5-2013

Digital Labeling and Narrative Mapping in Mobile Remote Audio Signage: Verbalization of Routes and Generation of New Verbal Route Descriptions from Existing Route Sets

Tharun Tej Tammineni
Utah State University

Recommended Citation

Tammineni, Tharun Tej, "Digital Labeling and Narrative Mapping in Mobile Remote Audio Signage: Verbalization of Routes and Generation of New Verbal Route Descriptions from Existing Route Sets" (2013). *All Graduate Plan B and other Reports*. 301.
https://digitalcommons.usu.edu/gradreports/301

DIGITAL LABELING AND NARRATIVE MAPPING IN MOBILE REMOTE AUDIO SIGNAGE: VERBALIZATION OF ROUTES AND GENERATION OF NEW VERBAL ROUTE DESCRIPTIONS FROM EXISTING ROUTE SETS

by

Tharun Tej Tammineni

A report submitted in partial fulfillment
of the degree requirements for the degree

of

MASTER OF SCENCE

in

Computer Science

Approved:

Dr. Vladimir A. Kulyukin	Dr. Daniel Watson
Major Professor	Committee Member

Dr. Curtis Dyreson
Committee Member

UTAH STATE UNIVERSITY
Logan, Utah

2013

ABSTRACT

Independent navigation is a great challenge for people with visual impairments. In this project, we have designed and implemented an assisted navigation solution based on the ability of visually impaired travelers to interpret and contextualize verbal route descriptions. Previous studies have validated that if a route is verbally described in sufficient and appropriate manner then VI can use their orientation and mobility skills to successfully follow the route.

In this project, we do not consider the issue how the VI will interpret the route descriptions, but we aim to identify and generate new verbal route descriptions from the existing route descriptions. We discuss different algorithms that we have used for extracting the landmarks, building graphs and generation of new route descriptions from existing route info.

(68 pages)

ACKNOWLEDGMENTS

Foremost, I would like to express my sincere gratitude to my advisor Dr. Vladimir Kulyukin for the continuous support of my Masters Study and research, for his patience, motivation, enthusiasm and immense knowledge. It is largely because of him that I completed this work. He always gave me achievable deadlines for the project completion.

I thank my committee members, Dr. Dan Watson, Dr. Curtis Dyreson for their helpful feedback.

I would also like to thank all my friends who helped me in some way or other in completing the project.

Tharun Tej Tammineni

CONTENTS

TABLE OF TABLES

TABLE OF FIGURES

CHAPTER 1

INTRODUCTION

In a recent study by the World Health Organization (WHO, 2012), it was reported that a total of about 285 million people are visually impaired across the globe (VI), of whom 39 million are blind and 246 have low vision. Approximately 90% of visually-impaired individuals are living in developing countries. The main cause of visual impairment is uncorrected refractive errors. Accessibility and mobility are big problems to visually impaired people as compared to sighted individuals.

Usually, VI people are accompanied by another person in a given environment to aid and guide them. This says that the visually impairment causes them to be dependent on others, which can affect their mental health. Inability to navigate an environment is a major hurdle for VI individuals (LaPlante & Carlson, 2000).

VI people face challenges navigating in different environments. The motivation of this project is Shopping independently is the next biggest things that needs to happen after driving. One of the most challenging environments is the shopping complex (Food Marketing Institute Research Department, 2006). A recent study by the Food Marketing Institute Research Department (2006), shows that a typical modern supermarket store has an average of 45,000 products and has a median store size of 4,600 square meters.

There are assistive devices such as wheelchairs, scooters, canes, and guide dogs that help VI individuals be more mobile and independent. These may be helpful for navigation in open spaces (Helal, Moore, & Ramachandran, 2001), but in a place like a shopping mall, it is difficult for them to identify the products and get the required items independently.

The study in Kulyukin & Pentico (2007), states that grocery shopping is a barrier for independence, and they usually go with sighted guides or they do not go at all. So, there is a need for a device that would help them to do grocery shopping independently and a help them to navigate to different locations.

The navigation for a sighted person is totally different from a VI because the sighted person can make observations, such as noticing sidewalks, intersections, directions. But for a VI person he/she must physically encounter these sidewalks and intersection to actively sense them with a cane, but he/she may not know the orientations with respect to the direction.

The other motivation for building this automatic verbal route engine is that existing maps like Google Maps (2013) and Mapquest (2013) are geared primarily to sighted individuals. VI individuals require more detailed verbal route descriptions for independent travel.

The objective of this project is to help the VI person to navigate dynamic and complex environments by following verbal route directions and narrative maps. A narrative map is a verbal description of instructions that are used to assist people in navigation. Wayfinding is the process of planning how to get to a desired destination. We also aim at building new set of verbal route descriptions from the existing set of route descriptions.

This report is organized as follows: Chapter 2 provides the related work and information of different researches done on assistive technology. Chapter 3 analyses the task and provides brief information on different stages of shopping. We also briefly describe the solution we implement. Chapter 4 provides the technical details of the hardware and software components that are used in developing the app. Chapter 5 presents the demo of the app. Chapter 6 provides information regarding the existing maps, and then we discuss the verbal route framework that is used by VI people. Chapter 7 describes the process of extraction of the landmarks and the tools used. Chapter 8 provides the details of generating verbal routes and the algorithms used. A detailed example is also provided. Chapter 9 provides the demo of the navigation app. Chapters 10 and 11 discuss limitations, future work, and conclusions of the study.

CHAPTER 2

RELATED WORK

In order to assist VI people with shopping effectively in grocery stores, different tools have been invented. The user has two tasks when shopping: to identifying the product that he/she is holding, and to find where a product is located within the store. A few VI people have partial vision so they can navigate within the store, but they find difficulties in identifying and reading the labels of the products. A magnifying glass helps to some extent, but is not always useful for everyone.

A portable device called i.d. mate Quest (En-Vision America, 2013), is a talking barcode scanner that uses text-to-speech to help individuals identify products. It has a built-in database with additional information like instructions, ingredients, and item warnings. This device has the MP3 capabilities that allow the user to store and access audio files. Users can thus record instructions to help them in finding products, but this would not scale for large data sets. There are other devices that use image recognition to inform the details of the product to the user, but none of them are helpful in localizing the product. Some of the research has aimed in solving these issues.

RoboCart

RoboCart is a shopping cart with a robotic base (Brooks & Montanez, 2006). This cart helps the users to navigate to the respective product location that they have entered. The cart has a laser based Monte Carlo Markov localization (MCL), which interacts with the RFID tags placed on mats that are present on the floor to identify its location (Fox, 1998). Disadvantages of this tool are the cost and speed at which it travels. Shoppers were able to move faster than the RoboCart.

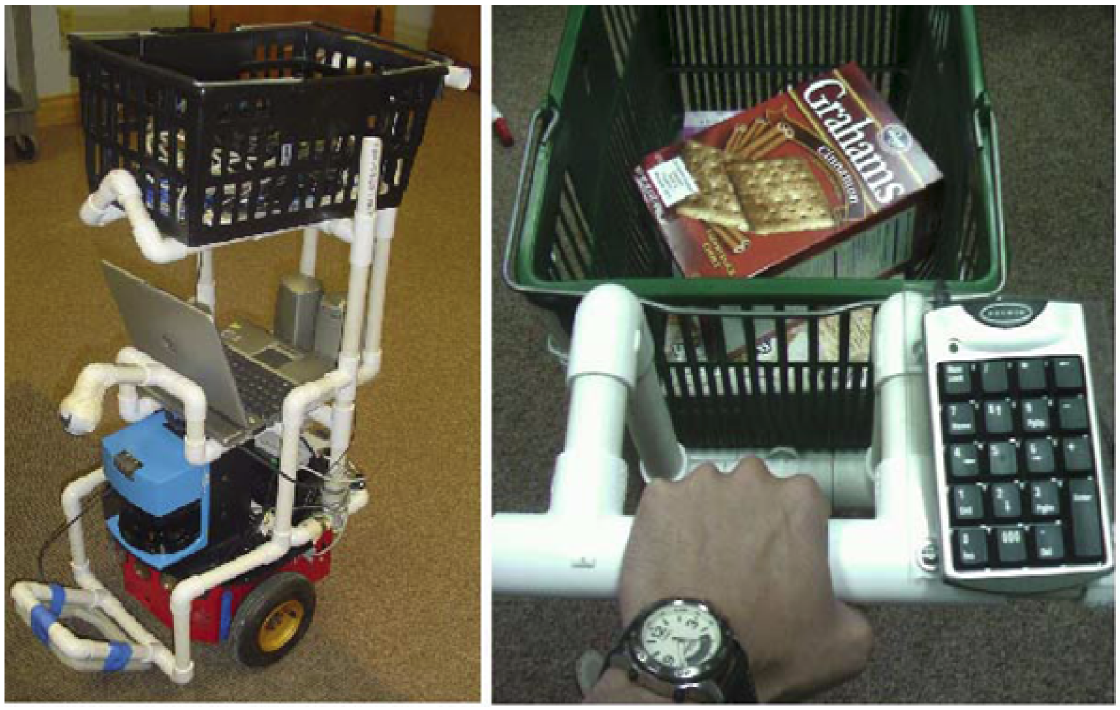


Figure 2.1 RoboCart (left); RoboCart's handle with Belkin 9-numeric keypad (right)

ShopTalk

ShopTalk is a wearable small-scale system that helps VI people to effectively identify specific products (Nicholson, Kulyukin, & Coster, 2009). This device consists of a barcode reader and a numeric keypad.

A topological map is built by connecting the store entrance, aisle entrances, open areas, and cashier lanes, and is stored in the computational device. When the user enters the data using the numeric keypad, the system generates verbal route and product search directions from the map. A super markets places barcodes on the front of shelves beneath the products; each barcode is a topological position for locating every product in the store through verbal directions.

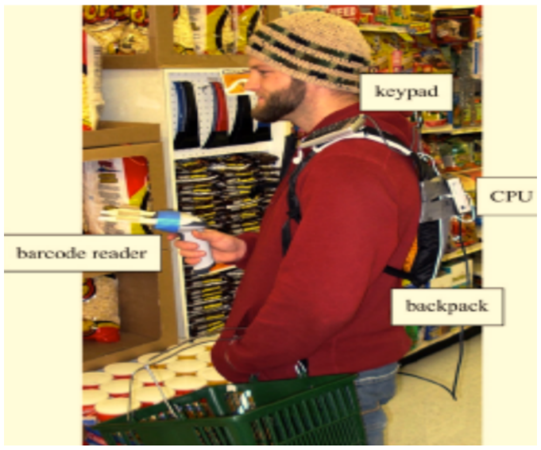


Figure 2.2 ShopTalk's hardware

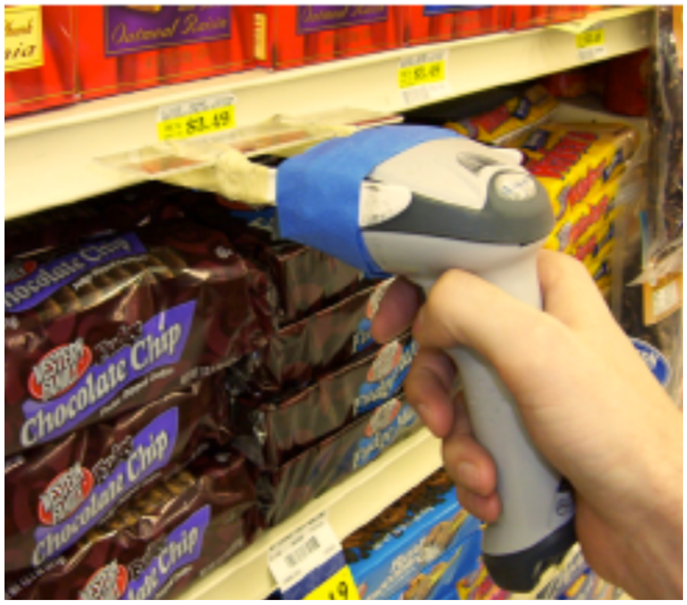


Figure 2.3 Barcode scanner with the added stabilizers, resting of a shelf lip

ShopMobile

ShopMobile is a camera equipped smartphone that is placed in a hard case with two plastic stabilizers (Kulyukin & Kutiyanawala, 2010). These stabilizers are used align the camera with the barcode. The barcode scanning method identifies the barcodes and upon successful detection the system requests the user to slide the phone along the shelf. Once barcode is identified, the user can pick up the product.

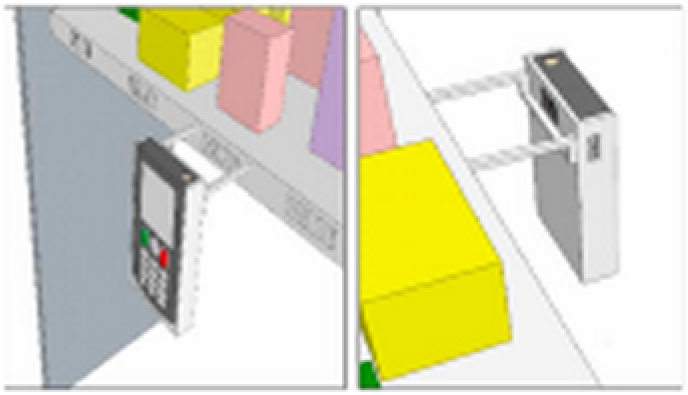


Figure 2.4 ShopMobile system consisting of camera equipped smart phone in hard

Navigation Systems

Talking Lights is a system that uses existing light fixtures in indoor environments to provide localization and navigation support (T.L., LLC, 2009). Since the tool uses store lighting systems, it is confined to indoor-based areas.

Talking Signs is a device developed by the Smith-Kettlewell Eye Research Institute that uses infrared transmitters for navigating the user (Crandall, Bentzen, Myers, & Brabyn, 2001). In this tool, transmitters encode and transmit recordings of human speech. The user holds a receiver while actively moving it in order to find the transmitted signals. Once the signal is detected Talking Signs decodes the speech and navigates the user towards the signals. It can be used in indoor and outdoor environments.

In his book, Lynch (1960) discusses on how the people perceive cities and, states that people orient themselves using mental maps in urban situations. He describes way-finding as a strategic link in which people organize the elements in a pattern. According to Lynch, the mental maps usually consist of five elements, namely paths, edges, districts, nodes, and landmarks that are necessary to organize urban mobility. Lynch also describes Legibility as the ease at which people can understand the layouts of a place. The five elements are explained below:

- **Paths** are the places where people usually travel. They can be walkways, streets, and railroads. These are the most frequent elements that people will have in their image while they travel.
- **Edges** can be boundaries, dividing lines, or line breaks in continuity. These are usually linear elements that are not considered as paths by the observer. Edges can be railroad cuts, shores, and walls.
- **Districts** are large sections of city separated by identity or character. These are medium-to-large areas of the city, usually believed as two dimensional, in which the persons can mentally enter. Districts can be center, midtown, and suburbs.
- **Nodes** are usually strategic focus points in the city that are used for orientation like junctions and squares. These can be places of a break in transportation, a crossing or convergence of paths, or moments of shift from one structure to another.
- **Landmarks** are points of references used to identify where the person is located. They can be physical objects like buildings, signs, or stores.

In order to make sure that the person's orientation is correct, , a clear mental map is required. A mental map is an individual's own map of their known world, and makes people feel emotionally secure. Lynch states that mobility is a process of identifying the environment through clear maps rather than just free flowing movement. He says that design should be made in three related movements, namely mapping, learning, and shaping. First, people should have clear mental map of their environment. Second, in the given environment they should learn how to travel. Third, in the given environment people should be able to operate and act.

CHAPTER 3

TASK ANALYSIS

Shopping usually involves three stages for a sighted individual who has a list of items that needs to be purchased. In the first stage the user pulls a cart in a particular direction and starts searching for the product that is listed on his/her shopping list. Let us call this current item the *target* item. In the second stage, the user searches for the target item and proceeds to the second item on the shopping list if the target item is found. In the third stage, the user travels from the last target item to the cashier counter, pays, and leaves the store.

The first and third stages are simpler than the second stage. Here we will focus on the second stage of the shopping task. This second stage can be seen as two sub stages: product localization and product identification. In product localization, the particular product location is identified, and then the user navigates to that aisle. In product identification the user makes sure that the product he/she is looking at is the exact target item. This is important because two products may look similar in shape and dimensions until the user reads the text on the product. For example, two different cereal boxes may appear to be similar in shape and dimension but hold different types of cereal. In this study, we will only focus on product localization, i.e. navigating the user towards the aisle that contains the desired products.

To meet this aim, we generate a set of verbal route descriptions for the user to navigate effectively within the store, as well as in outside environments. We use the concept of RIAS in order to find the aisles that contain different products.

In our experiment it is assumed that the shopping store is equipped with RIAS transmitters and that the user will be carrying a RIAS receiver that has Bluetooth enabled in it. Whenever the user moves around the shopping store he/she will receive different signals with respect to transmitters. Once the signal is received by the receiver it is captured by the Android mobile device using the Bluetooth communication. There will be a prebuilt database associated with this signal ID that is stored. Whenever we receive this ID, the device will check against the database and then give the aisle details in speech format so that the user can hear.

Edwards, Ungar, and Blades (1998) state that the way-finding process for sighted is different from the same process for the VI. VI people require clearer verbal route descriptions that include landmarks. Sometimes, the VI person will not be familiar with the adjacent landmarks and will miss the shortest routes to their destinations. In our experiment, we aim to generate a new set of verbal routes from the existing route descriptions. This concept works in any dynamic and complex environment. Wikipedia is considered as a source for our experiment's verbal route descriptions. We break each route description into sentences and identify the landmarks. A graph is constructed with all the extracted landmarks. We store the corresponding sentences for each landmark. In this experiment, it is assumed that each RIAS transmitter is associated with a landmark. We know the set of landmarks that are reachable from that particular landmark. Once the user selects his destination landmark then we traverse the graph, identify the landmarks, and replace it with sentences stored to frame new verbal route descriptions. A detailed procedure with examples is discussed in Chapters 8 and 9.

CHAPTER 4

ANDROID SMARTPHONE APPLICATION FOR DIGITAL LABELING AND NARRATIVE IN REMOTE INFRARED AUDIO SIGNAGE

The aim of this app is to help VI people to shop independently without requiring any assistance from other people. It is assumed that the shopping mall has RIAS transmitters installed on its premises. A shopper carries a wireless Bluetooth-enabled RIAS receiver and an Android smartphone with Wi-Fi or 3/4G and Bluetooth.

As the VI person walks through the store the RIAS receiver will receive the RIAS transmitter IDs. The Android application will poll the RIAS receiver via Bluetooth to receive the ID of the currently noted transmitter. Upon receiving the ID, the application will translate the ID to a URL. Once the URL is retrieved, the data is extracted and will be either shown in the browser or be read out loud using the Android text to speech module so that the VI person can hear the data.

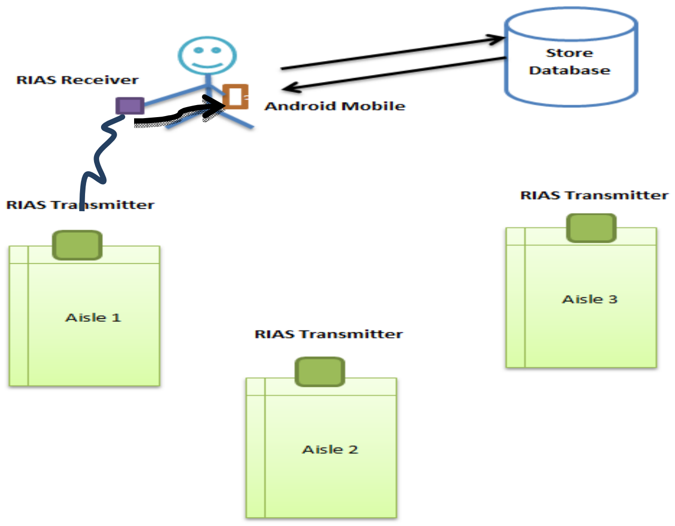


Figure 4.1 Demonstration of shopping using Android smartphone in RIAS equipped mall.

Hardware Used

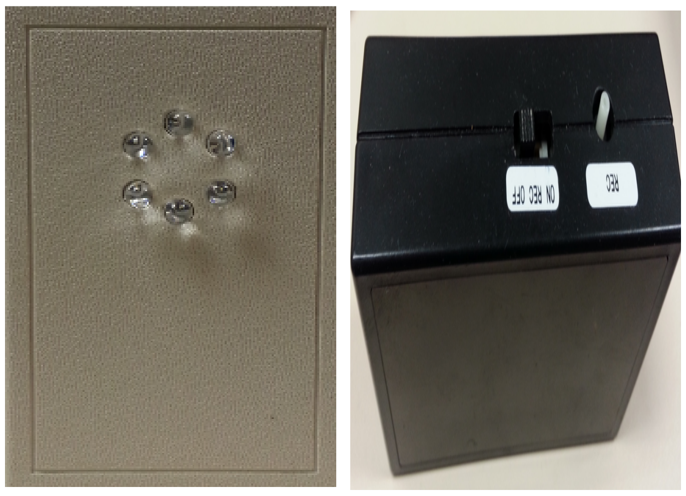


Figure 4.2 RIAS Transmitters fitted to aisles

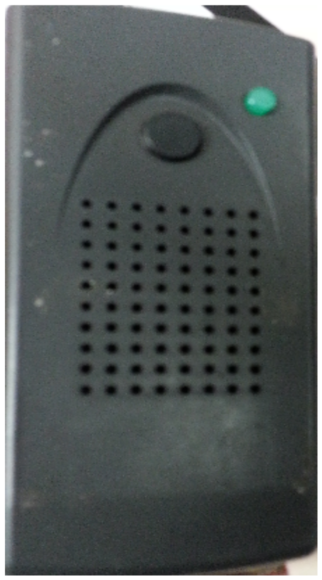


Figure 4.3 RIAS Receiver

Figure 4.4 Android Mobile Devices

Technical Details

The data related to the RIAS transmitter IDs are stored in the database as follows:

ID	URL TITLE
0x0100	Food snacks
0x0200	Frozen
0x0300	Breakfast
0x0400	Cosmetics
0x0500	Vegetables
0x0600	Electronics
0x0700	Dairy Products

Table 4.1 ID URL title database table

For developing this app, we used an Android mobile device that has Bluetooth enabled and software with a minimum SDK version of 10. It also has the ability to connect to the internet using Wi-Fi or 3/4G.

The Android SDK provides the classes that manage the Bluetooth functionality, such as scanning for devices, connecting with devices and managing the data transfer between them (Android, n.d.). The following are the few steps that are to be implemented for successful communication of the devices using the Bluetooth module.

1. **Setting Up Bluetooth:** Before starting the application it is necessary to check if the device has the Bluetooth support. If the mobile supports Bluetooth, but it is turned off, then we will prompt the user to turn on the Bluetooth by displaying a dialog box.
2. **Bluetooth Permissions:** If the application wants to use the Bluetooth features, then we need to declare either one of the permissions namely Bluetooth or Bluetooth Admin. If the application wants to request a connection, accept a

connection or transfer the data and then give the app Bluetooth permission in the Android manifest file.

3. **Finding Devices:** Once the Bluetooth is turned on we need to find the devices in which we are interested in communicating. In this step we scan for the Bluetooth enabled devices in the local area. We obtain the name and unique MAC address of each device. Using this we can initiate the connection. During the first connection, a pairing request is automatically presented to the user. Once pairing is complete, we can scan for already paired devices that are stored.
4. **Querying Paired Devices and Discovering Devices:** It is useful to check already paired devices to see if our target device already exists. We discover the devices through scanning for about 12 seconds, then the list of available devices with their Bluetooth names will be displayed. Since the discovery process is heavy and consumes lot of resources, we need to stop the discovery before a connection is established.
5. **Connecting Devices:** To establish the communication between the devices we need to establish a connection between them. This is a server client architecture where one will open a server socket and other device must initiate a connection. The server and client have established a successful connection if they are on same RFCOMM channel.
6. **Managing Connection and Establishing Communication:** After the two devices are connected then each one will have a Bluetooth socket. This socket is used for the data exchange. In our app there is a dedicated thread that will

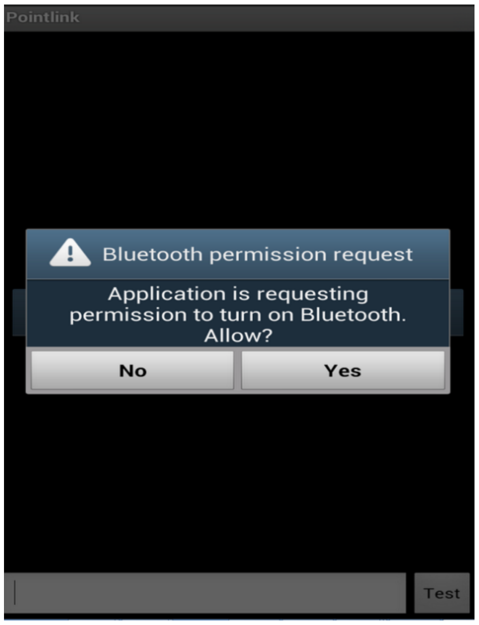
constantly scans the hardware device in order to fetch the RIAS IDs. Once the IDs are received, we perform identify the aisle in which the user currently resides.

CHAPTER 5

ANDROID APP DEMO

The following is a demo of the operations for establishing a connection.

Open the App: If the Bluetooth is not turned on then it will prompt the user to turn it on in order to proceed further.



Application is requesting permission to turn on Bluetooth. Allow?

Figure 5.1 Requesting user to turn on the Bluetooth

Menu Options: The user has 4 options available in the Menu as displayed.

- **Close option:** Used to close the app.
- **Search for a device:** Used to scan for the devices.
- **Use default device:** Used to store a default device that can be connected automatically.
- **Preferences:** Provides the user with different options that he can enable.

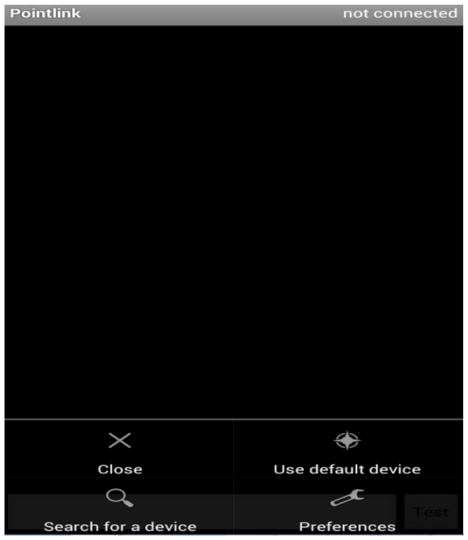
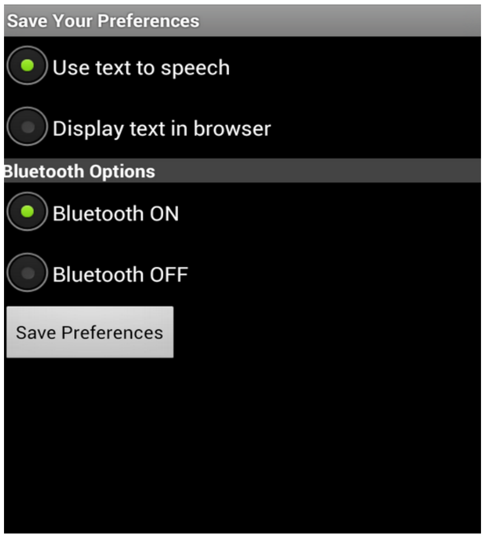


Figure 5.2 Menu Options Display

Preferences Screen: This screen provides the user different options. First, the user can either hear the description of the product or he/she can view it in the browser. Second, the user can use the Bluetooth or run the device without enabling the Bluetooth.


Save Your Preferences

[✓ Use text to speech]
Display text in browser

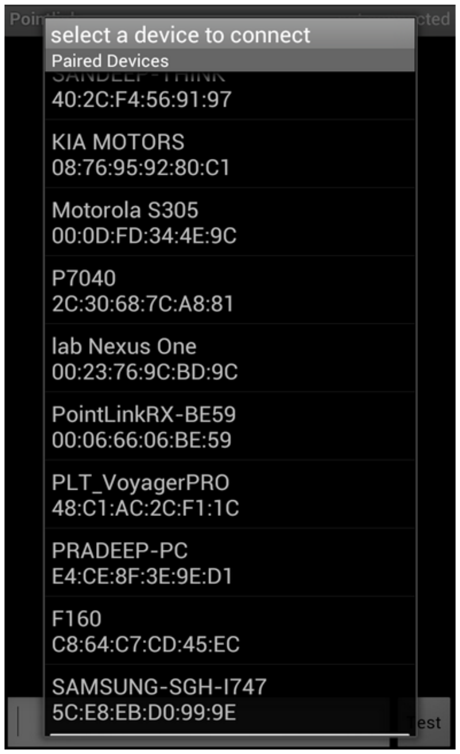
Bluetooth Options

[✓ Bluetooth ON]
Bluetooth OFF

Save Preferences

Figure 5.3 Preferences Screen

Scanning for devices: We scan and select the device with which want to establish the communication.


select a device to connect
Paired Devices

SANDEEP-THINK
40:2C:F4:56:91:97

KIA MOTORS
08:76:95:92:80:C1

Motorola S305
00:0D:FD:34:4E:9C

P7040
2C:30:68:7C:A8:81

lab Nexus One
00:23:76:9C:BD:9C

PointLinkRX-BE59
00:06:66:06:BE:59

PLT_VoyagerPRO
48:C1:AC:2C:F1:1C

PRADEEP-PC
E4:CE:8F:3E:9E:D1

F160
C8:64:C7:CD:45:EC

SAMSUNG-SGH-I747
5C:E8:EB:D0:99:9E

Figure 5.4 Showing list of Bluetooth enabled devices

Selecting the Target device: Once the target device is selected, the user will be asked to pair it.

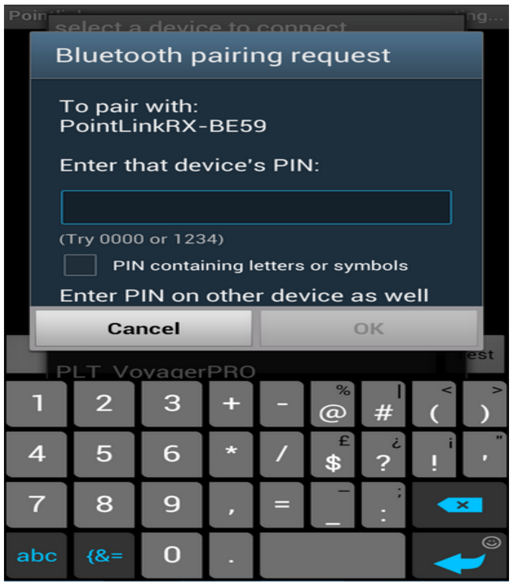
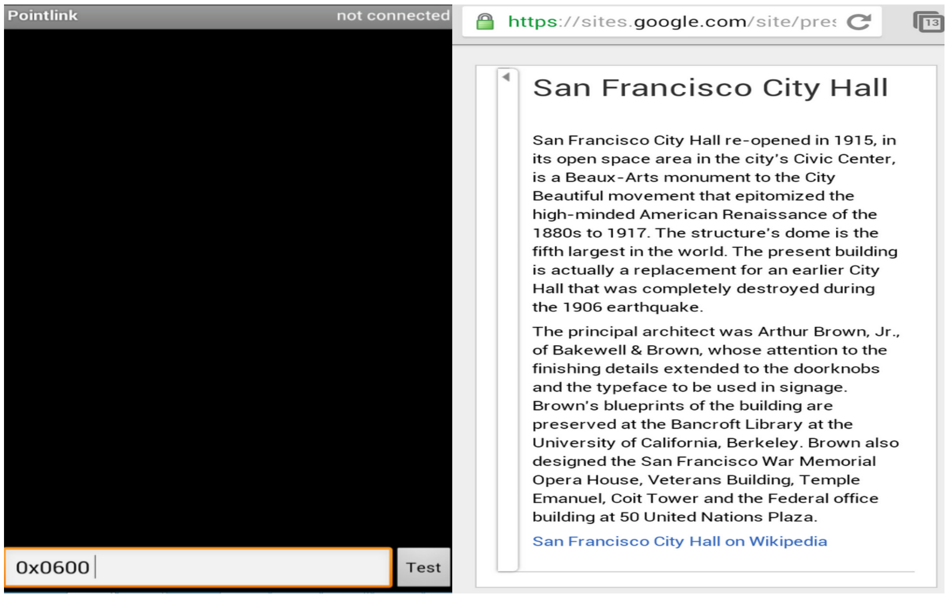

select a device to connect
Bluetooth pairing request
To pair with:
PointLinkRX-BE59
Enter that device's PIN:
(Try 0000 or 1234)
PIN containing letters or symbols
Enter PIN on other device as well
Cancel
OK
PLT VoyagerPRO

Figure 5.5 Pairing Request on the App

Connecting and Listening to the Receiver: The app will be listening to the receiver and when it receives the ID it would either display the text in the browser or it will be read out loud using the text to speech engine.



San Francisco City Hall

San Francisco City Hall re-opened in 1915, in its open space area in the city's Civic Center, is a Beaux-Arts monument to the City Beautiful movement that epitomized the high-minded American Renaissance of the 1880s to 1917. The structure's dome is the fifth largest in the world. The present building is actually a replacement for an earlier City Hall that was completely destroyed during the 1906 earthquake.

The principal architect was Arthur Brown, Jr., of Bakewell & Brown, whose attention to the finishing details extended to the doorknobs and the typeface to be used in signage. Brown's blueprints of the building are preserved at the Bancroft Library at the University of California, Berkeley. Brown also designed the San Francisco War Memorial Opera House, Veterans Building, Temple Emanuel, Coit Tower and the Federal office building at 50 United Nations Plaza.

San Francisco City Hall on Wikipedia

Figure 5.6 Receiving the ID and displaying on the browser

CHAPTER 6

VERBAL ROUTES FRAMEWORK

Introduction

Edwards, Ungar, and Blades (1998) state that the route descriptions provided by sighted people and VI people are different. VI people have longer memory in describing routes compared to that of sighted individuals. The route descriptions provided by the VI include tactile information and hazards like protrusions in the environment, which are not provided by sighted individuals.

Observations in the route descriptions provided by the VI included more information on distance, directions, landmarks, and obstacles along the routes (Bramberg, 1983). From this information we can understand that route descriptions provided for the VI must be different from the descriptions provided for sighted people. The map oriented websites like Google Maps (2013), MapQuest (2013), and YahooMaps (2013) are not useful for VI people because much of the information is presented in visual form.

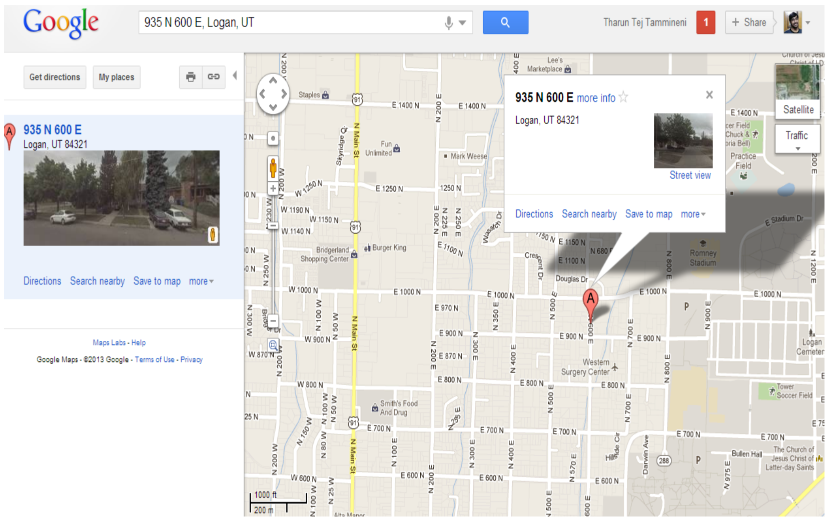
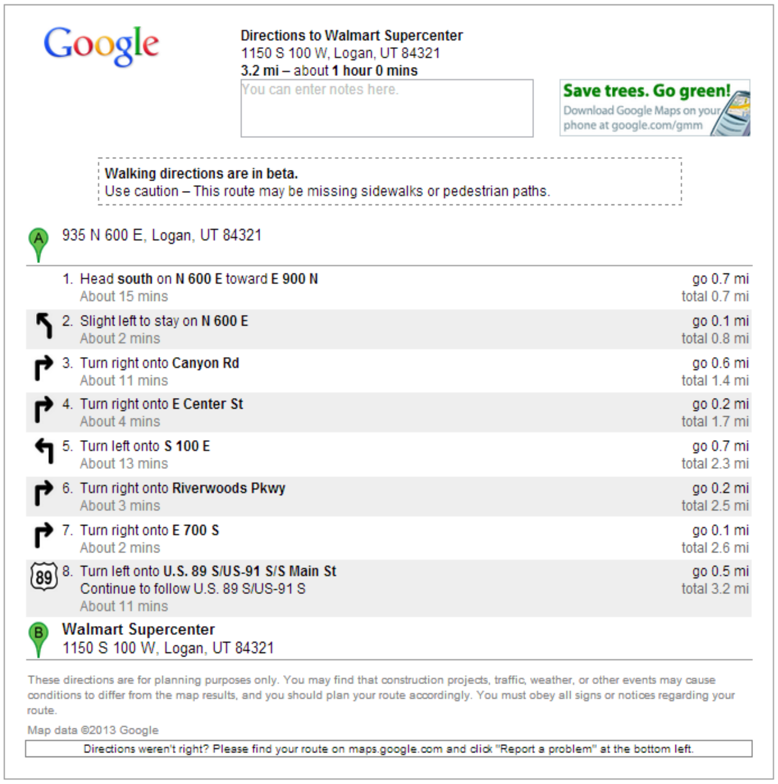


Figure 6.1 Google Maps standard user interface

Figure 6.1 shows the information in the standard user interface, which is not helpful for VI people. A Google map provides walking descriptions for particular routes. These route descriptions are clearly not really useful for the VI because a warning message stating “Use caution–This route may be missing sidewalks or pedestrian paths” is not clearly identified in the text. Other reasons that these tools are not useful for the VI are the lack of designation if the streets are one-way or two-way, lack of the number of intersections that are present, no information if intersections have traffic signals or stop signs, and no description of possible obstacles such as low hanging tree branches on the sidewalk. Moreover, the description does not say when exactly the turn is to be performed, it just has only turns and distances.



Directions to Walmart Supercenter
1150 S 100 W, Logan, UT 84321
3.2 mi – about **1 hour 0 mins**

You can enter notes here.

Save trees. Go green!
Download Google Maps on your phone at google.com/gmm

Walking directions are in beta.
Use caution – This route may be missing sidewalks or pedestrian paths.

935 N 600 E, Logan, UT 84321

1.	Head **south** on **N 600 E** toward **E 900 N** About 15 mins	go 0.7 mi total 0.7 mi
2.	Slight left to stay on **N 600 E** About 2 mins	go 0.1 mi total 0.8 mi
3.	Turn right onto **Canyon Rd** About 11 mins	go 0.6 mi total 1.4 mi
4.	Turn right onto **E Center St** About 4 mins	go 0.2 mi total 1.7 mi
5.	Turn left onto **S 100 E** About 13 mins	go 0.7 mi total 2.3 mi
6.	Turn right onto **Riverwoods Pkwy** About 3 mins	go 0.2 mi total 2.5 mi
7.	Turn right onto **E 700 S** About 2 mins	go 0.1 mi total 2.6 mi
8.	Turn left onto **U.S. 89 S/US-91 S/S Main St** Continue to follow U.S. 89 S/US-91 S About 11 mins	go 0.5 mi total 3.2 mi

Walmart Supercenter
1150 S 100 W, Logan, UT 84321

These directions are for planning purposes only. You may find that construction projects, traffic, weather, or other events may cause conditions to differ from the map results, and you should plan your route accordingly. You must obey all signs or notices regarding your route.

Map data ©2013 Google

Directions weren't right? Please find your route on maps.google.com and click "Report a problem" at the bottom left.

Figure 6.2 Example walking route generated by Google Maps

Volunteered Geographic Information

Volunteered Geographic Information (VGI) was initiated in order to overcome the shortcomings of the commercial mapping services discussed above. It encourages the volunteers to provide data used to build maps, so VGI sites often contain more information compared to others (Goodchild, 2007). The reason is that people living in a particular location will be more familiar with those areas than others. There are different VGI websites that provide professionally created GIS websites. One of them is OpenStreetMap, which provides tools that are used for creating large sets of map data (Haklay & Weber, 2008). These map data sets are similar to Google Maps. In OpenStreetMap it has a facility where a user can edit and add the information regarding landmarks such as sidewalks and buildings.

Wikimapia is a VGI site that takes advantage of user's local GIS knowledge (TerraMetrics, 2013). These maps contain additional information of building names, descriptions, photos etc. Trailpeak is a VGI site that allows users to edit, view and download information related to activities like hiking, biking, and kayaking in the U.S. and Canadian areas (TrailPeak, 2013). The important feature of this site is that users are familiar with the areas as they live or travel so they can upload and edit the information of those trails and be treated as more credible sources. These routes are helpful for the VI people to navigate comfortably.

Framework

From above studies we know that the VI people can use their everyday skills and abilities in order to travel independently if they have clear verbal route descriptions. In order to model the environment as a set of route descriptions, we filter the landmarks from text and build a directed graph of these landmarks. A landmark is defined as a location, object in the environment, building, or intersection that a VI person may mention in his/her route description.

A route description usually has three properties that navigate the person from start to end location, namely starting location, ending location, and a natural language description. In our app we store the natural language descriptions and we break them into sentences for analysis. From these route sentences we extract the landmarks and maintain them in a hierarchy. Each landmark is stored in a table that is associated with the sentence ID. This process of identifying the landmarks in the route sentences is known as auto tagging.

Auto Tagging serves different purposes. First, the users can export landmark tags along with natural language route sentences. Secondly, users having little information of route areas that they are travelling can get more detailed information of the landmarks present in their path. It also gives the sequence of landmarks that occur. Thirdly, we can develop new routes from these landmarks in natural language terms.

Route Analysis Engine

In the Route Analysis Engine we do two things: auto tagging and path interference. To perform auto tagging of landmarks we have used several different natural language processing tools described in Chapter 7. After auto tagging is complete, we infer new routes from the existing routes, which is known as path interference. For example, if we have a route from A to C with B as an intersection, and E to D with B as an intersection, then we can infer new routes from A to E and A to D with B as an intersection.

CHAPTER 7

LANDMARK AUTOTAGGING

The written route descriptions are usually provided in natural language text which can be understood by humans but not handled efficiently by computers. The data should be present in a well-structured format in order for computers to better understand. One of the techniques used to extract meaningful information from unstructured data is Information Extraction (IE) (Cowie & Lehnert, 1996).

Wide research on IE has been carried out over a decade, and now different tools are available that can be used for information extraction. One among them is the Apache OpenNLP, which is a machine learning based toolkit used for processing natural language text (The Apache Software Foundation, 2010). OpenNLP supports different natural language processing tasks such as:

1. Sentence Segmentation
2. Tokenization
3. Part-of- speech
4. Named entity extraction
5. Chunking
6. Parsing
7. Coreference resolution

Sentence Segmentation or Sentence Detection

This detector can detect sentences from a given text. It considers the punctuation character marks and intelligently identifies if the punctuation is really the end of the sentence or not. For Example, it takes care of notations like Mr. and Jan.. Sentence detection is the first step, even before the text is tokenized. In OpenNLP most of the components expect input in the form of segmented sentences.

API Usage:

- **Step 1**: “en-sent.bin” is the model used for sentence detection. It should be loaded first. InputStream modelIn = new FileInputStream("en-sent.bin");
- **Step 2:** Instantiate the SentenceDetectorME. SentenceDetectorME sentenceDetector = new SentenceDetectorME(model);
- **Step 3:** The Sentence Detector can output an array of strings where each string is one sentence. String sentences[] = sentenceDetector.sentDetect(" The route begins at the junction of SR-48 on the rural western end of West Jordan, near copperton. It continues north as a two-lane road and curves northwest past its junction with 7800 South. ");

This would trim out the extra spaces before, between, and after the input string. The sentence array will contain two statements.

Tokenization

The OpenNLP tokenizer divides the input character sequence into tokens. Tokens can be words, punctuation, or numbers. OpenNLP has different tokenizer implementations:

- **Whitespace Tokenizer:** The sequence of words containing non spaces are identified as tokens.
- **Simple Tokenizer:** Tokens are sequences of same character.
- **Learnable Tokenizer:** Detects tokens based on the probability model.

Different OpenNLP components work with text tokenized in a specified manner. We should make sure that tokenization is done in the required format of the components.

API Usage:

- **Step 1:** “en-token.bin” is the model used for sentence tokenization. It should be loaded first.
- **Step 2:** Instantiate Tokenizer. Tokenizer tokenizer = new TokenizerME(model);
- **Step 3:** The tokenizer outputs the array of strings where each string is an token. String tokens[] = tokenizer.tokenize("The highway widens to four lanes."); The output tokens contains “The”, “highway”, “widens”, “to”, ”four”, ”lanes”,”.”

Part-Of-Speech Tagger

The POS tagger identifies the tokens with their respective word type based on the context used and tokens itself. To predict the correct pos tag from the tag set, OpenNLP POS tagger uses a probability model. A token can belong to multiple pos tags depending on the context used and the token. A tag dictionary is used in order to limit the possible tags for a token.

API usage:

- **Step 1:** “en-pos-maxnet.bin” is the model used for POS. We also load the pos model into memory.

  modelIn = new FileInputStream("en-pos-maxent.bin"); POSModel model = new POSModel(modelIn);
- **Step 2:** POSTagger is instantiated.

POSTaggerME tagger = new POSTaggerME(model);

Step 3: The input should be a tokenized sentence represented by array. This will tag the most likely pos tag sequence for a sentence. The tags array will have one part-of-speech tag for each token in the input array.

String sent[] = new String[]{“The”, “highway”, “widens”, “to”, ”four”, ”lanes”,”.”}; String tags[] = tagger.tag(sent)

Named Entity Recognition

The Named Entity Recognition is used to find the named entities and numbers in a given text. It is used to identify the landmarks from the text. The entities can be of different types like name of the person or name of the location, so it has different models that are trained based on the entity type. For example to find the name of a person in the text we use the “en-ner-person.bin” model. To find names of locations we use the “en-ner-location.bin” model. OpenNLP has different pre-trained name finder models.

API Usage:

- **Step 1:** We load the name finder

  InputStream modelIn = new FileInputStream("en-ner-location.bin");

  TokenNameFinderModel model = new TokenNameFinderModel(modelIn);

- **Step 2:** NameFinderME is instantiated.

  NameFinderME nameFinder = new NameFinderME(model);

  Few important things that are to be taken care here are NameFinderME should be called from only one thread as it is not thread safe. The input for this can be of any form like documents, sentences, tokens.

- **Step 3:** nameSpans arrays contains the named entities.

```
String sentence[] = new String[]{ "freeway", "connects","the", "county",
"Salt"  "Lake",  "City"  };
Span nameSpans[] = nameFinder.find(sentence);
```

Problems and Solutions: A problem we noticed here is identifying exact landmark names. For example, we get the input in the form of individual tokens where the set of tokens may form a landmark and an individual token may not. In our case we receive the tokens “Salt,” “Lake,” “City” and we know it is a name of city when combined “Salt Lake City.” However, when the individual token is considered it is not a name of the city.

One approach for addressing this issue is to store these landmarks in a file and load it into the memory and scan for words starting with the prefix of the current word. The data structure that we use to store these landmarks is Trie (Wikimedia Foundation, 2013).

Trie Datastructure: Trie is an ordered tree data structure that stores key values as strings. An important property it possesses is all the descendants of a node will have the same prefix of the string associated with that node and root is an empty string.

We use this data structure in order to store the tokens in each of the node values. Whenever we get a token, we check it against this trie and check if there are any words starting with the same prefix, and if so then we continue to search the next token by combining it with previous. In this way we search until the word is found or it has reached the end of the tree.

Figure 7.1 shows the sample format of the trie data structure we use to store the landmarks. When we get a token we check it against this data structure, and if it returns true then we will combine this current token with its next incoming token and check it in this way.

For example, let the first token be “SAN.” This is checked against the data structure and returns true. Now we will combine this with the next token, “JOSE,” so our search token will be “SAN JOSE.” When we search this combined token we get a true value returned so we will check if it is contained in our database. If so we stop here and continue with other tokens or else we combine it with the next token and search until we get false or the desired word is found.

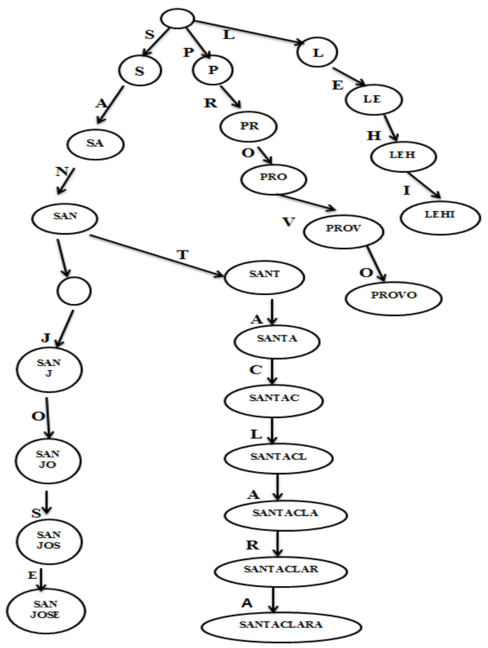


Figure 7.1 Trie data structure representing tokens “SAN JOSE,” “SANTACLARA,” “LEHI,” and “PROVO”

Chunking

In chunking the text is arranged in the form of groups, noun groups, and verb groups.

API Usage:

- **Step 1:** We use “en-chunker.bin” model for chunker. It should be loaded on to memory from disc.

  InputStream modelIn = new FileInputStream("en-chunker.bin");

  ChunkerModel model = new ChunkerModel(modelIn)

- **Step 2:** ChunkerME is instantiated.

  ChunkerME chunker = new ChunkerME(model)

- **Step 3:** Now the chunker is ready to tag data. It has a method called chunk which takes two inputs namely array of Strings from text and array of POS with respect to the words.

  String tag[] = chunker.chunk(sent, pos); sent as an array of strings, pos is an array of parts of speech for each string in sent array.

These are the few apis that we have considered for our purpose. We describe the different algorithms that we have used in the next chapter.

CHAPTER 8

PATH INTERFERENCE

In his study, Golledge (1993) states that VI people see the world in terms of routes. They usually remember the routes as a set of landmarks. VI people may fail to identify that the routes share common areas and landmarks if they are in an unfamiliar area, which makes them travel in longer routes potentially missing short cuts. We aim to solve this problem by generating new routes from the existing set of routes.

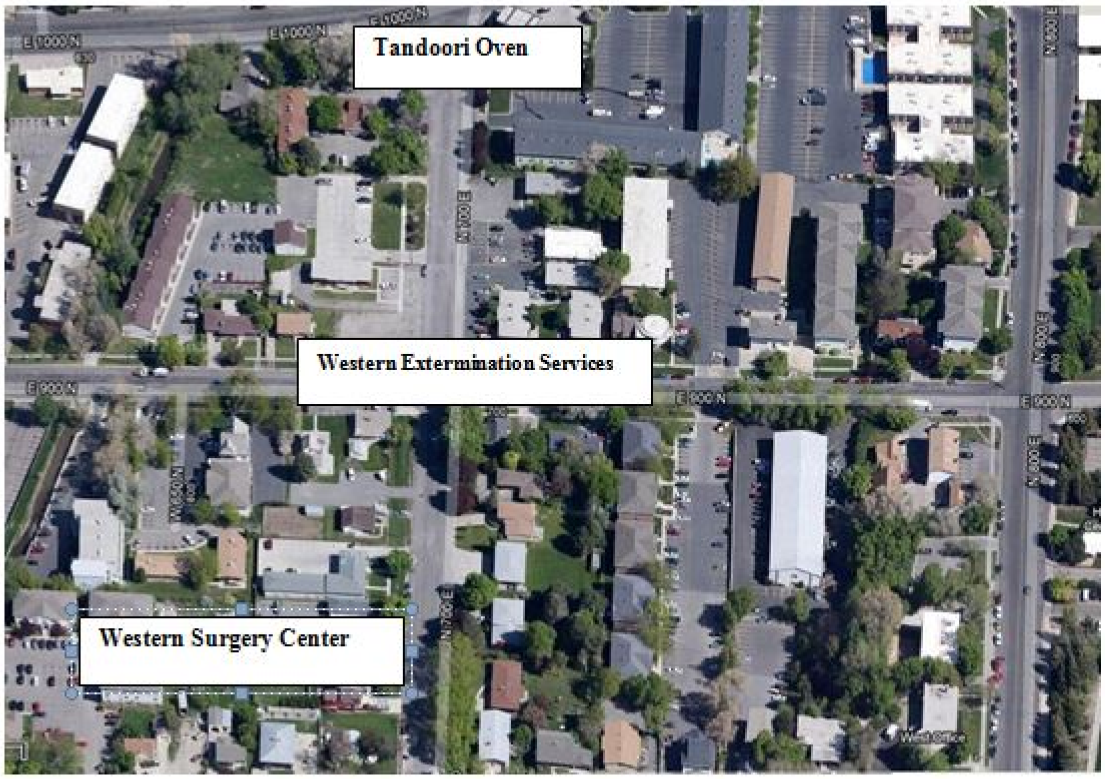


Figure 8.1 Route Map from Tandoori Oven to Western Surgery Center

The Figures 8.1 and 8.2 show the sample route maps of two different start and end locations that sharing the same intersection point. From these two maps we can generate new set of routes. In the following example we will show how the landmarks are extracted, construction of graphs using these landmarks, and traversing and displaying route descriptions for any two given landmarks.

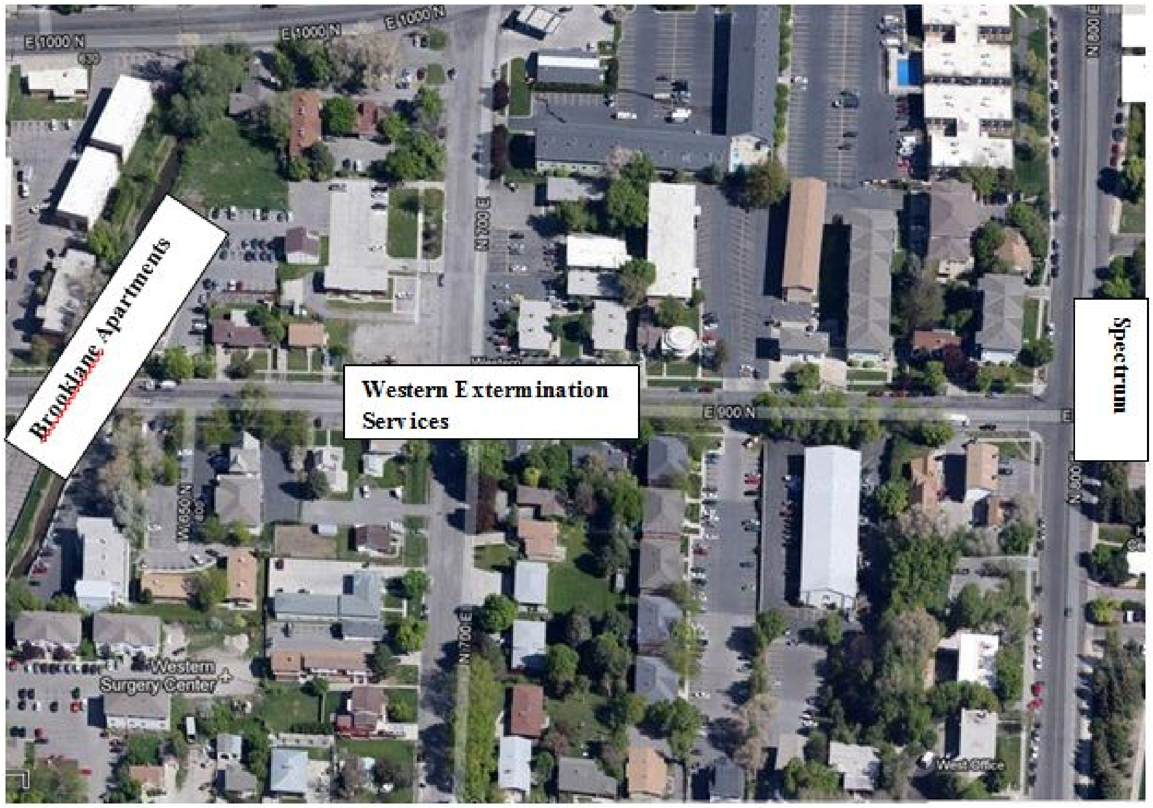


Figure 8.2 Route Map from Brooklane Apartments to Spectrum

Sample Route Description: *SR-152 begins at the four-way intersection of Van Winkle Expressway and 900 East just south of Big Cottonwood Creek (SR-71 turns west from 900 East onto Van Winkle Expressway, which defaults onto 700 East). Immediately to the east, there is an unsignalized three-way intersection (at which SR-152 traffic does not stop) with Murray-Holladay road (former SR-174), which heads straight east (SR-71 intersects the western portion of former SR-174 just west of SR-152's starting point). From there, SR-152 heads southeast as a* *four-lane divided highway with limited at-grade access through an undeveloped, low-lying, and wooded strip of land. For a short distance in this area, the road forms the border between Murray and Millcreek Township, but south of that, SR-152 is always the border between Murray and Holladay. Past an intersection at the terminus of former SR-181 (1300 East), the route dips south-southeast to intersect with 5600 South near Cottonwood High School and a small commercial area. The route once again veers to the southeast before crossing the Jordan and Salt Lake Canal in a low-density residential area (including horse properties). Upon reaching the intersection with Vine Street (former SR-173) and 6100 South (a very short street connecting to the northern section of Highland Drive), the Van Winkle Expressway enters a commercial area and soon turns into (the southern section of) Highland Drive; a short one-way segment of Highland Drive provides an alternate route between the (much wider) northern and southern segments of that street for northbound vehicles. South of the 6100 South intersection, SR-152 has six lanes and sidewalks and loses its wide unpaved median in favor of a center turn lane. The route terminates seven-tenths of a mile later at an interchange with I-21* (The Apache Software Foundation, 2010).

We used Wikipedia as a source for extracting the text. During the extraction of text if any word was hyperlinked, then it is stored in a file in order to check if it is a valid landmark or not. The following are the different stages in path inference:

Stage 1: Landmark Autotagging

- **Step 1:** The text is divided into sentences using the OpenNLP sentence splitter.

- **Step 2:** Each sentence is the given to OpenNLP sentence tokenizer in order to divided the sentence into tokens.
- **Step 3:** Each token is passed to OpenNLP, The NamedEntityRecognition is a module of OpenNLP that it has already trained with set of popular data. If the token matches the trained data then it will be identified as a landmark.
- **Step 4:** If the token is identified as a landmark by the tool then we store it. If the POS tagger identifies it as a noun and tool does not identify it as a landmark then we check this token against our custom build data structure called Trie, in order to verify if this is a valid landmark.
- **Step 5:** If our data structure returns true, then we will store this into our set of landmarks.

The following are the algorithms that are used to store the tokens and search the tokens.

```
        Function INSERT_INTO_TRIE(Token, CurrentNode)
        Input Params:
                    Token: Input string that needs to be stored.
                    CurrentNode: Node pointing to our Trie data structure.
1.      If Token.length = 0 // Empty string.
2.        Set the current word as true.
3.      End If.
4.      for index =0 to Token.length
5.          character = Token.charAt(index)
6.          childNode = character
7.          If childNode = null
8.            childNode = new Trie(character)
9.            Add to current childNodes(childNode, character)
10.         Else
11.           Point childNode to current.
12.         End If Else
13.         If index = Token.length
14.           Set current word as true.
15.         End If
16.     End For
```

Figure 8.3 Algorithm to insert the tokens in to Trie data structure

The above algorithm first checks if the input token is empty or if its length is zero. If it is zero, then it would set the marker as zero. If the input string length is greater than zero then it will repeat for each character by checking if it is present in the child node of the current node. If so, it is set to the child node. If the character is not present then we will create a new node and will point our node to this newly created node. The marker flag to true if we reach the end of the token.

```
Function PRINT_WORDS_MATCHING_PREFIX(Prefix, Node, EntirePrefix)

Input Params:
            Prefix: Current keyword
            Node: Node pointing to our Trie data structure.
            EntirePrefix: Initially it will be an empty string. We use it in our
            recursive calls for sending the word.

1.  If Prefix is not empty
2.    character = Prefix.charAt(0)
3.    TrieNode child = getChildNode(character)
4.    If child is not null
5.      word = EntirePrefix + child
6.      If child is word
7.        Print word
8.      End If
9.      PRINT_WORDS_MATCHING_PREFIX
10.     (prefix.substring(Prefix.indexOf(c)+1), child, word);
11.   End If
12.
13. Else
14.     //denotes reaching end of prefix,
15.     // begin traversing to get matching words.
16.     Map<Character, TrieNode> map = current.getChildNodes();
17.     If map is not null
18.       For each character in map
19.           TrieNode child = getChildNode(character)
20.           If child is not null
21.             word = EntirePrefix +child
22.             If child is word
23.               Print word
24.             End If
25.             PRINT_WORDS_MATCHING_PREFIX
26.             ("", child, word);
27.           End If
28.       End For
29.     End If
30. End If Else
```

Figure 8.4 Algorithm to search for words matching the prefix in Trie data structure

Stage 2: Building a Graph of Landmarks

Once the landmark auto tagging is complete, we build a graph with its landmarks.

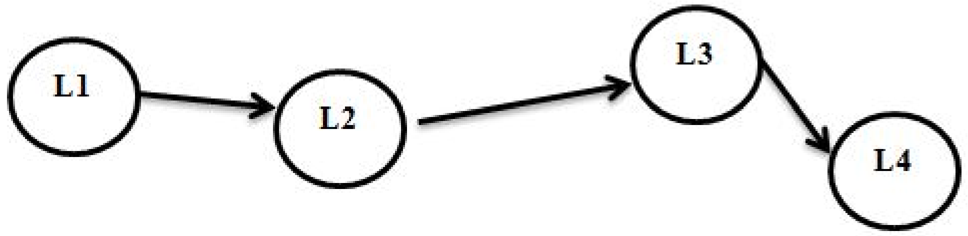


Figure 8.5 Partial Graph representing Landmarks using few set of Landmarks

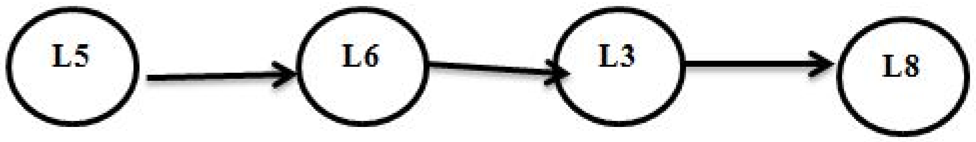


Figure 8.6 Partial Graph representing Landmarks using other few set of Landmarks

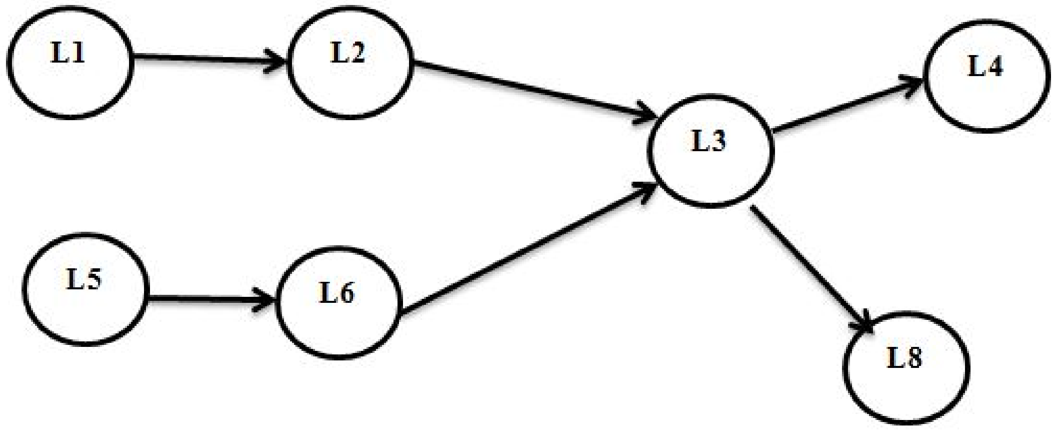


Figure 8.7 Final Graph with intersections representing Landmarks using all Landmarks

For example, if you want to find a route between L5 and L4 then path will be L5$\Rightarrow$ L6 $\Rightarrow$L3$\Rightarrow$L4 after traversing the graph. We can notice that we have inferred a new route from existing set of routes.

Now after performing the above two stages, Landmark Autotagging and graph construction, we now traverse the graph for two given points and find the landmarks that are involved in the path. After this, we will replace this set of landmarks with the corresponding verbal route descriptions from the database.

Stage 3: Traversing the Graph

In this phase, we will traverse the graph with given two landmarks namely start and end landmarks and find the path. As we have the graph represented in the adjacency list we first look for the end landmark in the graph and then we will see what the landmarks that are prior to this are, and for each such landmark we repeat it until we find by placing them in a Queue. Once the source is found then we will give the set of landmarks that are in the path from source to destination.

After the graph traversal is done we will have a list of landmarks that are in the path if route is found. Now for each landmark we extract the corresponding sentence from the database and replace them to form a new verbal route description. In order to clearly understand this example we will go through an example step by step.

```
        Function SAVE_ADJACENCY_LANDMARKS(Ladmarks[])
        Params:
                Landmarks[] : It is an array of landmarks

1.  If Landmarks is not empty and is not null
2.    For index =0 to Landmarks length-1
3.        //query database to check if the current landmark is present already.
4.        List list = query database
5.        If list size is greater than zero
6.          //We will update the particular record.
7.          String existingLandmarks = query database to get landmarks.
8.          If(index+1 < landmarks.length)
9.          existingAdjacencyLandmarks +=","+landmarks[i+1]; //Storing as comma
10.                                                          // separated values
11.         End If
12.         Update the database with above values.
13.       Else
14.           String adjacencyLandmarks = ""
15.           If(index+1 < landmarks.length)
16.             adjacencyLandmarks=landmarks[i+1]
17.           End If
18.           Insert above values to database.
19.       End If Else
20.   End For
21. End If
```

Figure 8.8 Algorithm to store the landmarks as a Adjacency List in the database

```
Function : FIND_PATH(START, DESTINATION)
Params :
    START: Name of the start landmark.
    DESTINATION: Name of the landmark that needs to be reached.

1.  Map<String, String> resultingPathMap
2.  Stack finalLanmarks //That contains all the intermediate landmarks.
3.  Queue landmarksQueue
4.  Boolean Flag
5.  List adjacencyList
6.  List tempRemovedLandmarks
7.  landmarks.enque(START)
8.  while (landmarksQueue is not Empty)
9.          tmpLandmark = landmarksQueue.dequeue()
10.         tempRemovedLandmarks.add(tmpLandmark)
11.         If (tmpLandmark is DESTINATION)
12.             Flag = true
13.             Break
14.         End If
15.         adjacencyList = getAdjacencyMap(tmpLandmark) // from database
16.         For each landmark in adjacencyList
17.             If ( landmark is not present in tempRemovedLandmarks)
18.                 landmarksQueue.enquue(landmark);
19.                 resultingPathMap.add(landmark, tmpLandmark)
20.             End If
21.         End For
22. End while
23. If Flag is true
24.     finalLanmarks = Landmarks from resultingPathMap //Iterate and assign.
25. End If
```

Figure 8.9 Algorithm for graph Traversal between the start and destination landmarks

Example Illustration: Let us take the following route descriptions and see how we can apply the above algorithms.

The route begins at a partial diamond interchange at I-15 on exit 317. The route continues through Bountiful and turns north on Main Street, a two-lane undivided road. The route continues north into Centerville (Wikimedia Foundation, 2013).

Stage 1: Landmark Autotagging: We find the landmarks as described in stage 1 and store them in a database. We break the given text into sentences and pass each sentence in order to identify the landmark.

- Step 1: The route begins at a partial diamond interchange at I-15 on exit 317.

  Landmarks :{ diamond interchange, I-15}
- Step 2: The route continues through Bountiful and turns north on Main Street, a two-lane undivided road.

  Landmarks :{ Bountiful, Main Street}
- Step 3: The route continues north into Centerville

  Landmarks :{ Centerville}

The following are the tables that are used to store the data.

Route_URL	Sentence_ID	Sentence
http://en.wikipedia.org/wiki/Utah_State_Route_131	S1	The route begins at a partial diamond interchange at I-15 on exit 317.
http://en.wikipedia.org/wiki/Utah_State_Route_131	S2	The route continues through Bountiful and turns north on Main Street, a two-lane undivided road.
http://en.wikipedia.org/wiki/Utah_State_Route_131	S3	The route continues north into Centerville.

Table 8.1 Route Sentences Table storing all the sentences

ID	Sentence_ID	Landmark
1	S1	diamond interchange
2	S1	I-15
3	S2	Bountiful
4	S2	Main Street
5	S3	Centerville

Table 8.2 Table containing route landmarks

Stage 2: Graph Construction:

We use the algorithm shown in Figure 8.7 in order to construct and save the graph in the adjacency list. Input will be the array of landmarks for a particular route URL. For example we will take the array of landmark of the route shown above [diamond interchange, I-15, Bountiful, Main Street, Centerville].

Now we clearly know that landmarks I-15, Bountiful, Main Street, Centerville can be reached from the diamond interchange. So we place this in the adjacency list. We similarly do this for each landmark and store it. If the landmark is already in this table then we update the adjacent landmarks list.

ID	**Landmark**	**Adjaceny_Landmarks**
1	Diamond interchange	I-15
2	I-15	Bountiful
3	**Bountiful**	**Main Street, Centerville**
4	Main Street	Centerville

Table 8.3Table storing adjacent landmarks

If we have another list of landmarks such as [Bountiful, Farmington], we know that Bountiful is already present so we just update the adjacency list as shown in the following table.

ID	**Landmark**	**Adjaceny_Landmarks**
1	Diamond interchange	I-15
2	I-15	Bountiful
3	**Bountiful**	**Main Street, Centerville, Farmington**
4	Main Street	Centerville

Table 8.4 Table showing updated list of adjacent landmarks

Stage 3: Graph Traversal: For given any two points we traverse the graph and check if the path exists. If so we will have all the landmarks that are in the path. For each landmark in the path we extract the corresponding sentence and replace it to form a set of verbal route descriptions.

For example, we have Start Landmark be I-15 and End Landmark is Farmington. We will push our start landmark in to a queue. Now the queue has [I-15]. While the queue is not empty we dequeue the Queue and we get value as I-15. We check if this is our destination, and if so we stop and print, if not for each landmark we will add it in to the queue and continue, so the queue will have [Bountiful, Main Street, Centerville]. Similarly we update the queue with Bountiful adjacency list as [Main Street, Centerville, Farmington]. We continue this until we find our destination. The final list of landmarks involved in the path is [I-15, Bountiful, Farmington].

Now for each of the landmarks we extract the corresponding sentence from the Route Sentences table:

I-15: The route begins at a partial diamond interchange at I-15 on exit 317.

Bountiful: The route continues through Bountiful and turns north on Main Street, a two-lane undivided road.

Farmington: As the highway enters Farmington, 200 East becomes State Street. So the final route description will be: The route begins at a partial diamond interchange at I-15 on exit 317. The route continues through Bountiful and turns north on Main Street, a two-lane undivided road. As the highway enters Farmington, 200 East becomes State Street.

CHAPTER 9

ROUTE NAVIGATION ANDROID APP

The current application is paired with the new RIAS transmitter. The application has been tested on Android 2.3+. and Android 4.2. Android 2.2. does not work with the application. The literature and Bluetooth blogs state that Bluetooth is not stable on Android 2.2, which appears to be the case. The application can operate in two modes.

Mode 01

On receiving the transmitter's ID, Mode 01 retrieves the text from a URL associated with the transmitter’s ID. The database has the following schema:

ID	URL
x0900	http://some.url.org

Table 9.1 Database for ID and corresponding URL

When the application retrieves "0x0900" from the RIAS transmitter, the application translates this ID into http://some.url.org, connects to it via Wi-Fi or 3/4G, and retrieves the associated text.

The user can specify how the info is to be displayed. The info can be displayed either via a browser or via a TTS. If the user specifies that the info can be displayed in a browser, then the default browser (this is any Android component that can handle an Intent whose action is set to ACTION_VIEW) displays it. If the user specifies that the info is to be TTSed, then the app splits the text into sentences and starts reading it to the user sentence by sentence. The user can interrupt the TTS stream at any point. The app also handles events, such as other applications popping up or incoming phone calls. If, for example, there is an incoming phone call, the app stops the TTSing. When the user comes back to the app, the TTSing starts from the place where it stopped.

Mode 02

The application has been recently extended to work in simulated mode and generate route descriptions from landmark A to landmark B. The application currently uses a set of Wikipedia route descriptions. Specifically, a database has been created of Wikipedia route descriptions from Utah. Each route description is split into sentences, and each sentence is searched for landmarks. Route descriptions are indexed in terms of landmarks contained in individual sentences.

The databases are turned into a directed connected graph of landmarks. In Mode 02, the user can simulate the event of the Android phone receiving the ID from a RIAS transmitter. It is assumed that each ID is associated with a landmark. To simulate this detection, the user can choose the detected landmark from a list of landmarks.

For example, suppose that this landmark is A. The system then finds all landmarks associated with the detected landmark via a path of landmarks in the constructed graph and presents the user with the list of found landmarks. The user can select the second landmark, which we will call B. The system then generates a route description from the sentence database extracted from the Wikipedia route descriptions and presents it to the user. This description can be either displayed in a browser or TTSed as in Mode 01.

Application Installation Instructions

The following are the steps for installing and running the app. Note that to run this application, you need to have either WiFi Connection or 3/4G connection.

- **Step 1**: Install the **VerbalRouteNavigation.apk**. The easiest way is to email this app as an attachment and then access the email message from your Android phone and press download. If you have an Eclipse IDE with the Android plugin, you can use the DDMS perspective to push the app onto your smartphone.
- **Step 2:** After successful installation of the app in smartphone, click on the VerbalRouteNavigation app icon. You will see a screen with edit text field and Test button. Enter a sample ID 0x0600 in the edit text field and hit Test button.
- **Step 3:** Now you will see a list of destination places that can be reached from the received ID (which represents a particular landmark). Select a landmark from the list. This is the end landmark. Once the end landmark is chosen the route description is displayed and read aloud to the user using the Text to speech Engine.

Database Schema

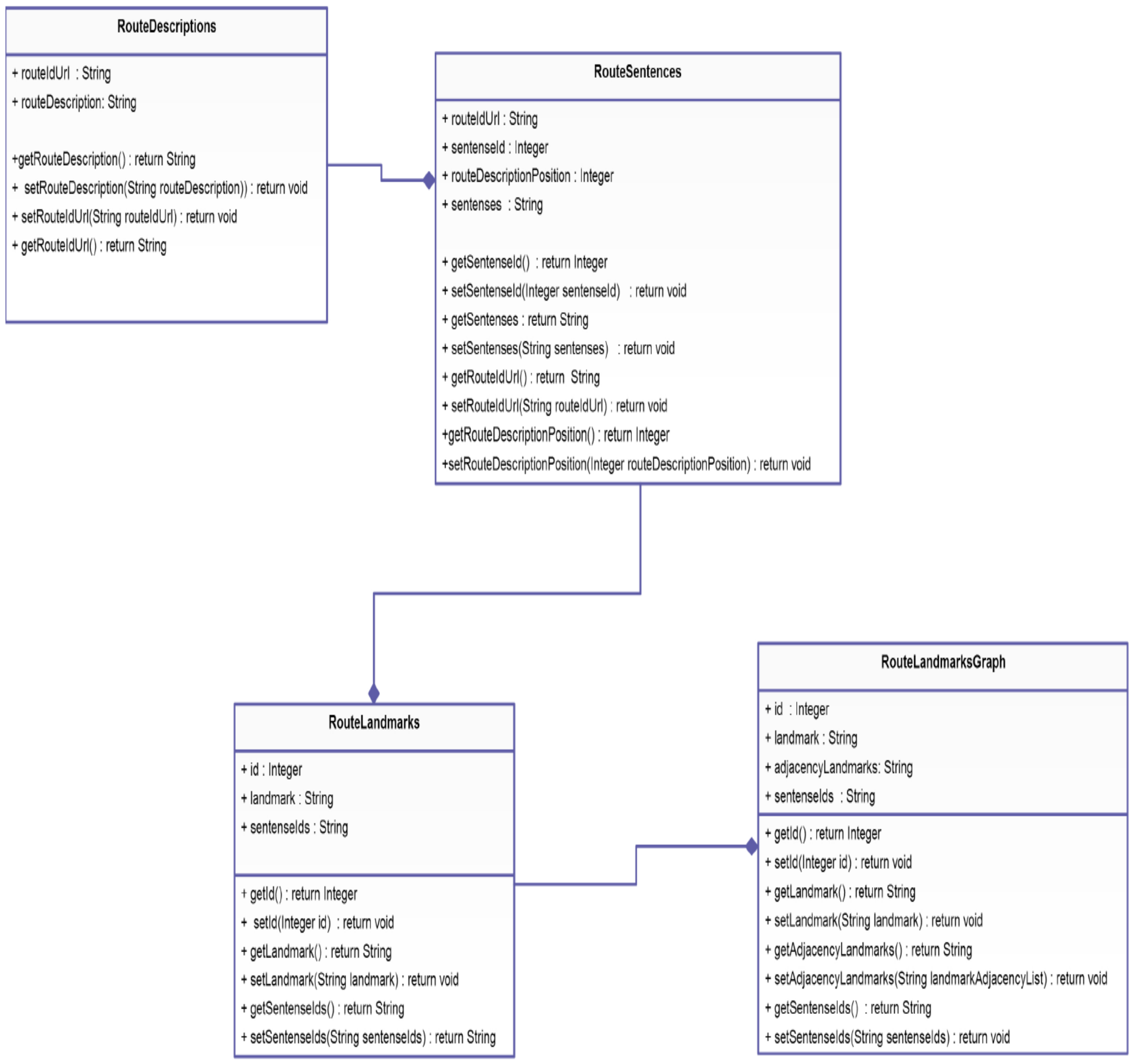

RouteDescriptions
+ routeIdUrl : String
+ routeDescription: String
+getRouteDescription() : return String
+ setRouteDescription(String routeDescription)) : return void
+ setRouteIdUrl(String routeIdUrl) : return void
+ getRouteIdUrl() : return String
RouteSentences
+ routeIdUrl : String
+ sentenseId : Integer
+ routeDescriptionPosition : Integer
+ sentenses : String
+ getSentenseId() : return Integer
+ setSentenseId(Integer sentenseId) : return void
+ getSentenses : return String
+ setSentenses(String sentenses) : return void
+ getRouteIdUrl() : return String
+ setRouteIdUrl(String routeIdUrl) : return void
+getRouteDescriptionPosition() : return Integer
+setRouteDescriptionPosition(Integer routeDescriptionPosition) : return void
RouteLandmarks
+ id : Integer
+ landmark : String
+ sentenseIds : String
+ getId() : return Integer
+ setId(Integer id) : return void
+ getLandmark() : return String
+ setLandmark(String landmark) : return void
+ getSentenseIds() : return String
+ setSentenseIds(String sentenseIds) : return String
RouteLandmarksGraph
+ id : Integer
+ landmark : String
+ adjacencyLandmarks: String
+ sentenseIds : String
+ getId() : return Integer
+ setId(Integer id) : return void
+ getLandmark() : return String
+ setLandmark(String landmark) : return void
+ getAdjacencyLandmarks() : return String
+ setAdjacencyLandmarks(String landmarkAdjacencyList) : return void
+ getSentenseIds() : return String
+ setSentenseIds(String sentenseIds) : return void

Figure 9.1 Database Schema Used

Route Descriptions Table: This table contains a route URL id and route description. The *routeurlId* is the primary key. We have extracted the route descriptions from Wikipedia and have stored it in this table.

Route Sentences Table: For each route description from Route Descriptions table we split the paragraph in to sentences using NLP tool and store it in this table. Auto generated sentenseId field is used as a primary key. This table also includes different attributes namely routeIdUrl (represent the url from where it is extracted), sentenseId (an integer to represent sentese id), routeDescriptionPosition (position of the sentence in the particular route description), sentences (sentence from the description).

Route Landmarks Table: This table is used to store the landmarks that are extracted from the sentences using NLP tool. Each landmark has a unique ID and the sentence ID that is associated with it.

Route Landmarks Graph Table: This table stores the directed graph in the form of adjacency list. For each landmark the adjacent landmarks that can be reached from this particular landmark are stored here.

Android App Execution Technical Details

Assume that we receive an id from a RIAS receiver. The ID has the format 0x0n00, where *n* can range from 1 to 9. For example, 0x0100, 0x0600, etc. This restriction is for demo purposes only. The value of *n* can be arbitrarily large and depends on the number of destinations in the database. For each ID stored in the database there is a corresponding landmark and for each land mark there is a list of target landmarks that can be reached from a given landmark.

Sample Screen 1: Let us say that 0x0200 is the ID received by the App. We know the current location based on this ID. Based on this particular ID received by the smartphone the navigator's current location is determined.

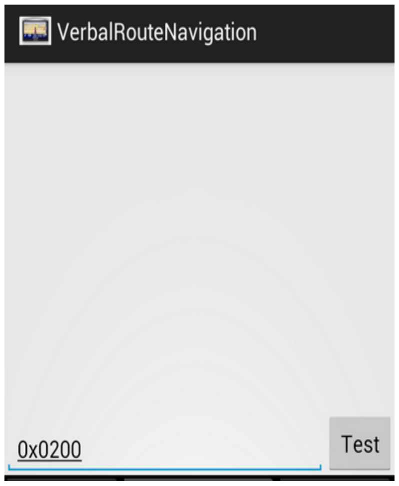


Figure 9.2 App receiving ID

Sample Screen 2: When the Test button is punched based on the current location we will display a list of target locations that can be reached.

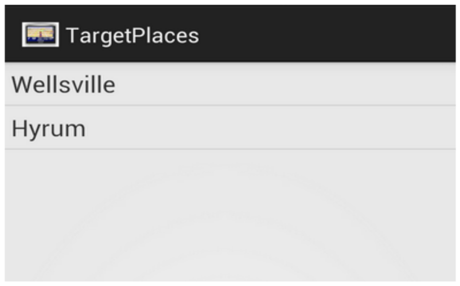


Figure 9.3 Landmarks that can be reached from received ID landmark

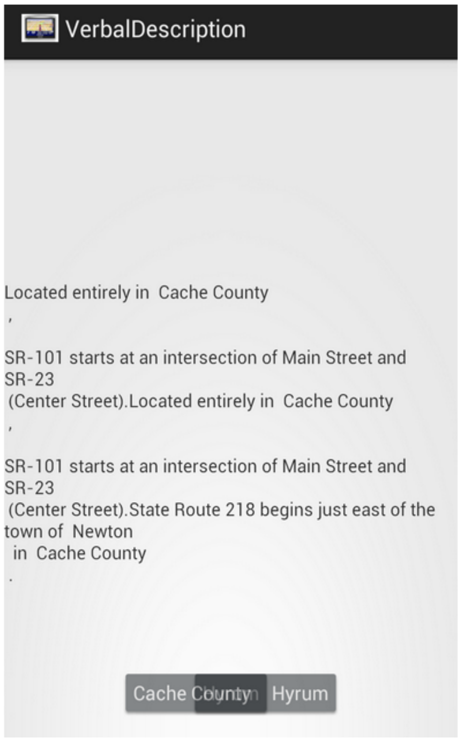
Once the target location is selected a REST GET call is made to a server with two parameters namely **from (location)** and **to (location).**

On receiving this request by the server, we will traverse the graph between **from** location and **to** location using the adjacency list stored in the RouteLandmarksGraph table, which gives the set of landmarks that are involved in the journey.

Then, for each of these landmarks, we will replace it with the verbal sentences that are stored in RouteSentences table and return the route description to the user in the JSON format.

When we receive this route description on client side (Android) we show it to the user and read out using the text to speech engine.

Sample Screen 3: Let us say that the user has selected Hyrum. Now route description from Cache County(current location) to Hyrum (target location) is displayed and read out using Text to speech.



Located entirely in Cache County
 ,

SR-101 starts at an intersection of Main Street and SR-23
 (Center Street).Located entirely in Cache County
 ,

SR-101 starts at an intersection of Main Street and SR-23
 (Center Street).State Route 218 begins just east of the town of Newton
  in Cache County
 .

Figure 9.4 Route description between the start and end lanmakrs

CHAPTER 10

LIMITATIONS AND FUTURE WORK

Active research is ongoing in order to identify the current location of the user. We have used the RIAS transmitter for identifying this, and signal modulation can be a problem at receiving end if two signals collide. If GPS systems get improved this hardware can be eliminated. In verbal route description generation an initial set needs to be completed if there is any addition of new route descriptions. It takes more time because we have to scan every word for identifying landmarks. We are using two procedures for extracting landmarks, but if the tool is mature enough then we can use only one procedure thereby reducing the setup time to some extent. Landmarks are redundant, and while framing the verbal route description from the graph of landmarks we are not sure which sentence should be replaced if there are many. This can be avoided if the environment is small enough to avoid redundant landmarks. This idea can be extended to any complex and dynamic environments. GPS advancements can be used to locate the user’s current location.

CHAPTER 11

CONCLUSION

In this report we first discussed previous research and products built as part of Computer Science Assistive Technology (CSATL). ShopTalk, ShopMobile, and RoboCart are few products built in the CSATL lab. We explored existing map providers that are useful for sighted persons but not as helpful for VI people as they lack some important information represented verbally. These maps also do not have any information regarding the landmarks involved in user paths. We developed a verbal route engine that generates a new set of routes from the existing set, which includes a detailed description of routes and landmarks involved in the path. We implemented this as Software As A Service (SAAS), which is deployed over the cloud and can be used by any client using REST communication. We used OpenNLP tool, as well as our custom procedure, in order to identify the landmarks. This takes some significant amount of time to do the initial setup. We can reduce the setup time only when the tool is mature enough to identify all the landmarks effectively.

REFERENCES

Android. (n.d.) Bluetooth. Retrieved from http://developer.android.com/guide/topics/connectivity/bluetooth.html

The Apache Software Foundation. (2010). Welcome to Apache OpenNLP. Retrieved from http://en.wikipedia.org/wiki/Utah_State_Route_152

Bramberg, M. (1983). Language and geographich orientation for the blind. In Speech, Place, and Action, R. J. Jarvella and W. Klein, Eds. John Wiley and Sons Ltd., p.203-218

Brooks, C.H., & Montanez, N. (2006). Improved annotation of the blogosphere via autotagging and hierarchical clustering. *Proceeding of the 15th International Conference on World Wide Web, ACM,* (pp.625-632).

Cowie, J., & Lehnert, W. (1996). Information extraction. *Communications of the ACM 39, 1*, 80-91.

Crandall, W., Bentzen, B. L., Myers, L., & Brabyn, J. (2001). New orientation and accessibility option for persons with visual impairments: Transportation applications for remote infrared audible signage.

Edwards, R., Ungar, S., & Blades, M. (1998). Route descriptions by visually impaired and sighted children from memory and from maps. *Journal of Visual Impairment and Blindness 92*(7), 512-521.

En-Vision America. (2013). i.d. mate Quest. Retrieved from http://www.envisionamerica.com/products/idmate/

Food Marketing Institute Research Department. (2006). The food retailing industry speaks: Annual state of the industry review.

Fox, D. (1998). *Markov localization: A probabilistic framework for mobile robot localization and navigation* (Doctoral dissertation). University of Bonn.

Golledge, R. G. (1993). Geography and the disabled: A survey with special reference to vision impaired and blind populations. *Transactions of the Institute of Britis Geographers, 18*, 63-65.

Google. (2013). Google Maps. Retrieved April 2013 from https://maps.google.com/

Goodchild, M. F. (2007). Citizens as voluntary sensors: Spatial data infrastructure in the World of Web 2.0. *International Journal of Spatial Data Infrastructures Resources, 2*, 24-32.

Haklay, M., & Weber, P. (2008). OpenStreetMap: User-generated street maps. *IEEE Pervasive Computer, 7*(4), 12-8.

Helal, A. S., Moore, S. E., & Ramachandran, B. (2001). Drishti: An integrated navigation system for visually impaired and disabled. *Proceedings of the 5th IEEE International symposium on Wearable Computers* (pp. 149). Washington, D.C.

Kulyukin, V., Gharpure, C., & Pentico, C. (2007). Robots as interfaces to haptic and locomotor spaces. *Proceedings of the ACM Conference on Human-Robot Interaction (HRI 2007)* (pp. 325-331). Washington, D.C.

Kulyukin, V. & Kutiyanawala, A. (2010). From ShopTalk to ShopMobile: Vision-based barcode scanning with mobile phones for independent blind grocery shopping.

Kulyukin, V. & Nicholson, J. (2012). Toward blind travel support through verbal route directions: A path inference algorithm for inferring new route descriptions from existing route directions.

LaPlante, M. P. & Carlson, D. (2000). Disability in the United States: Prevalence and causes. Washington, DC: U.S. Department of Education.

Lynch, K. (1960). The Image of the City. Cambridge MA: MIT Press.

MapQuest. (2013). MapQuest. Retrieved April 2013 from http://www.mapquest.com/

Nicholson, J., Kulyukin, V., & Coster, D. (2009). ShopTalk: Independent blind shopping through verbal route directions and barcode scans. *Open Rehabilitation Journal, 2*, 11-23.

TerraMetrics. (2013). Wikimapia. Retrieved April 2013 from http://wikimapia.org/

T. L., LLC. (2009). Talking lights systems. Retrieved from http://www.talking-lights.com

TrailPeak. (2013). TrailPeak. Retrieved April 2013 from http://www.trailpeak.com/

Wikimedia Foundation. (2013). Trie. Retrieved from http://en.wikipedia.org/wiki/Trie

Wikimedia Foundation. (2013). Utah State route 131. Retrieved from http://en.wikipedia.org/wiki/Utah_State_Route_131

World Health Organization. (2012). Visual impairment and blindness: Fact sheet no 282. Retrieved from http://www.who.int/mediacentre/factsheets/fs282/en/index.html

Yahoo. (2013). YahooMaps. Retrieved April 2013 from http://maps.yahoo.com/