**Chapter 1**

**PREAMBLE**

**1.1 Introduction**

During the last decades the risk of agricultural farms being accidentally damaged or defiled during routine animal intrusion has increased dramatically over the years.

Researches regarding animals in image processing have been an important field to numerous applications. Many algorithms and methods have been developed by human being in order to have a better understanding on animal behavior. Besides, these applications also can act as a warning system to human being from intrusion of dangerous wild animal for early precaution measures. These applications can be narrowed down to three main branches, namely detection, tracking and identification of animal.

The first branch, which is the animal detection, has been applied in various fields of real-life application. The second branch, which is the animal tracking, is the main topic in monitoring animal locomotive behavior and its interaction with the environment. By tracking the animal movements, it helps human to have a better understanding on how the animal interacts with its environment. The third branch, which is the animal identification, is used to identify the detected animal.

Existing approaches in animal detection are:

1. **Researches on Animal Detection by Human Eyes:** Animal detection by human eyes has been considered as the most reliable detection method if seen from the computational point of view. This is because the image structure in natural images is complex. It is found that a human observer is able to decide whether a briefly flashed animal scene contain an animal as fast as 150ms.

Disadvantages:

* Human eyes can easily get tired causing decreasing of effectiveness.
* Human eyes cannot work 24 hours a day to perform animal detection.

2. **Researches on Power Spectra:** The researchers also have tried to find whether the presence of animal in the image scene will change the power spectral of the image or not. This is constructed by transforming the images from spatial domain into the frequency domain, by using transformation function such as the Fourier transform.

Disadvantages:

* Human observer will not prefer to use this approach if they want to quickly detect the animal.

3. **Animal Detection Using Face Detection Approach:** The animal faces are measured by utilizing face detection method with different local contrast configuration of luminescence channel to detect the image region of animal faces.

Disadvantages:

* The video capture by human might scare off the animals and might lead to inability of detection the type of animal.

4. **Animal Detection Based on Thresholding Segmentation Method**: The object is found by using threshold segmentation method after obtaining the background image.

Disadvantages:

* Different appropriate threshold should be chosen for different background scene.
* It is very difficult to set the precise threshold for segmentation method.

**1.2 Existing System:**

Existing system provides an environmentally friendly, all-weather surveillance system for accurate and reliable large animal intrusion detection and tracking.

* The system includes multiple solar-powered sensor stations, consisting of 360-degree coverage radars for detection and tracking of animals within the monitored area and processing units that are used to collect and process data from the sensors.
* These units also report events and transmit alarms to local or remote sites/devices (e.g. flashing beacons) through wireless communication.
* Warnings are switched off automatically soon after the animals leave the area. It filters out detections of vehicles and smaller animals (e.g. skunks, raccoons and birds) as well as detections beyond the monitored area to ensure that only large animal detections within the monitored area will activate the warnings.

**1.2.1 Limitations of the Existing system**

* The system is always active and hence results in power inefficiency i.e. even when there is no animal around the selected area of interest the system must be active.
* It doesn’t give the type of animal that has entered the farm.
* It is highly expensive to implant the system onto the farm.
* It doesn’t give precise of the real time position of the animal that has intruded.

**1.3 Problem Statement**

 **Crop vandalization** is becoming a very common phenomenon in almost all the crop yielding states. Wild animals like monkeys, estray animals especially cows and buffaloes, wild dogs, nilgais, bisons, elephant’s deer, wild pigs and even birds like parakeets cause a lot of damage to crops either by running over them or eating them and vandalizing them completely. This leads **to poor yield of crops**. These animals attack on fruit orchards and destroy the flowerings and fruits. In both cases, this leads to **significant financial loss** to the farmers and orchard owners. The problem is so pronounced that sometimes farmers decide to leave the area barren due to these animal attacks.

* 1. **Objective**

To detect an intrusion in a farm and to notify the relevant personnel with the live geographic location and the visuals of the intruder that provides the person to react in time and prevent any damage caused to the farm.

**1.5 Proposed System**

The proposed approach uses a set of motion sensors, whose count depends upon the area of the farm, which triggers the selective cameras when the sensors detect a motion of an animal or a human. The camera captures the visuals of the moving animal which and tracks the moving object into the farm. Additionally, it transmits the visuals to MQTT server for image processing to identify the object of intrusion. Once the moving object is identified to be an animal, the intruder (animal) along with it’s geographic location (WITHOUT EMPLOYING A GPS SENSOR) in the farm is sent from server to the output display that is developed over OpenGL library.

**1.5.1 Advantages of Proposed System**

* **It is power efficient as the system is triggered only if there is any animal detected within the farm. Until then the system is off.**
* **It is cost efficient as the system consists of only a desired and limited number of sensors and rest of the task is delegated to the cameras.**
* **It gives the precise geographic location of the object within the farm.**
* **It tracks the path followed by the animal so as to locate it’s next location in the farm. This makes the tracking faster as the animal’s path is obtained beforehand with 80% accuracy.**
* Uses OpenCV, which is an open-source platform and has faster processing power due to better language integration.
* Vision-based approach reduces complexity of existing methods.
* The real time tracked path is displayed live onto the monitor of the person in charge.

**1.6 Phase Description**

* **Table 1.6: Phase Description**

|  |  |  |
| --- | --- | --- |
| **Phase** | **Task** | **Description** |
| Phase 1 | Analysis | Analyzing the core of the IEEE paper and provide Literature review based on analysis. |
| Phase 2 | Literature survey | Collect raw data and elaborate on literature surveys. |
| Phase 3 | System analysis | Analyses the requirements of the project and lists the specific requirements needed. |
| Phase 4 | Design | Object designing and Functional description |
| Phase 5 | Implementation | Implement the code based on the object specification |
| Phase 6 | Testing | Test the project according to Test Specification |
| Phase 7 | Documentation | Prepare the document for this project with conclusion and future enhancement. |

**1.7 Organization of the project report**

The project report is organized as follows:

**Chapter 2: Literature Review** - Gives a brief overview of the survey papers and the research sources that have been studied to establish a thorough understanding of the project under consideration.

**Chapter 3 Theoretical Background** - Establishes groundwork for the proposed project by giving a detailed analysis of the project topic, existing research relevant to the project, arguments in favor and against the existing solutions and finally explores the motivation behind the proposed solution.

**Chapter 4 System Requirement Specification** - Discusses in details about the different kinds of requirements needed to successfully complete the project.

**Chapter 5 System Analysis** - gives details about several analysis that are performed to facilitate taking decision of whether the project is feasible enough or not.

**Chapter 6** **System** **Design -** Gives the design description of the project, conceptual and detailed design well supported with design diagrams.

**Chapter 7 Implementation** - Discusses the implementation details of the project and reasons the use of the programming language and development environment.

**Chapter 8 Testing -** Briefs the testing methods used for testing the different modules in the project.

**Chapter 9 Results and Performance Analysis -** Gives the snapshots and graphs of the proposed protocols.

**Chapter 10 Conclusion and Future Scope -** Gives the concluding remarks of the project, throwing light on its future aspects.

**References** Lists the websites and references referred during the project work.

**Chapter 2**

**LITERATURE SURVEY**

Literature survey is mainly carried out in order to analyze the background of the current project which helps to find out flaws in the existing system and guides on which unsolved problems we can work out. So, the following topics not only illustrate the background of the project but also uncover the problems and flaws which motivated to propose solutions and work on this project.

***[1] “Li, L.; Huang, W.; Gu, I.; Tian, Q. Statistical modeling of complex backgrounds for foreground***

***object detection. IEEE Trans. Image Process. 2004, 13, 1459–1472.”***

This presents a region-based method for background subtraction. It relies

on color histograms, texture information, and successive division of candidate rectangular

image regions to model the background and detect motion. Our proposed algorithm uses

this principle and combines it with Gaussian Mixture background modeling to produce

a new method which outperforms the classic Gaussian Mixture background subtraction

method. Our method has the advantages of filtering noise during image differentiation and

providing a selectable level of detail for the contour of the moving shapes. The algorithm

is tested on various video sequences and is shown to outperform state-of-the-art background

subtraction methods.

***[2] “***L. Agapito, R. Hartley, and E. Hayman. Linear calibration of a rotating and zooming camera. In

*Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, pages 15–21, 1999.”

This is a method for PTZ camera auto-calibration over the camera’s zoom range.

This method is based on the minimization of re-projection errors of feature points detected in image captured by the camera at different orientations and zoom levels. Results obtained over both synthetic and real data show that a full zoom range, complete field of view, pan-tilt-zoom camera calibration is possible.

Also in this work, a simulator capable of generating highly flexible, real data only, test scenarios with multiple events having ground truth motion is proposed.

***[3] “ Proceedings of the International MultiConference of Engineers and Computer Scientists 2013 Vol I, IMECS 2013, March 13 - 15, 2013, Hong Kong”***

For the moving object detection part, we have combined the Wroskian’s change detection method to detect the moving pixels and refine the result by utilizing the neighbor pixels concept to reduce the noise resulted from imperfect alignment of successive images. The detection noise is further reduced by analyzing the detection consistency across successive image frames. This approach has effectively detects the moving object while reducing the noise.

***[4] “International Journal of Innovative Research in Computer and Communication Engineering (An ISO 3297: 2007 Certified Organization) Vol. 2, Issue 2, February 2014”***

The goal of object tracking is segmenting a region of interest from a video scene and keeping track of its motion, positioning and occlusion. The object detection and object classification are preceding steps for tracking an object in sequence of images. Object detection is performed to check existence of objects in video and to precisely locate that object. Then detected object can be classified in various categories such as humans, vehicles, birds, floating clouds, swaying tree and other moving objects. Object tracking is performed using monitoring objects’ spatial and temporal changes during a video sequence, including its presence, position, size, shape, etc. Object tracking is used in several applications such as video surveillance, robot vision, traffic monitoring, Video inpainting and Animation. This

paper presents a brief survey of different object detection, object classification and object tracking algorithms available in the literature including analysis and comparative study of different techniques used for various stages of tracking.

**Chapter 3**

# THEORETICAL BACKGROUND

Theoretical background highlighting some topics related to project work. The description contains several topics which are worth to discuss and also highlight some of their limitation that encourage going on finding solution as well as highlights some of their advantages for which reason these topics and their features are used in this project.

**3.1 Introduction to OpenCV**

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. It has C++, C, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. It also provides basic data structures for image processing with efficient optimizations.

**3.2 History of OpenCV**

Officially launched in 1999, the OpenCV project was initially an [Intel Research](https://en.wikipedia.org/wiki/Intel_Research_Lablets) initiative to advance [CPU](https://en.wikipedia.org/wiki/Central_processing_unit)-intensive applications, part of a series of projects including [real-time](https://en.wikipedia.org/wiki/Real-time_computing) [ray tracing](https://en.wikipedia.org/wiki/Ray_tracing_(graphics)) and [3D display](https://en.wikipedia.org/wiki/3D_Display) walls. The main contributors to the project included a number of optimization experts in Intel Russia, as well as Intel’s Performance Library Team. In the early days of OpenCV, the goals of the project were described as:

* Advance vision research by providing not only open but also [optimized code](https://en.wikipedia.org/wiki/Code_optimization) for basic vision infrastructure.
* Disseminate vision knowledge by providing a common infrastructure that developers could build on, so that code would be more readily readable and transferable.
* Advance vision-based commercial applications by making [portable](https://en.wikipedia.org/wiki/Portability_(computer_science)), performance-optimized code available for free—with a license that did not require to be open or free themselves.

The first alpha version of OpenCV was released to the public at the [IEEE Conference on Computer Vision and Pattern Recognition](https://en.wikipedia.org/wiki/Conference_on_Computer_Vision_and_Pattern_Recognition) in 2000, and five betas were released between 2001 and 2005. The first 1.0 version was released in 2006. In mid-2008, OpenCV obtained corporate support from [Willow Garage](https://en.wikipedia.org/wiki/Willow_Garage), and is now again under active development. A version 1.1 "pre-release" was released in October 2008.

The second major release of the OpenCV was on October 2009. OpenCV 2 includes major changes to the [C++](https://en.wikipedia.org/wiki/C%2B%2B) interface, aiming at easier, more type-safe patterns, new functions, and better implementations for existing ones in terms of performance (especially on multi-core systems). Official releases now occur every six months and development is now done by an independent Russian team supported by commercial corporations.

In August 2012, support for OpenCV was taken over by a non-profit foundation OpenCV.org, which maintains a developer and user site.

* 1. **Features of OpenCV**
* **Color space Conversion**

A color space is a specific organization of [colors](https://en.wikipedia.org/wiki/Color). Colorspace conversion deals with converting an image that is represented in one color space to another color space, the goal being to make the translated image look as similar as possible to the original.

* **Geometric Transformation of Images**

OpenCV can be used to apply different geometric transformations to images like translation, rotation, affine transformation etc.

* **Image Thresholding**

This method is used to convert a greyscale image to Binary image. OpenCV provides various thresholding methods like Simple -thresholding, Adaptive thresholding, Otsu’s thresholding etc.

* **Smoothing Images**

It refers to an approximating function that attempts to capture important patterns in the data, while leaving out noise or other fine-scale structures/rapid phenomena. OpenCV provides various methods like Averaging and Gaussian filtering to smoothen images.

* **Morphological Transformations**

Morphological transformation are some simple operations based on the image shape. It is normally performed on binary images. It needs two inputs, one is out original image, second is called structuring element or kernel which decides the nature of operation. Two basic morphological operators are Erosion and Dilation

* **Image Segmentation**

Image segmentation is the process of partitioning a [digital image](https://en.wikipedia.org/wiki/Digital_image) into multiple segments ([sets](https://en.wikipedia.org/wiki/Set_(mathematics)) of [pixels](https://en.wikipedia.org/wiki/Pixel), also known as superpixels). Image segmentation is typically used to locate objects and [boundaries](https://en.wikipedia.org/wiki/Boundary_tracing) (lines, curves, etc.) The result of image segmentation is a set of segments that collectively cover the entire image, or a set of [contours](https://en.wikipedia.org/wiki/Contour_line) extracted from the image (see [edge detection](https://en.wikipedia.org/wiki/Edge_detection)).

* **Edge Detection**

Edge detection is the name for a set of mathematical methods which aim at identifying points in a [digital image](https://en.wikipedia.org/wiki/Digital_image) at which the [image brightness](https://en.wikipedia.org/wiki/Luminous_intensity) changes sharply or, more formally, has discontinuities. The points at which image brightness changes sharply are typically organized into a set of curved line segments termed *edges*.

* **Object Tracking**

The objective of video tracking is to associate target objects consecutive video frames. It has a variety of uses, some of which are: in human-computer interaction, security and surveillance, video communication and compression, augmented reality, traffic control, medical imaging and video editing.

**3.4 Applications of OpenCV**

In the recent past, OpenCV has found its way into a wide variety of applications and systems with vastly varying requirements and characteristics.

* 2D and 3D feature toolkits
* [Ego motion](https://en.wikipedia.org/wiki/Egomotion) estimation (estimating the 3D motion of a camera relative to a rigid scene)
* [Facial recognition system](https://en.wikipedia.org/wiki/Facial_recognition_system)
* [Gesture recognition](https://en.wikipedia.org/wiki/Gesture_recognition)
* [Human–computer interaction](https://en.wikipedia.org/wiki/Human%E2%80%93computer_interaction) (HCI)
* [Mobile robotics](https://en.wikipedia.org/wiki/Mobile_robotics)
* [Motion understanding](https://en.wikipedia.org/w/index.php?title=Motion_understanding&action=edit&redlink=1)
* [Object identification](https://en.wikipedia.org/w/index.php?title=Object_identification&action=edit&redlink=1)
* [Segmentation](https://en.wikipedia.org/wiki/Segmentation_(image_processing)) and recognition
* [Stereopsis](https://en.wikipedia.org/wiki/Stereopsis) stereo vision: depth perception from 2 cameras
* [Structure from motion](https://en.wikipedia.org/wiki/Structure_from_motion) (SFM)
* [Motion tracking](https://en.wikipedia.org/wiki/Video_tracking)
* [Augmented reality](https://en.wikipedia.org/wiki/Augmented_reality)

**3.5 Introduction to OpenGL**

Open Graphics Library (OpenGL) is a [cross-language](https://en.wikipedia.org/wiki/Language-independent_specification), [cross-platform](https://en.wikipedia.org/wiki/Cross-platform) [application programming interface](https://en.wikipedia.org/wiki/Application_programming_interface) (API) for rendering [2D](https://en.wikipedia.org/wiki/2D_computer_graphics) and [3D](https://en.wikipedia.org/wiki/3D_computer_graphics) [vector graphics](https://en.wikipedia.org/wiki/Vector_graphics). The API is typically used to interact with a [graphics processing unit](https://en.wikipedia.org/wiki/Graphics_processing_unit) (GPU), to achieve [hardware-accelerated](https://en.wikipedia.org/wiki/Hardware_acceleration) [rendering](https://en.wikipedia.org/wiki/Rendering_(computer_graphics)).

[Silicon Graphics](https://en.wikipedia.org/wiki/Silicon_Graphics) Inc., (SGI) started developing OpenGL in 1991 and released it in January 1992 applications use it extensively in the fields of [computer-aided design](https://en.wikipedia.org/wiki/Computer-aided_design) (CAD), [virtual reality](https://en.wikipedia.org/wiki/Virtual_reality), [scientific visualization](https://en.wikipedia.org/wiki/Scientific_visualization), information visualization, [flight simulation](https://en.wikipedia.org/wiki/Flight_simulator), and [video games](https://en.wikipedia.org/wiki/Video_game). OpenGL is managed by the [non-profit](https://en.wikipedia.org/wiki/Non-profit_organization) [technology consortium](https://en.wikipedia.org/wiki/Technology_consortium) [Khronos Group](https://en.wikipedia.org/wiki/Khronos_Group).

**3.6 History of OpenGL**

In the 1980s, developing software that could function with a wide range of graphics hardware was a real challenge. Software developers wrote custom interfaces and drivers for each piece of hardware. This was expensive and resulted in multiplication of effort.

By the early 1990s, [Silicon Graphics](https://en.wikipedia.org/wiki/Silicon_Graphics) (SGI) was a leader in 3D graphics for workstations. Their [IRIS GL](https://en.wikipedia.org/wiki/IRIS_GL) API was considered state-of-the-art and became the de facto industry standard, overshadowing the open standards-based [PHIGS](https://en.wikipedia.org/wiki/PHIGS). This was because IRIS GL was considered easier to use, and because it supported [immediate mode](https://en.wikipedia.org/wiki/Immediate_mode_(computer_graphics)) rendering. By contrast, PHIGS was considered difficult to use and outdated in functionality.

SGI's competitors (including [Sun Microsystems](https://en.wikipedia.org/wiki/Sun_Microsystems), [Hewlett-Packard](https://en.wikipedia.org/wiki/Hewlett-Packard) and [IBM](https://en.wikipedia.org/wiki/IBM)) were also able to bring to market 3D hardware, supported by extensions made to the PHIGS standard. This in turn caused SGI market share to weaken as more 3D graphics hardware suppliers entered the market. In an effort to influence the market, SGI decided to turn the IrisGL API into an open standard – **OpenGL**.

However, SGI had many software customers for whom the change from IrisGL to OpenGL would demand significant investment. Moreover, IrisGL had API functions that were irrelevant to 3D graphics. For example, it included a windowing, keyboard and mouse API, in part because it was developed before the [X Window System](https://en.wikipedia.org/wiki/X_Window_System) and Sun's [NeWS](https://en.wikipedia.org/wiki/NeWS). And, IrisGL libraries were unsuitable for opening due to licensing and patent issues. These factors required SGI to continue to support the advanced and proprietary [Iris Inventor](https://en.wikipedia.org/wiki/Open_Inventor) and [Iris Performer](https://en.wikipedia.org/wiki/Iris_Performer) programming APIs while market support for OpenGL matured.

One of the restrictions of IrisGL was that it only provided access to features supported by the underlying hardware. If the graphics hardware did not support a feature, then the application could not use it. OpenGL overcame this problem by providing support in software for features unsupported by hardware, allowing applications to use advanced graphics on relatively low-powered systems.

In 1992, SGI led the creation of the [OpenGL Architecture Review Board](https://en.wikipedia.org/wiki/OpenGL_Architecture_Review_Board) (OpenGL ARB), the group of companies that would maintain and expand the OpenGL specification in the future.

In 1994, SGI played with the idea of releasing something called "[OpenGL++](https://en.wikipedia.org/wiki/OpenGL%2B%2B)" which included elements such as a scene-graph API (presumably based on their [Performer](https://en.wikipedia.org/wiki/Performer_(Computer_Graphics_API)) technology). The specification was circulated among a few interested parties – but never turned into a product

[Microsoft](https://en.wikipedia.org/wiki/Microsoft) released [Direct3D](https://en.wikipedia.org/wiki/Direct3D) in 1995, which eventually became the main competitor of OpenGL. On December 17, 1997 Microsoft and SGI initiated the [Fahrenheit](https://en.wikipedia.org/wiki/Fahrenheit_graphics_API) project, which was a joint effort with the goal of unifying the OpenGL and Direct3D interfaces (and adding a scene-graph API too). In 1998, Hewlett-Packard joined the project. It initially showed some promise of bringing order to the world of interactive 3D computer graphics APIs, but on account of financial constraints at SGI, strategic reasons at Microsoft, and general lack of industry support, it was abandoned in 1999.

In July 2006 the OpenGL Architecture Review Board voted to transfer control of the OpenGL API standard to the Khronos Group.[[](https://en.wikipedia.org/wiki/OpenGL#cite_note-18)

**3.7 Features of OpenGL**

Deprecated features include:

* All fixed-function vertex and fragment processing.
* Direct-mode rendering, using glBegin and glEnd.
* Display lists.
* Indexed-color rendering targets.
* OpenGL Shading Language versions 1.10 and 1.20.

**Summary**

This chapter mainly concentrates on the basic theoretical background related to the topic of focus. It gives information about the platform on which this application has been developed and discussed about the Developing environment. Applications of OpenCV are also discussed in this chapter.

**Chapter 4**

**SYSTEM REQUIREMENT SPECIFICATION**

Software requirement Specification is a fundamental document, which forms the foundation of the software development process. It not only lists the requirements of a system but also has a description of its major feature

**4.1 Functional Requirement**

Functional Requirement defines a function of a system and how the system must behave when presented with specific inputs or conditions. These may include calculations, data manipulation and processing and other specific functionality. In this system following are the functional requirements:

* Input test case must not have compilation and runtime errors.
* The system must not stop working when kept running for even a long time.
* The system must function as expected for every set of test case provided.
* The system should generate the output for given input test case and input parameters.
* The system should generate on-demand services.

**4.2 Non Functional Requirement**

Non-functional requirements are the requirements which are not directly concerned with the specific function delivered by the system. They specify the criteria that can be used to judge the operation of a system rather than specific behaviors. They may relate to emergent system properties such as reliability, response time and store occupancy. Non-functional requirements arise through the user needs, because of budget constraints, organizational policies, the need for interoperability with other software and hardware systems or because of external factors such as:

* Product Requirements
* Organizational Requirements
* User Requirements
* Basic Operational Requirements

In systems engineering and requirements engineering, a non-functional requirement is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors. This should be contrasted with functional requirements that define specific behavior or functions. The plan for implementing non-functional requirements is detailed in the system architecture. Broadly, functional requirements define what a system is supposed to do and non- functional requirements define how a system is supposed to be. Functional requirements are usually in the form of “system shall do <requirement>”, an individual action of part of the system, perhaps explicitly in the sense of a mathematical function, a black box description input, output, process and control functional model or IPO Model. In contrast, non-functional requirements are in the form of “system shall be <requirement>”, an overall property of the system as a whole or of a particular aspect and not a specific function. The systems' overall properties commonly mark the difference between whether the development project has succeeded or failed.

The non-functional requirements of our project are:

* Response time – The time the system takes to load and the time for responses on any action the user does.
* Processing time - How long is acceptable to perform key functions or export / import data?
* Throughput – The number of operations the system needs to handle must be kept in mind.
* Storage - The amount of data to be stored for the system to function.
* Locations of operation - Geographic location, connection requirements and the restrictions of a local network prevail.
* Architectural Standards – The standards needed for the system to work and sustain.

**4.2.1 Product Requirements**

* **Portability:** Since the SLR system is designed to run using OpenCV (whose library is written in C), the system is portable**.**
* **Correctness:** It follows a well-defined set of procedures and rules to compute and also rigorous testing is performed to confirm the correctness of the data.
* **Ease of Use:** The front end is designed in such a way that it provides an interface which allows the user to interact in an easy manner.
* **Modularity:** The complete product is broken up into many modules and well-defined interfaces are developed to explore the benefit of flexibility of the product.
* **Robustness:** This software is being developed in such a way that the overall performance is optimized and the user can expect the results within a limited time with utmost relevancy and correctness.

Non-functional requirements are also called the qualities of a system. These qualities can be divided into execution quality & evolution quality. Execution qualities are security & usability of the system which are observed during run time, whereas evolution quality involves testability, maintainability, extensibility or scalability

**4.2.2 Organizational Requirements**

**Process Standards:** IEEE standards are used to develop the application which is the standard used by the most of the standard software developers all over the world.

**Design Methods:** Design is one of the important stages in the software engineering process. This stage is the first step in moving from problem to the solution domain. In other words, starting with what is needed design takes us to work how to satisfy the needs.

**4.2.3 User Requirements**

The user requirements document (URD) or user requirements specification is a document usually used to software engineering that specifies the requirements the user expects from software to be constructed in a software project. Once the required information is completely gathered it is documented in a URD, which is meant to spell out exactly what the software must do and becomes part of the contractual agreement. A customer cannot demand feature not in the URD, whilst the developer cannot claim the product is ready if it does not meet an item of the URD.

The URD can be used as a guide to planning cost, timetables, milestones, testing etc. The explicit nature of the URD allows customers to show it to various stakeholders to make sure all necessary features are described.

Formulating a URD requires negotiation to determine what is technically and economically feasible. Preparing a URD is one of those skills that lies between a science and economically feasible. Preparing a URD is one of those skills that lies between a science and an art, requiring both software technical skills and interpersonal skills.

**4.2.4 Basic Operational Requirements**

Operational requirement is the process of linking strategic goals and objectives to tactic goals and objectives. It describes milestones, conditions for success and explains how, or what portion of, a strategic plan will be put into operation during a given operational period, in the case of, a strategic plan will be put into operation during a given operational period, in the case of commercial application, a fiscal year or another given budgetary term. An operational plan is the basis for, and justification of an annual operating budget request. Therefore, a five-year strategic plan would typically require five operational plans funded by five operating budgets.

Operational plans should establish the activities and budgets for each part of the organization for the next 1-3 years. They link the strategic plan with the activities the organization will deliver, and the resources required to deliver them.

An operational plan draws directly from agency and program strategic plans to describe agency and program missions and goals, program objectives, and program activities. Like a strategic plan, an operational plan addresses four questions:

* Where are we now?
* Where do we want to be?
* How do we get there?

The customers are those that perform the eight primary functions of systems engineering, with special emphasis on the operator as the key customer. Operational requirements will define the basic need and, at a minimum, will be related to these following points:

**Mission profile or scenario:** It describes about the procedures used to accomplish mission objective. It also finds out the effectiveness or efficiency of the system.

**Performance and related parameters:** It points out the critical system parameters to accomplish the mission

**Utilization environments:** It gives a brief outline of system usage. Finds out appropriate environments for effective system operation.

**Operational life cycle:** It defines the system lifetime

**4.3 Hardware Requirements**

* NodeMCU **:** Microcontroller with inbuilt WIFI shield
* Motion Sensor : **:** PIR
* Webcam **:** 2MP and above
* Processors **:** Pentium IV and above
* Processor Speed **:** 3.00 GHZ
* RAM **:** 2 GB
* Storage **:** 20 GB
* Monitor **:**  15 inches
* Keyboard **:**  Standard 102 keys
* Mouse  **:** Standard 3 buttons

**4.4 Software Requirements**

* Operating system : Ubuntu 16.10
* Coding language : C++/embedded c
* Tools : Arduino IDE 18.01
* Library : OpenGL

**Summary**

This chapter gives details of the functional requirements, non-functional requirements, resource requirements, hardware requirements, software requirements etc. Again, the non-functional requirements in turn contain product requirements, organizational requirements, user requirements, basic operational requirements etc.

**Chapter 5**

**SYSTEM ANALYSIS**

**Overview**

Analysis is the process of finding the best solution to the problem. System analysis is the process by which we learn about the existing problems, define objects and requirements and evaluates the solutions. It is the way of thinking about the organization and the problem it involves, a set of technologies that helps in solving these problems. Feasibility study plays an important role in system analysis which gives the target for design and development.

**5.1 Feasibility Study**

All systems are feasible when provided with unlimited resource and infinite time. But unfortunately, this cond4ition does not prevail in practical world. So, it is both necessary and prudent to evaluate the feasibility of the system at the earliest possible time. Months or years of effort, thousands of rupees and untold professional embarrassment can be averted if an ill-conceived system is recognized early in the definition phase. Feasibility & risk analysis are related in many ways. If project risk is great, the feasibility of producing quality software is reduced. In this case there are three primary areas of interest:

**5.1.1 Performance Analysis**

For the complete functionality of the project work, the project is run with the help of healthy networking environment. Normally, the OS is windows 7. The main theme of this project is designing a system that converts signs into speech. Performance analysis is done to find out whether the proposed system is time efficient and accurate. It is essential that the process of performance analysis and definition must be conducted in parallel.

**5.1.2 Technical Analysis**

System is only beneficial only if it can be turned into information systems that will meet the organization’s technical requirement. Simply stated this test of feasibility asks whether the system will work or not when developed & installed, whether there are any major barriers to implementation. Regarding all these issues in technical analysis there are several points to focus on:-

**Changes to bring in the system:** All changes should be in positive direction, there will be increased level of efficiency and better customer service.

**Required skills:** Platforms & tools used in this project are widely used. So the skilled manpower is readily available in the industry.

**Acceptability:** The structure of the system is kept feasible enough so that there should not be any problem from the user’s point of view.

**5.1.3 Economical Analysis**

Economic analysis is performed to evaluate the development cost weighed against the ultimate income or benefits derived from the developed system. For running this system, we simply need a computer. All the features in this system run even on the other Operating Systems. So the system is economically feasible enough.

**Summary**

The main aim of this chapter is to find out whether the system is feasible enough or not. For these reasons’ different kinds of analysis, such as performance analysis, technical analysis, economic analysis etc. is performed.

**Chapter 6**

**SYSTEM DESIGN**

**Overview**

Design is a meaningful engineering representation of something that is to be built. It is the most crucial phase in the developments of a system. Software design is a process through which the requirements are translated into a representation of software. Design is a place where design is fostered in software Engineering. Based on the user requirements and the detailed analysis of the existing system, the new system must be designed. This is the phase of system designing. Design is the perfect way to accurately translate a customer’s requirement in the finished software product. Design creates a representation or model, provides details about software data structure, architecture, interfaces and components that are necessary to implement a system. The logical system design arrived at as a result of systems analysis is converted into physical system design.

**6.1 System development methodology**

System development method is a process through which a product will get completed or a product gets rid from any problem. Software development process is described as a number of phases, procedures and steps that gives the complete software. It follows series of steps which is used for product progress. The development method followed in this project is waterfall model.

**6.1.1 Model phases**

The waterfall model is a [sequential](http://en.wikipedia.org/wiki/Sequence) software development process, in which progress is seen as flowing steadily downwards (like a [waterfall](http://en.wikipedia.org/wiki/Waterfall)) through the phases of Requirement initiation, [Analysis](http://en.wikipedia.org/wiki/Analysis), [Design](http://en.wikipedia.org/wiki/Design), Implementation, [Testing](http://en.wikipedia.org/wiki/Software_testing) and [maintenance](http://en.wikipedia.org/wiki/Software_maintenance).

**Requirement Analysis:** This phase is concerned about collection of requirements of the system. This process involves generating document and requirement review.

**System Design:** Keeping the requirements in mind the system specifications are translated into a software representation. In this phase the designer emphasizes on: algorithm**,** data structure**,** software architecture etc.

**Coding:** In this phase programmer starts his coding in order to give a full sketch of product. In other words, system specifications are only converted into machine readable compute code.

**Implementation:** The implementation phase involves the actual coding or programming of the software. The output of this phase is typically the library, executables, user manuals and additional software documentation

**Testing:** In this phase all programs (models) are integrated and tested to ensure that the complete system meets the software requirements. The testing is concerned with verification and validation.

**Maintenance:** The maintenance phase is the longest phase in which the software is updated to fulfill the changing customer need, adapt to accommodate change in the external environment, correct errors and oversights previously undetected in the testing phase, enhance the efficiency of the software.

The project abides by prototyping model. The prototyping model is a systems development method in which a prototype is built, tested and then reworked as necessary until an acceptable prototype is finally achieved from which the complete system or product can now be developed.

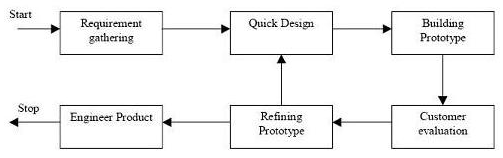
****

Figure 6.1.1 – Prototyping model

**6.2 Design Using UML**

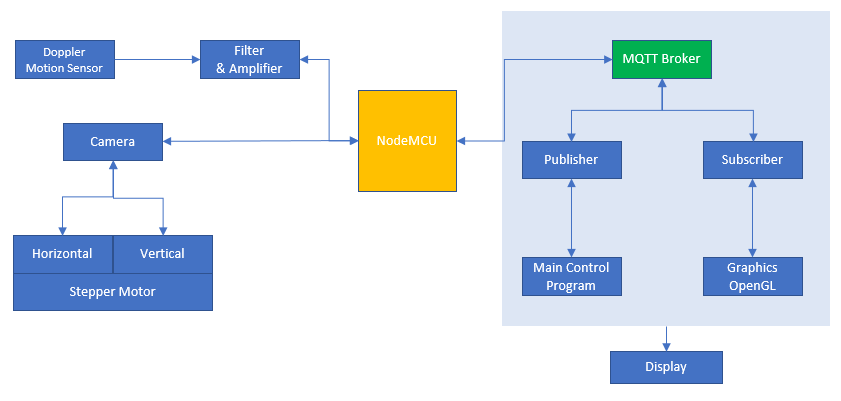
Designing UML diagram specifies, how the process within the system communicates along with how the objects with in the process collaborate using both static as well as dynamic UML diagrams since in this ever-changing world of Object Oriented application development, it has been getting harder and harder to develop and manage high quality applications in reasonable amount of time. As a result of this challenge and the need for a universal object modeling language everyone could use, the Unified Modeling Language (UML) is the Information industries version of blueprint. It is a method for describing the systems architecture in detail. Easier to build or maintains system, and to ensure that the system will hold up to the requirement changes.

**Table 6.2: Symbols used in UML**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl. No** | **Symbol** | **Symbol** | **Description** |
| 1 | Class |  | Classes represent a collection of similar entities grouped together. |
| 2 | Association |  | Association represents a static relation between classes. |
| 3 | Aggregation |  | Aggregation is a form of association. It aggregates several classes into a single class. |
| 4 | Composition |  | Composition is a special type of aggregation that denotes a strong ownership between classes. |
| 5 | Actor |  | Actor is the user of the system and other external entity hat react with the system. |
| 6 | Use Case |  | A use case is an interaction between system and the external environment. |
| 7 | Relation (Uses) |  | It is used for additional purpose communication. |
| 8 | Communication |  | It is the communication between use cases. |
| 9 | State |  | It represents the state of process. Each state goes through various flows. |
| 10 | Initial State |  | It represents initial state of object. |
| 11 | Final State |  | It represents final state of object. |
| 12 | Control Flow |  | It represents decision making process for object. |
| 13 | Decision Box |  | It represents the decision making process from a constraint. |
| 14 | Data Process/ State |  | A circle in a DFD represents a state or process which has been triggered due to some other event or action. |
| 15 | External Entity |  | It represents external entity such as Keyboard, sensors, etc which are used in the system. |
| 16 | Transition |  | It represents any communication that occurs between processes. |
| 17 | Object Lifeline |  | Object lifeline represents the vertical dimension that object communicates. |
| 18 | Message |  | It represents messages exchanged. |

**6.2.1 Architectural Design**

The overall logical structure of the project is divided into processing modules and a conceptual data structure is defined as Architectural Design.



**Figure 6.2.1:** System Architecture for Animal farm intrusion detection

Figure 6.2.1 shows the Architecture design where the motion sensor output is filtered and amplified and is sent to MQTT broker through NodeMCU. The broker further sends command to trigger the camera through a controller via publisher messages.

The camera further captures the visual and communicates with broker about the visual and receives the information of the number of degrees for it to turn to track the object of interest. Along with the tracking of the object the broker also processes the visual sent by the camera to identify the type of intrusion.

**6.2.2 Data Flow Diagram**

A data flow diagram (DFD) is graphic representation of the "flow" of data through an information system. A data flow diagram can also be used for the visualization of data processing (structured design). It is common practice for a designer to draw a context-level DFD first which shows the interaction between the system and outside entities. DFD’s show the flow of data from external entities into the system, how the data moves from one process to another, as well as its logical storage. There are only four symbols:

**1.** Squares representing *external entities*, which are sources and destinations of information entering and leaving the system.

**2.** Rounded rectangles representing *processes*, in other methodologies, may be called 'Activities', 'Actions', 'Procedures', 'Subsystems' etc. which take data as input, do processing to it, and output it.

**3.** Arrows representing the *data flows*, which can either, be electronic data or physical items. It is impossible for data to flow from data store to data store except via a process, and external entities are not allowed to access data stores directly.

**4.** The flat three-sided rectangle is representing data stores should both receive information for storing and provide it for further processing.

Figure 6.2.2.1 shows the data flow of the proposed system. The intrusion is detected by the PIR Doppler motion sensor and the message is sent to respective camera to focus itself onto the intrusion via MQTT protocol using MQTT broker. The cameras the track the intrusion and locate it in the farm using OpenGL.

**Figure 6.2.2.1:** Data Flow Diagram for Animal farm intrusion detection system

**6.2.3 Flow Chart**

Flow Chart is used to depict an algorithmic flow of data in a process. With the pre-processing steps such as gray scaling, blurring, thresholding and binarization, a binary image is obtained on which the sign interpretation takes place.

**Figure 6.2.3:** Flowchart for Animal farm intrusion detection

Figure 6.2.3.1 shows the data flow of the proposed system. sThe intrusion is detected by the PIR Doppler motion sensor and the message is sent to respective camera to focus itself onto the intrusion via MQTT protocol using MQTT broker. The cameras the track the intrusion and locate it in the farm using OpenGL

**6.2.4 Class Diagram**

UML class diagram shows the static structure of the model. The class diagram is a collection of static modeling elements, such as classes and their relationships’, connected as a graph to each other and to their contents.

The class diagram is the main building block of object-oriented modeling. It is used both for general conceptual modeling of the systematic of the application, and for detailed modeling translating the models into programming code. Class diagrams can also be used for data modeling. The classes in a class diagram represent both the main objects and or interactions in the application and the objects to be programmed.

**Fig 6.2.4:** Class diagram for Animal farm intrusion detection

**6.2.5 Use Case Diagram**

A use case defines a goal-oriented set of interactions between external entities and the system under consideration. The external entities which interact with the system are its actors. A set of use cases describe the complete functionality of the system at a particular level of detail and it can be graphically denoted by the use case diagram.

**Figure 6.2.5.1:** Use Case diagram for Animal farm intrusion detection

**Figure 6.2.5.2:** Use Case diagram with components

**6.2.6 Activity Diagram**

An activity diagram shows the sequence of steps that make up a complex process. An activity is shown as a round box containing the name of the operation. An outgoing solid arrow attached to the end of the activity symbol indicates a transition triggered by the completion.

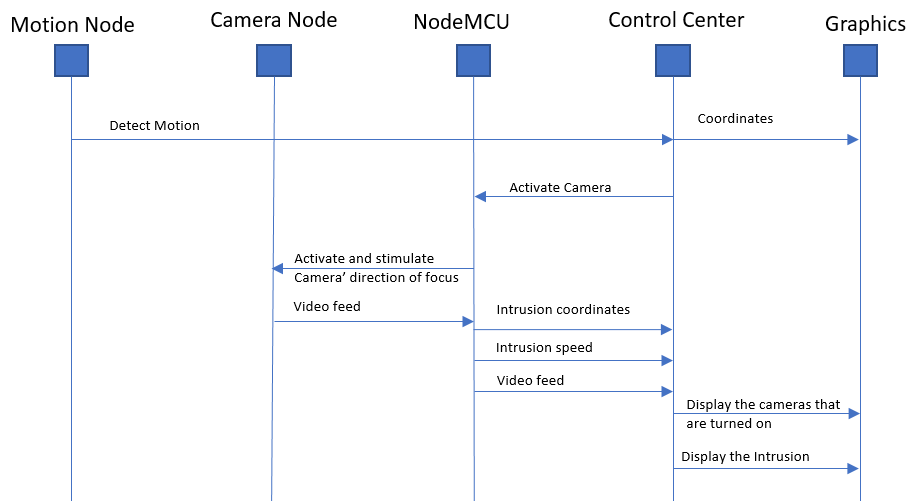
**Figure 6.2.6:** Activity Diagram for Animal farm intrusion detection

**6.2.7 Sequence Diagram**

Sequence diagram are an easy and intuitive way of describing the behavior of a system by viewing the interaction between the system and the environment. A sequence diagram shows an interaction arranged in a time sequence. A sequence diagram has two dimensions: vertical dimension represents time; the horizontal dimension represents the objects existence during the interaction.

**Basic elements:**

* Vertical rectangle: represent the object is active (method is being performed).
* Vertical dashed line: represent the life of the object.
* ”X”: represent the life end of an object. (Being destroyed from memory)
* Horizontal line with arrows: messages from one object to another.

****

**Figure 6.2.7:** Sequence Diagram for Animal farm intrusion detection

Figure 6.2.7 shows the sequence diagram for the proposed system.

**Summary**

This chapter mainly concentrates on few fundamental design concepts such as system development methodology, system architecture, class diagram, flowchart, sequence diagram, use-case diagram, activity diagram, data flow diagram etc.

**Chapter 7**

**IMPLEMENTATION**

The implementation phase of the project is where the detailed design is actually transformed into working code. Aim of the phase is to translate the design into a best possible solution in a suitable programming language. This chapter covers the implementation aspects of the project, giving details of the programming language and development environment used. It also gives an overview of the core modules of the project with their step by step flow.

The implementation stage requires the following tasks.

* Careful planning.
* Investigation of system and constraints.
* Design of methods to achieve the changeover.
* Evaluation of the changeover method.
* Correct decisions regarding selection of the platform
* Appropriate selection of the language for application development

The whole processing is done in the control center using the MQTT broker to pass messages via MQTT protocol.

**7.1 Detecting motion**

The first step is to detect the motion whereby we trigger the cameras that are required only to track the intrusion from that specific location in the farm. This is an advantage for the system as it saves up enormous power.

**7.2 Capturing an Image**

The next step to capturing an image involves obtaining a stream of images, each of which is further broken down and referred to as frames. A **webcam** is a video camera that feeds or streams its image in real time to or through a computer to computer network. When "captured" by the computer, the video stream may be saved, viewed or sent on to other networks via systems such as the internet, and email as an attachment.

A common fundamental step of video indexing, called video segmentation, is to divide the video stream into a set of segments (shots).A shot is described as an unbroken sequence of frames captured from one camera. In general, it consists of frames that have consistent visual characteristics, such as color, motion, object and texture. In order to segment a video sequence into shots, we need to define a dissimilarity or similarity methods among frames.

**7.3 Preprocessing**

In [imaging science](https://en.wikipedia.org/wiki/Imaging_science), image processing is processing of images using mathematical operations by using any form of [signal processing](https://en.wikipedia.org/wiki/Signal_processing) for which the input is an image, a series of images, or a video, such as a [photograph](https://en.wikipedia.org/wiki/Photograph) or [video frame](https://en.wikipedia.org/wiki/Video_frame); the output of image processing may be either an image or a set of characteristics or [parameters](https://en.wikipedia.org/wiki/Parameter) related to the image. Most image-processing techniques involve treating the image as a [two-dimensional](https://en.wikipedia.org/wiki/Two-dimensional) [signal](https://en.wikipedia.org/wiki/Signal_(electrical_engineering)) and applying standard signal-processing techniques to it. Images are also processed as [three-dimensional](https://en.wikipedia.org/wiki/Three-dimensional) [signals](https://en.wikipedia.org/wiki/Signal_(electrical_engineering)) where the third-dimension being time or the z-axis.

**7.3.1 Grayscale Conversion**

 A grayscale or greyscale [digital image](https://en.wikipedia.org/wiki/Digital_image) is an image in which the value of each [pixel](https://en.wikipedia.org/wiki/Pixel) is a single [sample](https://en.wikipedia.org/wiki/Sample_(signal)), that is, it carries only [intensity](https://en.wikipedia.org/wiki/Luminous_intensity) information. Images of this sort, also known as [black-and-white](https://en.wikipedia.org/wiki/Black-and-white), are composed exclusively of shades of [gray](https://en.wikipedia.org/wiki/Grey), varying from black at the weakest intensity to white at the strongest.

Grayscale images are distinct from one-bit bi-tonal black-and-white images, which in the context of computer imaging are images with only the two [colors](https://en.wikipedia.org/wiki/Color), [black](https://en.wikipedia.org/wiki/Black), and [white](https://en.wikipedia.org/wiki/White) (also called *bilevel* or [*binary images*](https://en.wikipedia.org/wiki/Binary_image)). Grayscale images have many shades of gray in between.

Grayscale images are often the result of measuring the intensity of light at each pixel in a single band of the [electromagnetic spectrum](https://en.wikipedia.org/wiki/Electromagnetic_spectrum) (e.g. Infrared, [visible light](https://en.wikipedia.org/wiki/Visible_spectrum), [ultraviolet](https://en.wikipedia.org/wiki/Ultraviolet), etc.), and in such cases they are monochromatic proper when only a given [frequency](https://en.wikipedia.org/wiki/Frequency) is captured.

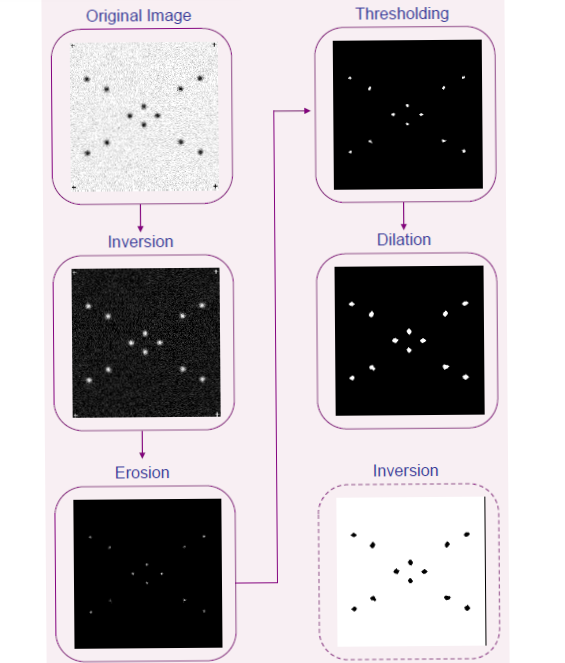


Figure 7.3.1.1: Image processing procedure followed in the project

The captured image is sent to the MQTT broker for further processing. The broker sends the input to the motors so as to turn the camera along with the intrusion that is detected and also it displays the output onto the OpenGL to locate the intrusion within the farm.

**Chapter 8**

**TESTING**

System testing is actually a series of different tests whose primary purpose is to fully exercise the computer-based system. Although each test has a different purpose, all work to verify that all the system elements have been properly integrated and perform allocated functions. The testing process is actually carried out to make sure that the product exactly does the same thing what is supposed to do. In the testing stage following goals are tried to achieve:

* To affirm the quality of the project.
* To find and eliminate any residual errors from previous stages.
* To validate the software as a solution to the original problem.
* To provide operational reliability of the system.

**8.1 Testing Methodologies**

There are many different types of testing methods or techniques used as part of the software testing methodology. Some of the important testing methodologies are:

* + 1. **White box testing**

White box testing (clear box testing, glass box testing, and transparent box testing or structural testing) uses an internal perspective of the system to design test cases based on internal structure. It requires programming skills to identify all paths through the software. The tester chooses test case inputs to exercise paths through the code and determines the appropriate outputs. While white box testing is applicable at the unit, integration and system levels of the software testing process, it is typically applied to the unit. While it normally tests paths within a unit, it can also test paths between units during integration, and between subsystems during a system level test.

Though this method of test design can uncover an overwhelming number of test cases, it might not detect unimplemented parts of the specification or missing requirements, but one can be sure that all paths through the test object are executed. Using white box testing we can derive test cases that:

* Guarantee that all independent paths within a module have been exercised at least once.
* Exercise all logical decisions on their true and false sides.
* Execute all loops at their boundaries and within their operational bounds.
* Execute internal data structure to assure their validity
  + 1. **Black box testing**

Black box testing focuses on the functional requirements of the software. It is also known as functional testing. It is a [software](http://www.webopedia.com/TERM/B/software.html) testing technique whereby the internal workings of the item being tested are not known by the tester. For example, in a black box test on software design the tester only knows the inputs and what the expected outcomes should be and not how the program arrives at those outputs.

The tester does not ever examine the programming [code](http://www.webopedia.com/TERM/B/code.html) and does not need any further knowledge of the program other than its specifications. It enables us to derive sets of input conditions that will fully exercise all functional requirements for a program. Black box testing is an alternative to white box technique. Rather it is a complementary approach that is likely to uncover a different class of errors in the following categories:-

* Incorrect or missing function.
* Interface errors.
* Performance errors.
* Initialization and termination errors.
* Errors in objects.

**Advantages**

* The test is unbiased as the designer and the tester are independent of each other.
* The tester does not need knowledge of any specific programming languages.
* The test is done from the point of view of the user, not the designer.
* Test cases can be designed as soon as the specifications are complete.

**8.2 Unit Testing**

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

**8.2.1 Functions for Input**

**Table 8.2.1: Functions for Neighbors Management**

|  |  |  |
| --- | --- | --- |
| **Function** | **Tests done** | **Remarks** |
| cv2.VideoCapture(device) | Tested to check if video has been captured from a camera | Success |
| cv2.VideoCapture.isOpened() | Tested to check if initializing video is done correctly | Success |
| cv2.VideoCapture.read() | Tested to check if a frame is grabbed and retrieved from the captured video. | Success |

**8.2.2 Functions for Output**

**Table 8.2.3: Functions for Output**

|  |  |  |
| --- | --- | --- |
| **Function** | **Tests done** | **Remarks** |
| cv.PutText(img, text, org, font, color) | Tested to check if specified text string is rendered in the image. | Success |
| cv2.imshow(name,image) | Tested to check if image is displayed in the specified window. | Success |

**8.3 Integration Testing**

  Upon completion of unit testing, integration testing begins. Individual modules are combined and tested as a group. Integration testing is black box testing. The purpose of integration testing is to ensure distinct components of the application still work in accordance to user requirements. Integration testing is considered complete, when actual results and expected results are either in line or differences are explainable based on client input. It concentrates on data transfer between modules. Integration testing is a logical extension of unit testing. Two units that have already been tested are combined into a component and the interface between them is tested. Integration testing identifies problems that occur when units are combined. The errors that arise can be attributed to those occurring due to the combination of modules, resulting from errors across interface.

The Integration Testing Table 8.3 shows the functions that are combined into different classes and the module tested for its functionality. Finally, all the modules are integrated and tested. This is important to check for error-free interaction between various classes and its modules. The integration testing table shows the important modules integrated.

**Table 8.3: Integration Testing**

|  |  |  |
| --- | --- | --- |
| **Modules** | **Functions integrated** | **Tests done** |
| Input Module | * Functions used in Video Capture * Functions used in Image Frame Capture | Tested the function of Input Module. |
| Image Processing and Feature Extraction Module | * Functions used in Grayscale Conversion * Functions used for Gaussian Filter * Functions used in Thesholding and Binarization * Functions used in Orientation Detection * Functions used in Contours and Convex Hull calculation * Functions used in Peak Detection * Functions used in Digit/Alphabet Mapping * Functions used in Gesture Interpretation | Tested the function of Image Processing and Feature Extraction Module. |
| Output Module | * Functions used in Text Representation. * Functions used in Displaying Images. | Tested the function of Output Module. |

**8.4 System Testing**

System testing of software or hardware is testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. System testing falls within the scope of black box testing, and as such, should require no knowledge of the inner design of the code or logic.

As a rule, system testing takes, as its input, all the "integrated" software components that have passed integration testing and also the software system itself integrated with any applicable hardware system(s). The purpose of integration testing is to detect any inconsistencies between the software units that are integrated together (called assemblages) or between any of the assemblages and the hardware. System testing is a more limited type of testing; it seeks to detect defects both within the "inter-assemblages" and also within the system as a whole.

System testing is performed on the entire system in the context of a Functional Requirement Specification(s) (FRS) and/or a System Requirement Specification (SRS). System testing tests not only the design, but also the behavior and even the believed expectations of the customer. It is also intended to test up to and beyond the bounds defined in the software/hardware requirements specification(s).

The following examples are different types of testing that should be considered during System testing:

* Graphical user interface testing
* Usability testing
* Software performance testing
* Compatibility testing
* Exception handling
* Load testing
* Volume testing

Although different testing organizations may prescribe different tests as part of System testing, this list serves as a general framework or foundation to begin with.

**8.5 Quality Assurance**

Quality assurance consists of the auditing and reporting functions of management. The goal of quality assurance is to provide management with the data necessary to be informed about product quality, thereby gaining insight and confident that the product quality is meeting its goals. This is an ―umbrella activity‖ that is applied throughout the engineering process. Software quality assurance encompasses:

* Analysis, design, coding and testing methods and tools
* Formal technical reviews that are applied during each software engineering
* Multitier testing strategy
* Control of software documentation and the change made to it
* A procedure to ensure compliance with software development standards.
* Measurement and reporting mechanisms.

Quality Assurance (QA) is a way of preventing mistakes or defects in manufactured products and avoiding problems when delivering solutions or services to customers. QA is applied to physical products in pre-production to verify what will be made meets specifications and requirements, and during manufacturing production runs by validating lot samples meet specified quality controls. QA is also applied to software to verify that features and functionality meet business objectives, and that code is relatively bug free prior to shipping or releasing new software products and versions. Quality Assurance refers to administrative and procedural activities implemented in a quality system so that requirements and goals for a product, service or activity will be fulfilled. It is the systematic measurement, comparison with a standard, monitoring of processes and an associated feedback loop that confers error prevention. This can be contrasted with quality control, which is focused on process output.

Two principles included in Quality Assurance are: "Fit for purpose", the product should be suitable for the intended purpose; and "Right first time", mistakes should be eliminated. QA includes management of the quality of raw materials, assemblies, products and components, services related to production, and management, production and inspection processes. Suitable quality is determined by product users, clients or customers, not by society in general. It is not related to cost, and adjectives or descriptors such as "high" and "poor" are not applicable. For example, a low priced product may be viewed as having high quality because it is disposable, where another may be viewed as having poor quality because it is not disposable.

Software quality assurance (SQA) consists of a means of monitoring the software engineering processes and methods used to ensure quality. The methods by which this is accomplished are many and varied, and may include ensuring conformance to one or more standards, such as ISO 9000 or a model such as CMMI.SQA encompasses the entire software development process, which includes processes such as requirements definition, software design, coding, source code control, code reviews, software configuration management, testing, release management, and product integration.

**8.5.1 Quality Factors**

An important objective of quality assurance is to track the software quality and assess the impact of methodological and procedural changes on improved software quality. The factors that affect the quality can be categorized into two broad groups:

* Factors that can be directly measured.
* Factors that can be indirectly measured

These factors focus on three important aspects of a software product

* Its operational characteristics
* Its ability to undergo changes
* Its adaptability to a new environment.
* Effectiveness or efficiency in performing its mission
* Duration of its use by its customer.

In the context of software engineering, software quality refers to two related but distinct notions that exist wherever quality is defined in a business context: Software functional quality reflects how well it complies with or conforms to a given design, based on functional requirements or specifications. That attribute can also be described as the fitness for purpose of a piece of software or how it compares to competitors in the marketplace as a worthwhile product; Software structural quality refers to how it meets non-functional requirements that support the delivery of the functional requirements, such as robustness or maintainability, the degree to which the software was produced correctly.

Structural quality is evaluated through the analysis of the software inner structure, its source code, at the unit level, the technology level and the system level, which is in effect how its architecture adheres to sound principles of software architecture outlined in a paper on the topic by OMG. In contrast, functional quality is typically enforced and measured through software testing. Historically, the structure, classification and terminology of attributes and metrics applicable to software quality management have been derived or extracted from the ISO 9126-3 and the subsequent ISO 25000:2005 quality model, also known as Square. Based on these models, the Consortium for IT Software Quality (CISQ) has defined five major desirable structural characteristics needed for a piece of software to provide business value: Reliability, Efficiency, Security, Maintainability and (adequate) Size.

Software quality measurement quantifies to what extent a software or system rates along each of these five dimensions. An aggregated measure of software quality can be computed through a qualitative or a quantitative scoring scheme or a mix of both and then a weighting system reflecting the priorities. This view of software quality being positioned on a linear continuum is supplemented by the analysis of "critical programming errors" that under specific circumstances can lead to catastrophic outages or performance degradations that make a given system unsuitable for use regardless of rating based on aggregated measurements.

Such programming errors found at the system level represent up to 90% of production issues, whilst at the unit-level, even if far more numerous, programming errors account for less than 10% of production issues. As a consequence, code quality without the context of the whole system, as W. Edwards Deming described it, has limited value.

To view, explore, analyze, and communicate software quality measurements, concepts and techniques of information visualization provide visual, interactive means useful, in particular, if several software quality measures have to be related to each other or to components of a software or system.

**8.6 Intrusion detection during Testing**

****

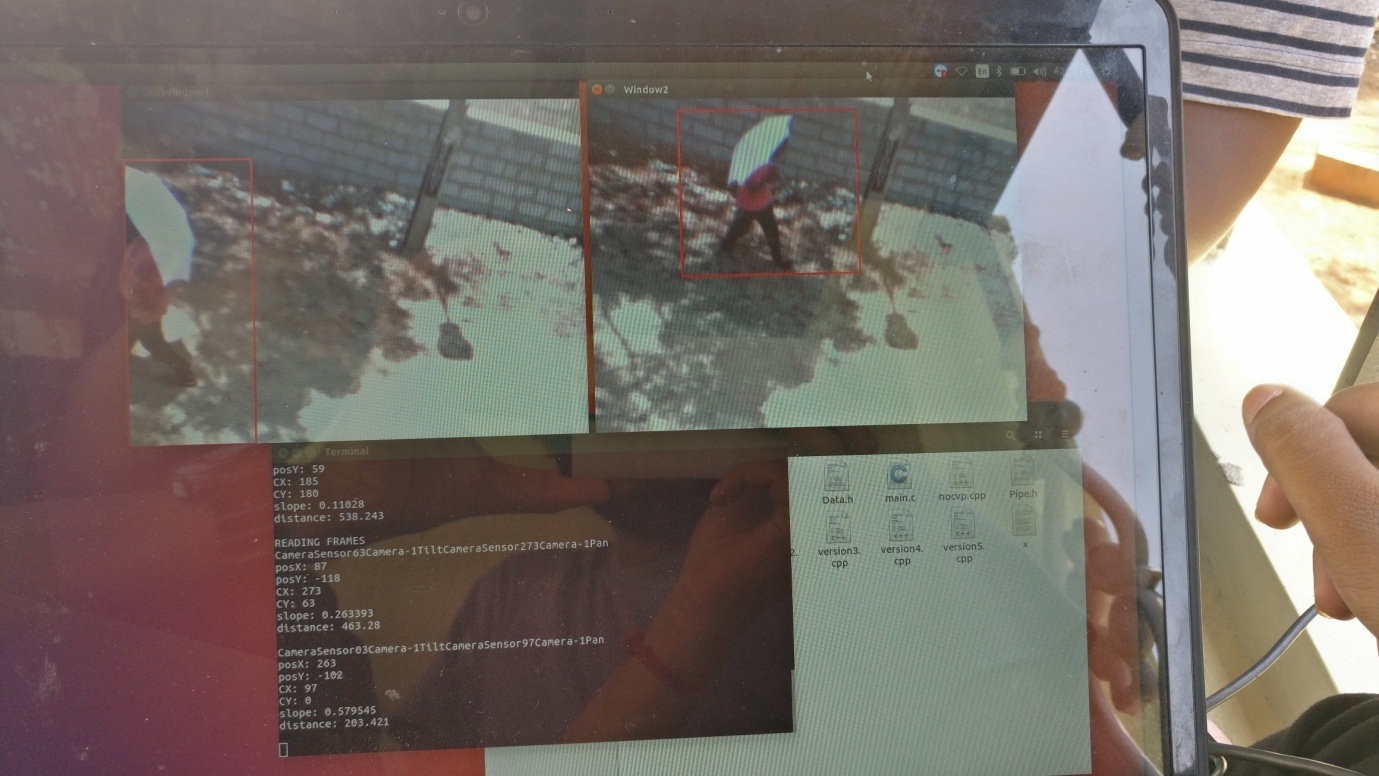
****

Figure: Sample motion detection

**Summary**

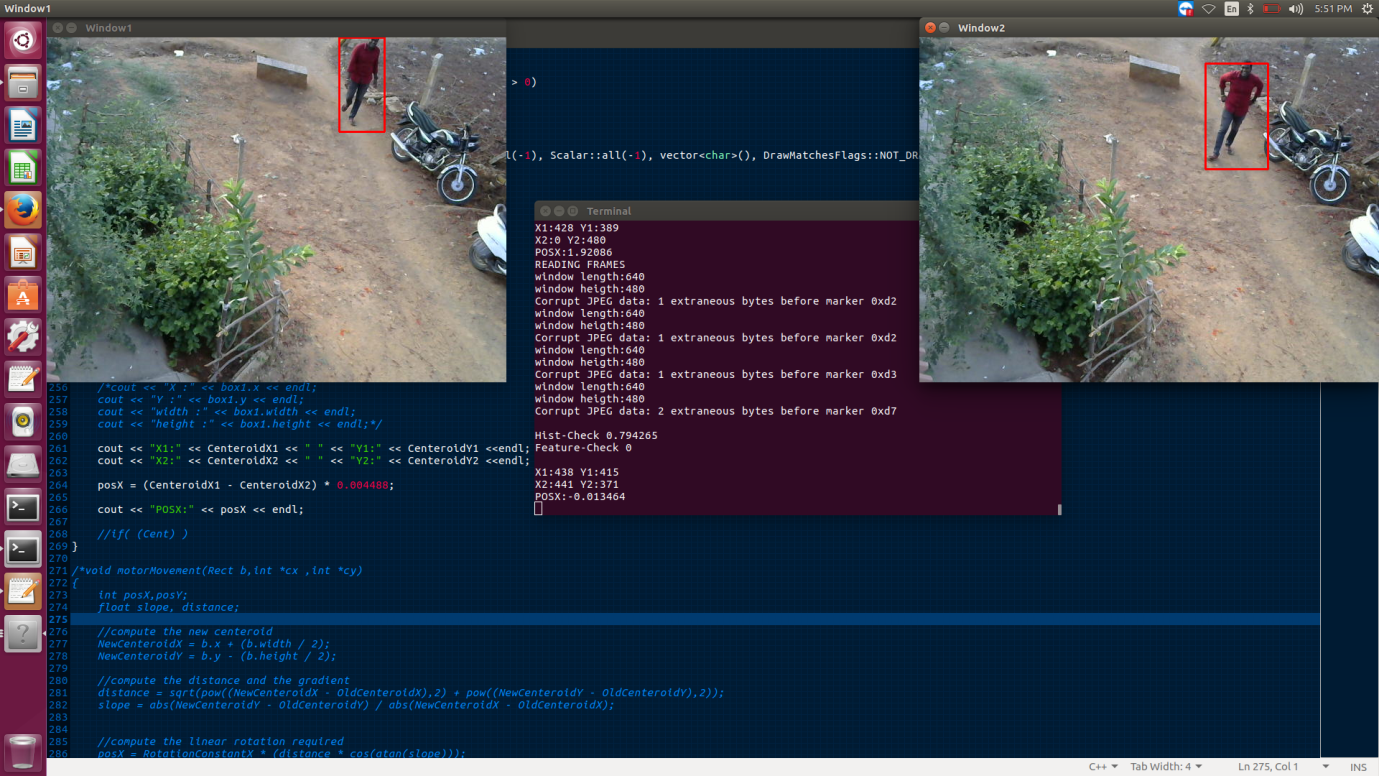
The chapter discusses the tests that are done on the system to check its functionality. Testing is carried out at three different levels from the module level to the system level checking for errors at each stage. The remarks have also been documented

**Chapter 9**

**RESULTS AND EXECUTION**

The following snapshots define the results or outputs that we will get after step by step execution of each proposed protocol for different values of time and speed.

**9.1 Snapshots**



**Figure : motion tracking**

**Summary**

The animal intrusion detection system correctly tracks and maps the intrusion onto the farm which is integrated and displayed onto OpenGL output window on the display.

Sign language recognition uses the properties of binarization, convex hull, and list processing to convert sign to its image representation.

**Chapter 10**

**CONCLUSION & FUTURE ENHANCEMENT**

**10.1 Conclusion**

The C++ based OpenCV application provides an error free detection and tracking of the intrusion. It follows a cost-effective approach in identifying intrusion. It follows image processing approaches that effectively uses the standard OpenCV library methods.

The disadvantage in this project is that the application works well on single intrusion detection.

**10.2 Future Enhancement**

In future, the platform can be trained to recognize images using machine learning based approaches in OpenCV (e.g.s Frame Differencing) that is independent of the intrusion. It can be used to monitor the crop growth depending on the images of the crops during various stages of the crop growth. This platform can be used for security cameras as well.