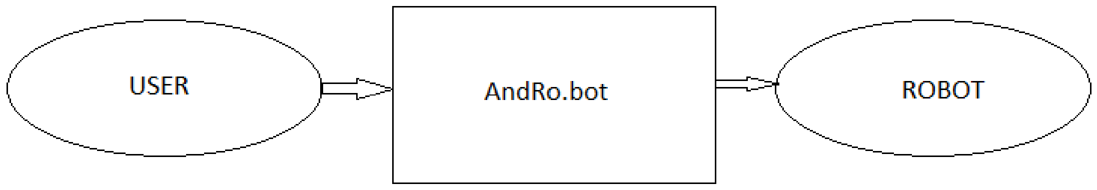
CHAPTER 1

INTRODUCTION

1.1 General Overview

It is usually seen in hobbyists' robotics that a highly complex hardware system is created to provide a few basic functionalities. This has resulted in robots that are expensive in price, but limited in functionalities. In our project, we try to build a flexible, robust, secure and high performance software platform that would shift the complexity from hardware to software in hobbyist robotics projects and enable the implementation of a wide range of features.



[Fig. 1: High-level System Architecture]

The AndRo.bot software runs on the Android software stack. It communicates with the user using HTTP or SMS and with the robot hardware Bluetooth. It makes no use of any sensor, apart from the sensors that comes built-in with the Android smart-phone. Some of the internal features that make this platform preferable over stand-alone solutions are:

1. MD5 and SHA1 based password authentication
2. Built-in HTTP server
3. RIA based Human-Robot-Interaction (HRI) using HTML5 / CSS3 / JQuery
4. SMS based HRI
5. Uses latest version of openCV for Computer Vision
6. Client-side Real-time Image Processing
7. Customizable using Android Preference

The software is made as general as possible and it enables the user to implement any type of algorithm that he may require for performing his tasks. The ability of the software platform is demonstrated by implementing these core functionalities:

- Live Image Streaming
- Face Detection
- Line Following
- Object Tracking
- Automatic Guided Vehicle (AGV)
- Voice Synthesis
- Voice Recognition
- Motion/Proximity/Magnetic Detection
- Google Maps based mapping

Live Image Streaming, which refers to images delivered live over the Internet, requires a camera for the media, an encoder to digitize the content, a media publisher, and a content delivery network to distribute and deliver the content. A client media player can begin playing the data before the entire file has been transmitted. Distinguishing delivery method from the media distributed applies specifically to telecommunications networks, as most other delivery systems are either inherently streaming (e.g., radio, television) or inherently non-streaming (e.g., books, video cassettes, audio CDs). For example, in the 1930s, muzak was among the earliest popularly available streaming media; nowadays Internet television is a common form of streamed media. The term "streaming media" can apply to media other than video and audio such as live closed captioning, stock ticker, and real-time text, which are all considered "streaming text". The term "streaming" was first used in the early 1990s as a better description for video on demand on IP networks; at the time such video was usually referred to as "store and forward video", which was misleading nomenclature.

Face Detection (FD) is a computer technology that determines the locations and sizes of human faces in arbitrary (digital) images. It detects facial features and ignores anything else, such as buildings, trees and bodies. Face detection can be regarded as a specific case of object-class detection. In object-class detection, the task is to find the locations and sizes of all objects in an image that belong to a given class. Examples include upper torsos, pedestrians, and cars. Early face-detection algorithms focused on the detection of frontal human faces, whereas newer algorithms attempt to solve the more general and difficult problem of multi-view face detection. That is, the detection of faces that are either rotated along the axis from the face to the observer (in-plane rotation), or rotated along the vertical or left-right axis (out-of-plane rotation), or both. The newer algorithms take into account variations in the image or video by factors such as face appearance, lighting, and pose.

Line-following robots are some of the earliest Automatic Guided Vehicle (AGVs). They might follow a visual line painted or embedded in the floor or ceiling or an electrical wire in the floor. Most of these robots operated a simple ‘keep the line in the center sensor’ algorithm. They could not circumnavigate obstacles; they just stopped and waited when something blocked their path. Many examples of such vehicles are still sold, by Transbotics, FMC, Egemin, HK Systems and many other companies. Several competitions are held worldwide for line-following robots. IIT Robotix is an example of such event in India. Some rules for such a competition can be accessed using the following QR.

[Fig 2 : QR link to line-follower rules]

Object Tracking is the process of locating a moving object (or multiple objects) over time using a camera. It has a variety of uses, some of which are: human-computer interaction, security and surveillance, video communication and compression, augmented reality, traffic control, medical imaging and video editing. Video tracking can be a time consuming process due to the amount of data that is contained in video. Adding further to the complexity is the possible need to use object recognition techniques for tracking.

Tele-operation indicates operation of a machine at a distance. It is similar in meaning to the phrase "remote control" but is usually encountered in research, academic and technical environments. It is most commonly associated with robotics and mobile robots but can be applied to a whole range of circumstances in which a device or machine is operated by a person from a distance. *Automatic Guided Vehicle* is one that can be tele-operated using remote devices.

Speech or *voice synthesis* is the artificial production of human speech. A computer system used for this purpose is called a speech synthesizer, and can be implemented in software or hardware products. A text-to-speech (TTS) system converts normal language text into speech; other systems render symbolic linguistic representations like phonetic transcriptions into speech. The quality of a speech synthesizer is judged by its similarity to the human voice and by its ability to be understood. An intelligible text-to-speech program allows people with visual impairments or reading disabilities to listen to written works on a home computer.

In computer science, speech or *voice recognition* is the translation of spoken words into text. It is also known as "automatic speech recognition", "ASR", "computer speech recognition", "speech to text", or just "STT". Some SR systems use "training" where an individual speaker reads sections of text into the SR system. These systems analyze the person's specific voice and use it to fine tune the recognition of that person's speech, resulting in more accurate transcription. Systems that do not use training are called "Speaker Independent" systems. Systems that use training are called "Speaker Dependent" systems. Speech recognition applications include voice user interfaces such as voice dialing (e.g. "Call home"), call routing (e.g. "I would like to make a collect call"), domestic appliance control, search (e.g. find a podcast where particular words were spoken), simple data entry (e.g., entering a credit card number), preparation of structured documents (e.g. a radiology report), speech-to-text processing (e.g., word processors or emails), and aircraft (usually termed Direct Voice Input).

Motion detection is the process of detecting a change in position of an object relative to its surroundings or the change in the surroundings relative to an object. Magnetic detection refers to the detection of change in angular orientation or the introduction of any electromagnetic field. Proximity detection detects when an object comes in the vicinity of the device. Motion (or magnetic or proximity) detection can be achieved by both mechanical and electronic methods.

They can be detected by: Infrared, Optics, Radio Frequency Energy, Sound, Vibration and Magnetism.

Mapping usually refers to map-making and often used instead of cartography. Mapping term is also sometimes used for geospatial data collection (e.g. LIDAR mapping) but in fact it is not mapping because a map is created through some cartographic works (i.e. determining the scale/level of detail and content of geographic or cartographic database, entry criteria and symbol specification for geospatial objects, generalization, layout design etc.). In other words, the acquisition of data with (geographic) coordinates directly from terrain or imagery does not mean mapping but surveying.

1.2 Motivation

Traditionally, the development of robots has required a wide range of complexity in hardware. The reception of a data from external environment required the usage of sensors to detect factors like electromagnetic spectrum, sound, touch, chemical sensors, temperature, range to things in the environment and attitude.

In recent years, the field of computer science has seen rapid developments in the fields of Computer Vision and in the construction of handheld computers (e.g., smart-phones, tablets, etcetera.). This can largely be attributed to Moore's Law, which is the observation that over the history of computing hardware, the number of transistors on integrated circuits doubles approximately every two years. This doubling of transistors has roughly made possible the doubling of the computing power of devices and softwares running on the Von Neumann architecture. Examples of this phenomenon are the aforementioned developments in the fields of computer vision and handheld computers.

Computer Vision is a field that includes methods for acquiring, processing, analyzing, and understanding images and, in general, high-dimensional data from the real world in order to produce numerical or symbolic information, *e.g.*, in the forms of decisions. A theme in the development of this field has been to duplicate the abilities of human vision by electronically perceiving and understanding an image. This image understanding can be seen as the disentangling of symbolic information from image data using models constructed with the aid of geometry, physics, statistics, and learning theory. Computer vision has also been described as the enterprise of automating and integrating a wide range of processes and representations for vision perception. Because construction of computer vision algorithms is a system-level task, most application developers use libraries for such tasks. OpenCV is an example of such a library.

OpenCV (*Open Source Computer Vision Library*) is a library of programming functions mainly aimed at real-time computer vision, developed by Intel, and now supported by Willow Garage and Itseez. It is free for use under the open source BSD license. The library is cross-platform. It focuses mainly on real-time image processing. The library is written in C++ and its primary interface is in C++, but it still retains a less comprehensive though extensive older C interface. There are now full interfaces in Python, Java and MATLAB/OCTAVE (as of version 2.5). The API for these interfaces can be found in the online documentation. Wrappers in other languages such as C#, Ch, Ruby have been developed to encourage adoption by a wider audience.

Similar developments have also been seen in the construction of handheld computers. A Handheld PC is a computer built around a form factor which is smaller than any standard laptop computer. It is sometimes referred to as a Palmtop. The first handheld device compatible with desktop IBM personal computers of the time was the Atari Portfolio of 1989. Another early model was the Poqet PC of 1989 and HP 95LX of 1991. Other DOS compatible hand-held computers also existed. Some Handheld PCs use Microsoft's Windows CE operating system. Devices such as smart-phones and tablets running on operating systems like iOS or Android are newer examples of handheld computers. Because of the cost ineffectiveness and the proprietary nature of the devices running iOS, Android has emerged as the better option among the lot. The testament to this provided by the fact that Android did have a worldwide smart-phone market share of 75% during the third quarter of 2012, with 750 million devices activated in total and 1.5 million activations per day.

Android is a Linux-based operating system designed primarily for touch-screen mobile devices such as smart-phones and tablet computers. Initially developed by Android, Inc., which Google backed financially and later bought in 2005, Android was unveiled in 2007 along with the founding of the Open Handset Alliance: a consortium of hardware, software, and telecommunication companies devoted to advancing open standards for mobile devices. The first Android-powered phone was sold in October 2008.

Android provide a rich API and a user-friendly development environment for application software development. Android 'apps' run on the Dalvik Virtual Machine, which is basically a FLOSS implementation of the Java Virtual machine. This makes possible the usage of a large number of modules for Java. Also, usually Android devices come with several sensors embedded into them.

Robotics has traditionally depended, as aforementioned, on sensors and hardware complexity. In the last few years, robots have shifted complexity from hardware to software (e.g., Honda Asimo, Curiosity Mars Rover, etc.). Because of the said developments of in the field of computer vision and handheld computing, we were motivated to experiment with the possibility of using Android only for the construction of a robotic software.

1.3 Objective

The objective of this project is to create a generalized software platform that can be used to solve a wide variety of robotics problems.

We plan to do this by creating the software in such a manner that it reside between the user and the actual robot hardware. Emphasis would be provided on making the HRI as user-friendly as possible. Also, we provided special attention to a few quality attributes, namely security, performance, usability and efficiency.

To sum up, we can enumerate the objectives of our project as:

- To shift complexity from hardware to software in hobbyist robotics,
- To implement recent ideas and innovations for Human Robot Interaction (HRI),
- To create a generalized software using only the Android APIs, sensors and camera for solving several prominent problems in hobbyist robotics.

1.4 Literary Survey

Green, S.A., Billinghurst, M., Chen, X., Chase, G.J. (2008, Human-Robot Collaboration: A Literature Review and Augmented Reality Approach in Design. International Journal of Advanced Robotic Systems, 5(1), pp. 1-18.) have pointed out that NASA's vision for space exploration stresses the cultivation of human-robotic systems. Similar systems are also envisaged for a variety of hazardous earthbound applications such as urban search and rescue. Recent research has pointed out that to reduce human workload, costs, fatigue driven error and risk, intelligent robotic systems will need to be a significant part of mission design. However, little attention has been paid to joint human-robot teams. Making human-robot collaboration natural and efficient is crucial. In particular, grounding, situational awareness, a common frame of reference and spatial referencing are vital in effective communication and collaboration. Augmented Reality (AR), the overlaying of computer graphics onto the real worldview, can provide the necessary means for a human-robotic system to fulfill these requirements for effective collaboration. They reviewed the field of human-robot interaction and augmented reality, investigates the potential avenues for creating natural human-robot collaboration through spatial dialogue utilizing AR and proposes a holistic architectural design for human-robot collaboration.

Companies such as Honda (Honda 2007), Toyota (Toyota 2007) and Sony (Sony 2007) are also interested in developing consumer robots that interact with humans in the home and workplace. There is growing interest in the field of human-robot interaction (HRI) as can be determined by the inaugural conference for HRI (HRI2006 2006). The Cogniron project (COGNIRON 2007), MIT Media lab (Hoffmann and Breazeal 2004) and the Mitsubishi Electric Research Laboratories (Sidner and Lee 2005) recognize the need for human-robot collaboration as well, and are currently conducting research in this emerging area.

Human-robot teams are used in Urban Search and Rescue (USAR). Robots are teleoperated and used mainly as tools to search for survivors. Studies completed on human-robot interaction for USAR reveal that the lack of situational awareness has a negative effect on performance (Murphy 2004), (Yanco, Drury et al. 2004). The use of an overhead camera and automatic mapping techniques improve situational awareness and reduce the number of navigational errors (Scholtz 2002; Scholtz, Antonishek et al. 2005). USAR is conducted in uncontrolled, hazardous environments with adverse ambient conditions that affect the quality of sensor and video data. Studies show that varying the level of robot autonomy and combining data from multiple sensors, thus using the best sensors for the given situation, increases the success rate of identifying survivors (Nourbakhsh, Sycara et al. 2005).

'Machine Vision Giving Eyes to Robots: Resources in Technology'. (March 1990, *Technology Teacher*, 49: 6, 21-28.) introduces machine vision, which can be used for inspection, robot guidance and part sorting. The future for machine vision will include new technology and will bring vision systems closer to the ultimate vision processor, the human eye.

Braggins, Don. (1995, 'A critical look at robot vision'. *The Industrial Robot*, 22(6): 9-12.) shows how one sort of robot vision is the use of robot-mounted cameras for inspection by vision techniques - in effect the vision is not providing data for the robot controller but passing the eye over something to be inspected in much the same way that a human inspector might examine an assembly from a number of different angles. The barriers to robot vision include: 1. It has been difficult to communicate data about coordinates to the robot. 2. Computer vision deals with the information present in a 2-dimensional representation of the 3-dimensional world. The future of vision systems includes robust stereo systems that rely on trigonometry to calculate where things are.

Grimson, W.E.L. and J.L. Mundy. (March 1994, 'Computer Vision applications'. *Communications of the ACM*, 37(3): 44-51.) describes how Computer vision provides a primary method for understanding how to make intelligent decisions about an environment, on the basis of sensory inputs. Vision systems only receive measurements of reflected brightness as input. Image brightness is not generally independent, and the goal of computer vision is to determine sufficient additional constraints to invert brightness into scene parameters. One class of methods achieves inversion by fixing some scene parameters. A 2nd class of methods achieves inversion by restricting the problem domain. A 3rd class achieves inversion by acquiring additional images. In general, vision methods seek to extract scene parameters, such as surface material type and object shape from image brightnesses, and to use such extracted parameters to match against known-object models to support tasks such as recognition. While general-purpose vision systems remain an area of active research, considerable progress in limited domains such as those listed has led to a number of areas of successful application.

The UWA telerobot project was first started by Ken Taylor and B. Dalton(1997, 'Issues in Internet Telerobotics'. *In International Conference on Field and Service Robotics (FSR 97)*). The framework has been designed to minimize the work required to connect a robot or other device to the Internet, while providing the basic functionality needed for a collaborative distributed system.

Most of the early World Wide Web controlled Robots as noted by K. Goldberg, K. Mascha, M. Genter, N. Rothenberg, C. Sutter, and J. Wiegley. (May 1995, ‘Desktop teleoperation via the World Wide Web’. In Proceedings of IEEE International Conference on Robotics and Automation) used the Common Gateway Interface(CGI) to interface between web browsers and the physical device being controlled.

The introduction of Java is probably the most significant as it allows code to be executed on the client side, whereas previously all code had to be in the CGI process. It is being used by a number of web telerobotics projects including the PUMA paint robot and the NASA Pathfinder interface as pointed out by Paul G. Backes, Kam S. Tao, and Gregory K. Tharp. (May 1998, ‘Mars pathfinder mission Internet-based operations using WITS’. In Proceedings of IEEE International Conference on Robotics and Automation, pages 284-291)

CHAPTER 2

SYSTEM REQUIREMENT SPECIFICATION

2.1 System Feasibility Study

A feasibility study is an evaluation and analysis of the potential of the proposed project which is based on extensive investigation and research to give full comfort to the decisions makers. Feasibility studies aim to objectively and rationally uncover the strengths and weaknesses of an existing business or proposed venture, opportunities and threats as presented by the environment, the resources required to carry through, and ultimately the prospects for success. In its simplest terms, the two criteria to judge feasibility are cost required and value to be attained. As such, a well-designed feasibility study should provide a historical background of the business or project, description of the product or service, accounting statements, details of the operations and management, marketing research and policies, financial data, legal requirements and tax obligations. Generally, feasibility studies precede technical development and project implementation.

There are number of steps in the feasibility study, some of them are

- Forming a team for the specific project and appointing a suitable leader.
- Preparing layouts and flowcharts of the system.
- Enumerate the candidate systems.
- Identify and describe the characters of the candidate systems.
- Determining the performance of each candidate system with the standards set.
- Reviewing the performance of the system and performing the cost analysis.
- Selecting the best candidate for the system and preparing the final report for the management.

A feasibility study evaluates the project's potential for success; therefore, the perceived objectivity is an important factor in the credibility to be placed on the study by potential investors and lending institutions. It must therefore be conducted with an objective, unbiased approach to provide information upon which decisions can be based.

The acronym TELOS refers to the five areas of feasibility - Technical, Economic, Legal, Operational, and Scheduling.

Technology and system feasibility is based on an outline design of system requirements, to determine whether the company has the technical expertise to handle completion of the project. When writing a feasibility report, the following should be taken to consideration:

- A brief description of the business to assess more possible factor/s which could affect the study
- The part of the business being examined
- The human and economic factor
- The possible solutions to the problems

At this level, the concern is whether the proposal is both *technically* and *legally* feasible (assuming moderate cost).

Legal Feasibility determines whether the proposed system conflicts with legal requirements, e.g. a data processing system must comply with the local Data Protection Acts.

Operational feasibility is a measure of how well a proposed system solves the problems, and takes advantage of the opportunities identified during scope definition and how it satisfies the requirements identified in the requirements analysis phase of system development

The operational feasibility assessment focuses on the degree to which the proposed development projects fits in with the existing business environment and objectives with regard to development schedule, delivery date, corporate culture, and existing business processes.

Economic Feasibility is undertaken with the intention to determine the positive economic benefits to the organization that the proposed system will provide. It includes quantification and identification of all the benefits expected. This assessment typically involves a cost/ benefits analysis.

The technical feasibility assessment is focused on gaining an understanding of the present technical resources of the organization and their applicability to the expected needs of the proposed system. It is an evaluation of the hardware and software and how it meets the need of the proposed system.

Schedule feasibility is a measure of how reasonable the project timetable is. Given our technical expertise, are the project deadlines reasonable? Some projects are initiated with specific deadlines. You need to determine whether the deadlines are mandatory or desirable.

Resource feasibility involves questions such as how much time is available to build the new system, when it can be built, whether it interferes with normal business operations, type and amount of resources required, dependencies,

Cultural feasibility involves the evaluation of the project for its impact on the local and general culture. For example, environmental factors need to be considered and these factors are to be well known. Further an enterprise's own culture can clash with the results of the project.

In case of a new project, financial viability can be judged on the following parameters:

- Total estimated cost of the project
- Financing of the project in terms of its capital structure, debt equity ratio and promoter's share of total cost
- Existing investment by the promoter in any other business
- Projected cash flow and profitability

The financial viability of a project should provide the following information:

- Full details of the assets to be financed and how liquid those assets are.
- Rate of conversion to cash-liquidity
- Project's funding potential and repayment terms.
- Sensitivity in the repayments capability to the following factors:
- Time delays.
- Mild slowing of sales.
- Acute reduction/slowing of sales.
- Small increase in cost.
- Large increase in cost.
- Adverse economic conditions.

2.2 System Analysis

Systems analysis is the study of sets of interacting entities, including computer systems analysis. This field is closely related to requirements analysis or operations research. It is also "an explicit formal inquiry carried out to help someone (referred to as the decision maker) identify a better course of action and make a better decision than he might otherwise have made."

The terms analysis and synthesis come from Greek where they mean respectively "to take apart" and "to put together". These terms are used in scientific disciplines from mathematics and logic to economics and psychology to denote similar investigative procedures. Analysis is defined as the procedure by which we break down an intellectual or substantial whole into parts. Synthesis is defined as the procedure by which we combine separate elements or components in order to form a coherent whole. Systems analysis researchers apply methodology to the analysis of systems involved to form an overall picture. System analysis is used in every field where there is a work of developing something. Analysis can also be defined as a series of components that perform organic function together.

The development of a computer-based information system includes a systems analysis phase which produces or enhances the data model which itself is a precursor to creating or enhancing a database. There are a number of different approaches to system analysis. When a computer-based information system is developed, systems analysis (according to the Waterfall model) would constitute the following steps:

- The development of a feasibility study, involving determining whether a project is economically, socially, technologically and organizationally feasible.
- Conducting fact-finding measures, designed to ascertain the requirements of the system's end-users. These typically span interviews, questionnaires, or visual observations of work on the existing system.
- Gauging how the end-users would operate the system (in terms of general experience in using computer hardware or software), what the system would be used for and so on.

Another view outlines a phased approach to the process. This approach breaks systems analysis into 5 phases:

- Scope Definition
- Problem analysis
- Requirements analysis
- Logical design
- Decision analysis

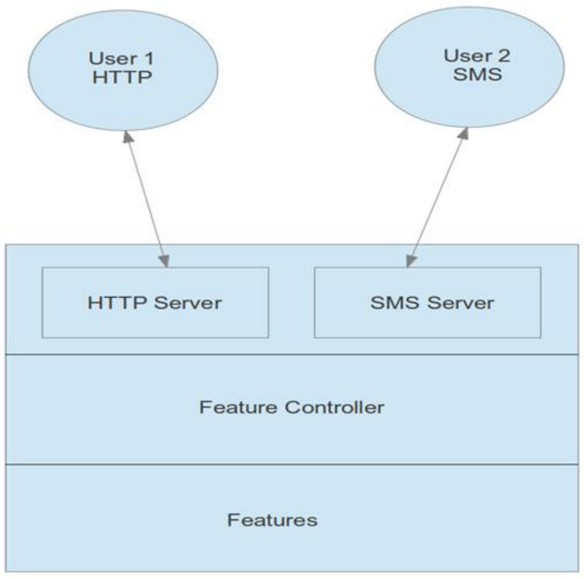
Use cases are a widely-used systems analysis modeling tool for identifying and expressing the functional requirements of a system. Each use case is a business scenario or event for which the system must provide a defined response. Use cases evolved out of object-oriented analysis; however, their use as a modeling tool has become common in many other methodologies for system analysis and design.

Existing hobbyist robots mainly use infrared sensors for line following and ultrasound sensors for obstacle avoidance. Tasks that require image processing and computer vision, like face detection and motion detection, are usually carried out in a PC (most commonly using MATLAB.) IP Cameras are used to stream data to the PC if the camera is to be placed remotely. Panoramic and time-lapse photography are usually done manually by placing the camera on some stand (viz. tripod) for stability. Voice synthesis is done using phonetics addition or specialized hardware. Voice recognition may be done by pattern matching, but they require lots of data to match against for any level of decent accuracy.

Using hardware sensors or IP cameras have the drawback of being costly. Also, using IP camera remotely requires a fast network connection to stream images to a PC in real-time. Manual panoramic or time-lapse photography requires the photographer to be present in the location. Voice recognition is hard even on a very powerful PC because of the large pattern-database requirement.

We plan to use the on-board Android’s camera to replace the sensors and IP camera as were being used in previous systems. We also plan to use the Android to carry out the image processing and computer vision on-board in real-time. This removes the need for a PC and any fast network connection. The voice recognition will be done using third-party applications over the cloud(i.e. Google cloud-based services). Some real-time image processing, which are not absolutely required by the robot, are done in the user-side using the RIA. The cloud-based RIA makes the UI truly portable and platform independent. We will use a simple handshake using MD5, SHA1 and AES to provide minimal security to the robot. In short, we are trying to create a flexible and secure robotic platform using modern and powerful tools and techniques. These tools will let us create a robot that can exhibit features which are usually difficult to implement in hobbyist robots.

The tools we will use are fully open source. Even the software codes we'll create are open sourced. Different features will need different techniques for implementation. Our robot is built using a five layered portable architecture that facilitates implementation of various features easily and without affecting the other features. Some features, like voice control and voice synthesis, use well known solutions, while others, like computer vision based line-following and obstacle-avoidance, will require a lot of hit-and-trials. Because we are using several systems internally *('standing on the shoulders of giants')*, we'll need to use a variety of languages (HTML5, CSS3, JavaScript, Java, C) and libraries (JQuery, openCV, Crypto, Pixastic, etc.) to create our software. The administrator will need a device with Internet connectivity and a HTML5-compitable browser.



[Fig 3 : High Level System Architecture]

2.3 System Requirements

- Hardware
    - Android smart-phone/tablet
    - User PC capable of running a modern HTML5 compatible browser
    - Robot hardware – 89c51, HC05
- Languages
    - HTML5, CSS3 and JavaScript
    - Java
    - C
    - Python (Decaprated)
- Extra Libraries / Modules
    - JQuery
    - Pixastic
    - Crypto
    - openCV
    - NanoHTTPD

CHAPTER 3

DESIGN

3.1 System Design

Systems design is the process of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. Systems design could be seen as the application of systems theory to product development. There is some overlap with the disciplines of systems analysis, systems architecture and systems engineering.

If the broader topic of product development "blends the perspective of marketing, design, and manufacturing into a single approach to product development, then design is the act of taking the marketing information and creating the design of the product to be manufactured. Systems design is therefore the process of defining and developing systems to satisfy specified requirements of the user.

Until the 1990s systems design had a crucial and respected role in the data processing industry. In the 1990s standardization of hardware and software resulted in the ability to build modular systems. The increasing importance of software running on generic platforms has enhanced the discipline of software engineering.

Object-oriented analysis and design methods are becoming the most widely used methods for computer systems design. The UML has become the standard language in object-oriented analysis and design. It is widely used for modeling software systems and is increasingly used for high designing non-software systems and organizations.

The logical design of a system pertains to an abstract representation of the data flows, inputs and outputs of the system. This is often conducted via modeling, using an over-abstract (and sometimes graphical) model of the actual system. In the context of systems design are included. Logical design includes ER Diagrams i.e. Entity Relationship Diagrams.

The physical design relates to the actual input and output processes of the system. This is laid down in terms of how data is input into a system, how it is verified/authenticated, how it is processed, and how it is displayed as output. In Physical design, following requirements about the system are decided.

- Input requirement,
- Output requirements,
- Storage requirements,
- Processing Requirements,
- System control and backup or recovery.

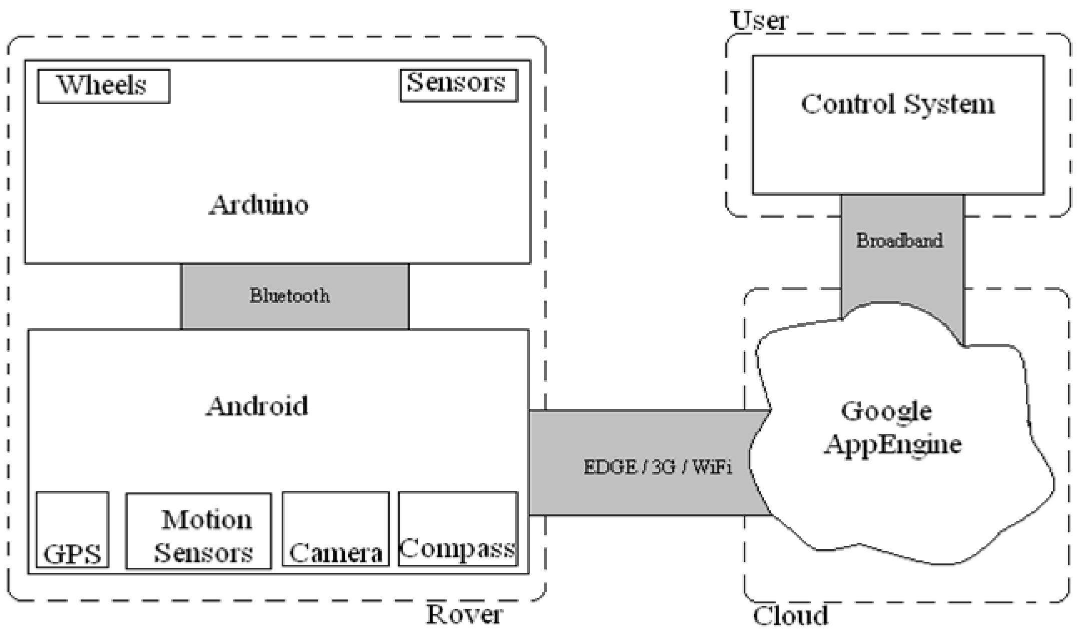
Put another way, the physical portion of systems design can generally be broken down into three sub-tasks:

- User Interface Design
- Data Design
- Process Design

User Interface Design is concerned with how users add information to the system and with how the system presents information back to them. Data Design is concerned with how the data is represented and stored within the system. Finally, Process Design is concerned with how data moves through the system, and with how and where it is validated, secured and/or transformed as it flows into, through and out of the system. At the end of the systems design phase, documentation describing the three sub-tasks is produced and made available for use in the next phase.

Physical design, in this context, does not refer to the tangible physical design of an information system. To use an analogy, a personal computer's physical design involves input via a keyboard, processing within the CPU, and output via a monitor, printer, etc. It would not concern the actual layout of the tangible hardware, which for a PC would be a monitor, CPU, motherboard, hard drive, modems, video/graphics cards, USB slots, etc. It involves a detailed design of a user and a product database structure processor and a control processor. The H/S personal specification is developed for the proposed system.

The initial system design was composed of many sub-systems and was rudimentary in nature. It can be described by the following high-level block diagram:



[Fig 4 : Initial System Design]

While this initially looked like a good scheme, the architecture has a few defects which surfaced only when we actually implemented and tested the architecture. They are enumerated below:

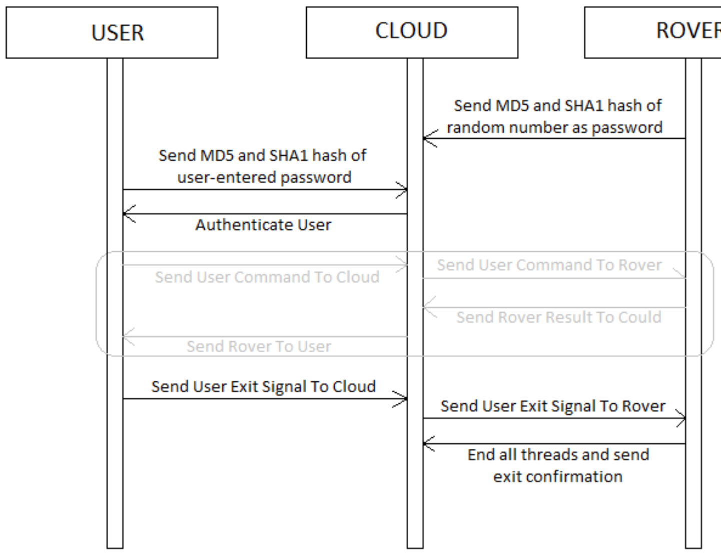
1. Redundancy: Because the cloud acts as a broker between the robot and the user, an unwanted redundancy is introduced. This is because one party, when it needs to inform something to the other party, sends the data to the server on the cloud and not actually to the other party.

2. Latency: The cloud can inform any party about any new data only when the said party polls for new data using HTTP GET. This introduces a latency between the time when the data is posted and when it is retrieved. This latency, ranging from less than a hundred milliseconds to more than a few seconds, although not astronomical, is undesirable. Although, W3C has introduced Web Sockets forHTML5 to enable a full-duplex connection over HTTP in late- 2011, it is not yet widely available.

3. Resource Utilization: Repeated polling means that the rover will post data to the cloud and poll the cloud server for new user data at all times. These posting and polling would take place even when there is no need for such activity. Thus resource is misused to a great extent. And on a mobile platform like Android (a typical phone may have only 1 GHz processor with 512 MB RAM), this mis-utilization of resource affects the system adversely.

To remove these disadvantages, we removed the server application on the cloud and replace it by creating and embedding our own custom HTTP server in the robot. We created a Communication Layer that rests between the user and the core robotic features. The Communication Layer consists of two Java classes. One of them acts as a HTTP server, and the other as a SMS server. These are implemented as Java threads using Android's Service. The HTTP server is created by extending and overwriting a few classes of NanoHTTPD. The SMS server is created simply by using Android’s APIs. For obvious reasons, some features (for example, Video Surveillance) are inaccessible through the SMS server.

The improved system design is given in Fig. 3


USER
CLOUD
ROVER
Send MD5 and SHA1 hash of random number as password
Send MD5 and SHA1 hash of user-entered password
Authenticate User
Send User Command To Cloud
Send User Command To Rover
Send Rover Result To Could
Send Rover To User
Send User Exit Signal To Cloud
Send User Exit Signal To Rover
End all threads and send exit confirmation

[Fig 5: Password authentication security over cloud in the decaprated initial design]

3.2 Input Design

Input facilities the entry of data into the computer system. Input design involves the selection of the best strategy for getting data into the computer system at the right time and as accurately as possible. This is because the most difficult aspect of input design in accuracy .The use of well-defined documents can encourage users to record data accurately without omission.

For example, if a customer's telephone number is a needed input data, the sales order form should have a specific line that is clearly labeled "customer telephone number". Having several lines labeled "customer information" would be less effective. This is because sometimes only the name and address would be filled in leaving out the telephone number.

Input design must capture all the data that the system needs, without introducing any errors. Input errors can be greatly reduced when inputting directly by using appropriate forms for data capture and well designed computer screen layout.

In our project, we are dealing with two types of inputs – one type from the user to control the robot and the other from the Android's sensors about the environment.

The user can provide input in one of the two ways:

- Textual (command-styled) input using SMS
- Graphical (point-and-click) input using the RIA

The SMS server takes three types of inputs that can be provided to get various informations about the device. The legal SMS formats that are taken as inputs are enumerated below:

- GET IP
- GET LOC
- GET STAT

The RIA is a full GUI-based interface that allows the user to provide input by pointing and clicking. The TTS is provided textual input using the standard input.

Apart from the user inputs, the software also makes use of inputs provided by the Android sensors. The main sensors from the Android smart-phone/tablet used for the project are:

- Camera
- GPS
- Compass
- Accelerometer
- Proximity Meter
- Battery Meter
- Thermometer

Voice Recognition uses human voice as input – to convert it to text. In Blob/Object Detection, touch is used to initially pin-point the object to track or blob to follow.

3.3 Output Design

The output is provided in three manners as enumerated below:

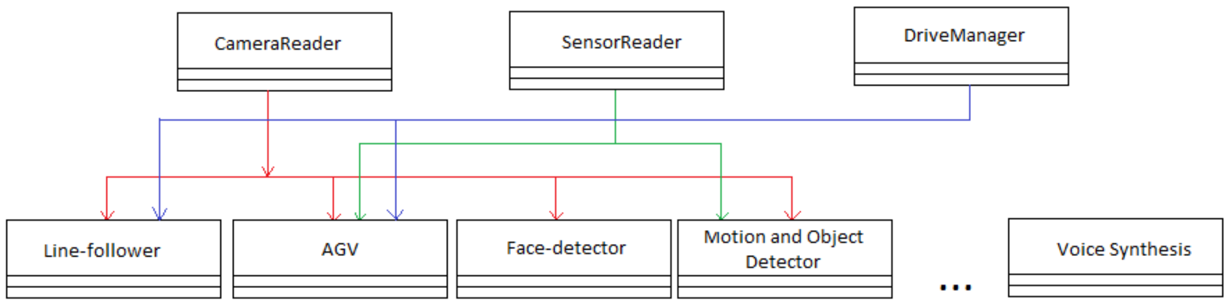
- SMS
- HTTP
- Bluetooth

SMS is used as an output only when the user uses SMS to provide input. If the input is GET IP, the software provides the IP of the device along with port on which the software is running. GET LOC is responded to by replying with an SMS containing the latitude and longitude of the device. Similarly, GET STAT is responded to by providing the battery and temperature status of the device to the sender.

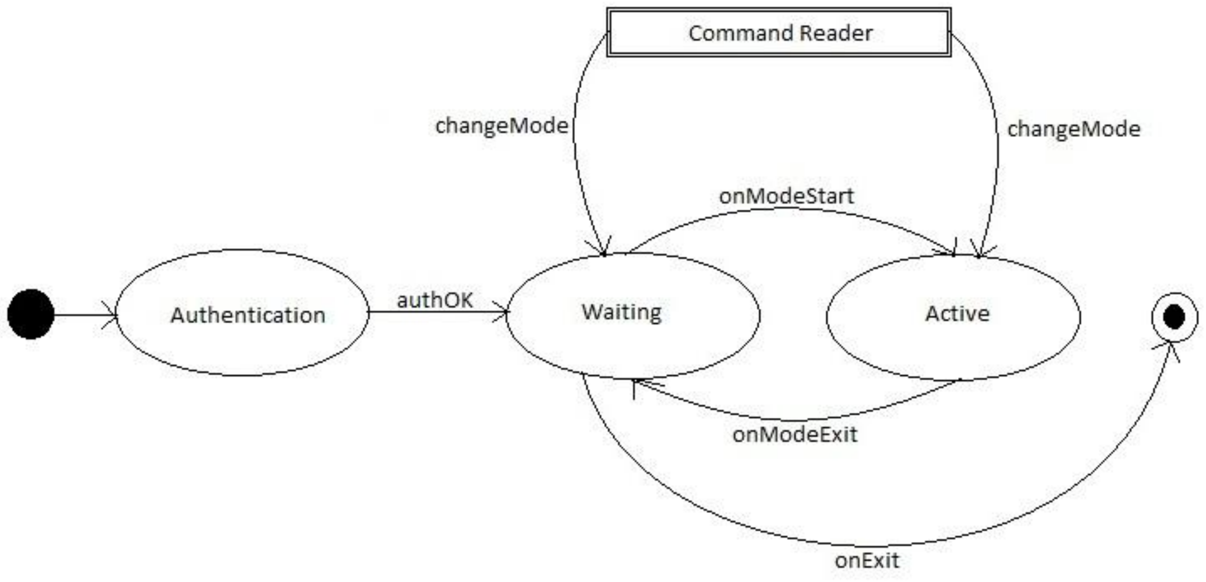
HTTP provides the main interface for the user. It is made as an RIA. The images are sent as text encoded using base64. These images are the rendered on the RIA sequentially using HTML5 Canvas. Texts and other data are provided to the RIA using JSON. JSON or JavaScript Object Notation, is a text-based open standard designed for human-readable data interchange.

Bluetooth is used to provide output to the peripheral robotic device.

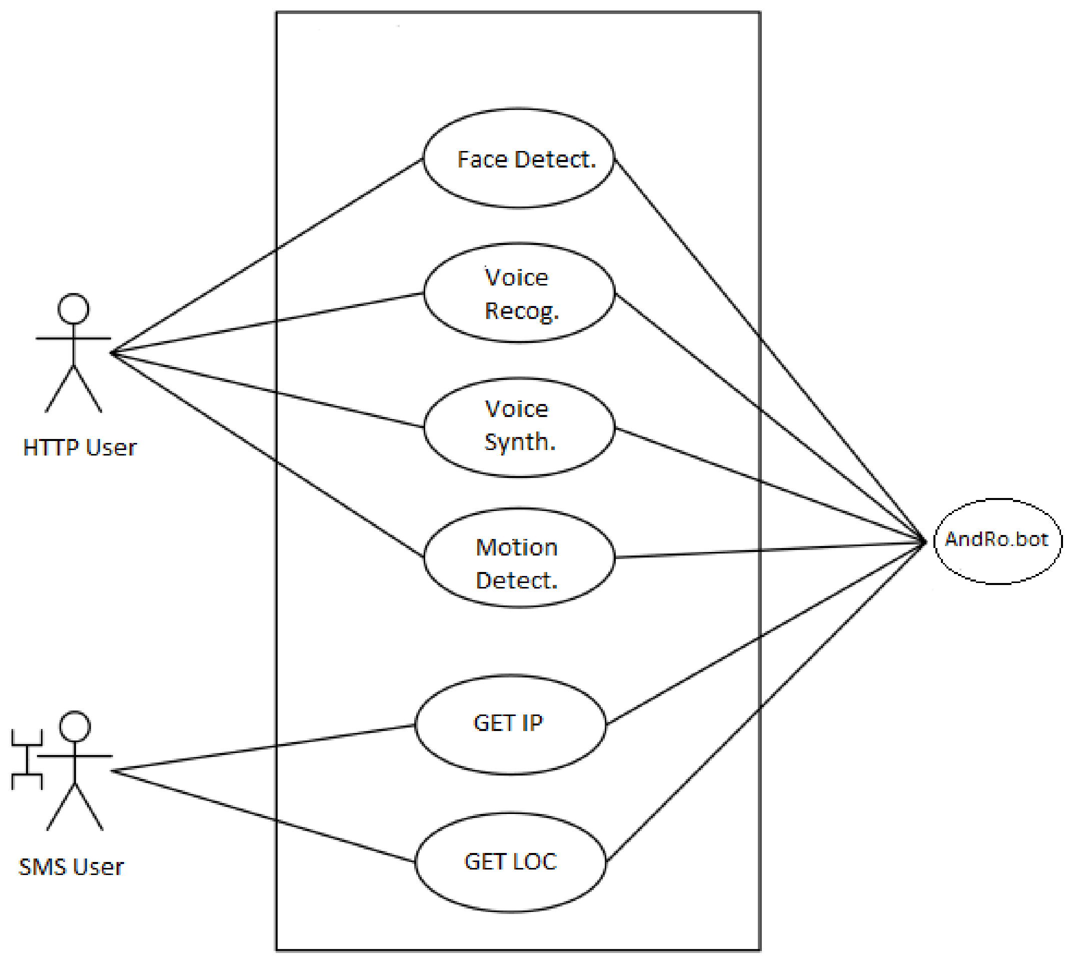
4.4 I/P, O/P and System Design



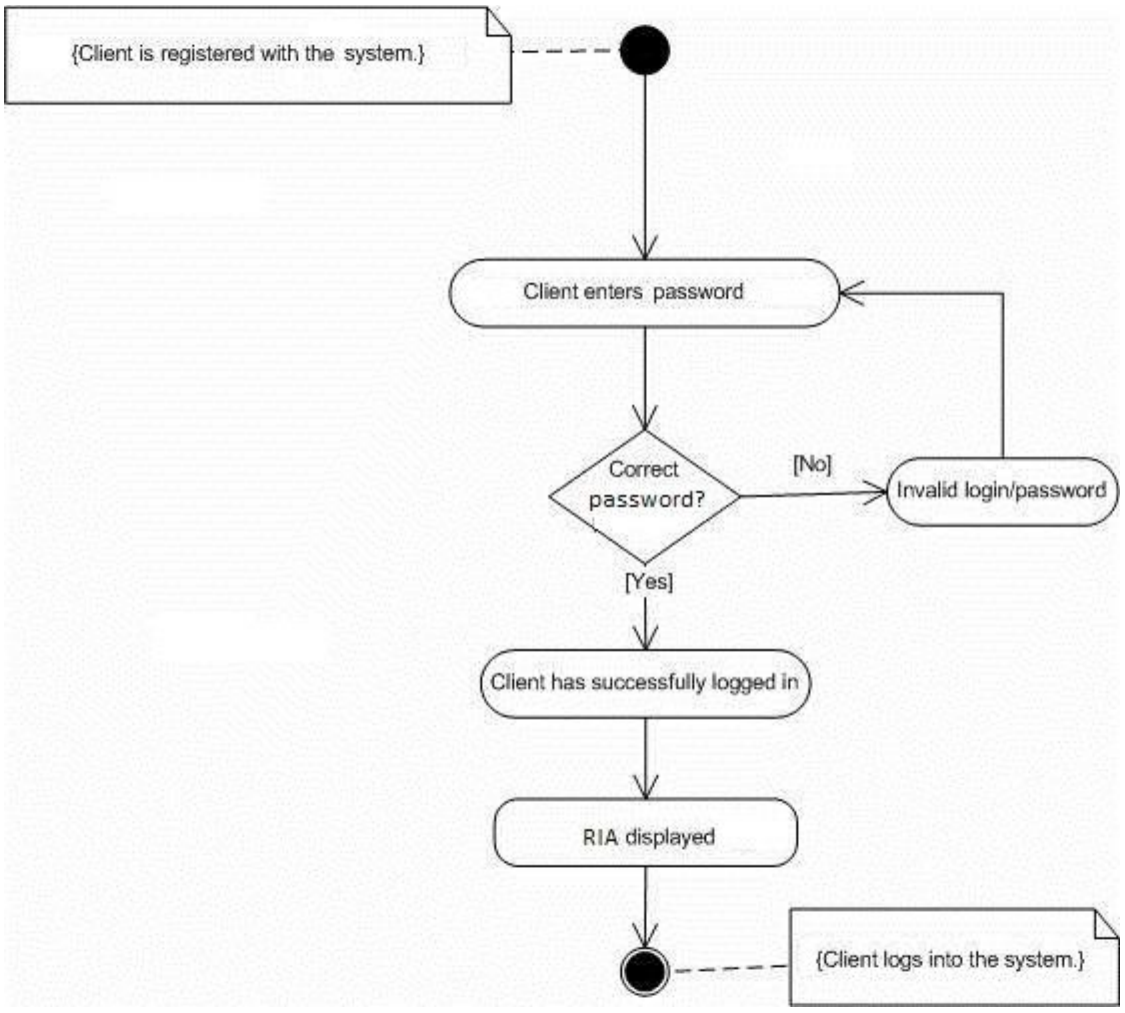
[Fig 6: Class Diagram showing relation between Features and Sensor/Camera Reader]



[Fig 7: State Diagram showing changing of modes in RIA]



[Fig 8: Use Case Diagram showing a few usage scenarios]


{Client is registered with the system.}
Client enters password
Correct password?
[No]
Invalid login/password
[Yes]
Client has successfully logged in
RIA displayed
{Client logs into the system.}

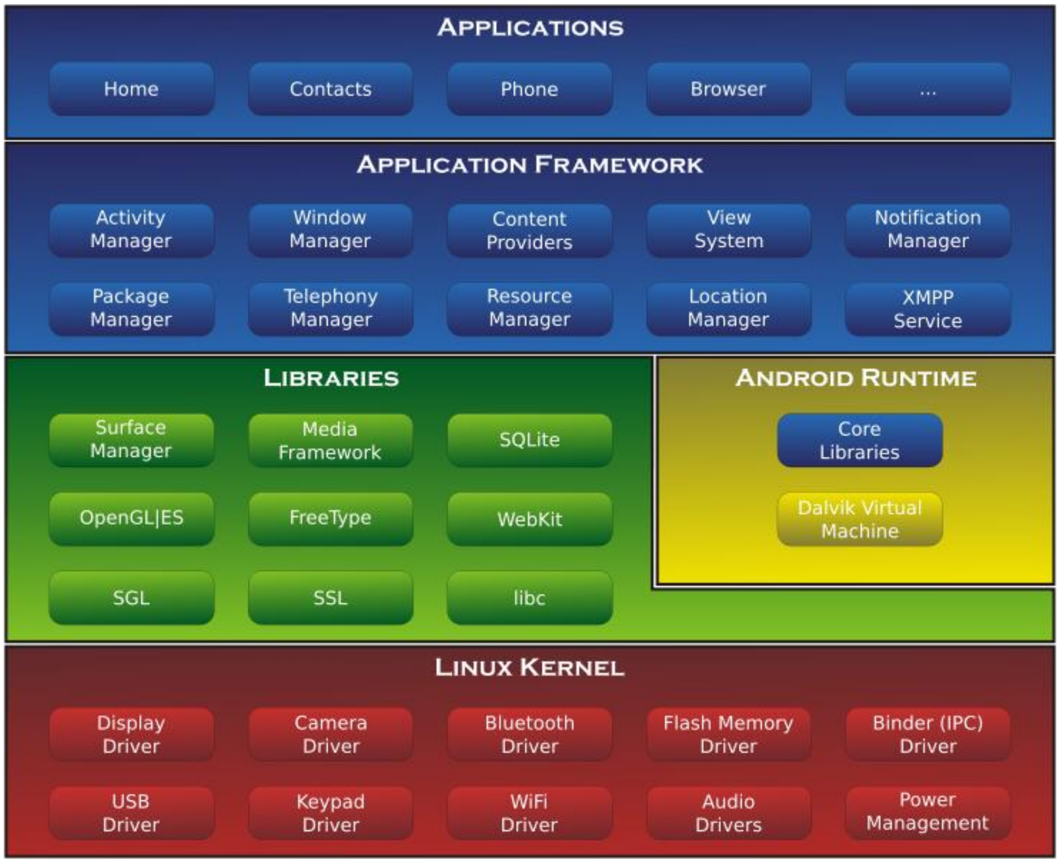
[Fig 9: Activity Diagram showing user login]

CHAPTER 4
IMPLEMENTATION

4.1 Platforms

4.1.1 Android

Android is a Linux-based operating system designed primarily for touch-screen mobile devices such as smart-phones and tablet computers. Initially developed by Android, Inc., which Google backed financially and later bought in 2005, Android was unveiled in 2007 along with the founding of the Open Handset Alliance: a consortium of hardware, software, and telecommunication companies devoted to advancing open standards for mobile devices. The first Android-powered phone was sold in October 2008.



[Fig 10: Android Architecture]

Android is open source and Google releases the code under the Apache License. This open source code and permissive licensing allows the software to be freely modified and distributed by device manufacturers, wireless carriers and enthusiast developers. Additionally, Android has a large community of developers writing applications ("apps") that extend the functionality of devices, written primarily in a customized version of the Java programming language. In October 2012, there were approximately 700,000 apps available for Android, and the estimated number of applications downloaded from Google Play, Android's primary app store, was 25 billion.

These factors have contributed towards making Android the world's most widely used smart-phone platform, overtaking Symbian in the fourth quarter of 2010, and the software of choice for technology companies who require a low-cost, customizable, lightweight operating system for high tech devices without developing one from scratch. As a result, despite being primarily designed for phones and tablets, it has seen additional applications on televisions, games consoles, digital cameras and other electronics. Android's open nature has further encouraged a large community of developers and enthusiasts to use the open source code as a foundation for community-driven projects, which add new features for advanced users or bring Android to devices which were officially released running other operating systems.

Android had a worldwide smart-phone market share of 75% during the third quarter of 2012, with 750 million devices activated in total and 1.5 million activations per day. The operating system's success has made it a target for patent litigation as part of the so-called "smart-phone wars" between technology companies. As of May 2013, a total of 900 million Android devices have been activated and 48 billion apps have been installed from the Google Play store.

Android's user interface is based on direct manipulation, using touch inputs that loosely correspond to real-world actions, like swiping, tapping, pinching and reverse pinching to manipulate on-screen objects. The response to user input is designed to be immediate and provides a fluid touch interface, often using the vibration capabilities of the device to provide haptic feedback to the user. Internal hardware such as accelerometers, gyroscopes and proximity sensors are used by some applications to respond to additional user actions, for example adjusting the screen from portrait to landscape depending on how the device is oriented, or allowing the user to steer a vehicle in a racing game by rotating the device, simulating control of a steering wheel.

Android devices boot to the homescreen, the primary navigation and information point on the device, which is similar to the desktop found on PCs. Android homescreens are typically made up of app icons and widgets; app icons launch the associated app, whereas widgets display live, auto-updating content such as the weather forecast, the user's email inbox, or a news ticker directly on the homescreen. A homescreen may be made up of several pages that the user can swipe back and forth between, though Android's homescreen interface is heavily customisable, allowing the user to adjust the look and feel of the device to their tastes. Third party apps available on Google Play and other app stores can extensively re-theme the homescreen, and even mimic the look of other operating systems, such as Windows Phone. Most manufacturers, and some wireless carriers, customize the look and feel of their Android devices to differentiate themselves from the competition.

Present along the top of the screen is a status bar, showing information about the device and its connectivity. This status bar can be "pulled" down to reveal a notification screen where apps display important information or updates, such as a newly received email or SMS text, in a way that does not immediately interrupt or inconvenience the user. In early versions of Android these notifications could be tapped to open the relevant app, but recent updates have provided enhanced functionality, such as the ability to call a number back directly from the missed call notification without having to open the dialer app first. Notifications are persistent until read or dismissed by the user.

Android has a growing selection of third party applications, which can be acquired by users either through an app store such as Google Play or the Amazon Appstore, or by downloading and installing the application's APK file from a third-party site. The Play Store application allows users to browse, download and update apps published by Google and third-party developers, and is pre-installed on devices that comply with Google's compatibility requirements. The app filters the list of available applications to those that are compatible with the user's device, and developers may restrict their applications to particular carriers or countries for business reasons. Purchases of unwanted applications can be refunded within 15 minutes of the time of download, and some carriers offer direct carrier billing for Google Play application purchases, where the cost of the application is added to the user's monthly bill. As of September 2012, there were more than 675,000 apps available for Android, and the estimated number of applications downloaded from the Play Store was 25 billion.

Applications are developed in the Java language using the Android software development kit (SDK). The SDK includes a comprehensive set of development tools, including a debugger, software libraries, a handset emulator based on QEMU, documentation, sample code, and tutorials. The officially supported integrated development environment (IDE) is Eclipse using the Android Development Tools (ADT) plugin. Other development tools are available, including a Native Development Kit for applications or extensions in C or C++, Google App Inventor, a visual environment for novice programmers, and various cross platform mobile web applications frameworks.

In order to work around limitations on reaching Google services due to Internet censorship in the People's Republic of China, Android devices sold in the PRC are generally customized to use state approved services instead.

Android consists of a kernel based on Linux kernel version 2.6 and, from Android 4.0 *Ice Cream Sandwich* onwards, version 3.x, with middleware, libraries and APIs written in C, and application running on an application framework which includes Java-compatible libraries based on Apache Harmony. Android uses the Dalvik virtual machine with just-in-time compilation to run Dalvik 'dex-code' (Dalvik Executable), which is usually translated from Java bytecode. The main hardware platform for Android is the ARM architecture. There is support for x86 from the Android-x86 project, and Google TV uses a special x86 version of Android. In 2013, Freescale announced Android on its i.MX processor, i.MX5X and i.MX6X series.

Android's Linux kernel has further architecture changes by Google outside the typical Linux kernel development cycle. Android does not have a native X Window System by default nor does it support the full set of standard GNU libraries, and this makes it difficult to port existing Linux applications or libraries to Android. Support for simple C and SDL applications is possible by injection of a small Java shim and usage of the JNI like, for example, in the Jagged Alliance 2 port for Android.

Certain features that Google contributed back to the Linux kernel, notably a power management feature called "wakelocks", were rejected by mainline kernel developers partly because they felt that Google did not show any intent to maintain its own code. Google announced in April 2010 that they would hire two employees to work with the Linux kernel community, but Greg Kroah-Hartman, the current Linux kernel maintainer for the stable branch, said in December 2010 that he was concerned that Google was no longer trying to get their code changes included in mainstream Linux. Some Google Android developers hinted that "the Android team was getting fed up with the process," because they were a small team and had more urgent work to do on Android.

In August 2011, Linus Torvalds said that "eventually Android and Linux would come back to a common kernel, but it will probably not be for four to five years". In December 2011, Greg Kroah-Hartman announced the start of the Android Mainlining Project, which aims to put some Android drivers, patches and features back into the Linux kernel, starting in Linux 3.3. Linux included the auto-sleep and wake-locks capabilities in the 3.5 kernel, after many previous attempts at merger. The interfaces are the same but the upstream Linux implementation allows for two different suspend modes: to memory (the traditional suspend that Android uses), and to disk (hibernate, as it is known on the desktop). The merge will be complete starting with Kernel 3.8, Google has opened a public code repository that contains their experimental work to re-base Android off Kernel 3.8.

The flash storage on Android devices is split into several partitions, such as "/system" for the operating system itself and "/data" for user data and app installations. In contrast to desktop Linux distributions, Android device owners are not given root access to the operating system and sensitive partitions such as system are read-only. However, root access can be obtained by exploiting security flaws in Android, which is used frequently by the open source community to enhance the capabilities of their devices, but also by malicious parties to install viruses and malware.

Whether or not Android counts as a Linux distribution is a widely debated topic, with the Linux Foundation and Chris DiBona, Google's open source chief, in favor. Others, such as Google engineer Patrick Brady disagree, noting the lack of support for many GNU tools, including glibc, in Android.

Since Android devices are usually battery-powered, Android is designed to manage memory (RAM) to keep power consumption at a minimum, in contrast to desktop operating systems which generally assume they are connected to unlimited mains electricity. When an Android app is no longer in use, the system will automatically suspend it in memory - while the app is still technically "open," suspended apps consume no resources (e.g. battery power or processing power) and sit idly in the background until needed again. This has the dual benefit of increasing the general responsiveness of Android devices, since apps don't need to be closed and reopened from scratch each time, but also ensuring background apps don't waste power needlessly.

Android manages the apps stored in memory automatically: when memory is low, the system will begin killing apps and processes that have been inactive for a while, in reverse order since they were last used (i.e. oldest first). This process is designed to be invisible to the user, such that users do not need to manage memory or the killing of apps themselves. However, confusion over Android memory management has resulted in third-party task killers becoming popular on the Google Play store; these third-party task killers are generally regarded as doing more harm than good.

Google provides major updates, incremental in nature, to Android every six to nine months, which most devices are capable of receiving over the air. The latest major update is Android 4.2 *Jelly Bean*.

Compared to its chief rival mobile operating system, namely iOS, Android updates are typically slow to reach actual devices. For devices not under the Nexus brand, updates often arrive months from the time the given version is officially released. This is caused partly due to the extensive variation in hardware of Android devices, to which each update must be specifically tailored, as the official Google source code only runs on their flagship Nexus devices. Porting Android to specific hardware is a time- and resource-consuming process for device manufacturers, who prioritize their newest devices and often leave older ones behind. Hence, older smart-phones are frequently not updated if the manufacturer decides it is not worth their time, regardless of whether the phone is capable of running the update. This problem is compounded when manufacturers customize Android with their own interface and apps, which must be reapplied to each new release. Additional delays can be introduced by wireless carriers who, after receiving updates from manufacturers, further customize and brand Android to their needs and conduct extensive testing on their networks before sending the update out to users.

The lack of after-sale support from manufacturers and carriers has been widely criticized by consumer groups and the technology media. Some commentators have noted that the industry has a financial incentive *not* to update their devices, as the lack of updates for existing devices fuels the purchase of newer ones, an attitude described as "insulting". *The Guardian* has complained that the complicated method of distribution for updates is only complicated because manufacturers and carriers have designed it that way. In 2011, Google partnered with a number of industry players to announce an "Android Update Alliance", pledging to deliver timely updates for every device for 18 months after its release. As of 2013, this alliance has never been mentioned since.

Android has an active community of developers and enthusiasts who use the Android source code to develop and distribute their own modified versions of the operating system. These community-developed releases often bring new features and updates to devices faster than through the official manufacturer/carrier channels, albeit without as extensive testing or quality assurance; provide continued support for older devices that no longer receive official updates; or bring Android to devices that were officially released running other operating systems, such as the HP TouchPad. Community releases often come pre-rooted and contain modifications unsuitable for non-technical users, such as the ability to overclock or over/undervolt the device's processor. CyanogenMod is the most widely used community firmware, and acts as a foundation for numerous others.

Historically, device manufacturers and mobile carriers have typically been unsupportive of third-party firmware development. Manufacturers express concern about improper functioning of devices running unofficial software and the support costs resulting from this. Moreover, modified firmwares such as CyanogenMod sometimes offer features, such as tethering, for which carriers would otherwise charge a premium. As a result, technical obstacles including locked bootloaders and restricted access to root permissions are common in many devices. However, as community-developed software has grown more popular, and following a statement by the Librarian of Congress in the United States that permits the "jailbreaking" of mobile devices, manufacturers and carriers have softened their position regarding third party development, with some, including HTC, Motorola, Samsung and Sony providing support and encouraging development. As a result of this, over time the need to circumvent hardware restrictions to install unofficial firmware has lessened as an increasing number of devices are shipped with unlocked or unlock*able* bootloaders, similar to the Nexus series of phones, although usually requiring that users waive their devices' warranties to do so. However, despite manufacturer acceptance, some carriers in the US still require that phones are locked down.

The unlocking and "hackability" of smartphones and tablets remains a source of tension between the community and industry, with the community arguing that unofficial development is increasingly important given the failure of industry to provide timely updates and/or continued support to their devices.

Android applications run in a sandbox, an isolated area of the system that does not have access to the rest of the system's resources, unless access permissions are explicitly granted by the user when the application is installed. Before installing an application, the Play Store displays all required permissions: a game may need to enable vibration or save data to an SD card, for example, but should not need to read SMS messages or access the phonebook. After reviewing these permissions, the user can choose to accept or refuse them, installing the application only if they accept.

The sandboxing and permissions system lessens the impact of vulnerabilities and bugs in applications, but developer confusion and limited documentation has resulted in applications routinely requesting unnecessary permissions, reducing its effectiveness. Several security firms, such as Lookout Mobile Security, AVG Technologies, and McAfee, have released antivirus software for Android devices. This software is ineffective as sandboxing also applies to such applications, limiting their ability to scan the deeper system for threats.

Research from security company Trend Micro lists premium service abuse as the most common type of Android malware, where text messages are sent from infected phones to premium-rate telephone numbers without the consent or even knowledge of the user. Other malware displays unwanted and intrusive adverts on the device, or sends personal information to unauthorized third parties. Security threats on Android are reportedly growing exponentially; however, Google engineers have argued that the malware and virus threat on Android is being exaggerated by security companies for commercial reasons, and have accused the security industry of playing on fears to sell virus protection software to users. Google maintains that dangerous malware is actually extremely rare, and a survey conducted by F-Secure showed that only 0.5% of Android malware reported had come from the Google Play store.

Google currently uses their Google Bouncer malware scanner to watch over and scan the Google Play store apps. It is intended to flag up suspicious apps and warn users of any potential issues with an application before they download it. Android version 4.2 *Jelly Bean* was released in 2012 with enhanced security features, including a malware scanner built into the system, which works in combination with Google Play but can scan apps installed from third party sources as well, and an alert system which notifies the user when an app tries to send a premium-rate text message, blocking the message unless the user explicitly authorizes it.

Android smartphones have the ability to report the location of Wi-Fi access points, encountered as phone users move around, to build databases containing the physical locations of hundreds of millions of such access points. These databases form electronic maps to locate smartphones, allowing them to run apps like Foursquare, Google Latitude, Facebook Places, and to deliver location-based ads. Third party monitoring software such as TaintDroid, an academic research-funded project, can, in some cases, detect when personal information is being sent from applications to remote servers.

The open source nature of Android allows security contractors to take existing devices and adapt them for highly secure uses. For example Samsung has worked with General Dynamics through their Open Kernel Labs acquisition to rebuild *Jelly Bean* on top of their hardened microvisor for the "Knox" project.

Android received a lukewarm reaction when it was unveiled in 2007. Although analysts were impressed with the respected technology companies that had partnered with Google to form the Open Handset Alliance, it was unclear whether mobile phone manufacturers would be willing to replace their existing operating systems with Android. The idea of an open source, Linux-based development platform sparked interest, but there were additional worries about Android facing strong competition from established players in the smart-phone market, such as Nokia and Microsoft, and rival Linux mobile operating systems that were in development. These established players were skeptical: Nokia was quoted as saying "we don't see this as a threat," and a member of Microsoft'sWindows Mobile team stated "I don't understand the impact that they are going to have."

Since then Android has grown to become the most widely used smart-phone operating system and "one of the fastest mobile experiences available." Reviewers have highlighted the open source nature of the operating system as one of its defining strengths, allowing companies such as Amazon (Kindle Fire), Barnes & Noble (Nook), Ouya, Baidu, and others to fork the software and release hardware running their own customized version of Android. As a result, it has been described by technology website Ars Technica as "practically the default operating system for launching new hardware" for companies without their own mobile platforms. This openness and flexibility is also present at the level of the end user: Android allows extensive customization of devices by their owners and apps are freely available from non-Google app stores and third party websites. These have been cited as among the main advantages of Android phones over others.

Despite Android's popularity, including an activation rate three times that of iOS, there have been reports that Google has not been able to leverage their other products and web services successfully to turn Android into the money maker that analysts had expected. The Verge suggested that Google is losing control of Android due to the extensive customization and proliferation of non-Google apps and services - for instance the Amazon Kindle Fire points users to the Amazon app store that competes directly with the Google Play store. Google SVP Andy Rubin, who was replaced as head of the Android division in March 2013, has been blamed for failing to establish a lucrative partnership with cell phone makers. The chief beneficiary of Android has been Samsung, whose Galaxy brand has surpassed that of Android in terms of brand recognition since 2011. Meanwhile other Android manufacturers have struggled since 2011, such as LG, HTC, and Google's own Motorola Mobility (whose partnership with Verizon Wireless to push the "DROID" brand has faded since 2010). Ironically, while Google directly earns nothing from the sale of each Android device, Microsoft and Apple have successfully sued to extract patent royalty payments from Android handset manufacturers.

The open and customizable nature of Android allows it to be used on other electronics, including laptops and netbooks, smartbooks, smart TVs (Google TV) and cameras (Nikon Coolpix S800c and Galaxy Camera). In addition, the Android operating system has seen applications on smart glasses (Google Glass), wristwatches, headphones, car CD and DVD players, mirrors, portable media players and landlines and Voice over IP phones. Ouya, an upcoming videogames console running Android, became one of the most successful Kickstarter campaigns, crowd-funding US$8.5m for its development, and was later followed by other Android-based video games consoles such as Project Shield from Nvidia.

In 2011, Google demonstrated "Android@Home", a new home automaton technology which uses Android to control a range of household devices including light switches, power sockets and thermostats. Prototype light bulbs were announced that could be controlled from an Android phone or tablet, but Android head Andy Rubin was cautious to note that "turning a lightbulb on and off is nothing new," pointing to numerous failed home automation services. Google, he said, was thinking more ambitiously and the intention was to use their position as a cloud services provider to bring Google products into customers' homes.

4.2 Languages

4.2.1. HTML5

HTML5 is a markup language for structuring and presenting content for the World Wide Web and a core technology of the Internet. It is the fifth revision of the HTML standard (created in 1990 and standardized as HTML 4 as of 1997) and, as of December 2012, is a W3C Candidate Recommendation. Its core aims have been to improve the language with support for the latest multimedia while keeping it easily readable by humans and consistently understood by computers and devices (web browsers, parsers, etc.). HTML5 is intended to subsume not only HTML 4, but also XHTML 1 and DOM Level 2 HTML.

Following its immediate predecessors HTML 4.01 and XHTML 1.1, HTML5 is a response to the observation that the HTML and XHTML in common use on the World Wide Web are a mixture of features introduced by various specifications, along with those introduced by software products such as web browsers, those established by common practice, and the many syntax errors in existing web documents. It is also an attempt to define a single markup language that can be written in either HTML or XHTML syntax. It includes detailed processing models to encourage more interoperable implementations; it extends, improves and rationalizes the markup available for documents, and introduces markup and application programming interfaces (APIs) for complex web applications. For the same reasons, HTML5 is also a potential candidate for cross-platform mobile applications. Many features of HTML5 have been built with the consideration of being able to run on low-powered devices such as smart-phones and tablets. In December 2011, research firm Strategy Analytics forecast sales of HTML5 compatible phones will top 1 billion in 2013.

In particular, HTML5 adds many new syntactic features. These include the new <video>, <audio> and <canvas> elements, as well as the integration of scalable vector graphics (SVG) content (that replaces the uses of generic <object> tags) and MathML for mathematical formulas. These features are designed to make it easy to include and handle multimedia and graphical content on the web without having to resort to proprietary plugins and APIs. Other new elements, such as <section>, <article>, <header> and <nav>, are designed to enrich the semantic content of documents. New attributes have been introduced for the same purpose, while some elements and attributes have been removed. Some elements, such as <a>, <cite> and <menu> have been changed, redefined or standardized. The APIs and Document Object Model (DOM) are no longer afterthoughts, but are fundamental parts of the HTML5 specification. HTML5 also defines in some detail the required processing for invalid documents so that syntax errors will be treated uniformly by all conforming browsers and other user agents.

The Web Hypertext Application Technology Working Group (WHATWG) began work on the new standard in 2004. At that time, HTML 4.01 had not been updated since 2000, and the World Wide Web Consortium (W3C) was focusing future developments on XHTML 2.0. In 2009, the W3C allowed the XHTML 2.0 Working Group's charter to expire and decided not to renew it. W3C and WHATWG are currently working together on the development of HTML5.

While HTML5 is often compared to Flash, the two technologies are very different. Both include features for playing audio and video within web pages, and for using Scalable Vector Graphics. HTML5 on its own cannot be used for animation and interactivity — it must be supplemented with CSS3 or JavaScript. There are many Flash capabilities that have no direct counterpart in HTML5.

Although HTML5 has been well known among web developers for years, it became the topic of mainstream media around April 2010 after Apple Inc's then-CEO Steve Jobs issued a public letter titled "Thoughts on Flash" where he concludes that "[Adobe] Flash is no longer necessary to watch video or consume any kind of web content" and that "new open standards created in the mobile era, such as HTML5, will win". This sparked a debate in web development circles where some suggested that while HTML5 provides enhanced functionality, developers must consider the varying browser support of the different parts of the standard as well as other functionality differences between HTML5 and Flash. In early November 2011, Adobe announced that it will discontinue development of Flash for mobile devices and reorient its efforts in developing tools utilizing HTML5.

HTML5 introduces elements and attributes that reflect typical usage on modern websites. Some of them are semantic replacements for common uses of generic block (<div>) and inline (<span>) elements, for example <nav> (website navigation block), <footer>(usually referring to bottom of web page or to last lines of HTML code), or <audio> and <video> instead of <object>. Some deprecated elements from HTML 4.01 have been dropped, including purely presentational elements such as <font> and<center>, whose effects have long been superseded by the much more powerful Cascading Style Sheets. There is also a renewed emphasis on the importance of DOM scripting (e.g., JavaScript) in Web behavior.

The HTML5 syntax is no longer based on SGML despite the similarity of its markup. It has, however, been designed to be backward compatible with common parsing of older versions of HTML. It comes with a new introductory line that looks like an SGML document type declaration, <!DOCTYPE html>, which triggers the standards-compliant rendering mode. As of 5 January 2009, HTML5 also includes *Web Forms 2.0*, a previously separate WHATWG specification.

In addition to specifying markup, HTML5 specifies scripting application programming interfaces (APIs) that can be used with JavaScript. Existing document object model (DOM) interfaces are extended and *de facto* features documented. There are also new APIs, such as:

- The canvas element for immediate mode 2D drawing. See Canvas 2D API Specification 1.0 specification
- Timed media playback
- Offline Web Applications
- Document editing
- Drag-and-drop
- Cross-document messaging
- Browser history management
- MIME type and protocol handler registration
- Microdata

- Web Storage, a key-value pair storage framework that provides behaviour similar to cookies but with larger storage capacity and improved API.

Not all of the above technologies are included in the W3C HTML5 specification, though they are in the WHATWG HTML specification. Some related technologies, which are not part of either the W3C HTML5 or the WHATWG HTML specification, are as follows. The W3C publishes specifications for these separately:

- Geolocation
- Web SQL Database, a local SQL Database (no longer maintained).
- The Indexed Database API, an indexed hierarchical key-value store (formerly WebSimpleDB).
- HTML5 File API, handles file uploads and file manipulation.
- Directories and System, an API intended to satisfy client-side-storage use cases not well served by databases.
- File Writer, an API for writing to files from web applications.
- Web Audio API, a high-level JavaScript API for processing and synthesizing audio in web applications.

HTML5 alone cannot provide animation within web pages. Either JavaScript or CSS3 is necessary for animating HTML elements. Animation is also possible using JavaScript and HTML 4 and within SVG elements through SMIL, although browser support of the latter remains uneven as of 2011.

4.2.2 CSS3

Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation semantics (the look and formatting) of a document written in a markup language. Its most common application is to style web pages written in HTML and XHTML, but the language can also be applied to any kind of XML document, including plain XML, SVG and XUL.

CSS is designed primarily to enable the separation of document content (written in HTML or a similar markup language) from document presentation, including elements such as the layout, colors, and fonts. This separation can improve content accessibility, provide more flexibility and control in the specification of presentation characteristics, enable multiple pages to share formatting, and reduce complexity and repetition in the structural content (such as by allowing for tableless web design). CSS can also allow the same markup page to be presented in different styles for different rendering methods, such as on-screen, in print, by voice (when read out by a speech-based browser or screen reader) and on Braille-based, tactile devices. It can also be used to allow the web page to display differently depending on the screen size or device on which it is being viewed. While the author of a document typically links that document to a CSS style sheet, readers can use a different style sheet, perhaps one on their own computer, to override the one the author has specified.

CSS specifies a priority scheme to determine which style rules apply if more than one rule matches against a particular element. In this so-called *cascade*, priorities or *weights* are calculated and assigned to rules, so that the results are predictable.

The CSS specifications are maintained by the World Wide Web Consortium (W3C). Internet media type (MIME type) text/css is registered for use with CSS by RFC 2318 (March 1998), and they also operate a free CSS validation service.

CSS has a simple syntax and uses a number of English keywords to specify the names of various style properties.

A style sheet consists of a list of *rules*. Each rule or rule-set consists of one or more*selectors*, and a *declaration block*.

Because not all browsers correctly parse CSS code, developed coding techniques known as CSS hacks can either filter specific browsers or target specific browsers (generally both are known as CSS filters). The former can be defined as CSS filtering hacks and the latter can be defined as CSS targeting hacks. Both can be used to hide or show parts of the CSS to different browsers. This is achieved either by exploiting CSS-handling quirks or bugs in the browser, or by taking advantage of lack of support for parts of the CSS specifications. Using CSS filters, some designers have gone as far as delivering different CSS to certain browsers to ensure designs render as expected. Because very early web browsers were either completely incapable of handling CSS, or rendered CSS very poorly, designers today often routinely use CSS filters that completely prevent these browsers from accessing any of the CSS. Internet Explorer support for CSS began with IE 3.0 and increased progressively with each version. By 2008, the first Beta of Internet Explorer 8offered support for CSS 2.1 in its best web standards mode.

An example of a well-known CSS browser bug is the Internet Explorer box model bug, where box widths are interpreted incorrectly in several versions of the browser, resulting in blocks that are too narrow when viewed in Internet Explorer, but correct in standards-compliant browsers. The bug can be avoided in Internet Explorer 6 by using the correct doctype in (X)HTML documents. CSS hacks and CSS filters are used to compensate for bugs such as this, just one of hundreds of CSS bugs that have been documented in various versions of Netscape, Mozilla Firefox, Opera, and Internet Explorer (including Internet Explorer 7).

Even when the availability of CSS-capable browsers made CSS a viable technology, the adoption of CSS was still held back by designers' struggles with browsers' incorrect CSS implementation and patchy CSS support. Even today, these problems continue to make the business of CSS design more complex and costly than it was intended to be, and cross-browser testing remains a necessity. Other reasons for the continuing non-adoption of CSS are: its perceived complexity, authors' lack of familiarity with CSS syntax and required techniques, poor support from authoring tools, the risks posed by inconsistency between browsers and the increased costs of testing.

Currently there is strong competition between the WebKit layout engine used in Apple Safari and Google Chrome, the similar KHTML engine used in KDE's Konqueror browser and Mozilla's Gecko layout engine used in Firefox — each of them is leading in different aspects of CSS. As of August 2009, Internet Explorer 8, Firefox 2 and 3 have reasonably complete levels of implementation of CSS 2.1.

4.2.3 JavaScript

JavaScript (JS) is an interpreted computer programming language. It was originally implemented as part of web browsers so that client-side scripts could interact with the user, control the browser, communicate asynchronously, and alter the document content that was displayed. Now though it has many uses involving popular game development and the creation of applications.

JavaScript is a prototype-based scripting language that is dynamic, weakly typed, and has first-class functions. Its syntax was influenced by the language C. JavaScript copies many names and naming conventions from Java, but the two languages are otherwise unrelated and have very different semantics. The key design principles within JavaScript are taken from the Self and Scheme programming languages. It is a multi-paradigm language, supporting object-oriented, imperative, and functional programming styles.

JavaScript's use in applications outside of web pages—for example, in PDF documents, site-specific browsers, and desktop widgets—is also significant. Newer and faster JavaScript VMs and frameworks built upon them (notably Node.js) have also increased the popularity of JavaScript for server-side web applications.

JavaScript was formalized in the ECMAScript language standard and is primarily used as part of a web browser (client-side JavaScript). This enables programmatic access to computational objects within a host environment.

The most common use of JavaScript is to write functions that are embedded in or included from HTML pages and that interact with the Document Object Model (DOM) of the page. Some simple examples of this usage are:

- Loading new page content or submitting data to the server via AJAX without reloading the page (for example, a social network might allow the user to post status updates without leaving the page)
- Animation of page elements, fading them in and out, resizing them, moving them, etc.
- Interactive content, for example games, and playing audio and video
- Validating input values of a web form to make sure that they are acceptable before being submitted to the server.
- Transmitting information about the user's reading habits and browsing activities to various websites. Web pages frequently do this for web analytics, ad tracking, personalization or other purposes.

Because JavaScript code can run locally in a user's browser (rather than on a remote server), the browser can respond to user actions quickly, making an application more responsive. Furthermore, JavaScript code can detect user actions which HTML alone cannot, such as individual keystrokes. Applications such as Gmail take advantage of this: much of the user-interface logic is written in JavaScript, and JavaScript dispatches requests for information (such as the content of an e-mail message) to the server. The wider trend of Ajax programming similarly exploits this strength.

A JavaScript engine (also known as *JavaScript interpreter* or *JavaScript implementation*) is an interpreter that interprets JavaScript source code and executes the script accordingly. The first JavaScript engine was created by Brendan Eich at Netscape Communications Corporation, for the Netscape Navigator web browser. The engine, code-named SpiderMonkey, is implemented in C. It has since been updated (in JavaScript 1.5) to conform to ECMA-262 Edition 3. The Rhino engine, created primarily by Norris Boyd (formerly of Netscape; now at Google) is a JavaScript implementation in Java. Rhino, like SpiderMonkey, is ECMA-262 Edition 3 compliant.

A web browser is by far the most common host environment for JavaScript. Web browsers typically create "host objects" to represent the Document Object Model (DOM) in JavaScript. The web server is another common host environment. A JavaScript web-server would typically expose host objects representing HTTP request and response objects, which a JavaScript program could then interrogate and manipulate to dynamically generate web pages.

Because JavaScript is the only language that the most popular browsers share support for, it has become a target language for many frameworks in other languages, even though JavaScript was never intended to be such a language. Despite the performance limitations inherent to its dynamic nature, the increasing speed of JavaScript engines has made the language a surprisingly feasible compilation target.

Because JavaScript runs in widely varying environments, an important part of testing and debugging is to test and verify that the JavaScript works across multiple browsers.

The DOM interfaces for manipulating web pages are not part of the ECMAScript standard, or of JavaScript itself. Officially, the DOM interfaces are defined by a separate standardization effort by the W3C; in practice, browser implementations differ from the standards and from each other, and not all browsers execute JavaScript.

To deal with these differences, JavaScript authors can attempt to write standards-compliant code which will also be executed correctly by most browsers; failing that, they can write code that checks for the presence of certain browser features and behaves differently if they are not available. In some cases, two browsers may both implement a feature but with different behavior, and authors may find it practical to detect what browser is running and change their script's behavior to match. Programmers may also use libraries or toolkits which take browser differences into account.

Furthermore, scripts may not work for some users. For example, a user may:

- use an old or rare browser with incomplete or unusual DOM support,
- use a PDA or mobile phone browser which cannot execute JavaScript,
- have JavaScript execution disabled as a security precaution,
- use a speech browser due to, for example, a visual disability.

To support these users, web authors can try to create pages which degrade gracefully on user agents (browsers) which do not support the page's JavaScript. In particular, the page should remain usable albeit without the extra features that the JavaScript would have added. An alternative approach that many find preferable is to first author content using basic technologies that work in all browsers, then enhance the content for users that have JavaScript enabled. This is known as progressive enhancement.

Assuming that the user has not disabled its execution, client-side web JavaScript should be written to enhance the experiences of visitors with visual or physical disabilities, and certainly should avoid denying information to these visitors.

Screen readers, used by the blind and partially sighted, can be JavaScript-aware and so may access and read the page DOM after the script has altered it. The HTML should be as concise, navigable and semantically rich as possible whether the scripts have run or not. JavaScript should not be totally reliant on mouse-specific events so as to deny its benefits to users who either cannot use a mouse or who choose to favor the keyboard for whatever reason. Equally, although hyperlinks and webforms can be navigated and operated from the keyboard, accessible JavaScript should not require keyboard events either. There are device-independent events such as onfocus and onchange that are preferable in most cases.

JavaScript should not be used in a way that is confusing or disorienting to any web user. For example, using script to alter or disable the normal functionality of the browser, such as by changing the way the back-button or the refresh event work, is usually best avoided. Equally, triggering events that the user may not be aware of reduces the user's sense of control as do unexpected scripted changes to the page content.

Often the process of making a complex web page as accessible as possible becomes a nontrivial problem where issues become matters of debate and opinion, and where compromises are necessary in the end. However, user agents and assistive technologies are constantly evolving and new guidelines and relevant information are continually being published on the web.

A common misconception is that JavaScript is similar or closely related to Java. It is true that both have a C-like syntax, the C language being their most immediate common ancestor language. They are both object-oriented and typically sandboxed (when used inside a browser). In addition, JavaScript was designed with Java's syntax and standard library in mind. In particular, all Java keywords were reserved in original JavaScript, JavaScript's standard library follows Java's naming conventions, and JavaScript's Math and Date objects are based on classes from Java 1.0.

However, the similarities end there. Java has static typing; JavaScript's typing is dynamic (meaning a variable can hold an object of any type and cannot be restricted). JavaScript is weakly typed while Java is strongly typed. Java is loaded from compiled bytecode; JavaScript is loaded as human-readable source code. Java's objects are class-based; JavaScript's are prototype-based. JavaScript also has many functional programming features based on the Scheme language.

4.2.4 Java

Java is a general-purpose, concurrent, class-based, object-oriented computer programming language that is specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere" (WORA), meaning that code that runs on one platform does not need to be recompiled to run on another. Java applications are typically compiled to bytecode (class file) that can run on any Java virtual machine (JVM) regardless of computer architecture. Java is, as of 2012, one of the most popular programming languages in use, particularly for client-server web applications, with a reported 10 million users. Java was originally developed by James Gosling at Sun Microsystems (which has since merged into Oracle Corporation) and released in 1995 as a core component of Sun

Microsystems' Java platform. The language derives much of its syntax from C and C++, but it has fewer low-level facilities than either of them.

The original and reference implementation Java compilers, virtual machines, and class libraries were developed by Sun from 1991 and first released in 1995. As of May 2007, in compliance with the specifications of the Java Community Process, Sun relicensed most of its Java technologies under the GNU General Public License. Others have also developed alternative implementations of these Sun technologies, such as the GNU Compiler for Java and GNU Classpath.

There were five primary goals in the creation of the Java language:

1. It should be "simple, object-oriented and familiar"
2. It should be "robust and secure"
3. It should be "architecture-neutral and portable"
4. It should execute with "high performance"
5. It should be "interpreted, threaded, and dynamic"

One characteristic of Java is portability, which means that computer programs written in the Java language must run similarly on any hardware/operating-system platform. This is achieved by compiling the Java language code to an intermediate representation called Java bytecode, instead of directly to platform-specific machine code. Java bytecode instructions are analogous to machine code, but they are intended to be interpreted by a virtual machine (VM) written specifically for the host hardware. End-users commonly use a Java Runtime Environment (JRE) installed on their own machine for standalone Java applications, or in a Web browser for Java applets.

Standardized libraries provide a generic way to access host-specific features such as graphics, threading, and networking.

A major benefit of using bytecode is porting. However, the overhead of interpretation means that interpreted programs almost always run more slowly than programs compiled to native executables would. Just-in-Time (JIT) compilers were introduced from an early stage that compiles bytecodes to machine code during runtime.

Google and Android, Inc. have chosen to use Java as a key pillar in the creation of the Android operating system, an open-source smart-phone operating system. Besides the fact that the operating system, built on the Linux kernel, was written largely in C, the Android SDK uses Java to design applications for the Android platform.

4.2.5 C

In computing, C is a general-purpose programming language initially developed by Dennis Ritchie between 1969 and 1973 at AT&T Bell Labs. Like most imperative languages in the ALGOL tradition, C has facilities for structured programming and allows lexical variable scope and recursion, while a static type system prevents many unintended operations. Its design provides constructs that map efficiently to typical machine instructions, and therefore it has found lasting use in applications that had formerly been coded in assembly language, most notably system software like the Unix computer operating system.

C is one of the most widely used programming languages of all time, and C compilers are available for the majority of available computer architectures and operating systems.

Many later languages have borrowed directly or indirectly from C, including C#, D, Go, Java, JavaScript, Limbo, LPC, Perl, PHP, Python, and UNIX's C shell. The most pervasive influence on these languages (excluding Python) has been syntactical, and they tend to combine the recognizable expression and statement syntax of C with underlying type systems, data models, and semantics that can be radically different. C++ started as a preprocessor for C and is currently nearly a superset of C.

Before there was an official standard for C, many users and implementers relied on an informal specification contained in a book by Dennis Ritchie and Brian Kernighan; that version is generally referred to as "K&R" C. In 1989 the American National Standards Institute published a standard for C (generally called "ANSI C" or "C89"). The next year, the same specification was approved by the International Organization for Standardizations an international standard (generally called "C90"). ISO later released an extension to the internationalization support of the standard in 1995, and a revised standard (known as "C99") in 1999. The current version of the standard (now known as "C11") was approved in December 2011.

C has a formal grammar specified by the C standard. Unlike languages such as FORTRAN 77, C source code is free-form which allows arbitrary use of whitespace to format code, rather than column-based or text-line-based restrictions. Comments may appear either between the delimiters /* and */, or (since C99) following // until the end of the line. Comments delimited by /* and */ do not nest, and these sequences of characters are not interpreted as comment delimiters if they appear inside string or character literals.

C source files contain declarations and function definitions. Function definitions, in turn, contain declarations and statements. Declarations either define new types using keywords such as struct, union, and enum, or assign types to and perhaps reserve storage for new variables, usually by writing the type followed by the variable name. Keywords such as char and int specify built-in types. Sections of code are enclosed in braces ({ and }, sometimes called "curly brackets") to limit the scope of declarations and to act as a single statement for control structures.

As an imperative language, C uses *statements* to specify actions. The most common statement is an *expression statement*, consisting of an expression to be evaluated, followed by a semicolon; as a side effect of the evaluation, functions may be called and variables may be assigned new values. To modify the normal sequential execution of statements, C provides several control-flow statements identified by reserved keywords. Structured programming is supported by if (-else) conditional execution and by do-while, while, and for iterative execution (looping). The for statement has separate initialization, testing, and reinitialization expressions, any or all of which can be omitted. break and continue can be used to leave the innermost enclosing loop statement or skip to its reinitialization. There is also a non-structured goto statement which branches directly to the designated label within the function. switch selects a case to be executed based on the value of an integer expression.

Expressions can use a variety of built-in operators and may contain function calls. The order in which arguments to functions and operands to most operators are evaluated is unspecified. The evaluations may even be interleaved. However, all side effects (including storage to variables) will occur before the next "sequence point"; sequence points include the end of each expression statement, and the entry to and return from each function call. Sequence points also occur during evaluation of expressions containing certain operators (&&, ||, ?: and the comma operator). This permits a high degree of object code optimization by the compiler, but requires C programmers to take more care to obtain reliable results than is needed for other programming languages.

Kernighan and Ritchie say in the Introduction of *The C Programming Language*: "C, like any other language, has its blemishes. Some of the operators have the wrong precedence; some parts of the syntax could be better." The C standard did not attempt to correct many of these blemishes, because of the impact of such changes on already existing software.

The C programming language uses libraries as its primary method of extension. In C, a library is a set of functions contained within a single "archive" file. Each library typically has a header file, which contains the prototypes of the functions contained within the library that may be used by a program, and declarations of special data types and macro symbols used with these functions. In order for a program to use a library, it must include the library's header file, and the library must be linked with the program, which in many cases requires compiler flags (e.g., -lm, shorthand for "math library").

The most common C library is the C standard library, which is specified by the ISO and ANSI C standards and comes with every C implementation. (Implementations which target limited environments such as embedded systems may provide only a subset of the standard library.) This library supports stream input and output, memory allocation, mathematics, character strings, and time values. Several separate standard headers (for example, stdio.h) specify the interfaces for these and other standard library facilities.

Another common set of C library functions are those used by applications specifically targeted for Unix and Unix-like systems, especially functions which provide an interface to the kernel.

These functions are detailed in various standards such as POSIX and the Single UNIX Specification.

Since many programs have been written in C, there are a wide variety of other libraries available. Libraries are often written in C because C compilers generate efficient object code; programmers then create interfaces to the library so that the routines can be used from higher-level languages like Java, Perl, and Python.

Historically, embedded C programming requires nonstandard extensions to the C language in order to support exotic features such as fixed-point arithmetic, multiple distinct memory banks, and basic I/O operations.

In 2008, the C Standards Committee published a technical report extending the C language to address these issues by providing a common standard for all implementations to adhere to. It includes a number of features not available in normal C, such as fixed-point arithmetic, named address spaces, and basic I/O hardware addressing.

4.2.6 Python

Python is a general-purpose, high-level programming language whose design philosophy emphasizes code readability. Python's syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C, and the language provides constructs intended to enable clear programs on both a small and large scale.

Python supports multiple programming paradigms, including object-oriented, imperative and functional programming styles. It features a dynamic type system and automatic memory management and has a large and comprehensive standard library.

Like other dynamic languages, Python is often used as a scripting language, but is also used in a wide range of non-scripting contexts. Using third-party tools, Python code can be packaged into standalone executable programs. Python interpreters are available for many operating systems.

CPython, the reference implementation of Python, is free and open source software and has a community-based development model, as do nearly all of its alternative implementations. CPython is managed by the non-profit Python Software Foundation.

The core philosophy of the language is summarized by the document "PEP 20 (The Zen of Python)", which includes aphorisms such as:

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Readability counts.

Python has a large standard library, commonly cited as one of Python's greatest strengths, providing tools suited to many tasks. This is deliberate and has been described as a "batteries included" Python philosophy. For Internet-facing applications, a large number of standard formats and protocols (such as MIME and HTTP) are supported. Modules for creating graphical user interfaces, connecting to relational databases, arithmetic with arbitrary precision decimals, manipulating regular expressions, and doing unit testing are also included. For software testing, the standard library provides the unittest and doctest modules.

Some parts of the standard library are covered by specifications (for example, the WSGI implementation wsgiref follows PEP 333), but the majority of the modules are not. They are specified by their code, internal documentation, and test suite (if supplied). However, because most of the standard library is cross-platform Python code, there are only a few modules that must be altered or completely rewritten by alternative implementations.

The standard library is not essential to run Python or embed Python within an application. Blender 2.49 for instance omits most of the standard library.

The Python Package Index, which is the official repository of third-party software for Python, contains over 25,000 "packages" covering a wide range of functionality, including:

- graphical user interface, web framework, multimedia, databases, networking and communications
- test frameworks, documentation tools, system administration

- scientific computing, text processing, image processing

An empirical study found that, for a programming problem involving string manipulation and search in a dictionary, scripting languages such as Python were more productive than conventional languages such as C and Java. Memory consumption was often "better than Java and not much worse than C or C++". Large organizations that make use of Python include Google, Yahoo!, CERN, NASA, ILM, and ITA. As of March 2013, Python ranks at position 8 in the TIOBE Programming Community Index.

4.3 Modules

4.3.1 JQuery

jQuery is a multi-browser JavaScript library designed to simplify the client-side scripting of HTML. It was released in January 2006 at BarCamp NYC by John Resig. It is currently developed by a team of developers led by Dave Methvin. Used by over 65% of the 10,000 most visited websites, jQuery is the most popular JavaScript library in use today.

jQuery is free, open source software, licensed under the MIT License. jQuery's syntax is designed to make it easier to navigate a document, select DOM elements, create animations, handle events, and develop Ajax applications. jQuery also provides capabilities for developers to create plug-ins on top of the JavaScript library. This enables developers to create abstractions for low-level interaction and animation, advanced effects and high-level, theme-able widgets. The modular approach to the jQuery library allows the creation of powerful dynamic web pages and web applications.

Microsoft and Nokia have announced plans to bundle jQuery on their platforms. Microsoft is adopting it initially within Visual Studio for use within Microsoft's ASP.NET AJAX framework and ASP.NET MVC Framework while Nokia has integrated it into their Web Run-Time widget development platform. jQuery has also been used in MediaWiki since version 1.16.

4.3.2 Pixastic

Pixastic is an experimental library which allows you to perform a variety of operations on images using just a bit of JavaScript. The effects supported out of the box include desaturation/greyscale, invert, flipping, brightness/contrast adjustment, hue/saturation, emboss, blur, and many more.

Pixastic works by utilizing the HTML5 Canvas element which provides access to raw pixel data, thereby opening up for more advanced image effects. This is where the "experimental" part comes into play. Canvas is only supported by some browsers and unfortunately Internet Explorer is not one of them. It is however well supported in both Firefox and Opera and support for Safari only just arrived with the recent Safari 4 (beta) release. Chrome currently works in the dev channel. A few of the effects have been simulated in IE using the age old proprietary filters. While these filters are much faster than their Canvas friends, they are few and limited. Hopefully we will one day have real Canvas on IE as well.

4.3.3 Crypto

CryptoJS is a growing collection of standard and secure cryptographic algorithms implemented in JavaScript using best practices and patterns. They are fast, and they have a consistent and simple interface.

4.3.4 OpenCV

OpenCV (*Open Source Computer Vision Library*) is a library of programming functions mainly aimed at real-time computer vision, developed by Intel, and now supported by Willow Garage and Itseez. It is free for use under the open source BSD license. The library is cross-platform. It focuses mainly on real-time image processing. If the library finds Intel's Integrated Performance Primitives on the system, it will use these proprietary optimized routines to accelerate itself.

OpenCV runs on Linux, Windows, Android, Maemo, FreeBSD, OpenBSD, iOS, BlackBerry 10 and OS X.

OpenCV is written in C++ and its primary interface is in C++, but it still retains a less comprehensive though extensive older C interface. There are now full interfaces in Python, Java and MATLAB/OCTAVE (as of version 2.5). The API for these interfaces can be found in the online documentation. Wrappers in other languages such as C#, Ch, Ruby have been developed to encourage adoption by a wider audience.

4.3.5 NanoHTTPD

NanoHTTPD is an open-source, small-footprint web server that is suitable for embedded applications, written in the Java 1.1 programming language. The source code consists of a single *.java* file. It can be used as a library component in developing other software (such as measurement, science, and database applications) or as a standalone ad-hoc style HTTP daemon for serving files.

Due to independence from Java features beyond JDK 1.1, NanoHTTPD is suited for embedded application development, and has been used to build, for example, Android software.

The original version, released in 2003, only included simple HTTP 1.0 features, but the software has been since forked and extended to support some more advanced techniques such as HTML5 video streaming or HTTP uploading through multipart extensions.

4.4 Features

4.4.1 Live Image Streaming

- Uploads live images from the Android camera to the user
- Images encoded with base64 for transfer over HTTP

4.4.2. Face Detection

- Detects a human face using Android camera
- Computer Vision using openCV
- Sets up Feature Detection' using 'Haar Cascade' and draws a rectangle in around the detected face
- Not very accurate, but decent and relatively fast

4.4.3. Line Following

- The robot moves over a line of high contrast relative to the background
- Uses openCV for Computer Vision
- Sets up a Color Blob Detector and tries to keep the blob in the center

- Done so by measuring position, length and breadth of contours of the detected colors' and then we find centre of contours:

  $X = X_{POS} + HEIGHT/2$

  $Y = Y_{POS} + WIDTH/2$

  We try to keep the center on the middle of the image.

4.4.4. Object Tracking

- Tracks a colored object in 3D space
- Uses algorithm similar to line following
- Tracks in Z axis by looking at contour size – if increases, object moving near; if decreases, object moving away

4.4.5. **AGV / RC Robot**

- In this the robot can be controlled via. the Android phone remotely using the Android’s built-in accelerometer.

4.4.6. **Voice Synthesis**

- The robot reads aloud any text provided remotely by the user
- This is implemented using the Android’s TTS server

4.4.7. **Voice Recognition**

- This allows a person to speak words into the Android phone which would be converted into text and displayed to the user
- It is implemented using cloud-based services through Android APIs

4.4.8. **Motion/Proximity/Magnetic Detection**

- Motion means any vibration to the phone and is detected using the built-in accelerometer
- Proximity is detected behind the camera using the proximity sensor
- Magnetic compass is used to detect change in angular orientation of the Android device or the introduction of electromagnetic fields in the vicinity

4.4.9. Google Maps based mapping

- Locates the position of the phone using Google Maps
- Uses Google Map JavaScript APIs and Android’s built-in GPS sensor

CHAPTER 5

TESTING

5.1 Software Testing

Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include, but are not limited to, the process of executing a program or application with the intent of finding software bugs (errors or other defects).

Software testing can be stated as the process of validating and verifying that a computer program/application/product:

- meets the requirements that guided its design and development,
- works as expected,
- can be implemented with the same characteristics,
- and satisfies the needs of stakeholders.

Software testing, depending on the testing method employed, can be implemented at any time in the development process. Traditionally most of the test effort occurs after the requirements have been defined and the coding process has been completed, but in the Agile approaches most of the test effort is on-going. As such, the methodology of the test is governed by the chosen software development methodology.

Different software development models will focus the test effort at different points in the development process. Newer development models, such as Agile, often employ test-driven development and place an increased portion of the testing in the hands of the developer, before it reaches a formal team of testers. In a more traditional model, most of the test execution occurs after the requirements have been defined and the coding process has been completed.

There are many approaches to software testing. Reviews, walkthroughs, or inspections are referred to as static testing, whereas actually executing programmed code with a given set of test cases is referred to as dynamic testing. Static testing can be omitted, and in practice often is.

Dynamic testing takes place when the program itself is used. Dynamic testing may begin before the program is 100% complete in order to test particular sections of code and are applied to discrete functions or modules. Typical techniques for this are either using stubs/drivers or execution from a debugger environment.

Static testing involves verification whereas dynamic testing involves validation. Together they help improve software quality.

Software testing methods are traditionally divided into white- and black-box testing. These two approaches are used to describe the point of view that a test engineer takes when designing test cases.

White-box testing (also known as clear box testing, glass box testing, transparent box testing and structural testing) tests internal structures or workings of a program, as opposed to the functionality exposed to the end-user. In white-box testing an internal perspective of the system, as well as programming skills, are used to design test cases. The tester chooses inputs to exercise paths through the code and determine the appropriate outputs. This is analogous to testing nodes in a circuit, e.g. in-circuit testing (ICT).

While white-box testing can be applied at the unit, integration and system levels of **the** software testing process, it is usually done at the unit level. It can test paths within a unit, paths between units during integration, and between subsystems during a system–level test. Though this method of test design can uncover many errors or problems, it might not detect unimplemented parts of the specification or missing requirements.

Techniques used in white-box testing include:

- API testing– testing of the application using public and private APIs
- Code coverage – creating tests to satisfy some criteria of code coverage (e.g., the test designer can create tests to cause all statements in the program to be executed at least once)
- Fault injection methods – intentionally introducing faults to gauge the efficacy of testing strategies
- Mutation testing methods
- Static testing methods

Code coverage tools can evaluate the completeness of a test suite that was created with any method, including black-box testing. This allows the software team to examine parts of a system that are rarely tested and ensures that the most important function points have been tested. Code coverage as a software metric can be reported as a percentage for:

- *Function coverage*, which reports on functions executed
- *Statement coverage*, which reports on the number of lines executed to complete the test

100% statement coverage ensures that all code paths, or branches (in terms of control flow) are executed at least once. This is helpful in ensuring correct functionality, but not sufficient since the same code may process different inputs correctly or incorrectly.

Black-box testing treats the software as a "black box", examining functionality without any knowledge of internal implementation. The tester is only aware of what the software is supposed to do, not how it does it. Black-box testing methods include: equivalence partitioning, boundary value analysis, all-pairs testing, state transition tables, decision table testing, fuzz testing, model-based testing, use case testing, exploratory testing and specification-based testing.

5.2 Levels of Testing

Tests are frequently grouped by where they are added in the software development process, or by the level of specificity of the test. The main levels during the development process as defined by the SWEBOK guide are unit-, integration-, and system testing that are distinguished by the test target without implying a specific process model. Other test levels are classified by the testing objective.

5.2.1 Unit testing

Unit testing, also known as component testing, refers to tests that verify the functionality of a specific section of code, usually at the function level. In an object-oriented environment, this is usually at the class level, and the minimal unit tests include the constructors and destructors.

These types of tests are usually written by developers as they work on code (white-box style), to ensure that the specific function is working as expected. One function might have multiple tests, to catch corner cases or other branches in the code. Unit testing alone cannot verify the functionality of a piece of software, but rather is used to assure that the building blocks the software uses work independently of each other.

Unit testing is a software development process that involves synchronized application of a broad spectrum of defect prevention and detection strategies in order to reduce software development risks, time, and costs. It is performed by the software developer or engineer during the construction phase of the software development lifecycle. Rather than replace traditional QA focuses, it augments it. Unit testing aims to eliminate construction errors before code is promoted to QA; this strategy is intended to increase the quality of the resulting software as well as the efficiency of the overall development and QA process.

Depending on the organization's expectations for software development, unit testing might include static code analysis, data flow analysis metrics analysis, peer code reviews, code coverage analysis and other software verification practices.

5.2.2 Integration testing

Integration testing is any type of software testing that seeks to verify the interfaces between components against a software design. Software components may be integrated in an iterative way or all together ("big bang"). Normally the former is considered a better practice since it allows interface issues to be localized more quickly and fixed.

Integration testing works to expose defects in the interfaces and interaction between integrated components (modules). Progressively larger groups of tested software components corresponding to elements of the architectural design are integrated and tested until the software works as a system.

5.2.3 System testing

System testing tests a completely integrated system to verify that it meets its requirements.

In addition, the software testing should ensure that the program, as well as working as expected, does not also destroy or partially corrupt its operating environment or cause other processes within that environment to become inoperative (this includes not corrupting shared memory, not consuming or locking up excessive resources and leaving any parallel processes unharmed by its presence).

5.2.4 Acceptance testing

At last the system is delivered to the user for Acceptance testing.

5.3 Types of Testing

5.3.1 Installation testing

An installation test assures that the system is installed correctly and working at actual customer's hardware.

5.3.2 Compatibility testing

A common cause of software failure (real or perceived) is a lack of its compatibility with other application software, operating systems (or operating system versions, old or new), or target environments that differ greatly from the original (such as a terminal or GUI application intended to be run on the desktop now being required to become a web application, which must render in a web browser). For example, in the case of a lack of backward compatibility, this can occur because the programmers develop and test software only on the latest version of the target environment, which not all users may be running. This results in the unintended consequence that the latest work may not function on earlier versions of the target environment, or on older hardware that earlier versions of the target environment was capable of using. Sometimes such issues can be fixed by proactively abstracting operating system functionality into a separate program module or library.

5.3.3 Smoke and sanity testing

Sanity testing determines whether it is reasonable to proceed with further testing.

Smoke testing is used to determine whether there are serious problems with a piece of software, for example as a build verification test.

5.3.4 Regression testing

Regression testing focuses on finding defects after a major code change has occurred. Specifically, it seeks to uncover software regressions, or old bugs that have come back. Such regressions occur whenever software functionality that was previously working correctly stops working as intended. Typically, regressions occur as an unintended consequence of program changes, when the newly developed part of the software collides with the previously existing code. Common methods of regression testing include re-running previously run tests and checking whether previously fixed faults have re-emerged. The depth of testing depends on the phase in the release process and the risk of the added features. They can either be complete, for changes added late in the release or deemed to be risky, to very shallow, consisting of positive tests on each feature, if the changes are early in the release or deemed to be of low risk.

5.3.5 Acceptance testing

Acceptance testing can mean one of two things:

- A smoke test is used as an acceptance test prior to introducing a new build to the main testing process, i.e. before integration or regression.
- Acceptance testing performed by the customer, often in their lab environment on their own hardware, is known as user acceptance testing (UAT). Acceptance testing may be performed as part of the hand-off process between any two phases of development.

5.3.6 Alpha testing

Alpha testing is simulated or actual operational testing by potential users/customers or an independent test team at the developers' site. Alpha testing is often employed for off-the-shelf software as a form of internal acceptance testing, before the software goes to beta testing.

5.3.7 Beta testing

Beta testing comes after alpha testing and can be considered a form of external user acceptance testing. Versions of the software, known as beta versions, are released to a limited audience outside of the programming team. The software is released to groups of people so that further testing can ensure the product has few faults or bugs. Sometimes, beta versions are made available to the open public to increase the feedback field to a maximal number of future users.

5.3.8 Functional vs. non-functional testing

Functional testing refers to activities that verify a specific action or function of the code. These are usually found in the code requirements documentation, although some development methodologies work from use cases or user stories. Functional tests tend to answer the question of "can the user do this" or "does this particular feature work."

Non-functional testing refers to aspects of the software that may not be related to a specific function or user action, such as scalability or other performance, behavior under certain constraints, or security. Testing will determine the flake point, the point at which extremes of scalability or performance leads to unstable execution. Non-functional requirements tend to be those that reflect the quality of the product, particularly in the context of the suitability perspective of its users.

5.3.9 Destructive testing

Destructive testing attempts to cause the software or a sub-system to fail. It verifies that the software functions properly even when it receives invalid or unexpected inputs, thereby establishing the robustness of input validation and error-management routines. Software fault injection, in the form of fuzzing, is an example of failure testing. Various commercial non-functional testing tools are linked from the software fault injection page; there are also numerous open-source and free software tools available that perform destructive testing.

5.3.10 Software performance testing

Performance testing is generally executed to determine how a system or sub-system performs in terms of responsiveness and stability under a particular workload. It can also serve to investigate, measure, validate or verify other quality attributes of the system, such as scalability, reliability and resource usage.

Load testing is primarily concerned with testing that the system can continue to operate under a specific load, whether that be large quantities of data or a large number of users. This is generally referred to as software scalability. The related load testing activity of when performed as a non-functional activity is often referred to as *endurance testing*. *Volume testing* is a way to test software functions even when certain components (for example a file or database) increase radically in size. *Stress testing* is a way to test reliability under unexpected or rare workloads. *Stability testing* (often referred to as load or endurance testing) checks to see if the software can continuously function well in or above an acceptable period.

There is little agreement on what the specific goals of performance testing are. The terms load testing, performance testing, reliability testing, and volume testing, are often used interchangeably.

5.3.11 Usability testing

Usability testing is needed to check if the user interface is easy to use and understand. It is concerned mainly with the use of the application.

5.3.12 Security testing

Security testing is essential for software that processes confidential data to prevent system intrusion by hackers.

5.3.13 Development testing

Development Testing is a software development process that involves synchronized application of a broad spectrum of defect prevention and detection strategies in order to reduce software development risks, time, and costs. It is performed by the software developer or engineer during the construction phase of the software development lifecycle. Rather than replace traditional QA focuses, it augments it. Development testing aims to eliminate construction errors before code is promoted to QA; this strategy is intended to increase the quality of the resulting software as well as the efficiency of the overall development and QA process.

Depending on the organization's expectations for software development, Development Testing might include static code analysis, data flow analysis metrics analysis, peer code reviews, unit testing, code coverage analysis, traceability, and other software verification practices.

5.4 Test Cases

5.4.1 Unit Testing

Unit Testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although is uncommon for coding and unit testing to be conducted as two distinct phases.

5.4.2 Test Strategy and Approach

Field testing will be performed by the user manually and functional tests have been done.

5.4.3 Test Objectives

The objectives behind testing our software are enumerated below:

- All connections must work properly
- All features must run as predicted
- The software can run in a variety of devices
- The HRI mechanisms must work properly

5.4.4 Integration Testing

Software Integration Testing is the incremental integration testing of two or more integrated software components on a single platform to produce failure caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up –software applications at the company level – interact without error.

5.4.5 Acceptance Testing

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

5.4.6 Test Results

Initial testing resulted in the discovery of many bugs, which were eventually eradicated. In the final test, all the test cases mentioned above passed successfully. No defects were encountered.

CHAPTER 6

EXAMPLE IMPLEMENTATIONS OF THE ROBOT

6.1 The Robot Algorithm

The Android software doesn’t expect any feedback from the robot, but sequentially provides output through HTTP (as JSON) or Bluetooth (as a string).

The robot may be of any type, but we expect it to be a two (or more) motors based differential drive. A differential wheeled robot is a mobile robot whose movement is based on two separately driven wheels placed on either side of the robot body. It can thus change its direction by varying the relative rate of rotation of its wheels and hence does not require an additional steering motion. To balance the robot, additional wheels or casters may be added.

We have implemented the robot in two phases. First, we created a software simulator that comes embedded inside RIA. Then we implemented the specifications and algorithm on the actual hardware based robot. An example implementation of the aforementioned specifications and algorithm in ANSI C is given below. It is to be noted that the code below, though executable, is not for actual usage.

6.2 The Software Simulator

The RIA also has a software simulator built into it. The simulator shows the required configuration of the DC motors for the robot. It also graphically shows the direction in which the robot is supposed to move.

The simulator is made with standard W3C compatible web technologies, namely HTML5, CSS3 and JavaScript. The two-dimensional graphical view is made using HTML5 Canvas and programmed using JavaScript.

6.3 The Robot Hardware

The Scorpius Rover is an example implementation of a hardware robot that uses the AndRo.bot application to control itself. It uses the specification of the Bluetooth output of the software to implement its internal algorithms. The detail about the Embedded C implementation of the robot (using PWM) is out of the scope of this project.

Scorpius uses the 89c51 microcontroller, HC51 Bluetooth module and L293 motor driver to provide a two DC motors differential drive based rover.

The AT89C51 is a CMOS 8-bit microcomputer with 4K bytes of Flash programmable and erasable read only memory (PEROM). The on-chip Flash allows the program memory to be reprogrammed in-system or by a ordinary nonvolatile memory programmer. Atmel AT89C51 is a powerful microcomputer/microcontroller (as they are used inter-changeably) which provides a highly-flexible and cost-effective solution to many embedded control applications. Features of AT89C51 are enumerated below:

- 128 bytes of RAM for storing running program,
- 32 I/O lines for communicating with other devices,
- two 16-bit timer/counters,
- a five vector two-level interrupt architecture,
- a full duplex serial port, on-chip oscillator and clock circuitry.

HC-05 is a class-2 Bluetooth module with Serial Port Profile, which can configure as either Master or slave. We can also use it as a drop-in replacement for wired serial connections, transparent usage. You can use it simply for a serial port replacement to establish connection between MCU, PC to your embedded project and etc. The HC-05 specifications are enumerated below:

- Bluetooth Protocol: Bluetooth Specification v2.0+EDR
- Frequency: 2.4GHz ISM band
- Modulation: GFSK(Gaussian Frequency Shift Keying)
- Emission power: ≤4dBm, Class 2
- Sensitivity: ≤-84dBm at 0.1% BER
- Speed: Asynchronous: 2.1Mbps(Max) / 160 kbps, Synchronous: 1Mbps/1Mbps
- Security: Authentication and encryption
- Profiles: Bluetooth serial port

- Power supply: +3.3VDC 50mA
- Working temperature: -20 ~ +75Centigrade
- Dimension: 26.9mm x 13mm x 2.2 mm

The L293 is an integrated circuit motor driver that can be used for simultaneous, bidirectional control of two small motors. The L293 is limited to 600 mA, but in reality can only handle much small currents unless you have done some serious heat sinking to keep the case temperature down. Unsure about whether the L293 will work with your motor? Hook up the circuit and run your motor while keeping your finger on the chip. If it gets too hot to touch, you can't use it with your motor. The L293 comes in a standard 16-pin, dual-in line integrated circuit package.

CHAPTER 6
CONCLUSION

6.1 Limitations

The software needs high bandwidth connection between user and robot, else performance degrades. There are no easy ways out, except increasing bandwidth.

Sensor Processing and Computer Vision are processor intensive. This can be improved using faster, multi-core processors. Many high-end smart-phones have quad-core processors; we can expect mid-range phone to have them soon courtesy of Moore’s Law. We can use GPU for even faster processing. We can even run C code directly using JNI, since C is about 10x faster than Java (even with JIT).

Static rules cannot perform in all (esp. unforeseen) situations. This defect can be corrected by removing static rules and replacing them with soft computing techniques (ANNs, neuro-fuzzy, GAs, EAs) for higher automation.

6.2 Prospective Usages

Being general-purpose, we can use our software for a wide variety of tasks. But some of the most common tasks that can be easily performed are:

- With extra robotic hardware
    - Military
    - Entertainment
    - Civilian
- Without hardware
    - Photography
    - Security (as an intercom)
    - Research

The usage can be improved by implementing any algorithm that the situation may demand. This is further possible because of the ease with which newer algorithms can be implemented.

6.3 Future Enhancements

The software can be enhanced using a variety of techniques, but the most prominent would be the application of soft-computing techniques.

Soft computing is a term applied to a field within computer science which is characterized by the use of inexact solutions to computationally hard tasks such as the solution of NP-complete problems, for which there is no known algorithm that can compute an exact solution in polynomial time. Soft computing differs from conventional (hard) computing in that, unlike hard computing, it is tolerant of imprecision, uncertainty, partial truth, and approximation. In effect, the role model for soft computing is the human mind.

The application of soft-computing would allow more seamless changing of modes, as required by the situation. Also, using heuristics would improve performance massively compared to brute-force techniques.

Soft Computing Techniques can easily be implemented in OpenCV using the built-in Machine Learning module.

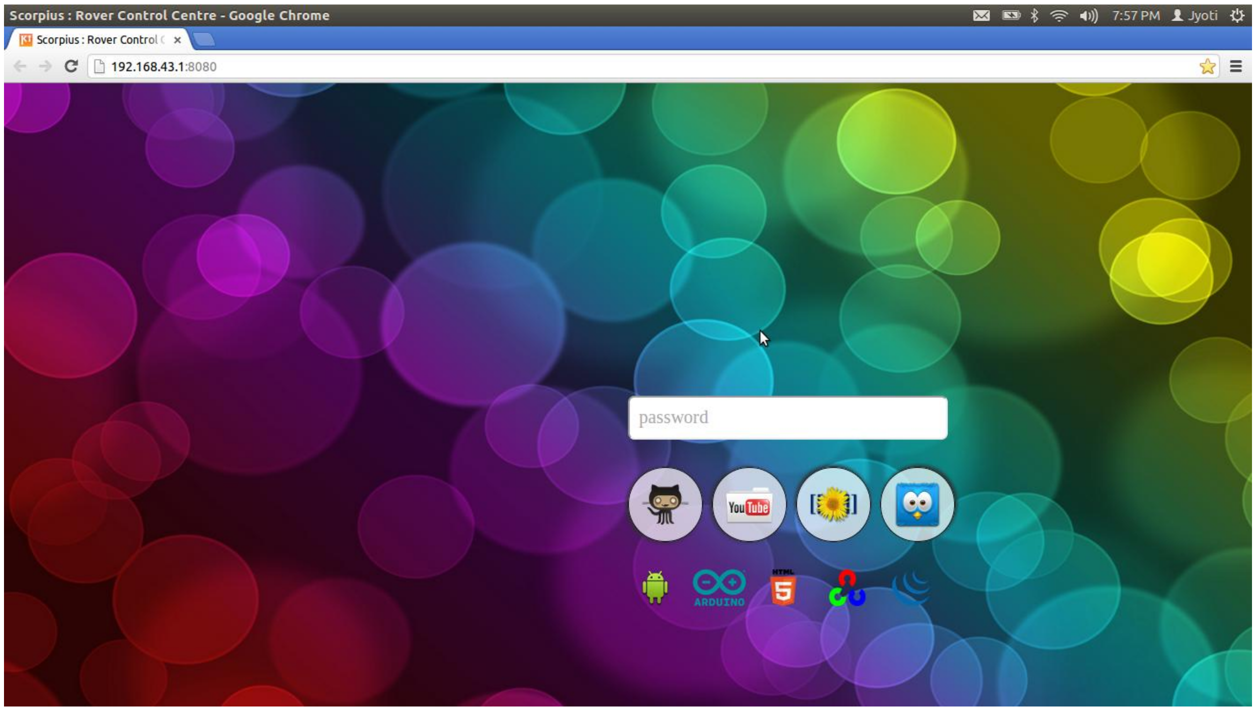
Another enhancement would be to create a full-duplex link between the software and the peripheral robotic device. This would allow the peripheral robot to provide feed-back to the device. This would result in more accuracy in the algorithms.

Finally, we can build a custom ROM by forking a version of Android and including OpenCV Manager and our software by default. This would result in a modified Android OS optimized for robotics.

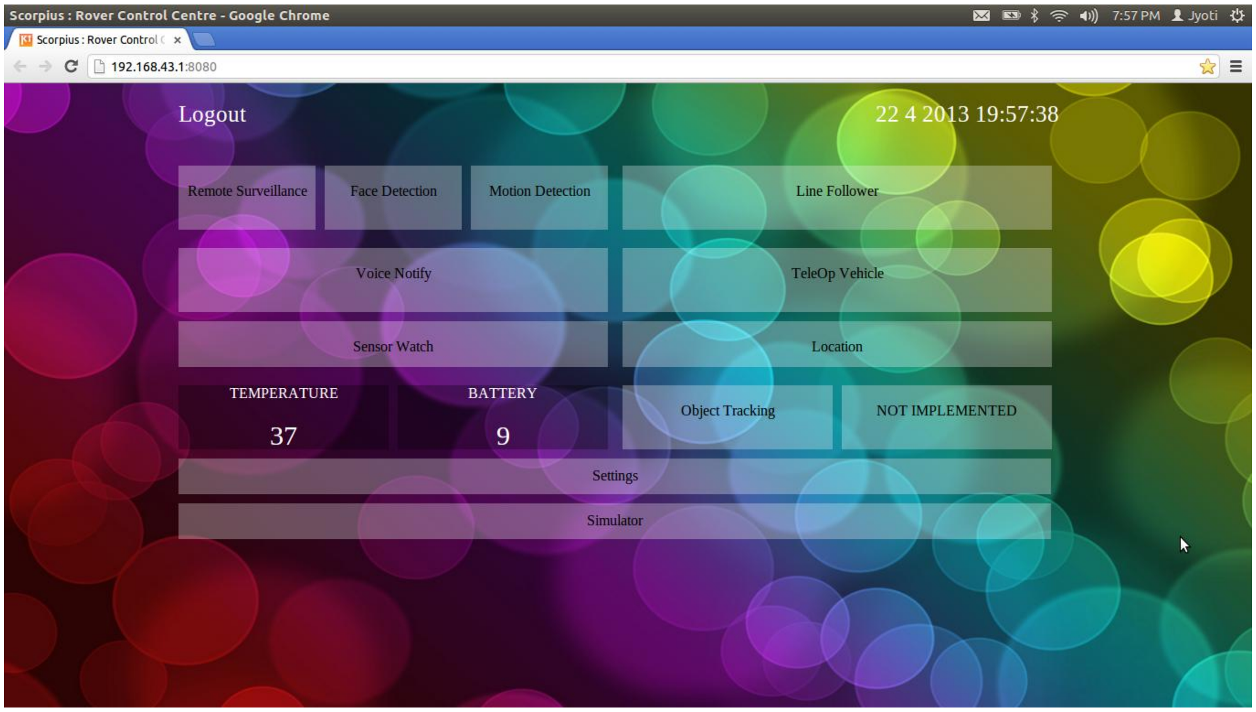
BIBLIOGRAPHY

- Green, S.A., Billinghurst, M., Chen, X., Chase, G.J. (2008, *Human-Robot Collaboration: A Literature Review and Augmented Reality Approach in Design*. International Journal of Advanced Robotic Systems, 5(1), pp. 1-18
- *Machine Vision Giving Eyes to Robots: Resources in Technology*. (March 1990, Technology Teacher, 49: 6, 21-28.
- Braggins, Don. (1995, *'A critical look at robot vision'*. The Industrial Robot, 22(6): 9-12.)
- Grimson, W.E.L. and J.L. Mundy. (March 1994, *'Computer Vision applications'*. Communications of the ACM, 37(3): 44-51
- Ken Taylor and B. Dalton(1997, *'Issues in Internet Telerobotics'*. In International Conference on Field and Service Robotics (FSR 97)).
- K. Goldberg, K. Mascha, M. Genter, N. Rothenberg, C. Sutter, and J. Wiegley. (May 1995, *'Desktop teleoperation via the World Wide Web'*. In Proceedings of IEEE International Conference on Robotics and Automation
- Paul G. Backes, Kam S. Tao, and Gregory K. Tharp. (May 1998*, 'Mars pathfinder mission Internet-based operations using WITS'*. In Proceedings of IEEE International Conference on Robotics and Automation, pages 284-291
- *Learning openCV Computer Vision and the openCV library* - Gary Bradski and Adrian Kaehlar
- Robert Laganière, *OpenCV 2 - Computer Vision Application Programming Cookbook -*
- J. F. DiMarzio, *Android - A Programmer's Guide*
- Mark L. Murphy, *Beginning Android*
- Michael McRoberts, *Beginning Arduino*
- Gordon McComb, *The Robot Builder's Bonanza (2nd Edition)*
- http://developer.android.com
- http://www.arduino.cc
- http://www.opencv.org

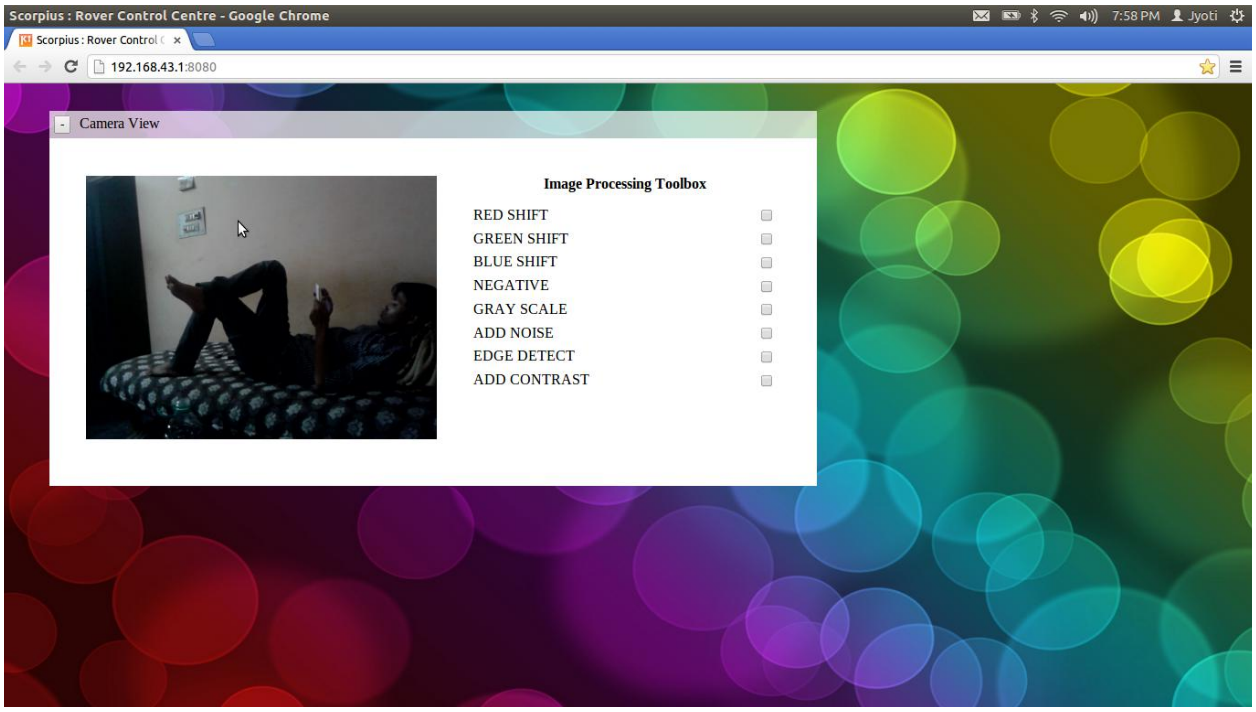
SCREENSHOTS



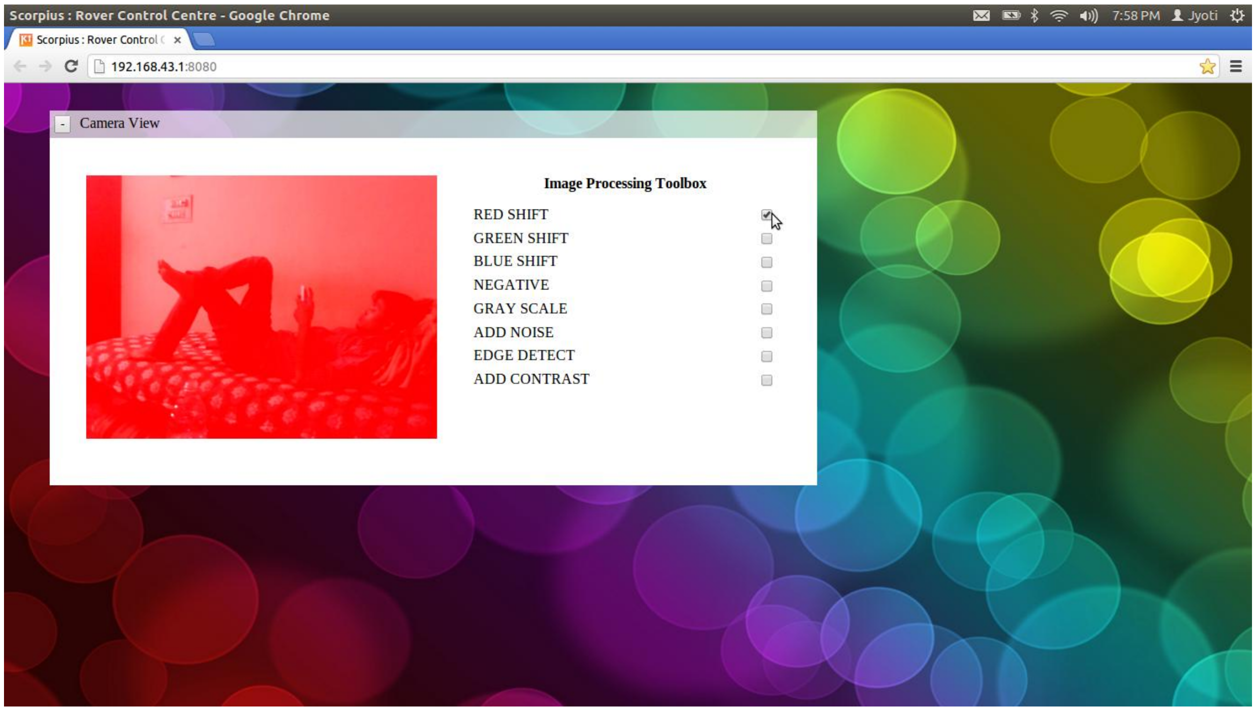
[Screenshot 1: Login Screen (RIA)]



[Screenshot 2: Control Panel (RIA)]



[Screenshot 3: Live Image Streaming with no Real Time Image Processing (RIA)]



[Screenshot 4: Live Image Streaming with Red Shift]

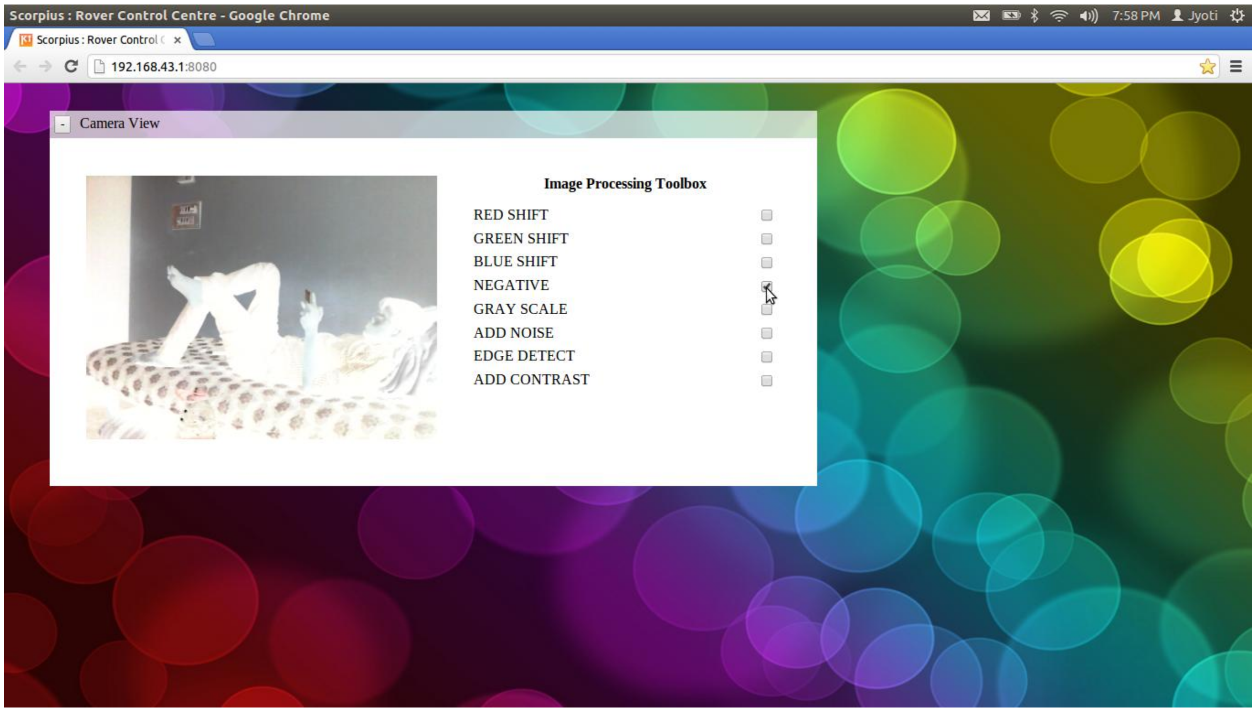

Scorpius : Rover Control Centre - Google Chrome
7:58 PM
Jyoti
Scorpius : Rover Control C
192.168.43.1:8080
Camera View

Image Processing Toolbox

RED SHIFT ☐
GREEN SHIFT ☐
BLUE SHIFT ☐
NEGATIVE ☑
GRAY SCALE ☐
ADD NOISE ☐
EDGE DETECT ☐
ADD CONTRAST ☐

[Screenshot 5: Live Image Streaming with Negative]

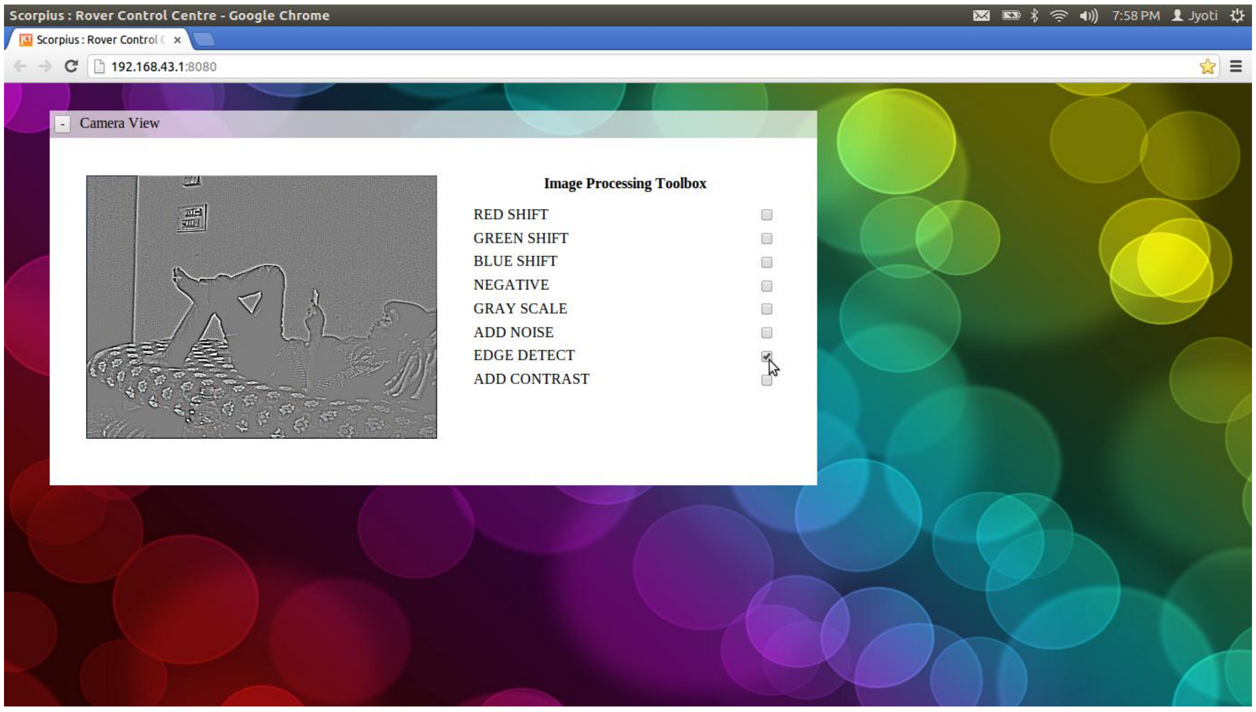

Scorpius : Rover Control Centre - Google Chrome
7:58 PM
Jyoti
Scorpius : Rover Control C
192.168.43.1:8080
Camera View

Image Processing Toolbox

RED SHIFT ☐
GREEN SHIFT ☐
BLUE SHIFT ☐
NEGATIVE ☐
GRAY SCALE ☐
ADD NOISE ☐
EDGE DETECT ☑
ADD CONTRAST ☐

[Screenshot 6: Live Image Streaming with Edge Detection]

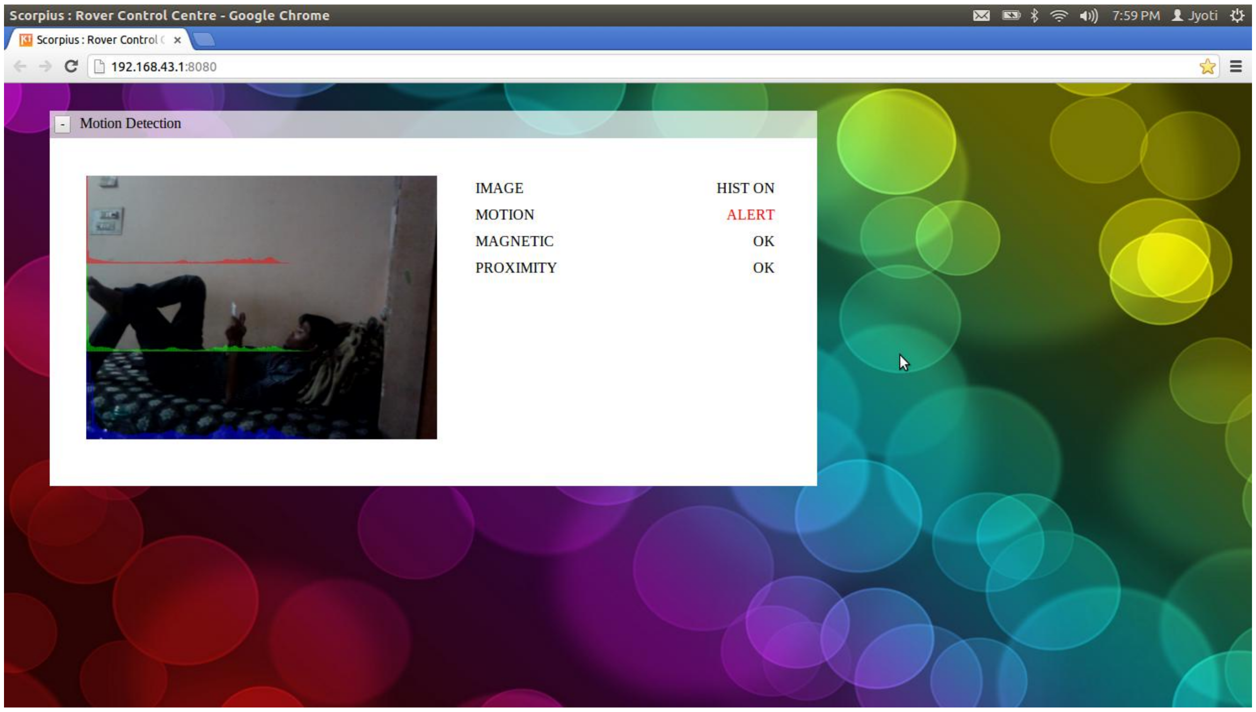

Scorpius : Rover Control Centre - Google Chrome
7:59 PM
Jyoti
Scorpius : Rover Control C
192.168.43.1:8080
Motion Detection

IMAGE	HIST ON
MOTION	ALERT
MAGNETIC	OK
PROXIMITY	OK

[Screenshot 7: Motion Detection with Histogram ON and Motion detected]

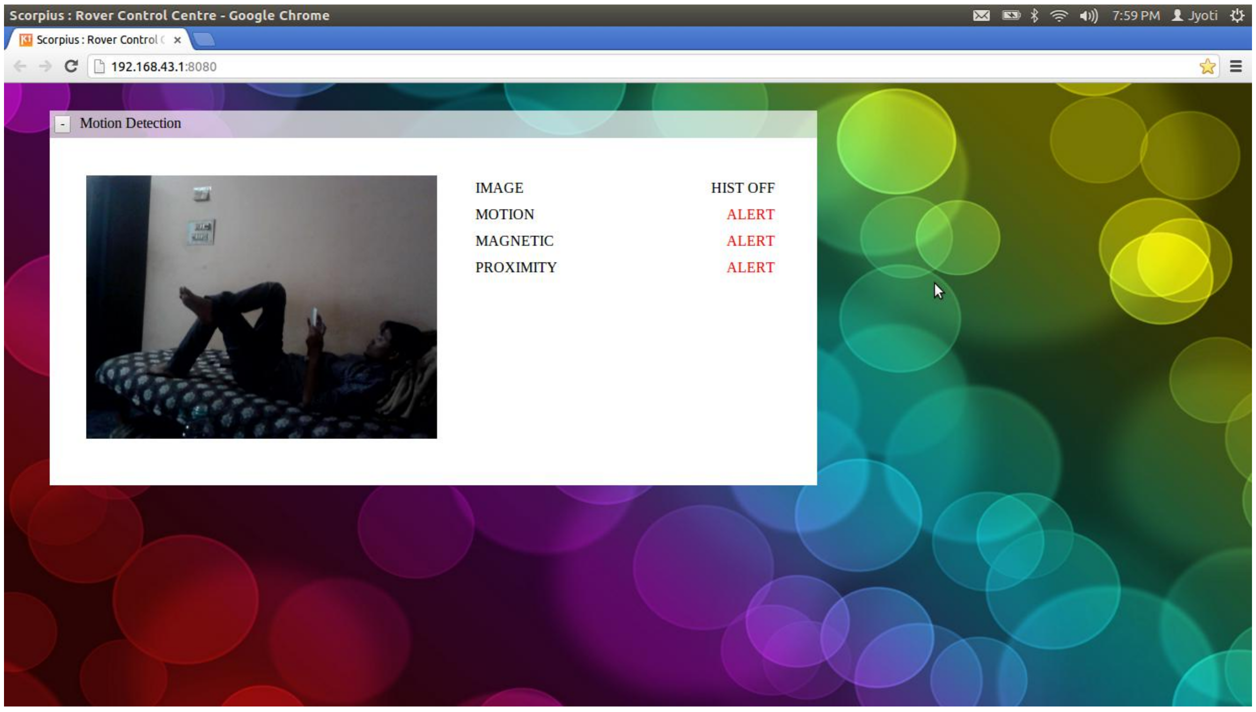
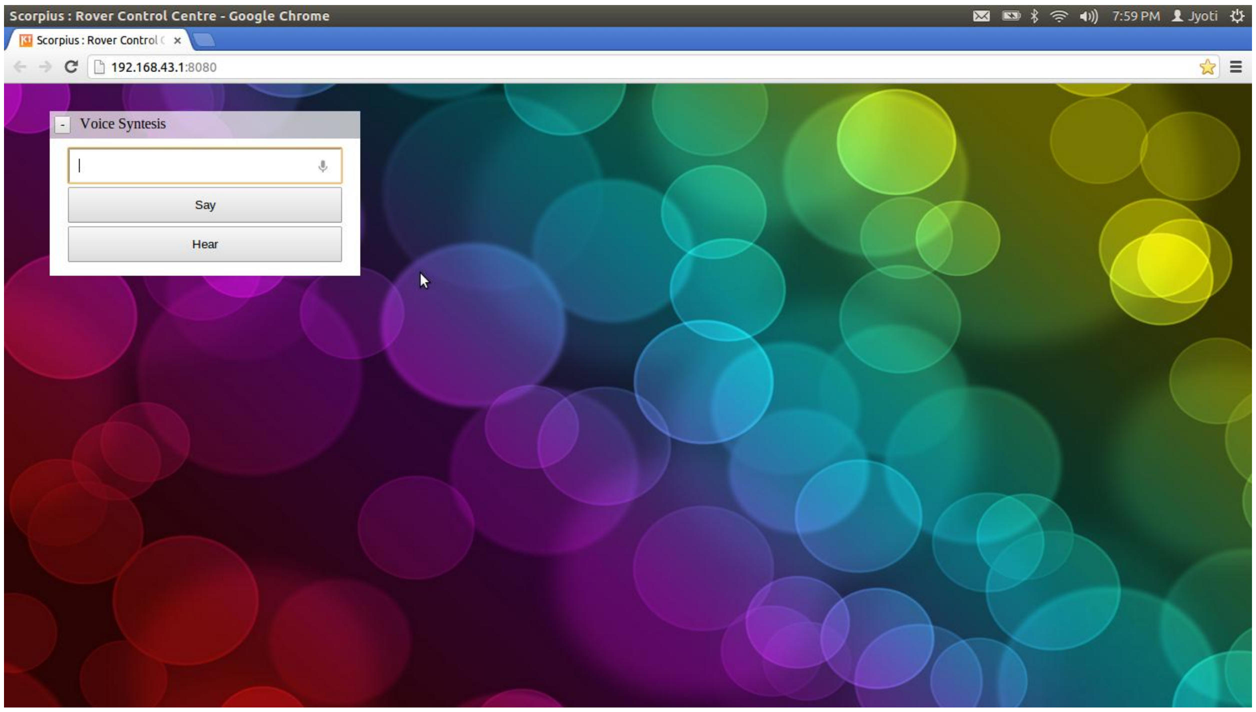
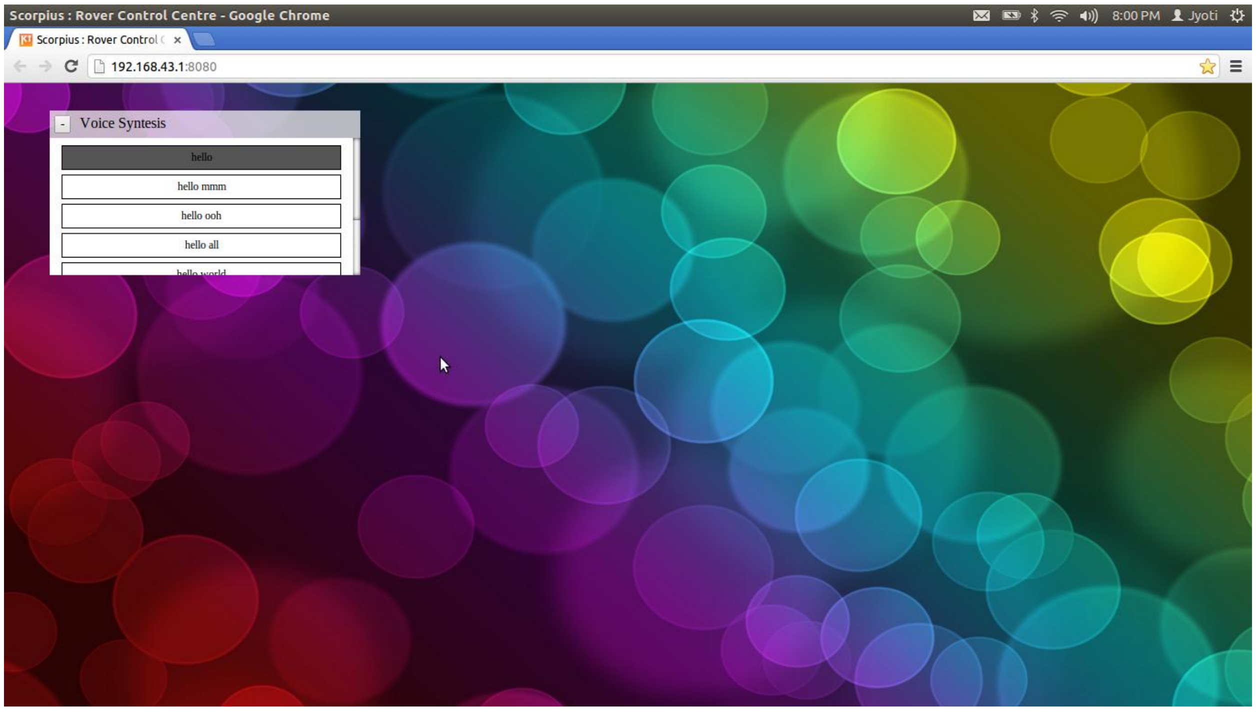

Scorpius : Rover Control Centre - Google Chrome
7:59 PM
Jyoti
Scorpius : Rover Control C
192.168.43.1:8080
Motion Detection

IMAGE	HIST OFF
MOTION	ALERT
MAGNETIC	ALERT
PROXIMITY	ALERT

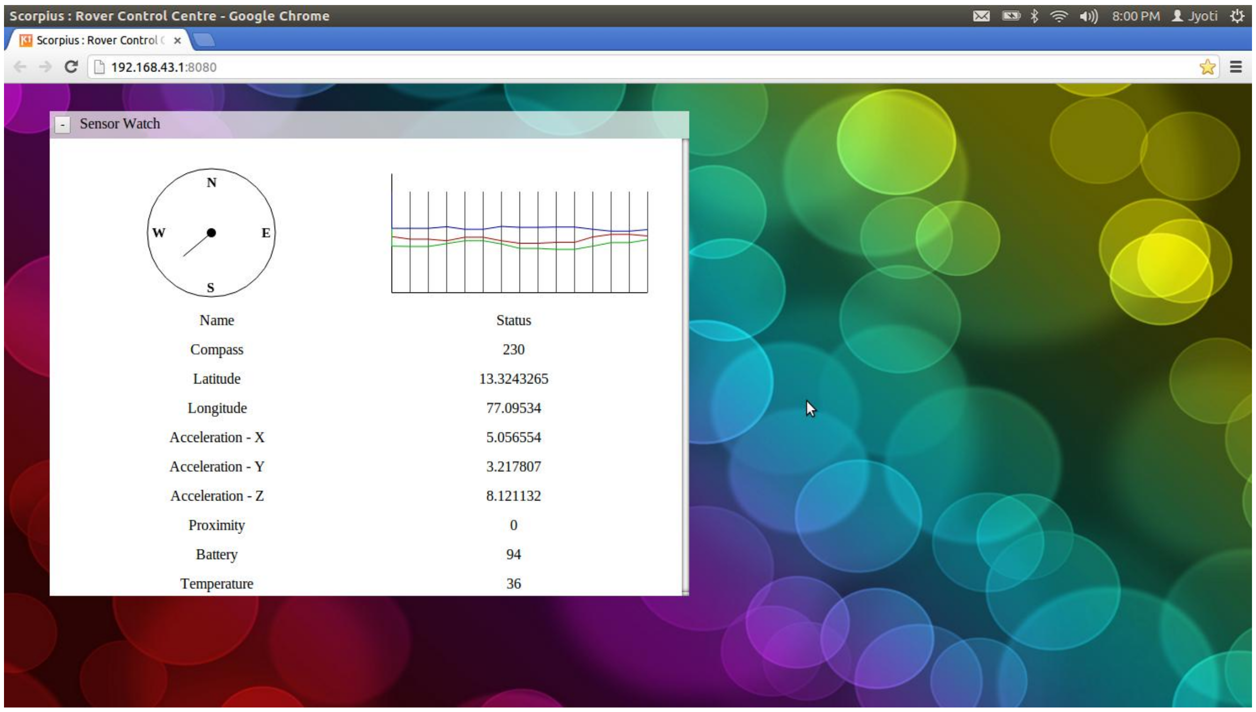
[Screenshot 8: Motion Detection with Histogram OFF and Motion, Compass and Proximity Detected]



[Screenshot 9: Voice Synthesis and Recognition]

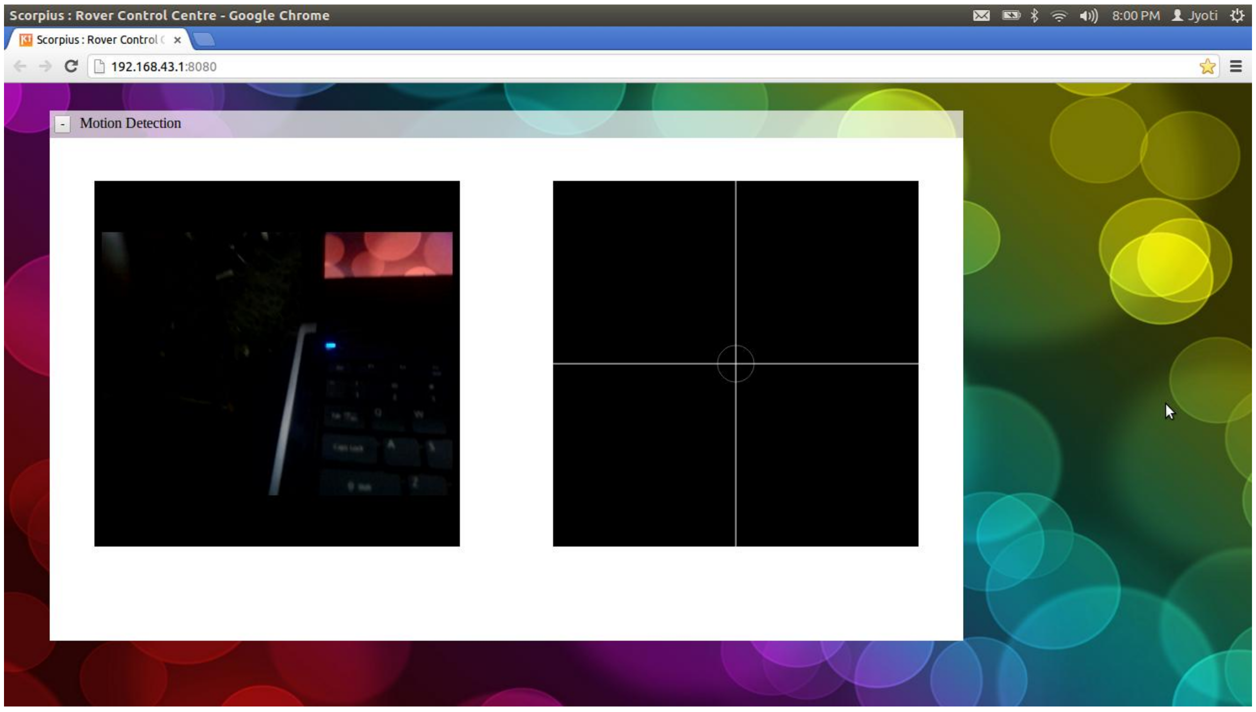


[Screenshot 10: Voice Recognition with text (‘Hello’) detected]

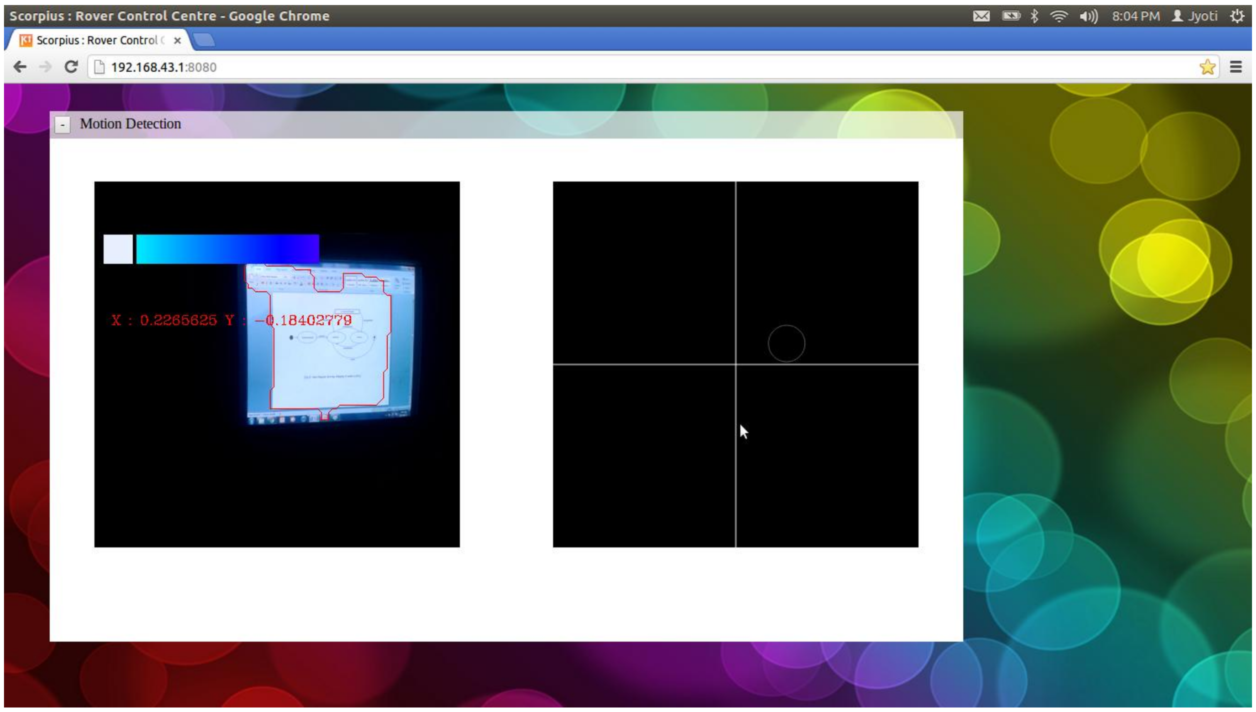

Scorpius : Rover Control Centre - Google Chrome
8:00 PM
Jyoti
Scorpius : Rover Control C
192.168.43.1:8080
Sensor Watch
N
W
E
S

Name	Status
Compass	230
Latitude	13.3243265
Longitude	77.09534
Acceleration - X	5.056554
Acceleration - Y	3.217807
Acceleration - Z	8.121132
Proximity	0
Battery	94
Temperature	36

[Screenshot 11: Sensor View]

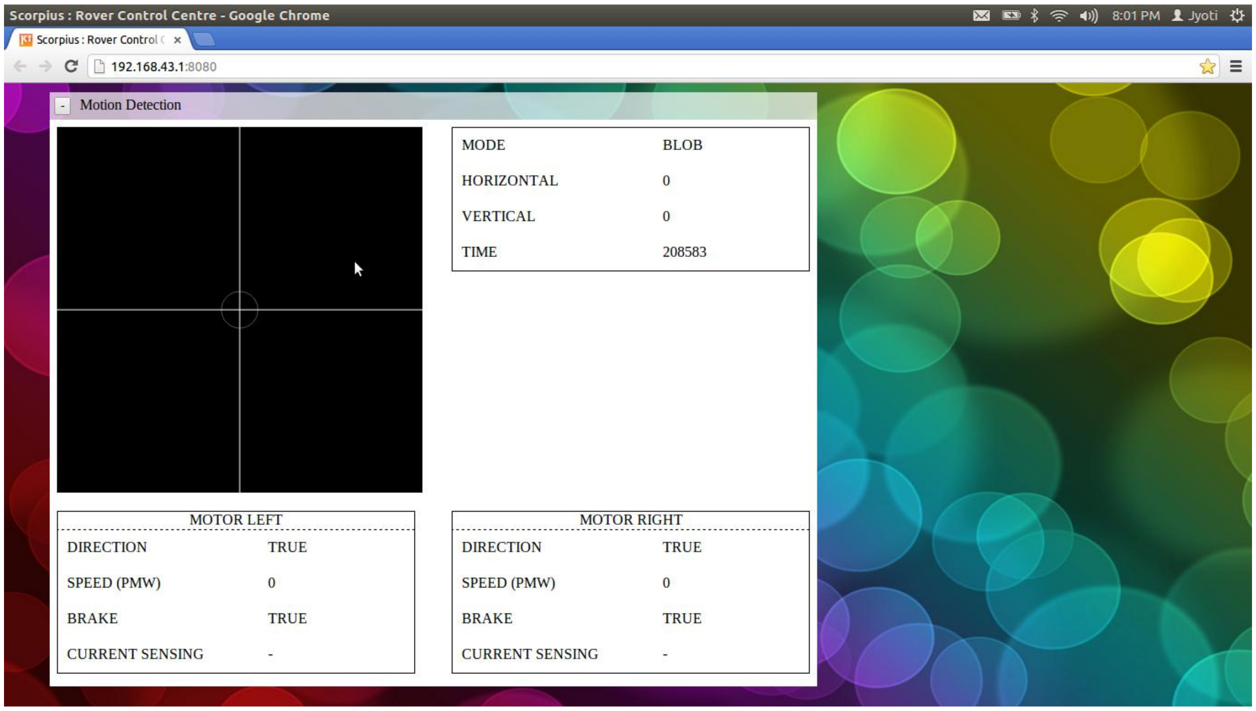

Scorpius : Rover Control Centre - Google Chrome
8:00 PM
Jyoti
Scorpius : Rover Control C
192.168.43.1:8080
Motion Detection

[Screenshot 12: Object Tracking while uninitialized]


Scorpius : Rover Control Centre - Google Chrome
8:04 PM
Jyoti
Scorpius : Rover Control C
192.168.43.1:8080
Motion Detection

```
X : 0.2265625 Y : -0.18402779
```

[Screenshot 13: Object Tracking while detecting object]

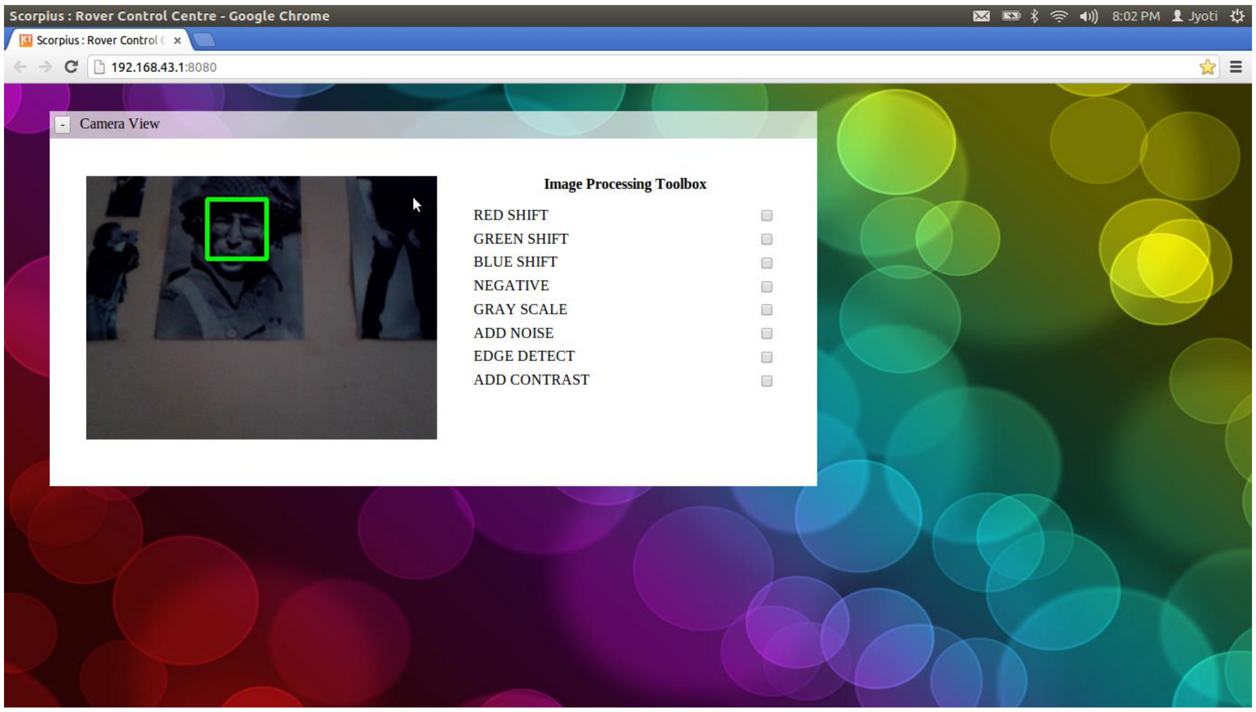

Scorpius : Rover Control Centre - Google Chrome
8:01 PM
Jyoti
Scorpius : Rover Control C
192.168.43.1:8080
Motion Detection

MODE	BLOB
HORIZONTAL	0
VERTICAL	0
TIME	208583

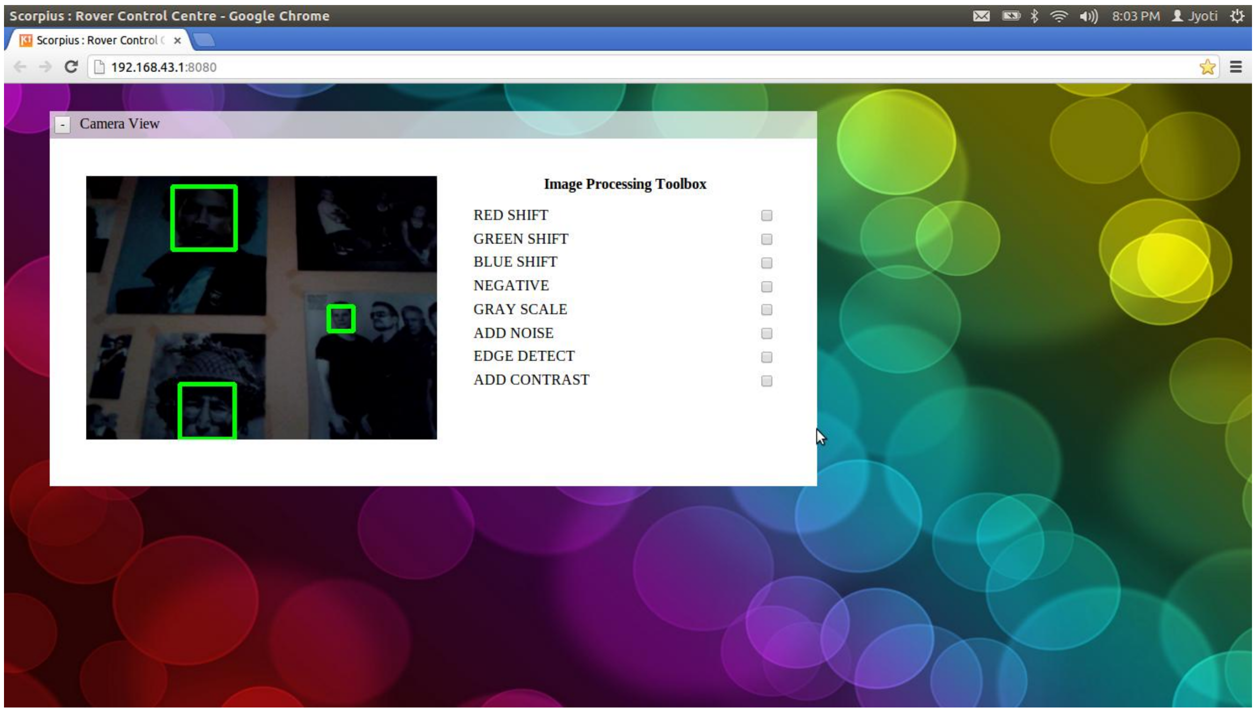
MOTOR LEFT	
DIRECTION	TRUE
SPEED (PMW)	0
BRAKE	TRUE
CURRENT SENSING	-

MOTOR RIGHT	
DIRECTION	TRUE
SPEED (PMW)	0
BRAKE	TRUE
CURRENT SENSING	-

[Screenshot 14: AGV/RC Robot View with Software Simulator]



[Screenshot 15: Face Detector with single face detected]



[Screenshot 16: Face Detector with multiple faces detected]