Android Powered Autonomous GPS Robot
Nishchay Shah¹, Harsh Shah², Aditya Sharma³, Prasanna J. Shete⁴
BE Student¹,²,³, Associate Professor⁴
Department of Computer Engineering
K.J. Somaiya College of Engineering, Mumbai, India

Abstract:
Smartphones have now become extremely powerful with multicore processors, large storage capacities and very rich functionalities. Android can therefore be used as a cost effective and easy platform to be the brain for controlling the robotic device. The purpose of such systems is to provide powerful computational android platforms coupled with simpler robot’s hardware architecture. Existing systems use Bluetooth or Wifi modules for remote controlling the robot. The proposed project deals with making an autonomous GPS robot device with obstacle avoidance using android smartphone.

I. INTRODUCTION

Majority of the mobile devices around the world are currently powered by Android OS. Rich functionalities, large storage capacities and faster processing speeds are some of the basic features provided by android smartphones; very inexpensively. Developing smartphones applications have now become much easier, all thanks to android development environment which provides an easier option for software engineers. By using Java, the most commonly used programming language, developers need not learn new programming languages. Android devices also provide an easy way to deal with hardware components. Android provides communication interfaces for USB, Wifi, Bluetooth and GPS; all of which are necessary components for a mobile robot. Software developers are interested in developing testing-based applications.

The idea is to develop an autonomous GPS robot using Android. Such autonomous robot can be used for many applications such as delivery robot, surveillance bot, etc. Amazon drone delivery system is one of the examples of mobile robots. This kind of a system not only concerns with cost effectiveness in delivering products but also leads to faster delivery of product to customers. The other application is surveillance bots. They can be covert observations or general area mapping. This project is conceptualized on similar ideas.

III. HARDWARE

The hardware requirements for the project include the following:

1. Chassis and Wheels: Acts as a base for the robot. Four wheels are attached to this chassis. Two wheels are attached to DC motors.


3. L293D Motor Driver: The L293D motor driver is responsible for controlling the DC motor movement and is controlled by the Arduino signals. The commands for movements are fed into Arduino board via smartphone which then in coordination with motor driver performs the movement.

4. Li-ion Batteries: These are required as a power supply for the L293D and DC motors. Total amount of the power required is 15V. However, for Arduino, the Android device acts as source of power.

Schematic representation of circuitry is shown below:
Software & Algorithms
The software is divided into various parts which are as follows:

**Phone mirroring**
The user/controller can utilize this application for surveillance, purpose, for example, in case of search-bots used during calamities like earthquakes. Hence in our project, a client-server based mirroring is implemented. The user is able to mirror the android device screen installed on the robot and will be able to feed in the destination coordinates, view the camera feed.

**Haversine- Bearing formula**
Haversine- Bearing formula is used to guide robot once we get the set of latitude and longitude to be covered in the path.

**Haversine Formula**
Haversine Formula calculates geographic distance on earth. Using two latitude-longitude values, we can easily calculate the spherical distance between the points by Haversine formula. Haversine formula is used to break path into straight lines.

\[
\begin{align*}
\sin^2 \left( \frac{\Delta \text{lat}}{2} \right) &+ \cos \text{lat}_1 \cdot \cos \text{lat}_2 \cdot \sin^2 \left( \frac{\Delta \text{lon}}{2} \right) \\
& = \frac{a}{\sin^2(\Delta \text{lat}/2) + \cos(\text{lat}_1) \cdot \cos(\text{lat}_2) \cdot \sin^2(\Delta \text{lon}/2)}
\end{align*}
\]

**Bearing Formula**
Bearing can be defined as direction or an angle, between the north-south line of earth or meridian and the line connecting the target and the reference point. While Heading is an angle or direction where you are currently navigating in. Bearing value is the angle from the heading value we need to take to align the robot in the right direction align the path from point A to B. It is measured from the North direction.

Let ‘R’ be the radius of Earth,
‘L’ be the longitude,
‘\theta’ be latitude,
‘\beta’ be Bearing,

- 0° : North
- 90° : East
- 180° : South
- 270°: West.

Bearing from point A to B, can be calculated as:

\[ \beta = \text{atan2}(X,Y) \]

where, X and Y are two quantities and can be calculated as:

\[ X = \cos \theta_a \cdot \sin \Delta L \]
\[ Y = \cos \theta_a \cdot \sin \theta_b - \sin \theta_a \cdot \cos \theta_b \cdot \cos \Delta L \]

The flowchart about the usage of the Haversine and Formula followed by obstacle handling is as follows:

**Canny Edge Detection Algorithm**
Canny Edge algorithm, developed by J.F. Canny, is one of most popular edge detection algorithm. This multistage algorithm has the following stages:

**Noise Reduction**
To smooth the image, a 5x5 Gaussian filter is applied to convolve with the image. The equation for a Gaussian filter kernel of size (2k+1) x (2k+1) is given by:

\[ H_{ij} = \frac{1}{2\sigma^2} \exp \left( -\frac{(i - (k + 1))^2 + (j - (k + 1))^2}{2\sigma^2} \right); 1 \leq i, j \leq (2k + 1) \]

A 5x5 is a good size for most cases. With increase in size of the filters, edge detector’s sensitivity to noise decreases.

**Intensity Gradient of the Image**
Sobel kernel filter is applied to the smoothed image in horizontal as well as vertical direction to get first derivative in horizontal (Gx) and vertical direction (Gy). From these two formulae, we find out the direction for each pixel and edge gradient as follows:

\[ \text{Edge Gradient} (G) = \sqrt{G_x^2 + G_y^2} \]

\[ \text{Angle} (\theta) = \tan^{-1} \left( \frac{G_y}{G_x} \right) \]

**Non-maximum Suppression**
Next step is to remove any unwanted pixel which is not a part of the edge. To achieve this, each and every pixel is checked if it is
a local maximum in its neighborhood in the direction of gradient. Check the image below:

Point A is on the edge in the vertical direction. Gradient direction is normal to the edge. Point B and C are in gradient directions. So A is checked with B and C to see if it forms a local maximum. If it does, it is considered for next stage, else, it is suppressed. In short, the result you get is a thin edged binary image.

IV. HYSTERESIS-

This stage decides whether an edge is a real edge or not. We need two threshold values - minVal and maxVal. Intensity gradients of the edge more than maxVal is sure to be a real edge and the one below minVal is sure to be non-edge, therefore discarded. The ones which lie in between maxVal and minVal are classified edges or non-edges based on their connectivity. If they are connected to edge pixels, they are considered to be part of edge pixel. Otherwise, they are discarded. Canny edge detection is implemented in Android using OpenCV (Open-source Computer Vision) library. OpenCV puts all the above in single function. The question now is that we need to implement obstacle avoidance using this Canny edge algorithm. Obstacle avoidance is one of the most important concepts of mobile robotics. First of all real time image is captured using the camera of the android device. This image is then converted to edge only version using the Canny Edge Algorithm as follows:

One can see that the obstacles are roughly outlined now. This helps to identify an object, but fails to give us a correct bearing, that is: what direction to go in order to avoid the obstacles. For this purpose we use the following modules or follow the steps given below:

**Side Fill**
The Side Fill module tops up the black area of an image starting from top to bottom and proceeds until a non-black pixel is found. It’s like, the water drops start from the top of the image, stopping only at the non-black pixels. The Side Fill output of the above image is as follows:

**Erode**
You will quickly notice the single width vertical lines that appear due to the holes where the edge detection fails. For removing this inconsistency we use Erode module, just eroding or shrinking the current image horizontally by an amount such that the resulting white areas would sufficient for the robot to pass without hitting the obstacle. The objects which are connected to each other will be separated. One’s that are too thin may disappear entirely. This module is useful for removing noise from an image. This module helps to remove all small objects and the remaining one’s to have smoother boundaries. The Erode module output of the above image is as follows:

**Smooth Hull**
The end result of the Erode module gives us a smooth boundaries, next step will be to smooth the entire structure to ensure that any point picked as the goal direction is in the middle of a potential path. Using the Smooth Hull, we can round out flat plateaus to give us better peaks i.e., smooth a blob's shape. The blob's perimeter is averaged within the specified window. The Smooth Hull output is as follows:

**Point Location**
Point Location is nothing but to locate or identify the highest
point which represents the most distant goal that the robot could head towards avoiding all the obstacles. The highest point is identified by a red square. This module provides a quick way to identify specific coordinates within the image based on their location. The Point Location output is as follows:

Finally we just merge the current point back into the original image. This will help us gauge or identify if that location appears to be a reasonable result. Given the location of point X is at 193 and the middle of the image at 160 (assuming 320x240 as value set in camera) we will move the robot straight. The final output depicting the highest point is as follows:

Significance of the project

1. Socio-Economic significance
The project will inspire other researchers and developers to develop system with more effective and enhanced features. The purposes of delivery and surveillance will be well served as compared to existing systems.

2. Technological significance
We want people to stay advanced with the technologies available. In addition, obstacle avoidance through smartphone may result in total new dimension for similar projects which is cheaper than installing camera on device or using ultrasonic modules for obstacle detection. Talk about GPS and demand for autonomous tech. Moreover, GPS provides critical positioning capabilities around the world with error area of 10 meters which reduces even further in our project as robot can be controlled closely by using Wifi. In addition, autonomous technology which means robots working with any guidance provided by humans is gaining pace and in future we are focused on delivering it.

System Architecture
The user android smartphone will communicate with that on the robot using the Phone mirror application which is Wifi based.

The android smartphone on the robot is connected to the Arduino board via USB cable which is attached to smartphone ba a OTG cable. The Arduino is connected to the L293D motor driver via jumper cables. The motor driver is responsible for rotating the motors. We have used 15V Li-ion battery as power supply, while the power supply for the Arduino board is through the connected Android smartphone. Multiple features are implemented using the smart phone’s capabilities.
2. OnRobot Application is as follows:
OnRobot application runs on the Android device installed on the robot. By using the phone mirroring application, the destination latitude and longitude are entered and submitted. For close control, the arrow buttons are provided. The capture button is used for real time surveillance. The track button is for user to mark his destination using markers on the map.

3. GPS Path Generation Snapshot:
By clicking on the map, markers appear which indicates the start and destination point, and corresponding path is generated. New destination marker can be directly set on the map.

4. Real time GPS Tracking Snapshot:
This activity keeps track of real time location of the robot along with the timestamp. The latitude, longitude and timestamp are stored in a file. It keeps track whether the robot is not stationary for more than the threshold time. If it is greater than the threshold time, the controller is notified about the location of robot by SMS.

5. Canny Edge Detection Snapshot:
This activity is responsible for real time canny edge obstacle detection.

V. CONCLUSION
Existing systems use separate GPS, IR, Bluetooth modules connected to Arduino board. Also, there are mechanism of controlling a surveillance robot using android mobile devices through socket programming or using in built accelerometer and wifi module. For all these systems, not only do we require skill to control the robot but also these existing systems are costly. For developing such systems, we need to handle all the modules like accelerometer, compass, wifi control etc. individually. Our system can be easily made using an android phone and is economically feasible, so users can install the application required for controlling the robot. This system can be used for educational purposes, and it can also be scaled into a delivery robot, like that of Amazon; or even a rescue bot in case of natural disasters.
VI. FUTURE SCOPE

The proposed project can be improved by building a more compact design of the robot. Along with the implemented Canny edge detection, usage of sensors can be incorporated to get more accurate results. Also, plenty of safety measures for the whereabouts of the bot can be implemented. It can be used in drone instead of ground-based implementation; with certain changes. GSM control can be provided for the robot.

VII. REFERENCES


[5]. https://en.wikipedia.org/wiki/Canny_edge_detector

[6]. www.docs.opencv.org

[7]. http://www.igismap.com/formula-to-find-bearing-or-heading-angle-between-two-points-latitude-longitude/