An Indoor Navigation System For Smartphones

Abhijit Chandgadkar
Department of Computer Science
Imperial College London

June 18, 2013

Abstract

Navigation entails the continuous tracking of the user's position and his surroundings for the purpose of dynamically planning and following a route to the user's intended destination. The Global Positioning System (GPS) made the task of navigating outdoors relatively straightforward, but due to the lack of signal reception inside buildings, navigating indoors has become a very challenging task. However, increasing smartphone capabilities have now given rise to a variety of new techniques that can be harnessed to solve this problem of indoor navigation.

In this report, we propose a navigation system for smartphones capable of guiding users accurately to their destinations in an unfamiliar indoor environment, without requiring any expensive alterations to the infrastructure or any prior knowledge of the site's layout.

We begin by introducing a novel optical method to represent data in the form of markers that we designed and developed with the sole purpose of obtaining the user's position and orientation. Our application incorporates the scanning of these custom-made markers using various computer vision techniques such as the Hough transform and the Canny edge detection. In between the scanning of these position markers, our application uses dead reckoning to continuously calculate and track the user's movements. We achieved this by developing a robust step detection algorithm, which processes the inertial measurements obtained from the smartphone's motion and rotation sensors. Then we programmed a real-time obstacle detector using the smartphone camera in an attempt to identify all the boundary edges ahead and to the side of the user. Finally, we combined these three components together in order to compute and display easy-to-follow navigation hints so that our application can effectively direct the user to their desired destination.

Extensive testing of our prototype in the Imperial College library revealed that, on most attempts, users were successfully navigated to their destinations within an average error margin of 2.1m.

Acknowledgements

I would like to thank Dr. William J. Knottenbelt for his continuous support and guidance throughout the project. I would also like to thank Prof. Duncan Gillies for his initial feedback and assistance on computer vision. I would also like to thank Tim Wood for his general advice on all aspects of the project. I would also like to thank all the librarians on the third floor of the Imperial College central library for allowing me to use their area to conduct my experiments. Finally, I would like to thank all my family and friends who helped me test my application.

Contents

Chapter 1

Introduction

Navigation is the process of accurately establishing the user's position and then displaying directions to guide them through feasible paths to their desired destination. The Global Positioning System (GPS) is the most common and the most utilised satellite navigation system. Almost every aircraft and ship in the world employs some form of GPS technology. In the past few years, smartphones have evolved to contain a GPS unit, and this has given rise to location-based mobile applications such as geofencing and automotive navigation for the common user. However, GPS has its limitations. In particular we are concerned with the lack of GPS signal reception in indoor environments. GPS satellites fail to deliver a signal to a device if there is a direct obstruction on its path. Therefore we have to consider alternate methods of achieving indoor navigation on a smartphone.

1.1 Motivation

Our motivation for this project stems from the fact that people are increasingly relying upon their smartphones to solve some of their common daily problems. One such problem that smartphones have not yet completely solved is indoor navigation. At the time of writing, there is not a single low-cost scalable mobile phone solution available in the market that successfully navigated a user from one position to another indoors.

An indoor navigation app would certainly benefit users who are unfamiliar with a place. Tourists, for instance, would have a better experience if they could navigate confidently inside a tourist attraction without any assistance. In places such as museums and art galleries, the application could be extended to plan for the most optimal or 'popular' routes. Such a system could also be integrated at airports to navigate passengers to their boarding

gates. Similarly an indoor navigation system could also benefit local users who have previously visited the location but are still unaware of the whereabouts of some of the desired items. These include supermarkets, libraries and shopping malls. The application could also benefit clients who install the system by learning user behaviours and targeting advertisements at specific locations.

1.2 Objectives

The objective of this project was to build a robust and flexible smartphone based indoor navigation system that met the following four criteria:

- High accuracy: The application should consistently guide users to their destinations within a reasonable distance.
- Low-cost: The application should not require any expensive infrastructural changes to obtain accurate positioning data. Clients will not be interested in large investments unless they financially benefit from it. Future maintenance costs on these equipment may further deter the choice of this solution.
- No pre-loaded indoor maps: The application should be able to navigate the user without requiring a pre-loaded map of the environment. Plotting the layout of a site is cumbersome and can diminish the flexibility of a solution. Only the position of the items/point of interests may be stored with respect to a site's frame of reference.
- Intuitive user interface (UI): The application should have an easy-to-use UI that displays navigation hints correctly based on the user's current state. The application should also take into account the obstacles surrounding the user to avoid displaying any incorrect hints. For instance, it should not tell users to go straight if there is an obstacle immediately ahead of them.

From our research we realised that various smartphone based solutions exist that accurately determine a user's current position. Some of them require no additional infrastructural changes while some even display navigation hints to the user. However none of these solutions integrate all the desired aspects of an indoor navigation system to meet the four criteria mentioned above.

1.3 Contributions

In this report we present an indoor navigation system for smartphones, which uses a combination of computer vision based techniques and inertial sensors to accurately guide users to their desired destinations. Our solution entails the scanning of custom-made markers in order to calibrate the user's position during navigation. Then it employs a dead reckoning algorithm to approximate user movements from the last known point. Finally our application uses this information along with an integrated vision based obstacle detector to display correct directions in real-time leading to the user's destination.

Our indoor navigation solution required the study and development of three individual components prior to their integration:

1. Position markers: These are custom markers that our application is capable of scanning from any angle using the smartphone camera. Colour is used to encode position data along with a direction indicator to obtain the angle of scanning. These markers were used to calibrate the user's position and orientation. OpenCV functions were used to detect circles and other features from the camera preview frames to decode these markers.

2. Obstacle detection: Our application detects obstacles in the environment in real-time using the smartphone camera. The purpose of this task was to avoid giving users directions towards a non-feasible path. The Hough line transform was primarily used for detecting all the boundary edges from the incoming preview frames.

3. Dead reckoning: Our application uses inertial dead reckoning to estimate the position and orientation of the user from the last scanned position marker. This enabled the application to always keep track of the user's position and also notify them if they reach their destination. To achieve this, the accelerometer signal was first pre-processed to reduce noise and then analysed for step detection. This was combined with the device's orientation to develop our algorithm.

The final application features the integration of these three components, as shown in figure 1.1, in order to calculate and correctly navigate the user to the next best position that would eventually lead them to their desired destination. Results from our evaluation demonstrated that our end product achieved just over 2m accuracy with the help of only eight position markers over a testing area of 25mx15m. In addition, we did not have to provide our application with an indoor map of the site.

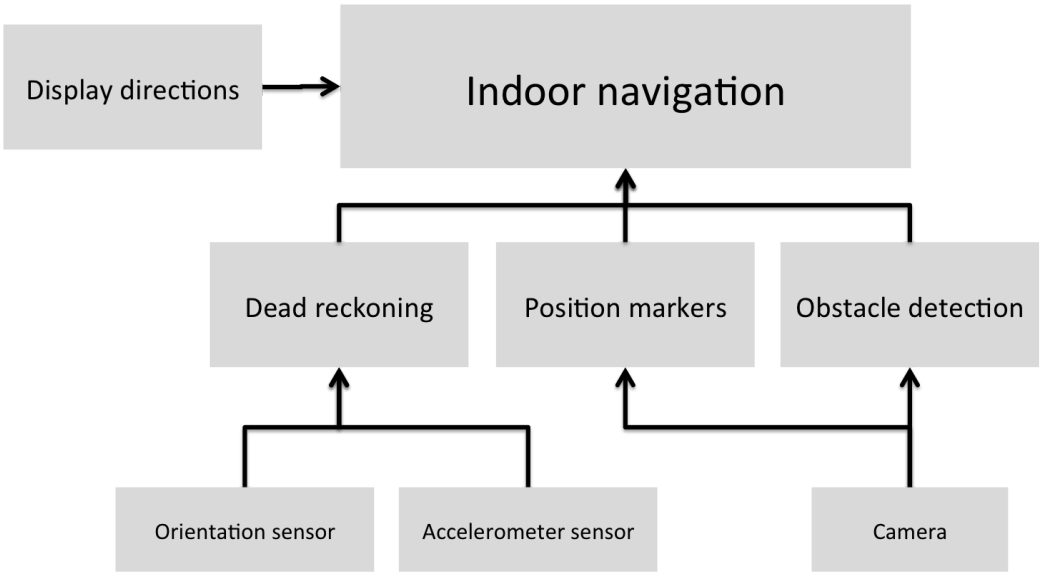


Figure 1.1: The image shows how all the main components integrate to make the final indoor navigation system

1.4 Report outline

Our entire report is structured on the basis of the three individual components mentioned in section 1.3 and their integration. Chapter 2 describes some of the related work in this domain and provides a technical background analysis of the various concepts required to achieve our solution. Chapters 3, 4 and 5 provide an in-depth explanation of our implementation for the position markers, our vision based obstacle detection mechanism and our dead reckoning algorithm respectively. Chapter 6 describes our approach to integrating these three components together as well as gives an overview of the entire system. Chapter 7 evaluates each of the individual components separately and then follows it up with a quantitative and qualitative analysis of the final product.

Chapter 2

Background

In this chapter, we begin by giving a brief overview on our choice of smartphone platform. Then we discuss some of the existing state-of-the-art research carried out in the domain of indoor navigation. We also assess why none of the current proposals meet our objective criteria. After that, we study various computer vision concepts that will be relevant across this entire report. Finally, we assess individually some of the related work conducted for the three components mentioned in section 1.3.

2.1 Smartphone development overview

We chose to develop the application on the Android platform due to the increasing number of Android users across the globe, the strong online community and fewer developer restrictions. In addition we also had previous programming experience on Android, and therefore we were familiar with most of their APIs. The prototype for our proposed solution would be developed and tested on the Samsung Galaxy S4. The smartphone's 13-megapixel camera and its two quad-core central processing units (CPU) further enhanced the performance of our application.

Sensors would also be crucial for our application. Most Android-powered devices have built-in sensors that measure the motion and the orientation of the device. In particular, we analysed the raw data retrieved from the accelerometer and the rotation vector. The accelerometer gives us a measure of the acceleration force in m/s^2 applied to the device on all the three physical axes (x, y, z). The rotation vector fuses the accelerometer, magnetic field and gyroscope sensors to calculate the degree of rotation on all the three physical axes (x, y, z)[10].

2.2 Related work

In the past few years, a great amount of interest has been shown to develop indoor navigation systems for the common user. Researchers have explored possibilities of indoor positioning systems that use Wi-Fi signal intensities to determine the subjects position[14][4]. Other wireless technologies, such as bluetooth[14], ultra-wideband (UWB)[9] and radio-frequency identification (RFID)[31], have also been proposed. Another innovative approach uses geo-magnetism to create magnetic fingerprints to track position from disturbances of the Earths magnetic field caused by structural steel elements in the building[7]. Although some of these techniques have achieved fairly accurate results, they are either highly dependent on fixed-position beacons or have been unsuccessful in porting the implementation to a ubiquitous hand-held device.

Many have approached the problem of indoor localisation by means of inertial sensors. A foot-mounted unit has recently been developed to track the movement of a pedestrian[35]. Some have also exploited the smartphone accelerometer and gyroscope to build a reliable indoor positioning system. Last year, researchers at Microsoft claim they have achieved metre-level positioning accuracy on a smartphone device without any infrastructure assistance[17]. However, this system relies upon a pre-loaded indoor floor map and does not yet support any navigation.

An altogether different approach applies vision. In robotics, simultaneous localisation and mapping (SLAM) is used by robots to navigate in unknown environments[8]. In 2011, a thesis considered the SLAM problem using inertial sensors and a monocular camera[32]. It also looked at calibrating an optical see-through head mounted display with augmented reality to overlay visual information. Recently, a smartphone-based navigation system was developed for wheelchair users and pedestrians using a vision concept known as ego-motion[19]. Ego-motion estimates a cameras motion by calculating the displacement in pixels between two image frames. Besides providing the application with an indoor map of the location, the method works well under the assumption that the environment has plenty of distinct features.

Localisation using markers have also been proposed. One such technique uses QR codes[1] to determine the current location of the user[13]. There is also a smartphone solution, which scans square fiducial markers in real time to establish the user's position and orientation for indoor positioning[24]. Some have even looked at efficient methods to assign markers to locations for effective navigation[6]. Although, scanning markers provide high precision

[1] www.qrcode.com

positioning information, none of the existing techniques have exploited the idea for navigation.

Finally, we also looked at existing commercial indoor navigation systems available on the smartphone. Aisle411 (aisle411.com) provided a scalable indoor location and commerce platform for retailers, but only displayed indoor store maps of where items were located to the users without any sort of navigation hints. The American Museum of Natural History also released a mobile app (amnh.org/apps/explorer) for visitors to act as their personal tour guide. Although, the application provides the user with turn-by-turn directions, it uses expensive Cisco mobility services engines to triangulate the device's position.

2.3 Computer vision

Computer vision is the study of concepts behind computer-based recognition as well as acquiring images and extracting key features from them. Our application heavily relies on some of these concepts. In particular, we are concerned with shape identification, edge detection, noise reduction, motion analysis and colour.

2.3.1 Hough Transform

The Hough transform is used to detect curves such as lines, circles, ellipses, etc. in an image. The idea behind Hough line transform is that every point in a binary image is treated as a point on a line that we are trying to detect. Therefore, it models all the different line equations that pass through that point and finds the line equation that has the most number of binary points.

Hough line transform

An equation of a line expressed in the Cartesian system looks as follows.

$$y = mx + c$$

In the polar coordinate system, we use the parameters r and θ to write the line equation as follows.

$$r = xcos(\theta) + ysin(\theta)$$

Then for every non-zero pixel in the binary image, we model all the possible line equations that pass through that point between $r > 0$ and $0 \leq \theta \leq 2$

A simple mathematical calculation of how a Hough line transform finds the equation of a detected line is given in Appendix A.

Hough circle transform

The Hough circle transform is similar to the Hough transform for detecting straight lines. An equation of a circle is characterised by the following equation.

$$(x - x_c)^2 + (y - y_c)^2 = r^2$$

In order to detect circles in a given image, the centre coordinate (x_c, y_c) of the circle and its radius r have to be identified. As three different parameters, x_c, y_c and r, are modelled, the graph would have 3-dimensions. Each non-zero pixel in the binary image will produce a conical surface as shown in figure 2.1[2].

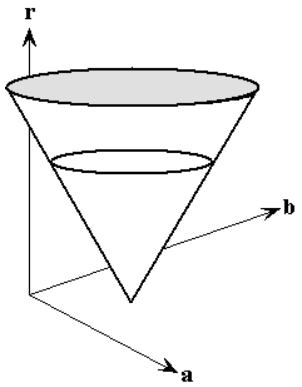


Figure 2.1: The image shows a cone formed by modelling all the possible radius of a circle with the centre point at a 2D coordinate

Once again, this process will be repeated for every non-zero pixel point and will result in several such cones plotted on the graph. This can conveniently be represented in a three-dimensional matrix. When the number of intersections exceeds a certain threshold, we consider the detected three-dimensional coordinate as our centre and radius.

2.3.2 Gaussian smoothing

Smoothing is an image processing operation primarily used to reduce noise. Filters are generally used to smooth (blur) an image. A filter uses a matrix of coefficients, called the kernel, and neighbouring pixel values to calculate the new intensity for every pixel in a given image. Amongst many different

[2]The image is taken from `http://www.cis.rit.edu/class/simg782.old/talkHough/HoughLecCircles.html`

filters, Gaussian filters are perhaps the most useful in our application. They are typically used to reduce image noise prior to edge detection.

The theory behind Gaussian filters stem from the following two-dimensional Gaussian function, studied in statistics, where μ is the mean and σ is the variance for variables x and y.

$$f(x,y) = Ae^{-\left(\frac{(x-\mu_x)^2}{2\mu_x^2} + \frac{(y-\sigma_y)^2}{2\sigma_y^2}\right)}$$

This formula produces a convolution matrix, called the Gaussian kernel, with values that decrease as the spatial distance increases from the centre point. Figure 2.2 can help to visualise the spread of the weights for a given pixel and its neighbours.

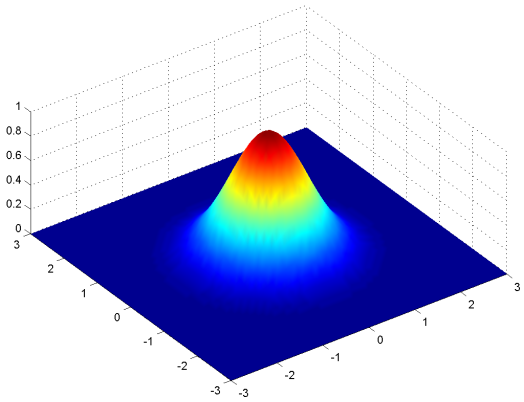


Figure 2.2: The image shows a plot of a two dimensional Gaussian function

When a Gaussian filter is applied to an image, each pixel intensity is convoluted with the Gaussian kernel and then added together to output the new filtered value for that pixel. This filter can be applied with different kernel sizes resulting in different levels of blurring. The larger the kernel size, the more influence the neighbouring pixels will have on the final image.

2.3.3 Canny edge detection

To detect edges, the intensity gradient of each pixel is examined to see if an edge passes through it or close to it. The most "optimal" edge detection technique was developed by John Canny in 1986[5]. The algorithm consists of four key stages.

1. **Noise reduction** - The Canny edge detector is highly sensitive to noisy environments. Therefore, a Gaussian filter is initially applied to the raw image before further processing.

2. **Finding the intensity gradient** - To determine the gradient strength and direction, convolution masks used by edge detection operators such as Sobel (shown below) are applied to every pixel in the image. This yields the approximate gradient in the horizontal and vertical directions.

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

The gradient strength/magnitude can then be calculated using the law of Pythagoras.

$$G = \sqrt{G_x^2 + G_y^2}$$

The direction of the edge can also be quickly determined.

$$\theta = tan^{-1}\left(\frac{G_y}{G_x}\right)$$

This angle is then rounded to one of 0°, 45°, 90°or 135°corresponding to horizontal, vertical and diagonal edges.

3. **Non-maximum suppression** - The local maxima from the calculated gradient magnitudes and directions are preserved whereas the remaining pixels are removed. This has the effect of sharpening blurred edges.

4. **Edge tracking using hysteresis thresholding** - Double thresholding is used to distinguish between strong, weak and rejected edge pixels. Pixels are considered to be strong if their gradient lies above the upper threshold. Similarly, pixels are suppressed if their gradient is below the lower threshold. The weak edge pixels have intensities between the two thresholds. The result is a binary image with edges preserved if they contain either strong pixels or weak pixels connected to strong pixels.

2.3.4 Colour

Colours have been previously used to encode data. Microsoft's High Capacity Color Barcode (HCCB) technology encodes data using clusters of coloured triangles and is capable of decoding them in real-time from a video stream[36]. Although their implementation is very complex, we can use the basic concept behind HCCB in our application.

Each distinct colour can be used to represent a certain value. Colours can be grouped together in a set format to encode a series of values. We have to take into account that smartphone cameras cannot distinguish between small variations of a certain colour in non-ideal situations, such as light green or dark green. Therefore we would be limited on the number of discrete values we can encode. Colour is typically defined using the "Hue Saturation Value" (HSV) model or the "Red Green Blue" (RGB) model.

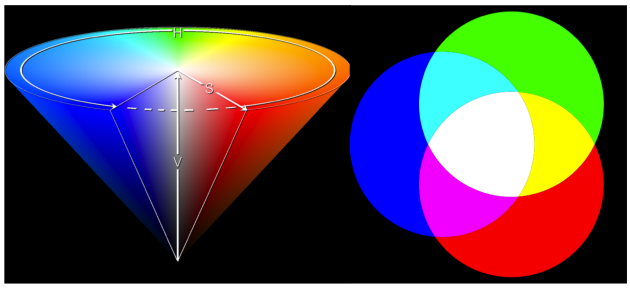


Figure 2.3: The left image shows the HSV model and the right image shows the RGB model. They both describe the same thing but with different parameters

The HSV model is more appropriate for the identification and comparison of colours. The difference in the hue component makes it easier to determine which range a colour belongs to. For example, the colour red has a hue component of 0 ± 15 while green has a hue component of 120 ± 15.

2.3.5 OpenCV

Open Source Computer Vision (OpenCV) is a library of programming functions for real time computer vision. It is released under a BSD license allowing us to use many of their optimised algorithms for academic and commercial purposes[27]. The library is cross-platform and ports to all the major mobile operating systems. For Android, the OpenCV manager app needs to be installed on the testing device prior to development. It is an Android service targeted to manage OpenCV library binaries on end users devices[26].

The library supports the calculation of Hough transforms, Canny edge detection and optical flow. It also provides various smoothing operations including Gaussian smoothing, as well as image conversion between RGB, HSV and grayscale. These algorithms are highly optimised and efficient, but they only produce real-time performance for low resolution images.

2.4 Positioning

In order to develop a navigation system, the application needs to be aware of the user's position. There are numerous methods available that solve the indoor positioning problem but we had to only consider those that were accessible on a smartphone device, and would minimise the number of infrastructure changes.

2.4.1 Barcode scanning

Barcodes could be placed in various locations across the building, encoded with their respective grid coordinates. The smartphone camera could then be used to take a picture of the barcode and decode the encoded data.

The simplest type of linear barcode is Code 39. To encode a given piece of data, a Code 39 encoding table is used. It contains the mapping between the 43 accepted symbols and their unique 12-bit binary code where '1' stands for a black bar and '0' stands for a white space of equivalent width. The same symbol can be described using another format based on width encoding. So narrow (N) represents a thinner bar/space (1/0) while wide (W) represents a broader bar/space (11/00). The barcode encoding for the '*' symbol is always used as the start and stop character to determine the direction of the barcode. In addition, a white space is always encoded between the characters in a barcode.

Users can regularly scan these position barcodes to keep the application up to date with the user's last position. Open-source barcode scanning libraries are available for smartphones and support the scanning of Code 39 barcodes. ZXing is very popular amongst the Android and iPhone developers[33]. It has a lot of support online and it is well documented. The other major advantage of using barcodes is that they are cheap to produce and can store any type of static data. However, for a navigation application, directions needed to be provided from the moment a user scans a barcode. Therefore, we would need to determine the user's orientation at the point of scanning. We cannot encode such information in any type of barcode. Another drawback with using barcode scanning libraries is their integration with the rest of the application. If our application has to scan barcodes, detect obstacles and provide users with correct directions all at the same time, we would need to thoroughly understand and modify the barcode scanning library to be able to extend and integrate it.

2.4.2 Location fingerprinting

Location fingerprinting is a technique that compares the received signal strength (RSS) from each wireless access point in the area with a set of pre-recorded values taken from several locations. The location with the closest match is used to calculate the position of the mobile unit. This technique is usually broken down in to two phases[36]:

1. **Offline sampling** - Measuring and storing the signal strength from different wireless routers at selected locations in the area
2. **Online locationing** - Collecting signal strength during run time and using data from the offline samples to determine the location of the mobile device

With a great deal of calibration, this solution can yield very accurate results. However, this process is time-consuming and has to be repeated at every new site.

2.4.3 Triangulation

Location triangulation involves calculating the relative distance of a mobile device from a base station and using these estimates to triangulate the user's position[16]. Distance estimates are made based on the signal strength received from each base station. In order to resolve ambiguity, a minimum of three base stations are required.

In free space, the received signal strength (s) is inversely proportionate to the square of the distance (d) from the station to the device.

$$s \propto \frac{1}{d^2}$$

Signal strength is affected by numerous factors such as interference from objects in the environment, walking, multipath propagation[3], etc. Therefore, in non-ideal conditions, different models of path attenuation need to be considered.

2.4.4 Custom markers

Markers can be designed tailored to meet our application requirements. Besides encoding the position coordinates, they could be extended to encode fiducial objects that allow the calculation of the user's orientation at the

[3]Multipath propagation causes signal to be received from two or more paths

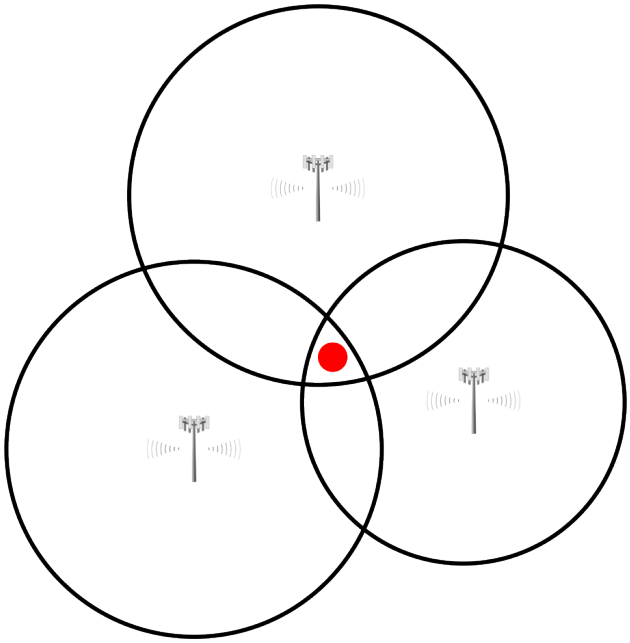


Figure 2.4: The image shows the trilateration of a device using the signal strength from three nearby cell towers

point of scanning. We would need to define our own encoding technique as well as develop a scanning application to decode the marker data. In order to extract key features and interpret the scanned image, we would need to apply some of the computer vision concepts mentioned in section 2.3

2.5 Obstacle detection

Our application needs to detect free space around the user in real-time in order to make a decision on which path (left, right, straight or backwards) to take in the short-term to reach the destination. For a smartphone implementation, the camera is the only self-contained technology available that we can exploit for this purpose.

Depth sensors are commonly used in Robotics[23] to avoid obstructions but very few have explored the problem using vision. A popular application of this problem is in road detection to aid autonomous driving. The approach taken by[30] computes the vanishing point to give a rough indication of the road geometry. Offline machine learning techniques have also been developed that use geometrical information to identify the drivable area[2]. However, the idea behind outdoor free space detection does not work well indoors due

to the absence of a general geometric pattern and the irregular positioning of challenging structures.

An interesting approach taken by a group in the 2003 RoboCup involved avoiding obstacles using colour[12]. Although this is a relatively straightforward solution and achieves a fast and accurate response, it restricts the use of an application to a certain location and is prone to ambiguous errors caused by other similar coloured objects in the environment. Another impressive piece of work combines three visual cues from a mobile robot to detect horizontal edges in a corridor to determine whether they belong to a wall-floor boundary[18]. However, the algorithm fails when strong textures and patterns are present on the floor.

There has been very little emphasis on solving the problem on a smartphone mainly due to the high computational requirements. There is nevertheless one mobile application tailored for the visually impaired that combines colour histograms, edge cues and pixel-depth relationship but works with the assumption that the floor is defined as a clear region without any similarities present in the surrounding environment[29].

There is currently a vast amount of research being conducted in this area. However, our focus was driven towards building a navigation system and not a well-defined free space detector. Therefore, for our application, we have adopted some of the vision concepts mentioned in literature such as boundary detection.

2.6 Dead reckoning

Given the initial position, our application needs to be aware of the user's displacement and direction to be able to navigate them to their destination. This process is known as dead reckoning. On a smartphone, there are two possible ways to accomplish this task without being dependent on additional hardware components.

2.6.1 Inertial sensors

The accelerometer sensor provides a measure of the acceleration force on all the three physical axes (x, y, z). Double integration of this acceleration data yields displacement as follows

$$v_f = v_i + a \cdot t$$

$$d = v_f \cdot t - 0.5 \cdot a \cdot t^2$$

However, due to the random fluctuations in the sensor readings, it is not yet possible to get an accurate measure of displacement even with filtering[4]. Nevertheless, the accelerometer data can be analysed to detect the number of footsteps. In that case, a rough estimate of the distance travelled can be made, provided the user's stride length is known. Furthermore, the orientation sensor can be employed simultaneously to determine the direction the user is facing. Using this information, the new position of the user can be calculated on each step as follows:

$$x_{new} = x_{old} + cos(orientation) \times stridelength$$
$$y_{new} = y_{old} + sin(orientation) \times stridelength$$

Inertial positioning systems have been very popular in literature. A dead reckoning approach using foot-mounted inertial sensors has been developed to monitor pedestrians accurately using zero velocity corrections[35]. A slightly different solution uses a combination of inertial sensors and seed nodes, arranged in a static network, to achieve real-time indoor localisation[15]. A smartphone-based pedestrian tracking system has also been proposed in indoor corridor environments with corner detection to correct error drifts[28]. Microsoft also recently developed a reliable step detection technique for indoor localisation[17] using dynamic time warping (DTW). DTW is an efficient way to measure the similarity between two waveforms. Over 10,000 real step data points were observed offline to define the characteristic of a 'real' step. A DTW validation algorithm was then applied to the incoming accelerometer data to see whether it formed a similar waveform to a 'real' step.

There are also several pedometer applications available on Android such as Accupedo[21] and Runtastic[22] but since we do not have access to their algorithms, we cannot reproduce the same results. However, we did find an open source pedometer project[3] which calculated distance from the user's step length but their implementation was neither efficient nor accurate.

Signal processing is the underlying principle behind any pedometer algorithm. Data received from the accelerometer forms a signal which would be needs in real-time to accurately detect user movements. This process initially involves noise filtering in order to cancel out any random fluctuations that may affect processing later on. Refer to section 2.7 for further details on digital filters. The next step involves detecting peaks and valleys from the acceleration waveform that correspond to footsteps. Then heuristic

[4]/urlhttp://stackoverflow.com/questions/7829097/android-accelerometer-accuracy-inertial-navigation

constrains and cross-correlation validations need to be applied to eliminate erroneous detections.

To calculate the direction of movement, we need to also consider the orientation of the device. This can be calculated using geo-magnetic field sensors and gyroscopes. However, we need to also convert this orientation from the world's frame of reference to the site's frame of reference.

2.6.2 Ego-motion

An alternate solution to dead reckoning uses a vision concept known as ego-motion. It is used to estimate the three-dimensional motion relative to the static environment from a given sequence of images. Our application can use the smartphone camera to feed in the live images and process them in real-time to derive an estimate of the distance travelled.

There has been some interesting work published, in recent times, relating to the application of ego-motion in the field of navigation. A robust method for calculating the ego-motion of the vehicle relative to the road has been developed for the purpose of autonomous driving and assistance[34]. It also integrates other vision based algorithms for obstacle and lane detection. Ego-motion has also been employed in robotics. A technique that combines stereo ego-motion and a fixed orientation sensor has been proposed for long distance robot navigation[25]. The orientation sensor attempts to reduce the error growth to a linear complexity as the distance travelled by the robot increases. However there has not been a great amount of work in this topic using smartphone technology. The only published work that we came across proposed a self-contained navigation system for wheelchair users with the smartphone attached to the armrest[19]. For pedestrians it uses step detection instead of ego-motion to measure their movement.

Technically, to compute the ego-motion of the camera, we first estimate the two-dimensional motion taken from two consecutive image frames. This process is known as the optical flow. We can use this information to extract motion in the real-world coordinates. There are several methods to estimate optical flow amongst which the LucasKanade method[20] is widely used.

In our application, the smartphone camera will be used to take a series of images for feature tracking. This typically involves detecting all the strong corners in a given image. Then the optical flow will be applied to find these corners in the next frame. Usually the corner points do not remain in the same position and a new variable ϵ has to be introduce, which models all the points within a certain distance of the corner. The point with the lowest ϵ is then regarded as that corner in the second image. Template matching will then be applied to compare and calculate the relative displacement between

the set of corners in the two images. This information can be used to roughly estimate the distance travelled by the user.

2.7 Digital signal filters

Raw sensor data received from smartphone devices contain random variations caused by interference (noise). In order to retrieve the meaningful information, digital filters need to be applied to the signal.

A low-pass filter is usually applied to remove high frequencies from a signal. Similarly, a high-pass filter is used to remove low frequency signals by attenuating frequencies lower than a cut-off frequency. A band-pass filter combines a low-pass filter and a high-pass filter to pass signal frequencies within a given range.

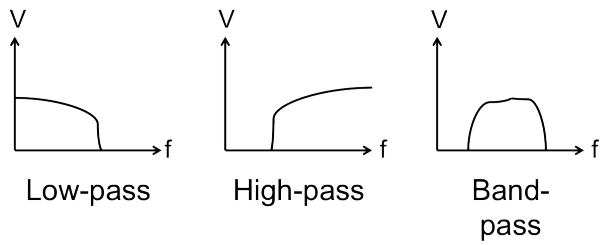


Figure 2.5: The image shows the three types of digital signal filters

Signal data can be analysed in the temporal domain to see the variation in signal amplitude with time. Alternatively, a signal can be represented in the frequency-domain to analyse all the frequencies that make up the signal. This can be useful for filtering certain frequencies of a signal. The transformation from the time-domain to the frequency-domain is typically obtained using the discrete Fourier transform (DFT). The fast Fourier transform (FFT) is an algorithm to compute the DFT and the inverse DFT.

Chapter 3

Position markers

We decided to develop our own custom markers with the purpose of obtaining the position of the user. Several of these markers would be placed on the floor and spread across the site. In particular, they would be situated at all the entrances and other points of interest such that application can easily identify them. Upon scanning, the application would start displaying directions from that position to their destination.

In this chapter, we start by discussing some of the other alternatives we considered before deciding to use custom markers and detail our reason as to why we did not choose any of these options. Then we proceed to describe the design of the marker specifying what data it encodes and how this data is represented. Then we start explaining our implementation for the smartphone scanner. Firstly, we explain how we detect the marker boundary using the Hough circle transform. Then we describe how the angular shift encoded in the marker helps us to calculate the orientation of the user. Finally, we explain the process of extracting the position data from the marker.

3.1 Alternate positioning systems

From our background research, we identified four smartphone-based solutions (triangulation, fingerprinting, barcodes and custom markers) that our application could have used to determine the position of the user, without requiring any expensive equipment.

A Wi-Fi based triangulation solution would have enabled our application to always keep track of the user's position without any form of user interaction, which follows for marker scanning techniques. However, Wi-Fi signals are susceptible to signal loss due to indoor obstructions, resulting in an imprecise reading. To overcome this problem, all the different types

of interference need to be considered along with the position of each access point. Since every site is structured differently, complex models for signal attenuation would need to be developed independently. [1] describes some further problems with triangulation.

The advantages of location fingerprinting are similar to triangulation. However, to achieve accurate results, fingerprinting requires a great amount of calibration work. This is a tedious process and would need to be replicated on every new site. In addition, several people have already raised privacy concerns for Wi-Fi access points[11].

At first, we strongly considered the option of placing barcodes around the site encoded with their respective positions. We even tested a few open-source barcode scanning libraries available on Android. However, we quickly realised that using an external library would affect its future integration with other features. Since Android only permits the use of the camera resource to one single view, we would have been unable to execute the obstacle detection mechanism simultaneously, unless we developed our own scanner. We could have also potentially extended the barcode scanning library by further studying and modifying a considerable amount of their codebase. The other major drawback with using barcodes was the inability to encode direction data needed to calibrate our application with the site's frame of reference. See section 3.2 for further information on this requirement.

Developing custom markers would give us complete control over the design of the marker, the scanning and its integration with the rest of the system. These custom markers would not only be designed to encode position data but also the direction. The only drawback would be that it takes a considerable amount of time to develop a bespoke scanner that gives highly accurate results. Nevertheless, we decided to take this approach as the benefits outweighed the disadvantages.

3.2 Marker design

For our design, we had to ensure that the marker encoded data relating to its position. We had to also ensure that the scanner was able to calculate the orientation of the user from the marker. Finally, the marker should be designed such that it could be scanned from any angle.

We achieved these criteria by encoding two pieces of information in our position markers:

1. A unique identifier (UID) - This UID will correspond to the coordinate position of the marker with respect to the site's Cartesian frame of reference. A map of UIDs to coordinate positions would be stored

locally or elsewhere. This gives us the additional flexibility of changing the position of the marker offline without the need of physically moving the marker. Upon scanning this feature, our application will be able to determine the position of the user.

2. A direction indicator - Upon scanning this feature, our application will be able to extract the angular variation of the marker from its normal position. This would allow the user to scan the marker from any direction. Furthermore, this angle will be also used to calibrate our application with the Cartesian grid representation of the site.

Colours are used to encode the UID. Currently our marker only supports three distinct colours - red, blue and green are used to represent the values 0, 1 and 2 respectively. We decided to choose these three colours because they are the furthest apart from each other in the HSV model. This will reduce erroneous detection as there will be lower chance of an overlap.

Each marker encodes six data digits with one extra digit for validation. Therefore, using the ternary numeral system, a number between 0 and 728 $(3^6 - 1)$ can be encoded by our marker. This allows for a total of 729 unique identifiers. The validation digit provides an extra level of correction and to a certain extent reduces incorrect detections. The first six data values detected are used to calculate the validation digit (v) as follows:

$$v = \left(\sum_{i=1}^{6} DataValue_i \right) \mod 3$$

If v is not equal to the extra validation digit detected from the marker, then the scanner discards that image frame and tries again.

The marker encodes the direction indicator using two parallel lines joined by a perpendicular line to form a rectangle. Figure 3.1 shows the structure of the marker with each colour section containing a number corresponding to the digit it represents in the UID's ternary representation.

As mentioned previously, the marker's direction indicator is also required to align the smartphone's orientation with respect to the site's frame of reference. From the built-in orientation sensors, our application can estimate the direction the device is facing. This direction does not necessarily represent the user's direction with respect to the site. For example, suppose the user is standing at the coordinate position (0,0) and the desired destination is at position (0,1). We can say that the destination is 'north' of the user with respect to the site's Cartesian grid. Let us assume that the orientation sensor tells the user that north is 180°away from the local 'north'. This would result in the application navigating the user in the opposite direction. To solve this

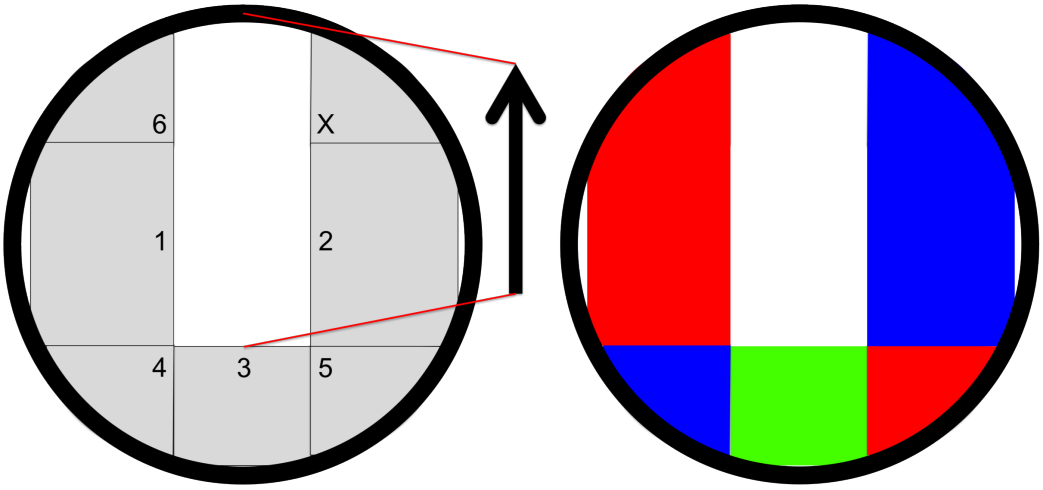


Figure 3.1: The left image shows the structure of the marker, and the right image shows a marker encoded with UID 48

problem, we use the marker's direction indicator to adjust our orientation sensors to take into account the difference in the measured angle. However, this is only useful if all the markers are placed such that their direction indicators are pointing to the local 'north'. Figure 3.2 shows how the markers need to be placed with respect to the site's Cartesian frame of reference.

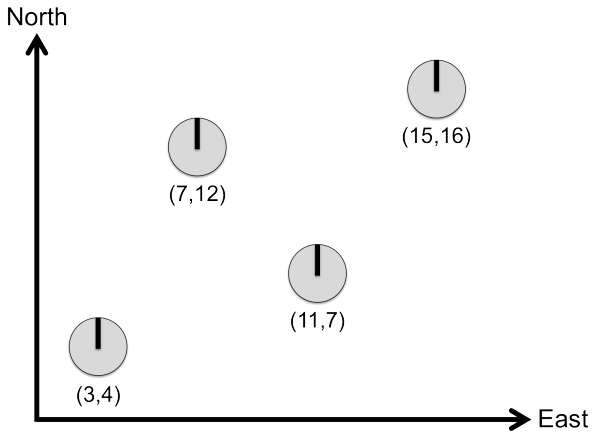


Figure 3.2: The image shows four markers placed such that their direction indicator is pointing to the local north

The markers could be of any reasonable size, but to be able to scan them while in a standing position they should be printed such that they occupy a complete A4 piece of paper.

3.3 Image gathering

The Android documentation recommended our client view to implement the *SurfaceHolder.Callback* interface in order to receive information upon changes to the surface. This allowed us to set up our camera configurations on surface creation and subsequently display a live preview of the camera data. The next step was to obtain the data corresponding to the image frames for analysis. The *Camera.PreviewCallback* callback interface was designed specifically for delivering copies of preview frames in bytes to the client. On every callback, we performed analysis on the preview frame returned and used a callback buffer to prevent overwriting incomplete image processing operations. For development, we did not initially use the preview callback interface. Instead, we took individual pictures of the markers to test our algorithm, and only implemented the callback once our algorithm achieved real-time performance.

The default camera resolution for modern smartphones is significantly high for an application to achieve real-time image processing. Therefore, we had to decrease the resolution of the preview frames prior to processing while still maintaining a certain level of quality. We decided upon a resolution of 640 x 480 as it provided a good compromise between performance and image quality. We also ensured that if a smartphone camera did not support this resolution, our application would select the one closest to it.

3.4 Circle detection

The first step was to detect the marker boundaries from the given image. In our background, we described Hough circles as a vision technique to identify the centre of the circle. The OpenCV library provided us with a function to find circles using the Hough transform. This function is capable of calculating the centre coordinates and the radius of all the circles from a given image, providing all the relevant parameters are set appropriately. However, the documentation suggests that the function generally returns accurate measurements for the centre of a circle but not the radius. This is one of the main reasons our markers were designed to not be dependent on the detected radius with the data being encoded surrounding the centre of the circle. Thus, our data extraction algorithm involves expanding out from the centre point.

Parameter	Description
image	Grayscale input image
circles	Output array containing the centre coordinates and radii of all the detected circles
method	Method used for detecting circles, i.e. using Hough transforms
dp	Inverse ratio of the accumulator resolution to the image resolution
minDist	The minimum distance between the centres of two circles
param1	The upper Canny threshold
param2	The accumulator threshold
minRadius	The minimum radius of a circle
maxRadius	The maximum radius of a circle

Table 3.1: OpenCV specification for the Hough circle transform

```
void HoughCircles(InputArray image, OutputArray circles
    , int method, double dp, double minDist, double
    param1, double param2, int minRadius, int maxRadius)
```

Prior to the Hough circle detection, the input image had to be converted to grayscale to detect the gradient change and filtered to remove noise. The image data from the smartphone camera is received in bytes. This is first converted from bytes to the YUV format and then to grayscale. OpenCV contains several image processing functions, allowing the conversion of an image between different colour models. OpenCV also provides a function to apply the Gaussian blur to an image with a specified window size. Figure 3.3 shows the process of marker detection from the original image to the detection of the centre of the circle.

```
void GaussianBlur(InputArray src, OutputArray dst, Size
    ksize, double sigmaX, double sigmaY, int borderType
    )
```

Parameter	Description
src	Input image
dst	Output image
ksize	Gaussian kernel size
sigmaX	Standard deviation in the horizontal direction for the Gaussian kernel
sigmaY	Standard deviation in the vertical direction for the Gaussian kernel
borderType	Method for pixel extrapolation

Table 3.2: OpenCV specification for the Gaussian blur

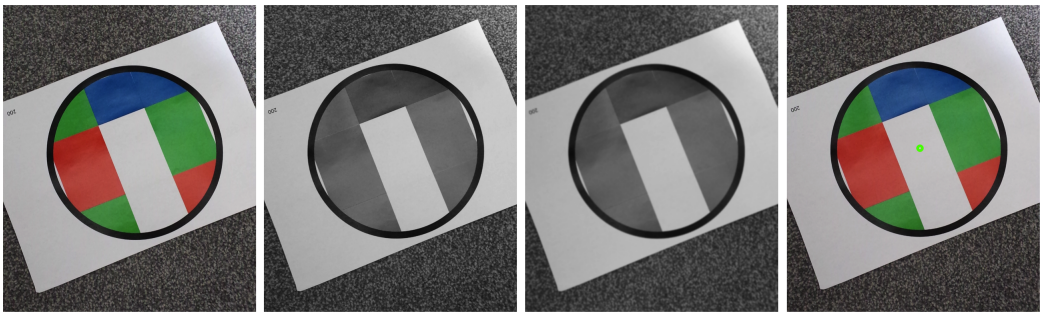


Figure 3.3: The image shows the process of circle detection - original input image, grayscaled, Gaussian blurred and centre detection

3.5 Angular shift

The angular shift is the difference in angle from the straight orientation of the screen to the direction of the marker as shown in figure 3.4. This angle is required to rotate the marker image to its natural orientation for data extraction. Later, we discuss how this angle is also used align the user's orientation with the site's frame of reference.

As stated previously, the direction indicator is encoded in the marker using two parallel lines joined by a perpendicular line drawn around the centre. We first obtain the equation of the two parallel lines and use the perpendicular line to determine the direction of the marker. The Hough line transform can be applied to detect all the lines in the marker. OpenCV provides us with an efficient implementation of the Hough line transform using probabilistic inference. The function returns a vector of all the detected line segments in the given input image. We have given the specification of this function in section 4.1.

Using the Java Line2D API, we calculated the shortest distance from the

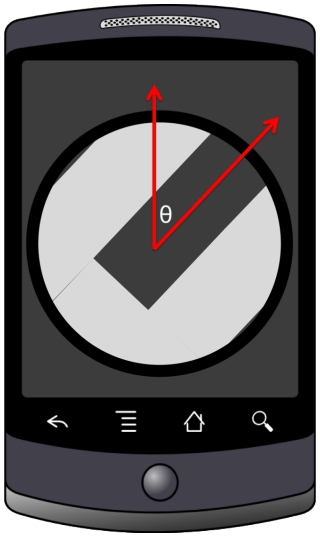


Figure 3.4: The image shows the angular shift θ

centre point of the marker to all the detected line segments. The closest line from the centre is selected as the first direction indicator. Then, we search for the second parallel line by checking the gradient of the line against all the detected line segments that are a certain distance away from the first direction indicator. If either of these indicators are not found, we abandon further processing and wait for the next preview frame. Otherwise, we continue to search for the perpendicular line allowing us to distinguish between the two possible direction scenarios (positive or negative angular shift). We then combine this information with the line equations of the direction indicators to calculate the angular shift. To reduce erroneous detections, we applied two further heuristics to our algorithm: (1) Minimum line length; and (2) Maximum distance from the centre.

Prior to the Hough line transform, the input image has to be first converted to a binary image with all the boundaries highlighted where a strong gradient change occurs. To achieve this, Canny edge detection can be performed on the filtered grayscale image obtained from the previous circle detection step. OpenCV provides us with a Canny function taking in parameters that define the upper and lower thresholds. We have given the specification of this function in section 4.1.

The next step involves rotating the image anticlockwise by the angular shift to obtain the natural orientation of the marker for data extraction. We first calculate the affine matrix for two-dimensional transformations using OpenCV's *getRotationMatrix2D()*. Then, we use this matrix to actually perform the transformation on the image using *warpAffine()*. Figure 3.5

illustrates the entire process of angular shift transformation.

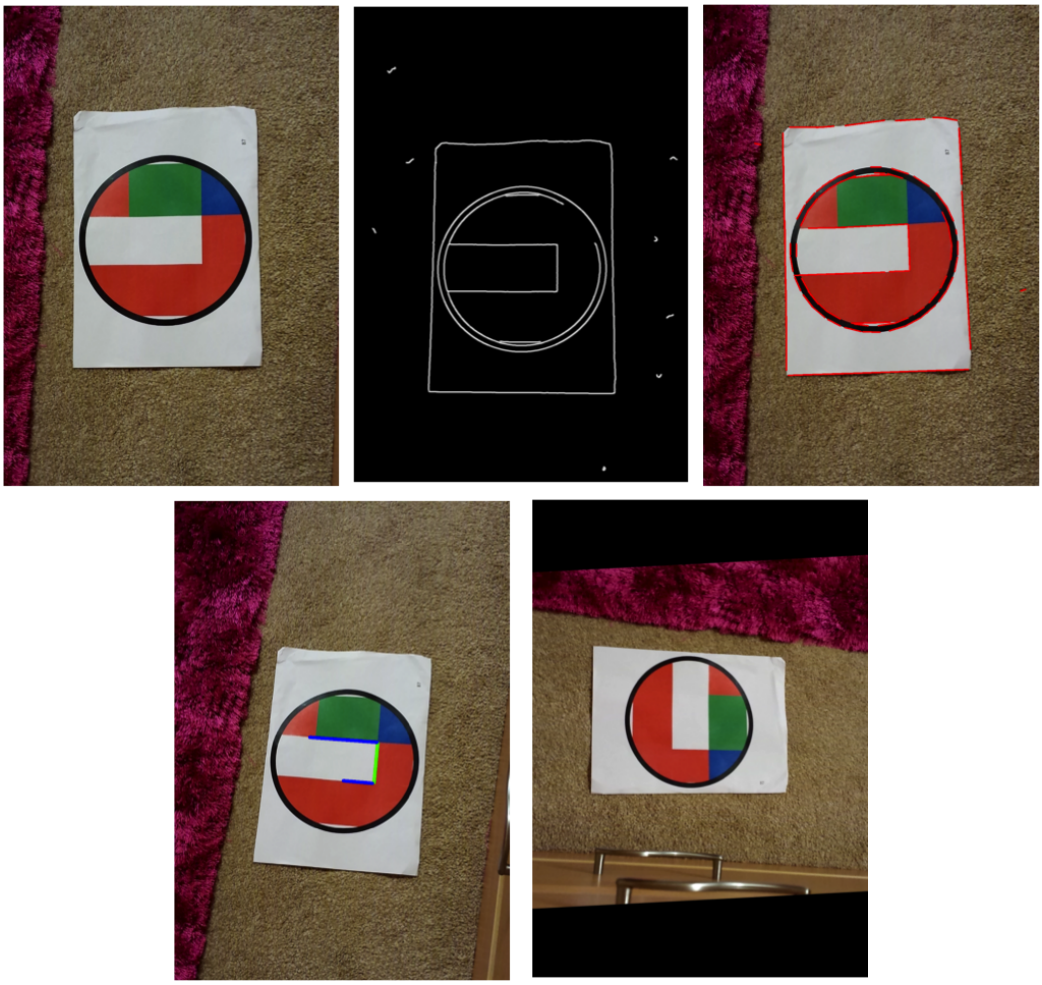


Figure 3.5: The image shows the process of angular shift transformation - original input image, Canny edge detection, line detection, direction indicator detection and rotation

```
Mat getRotationMatrix2D(Point2f center, double angle,
    double scale)
```

Parameter	Description
center	Center of rotation
angle	Angle of rotation
scale	Scale factor
return	Output 2x3 affine matrix

Table 3.3: OpenCV specification for the rotation matrix

Parameter	Description
src	Input image
dst	Output image
M	Affine transformation matrix
dsize	Size of output image
flags	Method of interpolation

Table 3.4: OpenCV specification for affine transformation

```
void warpAffine(InputArray src, OutputArray dst,
   InputArray M, Size dsize, int flags)
```

3.6 Data extraction

Once the marker image is aligned with the screen's portrait orientation, we can begin decoding the colours to extract the UID. This UID will be checked against a map of UIDs to two-dimensional coordinates in order to retrieve the marker's position. For testing, we stored this map locally. In the future, we plan to decouple this feature by storing this data on an online database.

The first step of the decoding process involves calculating the boundaries of the colour regions. We had previously detected all the edges during the angular shift transformation. We reuse this information to estimate the seven required borders highlighted in figure 3.6.

The position of these boundaries can be used to accurately determine the colours encoded in all the seven regions. Prior to this, we converted the rotated image to HSV to enable the comparison of colour using the hue component. Note that OpenCV defines the hue component scale from 0°to 180°.

Figure 3.7 illustrates the path taken to obtain the hue components from

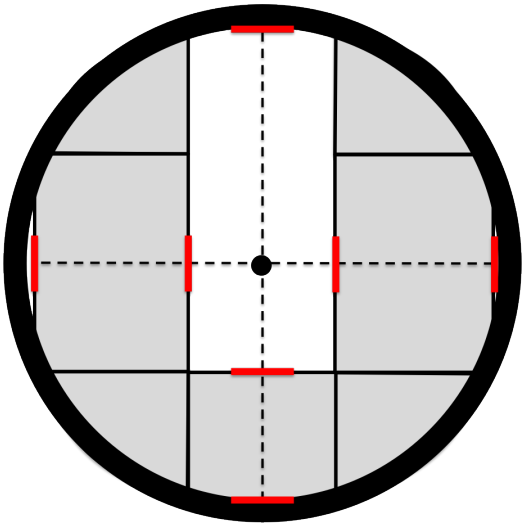


Figure 3.6: The image shows the seven border positions used to calculate the colour regions

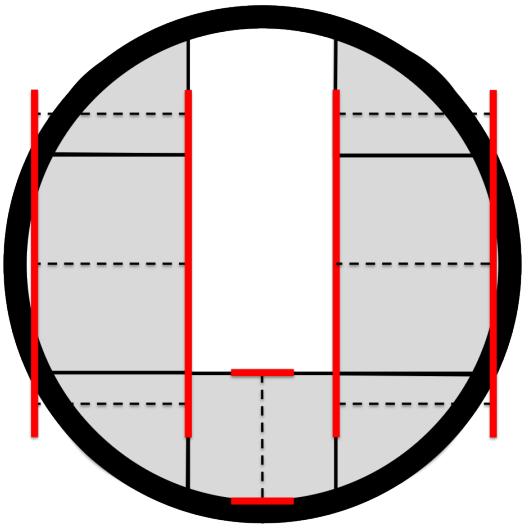


Figure 3.7: The image shows the path to calculate the value encoded by each colour region

all the seven regions. Our algorithm accumulates the hue values for each pixel in the specified path. Then the modal colour is calculated by counting the number of pixels recorded in each colour range. This process is repeated for all the seven regions. At present, we only consider the modal colour values if they are either red, blue or green. We use the colour encodings from the six data regions to calculate the UID and the seventh data region for validation.

Chapter 4

Obstacle detection

The purpose of the obstacle detector was to avoid giving directions to the user that led to an immediate obstacle. The only plausible solution to detect obstacles from a smartphone was to use the camera to roughly detect the object boundaries.

In this chapter, we discuss the process of boundary detection using the Hough line transform and the Canny edge detector. We also explain how we achieved real-time performance using OpenCV libraries. The last section describes how this boundary information is used to identify obstacles surrounding the user.

4.1 Boundary detection

Line detection was previously employed to detect the direction indicator on the position marker. Here, we apply the same technique with a different purpose. We looked at detecting object boundaries such as the one between a floor and a wall/shelf from a given image. Once again, we used Hough line transforms using the OpenCV *HoughLineP* function to retrieve all the line segments in an image. However, in this case, we had to consider the time performance of this function in order to achieve real-time boundary detection. The Hough line transform is a process intensive operation and to achieve faster results we had to sacrifice the precision with which lines were detected. In particular, we increased the angle resolution of the accumulator from 1°to 3°, which meant that lines with a very fine angle were not detected. This was acceptable as losing some of the accuracy of the boundary edges was not a major concern.

Parameter	Description
image	Binary input image
lines	Output array containing the two coordinate points of the detected line segments
rho	The distance resolution, usually 1 pixel for preciseness
theta	The angle resolution
threshold	Minimum number of intersections required for line detection
minLineLength	The minimum length of a line
maxLineGap	The maximum distance between two points belonging to the same line

Table 4.1: OpenCV specification for the Hough line transform

```
void HoughLinesP(InputArray image, OutputArray lines,
    double rho, double theta, int threshold, double
    minLineLength, double maxLineGap)
```

The process of retrieving the camera preview frames was exactly the same as for scanning position markers (section 3.3). In fact, we used the same class as before to also incorporate boundary detection. As a result, we were able to simultaneously compute results for both these tasks. Once again, prior to the Hough line transform, we applied Gaussian blur and Canny edge detection to these preview frames. While the function call to *GaussianBlur* remained unchanged, the thresholds for the Canny edge detector were modified such that only the strong edges were detected. Figure 4.1 summarises the entire process of boundary detection.

```
void Canny(InputArray image, OutputArray edges, double
    threshold1, double threshold2)
```

4.2 Obstacle detection

The boundary detection enabled us to approximately plot the obstruction boundaries surrounding the user. The next step involved examining these

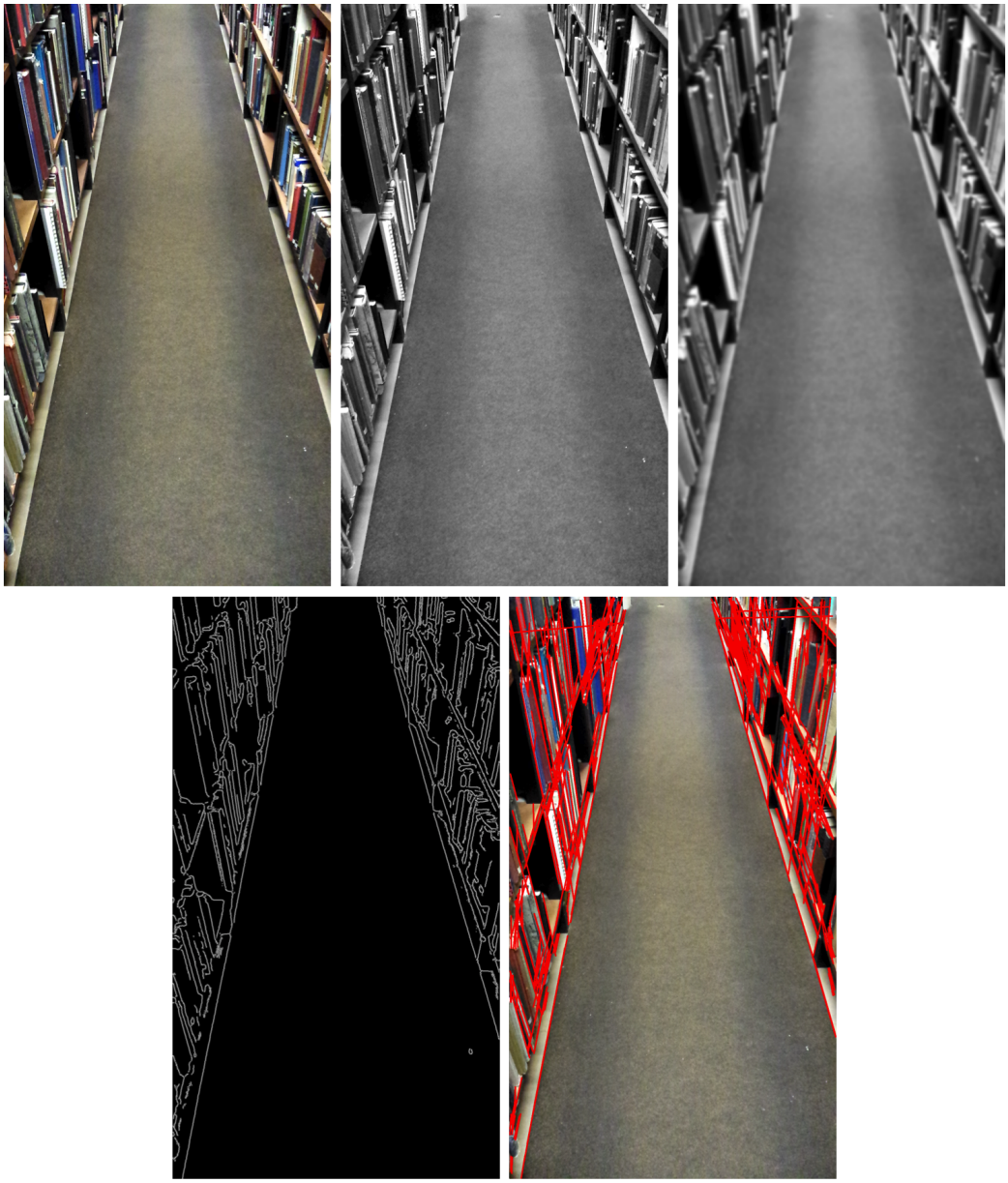


Figure 4.1: The image shows the process of boundary detection - original input image, grayscaled, Gaussian blurred, Canny edge detection and Hough line transform

boundaries to see if there was an edge on the left, right and in front of the user. To achieve this, we used the Java Line2D API to check for line segment intersections.

We first considered detecting obstacles straight ahead of the user. We

Parameter	Description
image	Grayscale input image
edges	Binary output with the edges highlighted
threshold1	Lower threshold used for Canny edge detection
threshold2	Upper threshold used for Canny edge detection

Table 4.2: OpenCV specification for the Canny edge detection

noticed that the object boundaries formed by the obstacles in front of the user were almost always horizontal. Therefore, we searched through all the detected boundary lines and only retained those that had an angle of 180°± 25°or 0°± 25°. Figure 4.2 illustrates this process. We then used an accumulator to count the number of intersections between these lines and vertical lines. The vertical lines signify the user walking straight ahead. If this number of intersection is quite high, it would indicate that there is an obstacle in front of the user, assuming that he is holding the phone in the direction of movement.

For detecting obstacles on the left and the right side of the user, we used a similar approach. We noticed that the object boundaries on the side were slightly slanted and very close to forming a vertical line. Therefore, for detecting obstacles on the left hand side, we decided to only retain boundary lines that had an angle of 75°± 25°or 255°± 25°. For the right side, we only looked at lines with an angle of 105°± 25°or 285°± 25°. Figure 4.3 illustrates this process. Then, we checked for the number of line intersections horizontally, representing the user's side movements. However, for detecting obstacles on the left, we only checked the left half portion of the image and similarly the right half for detecting obstacles on the right.

Our current implementation of obstacle detection has two key limitations. One is that the user has to hold the smartphone with a slight tilt (35°± 20°) such that the back camera is always facing the floor, as shown in figure 4.4. The reason is that if the phone is held perpendicular to the ground it is not possible to determine the depth of an obstacle just by looking at an image. Therefore, we would not be able to conclude whether an obstacle lies right in front of the user or further away. By forcing the user to hold the phone in the desired position, we can almost guarantee that an obstacle detected is immediately ahead or to the side of the user.

The second limitation is that the floor should not contain any patterns. This is mainly due the fact that our algorithm falsely interprets the patterns

Figure 4.2: The images show the preservation of horizontal lines for detecting obstacles ahead of the user in two different scenarios

as obstacles. Some of the free space detectors in literature also introduce this restriction[18]. The process of distinguishing complex floor texture from actual obstacles would have been a time-consuming endeavour.

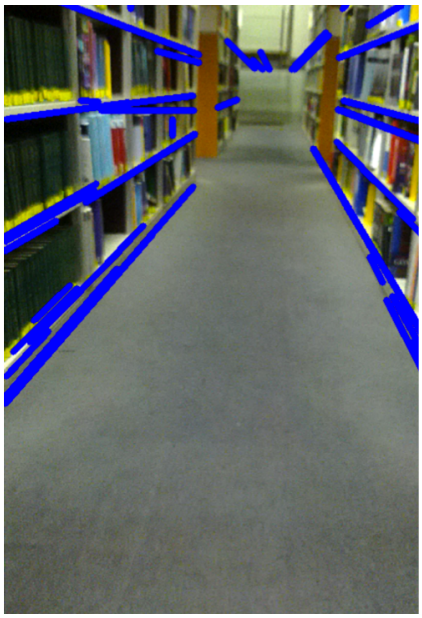


Figure 4.3: The image shows the preservation of side lines for detecting obstacles left and right of the user

Figure 4.4: The image shows the correct way to hold the phone for obstacle detection

Chapter 5

Dead reckoning

In the absence of a nearby position marker, our application still needs to keep track of the user's position and orientation. In this chapter, we begin by describing our initial foray into calculating the ego-motion of the device to solve this problem and explain our reasoning behind choosing the inertial approach instead. We give a brief overview on the Android sensor API that we use to receive the inertial measurements. Then we explain our approach to filtering noise from the sensor data. The penultimate section describes how we developed our step detection algorithm while the final section explains the process of combining the detected footsteps along with the orientation sensor to estimate the new position of the user.

5.1 Initial approach

Our initial approach was to calculate the ego-motion of the smartphone camera, and subsequently use that to estimate the distance travelled by the user. To calculate the ego-motion, we first had to measure the optical flow. This required our application to track features from two consecutive images. Once again, we had to convert the incoming preview frames into grayscale and apply Gaussian blur, using the respective OpenCV functions previously mentioned. We then used OpenCV's highly optimised image processing library to detect all the strong corners from a given image. The *goodFeaturesToTrack()* function uses the Shi-Tomasi corner detector algorithm to highlight all the key features.

Parameter	Description
image	Grayscale input image
corners	An output array containing the coordinate positions of all the corners
maxCorners	Maximum limit to the number of strong corners detected
qualityLevel	Defines the minimum quality level for each potential corner
minDistance	Minimum distance between two corners
mask	Specifies a particular region from the image to extract features from
blockSize	The size used for calculating the covariance matrix of derivatives over the neighbourhood of each pixel

Table 5.1: OpenCV specification for feature tracking

```
goodFeaturesToTrack(InputArray image, OutputArray
    corners, int maxCorners, double qualityLevel, double
     minDistance, InputArray mask, int blockSize)
```

From the corners detected in the first image, we can use optical flow to find the new positions of these corners in the second image. This would give us an estimate of the feature displacement for that time frame. The OpenCV implementation for the optical flow uses the iterative Lucas-Kanade algorithm with pyramids. Figure 5.1 shows the results we obtained by moving the camera slowly to the left.

```
void calcOpticalFlowPyrLK(InputArray prevImg,
    InputArray nextImg, InputArray prevPts,
    InputOutputArray nextPts, OutputArray status,
    OutputArray err)
```

At this stage, we realised that the feature tracking and the optical flow algorithms required a large amount of computation time to achieve the desired results. This would prevent our application to perform real-time operations as well as quickly drain the smartphone's battery. A potential solution

Figure 5.1: The first two images show the movement of the camera from left to right and the third image shows the feature displacement calculated by the optical flow

Parameter	**Description**
prevImg	First input image
nextImg	Second input image
prevPts	An array containing the coordinate positions of all the corners detected in the first image
nextPts	An output array containing the new coordinate positions of the corners detected from the first image
status	An output status array
err	An output error array

Table 5.2: OpenCV specification for calculating the optical flow

considered reducing the resolution of the preview frame further. However, this had a direct impact on the quality of the features detected. Furthermore, we realised that ego-motion would only be successful if the environment contained a large amount of distinct features. This contradicted our earlier assumption for obstacle detection, which expected the user to hold the smartphone camera facing towards a plain carpet.

We also observed that the optical flow algorithm did not function as expected when there was a great displacement between two consecutive image frames. This problem was made apparent while walking at an average pace. To further complicate the problem, the preview frames were slightly blurred due to the shaking camera movements. At this point, it was quite obvious that a large amount of time would have to be spent to achieve accurate

distance estimations using an ego-motion based approach. Therefore, we decided to explore the inertial method of dead reckoning. Since a pedometer algorithm had been previously achieved by many, we were more confident with this approach.

5.2 Sensors

Nowadays, all smartphones are equipped with a wide variety of sensors. For our application, we only considered the accelerometer, the geo-magnetic field and the gyroscope sensors. The accelerometer enables us to monitor motion relative to the world's frame of reference. The remaining two sensors can be combined with the accelerometer to give an accurate reading of the device's orientation also relative to the world's frame of reference. Our aim is to make use of these sensors to track the user's position relative to the site's frame of reference.

The Android API for sensor events is well documented making it simple to integrate with our application. Figure 5.2 from the Android documentation[1] illustrates the coordinate system used by the sensors. To receive data from a particular sensor, our client registers itself to listen to the changes from that sensor type using a *SensorListener* interface. Our application registers with two sensor types: linear acceleration and rotation vector.

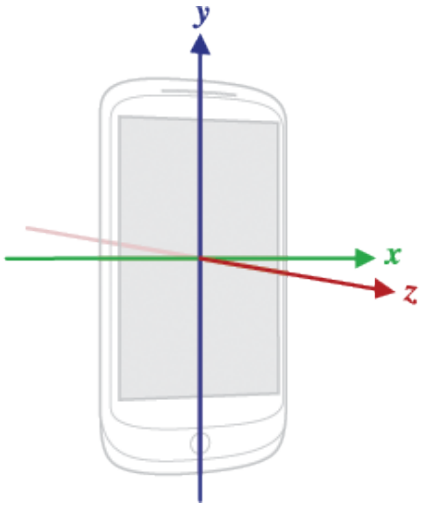


Figure 5.2: The image shows the coordinate system used by the Android sensor API

[1] `http://developer.android.com/reference/android/hardware/SensorEvent.html`

5.2.1 Linear acceleration

This sensor type is used to measure the acceleration on the device on all the three axes excluding the force of gravity. As stated previously in the background chapter, we cannot apply double integration on the linear acceleration data to obtain distance as the accuracy deteriorates significantly with increasing distance. Instead, we can use this data to detect footsteps. The accelerometer sensor returns the acceleration of the device (A_d) influenced by the force of gravity.

$$A_d = -g - \sum F/m$$

g is the force of gravity which is -9.81 m/s^2 when on a flat surface
F is the force applied to the sensor
m is the mass of the device

Therefore, to calculate the linear acceleration we subtract the force of gravity from the acceleration data A_d.

$$Linearacceleration = acceleration - gravity$$

This sensor type returns three values corresponding to the linear acceleration on each axes. However, we are only concerned with the value from one of these axes. Our assumption for free-space detection expects the user to always hold the phone in a specific position. Therefore, to measure the movement of the user, we only need to consider the acceleration force in that direction. As you can see from figure 5.2 and figure 4.4, we can approximate the acceleration force from the user's movement based on the orientation of the device and the magnitude of the acceleration force on the y-axis.

5.2.2 Rotation vector

This sensor type uses sensor fusion to combine data from the accelerometer, geo-magnetic field and gyroscope to provide an accurate representation of the devices' orientation. When a step is detected, this orientation data is used to map the direction of the user's movements, providing the user always walks in the direction of the smartphone's y-axis.

The data obtained from the rotation vector represents the orientation of the device as a combination of an angle and an axis, shown in figure 5.3 taken from the Android documentation[2]. To calculate each of the device's

[2] `http://developer.android.com/reference/android/hardware/SensorManager.html`

individual orientation angle for each axis, we need to first calculate the rotation matrix from the rotation vector. The Android sensor API provides us with the function *getRotationMatrixFromVector()*, and also *getOrientation()* to calculate these angles.

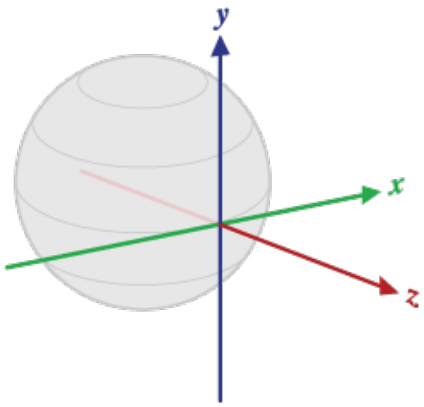


Figure 5.3: The image shows the coordinate system used by the rotation vector

The coordinate system used by *getOrientation()*, as shown in figure 5.4 taken from the Android documentation, is different from the coordinate system defined for the rotation matrix.

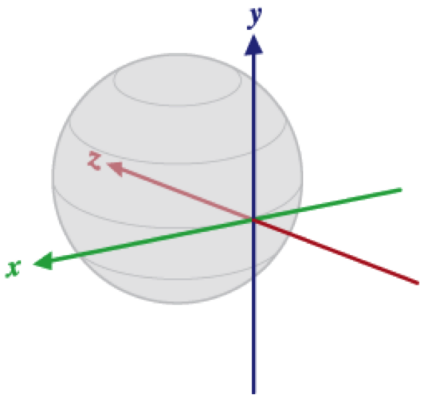


Figure 5.4: The image shows the coordinate system used by the orientation vector

This function returns an array of size three containing the angle of rotation, in radians, around the z-axis (azimuth), the x-axis (pitch), and the y-axis (roll). Here we are only concerned with the azimuth, which represents the horizontal angle formed by the difference in the y-axis from the magnetic

north. The azimuth lies between 0°and 359°, with 0 corresponding to the north. Therefore, this value can be used to monitor the direction of the user's movement whenever a user takes a step. However, to be useful indoors, this direction has to be first calibrated with the site's frame of reference. This can be done by scanning any marker on the site, providing the marker's direction indicator is parallel to the site's y-axis. In other words, the marker points to the north of the site.

5.3 Signal filtering

The sensor values received from both the sensor types contain a lot of noise. Thus, it is not possible to perform any kind of analysis on this incoming information. The signal is mainly affected by high-frequency signals and also some low frequency signals.

Initially, we decided to use the discrete Fourier transform (DFT) to convert the input time-domain data to the frequency-domain. Thereafter, we would use the frequencies to process the signal to filter the data. We used the fast Fourier transform to compute the DFT using the Cooley-Tukey algorithm from the Columbia university[3].

We passed in the amplitude at each time step and the FFT algorithm returned half real and half imaginary values. Each real value represented the power of every sampled frequency. For filtering, we decided to zero out certain high-frequencies in order to remove the noise. This is known as the brick-wall frequency-domain filter. We then reconstructed the signal from the frequency-domain to the time-domain using the inverse fast Fourier transform by calling FFT but with the real and imaginary values swapped around. Although the theory was technically correct, we did not obtain the expected result. We soon realised that this was not the correct way to achieve noise filtering. Using this filtering technique, we were essentially convolving the time-domain data with a sinc function, shown in figure 5.5 taken from MathWorks[4]. This effect is termed as ringing caused by the Gibbs phenomena and is not desirable.

We then decided to look at more practical ways of filtering noise. We used a discrete-time low-pass filter to remove the high-frequency signals and a high-pass filter to remove the low-frequency signals.

[3] `http://www.ee.columbia.edu/~ronw/code/MEAPsoft/doc/html/FFT_8java-source.html`

[4] `http://www.mathworks.co.uk`

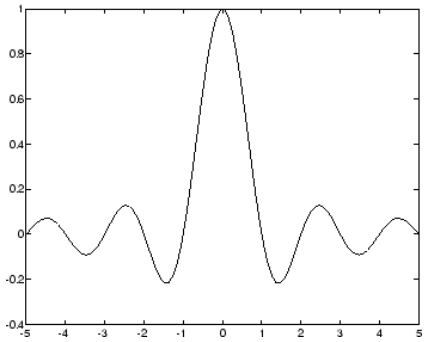


Figure 5.5: The image shows the plot of a sinc function

```
for (int i = 0; i < in.length; i++){
  out[i] = out[i] + ALPHA * (in[i] - out[i]);
}
```

```
for (int i = 0; i < in.length; i++){
  out[i] = BETA * (out[i] + in[i] - oldIn[i]);
}
```

The value of ALPHA was used for calibrating the low pass filter while the value of BETA was used for the high pass filter, such that the relevant properties of the signal were preserved. We applied these filters to data received from both the sensor types. Figure 5.6 shows the attenuation of the high frequency components from the original signal.

5.4 Footstep detection

Although technically the double integration of acceleration should yield distance, our background research suggested that the poor accuracy of the accelerometer produces a great margin of error. As a result, we decided to estimate displacement by developing a pedometer algorithm. Firstly, we converted the raw sensor data into a signal in the time-domain using a graph API in Android[5]. Then, we proceeded to analyse the properties of this signal by observing several footsteps from people with various different stride

[5] `http://androidplot.com/`

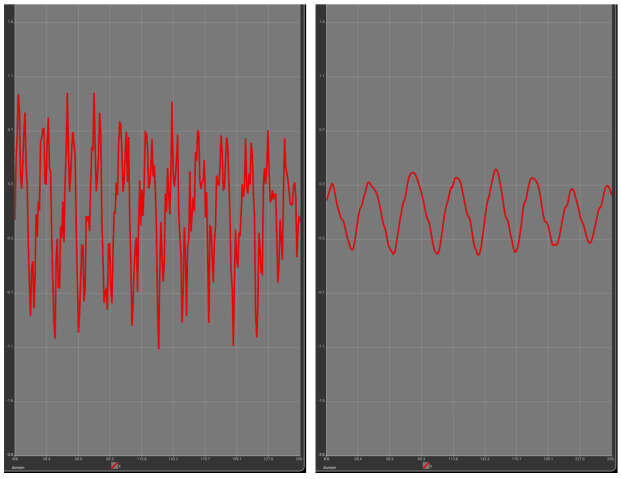


Figure 5.6: The image shows the accelerometer data before and after filtering

lengths. This helped us establish and model the change in acceleration of the smartphone when a user is walking. Figure 5.7 shows the characteristics of an acceleration waveform used for step detection.

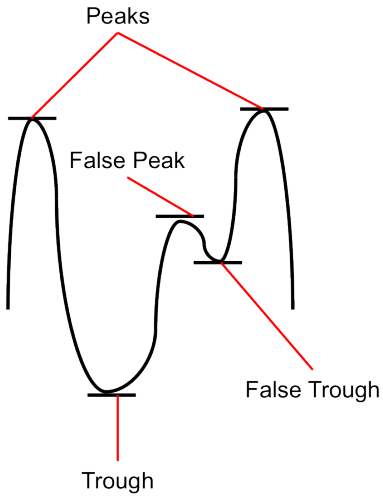


Figure 5.7: The image shows the characteristics of an acceleration waveform used for step detection

The basis of our pedometer algorithm relied upon detecting the peaks and troughs from the time-domain waveform. Essentially, when a peak was detected, we started searching for the next trough. Similarly, after a trough was detected, we started searching for the next peak. This sinusoidal at-

tribute of the signal along with other amplitude characteristics constituted an average footstep. Taking the first derivative of the accelerometer data allowed us to calculate and distinguish between a peak and a trough. At times, when a second peak was detected while looking for a trough, we discarded the first peak and used this new peak as the starting reference for identifying a footstep.

We applied three further simple heuristics to reduce false positives.

1. Maximum time duration for one step - If a step took longer than 1.5 seconds, we would not consider it.

2. Time difference between a peak and a trough - We observed that a trough occurred a certain time distance (0.25 - 0.8 seconds) between the two peaks. Then, we simply checked whether the trough fell within this range.

3. Minimum number of initial steps - The user has to take 2-3 initial steps before the algorithm starts detecting individual footsteps.

5.5 Distance and direction mapping

The last step combined the pedometer algorithm, the user's stride length and the orientation of the phone to estimate the new position of the user as follows:

$$x_{new} = x_{old} + cos(orientation) \times STRIDE\_LENGTH$$
$$y_{new} = y_{old} + sin(orientation) \times STRIDE\_LENGTH$$

In order for the new position to be calculated with respect to the site's frame of reference, the orientation, in particular the azimuth, needs to be subtracted by the angular shift obtained from scanning a position marker. Refer to section 6.2 for more details on how these two components are integrated.

Chapter 6

Integration of navigation system

The final stage of development involved combining the data obtained from position markers, dead reckoning and the obstacle detector to calculate the appropriate direction that would gradually lead the user to their destination. This is the final step needed to complete our indoor navigation system.

In this chapter, we briefly explain how the location where our system will be employed should be setup. Then, we discuss how we integrated the three individual components together to display the appropriate directions. Finally, we give a high level overview of our entire system through a class diagram and a sequence diagram.

6.1 Location setup

The indoor site should be viewed as a two-dimensional Cartesian grid. We recommend a grid spacing of 1 metre or less. This allows for a more accurate representation of the positions of all the points of interest. The grid should have a well-defined boundaries with a fixed origin representing (0,0). All the position coordinates should be positive. Currently, our solution only supports navigation on a single floor.

Markers should be placed on the floor at entrances and other key intersections. Although to achieve more accurate results, they could be placed more frequently around the site. It is also important to map their encoded UIDs with their corresponding grid coordinates. Furthermore, all the markers should be placed such that their direction indicator points towards the north of the site (y-axis) as previously illustrated in figure 3.2.

The exact position coordinates of all the items or points of interest need to

be recorded offline. Essentially, these are all the possible indoor destinations that our application navigates the user to. For our testing, we stored this data locally as there were not many destination points. However, to decouple this task and make it scalable, this data needs to be stored in a file or database. An internet connection would be required to fetch this data when necessary.

6.2 Final integration

We created a central module solely responsible for listening to the changes of all the individual components. Whenever one of these components acquired a new piece of information, it would notify this central module. This module contained the position coordinates for the user's current estimated location and the location of the destination. It would use this information along with the knowledge of the obstacle detector to calculate the direction to display. We proceeded with our implementation by integrating each component one-by-one. Figure 6.1 illustrates a high level overview on how the main components integrate to make the final indoor navigation system.

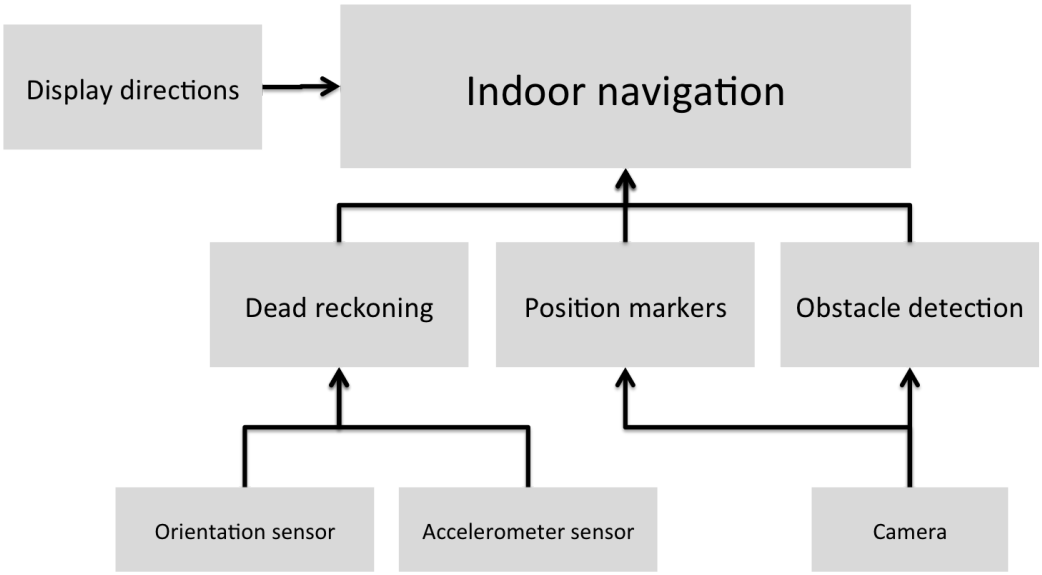


Figure 6.1: The image shows how all the main components integrate to make the final indoor navigation system

Position markers

On every successful scan of a position marker, the encoded UID is retrieved and checked for the corresponding position coordinate of the marker. When

this is known, the scanner would notify the central module about this new position. Then, the central module would immediately update its current estimated position to the one just received and recalculate the directions to display.

The markers also provide the central module with the angular shift to the site's local north. As previously mentioned, this offset is used to calibrate the device's orientation with respect to the site's frame of reference. This is important for correctly tracking a user's movements through dead reckoning.

Dead reckoning

There were two aspects to integrating this component. One was the orientation (azimuth), which gave the direction the user was walking towards and the other aspect was the pedometer algorithm. This orientation was not only important for dead reckoning but also for adjusting the currently displayed direction. For example, if the direction currently displayed told the user to walk straight but instead he turned right, the direction would need to change to tell the user to turn left. Also note that when the sensor detected a change in the device's orientation, we would straightaway subtract the measurement with the angular shift offset retrieved from the position marker.

$$orientation = azimuth - angularshift$$

With regards to the pedometer, on every step detected, our central module would update the current position estimate appropriately based on the user's stride length. In addition, it would reassess this position with respect to the destination and display a new direction if necessary.

Obstacle detector

Integrating this component ensured that the direction displayed by our application was feasible. Whenever there was a change in the navigation direction displayed, we would check with the obstacle detector on whether there was an obstacle in that direction. If there were no obstacles, then the direction displayed would remain the same. However, if it did detect an obstacle in that direction, we would ensure that the application navigated the user around the obstacle. We achieved this by updating the direction hint to point 90°clockwise, i.e. to the right. While the user follows this direction, our application checks whether the obstacle (now on the left side) still exists. If it does not exist, the direction indicator is returned to its original angle. If it does, then the user would continue till it eventually disappears. In case there is another obstacle on the new direction path, then once again we repeat the

same step by further updating the direction hint to point 90°clockwise till the obstacle disappears. In the future, this algorithm can be improved to take into account the user's past movements and learn to not take a path with long obstacles.

Displaying navigation direction

On every position update or an orientation update, the navigation direction to display is recalculated. To keep the UI simple and easy to follow, the only possible direction hints our application displayed were to either go straight, left, right or backwards, as shown in figure 6.2. Therefore, it was important for the user to hold the device in the position shown. Otherwise, there was scope for misinterpretation.

Figure 6.2: The image shows the four navigation hints

To calculate the direction to display out of the four, we computed the inverse tangent angle based on the current position (c) and the destination position (d) and then rounded the value to the nearest 90°.

$$directionAngle = Round(\frac{\tan^{-1}(\frac{d_y - c_y}{d_x - c_x})}{90^\circ}) \times 90^\circ$$

If the direction angle is computed as 0°, the navigation direction would point straight ahead. Subsequently, the direction would move clockwise as the angle increases.

To avoid constant fluctuations due to rounding when at a diagonal position, we ensure that the direction remains the same till the user either reaches the horizontal or the vertical axes formed by the destination assuming no obstacles are encountered. Figure 6.3 illustrates this problem.

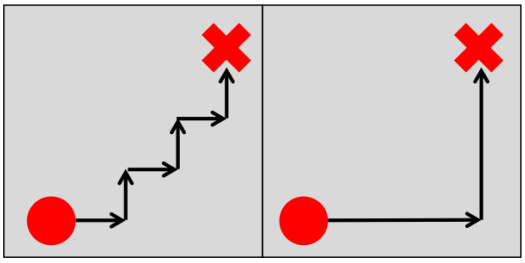


Figure 6.3: The image on the left shows the diagonal fluctuations in the navigation direction and the image on the right shows how we fix this problem

6.3 System architecture

Figure 6.4 gives a high level class diagram of our implementation followed by a sequence diagram in figure 6.5, illustrating the interaction between different processes.

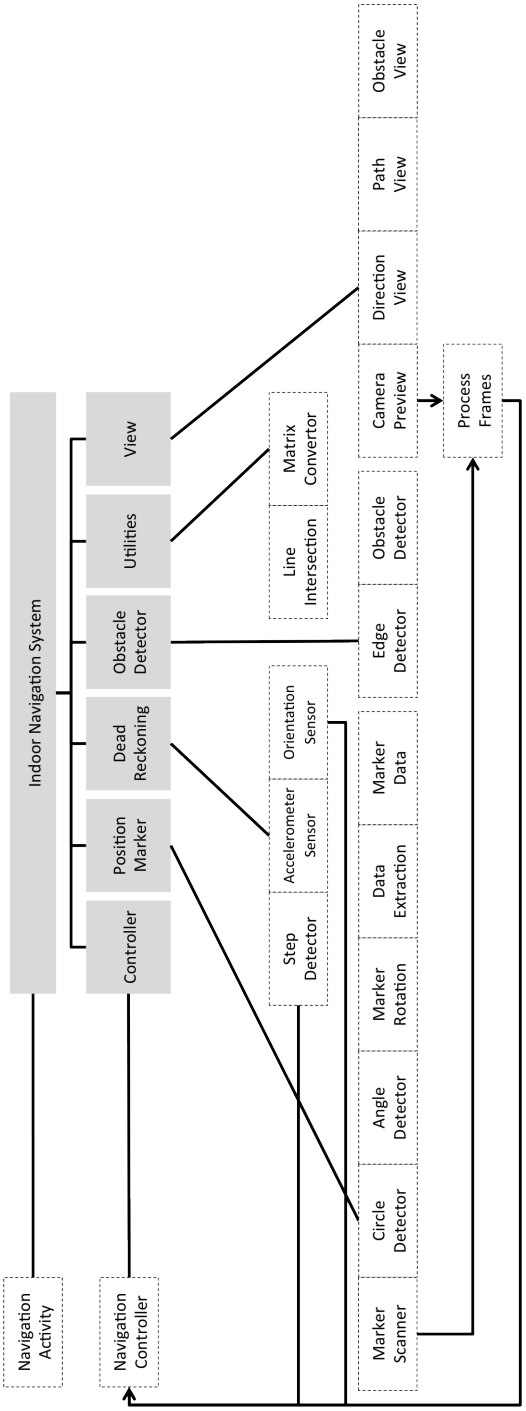


Figure 6.4: The image shows the class diagram of our proposed navigation system with the packages highlighted in grey and the classes in white

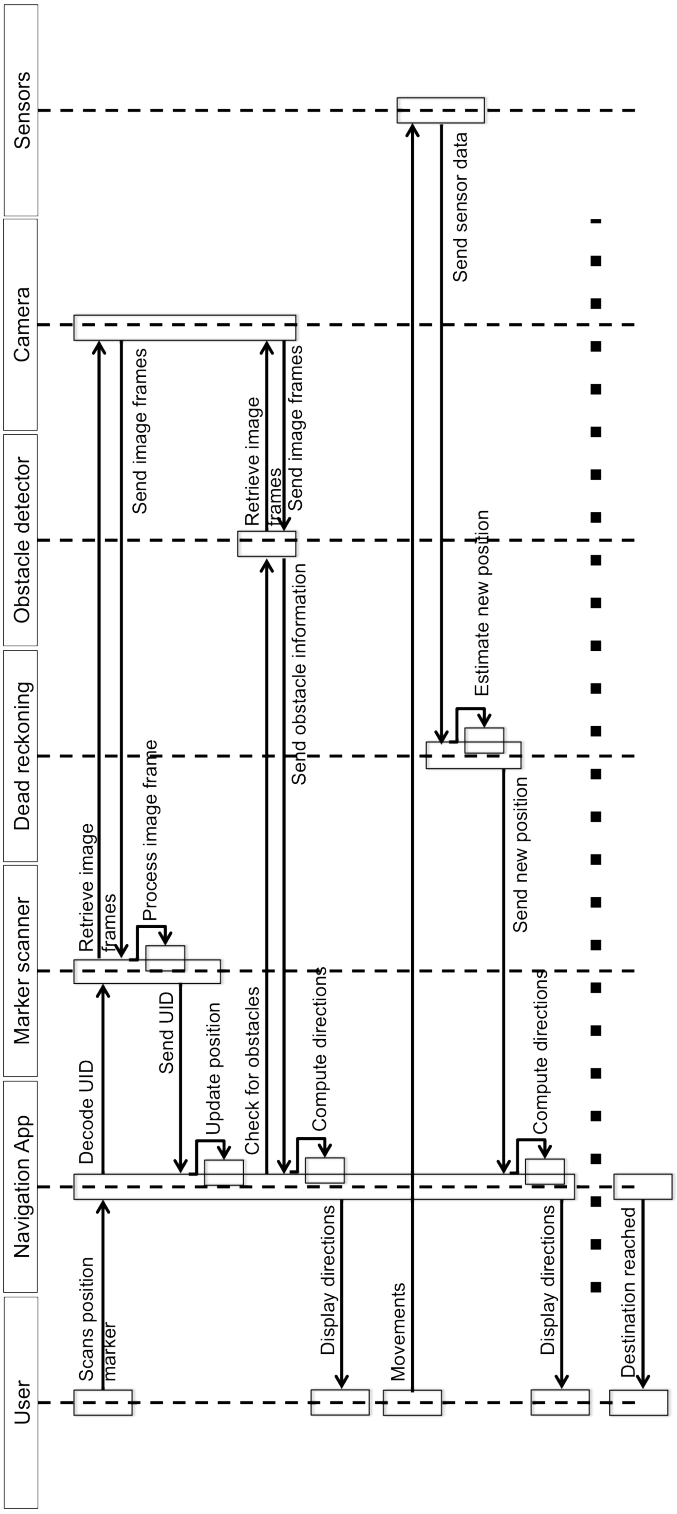


Figure 6.5: The image shows the sequence diagam of the entire system

Chapter 7

Evaluation

Our application serves to navigate a user to their desired indoor destinations. Therefore, we evaluated the performance of our application quantitatively and qualitatively by measuring how well it achieves this goal. Prior to that, we evaluated all the individual components used to build the overall application. In particular, we tested the accuracy of our position marker scanner, the precision of our dead reckoning algorithm and the time performance of our visual obstacle detector. This enabled us to validate and assess the strengths and weaknesses of each component separately before integrating them together to produce and evaluate our final smartphone based indoor navigation system.

We tested our application on the Samsung galaxy S4, which contains all the required sensors and has a back camera supporting the 640 x 480 resolution. We printed all the necessary position markers on A4 pages of paper.

7.1 Evaluating position markers

The accuracy of scanning applications is measured by the number of times it correctly decodes the data encoded in a marker. We followed this standard of measurement for our application by scanning our position markers in various different conditions. Specifically, we tested our scanner under different lighting conditions, proximity of the device and the angle of scanning.

Results

Table 7.1 shows the results obtained from scanning 10 different UID encoded position markers. The lighting conditions were used as the main criteria. We altered the distance and the angle parameters on every test (figure 7.1). If

Lighting conditions	Total scans	Correct	Wrong	Timeout
Bright	20	1	3	16
Normal	20	17	1	2
Low-light	20	0	0	20
Shadow	20	16	2	2

Table 7.1: Results for the position marker scanner

the application failed to scan a marker within 15 seconds, we marked it as a timeout.

Figure 7.1: The image shows some of the tests we conducted by altering parameters such as distance, angle and light

Analysis

Our application failed to detect any of the markers when the lighting conditions were poor. This was partly due to the low resolution of the camera as well as the conversion of the original image in to grayscale for the Hough circle transform. This made it impossible for the algorithm to detect the

centre of the circle. Our results also showed that the scanner failed to detect markers when there was a lot of light falling on the paper. In fact, the paper reflected so much light that the colours were slightly challenging to recognise even for the naked eye. As a result, our algorithm timed out most of the time as it could not decipher some of the colour regions. There were also a few wrong results, mainly caused due to the incorrect detection of the bright colour regions. Our application performed well under normal and shadowy lighting conditions. Nevertheless, there were a few errors and timeouts on both the occasions but these only occurred when the camera was held at a slanted angle and far away from the marker. The reason some of the results were incorrect were because the application failed to detect the right marker boundaries. However, we noticed that the errors quickly auto corrected themselves on the next preview frame due to the real-time nature of the application.

From our findings, we quickly realised that printing colour encoded markers on paper was not a good idea as it was susceptible to abnormal lighting conditions. Therefore, in a real case scenario, we recommend position markers to be printed on a more durable mat surface and the area next to the marker should always receive adequate light.

We decided to compare our results with some of the state-of-the-art barcode scanners available for smartphones. Online research [1] shows that most of these scanning applications achieve around 81%-96% accuracy. Under normal lighting conditions, we achieved 85% accuracy. Although this is a favourable result, we have to consider that our position markers cannot store a large amount of data, and therefore we should be making fewer errors. However, we can also claim that our application is capable of decoding markers from a larger distance and can enable the scanner to calculate the user's orientation.

Speed is also an important metric for scanning applications. However, we decided not to analyse the speed of our scanner quantitatively as accuracy was our highest priority. Nevertheless, under ideal lighting conditions and within a certain distance, our application almost always decodes a position marker within milliseconds. Besides, the only reason our application would take a large amount of time would be because the user is either too far away or at a really slanted angle making it too difficult to detect the centre of the circle or the direction indicator. Since the scanning operates in real-time, users are able to quickly alter the position of their device to get a better view of the marker. Since this process is not time-consuming, the user can scan some of these markers while they are being navigated in order to feed

[1] `http://www.scandit.com/barcode-scanner-sdk/features/performance/`

Features presented	Boundary detection	Obstacle detection	Total time
Few	20	1	3
Moderate	20	17	1
High	20	0	0

Table 7.2: Results for our obstacle detector

the application with a more accurate estimation of their current position.

7.2 Evaluating our obstacle detection algorithm

We examined the time taken by our algorithm to detect obstacles from the camera to ensure it was able to function in real-time. If the algorithm was not able to process the incoming preview frames fast enough, it would have been useless to include this feature in our final product. We ran several stress tests to ensure that our algorithm functioned as expected within a certain time when presented with a large number of obstacles.

Results

We measured the time taken from the point when our implementation received a preview frame till the successful identification of obstacles on the left, right and ahead of the user. We held the device in a static position while it was processing the preview frames from the camera. This allowed us to calculate the average time from a sample of 100 frames. Table 7.2 shows the time performance of our algorithm in milliseconds when presented with different levels of features in the environment.

Analysis

Our results showed that the average time taken to detect obstacles was approximately 173ms in the worst case. This allowed for our algorithm to process up to 6 frames per second (FPS). Under normal stress levels, our application was capable of processing 9-10 FPS on average. Although this was slightly less than the number of frames processed per second by the human eye (10-12 FPS), this lag was not noticeable from our application. However, we have to also consider that our test device, the Samsung galaxy S4, has a very high computational unit that not many smartphones currently possess.

Nevertheless, it is predicted that many other smartphones in the future will follow this specification.

We briefly considered applying this algorithm to a setting which had a textured carpet. We found that our algorithm falsely detects obstacles caused by patterns on the floor. This is a very serious drawback of our system as many sites do not always have a plain carpet. Further image processing would need to be developed in order to distinguish between the floor patterns and actual obstacles.

The results that we obtained certainly convey that this mechanism can be used for a real-time application. This was because our implementation utilised some of the OpenCV functions, which are arguably the most efficient vision algorithms currently available on the Android platform. We also reduced the resolution of the image significantly and also altered other parameters like the angle resolution to achieve faster performance.

7.3 Evaluating our dead reckoning algorithm

We assessed the performance of our dead reckoning algorithm by first measuring the accuracy of our pedometer algorithm and then its integration with the orientation sensor in monitoring user's movements.

7.3.1 Pedometer accuracy

To test the accuracy of our application's pedometer, we compared the number of steps detected from the actual number of steps taken. We asked five people with different stride lengths to walk along a fixed path for a fixed number of steps while our pedometer algorithm was running in the background. On each attempt, we recorded the steps detected by our application and averaged the results. Note that the fixed paths consisted of users walking along a straight path as well as turning to avoid obstacles.

Results

We asked the users to count how many steps they were walking and stop when they reached the target number of steps. We asked them to do this twice and averaged the results for all the ten samples. Table 7.3 shows the results we obtained from this investigation.

In addition to our findings, we repeated the same experiment with fewer samples using the most popular pedometer app on the Android market - Accupedo. Table 7.4 shows the results obtained from this app.

Number of steps	Steps detected (Average)	Range
5	5.5	4-6
10	9.3	6-13
25	22.3	18-27
50	45.4	38-51

Table 7.3: Results for our pedometer

Number of steps	Steps detected (Average)	Range
5	0.0	0-0
10	12.8	0-17
25	26.3	25-30
50	52.6	49-54

Table 7.4: Results for the Accupedo pedometer

Analysis

Our pedometer algorithm did not correctly recognise the actual number of steps on most of the attempts. The average results, however, were not too discouraging. They were very close to the actual number of steps taken. We also noticed that the average margin of error increased as the number of steps walked increased. This was bound to occur as with increasing distance there was a high chance that a step may not be detected. To find the cause of this discrepancy, we analysed the accelerometer signal to check where the peaks were not successfully detected. We discovered that our algorithm failed to identify steps when users made a sharp turn around an obstacle. The wave formed by this movement was different to when users walked normally. To solve this issue, we would have needed to distinguish the two waveforms and establish when to use the appropriate analysis.

The average number of steps detected using Accupedo were very close to the actual steps taken. In comparison, Accupedo produced more accurate results than our application. We also noticed that their algorithm generally over-approximates while ours failed to detect some of the steps and thus under-approximates. The other significant difference was that their application did not function as expected when users walked a small number of steps. In their description, they mention that they wait for the first 4-12 steps before displaying the pedometer count. In our case, it was vital that our application also detect and display small movements. Otherwise, this would have a great impact on the accuracy of our integrated application.

7.3.2 Positioning accuracy

We assessed the overall accuracy of our dead reckoning algorithm by measuring the position drift between the estimated position of the user and their actual position on the site. To accomplish this, we designed three fixed paths, as shown in figure 7.2, and observed five test users walk along these paths. We designed these paths such that they returned the user to their original starting location. We were also mapping user movements in real-time and displaying them to the user. This allowed us to calculate the position drift when the user returned back to the starting marker by finding the difference in distance between the user's estimated position according to our application and their actual final position.

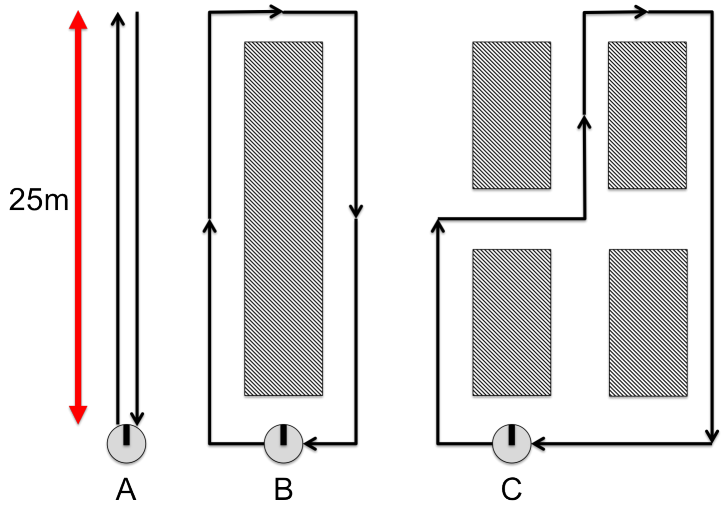


Figure 7.2: The image shows the fixed paths used for testing

Results (Single marker)

Users were asked to scan a position marker, follow the given path and then scan the starting position marker again after returning back. We modified our application slightly for this test in order to automatically calculate and display the position drift on the second scan. Table 7.5 shows the results obtained after averaging the recorded values. Note that users were asked to walk along each path twice.

Path Type	Error margin (meters)
A	1.9
B	3.4
C	5.7

Table 7.5: Results for the positioning accuracy achieved using a single marker

Analysis (Single marker)

From our results, we can see that the accuracy of our dead reckoning algorithm is not sufficient enough to provide us with a good estimate of the user's position. We can also observe that the position drift increases as the path gets longer and more complex. This is a direct result of our imprecise pedometer algorithm.

As a result, we decided to extend our dead reckoning technique by allowing users to scan additional position markers placed at various positions on the original fixed route. This allowed the application to readjust the user's position estimate on every scan and also prevented large position drifts from occurring with increasing distance. We repeated the same experiment from before with the changes shown in figure 7.3.

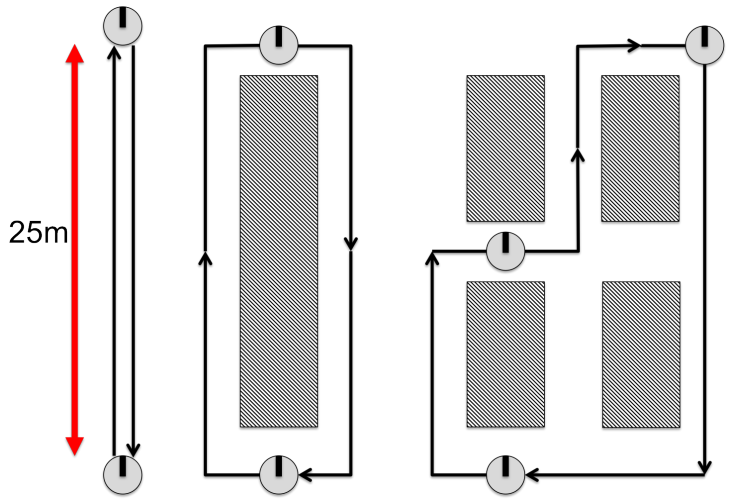


Figure 7.3: The image shows the fixed paths setup with multiple markers

Path Type	Error margin (metre)
A	0.7
B	1.3
C	2.3

Table 7.6: Results for the positioning accuracy achieved using multiple markers

Results (Multiple markers)

Table 7.6 shows the results obtained from having multiple markers on the fixed paths.

Analysis (Multiple markers)

The results show a vast improvement in the accuracy of our algorithm. Introducing multiple markers would certainly benefit the overall navigation system, but we have to be careful to not use them excessively. They should be placed within a reasonable distance from each other but also easily accessible from any passage. Position markers were initially developed for the purpose of identifying the user's initial position. However, now we have seen that they can be used to recalibrate position estimates. We shall consider this when we evaluate the end product.

7.4 Evaluating the integration of navigation system

Our final analysis comprised of assessing the overall application's performance in successfully navigating the user indoors. We performed both quantitative and qualitative analysis by testing our end product in a real case scenario.

7.4.1 Test location setup

We used the third floor of the Imperial College central library to test our application. A major portion of this site is made up of several rows of aisles with the entire floor covered by a plain carpet. All the passages have at least one exit meaning there are no dead ends. This created ideal conditions for our obstacle detection mechanism. Essentially, we were trying to build

a platform for a dummy application that allows users to navigate their way through the library to find their desired books.

We placed eight position markers at various locations around the site and also ensured that all the direction indicators pointed to the local north of the library. Then, we introduced nine destinations to mimic the location of the books. We stored the position coordinates for all these attributes locally in our application. Figure 7.4 shows the layout of the site as well as highlights the location of the position markers and the nine destinations.

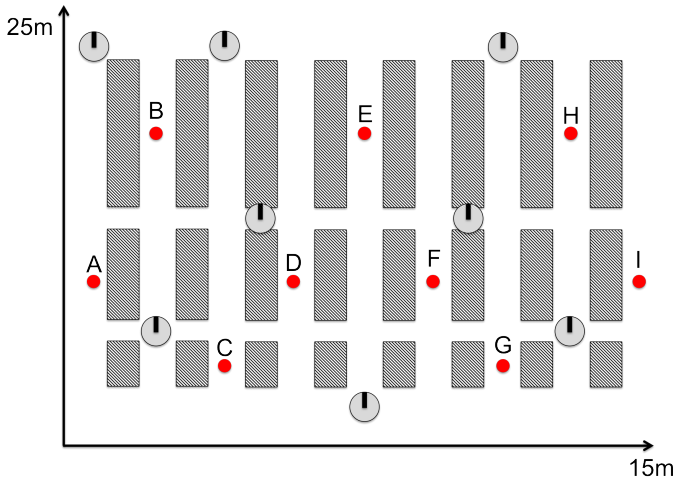


Figure 7.4: The image shows the layout of the test site with position markers (grey) and the destinations (red)

7.4.2 Quantitative analysis

The quantitative analysis involved measuring the accuracy of our application in navigating the user to their desired destination. When our application believed that the user was within half a metre of their destination, it notified them. We used this point as a reference to calculate the physical distance between the actual destination and the location where the user was notified. Seven test users were asked to follow the directions displayed by our application as well as the position markers to reach all of the nine destinations. Users did not have any previous knowledge about the location of these destinations as that could have possibly influenced the final results. The starting positions varied on every attempt but we ensured that every position marker was used at least once as the source of navigation. Prior to this, we briefly trained the users regarding the correct position to hold the smartphone de-

Destination	Error margin (metre)
A	1.7
B	2.6
C	1.1
D	2.6
E	1.2
F	3.0
G	1.8
H	1.5
I	3.3

Table 7.7: Accuracy achieved for our indoor navigation system

vice. We had to also input a rough estimate of the user's stride length for personalisation.

Results

Table 7.7 shows the distance offsets in metre for all the destinations. All the values were rounded to the nearest ten centimetres as millimetre level precision was not required.

Analysis

The results from our investigation showed that our navigation system on average guided users within 2.1 metres of their intended destination. Although this level of accuracy is generally sufficient for indoor navigation, we did observe that at many times users ended up in the wrong aisle. Due to the narrow passages in the library, our dead reckoning algorithm failed to distinguish between correct aisle. This was a direct impact from the inaccuracies discovered in section 7.3. A more accurate step detection algorithm would have definitely eliminated these false positives. However, in larger settings like museums and fairs, this would not be necessary.

We also noticed that the time taken by users to reach their destination varied. This was partly due to users walking fast and not following the directions correctly as well as the position drift resulting from our dead reckoning algorithm. Therefore, users did not always follow the optimal path. Besides, accuracy was always our priority and not the shortest distance to the destination. Having multiple markers greatly improved the accuracy as it enabled the application to readjust position estimates when the user had a chance to scan a nearby marker.

Solution	Error margin (metre)
Ego-motion	2.7
Triangulation	1.7
Inertial	1.5

Table 7.8: Accuracy achieved with other indoor navigation systems

We compared the accuracy of our application with other smartphone based indoor navigation systems previously mentioned in our background research. In particular, we looked at the accuracy achieved through ego-motion[19], Wi-Fi triangulation[16] and a solution using inertial sensors for dead reckoning[17]. Table 7.8 shows the mean distance error of these three techniques in metre.

Both the Wi-Fi triangulation and the inertial solutions achieved less that 2 metre accuracy. The triangulation paper also claims that they obtain more accurate results with a hybrid solution combining Wi-Fi and inertial measurements. Furthermore, the latter claim that they tested their solution while the device was inside the pocket. We went on to analyse the testing method employed by both of these techniques. We found that the area of the test locations were both slightly larger than the one we used. Nevertheless, the paths they considered were very simple and did not contain many obstacles as their focus was more on guiding the user from one room to another. On the contrary, our application was tested thoroughly with several shelves and narrow passages in our path. However, the major drawback of our application was that it required more user interaction such as scanning position markers as opposed to these solutions. Our accuracy, nonetheless, was not too worse off in comparison. We even managed to obtain a more accurate result compared to the ego-motion approach.

7.4.3 Qualitative analysis

The verbal feedback received from the test users were mainly positive. They could see the potential benefits of the application in various different industries that we had previously not considered such as hospitals and conferences. However, some of the users did raise a few concerns regarding the power consumption of the application as well as the user interface. Our application currently performs a large amount of computation in real-time taking in data constantly from the camera and the sensors. This drains the smartphone battery rapidly. A potential solution would be to remove the real-time obstacle detection and instead allow users to use their common sense to avoid

obstacles while following the given directions. This would eliminate a bulk of the vision processing and would certainly increase battery life compared to before. With respect to the user interface, some suggested to display a map of the site as it would help to visualise the surroundings and their position with respect to the site. This contradicted with one of our key objectives, which was to not pre-load an indoor map of the site. As a result, we decided to extend our application interface to display a mini-map showing only the orientation of the user and the direction of the destination. Although this did not reveal the full picture, it certainly aided navigation without requiring the application to store any indoor maps.

7.5 Summary

Our results suggest that our application provides a reasonably accurate and a flexible medium to allow users to navigate indoors without requiring many infrastructural changes. Nevertheless, we also realised that the average accuracy of our application was not satisfactory in certain indoor environments such as libraries and small supermarkets, which typically have narrow aisles. This was largely due our faulty dead reckoning implementation which failed to detect a step when users made a sharp turn. In our evaluation, we managed to suppress this problem to a certain extent by introducing more position markers around the site. Although this improved the accuracy significantly, it resulted in more user interaction. The lighting condition in the site would also largely influence the accuracy and the time taken to decode position markers. This was an important factor to consider when placing position markers. Printing these markers on a mat anti-reflective surface would certainly allow for their successful detection even under bright conditions. However, we had to avoid placing these position markers under low levels of light.

The application of obstacle detection to indoor navigation was rather innovative. It allowed the navigation system to assess the surrounding environment up to a certain extent and then verify whether users would be able to walk in a given direction from the current position. Nevertheless, to successfully integrate and test this mechanism, we had to impose an important restriction on the site, i.e. the floor should not have any patterns. Furthermore, the user had to also keep the phone in a set position throughout their entire indoor journey as our algorithm relied upon the constant preview frames received from the smartphone camera. This also significantly increased the battery consumption of the device. Here, we have to agree that the limitations of using this component outweigh the benefits it brings

to navigation. A potential solution for the future could be to remove this component completely and instead let users use their intuition to avoid obstacles and at the same time keep up with the given direction hint. To aid navigation further, we could supply our application with an indoor map of the site. Although this goes against one of our key objectives, our qualitative analysis revealed that users actually prefer to know the layout of the site and their current position with respect to their destination.

Chapter 8

Conclusion

Throughout the report, we have demonstrated a scalable smartphone based solution for indoor navigation that requires minimal changes to the site infrastructure. We have also assessed all the individual components required to build this final application separately before integrating them together for a thorough evaluation of our end product in a library setting. Our solution combines various computer vision concepts with inertial measurements to estimate a user's position as well as detect obstacles around them.

In this chapter, we have summarised the key outcomes from the project and also outlined all the possible extensions that can be made to the current system in the future.

8.1 Summary

As part of building an indoor navigation system, we designed our own custom markers which were used to encode integers as well as a direction indicator for obtaining the user's position and orientation. We had to study various computer vision techniques, such as Hough transforms and Canny edge detection, to develop a camera based scanner to decode these markers. Our analysis showed that it was possible to almost always extract the encoded data within milliseconds under normal lighting conditions. Furthermore, our scanner was capable of decoding data from any angle even when the user was standing at a distance. This gave the user the advantage of scanning position markers without havi7ng to bend down. However, we also noted that the markers did not function as expected when the lighting condition were either too bright or too low.

We followed up on these vision techniques to also develop an obstacle detector using the smartphone camera. After assessing the time performance

of this component, we were able to validate its usability in a real-time application. Therefore, we decided to include this mechanism in our final system for checking the feasibility of the user moving to a new position.

From here, we focussed our attention on building a pedometer algorithm using the inertial sensors on the smartphone. This involved filtering and analysing the accelerometer signals to correctly detect a step and then keeping track of these user movements as part of the dead reckoning process. Our initial evaluation for this feature revealed that there was a significant margin of error in the user's position with increasing distance. This prompted us to place more position markers around the site. Thus we had to request users to scan additional markers while being navigated whenever there was one in proximity. This significantly improved the accuracy of our dead reckoning algorithm.

The final step comprised of combining these three components together in order to display directions that would eventually lead the user to their destination. A full functional testing of the application demonstrated that our system was capable of guiding the user to their intended destination within a small margin of error (2.1 metres). In general, this level of accuracy was adequate enough to achieve indoor navigation.

8.2 Future work

There is a great amount of scope for future improvements to our current implementation that can enhance both the performance and the user experience of the application.

Improving pedometer accuracy

We identified the source of inaccuracy in our pedometer algorithm as the inability of our implementation to correctly detect a step when users walk around a steep corner. This was one of the main reasons our algorithm performed worse in comparison to the pedometer apps available on the market. In the future, we would need to model the characteristics of the accelerometer data separately for when users make a sharp turn. Then, we would need to distinguish this waveform from the one where users walk in a straight line to be able to correctly detect all the steps.

Navigation across several floors

At the moment, our application can only navigate the user on a single floor. For our application to support navigation across multiple floors, we would

need to detect and distinguish between the steps taken to walk up and down the staircase. This entails complex signal processing and even then our implementation may not achieve the desired result. A much simpler alternative entails the user scanning a position marker on reaching the destination floor. For instance, the application would navigate the user to the nearest staircase and then tell them to walk up or down a certain number of floors. A position marker would be placed at the entrance and at the exit near the staircase of every floor. When the user reaches the requested floor, they can scan this position marker and continue as before. Similarly, this could also be applied to lifts and escalators.

Marker design

The current position markers are susceptible to poor lighting conditions. This is due to the markers encoding data using colours. If too much or too little light falls upon the marker, then the camera preview would not be able to correctly identify the colours. A more robust design could use shapes and patterns to encode data like in QR codes. The ability to store more data in these markers would be beneficial for larger site. Another problem with the current marker design is that they only work really well when placed on the floor. When they are stuck to the wall, they only allow scanning if the user is right in front of it and not at a slanted angle. This is due to the incorrect detection of the circle centre when the user is at an angle. The marker design could be changed to incorporate its correct detection from every angle.

Obstacle detection

Our obstacle detection mechanism currently detects all the linear edges in the environment. Due to time limitations, we were unable to expand this further to be able to distinguish between floor patterns and actual obstacles. A straightforward analysis on the line angles would help establish whether an obstacle has three dimensional properties. This could be used to eliminate some of the false positives. However, more work needs to be done with complex floor patterns.

Path planning

Once our indoor navigation system is setup, we could extend it to employ some form of optimal path planning that users can follow from the moment they enter the site. For instance, users can create their shopping lists offline and when they enter the supermarket, our application could use this data to plan and display directions from the start till the end. Museums and art

galleries could also offer special tours to visitors using this app navigating them through all their preferred exhibits. Overall, indoor navigation opens up a wide variety of extensions tailored to specific industries.

Hybrid dead reckoning approach

Our pedometer algorithm can be combined with ego-motion from the smartphone camera to yield a more accurate dead reckoning technique. The only concern is that the preview frames would be slightly blurred when users start walking. Image stabilisation algorithms exist that can fix this problem up to a certain extent. However, this part of vision is highly theoretical and would require a large amount of time to study and develop such a prototype.

Storing position information online

All the position information regarding the markers and the items were stored locally for testing. For a real case scenario, an online platform should be provided to store and fetch all these details. This data can be automatically loaded when the user is close to the location. However, if there is a significant amount of data stored relating to the site (for example, position of all the books in a library), the client should provide Wi-Fi access to all the users to retrieve this data.

Bibliography

[1] Free alliance. Why is wifi inadequate for real time asset or personnel tracking?

[2] Y. Alon, A. Ferencz, and A. Shashua. Off-road path following using region classication and geometric projection constraints. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2006*, volume 1, pages 689–696, June 2006.

[3] Levente Bagi. Pedometer - android app.

[4] P. Bahl and Padmanabhan V.N. Radar: an in-building rf-based user location and tracking system. In *2*, pages 775 – 784, March 2000.

[5] John Canny. A computational approach to edge detection. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, 1986.

[6] Sudarshan S. Chawath. Marker-based localizing for indoor navigation. In *Intelligent Transportation Systems Conference, 2007. ITSC 2007. IEEE*, pages 885 – 890, September 2007.

[7] Jaewoo Chung, Matt Donahoe, Chris Schmandt, and et al. Indoor location sensing using geo-magnetism mobisys, 2011.

[8] Andrew John Davidson. Real-time simultaneous localisation and mapping with a single camera. In *Proceedings of the Ninth IEEE International Conference on Computer Vision, 2003*, volume 2, pages 1403 – 1410, October 2003.

[9] Sinan Gezici, Tian Zhi, G.B. Giannakis, and et al. Localization via ultra-wideband radios: a look at positioning aspects for future sensor networks. *Signal Processing Magazine, IEEE*, 22:70–84, July 2005.

[10] Google. Andoid sensor guide.

[11] Ajay Gupta. Googles approach of a nomap wi-fi zone.

[12] Jan Hoffmann, Matthias Jungel, and Martin Lotzsch. A vision based system for goal-directed obstacle avoidance. In *RoboCup 2004*, volume 3276, pages 418–425, 2005.

[13] Sung Hyun Jang. A qr code-based indoor navigation system using augmented reality. In *RoboCup 2004*, March 2012.

[14] Kamol Kaemarungsi and Krishnamurthy Prashant. Modeling of indoor positioning systems based on location fingerprinting, 2004.

[15] L. Klingbeil and T. Wark. A wireless sensor network for real-time indoor localisation and motion monitoring. In *International Conference on Information Processing in Sensor Networks, 2008.*, pages 39–50, 2008.

[16] Manh Hung V. Le, Dimitris Saragas, and Nathan Webb. Indoor navigation system for handheld devices, 2009.

[17] Fan Li, Chunshui Zhao, Guanzhong Ding, and et al. A reliable and accurate indoor localization method using phone inertial sensors. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, pages 421–430, September 2012.

[18] Y. Li and S.T. Birchfield. Image-based segmentation of indoor corridor floors for a mobile robot, 2010.

[19] Jo Agila Bitsch Link, Felix Gerdsmeier, Paul Smith, and Klaus Wehrle. Indoor navigation on wheels (and on foot) using smartphones. In *Proceedings of the 2012 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, November 2012.

[20] B.D. Lucas, Kanade, and T. An iterative image registration technique with an application to stereo vision. In *7th International Joint Conference on Artificial Intelligence (IJCAI)*, April 1981.

[21] Android market. Accupedo app.

[22] Android market. Runtastic app.

[23] Gerardo Carrera Mendoza. Robot slam and navigation with multicamera computer vision, 2012.

[24] Alessandro Mulloni, Daniel Wagner, Dieter Schmalstieg, and Istvan Barakonyi. Indoor positioning and navigation with camera phones. In *Pervasive Computing, IEEE*, volume 8, pages 22–31, April 2009.

[25] Clark F. Olsona, Larry H. Matthies, Marcel Schoppers, and Mark W. Maimone. Rover navigation using stereo ego-motion. In *Robotics and Autonomous Systems*, volume 43, pages 215–229, June 2003.

[26] OpenCV. Opencv for android.

[27] OpenCV. Opencv wiki.

[28] Kwanghyo Park, Shin Hyojeong, and Hojung Cha. Smartphone-based pedestrian tracking in indoor corridor environments. In *Personal and Ubiquitous Computing*, volume 17, pages 359–370, December 2011.

[29] En Peng, Patrick Peursum, Ling Li, and Svetha Venkatesh. A smartphone-based obstacle sensor for the visually impaired. In *Proceedings of the 7th international conference on Ubiquitous intelligence and computing*, pages 590–604, 2010.

[30] Christopher Rasmussen. Grouping dominant orientations for ill-structured road following. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2004*, volume 1, pages 470–477, 2004.

[31] Ahmed Wasif Reza and Tan Kim Geok. Investigation of indoor location sensing via rfid reader network utilizing grid covering algorithm. In *Wireless Personal Communications*, volume 49, pages 67–80, June 2008.

[32] Martin A. Skoglund. *Visual Inertial Navigation and Calibration.* Linkping University Electronic Press, 2011.

[33] Open source. Zxing - an open-source, multi-format 1d/2d barcode image processing library.

[34] Gideon P. Stein, Ofer Mano, and Amnon Shashua. A robust method for computing vehicle ego-motion. In *Proceedings of the IEEE Intelligent Vehicles Symposium, 2000*, pages 362–368, 2000.

[35] Oliver Woodman and Robert Harle. Pedestrian localisation for indoor environments. In *Proceedings of the Tenth International Conference on Ubiquitous Computing (UbiComp 08)*, pages 362–368, 2008.

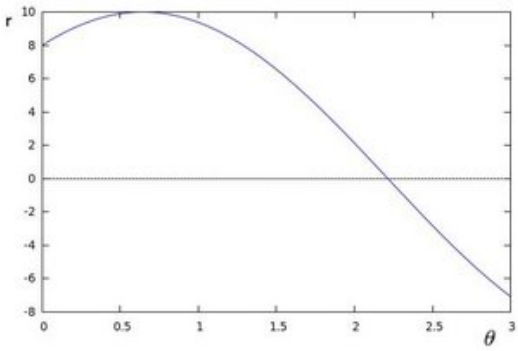
[36] Yan Xiaotan. A wlan fingerprinting and triangulation based location determination system.

Appendix A

Hough line transform example

The following example and the corresponding images are taken from the OpenCV documentation for Hough line transform[1]

Let us take an example where the binary image contains a point x = 8 and y = 6. If we plot all the possible values of θ and r for that point, we obtain the following sinusoidal curve:

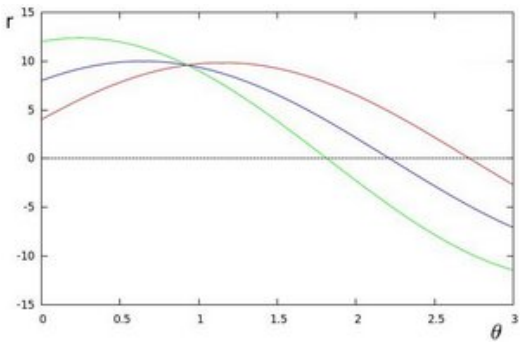


We repeat the process for every binary point in the image. So suppose there are two other points in the image with coordinates (9, 4) and (12, 3), we then obtain the following plot.

We can see that the three curves intersect when r = 0.925 and = 0.96. So there is a high probability that there exists a line in the original image with the following equation.

$$0.925 = xcos(0.96) + ysin(0.96)$$

[1] `http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/hough_lines/hough_lines.html`



There exist many such intersection points in the graph corresponding to the different lines detected in the image. Furthermore, we can set a threshold for the minimum number of intersections to eliminate any spurious line detections.