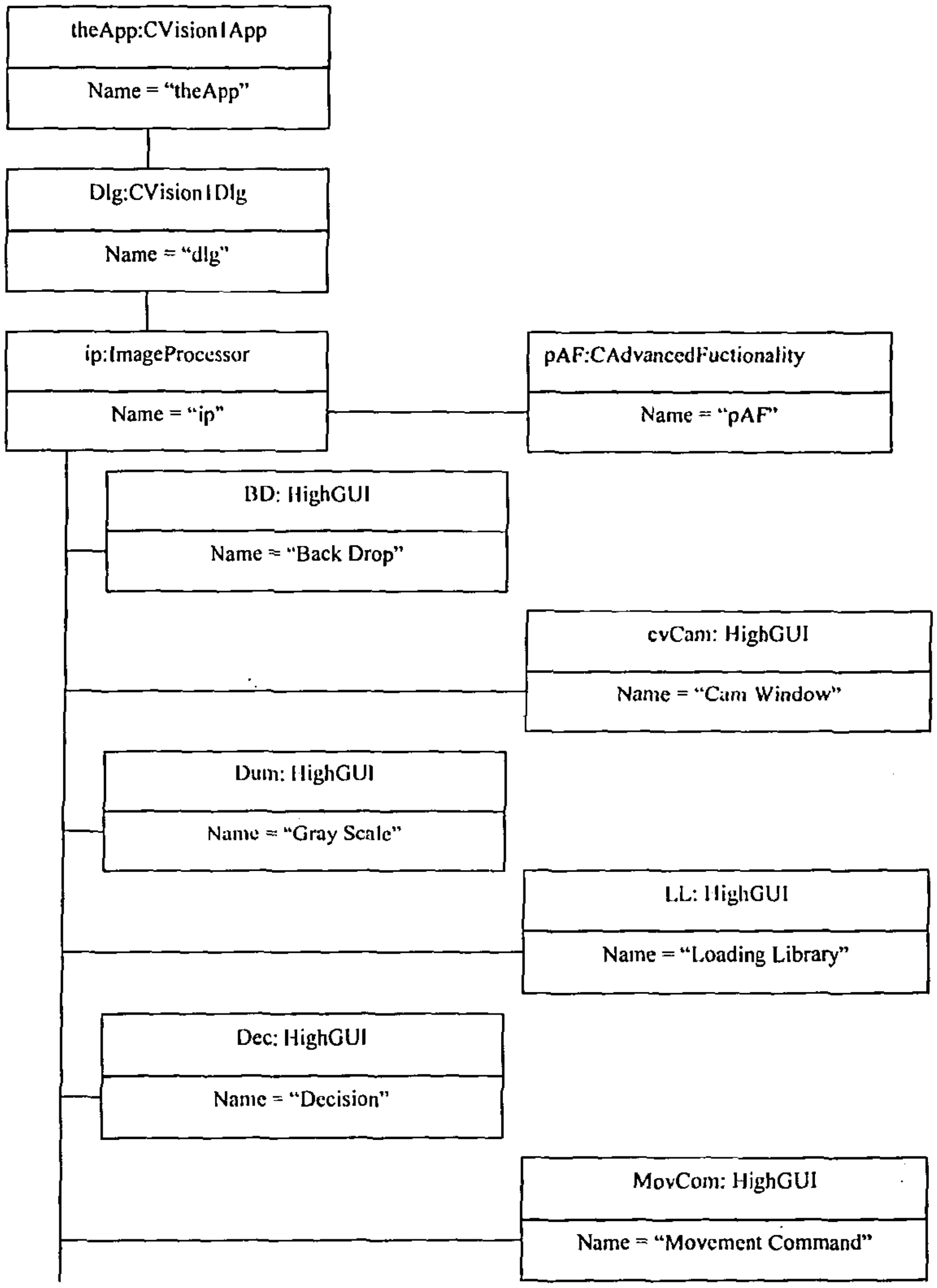
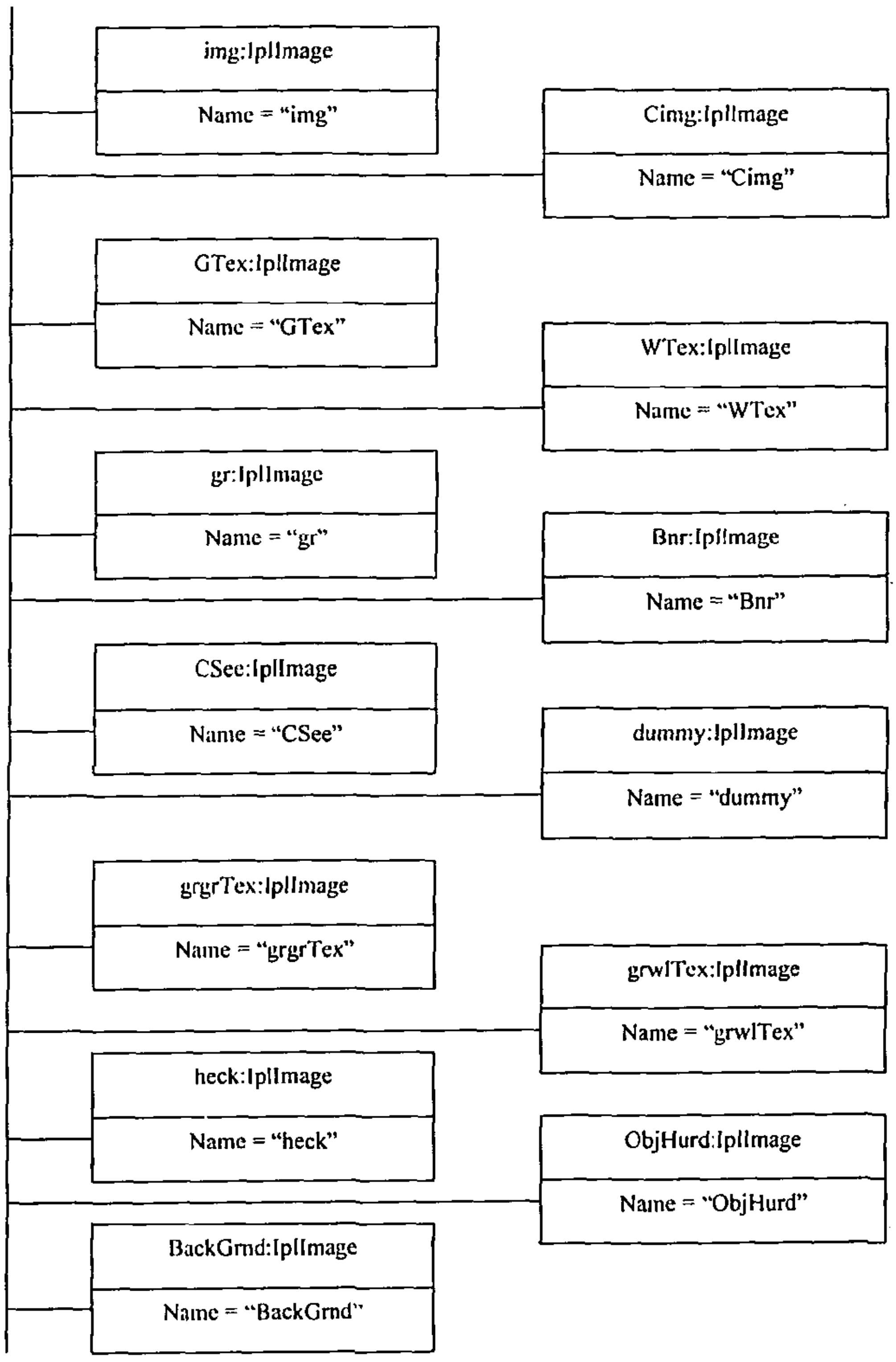


Fig 3-7: Object Diagram for View Channels


theApp:CVision1App
Name = "theApp"
Dlg:CVision1Dlg
Name = "dlg"
ip:ImageProcessor
Name = "ip"
pAF:CAdvancedFuctionality
Name = "pAF"
BD: HighGUI
Name = "Back Drop"
cvCam: HighGUI
Name = "Cam Window"
Dum: HighGUI
Name = "Gray Scale"
LL: HighGUI
Name = "Loading Library"
Dec: HighGUI
Name = "Decision"
MovCom: HighGUI
Name = "Movement Command"

Continued...


img:IplImage
Name = "img"
Cimg:IplImage
Name = "Cimg"
GTex:IplImage
Name = "GTex"
WTex:IplImage
Name = "WTex"
gr:IplImage
Name = "gr"
Bnr:IplImage
Name = "Bnr"
CSee:IplImage
Name = "CSee"
dummy:IplImage
Name = "dummy"
grgrTex:IplImage
Name = "grgrTex"
grwlTex:IplImage
Name = "grwlTex"
heck:IplImage
Name = "heck"
ObjHurd:IplImage
Name = "ObjHurd"
BackGrnd:IplImage
Name = "BackGrnd"

Continued...

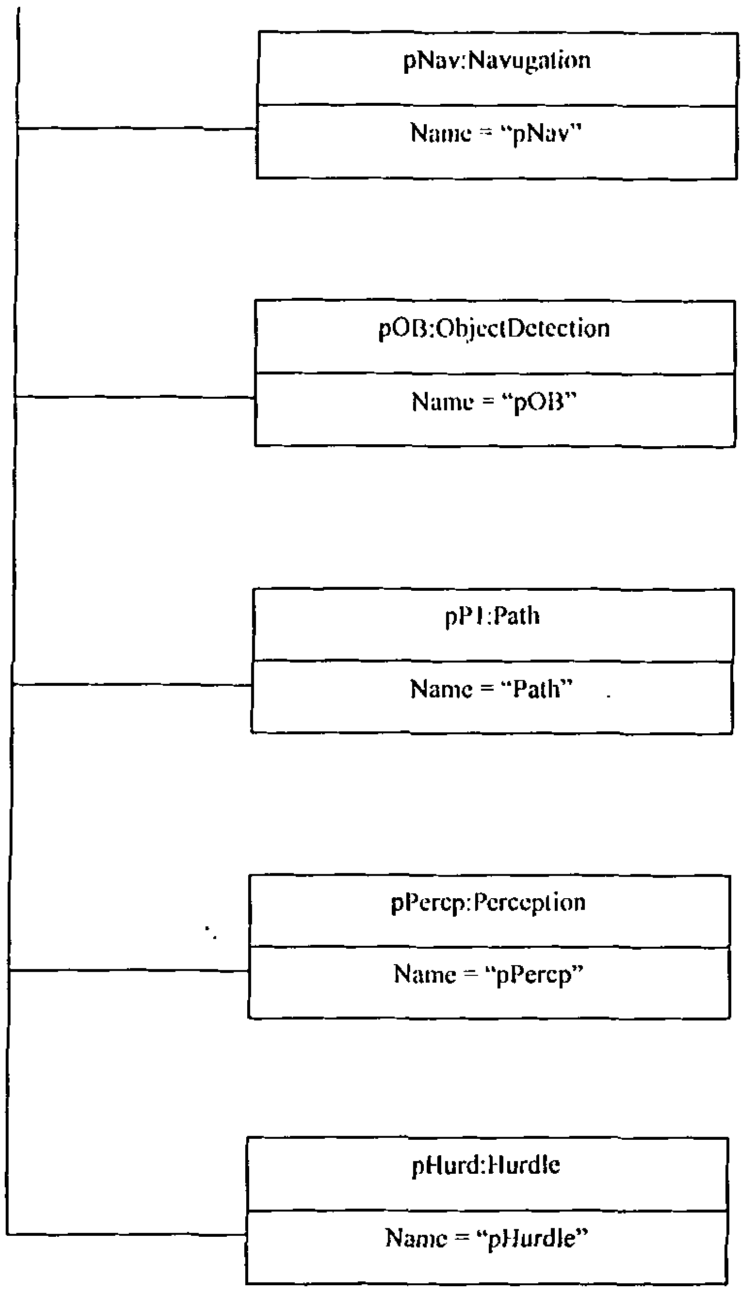


Fig 3-8: Object Diagram for Real Time processing

3.5 Behavioral Model

This part of the analysis model represents the dynamic or behavioral aspects of the system. It also depicts the interactions or collaborations between various structural elements described in the user model and structural model views.

3.5.1 Interaction Diagrams

An *Interaction diagram* shows, step-by-step, one of the flows through a use-case. There are two types of Interaction diagrams:

- **Sequence Diagram** represents dynamic behavior which time oriented. It can show the focus of control.
- **Collaboration Diagram** represents dynamic behavior which message oriented. It can show the data flow.

3.5.1.1 Sequence Diagram

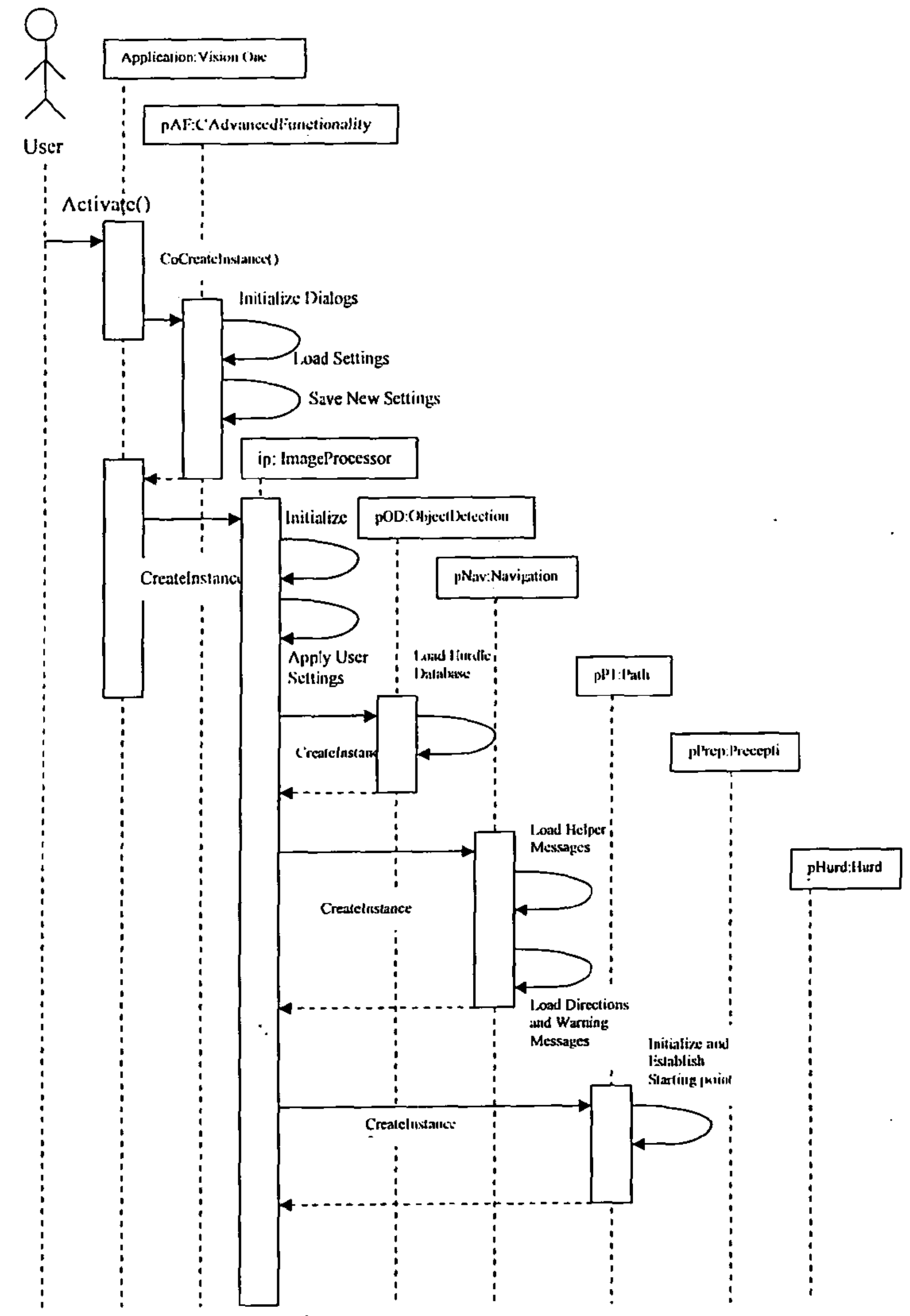
A Sequence diagram shown in Figure 3-9 is an interaction diagram, which is ordered by time; it is read form the top to the bottom.

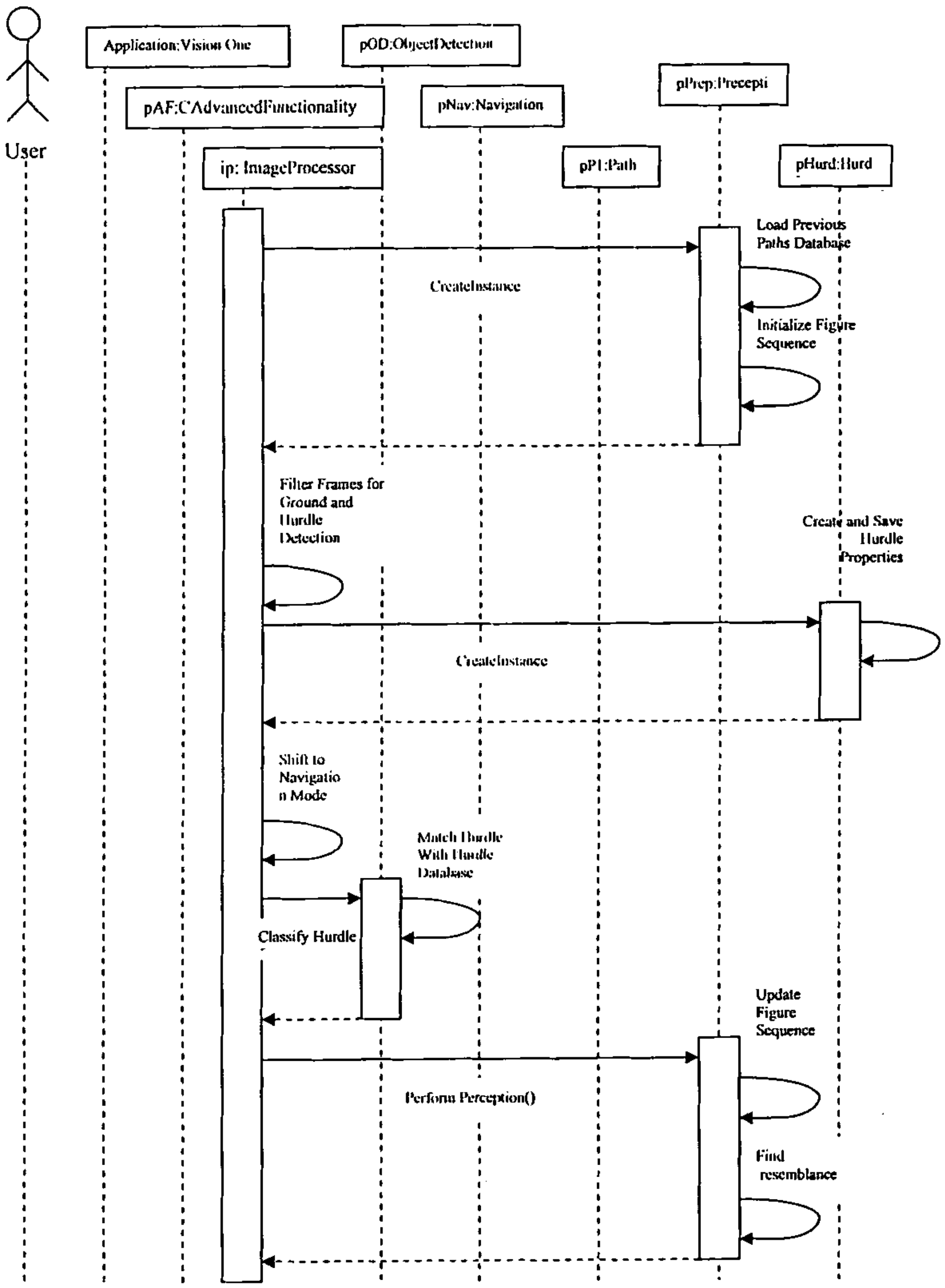
We can read this diagram by looking at the objects and messages. The objects that participate in the flow are shown in rectangles across the top of the diagram.

The actor objects, involved in the use-case are also shown in the diagram.

Each object has a *lifeline*, drawn as a vertical dashed line below the object. A message is drawn between the lifelines of two objects to show that the objects communicate. Each message represents one object making a function call of another.

Messages can also be reflexive, showing that an object is calling one of its own operations.


Application:Vision One
pAF:CAdvancedFunctionality
User
Activate()
CoCreateInstance()
Initialize Dialogs
Load Settings
Save New Settings
ip: ImageProcessor
Initialize
pOD:ObjectDetection
CreateInstance
pNav:Navigation
Apply User Settings
Load Hurdle Database
pPl:Path
CreateInstance
pPrep:Preceptl
Load Helper Messages
pHurd:Hurd
CreateInstance
Load Directions and Warning Messages
Initialize and Establish Starting point
CreateInstance


User
Application:Vision One
pOD:ObjectDetection
pAF:CAdvancedFunctionality
pNav:Navigation
pPrep:Precepti
ip: ImageProcessor
pPI:Path
pHurd:Hurd
Load Previous Paths Database
CreateInstance
Initialize Figure Sequence
Filter Frames for Ground and Hurdle Detection
Create and Save Hurdle Properties
CreateInstance
Shift to Navigation Mode
Match Hurdle With Hurdle Database
Classify Hurdle
Update Figure Sequence
Perform Perception()
Find resemblance

continued...

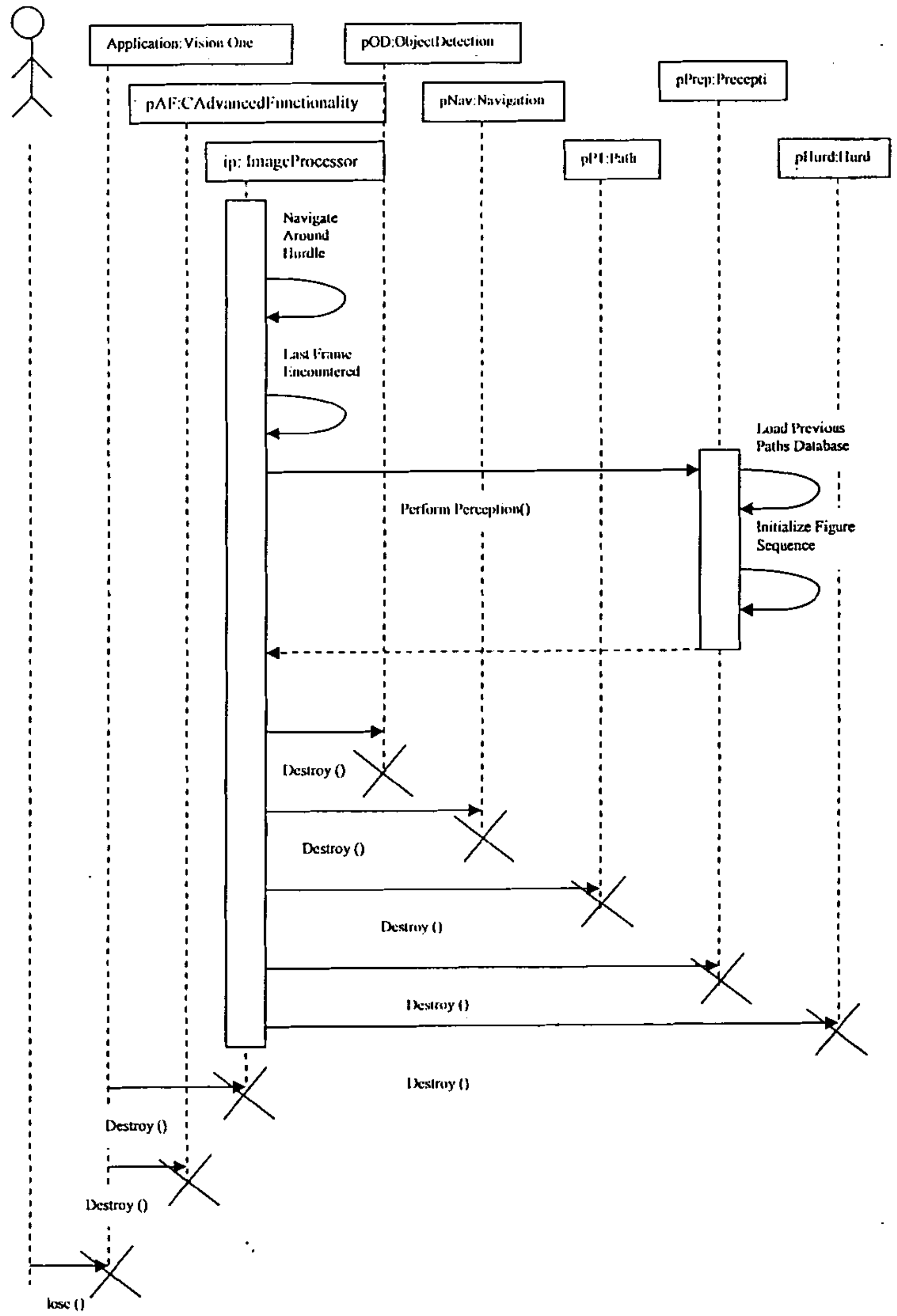


Fig 3-9: Sequence Diagram for Real-Time/Post Processing navigation with a single hurdle encounter

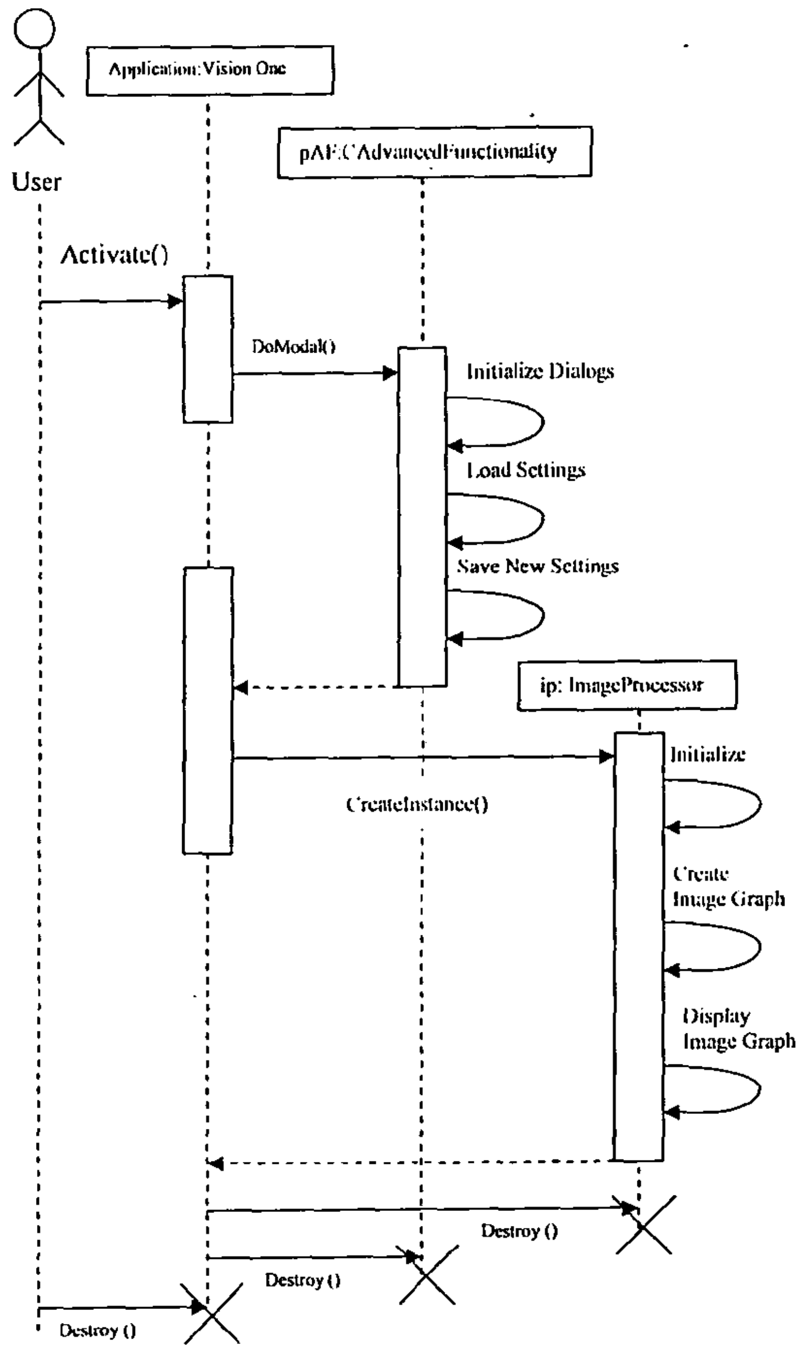


Fig 3-10: Sequence Diagram for displaying graph (Gray scale / RGB) of an image

3.6 State Transition Diagram

State Transition diagrams show the life cycle of a single object, from the time it is created until it is destroyed. Unlike Activity diagram that is activity-oriented, *State Transition* diagram is *event* oriented.

Following are different *State Transition* diagrams identified in the project "*Perception Based Obstacle Detection and Avoidance System for Autonomous Vehicles with Self-Localization Capability*"

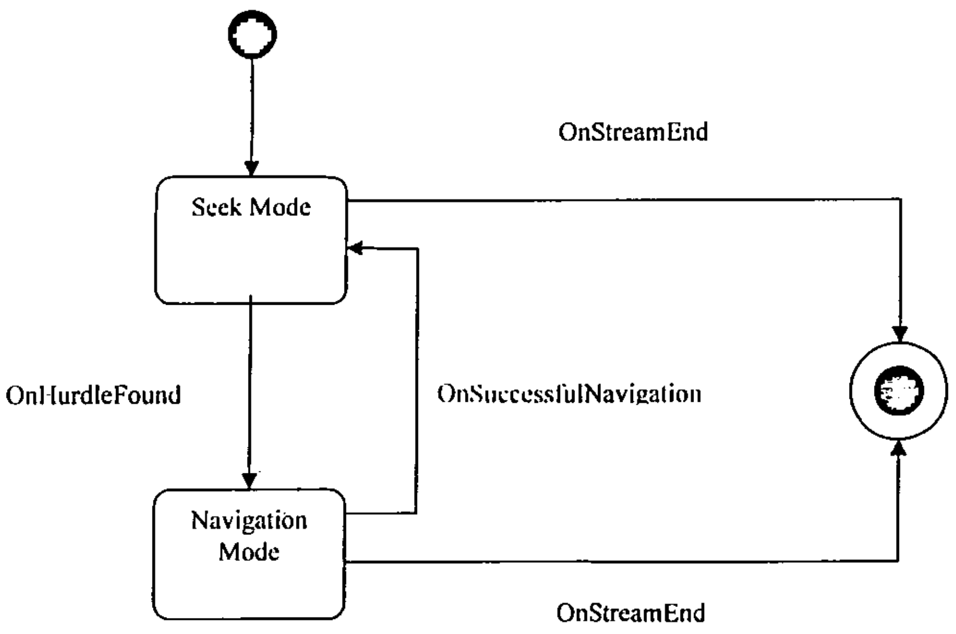


Fig 3-11: State Transition Diagram for Class ImageProcessor

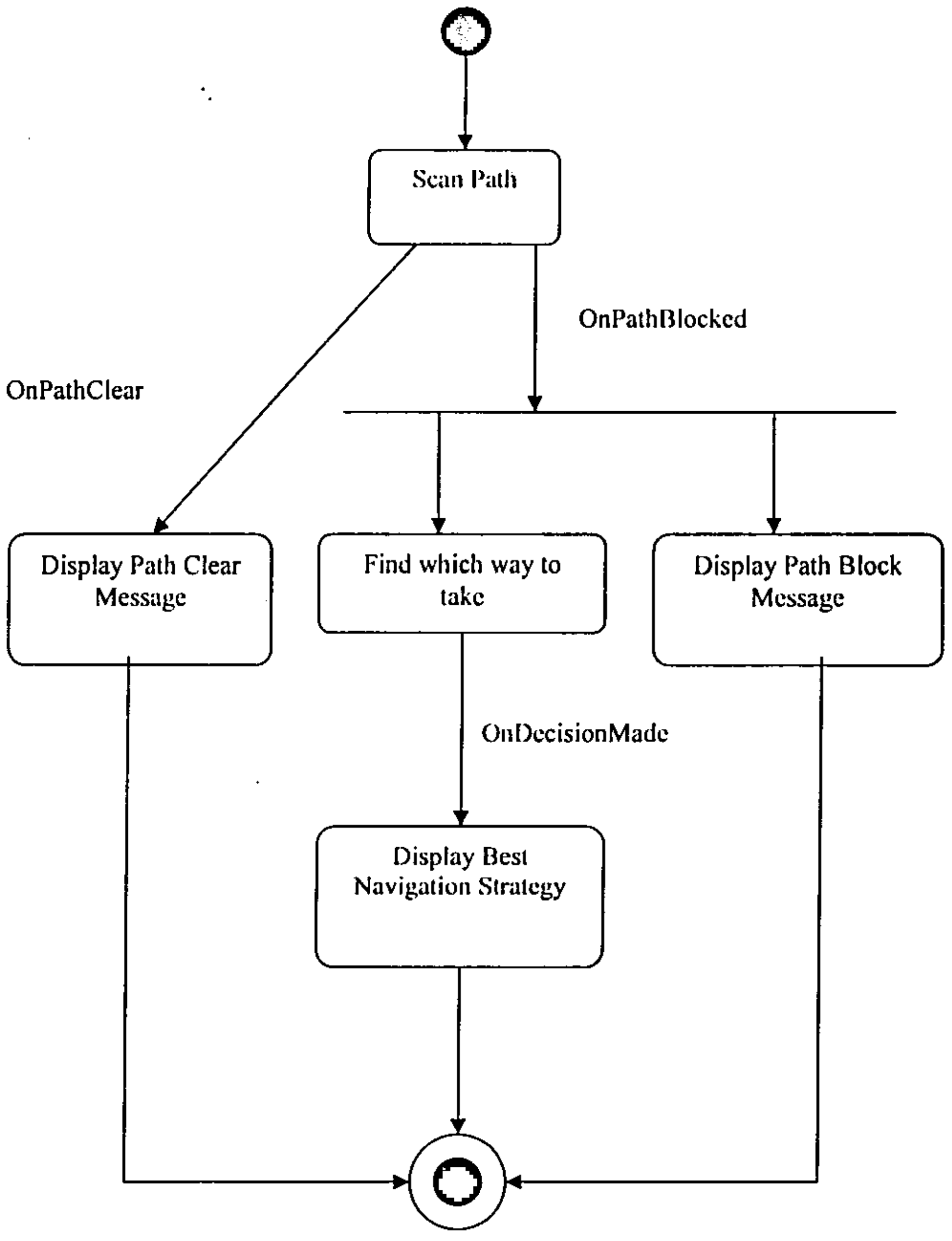

Scan Path
OnPathBlocked
OnPathClear
Display Path Clear Message
Find which way to take
Display Path Block Message
OnDecisionMade
Display Best Navigation Strategy

Fig 3-12: State Transition Diagram for Class Navigation

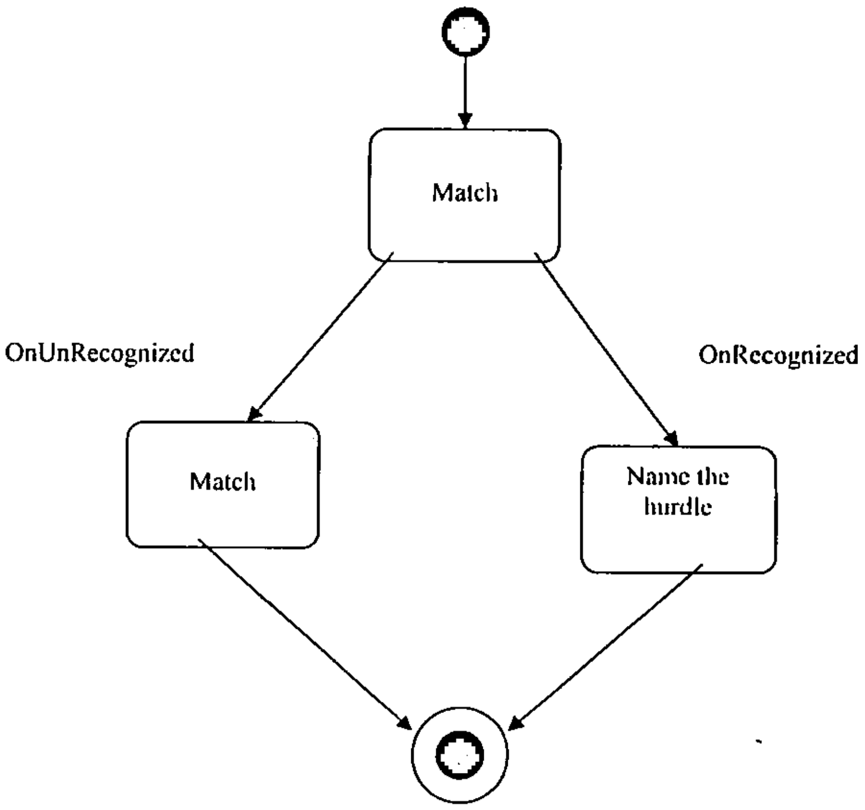


Fig 3-13: State Transition Diagram for class ObjectDetection

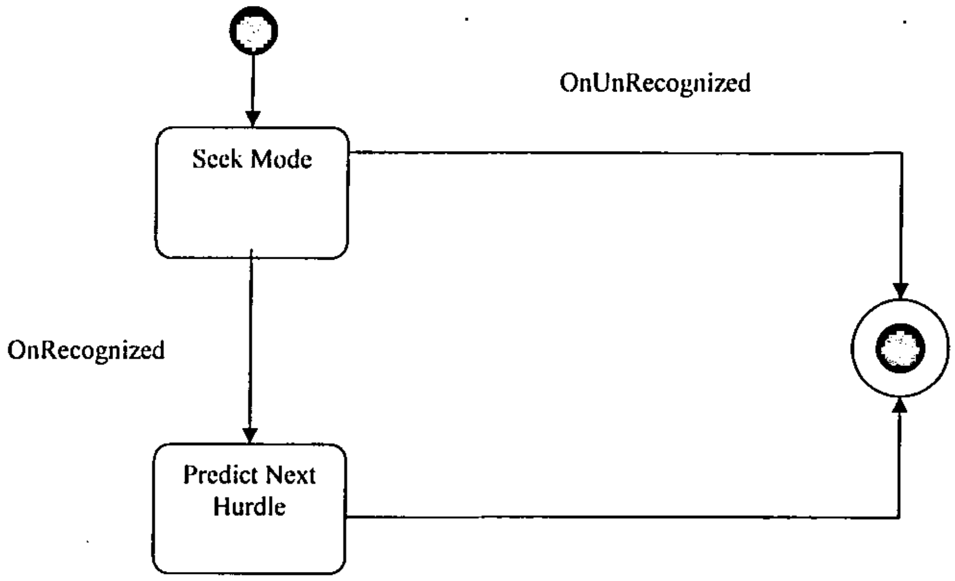


Fig 3-14: State Transition Diagram for Class Perception

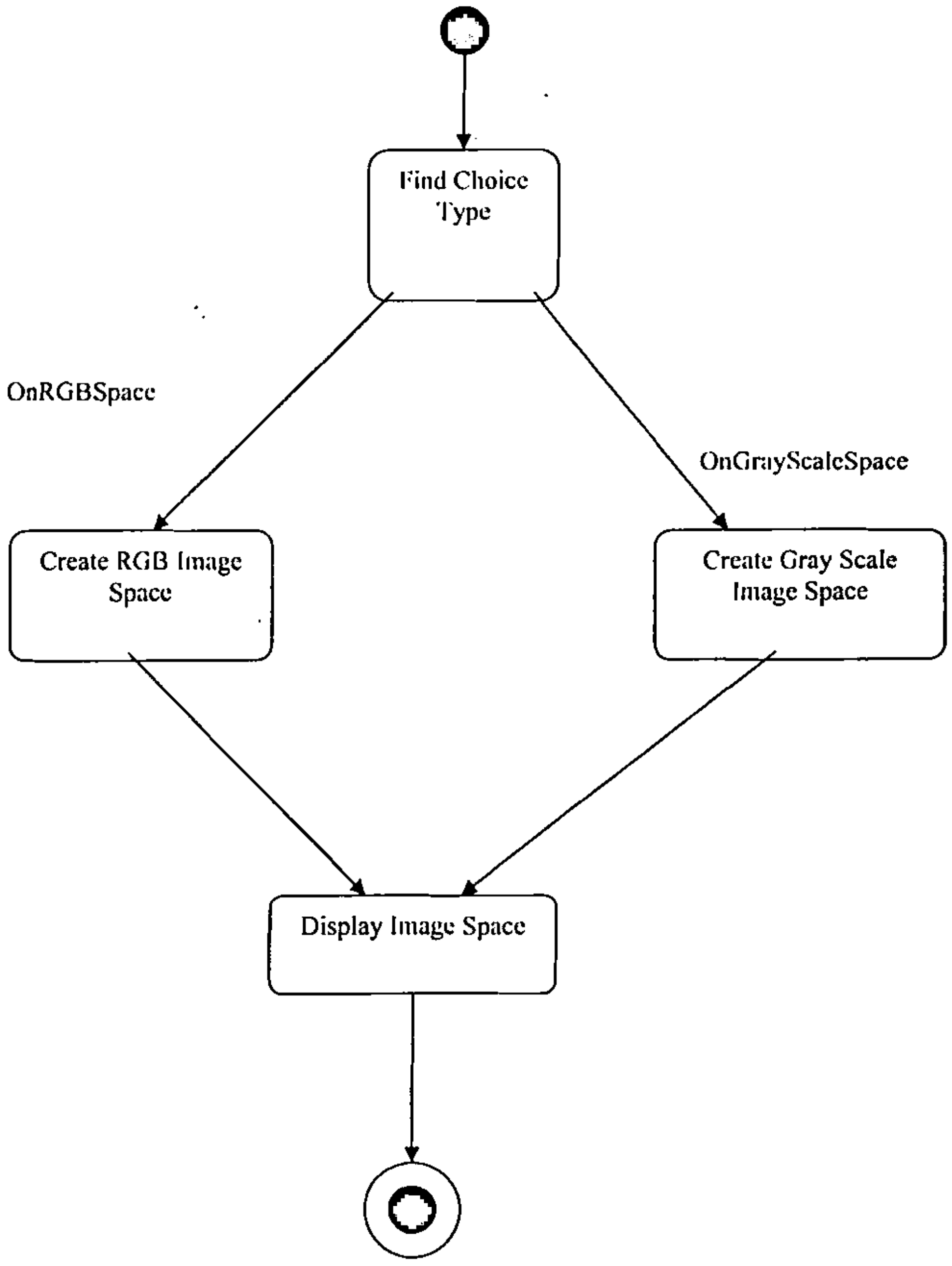


Fig 3-15: State Transition Diagram for Class Perception

3.7 Activity Diagram

Activity diagrams show the life cycle of a single object, from the time it is created until it is destroyed. Unlike State Transition diagram that is event oriented, Activity diagrams are activity-oriented.

Following are different *Activity* diagrams identified in the project "*Perception Based Obstacle Detection and Avoidance System for Autonomous Vehicles with Self-Localization Capability*"

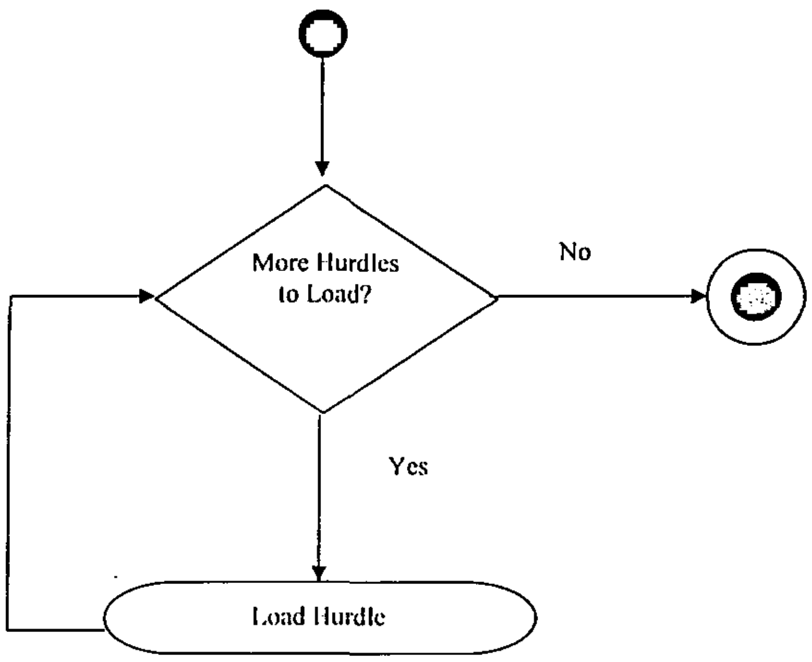


Fig 3-16: Activity Diagram for Load Paths activity

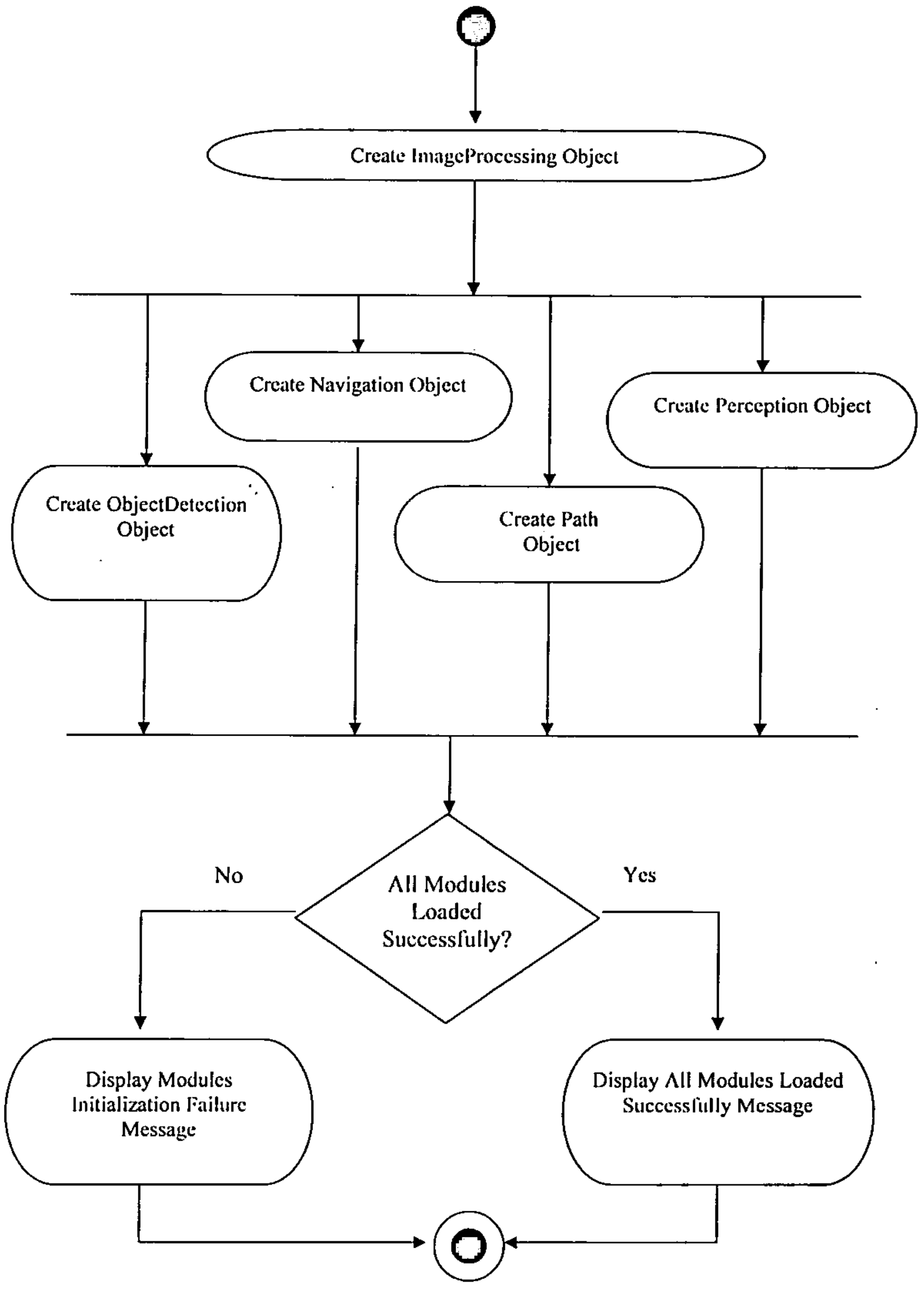

Create ImageProcessing Object
Create Navigation Object
Create Perception Object
Create ObjectDetection Object
Create Path Object
All Modules Loaded Successfully?
No
Yes
Display Modules Initialization Failure Message
Display All Modules Loaded Successfully Message

Fig-17: Activity Diagram Initialize Modules Activity

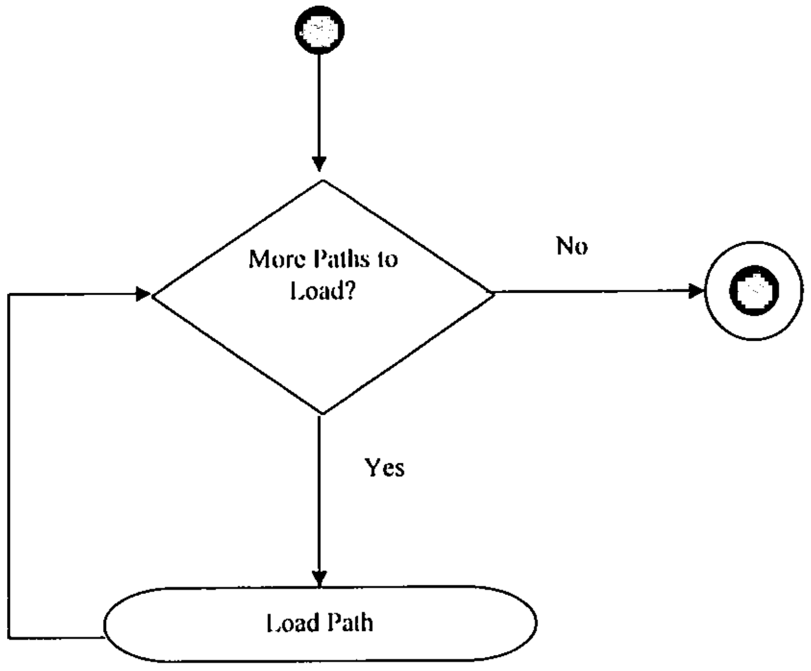


Fig 3-18: Activity Diagram for Load Paths activity

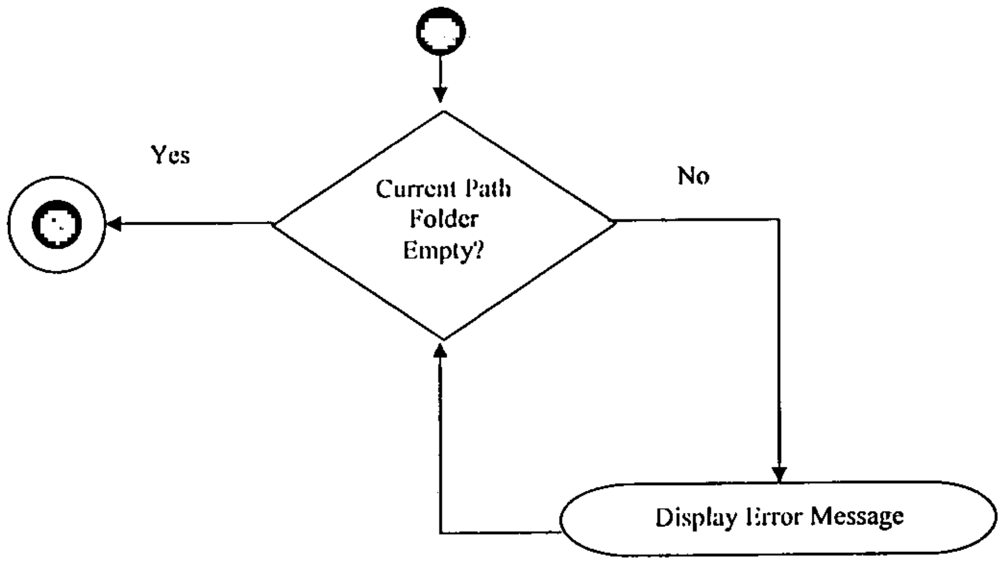


Fig 3-19: Activity Diagram for Validate folder activity

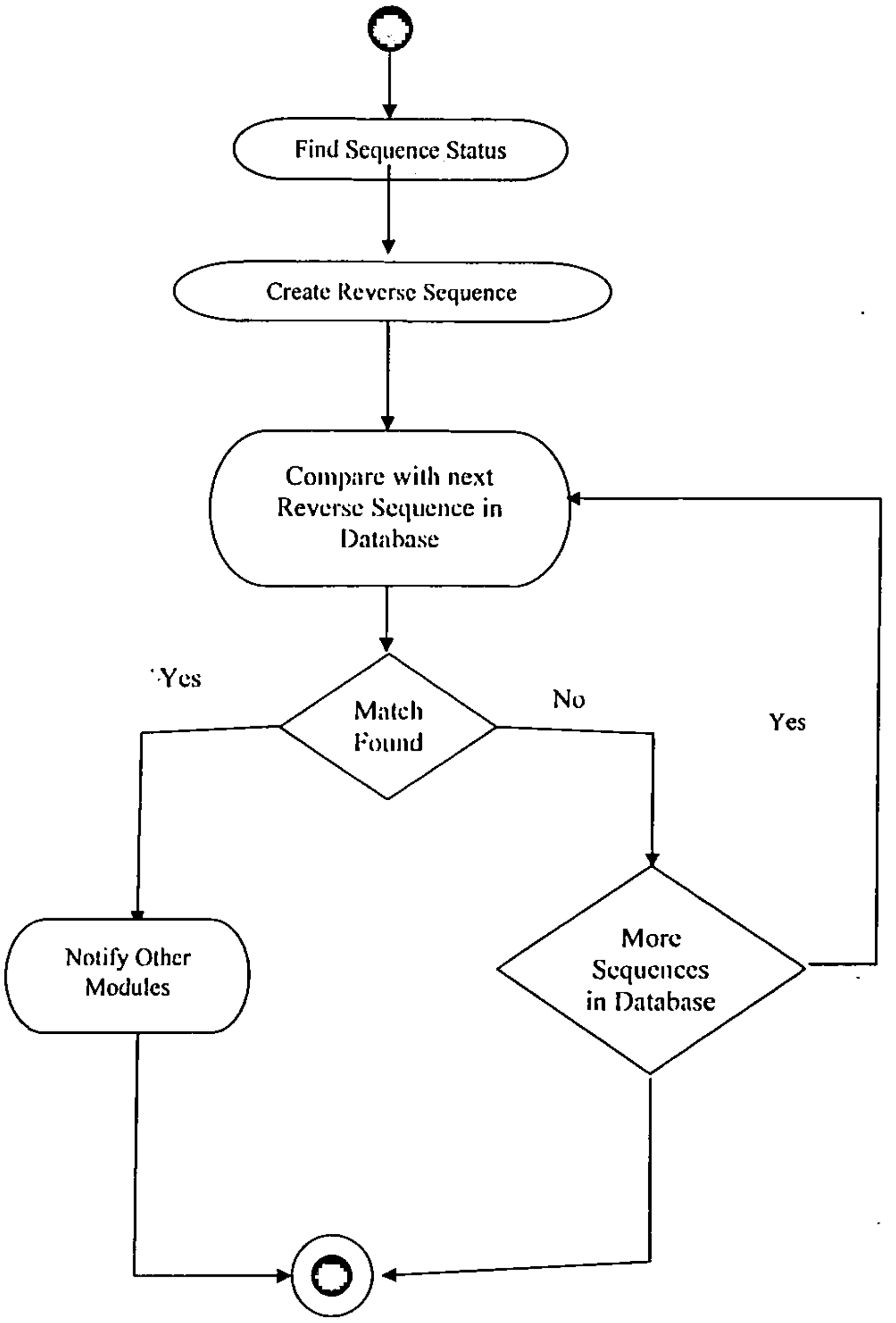


Fig 3-20: Activity Diagram for Match Hurdle Sequence in Reverse Activity

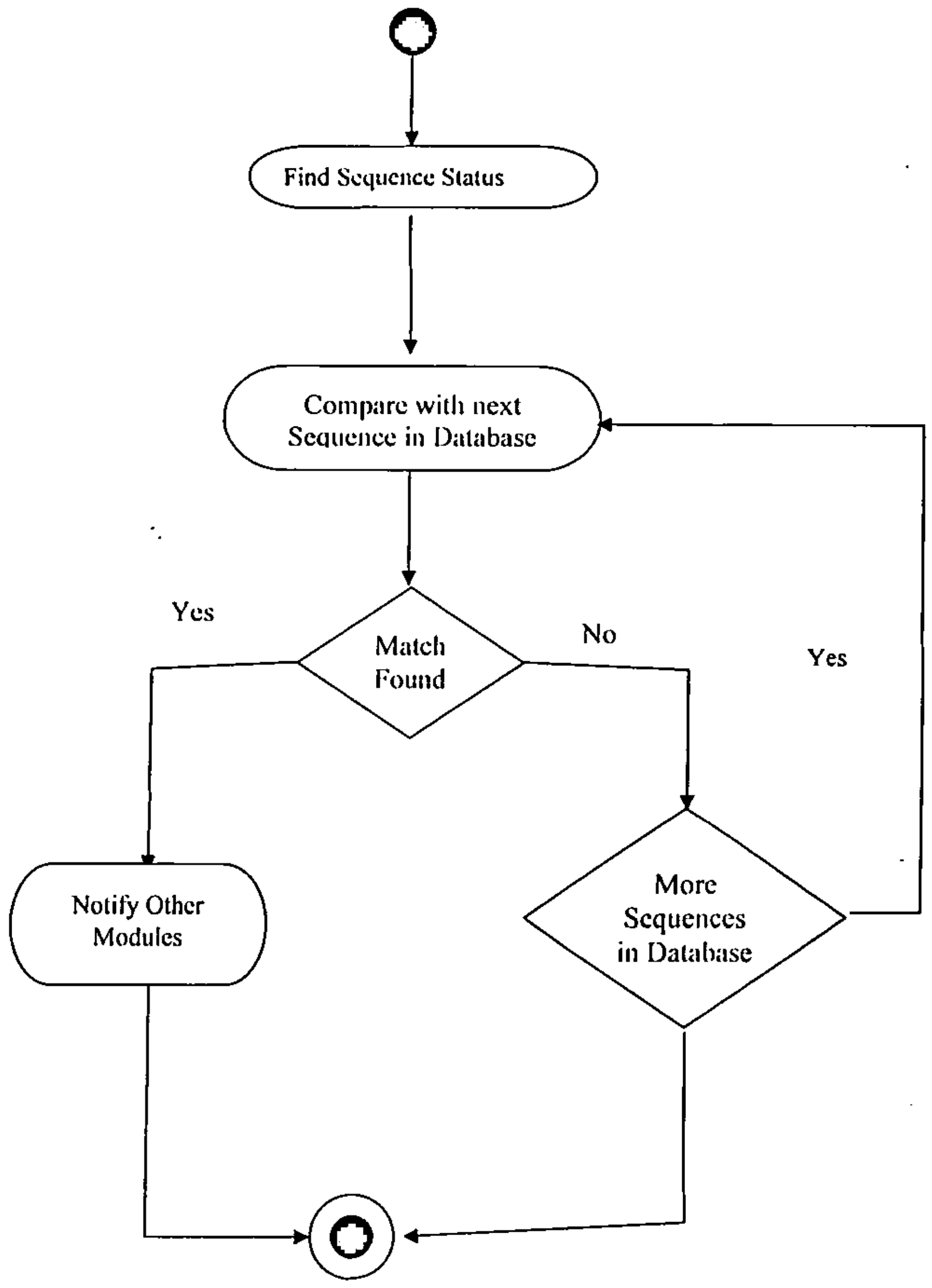

Find Sequence Status
Compare with next Sequence in Database
Match Found
Yes
No
Notify Other Modules
More Sequences in Database
Yes

Fig 3-21: Activity Diagram for Match Hurdle Sequence Activity

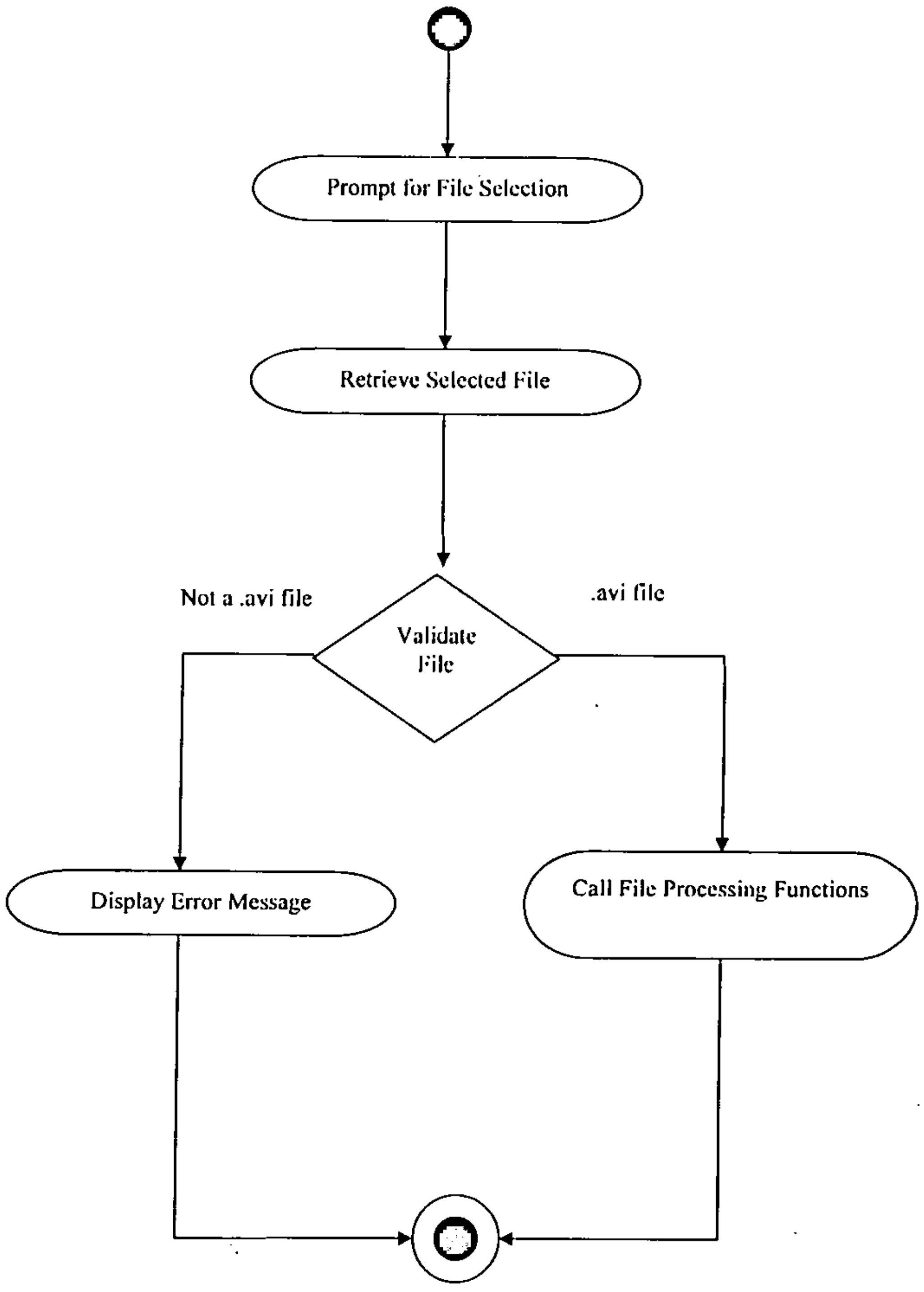


Fig 3-22: Activity Diagram for Choose file for processing activity

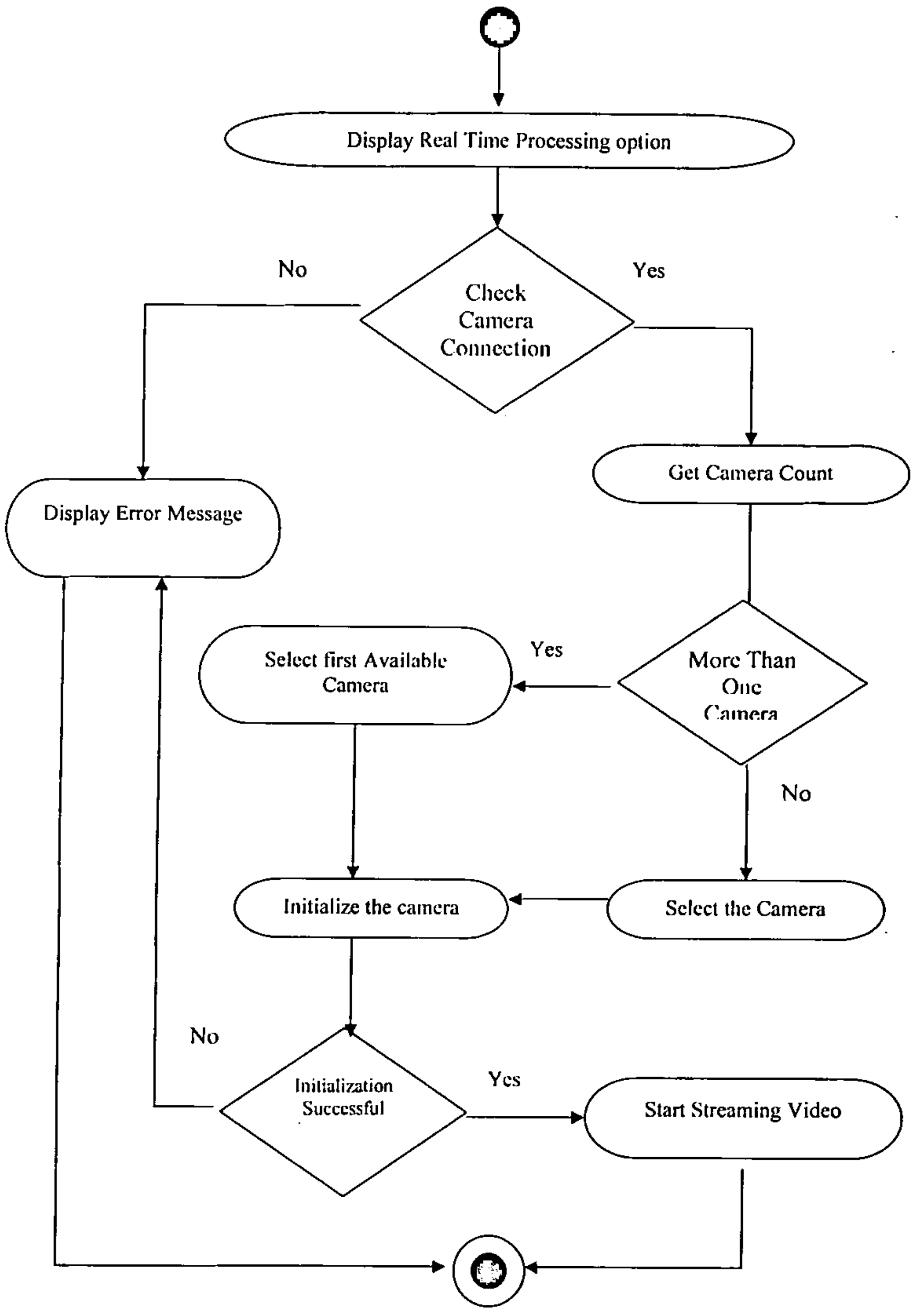


Fig 3-23: Activity Diagram for Real Time Processing Activity

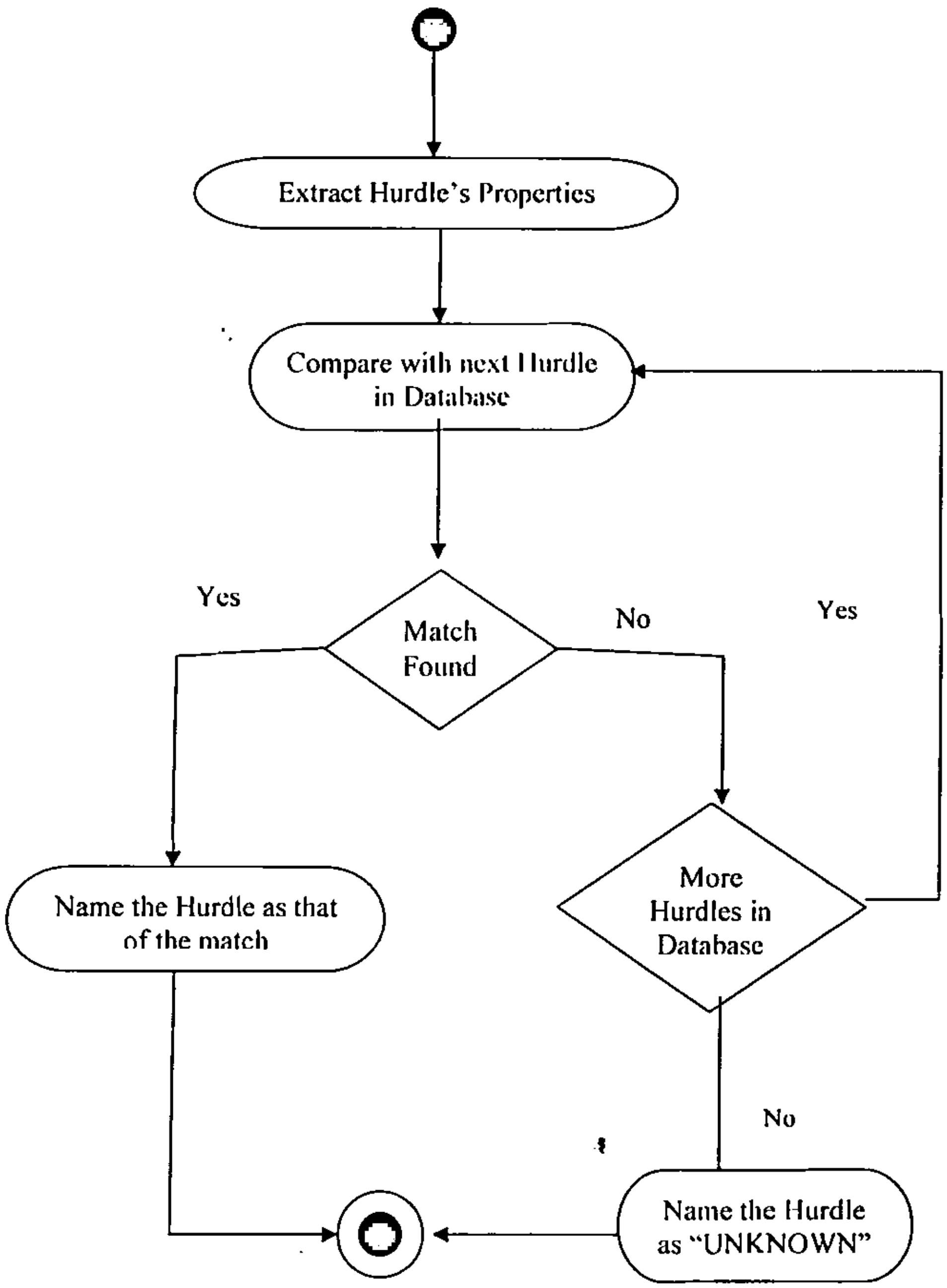

Extract Hurdle's Properties
Compare with next Hurdle in Database
Yes
Match Found
No
Yes
Name the Hurdle as that of the match
More Hurdles in Database
No
Name the Hurdle as "UNKNOWN"

Fig 3-24: Activity Diagram for Recognize activity

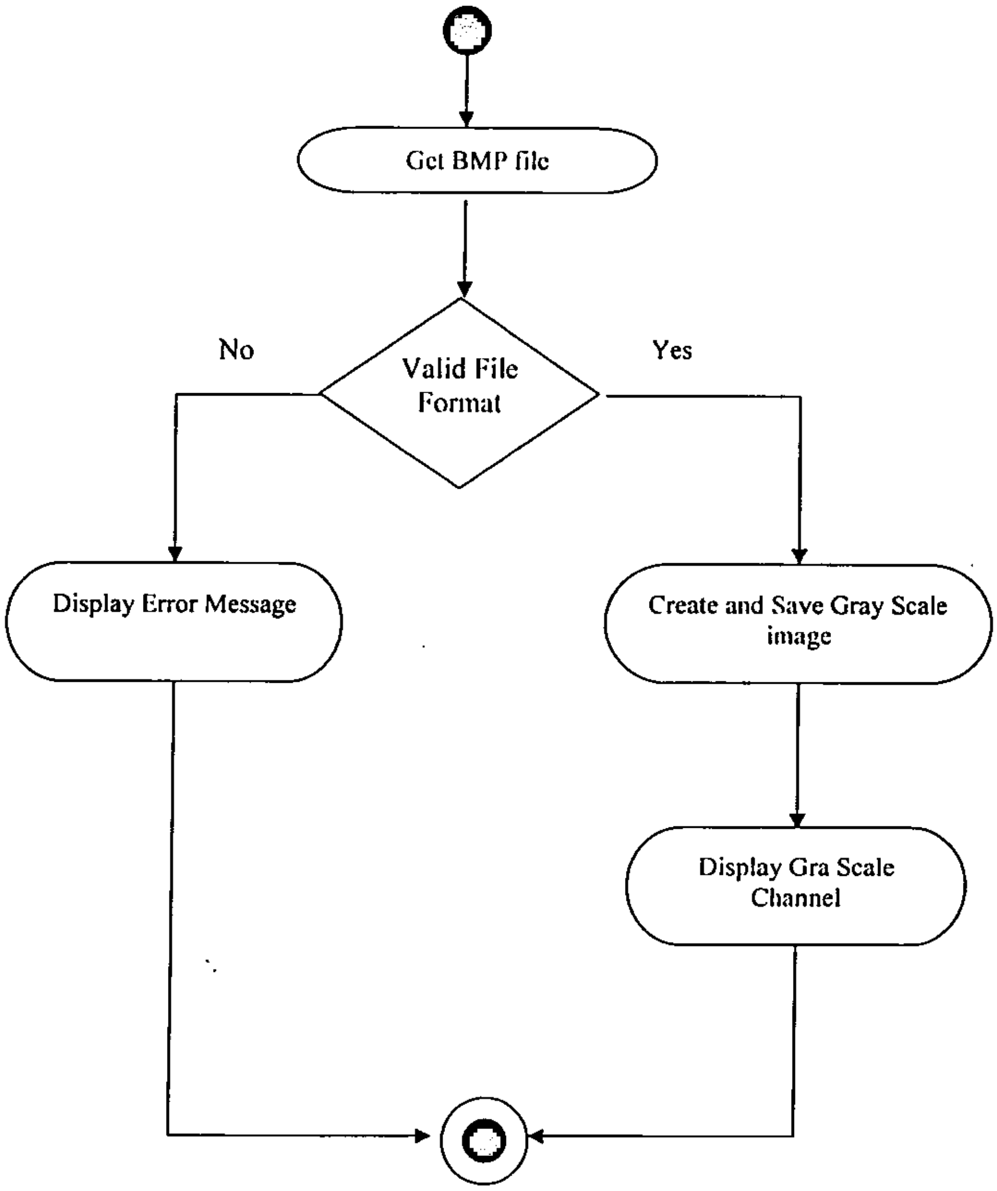


Fig 3-25: Activity Diagram for View Gray Scale channel Graph Activity

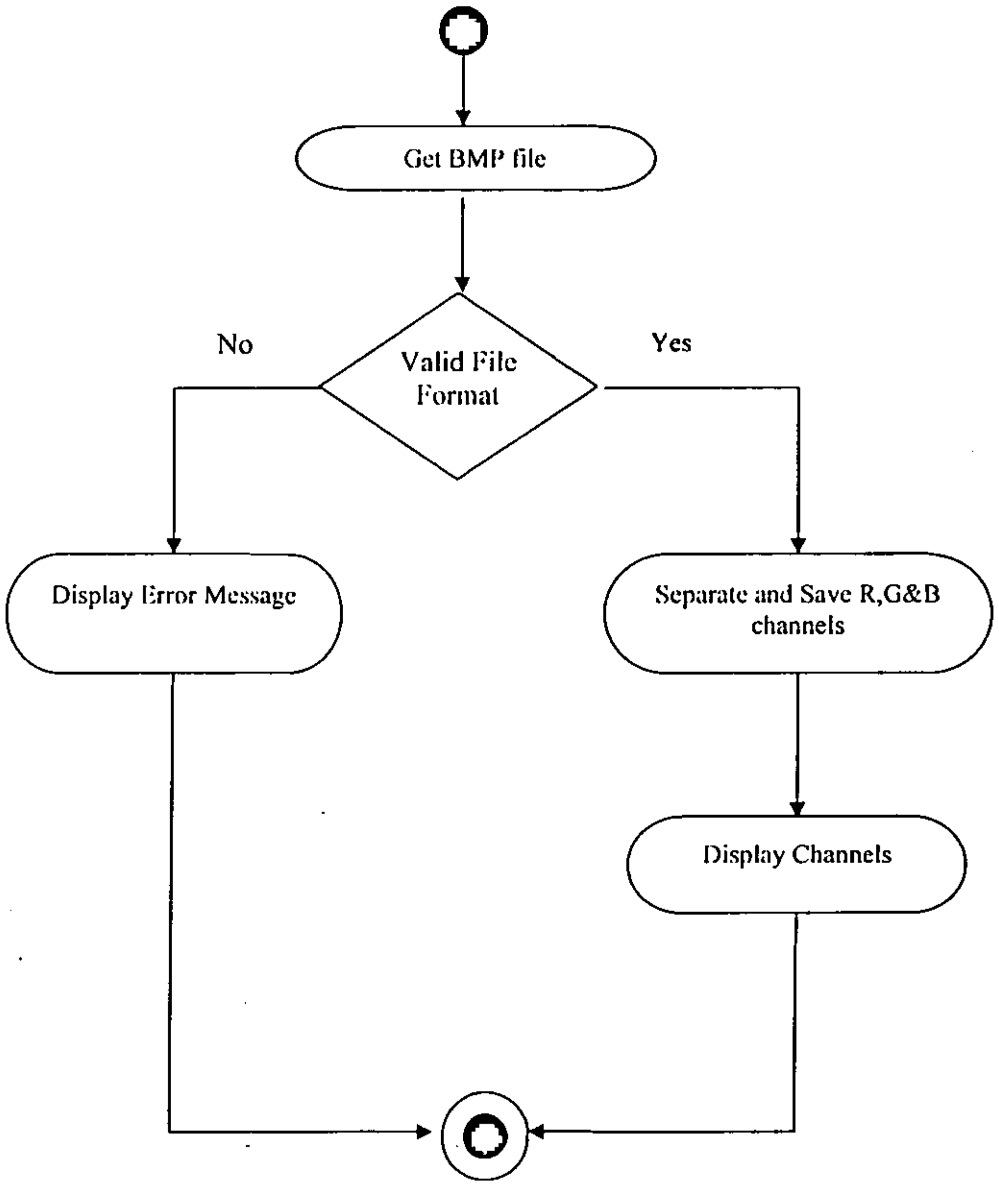


Fig 3-26: Activity Diagram for View RGB channel Graph Activity

3.8 Conclusion

In this chapter we have given detailed description of our design phase of the system development. In the beginning of this chapter, we have given an introduction to the basic concepts of design and different techniques to carry out design phase. Our approach to design was totally object oriented. We have thoroughly documented this phase and given diagrams where ever possible for the convenience of someone trying to understand our system. Due to complexity of the system, some diagrams can not come on a single page and require to be stretched to multiple pages. In such cases, we have mentioned this at the last line of such diagram with a "**continued...**" which means the diagram stretches to next page.

Chapter 4
Implementation

4. IMPLEMENTATION

This is the second last activity in the project and comes before testing of the whole program. However the partial testing can be done during implementation after completion of every module.

This is very important phase in the software engineering paradigm because no matter how efficiently analysis has been done or how brilliantly the design has been prepared, it all depends on what you present in form of implementation. Although programming is an outgrowth of analysis and design, all the programming and implementation skills have to be applied here, because any inefficiency on part of the programmer will hammer the quality of the software. Another important aspect of this phase is that, although this phase is succeeded by the testing phase, but during the implementaion phase the programmer is best equiped for glass box testing of the software, because at this stage he has the access to the code.

4.1 Implementation Techniques

The following techniques are used during the implementation of our project.

4.1.1 Object-Oriented Programming

Although all areas of object technologies have received significant attention within the software community, no subject has produced more books, more discussion, and more debate than *object-oriented programming* (OOP).

The software engineering viewpoint stresses OOA and OOD and considers OOP (coding) an important, but secondary, activity that is an outgrowth of analysis and design. The reason for this is simple. As the complexity of systems increases, the design architecture of the end product has a significantly stronger influence on its success than the programming language that has been used. And yet, "language wars" continue to rage.

4.1.2 Component-Based Programming

In the software engineering context, reuse is an idea both old and new programmers have stressed upon, since the earliest days of computing, but the early approach to reuse was ad hoc. Today, complex, high-quality computer-based systems must be built in very short time periods. This mitigates toward a more organized approach to reuse.

4.2 Implementation Tools

Our software is developed using two tools Microsoft Visual C++ and Intel's free Computer Vision tool called OpenCV. There are some reasons to select Visual C++ and OpenCV. The basic reason is VC++ provides ease for developing event-based, GUI applications for Windows based operating system. While OpenCV is emerging as a very strong Computer-Vision tool capable of handling real time processing needs. Along with that, OpenCV is compatible with VC++.

4.2.1 Microsoft Visual C++

Microsoft Visual C++.NET 2003 provides the dynamic development environment for creating Microsoft Windows–based and Microsoft .NET–based applications, dynamic Web applications, and XML Web services using the C++ development language. Visual C++ .NET includes the industry-standard Active Template Library (ATL) and Microsoft Foundation Class (MFC) libraries, advanced language extensions, and powerful integrated development environment (IDE) features that enable developers to edit and debug source code efficiently.

It provides developers with a proven, object-oriented language for building powerful and performance-conscious applications. With advanced template features, low-level platform access, and an optimizing compiler, Visual C++.NET delivers superior functionality for generating robust applications and components. The product enables developers to build a wide variety of solutions, including Web applications, smart-client Microsoft Windows-based applications, and solutions for thin-client and smart-client mobile devices. C++ is the world's most popular systems-level language, and Visual C++.NET 2003 gives developers a world-class tool with which to build software.

4.2.2 OpenCV

The OpenCV Library is mainly aimed at real time computer vision. Some example areas would be Human-Computer Interaction (HCI); Object Identification, Segmentation, and Recognition; Face Recognition; Gesture Recognition; Motion Tracking, Ego Motion, and Motion Understanding; Structure From Motion (SFM); and Mobile Robotics. The OpenCV Library is a collection of low-overhead, high-performance operations performed on images.

The OpenCV Library is a way of establishing an open source vision community that will make better use of up-to-date opportunities to apply computer vision in the growing PC environment. The software provides a set of image processing functions, as well as image and pattern analysis functions. The functions are optimized for Intel® architecture processors, and are particularly effective at taking advantage of MMXtechnology. The OpenCV Library has platform-independent interface and supplied with whole C sources. OpenCV is open.

4.3 Implementation Strategy

First problem encountered was trying to make the Intel's tool OpenCV compatible with Microsoft's VC++. Despite the Intel's claim of case, it is a real task just to make them work together in harmony and it took some considerable amount of time.

As mentioned earlier, no Imgc Computing background meant that we had to explore the field in parallel with persuit of the project. Hence the implementation strategy followed was iterative increasing in complexity and can be better described as below:

1 - Loading and displaying simple images of different formats using OpenCV and VC++ and manipulating pixel values of images.

2- Creating gray scale image of a given image along with intensity vs. frequency graph of an image after converting it into gray image.

3 - Separating the RGB channels of an image and creating intensity vs. frequency graph of an image in RGB space.

4 - Examining the texture of intended ground type in order to find a low computational-cost way to separate the intended ground plane from the rest of the environment of an image.

5 - Testing ground detection algorithm in different scenarios of single images and then converting ground detection algorithm properties to apply on an image stream.

6 - Applying ground detection algorithm on Real-Time image stream.

7 - Creating logic to detect hurdles on the ground.

8 - Incorporating hurdle detection with ground detection algorithm and making them one entity.

9 - Establishing higher level logic for making decisions regarding navigation, hurdle detection and avoidance etc.

10 – Combining all the above into one whole.

11 – Adding ability to Pos-Process along with Real-Time Processing.

12 - Testing in different scenarios (controlled environment).

13 – Creating easy GUI for the user.

4.4 OpenCV and IPL Image Library Functions Description

Following are some important functions and vriables of OpenCV and image library called IPL used in the implementation.

IplImage*	Pointer to an image in IPL pre-defined format
cvCloneImage()	Used to create a copy of an image
cvSetZero()	Set all channel values of an image to zero
cvLoadImage()	To load an image
cvSaveImage()	To save an image
cvCreateImageHeader()	Creating IPL type image header
cvNamedWindow()	To create a window with defined properties
cvShowImage()	To show an Image in a window
cvCreateImage()	Creating IPL type image
cvSize()	Width and height retriever function
cvCreateMemStorage()	Creating a memory storage block
cvWaitKey()	To wait for a specified period of time
cvLine()	To draw line of desired properties
cvPoint	A two dimensional integer type variable
cvReleaseImage()	To release the memory space occupied by an image
cvDestroyWindow()	To destroy a window

4.5 Implementation Details

Since the implementation of this project consists of more than ten thousand lines of code, it would be in-appropriate to put all the trivial code or even less trivial code inside implementation documentation. Hence the concentration during preparing implementation documentation is given to reduce the clutter as much as possible and strip the code to bare minimum in order to make it understandable. Implementation details contain source code as well as suedo-code (where whichever is appropriate) in order to make it easier to understand.

4.5.1 Classes in the System

Following are the most prominent classes in the system, objects of these classes are at the core of the software developed.

1 – ImageProcessor	4- ObjectDetection
2 – Hurdle	5 – Path
3 – Navigation	6 - Perception

4.5.1.1 Declaration of Class Objects

```
CVision1App theApp;

CAdvancedFunctionality* pAF;

ImageProcessor ip;

Navigation* pNav;

Perception* pPrcp;

Hurdle* pHurd;

Path* pP1;

ObjectDetection* pOD;
```

4.5.2 Implementation of Different User Options

Following is the implementation of some important options available to the user.

4.5.2.1 Choosing Post-Processing Option

```
void CVision1Dlg::OnPostProcessing()

{
        CFileDialog dlg(TRUE, _T("*.avi"), "",
        OFN_FILEMUSTEXIST|OFN_PATHMUSTEXIST|
        OFN_HIDEREADONLY,
        "Audio Video Interleaved Files (*.avi)|*.avi|", NULL);

        char title[]= {"Choose AVI File"};

        dlg.m_ofn.lpstrTitle = title;

        if (dlg.DoModal() == IDOK) {

        // contains the selected filename
        CString path= dlg.GetPathName();
        ImageProcessor ip(1, path);

}
}
```

4.5.2.2 Choosing Real -Time Processing Option

Implementation of Real-Time in form of psuedo code is as follows

```
1. Check whether camera connected

2. If camera not connected than display message and exit

3. If more than one cameras connected, select the first camera

4. Initialize the camera

5. Start sreaming
```

The implementation in form of code is as follows:

```
CvCapture* capture = 0;

//check camera connected
//check if connected than how many

capture = cvCaptureFromCAM(-1); //select the first available camera

if( !cvGrabFrame( capture )) //camera cannot be initialized
{
        //display error message
}
else
{
        //start streaming
}
```

4.5.2.3 Choosing View RGB Space Option

```
void CAdvancedFunctionality::OnRgbs()
{
        CFileDialog dlg(TRUE, _T("*.bmp"), "",
        OFN_FILEMUSTEXIST|OFN_PATHMUSTEXIST|
        OFN_HIDEREADONLY,
        "Image Files (*.bmp; *.jpg) |*.bmp;*.jpg|",
        NULL);

        char title[]= {"Choose BMP/JPG File"};
        dlg.m_ofn.lpstrTitle = title;

        if (dlg.DoModal() == IDOK) {

        // contain the selected filename
        CString path= dlg.GetPathName();

        ImageProcessor::im_ImgSp = 2;
        ImageProcessor ip(path);
        }
}
```

4.6 Ground and Hurdle Detection Algorithm

Ground and hurdle detection works at the very core of the developed system. All high level decisions are based on the findings of this algorithm. In a way, ground and hurdle detection algorithm is the work horse of the system.

Working of the algorithm from implementation point of view for one image frame is as follows:

1. Initialization necessary for that frame

2. Creating a gray-scale image of the image

3. Applying texture comparison with pre-stored ground template

4. Scanning the algorithmed image to classify abnormalities as hurldes or noise as follows:

4.1- Calling MasterMind() function for scanning the image

4.2- Calling MyFloodFill() function for classifying the anomalies

5. Notifying other modules of the result

These functions are performed as follows:

```
//all the necessary initialization of different variables
InitOnEvFrame();

//algorithm this frame for finding anomalies
TexCompareGround();

//loop for checking the result of find anomaly pahse


while(dCheck)
{

//increment the iterator by one
Rmove++;

//find if there is an anomaly at the current position of the iterator
if( ((uchar*)(dummy->imageData + dummy->widthStep* i))[Rmove] == 0 )
{
        //if anomaly found, find out its properties like dimensions etc, by calling
        //MyFloodFill with information currently available
        //anomally found calling Flood fill
        //please see the documentation of MyFloodFill()in the next topic
        nObjSer = MyFloodFill(Rmove, i, Lbound, Rbound, forBound);

            //anomaly's ending point for this row
            if(nObjSer > 0)
            Rmove = nObjSer;

            //find if you have reached row's end reached
            else if(nObjSer == -1)
            {
                //row's end reached
                //scan for upto the height of upper path bound
                if(--k >= Lpt2.y)
                {
                    //re-initialize Rmove to work on the upper row
                    //as row's end was reached
                    Rmove = Lbound;

                    i = k;
                }
                else
                //scanning this row completed successfully
                dCheck = false;
            }
            else
            {
            //if scanning this row completed was terminated abnormally
            AfxMessageBox("Abnormal Termination in finding anomally
            phase");
            dCheck = false;
            }
        }
    }

    //finds out dimensions of anomaly
    //classifies it as hurdle or noise,
    //works on multiple rows intelligently and updates Rmove counter of caller
    //function accordingly
    MyFloodFill(anomally pixel location, row number, other information)
    {
        //variables used
        int Rmove, Lmove, secPos, maxWid, maxHght, retVal;

        int j;                          // j simple iterator
```

```
bool valRcomp, valLcomp, valUp, dCheck;

//find whether thing detected is hurdle or not
bool tboo;
valRcomp = valLcomp = dCheck = true;

//we need this specifically to make the loop run for the first time
valUp = true;
maxWid = maxHght = 0;
Lmove = Rmove =  origPix;
secPos = origPix;
j = 0;
retVal = 1;

//initialize width array
InitWid();
wid[wi][0] = curRow;
wid[wi][2] = origPix;
SetBounds(curRow);

//Initializing while loop
while(!valRcomp)
{
if(Rmove < Rbound)
{
        //find out anomaly's end point
        if( ((uchar*)(dummy->imageData + dummy->widthStep*
        wid[wi][0]))[Rmove] != 0 )
        {
        //if this was the ending point
        AfxMessageBox("Found not black pix while going right in flood
        fill");
        //update the hurdle dimension array
                wid[wi][1] = Rmove - 1;
        //notify that object in this row completed
                valRcomp = true;
        //indicate the hurdle by turning it black in image called "heck"
        for(j = wid[wi][1]; j <= wid[wi][2]; j++)
        ((uchar*)(heck->imageData + heck->
        widthStep* wid[wi][0]))[j] = 0;
                retVal = 0;
        }
        }
        else
        {
        //you reached row's end but found no white pixel
```

```
                    wid[wi][3] = ++Rmove;

                //by force object in this row completed
                    valRcomp = true;

                //indicate the hurdle by turning it black in image called "heck"
                for(j = wid[wi][1]; j <= wid[wi][1]; j++)
                ((uchar*)(heck->imageData + heck->
                widthStep* wid[wi][0]))[j] = 0;
                retVal = -2;
                }

                Rmove++;
            }

    //out of while loop, calling SearchUp() to see if hurdle is in upper rows too
    //necessary initialization
            secPos = wid[wi][2];

    //SearchUp() returns bool value
    //it is intelligent enough that if called iteratively,
    //it can search and update anomaly's position accurately for each row
    while(SearchUp());

    //Calling InocGuilty(),InocGuilty() function is the judge of classifying the
    //anomally as hurdle or noise

    tboo = InocGuilty();            //v imp: true if thing was a hurdle
            if((tboo) && (ClearCheck))  //if anomaly was hurdle
            {
                ClearCheck = false;

                //Found a hurdle
                //Notify Navigation module immediately
                pNav->BckorClr(0);
            }
            else
            BckorClr(1);  //if anomaly was not hurdle
            if(retVal > 0)
            return secPos;
            else
            return retVal;

    }
```

4.7 Implementation of Intensity vs Frequency Graph for Gray-Scale Image

```
//array for storing gray scale values
int GreyC[256][2];
char Buff[100];

//initializtion of arrays
for(i = 0; i < 256; i++)
{

                GreyC[i][0] = i;
        GreyC[i][1] = 0;
}
//Window for displaying the result
cvNamedWindow("Separated Channels");
IplImage* sep1 = cvCreateImage();

//Initialize image
cvSetZero(sep1);
for( i = 0; i < fresh->height; i++)
{
        for( l = 0; l < fresh->width; l++)
        {
                .       .       .       .       .
                .       .       .       .       .
                .       .       .       .       .
                d = ((uchar*)(fresh->imageData + fresh->widthStep* i));
                GreyC[d][1] += 1;
        }
}

for(i = 0; i < 256; i++ )
{
        for( l = 0; l < GreyC[i][1]; l++)
        {
                .       .       .       .       .
                .       .       .       .       .
                .       .       .       .       .
                ((uchar*)(sep1->imageData + sep1->widthStep* i))[l] = i;
        }
}

//show the graph
cvShowImage("Separated Channels", sep1);
}
```

4.8 Implementation of Intensity vs Frequency Graph for RGB Image

```
//used for storing hits numbers for each pixel value
int rC[256], gC[256], bC[256];

//initialzation of arrays

        for(i = 0; i < 256; i++)
{
        rC[i] = 0;
        gC[i] = 0;
        bC[i] = 0;
}


cvNamedWindow("Separated Channels");
IplImage* sep3 = cvCreateImage();

//Initializing image
cvSetZero(sep3);

/ /R Channel

for( i = 0; i < fresh->height; i++)
{
        for( l = 0; l < fresh->width ; l++)
        {
        .       .       .       .       .
        .       .       .       .       .
        .       .       .       .       .
        d = ((uchar*)(fresh->imageData + fresh->widthStep* i))[l*3];
        rC[d] += 1;

        }
}


for(i = 0; i < 256; i++ )
{
//its quite possible to have more ocuurances of an hue value and and error can be
//generated
.       .       .       .       .
.       .       .       .       .
.       .       .       .       .
for( l = 0; l < rC[i]; l++)
((uchar*)(sep3->imageData + sep3->widthStep* i))[l*3] = i;
}

//G Channel
for( i = 0; i < fresh->height; i++)
{
        for( l = 0; l < fresh->width ; l++)
        {
        .       .       .       .       .
        .       .       .       .       .
        .       .       .       .       .
        .       .       .       .       .
        d = ((uchar*)(fresh->imageData + fresh->widthStep* i))[l*3];
        rC[d] += 1;

        }
}


for(i = 0; i < 256; i++ )
{
//its quite possible to have more ocuurances of an hue value and and error can be
//generated
        .       .       .       .       .
        .       .       .       .       .
        .       .       .       .       .
        for( l = 0; l < rC[i]; l++)
        ((uchar*)(sep3->imageData + sep3->widthStep* i))[l*3+2] = i;
}

// B Channel
for( i = 0; i < fresh->height; i++)
{
        for( l = 0; l < fresh->width ; l++)
        {
        .       .       .       .       .
        .       .       .       .       .
        .       .       .       .       .
        d = ((uchar*)(fresh->imageData + fresh->widthStep* i))[l*3+2];
        bC[d] += 1;

        }
}
```

```
for(i = 0; i < 256; i++ )
{
//its quite possible to have more ocuurances of an
//hue value and  error can be generated
.        .        .        .        .
.        .        .        .        .
.        .        .        .        .
((uchar*)(sep3->imageData + sep3->widthStep* i))[l*3+2] = i;
}

cvNamedWindow("Cloning");
IplImage* clone = cvCloneImage(sep3);
cvSetZero(clone);

for(i = 0; i < 256; i++ )
{
        //remeber rC[i] has also been set right above
        if(rC[i] > (clone->height - 149))
        rC[i] = (clone->height - 149);


        for( l = 0; l < rC[i]; l++)
        {

                ((uchar*)(clone->imageData + clone->widthStep*
                 ((clone->height) - l) ))[i*3+2] = i;
        }

}



for(i = 0; i < 256; i++ )
{
        //remeber rC[i] has also been set right above
        if(gC[i] > (clone->height - 149))
        gC[i] = (clone->height - 149);


        for( l = 0; l < gC[i]; l++)
        {

                ((uchar*)(clone->imageData + clone->widthStep*
                 ((clone->height - 150) - l) ))[(i + 260)*3+1] = i;
        }
        }


        for(i = 0; i < 256; i++ )
        {
                //remeber rC[i] has also been set right above
                if(bC[i] > (clone->height - 149))
                bC[i] = (clone->height - 149);


                for( l = 0; l < bC[i]; l++)
                {

                        ((uchar*)(clone->imageData + clone->widthStep*
                        ((clone->height - 150) - l) ))[(i + 520)*3] = i;
                }

        }
        //show the images
        cvShowImage("Cloning", clone);
        cvShowImage("Separated Channels", sep3);
```

4.9 Frame Processing Loop for Real – Time/Post – Processing

```
//start of loop
{
        //get next frame
        cvGrabFrame();

        //Windows artchitecture has the bad habit of having every frame from the
        //input stream Upside down, so every frame has to be put upright

        //turn the frame upright
        UpsideDownChk(frame);

        //initialization required for every frame
        InitOnEvFrame();

        //show the image captured from the device (cam/.avi file)
        cvShowImage( "cvcam window", frame );

        //send the frame to Ground and hurdle detection algorithm
        AllProcessing();
```

```
//draw the path lines, representing the critical area in front of the vehicle
DrawPath();

//display other windows that would help the user see the working
DisplayForCam();

//update frame count
FrNum++;

//tell the path object about this frame
pP1->DrawThePath(FrNum, HurdInFrame);

//are we in seek mode or navigation mode?
if((HurdInFrame == 1) && ( module == 0))
{
        if(clrFrames < 4)
        //Successfully Navigated around the hurdle\n shift to Detection module
        {
                //show "path clear" message instead of "path Blocked"
                pNav->BckorClr(5);

                //set arrow of movement command straight
                pNav->MoveImage(4);

                        //go back to detection module
                        module = 1;

                        //tell Navigation object that we have shifted to seek mode
                        pNav->module = 0;

                        //initialize clear frames count
                        clrFrames = 0;
                        //as indHurd is Zero Based
                        i = indHurd + 1;
        }
                clrFrames++;
}
else
clrFrames = 0;

//This frame is finished, initialize for the next frame
HurdInFrame = 0;

//it is the last frame to be processed so save it as end point
if(PNFrame == false)
```

```
pP1->SaveIt();

//Release the system resources used during this frame
//extremely important, else you'll have a great memory leak!!!
DestroyEm();

//break when you don't want to Process Next Frame
if(PNFrame == false)
break;

//wait for some time?
 if( cvWaitKey( 10 ) >= 0 )
break;
}//loop ends here
```

4.10 Conclusion

In this chapter we described our approach to system implementation. In the beginning of the chapter we describe different techniques of system implementation. Then we have given description of tools that we have used in implementation and described their importance. We also gave some important functions of an image processing library that we used in our implementation. In the end, we have given specific code to carry out some of the important functions used in the system. This code is well indented and description of each line is given to make it easy to understand.

Chapter 5
Testing

5. TESTING

The overall objective of the testing process is to identify the maximum number of errors in the code with a minimum amount of efforts. Finding an error is thus considered a success rather than failure. On finding an error, efforts are made to correct it.

5.1 Testing Process

Test consists of a number of test cases, where different aspects of the part of the project under test are checked. Each test case tells what to do, what data to use, and what results to expect. When conducting the test, the results including deviations from the planned test cases are not in a test protocol. Normally a deviation indicates an error in the system (although some times the test case is wrong, and the system is right). An error is noted and described in a test report for removal or directly removed by the programmer who developed that part.

5.2 General Types of Errors

Error can be of following types:

Functional error (e.g. function is not working correctly or missing).

Non-Functional error (e.g. performance is slow)

Logical error (e.g. error in algorithm, user interface errors is not considered as a logical error).

5.3 Testing Strategies

A strategy for software testing may be viewed as the spiral. Unit testing begins at the vortex of the spiral and concentrates on each unit (i.e., component) of the software as implemented in source code. Testing progresses by moving outward along the spiral to integration testing, where the focus is on design and the construction of the software architecture. Taking another turn outward on the spiral, we encounter validation testing, where requirements established as part of software requirements analysis are validated against the software that has been constructed. Finally, we arrive at system testing, where the software and other system elements are tested as a whole. To test computer software, we spiral out along stream-lines that broaden the scope of testing with each turn.

5.4 Testing the Software

Our software system is a real-time system, requiring heavy usage of system resources. Since the real-time systems require a particular activity to be completed within a bounded time interval, it is necessary that errors in the system must be carefully scanned and eliminated.

One major aspect of the system is the heavy use of memory space. This also makes the system vulnerable to memory leaks (dangling pointers) etc.

5.5 Features to be Tested

Following are the features that would be put under test to see their proper functionality and result.

1. Selecting Real-Time Processing option with no camera connected
2. Selecting Real-Time Processing option with multiple cameras connected
3. Selecting Post-Processing option with invalid file format
4. Selecting View Gray Space Graph option with invalid file format
5. Selecting View RGB Space Graph option with invalid file format
6. Validating "Current Path" folder
7. Navigation in Real-Time Mode with empty path
8. Navigation in Real-Time Mode with hurdles in the path
9. Navigation in Post-Processing Mode with empty path
10. Navigation in Post-Processing Mode with hurdles in the path

5.5.1 Selecting Real-Time Processing option with no camera connected

Input Specification:

Real-Time Processing option selection

Environmental Needs:

1. A Pentium® III or higher machine.
2. Windows 2000 (Family)

3. VC++ and OpenCV
4. Camera should be un-plugged

Expected Output:

Error message should be displayed and program should terminate gracefully.

Actual Output:

Error message displayed and program terminated gracefully

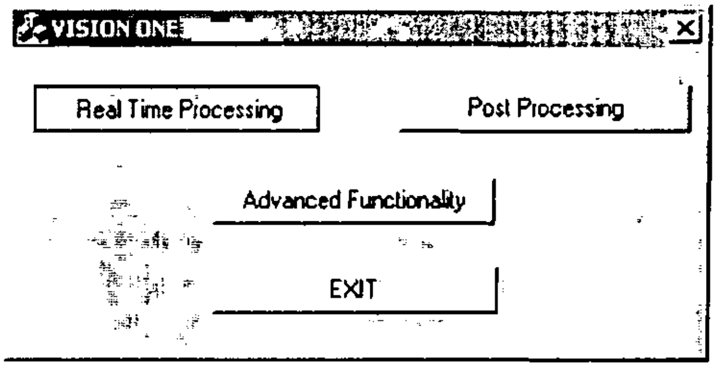
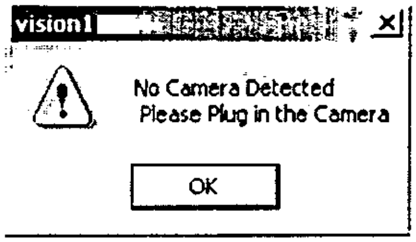


Fig 5-1: Interface visible to the user



No Camera Detected
Please Plug in the Camera

Fig 5-2: Message Displayed when no camera connected

5.5.2 Selecting Real-Time Processing option with multiple cameras connected

Input Specification:

Real-Time Processing option selection

Environmental Needs:

1. A Pentium® III or higher machine.
2. Windows 2000 (Family)
3. VC++ and OpenCV
4. More than one Cameras should be connected

Expected Output:

Multiple cameras connected message should be displayed and program should select the first available camera.

Actual Output:

Multiple cameras connected message displayed and first available camera selected.

5.5.3 Selecting Post-Processing option with invalid file format

Input Specification:

Post - Processing option selection with invalid file

Environmental Needs:

1. A Pentium® III or higher machine.
2. Windows 2000 (Family)
3. VC++ and OpenCV

Expected Output:

The user should not be allowed to select any format except .avi file format.

Actual Output:

The user not allowed selecting any format except .avi file format.

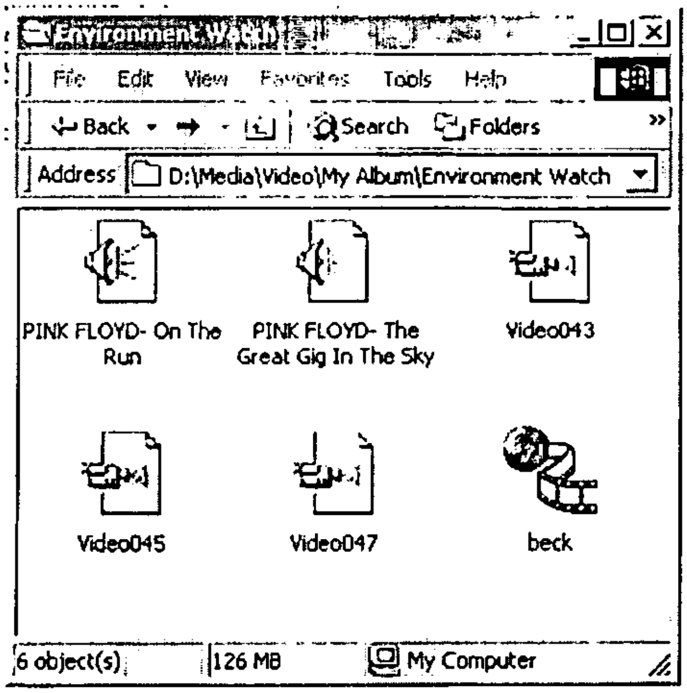

Environment Watch
File Edit View Favorites Tools Help
Back Search Folders
Address D:\Media\Video\My Album\Environment Watch
PINK FLOYD- On The Run
PINK FLOYD- The Great Gig In The Sky
Video043
Video045
Video047
beck
6 object(s)
126 MB
My Computer

Fig 5-3: Folder with various types of files

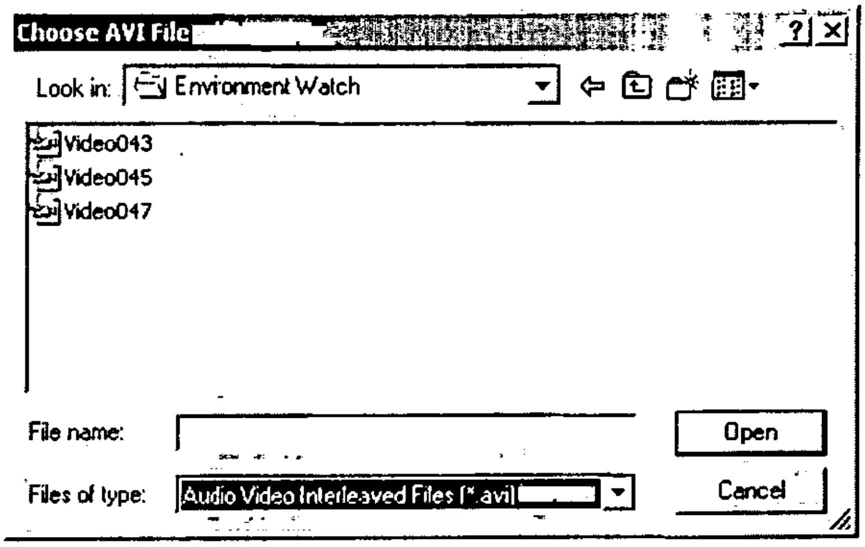

Choose AVI File
Look in: Environment Watch
Video043
Video045
Video047
File name:
Files of type: Audio Video Interleaved Files (*.avi)
Open
Cancel

Fig 5-4: Only files with .avi extention are allowed to be viewed in selection

5.5.4 Selecting View Gray Space Graph option with invalid file format

Input Specification:

An invalid file format selection

Environmental Needs:

1. A Pentium® III or higher machine.
2. Windows 2000 (Family)
3. VC++ and OpenCV

Expected Output:

The user should not be allowed to select any format except .bmp file format.

Actual Output:

The user not allowed to select any format except .bmp file format.

5.5.5 Selecting View RGB Space Graph option with invalid file format

Input Specification:

An invalid file format selection

Environmental Needs:

1. A Pentium® III or higher machine.
2. Windows 2000 (Family)
3. VC++ and OpenCV

Expected Output:

The user should not be allowed to select any format except .bmp file format.

Actual Output:

The user not allowed to select any format except .bmp file format.

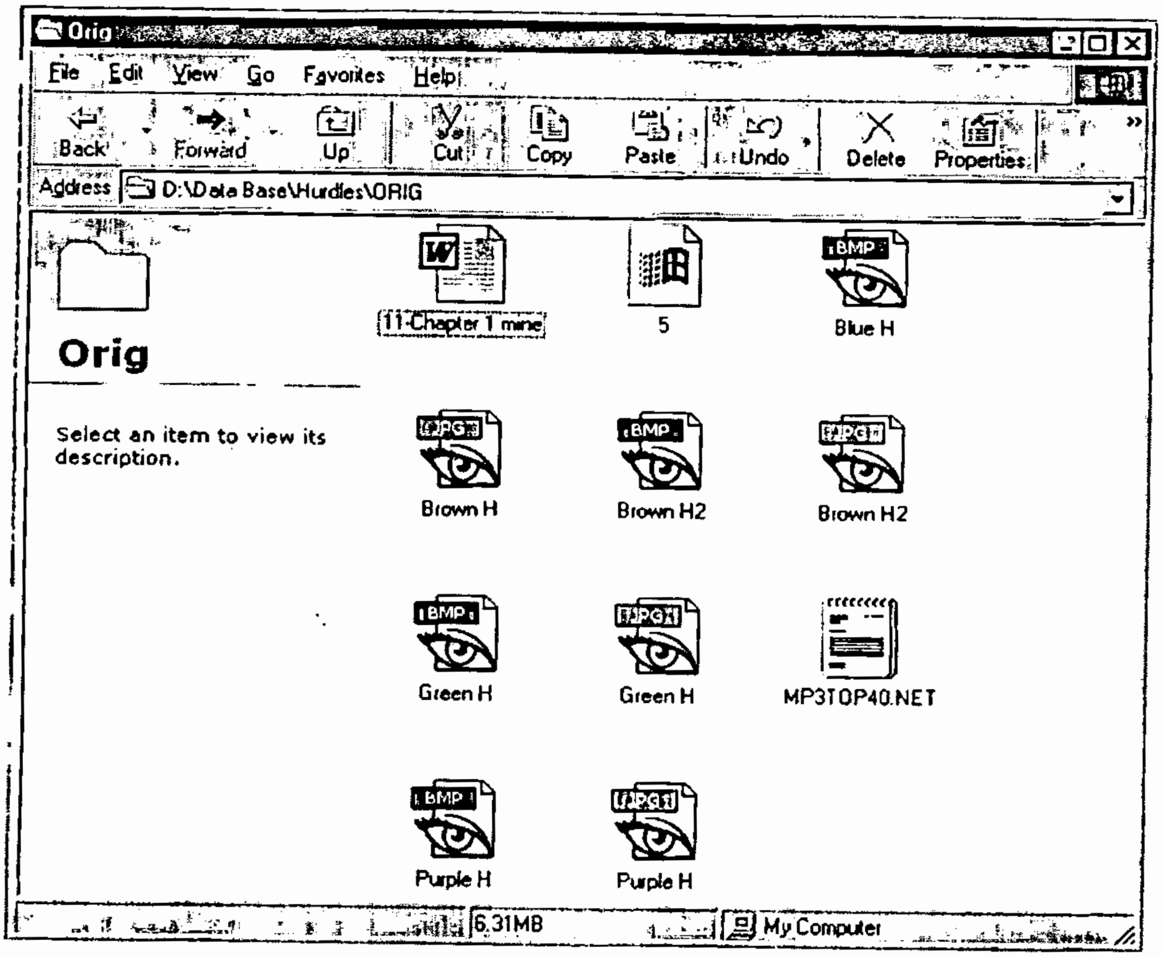

Orig
File Edit View Go Favorites Help
Back Forward Up Cut Copy Paste Undo Delete Properties
Address D:\Data Base\Hurdles\ORIG
Orig
Select an item to view its description.
11-Chapter 1 mine
5
Blue H
Brown H
Brown H2
Brown H2
Green H
Green H
MP3TOP40.NET
Purple H
Purple H
6.31MB
My Computer

Fig 5-5: Folder with different file formats

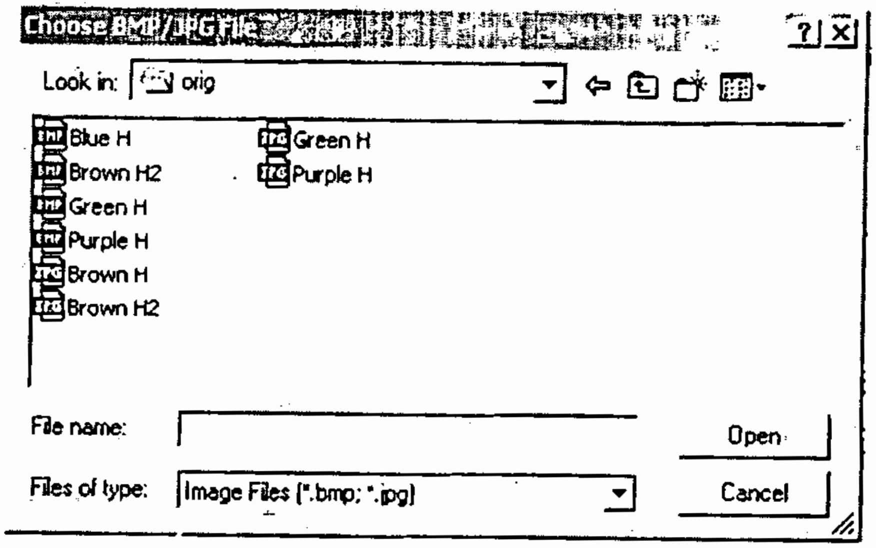


Fig 5-6: Only files with ".bmp" or ".jpg" extensions are allowed to be selected

5.5.6 Validating "Current Path" folder

Input Specification:

The folder "Current Path" should not be empty

Environmental Needs:

1. A Pentium® III or higher machine.
2. Windows 2000 (Family)
3. VC++ and OpenCV

Expected Output:

The program should prompt the user to empty "Current Path" folder, and should recheck the folder's status

Actual Output:

The program prompted the user to empty "Current Path" folder, and rechecked the folder's status

5.5.7 Navigation in Real-Time Mode with empty path

Input Specification:

Real-Time Processing option selection

Environmental Needs:

1. A Pentium® III or higher machine.
2. Windows 2000 (Family)
3. VC++ and OpenCV
4. At least one camera should be connected to the system

Expected Output:

The system should successfully navigate on the path and display result of navigation at the end.

Actual Output:

The system successfully navigated on the path and displayed result of navigation at the end.

5.5.8 Navigation in Real-Time Mode with hurdles in the path

Input Specification:

Real-Time Processing option selection

Environmental Needs:

1. A Pentium® III or higher machine.
2. Windows 2000 (Family)
3. VC++ and OpenCV
4. At least one camera should be connected to the system

Expected Output:

The system should successfully navigate on the path, should navigate around the hurdle and display result of navigation at the end.

Actual Output:

The system successfully navigated on the path, navigated around the hurdle and displayed result of navigation at the end.

5.5.9 Navigation in Post-Processing Mode with empty path

Input Specification:

Post - Processing option selection with a .avi file

Environmental Needs:

1. A Pentium® III or higher machine.
2. Windows 2000 (Family)
3. VC++ and OpenCV

Expected Output:

The system should successfully navigate on the path and display result of navigation at the end.

Actual Output:

The system successfully navigated on the path and displayed result of navigation at the end.

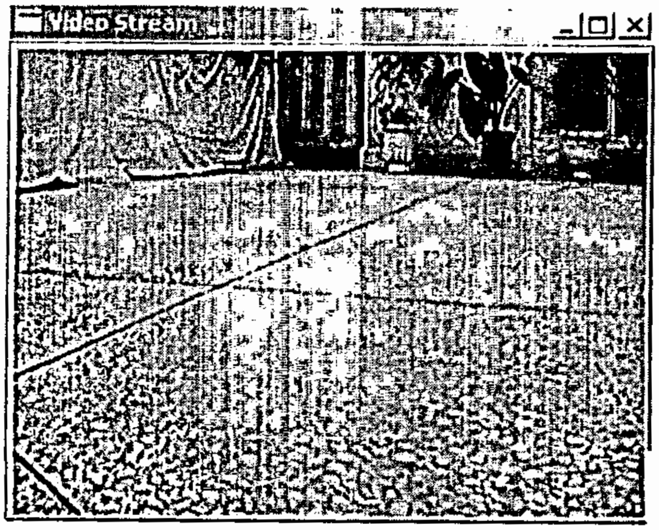


Fig 5-7: Video stream from .avi source

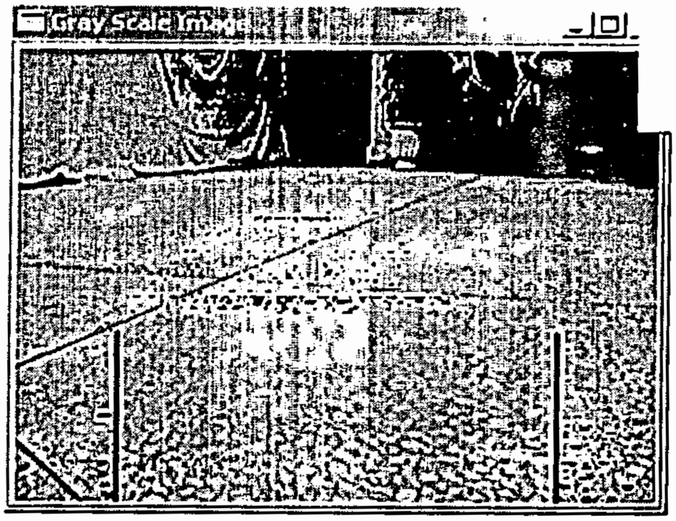


Fig 5-8: The gray-scale image with look-up table implementation

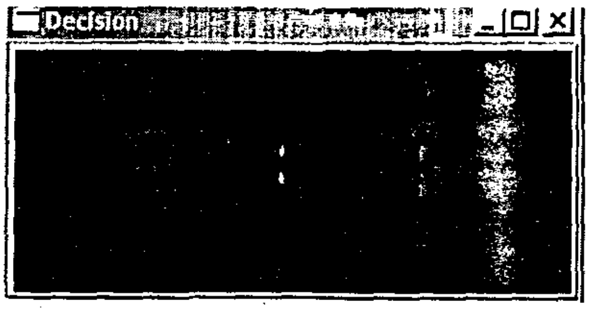
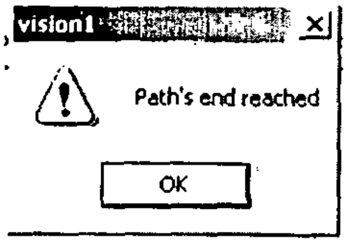


Fig 5-9: Decision made based on obstacle detection



Path's end reached

Fig 5-10: Message Displayed at the end of video stream

5.5.10 Navigation in Post-Processing Mode with hurdles in the path

Input Specification:

Post - Processing option selection with a .avi file

Environmental Needs:

1. A Pentium® III or higher machine.
2. Windows 2000 (Family)
3. VC++ and OpenCV

Expected Output:

The system should successfully navigate on the path, should navigate around the hurdle and display result of navigation at the end.

Actual Output:

The system successfully navigated on the path, navigated around the hurdle and displayed result of navigation at the end.

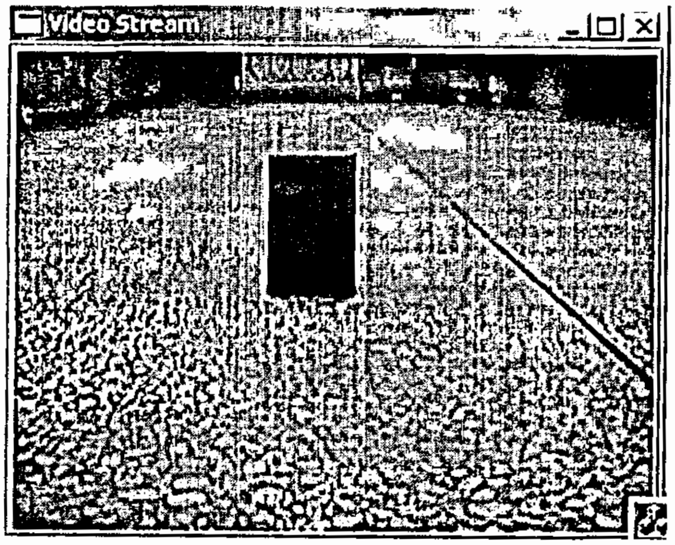


Fig 5-11: RGB image of a hurdle coming into Collision Course

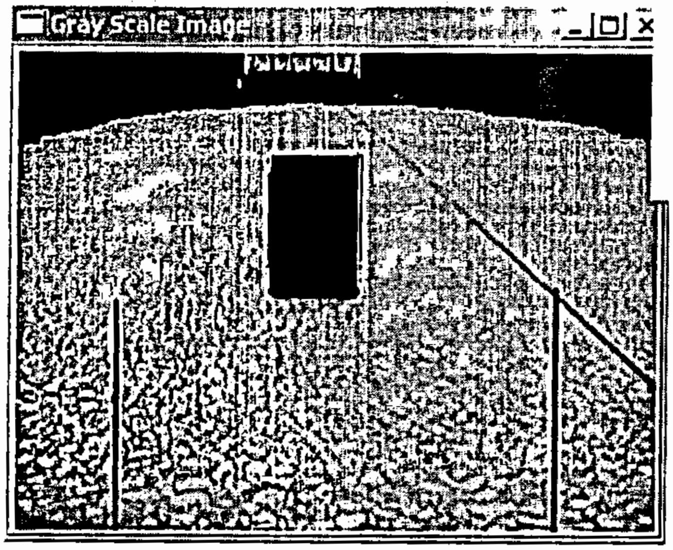


Fig 5-12: Gray image of a hurdle coming into Collision Course

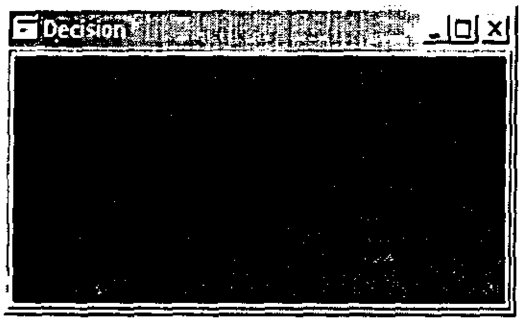


Fig 5-13: Warning message given to user

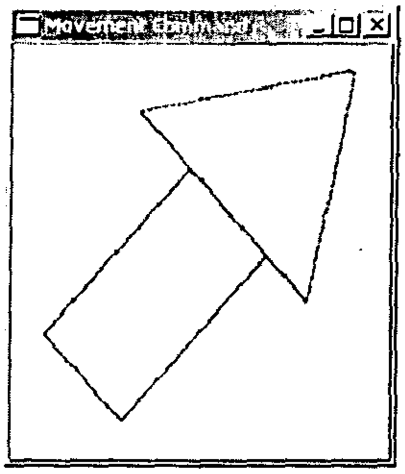


Fig 5-14: Suggested Hurdle avoidance course

5.6 Conclusion

In this chapter we have given detailed description of our system testing phase. In the beginning of this chapter, we have described various testing approaches. Next we have outlined features of the software system on which testing would be more emphasized. We have clearly described the course of action, input, expected output and actual output given by the system.

BIBLIOGRAPHY AND REFERENCES

1. Robotics and Machine Perception, The International Society for Optical Engineering (SPIE), International Technical Group Newsletter, March 2000 VOL. 9, NO. 1

2. Robotics and Machine Perception, The International Society for Optical Engineering (SPIE), International Technical Group Newsletter, August 2001 VOL. 10, NO. 2

3. Robotics and Machine Perception, The International Society for Optical Engineering (SPIE), International Technical Group Newsletter, March 2001 VOL. 10, NO. 1

4. A Path Following System for Autonomous Robots with Minimal Computing Power Andrew Thomson and Jacky Baltes Centre for Image Technology and Robotics, University of Auckland.

5. Obstacle Avoidance of Autonomous Mobile Robot using Stereo Vision Sensor, Masako Kumano Akihisa Ohya, Shin'ichi Yuta, Intelligent Robot Laboratory University of Tsukuba, Ibaraki, 305-8573 Japan.

6. Ground Plane Detection using Visual and Inertial Data Fusion, Jorge Lobo, Jorge Dias Institute of Systems and Robotics, Electrical Engineering Department, University of Coimbra, 3030 Coimbra, Portugal.

7. Obstacle Detection and Self-Localization without Camera Calibration Using Projective Invariants, Kyoung Sig Roh, Wang Heon Lee, In So Kweon, Dept. of Automation and Design Eng., Korea Advanced Institute of Science and Technology, 207-43, Cheongryangri-dong, Dongdaemoon-gu, Seoul, Korea.

8. Deriving and matching image fingerprint sequences for mobile robot localization, Autonomous System Labs, Carnegie Mellon, Diploma Work 1999-2000.

9. Obstacle Avoidance and Self-Localization System for Autonomous Vehicles, Muhammad Farooq Azam Khan, Dr. Sikander Hayat Khiyal, Department of Computer Science, Faculty of Applied Sciences, International Islamic University, Islamabad.

10. Computer Vision and Image Processing, A Practioner Approach using CVIPtools. Scott E. Umbagh.

11. Digital Image Processing. Rafael C. Gonzalez and Richard E. Woods. 1993 Prentice Hall Press.

12. Digital Image Processing Jan Teuber, 1992 Prentice Hall Press.

Research Paper

OBSTACLE AVOIDANCE AND SELF-LOCALIZATION SYSTEM FOR AUTONOMOUS VEHICLES

Muhammad Farooq Azam Khan,
Dr. Sikander Hayat Khiyal

Department of Computer Science,
Faculty of Applied Sciences, International Islamic University, Islamabad.

mfarooqsbox@yahoo.com, sikandar_hayat_khiyal@yahoo.com

Abstract: In this paper, we present a system for vision based autonomous navigation capable of obstacle avoidance as well as self-localization. At first a ground plane detection method is proposed which is capable of working on different most commonly occurring indoor/outdoor ground textures. The benefit of ground plane detection is that the system is not dependent on some specific property of obstacle to detect (such as size and orientation). Next, we describe our strategy for vehicle localization and implementation of proposed method. *Copyright © 2003 IFAC*

Keywords: Navigation system, path following, control, guidance, localization

1. INTRODUCTION

Self-localization, obstacle detection and avoidance are the basic requirements for successful navigation of any autonomous vehicle. Most of the vision based navigation systems concentrate on only one aspect of autonomous navigation. Some systems try to detect obstacles in their path while others try to find their relative position with respect to environment without the facility of obstacle avoidance.

The work by (Kumano, et al, 2000) uses stereo vision for obstacle detection. They use two monochrome CCD cameras equipped with about 90 degrees wide-angle lenses, which are fixed on the left and right side with the same height at the top of a robot. Two images are captured synchronously on an image processing board. One image is estimated from the other by the matrix calculated with relative positioning. If there is a certain difference of brightness, around there any 3D objects are detected. In their work (Roh, et al, 97) detect obstacles by comparing a pre-stored risk zone with a current risk zone. The positions of the detected obstacles are also determined by relative positioning.

Although a lot of work has been done in exploring the possibility of vision in autonomous navigation, much is left to be done to make it a part of our indoor/outdoor every day life. Goal of research in this field is to make vision based robots so reliable that they could take over the responsibility of trivial tasks (such as transporting files from one room to the other) and non-trivial tasks (like transporting hazardous materials). One way of enabling widespread use of autonomous vehicles is to make it economically sound by reducing the cost through stripping down the hardware required. For example, if the system can make use of the computing power of a personal computer, the data processing unit of the autonomous vehicle could be taken out. Instead we could load our navigation system on personal computer, make use of it and at other times use the computer for other purposes. Thus we can reduce the over all cost and increase the flexibility through vehicle size reduction.

This paper presents a vision-based path understanding and navigation technique without the need of exact camera calibration. We used two digital cameras mounted on top of a remote controlled vehicle and an ordinary personal computer for implementation and testing purposes. One camera is facing the front of the vehicle (used for navigation and obstacle detection), while the other is facing the near wall (for orientation and localization purposes). Figure 1 shows the arrangement of different devices in the system.

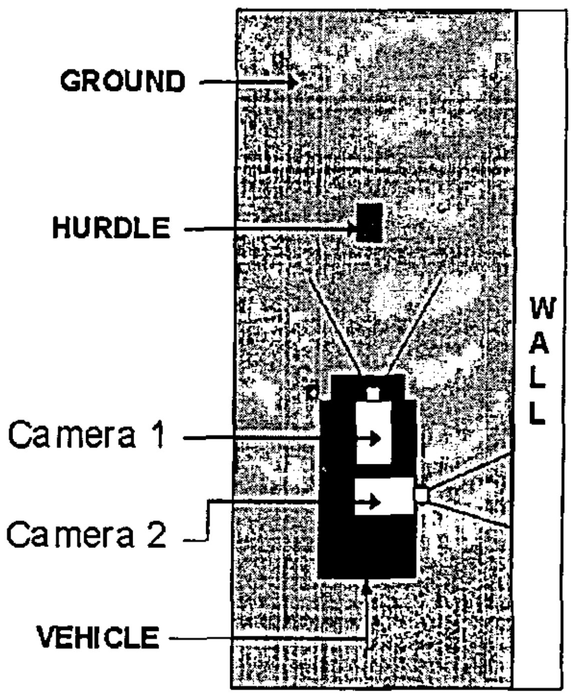


Fig. 1. Arrangement of different devices

For ground plane and obstacle detection part, the requirement for our system was to develop a specialized algorithm, light enough to be implemented in real time environment and accurate enough to detect any obstacles encountered. Nevertheless while reducing the computational load; there is a tradeoff of accuracy and time. The need was to find a balance in such a way as to allow computations to stay within real-time limit and still give accuracy to detect and navigate around obstacles.

For localization part, a method was needed that would understand the path, allowing intelligent decisions to be made to make navigation more efficient and less costly from both computation and accuracy point of views.

This paper is organized as follows: In Section 2 we present the method used for ground detection. Section 3 presents the method used for obstacle detection. Section 4 deals with navigation. Section 5 deals with how different paths are recognized and localization decisions are made. Implementation details are given in Section 6 and conclusion in section 7.

2. GROUND PLANE DETECTION

Since our objective was to develop system that could work on ordinary personal computer with no specialized equipment like frame grabber, the computational cost of processing each frame was a major issue. The computational power is also a consideration in selecting the image space to work in. In case of RGB image space, bits per pixel ratio of 24:1, thus ground plane/obstacle detection is more reliable but more computationally expensive. On the other hand in Gray scale image space, bits per pixel ratio of 8:1, thus ground plane/obstacle detection is less reliable but less computationally expensive. The ground plane detection algorithm was developed and tested on two different ground textures. One texture was quite simple while the other very complex. Figures 1 & 2 show the two textures under consideration. The texture in figure Fig. 2 was termed "plain-floor" while texture in Fig. 3 was termed as "chipsed-floor".

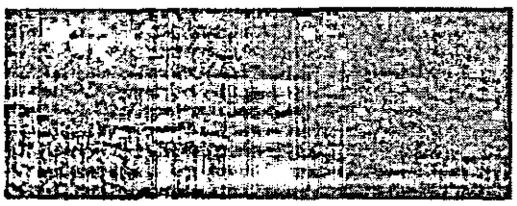


Fig. 2. Plain-floor

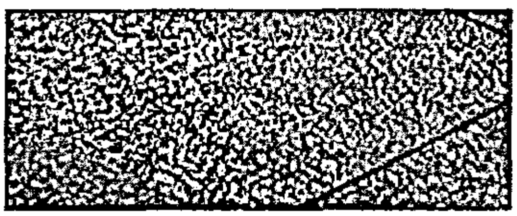


Fig. 3. Chipsed-floor

The idea behind these ground planes selection was that if the algorithm can work with relative accuracy on such planes, it could be expected to work on other indoor/outdoor plane textures such as roads (which has similar texture to that of plain-floor shown in Fig. 2). The ground plane detection algorithm works as follows:

2.1 <u>Look-Up Table Creation:</u>

First of all, a look-up table of 256 entries is created. This table is indexed with numbers ranging from 0 to 255. The value stored in the table can either be 0 or 1. The table is filled by finding the gray level pixel intensities that the ground plane can possibly take. A zero in the table indicates that the corresponding index value is not ground plane pixel intensity. While 1 in the table indicates that corresponding index value is ground plane pixel intensity. This could be done by creating a gray scale histogram of first few frames of "Critical Area" at the start of navigation (assumption being that those few frames do not have any obstacle in that specific area). "Critical Area" is the area within which any object is considered to be on a collision course. From that histogram we find pixel intensities that the ground plane takes on. Then we use Quick sort to sort the pixel intensities based on their frequency. This gives us intensities that occur most often in the ground plane. After sorting the intensities based on their frequency, we apply Quick sort once again, but this time a selected number of pixel intensities with top frequencies are sorted. The result after the second sorting is an arranged set of pixel intensities that are most likely to occur in a ground plane. By using this set, we mark contents of look-up table as one or zero. Our experiments with the plane-floor texture have shown that intensities values taken by the ground plane almost always lie within specific intensity region. In case of chipsed-floor, this region grows a lot more than that of the plane-floor. This is due to the fact that many stones of different pixel intensities are present on the ground plane. On creating a histogram of the ground texture, we consistently found some peak values of intensities in very close proximity, with other smaller peaks lying un-evenly on both sides (or one side in some cases) of these peak values.

2.2. <u>Look-Up Table Usage for Ground Plane Detection:</u>

Based on this look-up table, the system finds the pixels that are part of the ground. We achieve computational efficiency through the fact that the system needs to make only one comparison per pixel in order to decide whether it is ground plane or some obstacle/other environment feature. Figure 4 shows how areas within a frame are classified as ground plane.

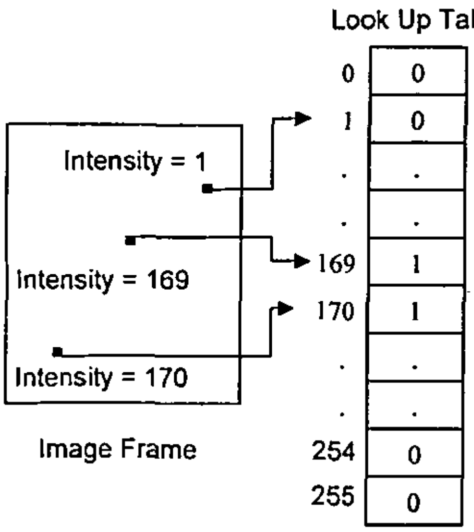


Fig. 4. Look-Up Table Usage. At every pixel, system accesses the contents of the Look-Up Table corresponding to the pixel intensity. If content of table is 1, that pixel is part of the ground, else it is not ground plane pixel.

Using the Look-Up Table, the system makes two iterative passes on every frame:

2.2.1- On the first pass, the system finds out the ground from the rest of the frame. This pass deals with the whole image. In this pass the look up table comparison is applied to the image frame. The objects not part of the floor are found based on their corresponding intensity level. Such objects/obstacles are marked by storing there pixel value as zero in the image, while the pixels classified as ground are left to their original intensity.

2.2.2- On the second pass, the system works on "Critical Area" which is much smaller area as compared to the whole image. Working of second pass can be divided into two parts.

2.2.2.1- First the small blackened patches are removed from the resultant image obtained from the first pass. These patches can be objects not big enough to be classified as obstacles. In case of chipsed-floor these patches are mostly due to small stones in the floor that were not present in the ground images used in creating look-up table (since even a large number of ground images can not assure that all possible stones in the chipsed-floor are taken into account).

2.2.2.2- Then the algorithm searches for any obstacles within the area that lie in course of vehicle motion and classifies them as potential obstacles.

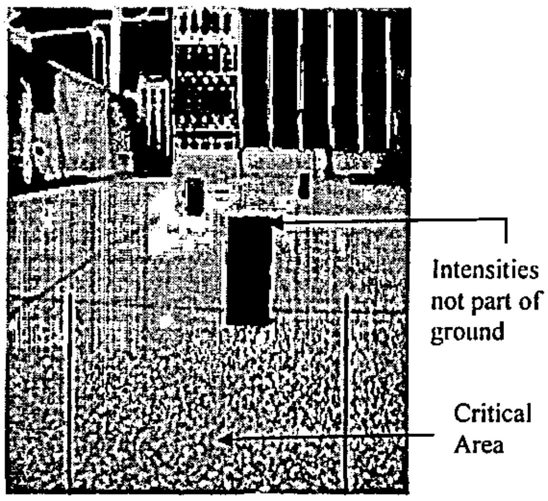


Fig. 5. Ground Plane Detection. Critical Area is the area between the two bars

Figure 5 shows the image frame after application of ground detection algorithm

Based on the results obtained from this floor detection module, information is passed to other system modules (navigation. object detection etc). These modules in turn make decisions such as moving left or right.

3. OBSTACLE DETECTION

Obstacles are detected on the basis of their relative size and pixel intensity. The dimensions of the "obstacle-to-seek" can be set to find obstacles of that or greater pixel size.

Since the obstacles are represented by pixel intensity of zero, the dimensions of the group of adjacent pixels having pixel intensity zero are counted, if it exceeds the pre-set obstacle dimension value, this group of pixels are considered to be part of an obstacle. In this way regions within the critical area are classified as obstacles.

If an obstacle is detected in a frame of video stream the first task is to find out its correct dimensions. In many cases, the obstacle is only a part of the entire obstacle, since it is quite possible for algorithm working on gray scale to miss-judge the correct obstacle dimensions. Hence on locating an obstacle, the system seeks to find the correct dimensions of the obstacle. The data structure used to store the information regarding the dimensions of the obstacle is such that we have access to the extreme left and right pixel coordinates of the obstacle in each row of image frame. By selecting appropriate intensity threshold, we move left and right one pixel at a time from left-most and right-most detected obstacle coordinates. At each pixel we calculate the intensity difference and find if the difference is greater than the threshold. On finding the difference greater than the threshold value, we store the previous pixel coordinate as the extreme obstacle coordinates. This task is carried out for each row containing the obstacle. This task of finding correct obstacle dimensions is necessary for safe navigation around the obstacle. Once an obstacle has been detected, it is tracked from one frame to the next as long as it is visible. Figure 6 shows an obstacle detected in the critical region of Figure 5.

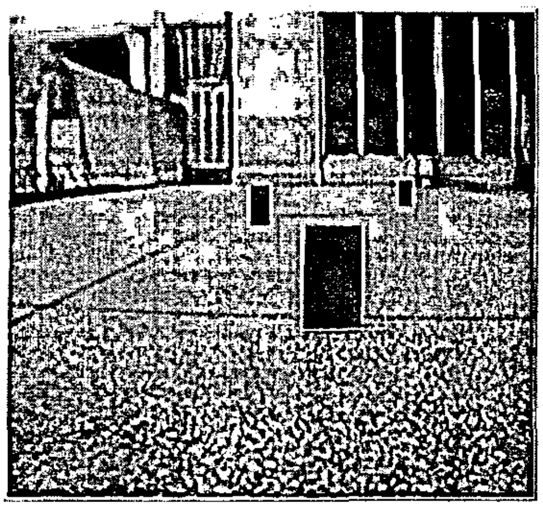


Fig. 6. Obstacle Detection. After finding the correct obstacle dimensions in gray scale, a rectangle is drawn around the obstacle in RGB image space.

4. NAVIGATION

The system provides on-screen instructions for navigation for example path clear/blocked, movement (forward/left/right etc). The system also provides instructions to navigate safely around any obstacle encountered. When an obstacle is encountered, the system finds on which side of the obstacle the vehicle can pass safely. If there is more space to maneuver on the left side of the obstacle, the system prompts for turning towards left.

Figure 7 & 8 show the navigational information given by the system in order to navigate around the obstacle in Figure 5.

Fig. 7. Path clear/blocked. System prompts that path in front of vehicle is blocked.

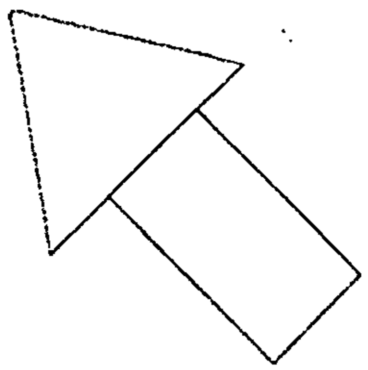


Fig. 8. Directional Command. Based on the free space available, the system prompts to move left to maneuver around the obstacle.

5. SELF-LOCALIZATION

Localization is an important task for autonomous navigation. In this regard many different techniques have been proposed. (Matsumoto, et al., 1996) proposed a model of the route, the "View-Sequenced Route Representation (VSRR)", for autonomous navigation. A VSRR consists of a sequence of view images, which have necessary information for localization, steering angle determination and obstacle detection. (Kosaka and Kak, 1992) implemented a system for a given environment using a CAD model based expectation map. This method constructed a complex database and required additional analysis for handling uncertainty.

In most vision-based approaches, databases for environments become complex because observed geometric properties are not invariant under the projective transformation. Thus, matching is also very complex and time consuming.

In many indoor/outdoor environments such as factories, university campuses, parking lots etc one aspect often seen is that different paths and corridors are almost identical. One good example could be the newly constructed International Islamic University, Islamabad campus, where there are different blocks which are duplicate copies of each other. Moreover the corridors within each block are also identical. In such environment the best localization system - the human localization system - also fails (new comers to the university can often be seen wandering around trying to find their destination). In such environments different sign boards are used to mark different paths and allow people to "localize" themselves (in our university's case you can only locate yourself by the name plates on the doors).

The work by (Pierre, et al, 01) proposed a method for mobile vehicle localization called "Finger Print Sequencing". According to this method, as the fingerprints of a person are unique, so are at each location the unique visual characteristics (save in pathological circumstances). So a unique virtual *fingerprint* of a location can be created. If locations are denoted by unique fingerprints in this manner, then the actual location of a mobile robot/vehicle *may be recovered by constructing a Fingerprint from* its current view and comparing to a database of known fingerprints. In the same way, our system uses different color codes (instead of sign-boards in case of humans) for self localization. The camera facing the wall scans frames to find color codes on the wall. Based on these color codes the system generates a three lettered code for each color patch encountered in a particular path and hence generates a string for each path. Each string starts with an "S" indicating start symbol and ends with an "E" indicating end symbol.

When the system is executed, it searches the database of different path folders it created on previous executions to extract information regarding the previous paths. Thus the system finds what color codes were encountered on a particular path and what was their sequence. While navigating on a particular path, the system continuously stores images of important events such as obstacles detection, color codes detection etc in a folder. The detected obstacles and color codes are sequentially numbered to take into account the sequence of occurrence.

The localization algorithm also works similarly to that of ground detection algorithm. In fact the algorithm shows the best results when applied to walls due to their even texture. First of all a Look-Up Table of wall is created in the similar manner as explained in case of ground detection algorithm. This results in color codes appearing as black, as shown in Figure 10. At the centre of image there are two vertical lines. The system seeks blacked out patches within these two lines. In order to reduce the computational load of camera facing the wall, the area between these two lines is kept as small as possible. The reason is that although localization is a very important task, but obstacle detection and avoidance still has a higher priority. On finding the blackened patch between these vertical lines, the system finds the correct dimensions of the color patch and extracts the patch. The top R, G and B channel values of the patch are extracted and compared with RGB channel values of pre-defined color codes. If a match is found, the corresponding color code is returned, else "unk" code is returned.

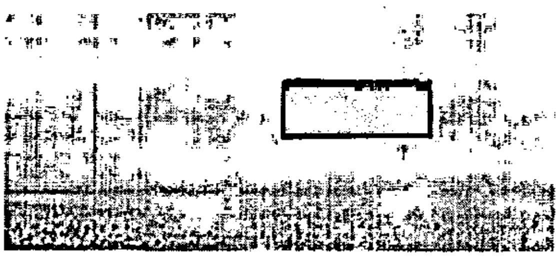


Fig. 9. Color patch on the wall.

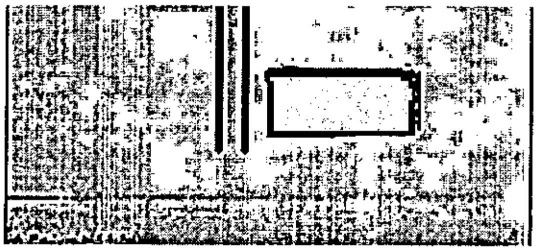


Fig. 10. Color Patch Detection. Color patch about to enter the detection region (the region between the two vertical lines).

6. IMPLEMENTATION

The system is implemented using VC++ 6.0 together with Intel's computer vision tool OpenCV (Beta 3.0). The system is tested using two ordinary digital cameras mounted on top of a remote controlled toy car. The platform used is Windows2000 on a 600 MHz AMD Athelon processor with 120 MB RAM.

The system is able to navigate successfully around different obstacles and correctly identify the path.

7. Conclusion

In this paper we have presented a path and obstacle detection method. It includes a floor plane detection algorithm. The method is computationally light enough to be implemented on an ordinary personal computer. The system is capable of Self-Localization using color codes on the wall.

REFRENCES

Kosaka, A. and A.C.Kak, (1992). "Fast Vision-Guided Mobile Robot Navigation Using Model-Based Reasoning and Prediction of Uncertainties," *CVGIP:Image Under.* Vol. 56, No.3, pp. 271~329.

Roh, K. S., W. H. Lee, I. S. Kweon (1997). "Obstacle Detection and Self-Localization without Camera Calibration Using Projective Invariants". *Proc of ICRA'97*
url: http://rcv.kaist.ac.kr/pub/papers/iros97.pdf

Pierre, L., (2001). "Deriving and matching image fingerprint sequences for mobile robot localization". *Proc of 2001 IEEE International Conference on Robotics and Automation, ICRA 2001.* Vol. 2, pp. 1609~1614.

Kumano, M., A. Ohya, S. Yuta, (2000). "Obstacle Avoidance of Autonomous Mobile Robot using Stereo Vision Sensor". *Proc. of 2nd International Symposium on Robotics and Automation,* pp. 497~502.

Matsumoto, Y., M. Inaba and H.Inoue, (1996). "Visual Navigation using View-Sequenced Route Representation," *Proc. of ICRA'96,* pp. 83~88.