BUILDING VISION AND VOICE BASED ROBOTS USING ANDROID

Pradeep N. , Assistant Professor
Dr. Mohammed Sharief, Professor
Dr. M. Siddappa, Professor
Dept. of Computer Science and Engineering, SSIT, Tumkur, Karnataka, India

Abstract:

This project uses Android, along with cloud based APIs and the Arduino microcontroller, to create a software-hardware system that attempts to address several current problems in robotics, including, among other things, line-following, obstacle-avoidance, voice control, voice synthesis, remote surveillance, motion & obstacle detection, face detection, remote live image streaming and remote photography in panorama & time-lapse mode. The rover, built upon a portable five-layered architecture, will use several mature and a few experimental algorithms in AI, Computer Vision, Robot Kinetics and Dynamics behind-the-scene to provide its prospective user in defense, entertainment or industrial sector a positive user experience. A minimal amount of security is achieved via. a handshake using MD5, SHA1 and AES. It even has a Twitter account for autonomous social networking.

Key Words: Voice Synthesis, Voice Control, Robot-User Interaction, Face Detection, Motion and Object Detection, Remote Surveillance, Panoramic and Time-lapse Photography, Line-following and Obstacle-Avoiding Robotics, AI, Computer Vision, Image Processing, Network Security

Introduction

Android, which is actually a software stack, running atop the Linux kernel, is one of the most successful operating systems available to mobile users today. This success can be attributed to the fact that Android makes development of application softwares ('apps') really easy by allowing developers to code in Java (among many other languages) and thus enabling them to take advantage of the innumerable packages and modules made available by the Java community. Also the *android.** package provides several APIs to access features that are traditionally unique to mobile platforms (if not only Android). Examples of such APIs are the ones that we can use to access the readings of the innumerable sensors that an Android smart-phone may have, like proximity meter, compass, GPS, accelerometer and, most importantly, the camera. The presence of these sensors often make Android a more lucrative computing platform compared to 'traditional' platforms like PCs. Like any other Linux distribution, Android supports the entire TCP/IP stack, right from working with raw sockets to making application-layer level HTTP requests. Android can connect to the Internet using any of Wi-Fi, 3G, GSM or EDGE. Moreover, it can usually make telephone calls and send and receive SMSes (which are attractive features, but almost always absent in PCs). It is because of Moore's Law that an Android phone of today is much more powerful than a desktop PC from last decade. It is normal for an Android phone to ship in with more than a 1 GHz processor and 512 MB RAM.

These facts have recently increased people's curiosity on using Android for uses, which till now were rarely associated with mobile phones. Devices like smart TVs and dashboards of automobiles are being run on Android. NASA is planning to launch three nano-satellites (named Alexander, Graham and Bell) running on Android smart-phones under a project named PhoneSat. We simply wanted to explore the possibility of using Android to control a robot.

Description Of The Robot

From the perspective of the Android phone, the robot is yet another application running on it. The application acts as a broker between the user and the robot hardware.

The communication between the user and the robot takes place through HTTP or SMS. The robot (comprising of the Android phone and other hardware) itself may connect to the Internet through Wi-Fi or 3G/GSM/EDGE. The user interacts with the robot using a RIA (Rich Internet Application) that relies heavily on HTML5, CSS3 and JQuery. Security, being one of the major quality attribute, is achieved using a basic technique. The robot authenticates the user by matching MD5 and SHA1 hash of passwords. This prevents the transmission of the actual password string across the network. Once authenticated, all further communication between the user and the robot are encrypted using AES with the password as the key.

The communication layer interacts with the user to receive new commands. When a new command arrives, it notifies the Controller Layer. The Controller Layer is responsible for stopping the current task and starting the new task as required by the user's commands.

The tasks that the rover can perform are basically Java classes. Some classes may contain methods to perform more than one tasks. These tasks enable some, but not all, features of the robot. The features, that are not implemented directly using Java in the Android application, are implemented using JavaScript in the RIA. Examples of such features are 'Voice Recognition' and 'Motion and Object Detection'.

Because the Communication Layer, the Controller Layer and the individual features are all separated from one another, we were able to pack in a whole lot of features into our robot.

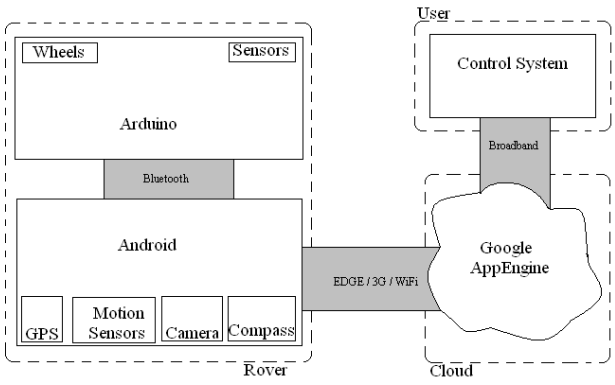
The features that the robot has are enumerated below:

1.Line Following
2.Obstacle Avoidance
3.Voice Synthesis
4.Voice Control
5.Robot-User Interaction
6.Face Detection
7.Motion and Object Detection
8.Remote Surveillance
9.Time Lapse Photography
10.Panoramic Photography

Features 1-8 address problems in robotics, while Features 9 and 10 address problems in photography. These features, working together, provide a robotic system that will enable its user to operate it with ease and efficiency.

Description Of The Architecture

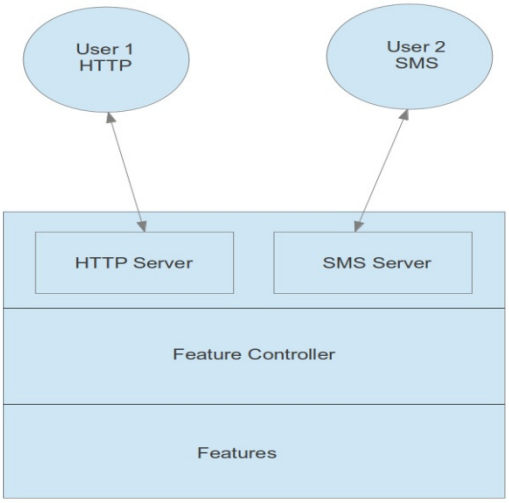
It can be said that, the outcome of our project would be a robotic software platform that allows the user to control the robot over HTTP. Our initial plan was to create a cloud application that acts as a synchronizing server between the robot and the user. The interested party does a HTTP POST to the cloud whenever it needs to update any information. Availability of new information is checked by repeatedly polling the cloud application for new data.



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 for HTML5 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.

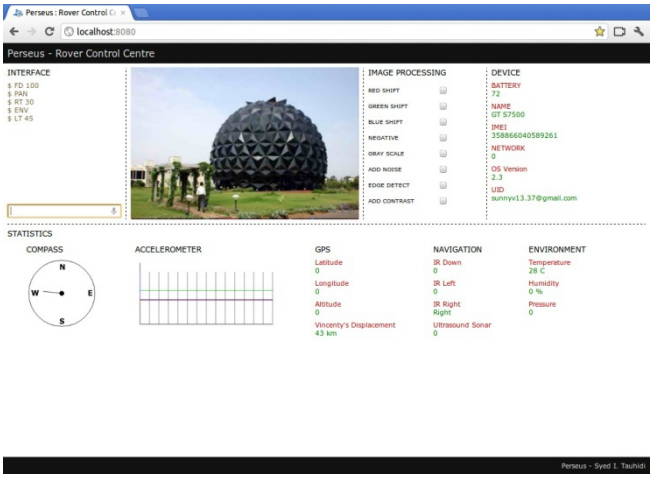
To remove these disadvantages, we removed the server application on the cloud and replace it by creating and embedding our own custom HTTP server in the robot. We created a Communication Layer that rests between the user and the core robotic features. The Communication Layer consists of two Java classes. One acts as as HTTP server and the other as an 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.



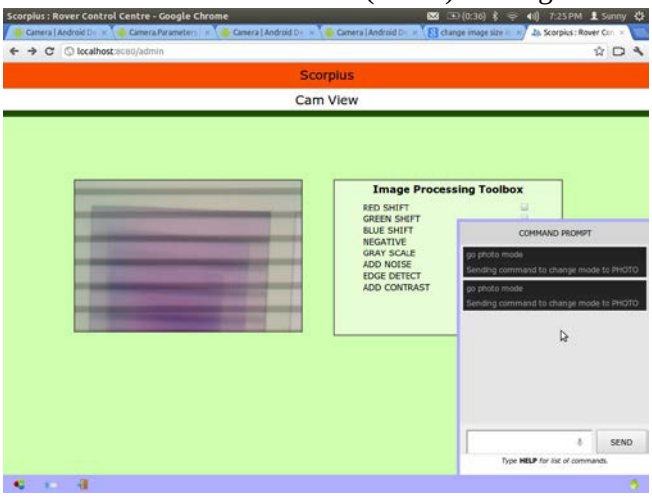
The cloud-based architecture is now deprecated, but an working implementation using Google appEngine on Python is available in the source code repository of the project. The repository is available at bit.ly/scorpius.

The removal of disadvantages of using a cloud based server made possible by integration of an embedded server within the application did enable us to create a new, more dynamic RIA which also has features like 'Voice Synthesis' and 'Motion and Object Detection' built into it.

Given below is an screen-shot of the RIA before the change in architecture. The entire web application is in a single page and the RIA couldn't boast of any feature. But on such a high-latency architecture, it was most apt.



All subsequent screen-shots will be of the new RIA (below) running on the new architecture.



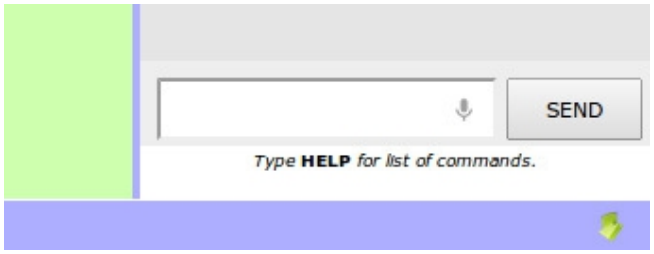
Description Of The Features

Several features are available to the robot through the Android application. Only three of them rely on the physical robot (line-following, obstacle avoiding and panoramic photo), the rest can be used on any Android phone, even without the robot.

Most of the features are implemented using Java, directly on the Android phone, but a few, like 'Motion and Object Detection' and 'Voice Recognition' are implemented using JavaScript on the RIA, which is hosted, as aforementioned, using an embedded HTTP server on the Android phone.

We use a wide variety of APIs, both locally and over the cloud, to implement these features. This saves time and resource by virtue of reusable codes, thus letting us, proverbially speaking, 'stand on the shoulders of giants'. All libraries and packages are in open source and so is this application.

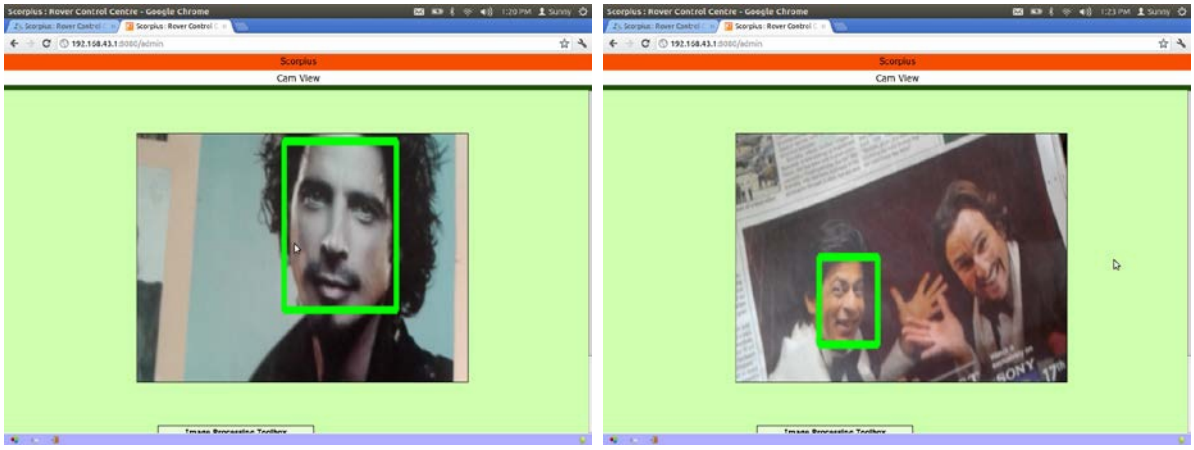
In the section below, we try to describe the various features, along with a summary explaining their implementation techniques.



Voice Recognition allows a user to control the robot using only his voice.

We implemented it using Google Chrome specific APIs. These APIs enable users to use Google Voice Recognition software over the cloud using just a few additional HTML5 tags and JavaScript-callbacks.

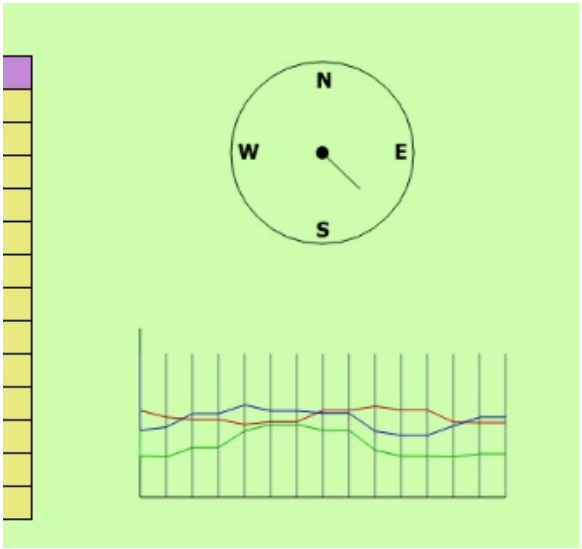
Voice Synthesis allows the user to type a string of text remotely and have the Android phone read out the string aloud. This is implemented using the Android's TTS (Text To Speech) APIs from the following packages : android.speech.tts.TextToSpeech and android.speech.tts.TextToSpeech.OnInitListener.



In the **Face Detection** mode, we process subsequent frames using the Android's camera. If and when a face is detected, a green box is drawn around the face on the output image. It is implemented using openCV and the LBP Cascade Classifier through XML in the Android application.The screen-shots show that the Cascade Classifier does a good job of finding faces in an image, but as can be seen from the third image, it still is not perfect.

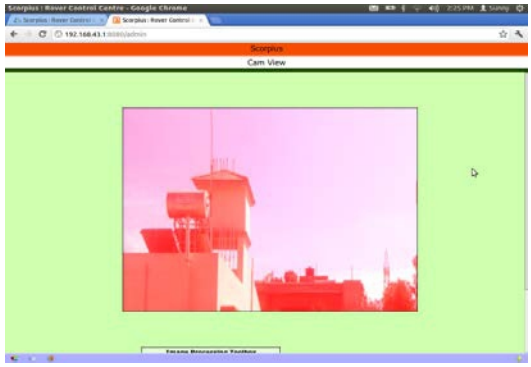
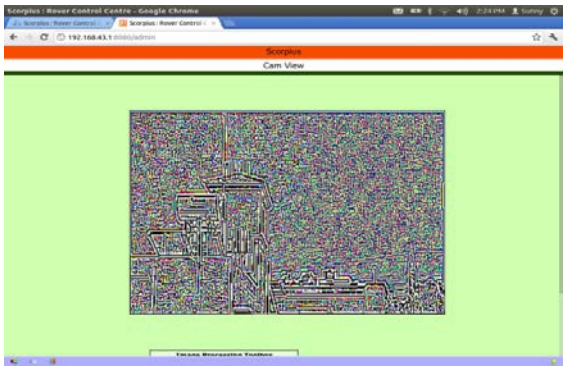
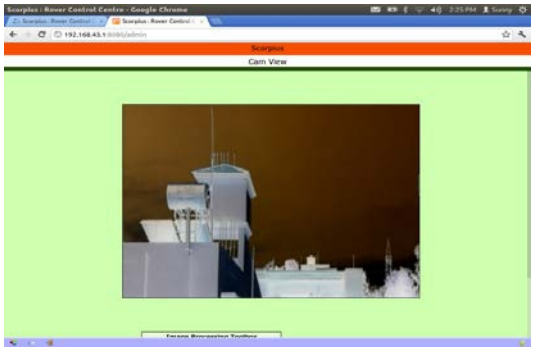
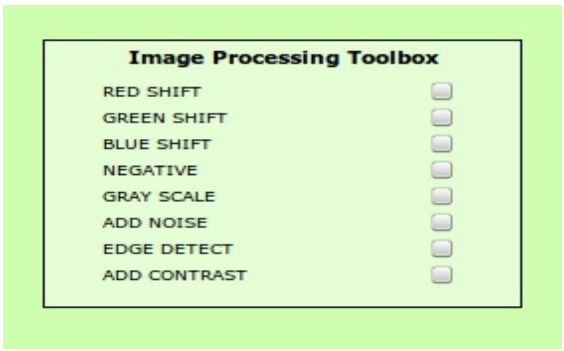
Motion and Object Detection is done by analyzing the sensor readings from Android phone. We primarily use three sensors – accelerometer for motion and vibration detection, proximity meter for object detection and compass to detect presence of electrical devices by looking for effects of electromagnetism.

Name	Status
Compass	134
Latitude	26.331844
Longitude	91.00237
Acceleration - X	3.064578
Acceleration - Y	-7.5082164
Acceleration - Z	4.903325
Proximity	5
IR Down	-
Sonar Front	-
Sonar Left	-
Sonar Right	-
Battery	68
Temperature	350



The Motion and Object Detection takes place in the RIA, which is an innovative, though still experimental, technique for **Robot-User Interaction.** The RIA is made using W3C standard tools, namely HTML5, CSS3 and JQuery-enhanced JavaScript. Some of the HTML5 specifications are very recent, and supported only on recent browsers. We expect the user to have the most recent version of Google Chrome.

The RIA uses pixastic.js for some real-time image processing whenever any video streaming is taking place.

The user can access the various image processing options on the RIA from a toolbox. The user can apply any single option or any combinations of the options by selection the required effect from the toolbox.

Remote Surveillance is the most basic form of live streaming, without the application of any computer vision algorithm on the input image.

Panoramic Photography enables us to capture wide-angled images using standard-lens camera. This is done by capturing several images and stitching them together using openCV.

Time-lapse Photography is the technique of capturing a video at a slow frame-rate and playing it back using normal frame-rate.

Line Follower is the name given to any robot that can move about in its environment by following a line on the ground. Traditionally people use IR sensors to detect lines on the ground, but we'll use only the Android's camera and openCV. There are two major techniques for finding and following a line using Computer Vision. One is to use Hough Line Transform, followed by detection of the vanishing point of the lines. We move the robot towards the angle which directs to the vanishing point. The other method is to set up a 8-bit Color Blob Detector and move the robot in such a way as to keep the blob at the center of the image.

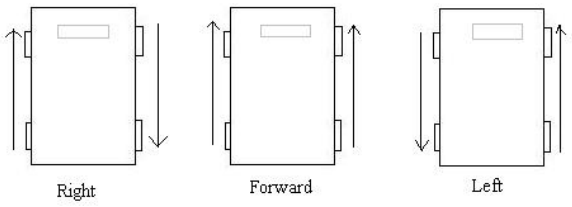
Obstacle Avoidance is another problem in robotics that is traditionally solved using hardware sensors, but which we attempt to solve using the Android's camera and openCV. We do this by looking at individual contours and attempting to navigate by keeping the robot from any colliding with any contour. As of this writing, the Obstacle Avoidance algorithm is yet to be implemented in the Android application.

The Robot-Android Interaction

The Android phone is connected to the robot using either microbridge or ADK using the USB. The Android provides two floating point variables in the range -1 to +1. One indicates angle of direction of motion, where -1 indicates extreme left, +1 indicates extreme right and 0 indicates straight motion. The other variable indicates speed of motion, where -1 indicates fast reverse, +1 indicates fast forward and 0 indicates no motion. These values can also be obtained from the embedded HTTP server over TCP/IP.

The Android application needs the robot to have just two motors with wheels. It doesn't take any reading from the robot and doesn't need the robot to have any sensor. But from the perspective of the robot, the Android phone is just another sensor. The robot may have its own sensors and use them in its own calculation (for example, to make hardware switch that turns off the robot in case of danger), but the Android phone need not know about those sensors.

As mentioned previously, the robot needs just two wheeled motors capable of bi-directional motion. These are used for all robot kinetics using differential drive, as illustrated below.



Description Of Security

Security, as any course in Software Architecture would tell us, is one of the most important 'quality attributes' in a software. Moreover, we need to be more careful about security in an application like ours that depends on wireless communication. Also, HTTP, unlike HTTPS, uses only plain-text, no cyphers. Hence, an application like ours would be vulnerable to sniffing and spooking attacks. Therefore, we have implemented some basic security within the application. The pseudocode for the Android application is given below:

1. Initialize auth = false, IP='127.0.0.1'
2. Wait for user to login from the RIA using MD5 and SHA1 hash of password. Check against user-saved password. If password matches, save user IP and turn auth = true. Else, exit.
3. If auth=true and IP = requestingUsersIP, then all communication between the RIA and the rover are to be encrypted using AES with the password as the key and then transmitted.

The following is the pseudocode for the RIA:

1. When user enters the password, the RIA authenticate the user by sending MD5 and SHA1 hash of entered string. If authentication succeeds, let user access control. Else, reload.
2. All communication between the RIA and the rover are to be encrypted using AES with the password as the key and then transmitted.

This security technique has the advantage of never transmitting password or plain-text over the network.

Observations

The cloud-based architecture, which looked promising initially, turned out to be a redundant system with unwanted latency. We removed these problems by using an embedded HTTP server, but that too comes with its own baggages.

Because the HTTP server has to run at all times, the battery drains rapidly. But this should not be a problem when running the phone in 'accessory mode' using ADK. The second problem is that most low-end mobile phones are not very effective in running multi-threaded applications like these because of the absence of multiple cores in the processor. But, even with these minor defects, the application would be suitable for usage in many areas.

Conclusion

Developing this project has enabled us to understand many aspects of creating a full-fledged real-time system. No other system of similar style and scale exists in the knowledge of the project members. We understood parts of AI and Computer Vision while making this project. The project will get refined as our understand AI and CV improves.

References:

1. *Learning openCV Computer Vision and the openCV library* - Gary Bradski and Adrian Kaehlar
2. *OpenCV 2 - Computer Vision Application Programming Cookbook* - Robert Laganière
3. *Android - A Programmer's Guide* - J. F. DiMarzio
4. *Beginning Android* - Mark L. Murphy
5. *Beginning Arduino* – Michael McRoberts
6. *The Robot Builder's Bonanza – 2nd Edition* – Gordon McComb
7. http://developer.android.com
8. http://www.arduino.cc
9. http://www.opencv.org
10. http://www.talkingelectronics.com/