CHALMERS

Framework for Smartphone Pedestrian Navigation by Augmentation of Navigation Instructions on Camera Snapshots

Master of Science Thesis

JENNY HELLMAN
CECILIA VIDLUND

Department of Signals and Systems
Signal Processing Group
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2011
Report No. Ex041/2011

REPORT NO. 2011:041

Framework for Smartphone Pedestrian Navigation by Augmentation of Navigation Instructions on Camera Snapshots

JENNY HELLMAN
CECILIA VIDLUND

Department of Signals and Systems
Signal Processing Group
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden June 9, 2011

Framework for Smartphone Pedestrian Navigation by Augmentation of Navigation Instructions on Camera Snapshots

Jenny Hellman
Cecilia Vidlund

Supervisor at Appello Systems AB: Srdan Kvrdic

Examiner: Irene Gu

Technical report no. 2011:041

Department of Signals and Systems
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Cover: Navigation instruction displayed on smartphone camera snapshot. The picture of the Samsung Galaxy S is a press image from Samsung and is used with their permission.

Teknologtryck
Göteborg, Sweden 2011

Abstract

GPS navigation systems suitable for cars are well presented on the market but the supply is rather poor when it comes to pedestrians. Cars travels in relatively high speed compared to pedestrians and therefore the requirements on sensor accuracy is lower. In this thesis a platform independent pedestrian navigation software library for usage on smartphones in an urban environment has been developed. The GUI was implemented in Android and navigation instructions are overlayed in the smartphone's camera view. From the developed application a picture is snapped of the user's surroundings, image processing methods such as Canny edge detector and Hough transform are applied for extracting information out of the snapshot's content. Map data, which is requested of surrounding streets based on the smartphone's GPS position, is matched with the snapshot content. The outcome is a resulting navigation instruction. The algorithm is applicable, however, obstacles and shadows within snapshots can result in faulty lines which has influence on the outcome. An accurate GPS position is crucial for retrieving reliable navigation instructions. Hence, there are room for improvements. This version functions best in narrow streets where no interferences of large obstacles or shadows are present.

Acknowledgements

The authors would like to thank our supervisor at Appello Systems AB, Srdan Kvrdic, for his guidance and support during the project. We are also grateful to all other employees at Appello Systems AB for their help with tricky problems as well as their company which we have valued during the thesis work.

We would also like to thank our supervisors at Chalmers, Irene Gu and Mats Viberg, for their valuable advice and feedback.

The thesis work has been carried out at Appello Systems AB in Göteborg, Sweden.

List of Abbreviations

AR	Augmented Reality
DDS	Drill Down Server
GPS	Global Positioning System
GUI	Graphical User Interface
HL	Hough Line
HT	Hough Transform
OS	Operating System
VM	Virtual Machine

List of Figures

List of Tables

Contents

Chapter 1

Introduction

GPS navigation systems suitable for car navigation are already well presented on the market. An example of such a system is Wisepilot™, developed by Appello System AB which is the company where this thesis is executed. When traveling by car, the traveling speed is high leading to low requirements of the system's accuracy. Compared to car navigation, a GPS navigation system aimed for pedestrians require higher accuracy. This is because the distance between two navigation points in an urban environment is often much shorter than between corresponding navigation points for car navigation. A shorter distance along with a relatively low traveling speed demands low margins of errors. Also, a pedestrian navigation device must be portable and user-friendly for a stressed situation. The exploited usage of smartphones laid a suitable foundation of an application aimed for this purpose, fulfilling the portable demand as well as being a device, conveniently as it is, already in the possession of many potential users.

A well suited concept for pedestrian navigation system could be Augmented Reality (AR). Using AR, navigation instructions could be displayed on a real-time display such as a smartphone's camera view. The user would only have to hold up their smartphone and the route directives would be overlaid on the "real" surroundings seen through the camera view. However, AR takes up a lot of computational power, resulting in long processing time on smartphones, and is thus not suitable for this thesis. Instead, a simplification of AR is developed. The user snaps a picture with a smartphone's camera and gets navigation instructions overlaid on top of the snapshot. Whenever the user is in need of new directives, a new picture is snapped.

Certain factors must be considered during the development of such a pedestrian navigation system. The system must be robust for diversity in street

qualities and weathers though a street can be made of asphalt, paving stones, pebbles or be covered in snow. The application has to work properly even with crowded streets resulting in lost information.

The objective of this thesis is to develop and implement a platform independent framework for smartphone pedestrian navigation for usage in an urban environment. The framework will be implemented in Java and demonstrated in an Android application where navigation instructions are augmented on a snapshot.

The step from snapping a picture to receiving navigation instructions is handled by the framework. No trust is put in the smartphone's compass, therefore information in the snapshot must be extracted so it can be matched with the user's surroundings for determining their correct direction. The data of the surroundings is invoked from Wisepilot$^{\text{TM}}$ where a request is made for navigation data, consisting of both map and route data. The request is based on the user's GPS position and destination target. With obtained data, the streets located in the vicinity of the user are considered known along with the route leading to the final destination. Streets constitute intersections in which routes change their paths, hence, the essential information to be extracted from the snapshot is thus: is there an intersection present and if so, how far away is it located. The result is matched with received map data and as a final step of the algorithm a route directive is overlaid on the snapshot.

Chapter 2

Background

In this chapter the background theory of used concepts and adopted methods are presented.

2.1 Android Platform

Android is an open source Operating System (OS) based on a modified Linux Kernel [1]. It was created by the Open Handset Alliance which was originally established by 34 companies including among others Google, HTC and Dell [1]. The OS itself is written in a mixture of C and C++, although the android applications are mainly written in Java. The applications are run on a Dalvik Virtual Machine (VM) which converts java classes to a format of its own (.dex).

Android was released in 2008, but it was not until the beginning of 2009 the first android controlled mobile phone was available on the market. Since then, several new updates have been released and the most recent one is Android 3.1 which was released for developers in May 2011 [2].

2.2 Global Positioning System

The Global Positioning System (GPS) was developed by the U.S. Department of Defence in 1978 [3] with the purpose of reinforcing the U.S. and its allies military forces [4]. GPS is a satellite system consisting of 24 different units which are continuously sending out one-way signals of their position and time. These signals are accessible for any GPS receiver and based on the transmitted information its current location can be determined on a three-dimensional form; latitude, longitude and altitude.

2.3 Canny Edge Detection

John F. Canny developed and gave name to a computationally approach to edge detection in the year 1986 [5]. By detecting edges in an image, structural information can be kept and redundant information discarded. The image of edges, which from this point onwards will be called edge image, is made binary (black and white) which reduces the amount of data to process drastically.

Edges are characterized by high 1st and 2nd derivatives [6] and there are several existing methods taking advantages of these qualities for extracting edges. Such methods is e.g. the Gradient filter detector which make use of the 1:st derivative or the Laplacian filter detector which make use of the 2nd derivative. The Canny edge detector uses both and was chosen because of its high performance against the simpler filtering methods.

The Canny method is developed based on three basic objectives [6]:

1. Low error rate - true edges should be found, false edges should not be returned from the detector.

2. Edge points well localized - found edges should be as close as possible to true edges.

3. Single edge point response - only one response to one single edge.

Finding unequivocal solutions for all these objectives was found to be unfeasible [6]. Hence, all three are considered and set up with mathematically equations and an optimized solution is numerically searched for.

The first step of the Canny edge detection procedure is to smooth the image by a 2D Gaussian filter [6] according to

$$G(x_f,y_f) = e^{-\frac{x_f^2+y_f^2}{2\sigma^2}}, \tag{2.1}$$

where x_f and y_f represent the 2D filter cells and σ represent the standard deviation. The smoothing filter enhances high intensity transitions in contrasts. The input image, $I(x,y)$, is convolved with the smoothing filter $G(x_f,y_f)$ resulting in a smoothed image I_s, see equation 2.2.

$$I_s(x,y) = G(x,y) \star I(x,y) \tag{2.2}$$

The gradient and its magnitude is calculated for each image pixel of the smoothed image according to equations 2.3 and 2.4 respectively [6].

$$\alpha(x,y) = tan^{-1}\left[\frac{\left(\frac{\partial I_s}{\partial y}\right)}{\left(\frac{\partial I_s}{\partial x}\right)}\right] \tag{2.3}$$

$$M(x,y) = \sqrt{\left(\frac{\partial I_s}{\partial x}\right)^2 + \left(\frac{\partial I_s}{\partial y}\right)^2} \tag{2.4}$$

Regions with high spatial derivatives are used for nonmaxima suppression, which means that pixels with low gradient magnitude are not interpreted as true edge points and thus suppressed, see equation 2.5.

$$g_N(x,y) = I_s(x,y) > M_{maxima\ suppression\ threshold} \tag{2.5}$$

The magnitude of the remaining unsuppressed pixels are manipulated based on two threshold-values, a high threshold T_H and a low threshold T_L [6]. Each pixel in $g_N(x,y)$ is compared to both thresholds resulting in two fresh binary (black and white) matrixes. Their pixel values becomes `'white'` if condition 2.6 or 2.7 is fulfilled and `'black'` if not.

$$g_{NH}(x,y) = g_N(x,y) \geq T_H \tag{2.6}$$

$$g_{NL}(x,y) = g_N(x,y) \geq T_L \tag{2.7}$$

The resulting differential matrix becomes the output image, I_{out}, of the Canny edge detector, according to equation 2.8.

$$I_{out} = g_{NL}(x,y) - g_{NH}(x,y) \tag{2.8}$$

2.4 The Hough Transform

The Hough transform (HT) was developed by Paul Hough in the year 1962 [6]. It is a method for detecting lines in an edge image $I(x,y)$. The principle of HT is to find the linear equation parameters, k and m according to equation 2.9, for each straight line representing a distinct edge. To achieve this, a transformation from Cartesian space (x,y) to parameter space (k,m) is made for all edge points.

$$y_i = kx_i + m, \tag{2.9}$$

Since a single edge point lacks a slope k a voting system is introduced to evaluate at which slopes and where edge points lies in straight lines [6]. The voting system ranks parameter pairs (k,m) for lines in a accumulator matrix $A(k,m)$. $A(k,m)$ is ranked high if there are many edge points detected in the corresponding line. A high ranking in turn indicate that those parameters are more likely to be a line representing a distinct edge.

In practice, each detected edge point (x_i,y_i) is evaluated. No *a priori* knowledge of an image's content is assumed and therefore lines in all possible directions are searched for. This is done by dividing the total range of all possible slopes in discrete steps, Δk, and then determining the matching Δm for each of these steps by equation 2.10 [6].

$$\Delta m = -\Delta k x_i + y_i, \tag{2.10}$$

For each parameter pair $(\Delta k,\Delta m)$ found, the corresponding cell in the accumulator matrix, $A(\Delta k,\Delta m)$, is increased by one.

When the parameters for all edge points have been determined, lines in the edge image are detected by large values in $A(k,m)$ [6]. This is since edge points which are aligned with each other share parameter pairs k,m but differs in $x-$ and $y-$position. Therefore, their cell in A will be increased a larger number of times than for edge points not being aligned and thus not sharing parameters.

However, there is an impractical issue when using the linear equation 2.9 which is that vertical lines cannot be found since k approaches infinity for this slope. Therefore, lines are represented in polar coordinates [7] instead according to equation 2.11.

$$r = xcos(\theta) + ysin(\theta) \tag{2.11}$$

The transformation from Cartesian space to parameter space follows the same principle described above with the difference that parameter space is constituted by θ and r (illustrated in figure 2.1) instead of k and m. So, each point is evaluated according to equation 2.11 where discrete steps of θ is applied resulting in parameter pairs $(\Delta\theta, \Delta r)$.

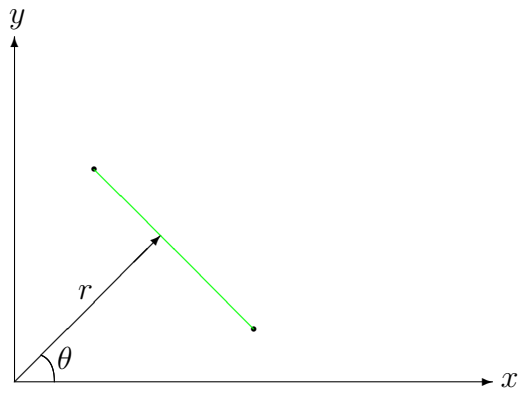


Figure 2.1: Polar parameters illustrated in Cartesian space.

Chapter 3

Implementation Procedure

This chapter describes the approach for developing the main framework as well as getting a final usable application. The chapter is divided into four sections: Camera and sensors, Image processing, Analysis of snapshot properties and Integration with Wisepilot^{TM}.

The different steps of the application's procedure are listed in the block diagram illustrated in figure 3.1. The main GUI of Wisepilot^{TM} is first displayed when starting the application. The user selects a destination target leading to a calculated route while the camera view is displayed with a choice of snapping a picture. The picture is snapped on a street in the surroundings and the resulting snapshot is processed as explained in section 3.2. The outcome is analyzed and classified into a snapshot type, see section 3.3. This type as well as the distance to a potential intersection are used as inputs in the matching procedure with information from Wisepilot^{TM} as is explained in section 3.4.

The result from the matching procedure determines the final navigation instruction. The instruction is displayed as an overlayed arrow on top of the original snapshot in the application.

3.1 Camera and sensors

The smartphone's hardware used when implementing the application are the camera, the orientation sensor and the GPS. The implementation of these hardware is done using Android Native.

The snapshot obtained from the camera is scaled to 480×640 pixels and

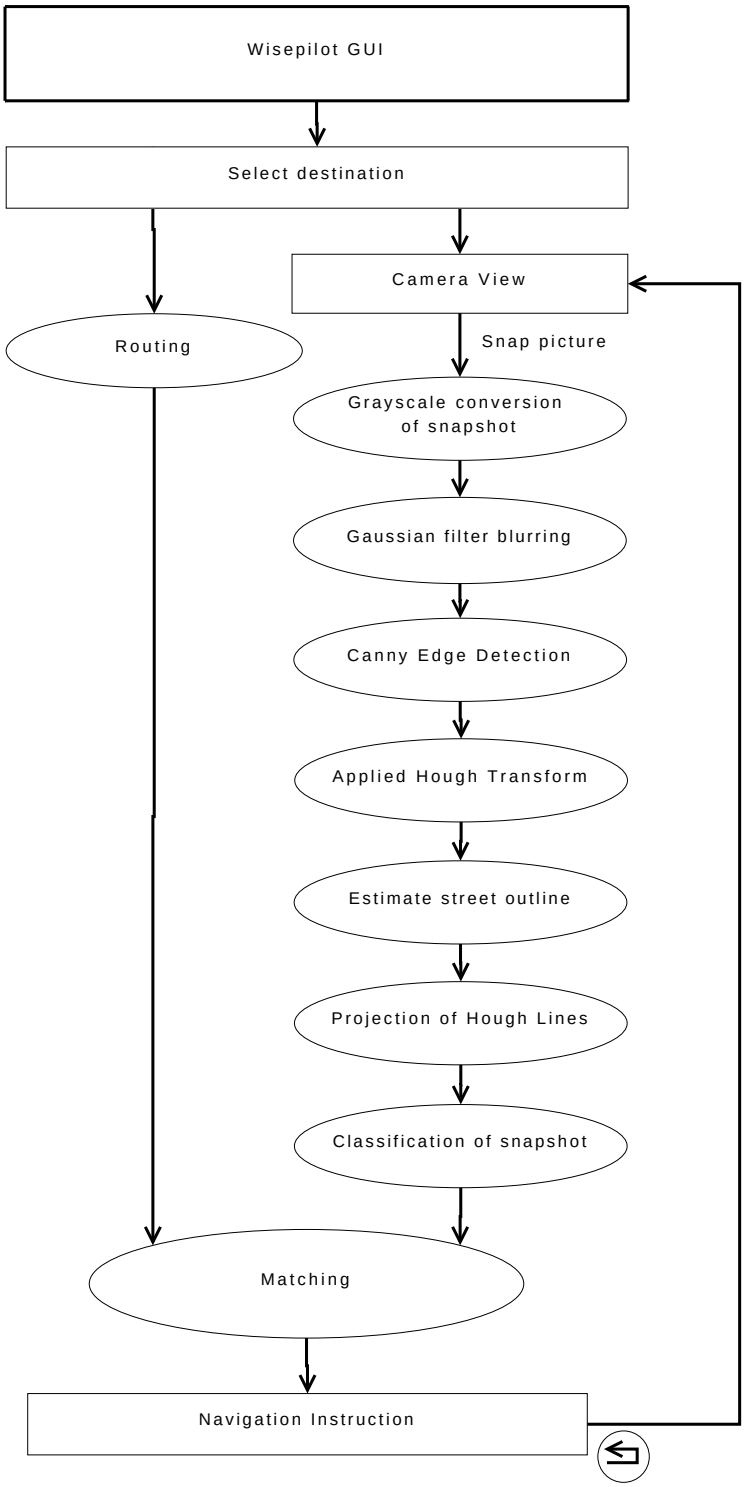


Figure 3.1: Block diagram describing the structure of the developed application.

delivered as an RGB bitmap. The snapshot size was selected out of a number of predetermined available sizes because of its relatively small size leading to a fast application. The camera has a vertical and a horizontal angle of view. The vertical angle of view is used when estimating distances in the snapshot.

The orientation sensor contains information about how the smartphone is held. It consist of a roll angle, a pitch angle and a compass direction, the roll and pitch are illustrated in figure 3.2. Roll is exploited in the distance estimation and is the only orientation parameter used in this thesis. A direction indication from the compass could be valuable for matching purposes with data from Wisepilot^{TM}. However, the built in compass is unreliable and does not have an implemented 3D mode which making it unusable in this application.

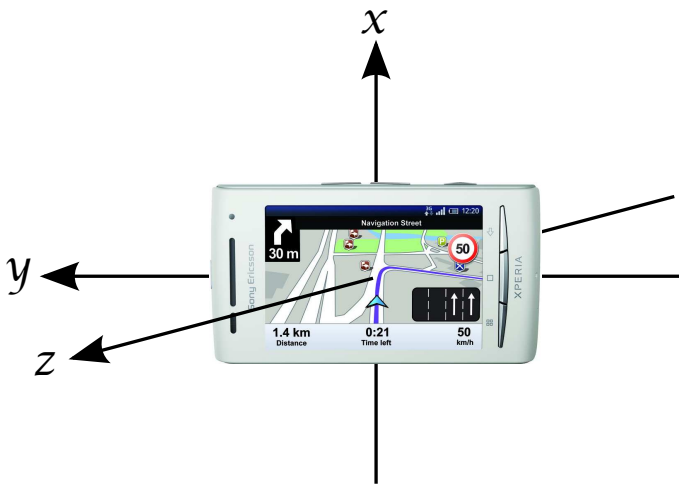


Figure 3.2: Adviced smartphone orientation according to navigation instructions described in Appendix A. The definition of the orientation angles θ_{pitch} and θ_{roll} is as quoted: "Pitch, rotation around x-axis (-180 to 180), with positive values when the z-axis moves toward the y-axis. Roll, rotation around y-axis (-90 to 90), with positive values when the x-axis moves toward the z-axis." [8]

The position of the user is obtained by the built-in GPS receiver. The acquired parameters are latitude, longitude and altitude.

3.2 Image processing

The snapshot is processed with the aim of enhancing street properties. In an urban environment, typical such properties are represented by edges forming perspective mode lines along with windows and at transitions from street to buildings. For this purpose it is suitable to apply an edge detecting method and the Canny edge detector was chosen due to its superior outcome compared to simpler methods. Detected edges are then converted into a more suitable type of data, i.e. lines. A reliable method for extracting lines out of edges is to apply Hough transform (HT) which thus is used. The image processing is made on a gray-scale transformation of the delivered snapshot.

This section describes the developed algorithm for enhancing street properties. Its input is the gray-scale image and its output the outcome of applied HT, i.e. Hough lines (HLs). HT is applied twice in the algorithm, once on the entire image and once on the area represented by cells 4 and 5 in figure 3.3. These cells contain information of the most distant areas in the snapshot and are handled separately for not being overlooked[1].

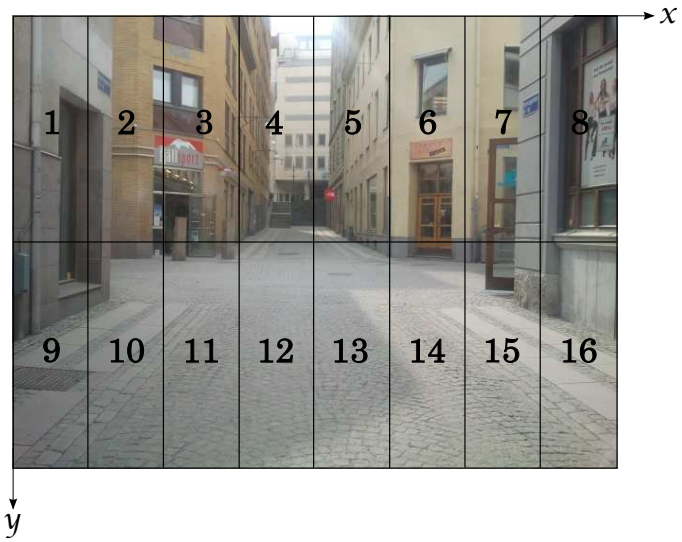


Figure 3.3: Image divided into cells. The most distant area of a snapshot is represented by cells 4 and 5 if the snapshot is taken as advised in Appendix A.

[1]The HLs found in cells 4 and 5 (figure 3.3) are often shorter than HLs found in other cells and will thus not be ranked as high.

3.2.1 Gaussian Filtering

The edges representing street properties are denoted by high intensity transitions. To find these transitions the gray-scale image is blurred by filtering with a 2D Gaussian formed spatial filter. Before the filter is applied the image is mirror padded to remove border influences in intensity transitions [6]. Filter values are determined by the Gaussian distribution and normalized according to

$$f(x,y) = \frac{1}{\sum_{x,y} e^{(-\frac{x^2+y^2}{2\pi\sigma^2})}} e^{(-\frac{x^2+y^2}{2\pi\sigma^2})} \tag{3.1}$$

where x and y represent the 2D filter cells and σ represent the standard deviation. The filter is applied on the padded image and results in a blurred image with well highlighted edges, see figure 3.4.

Figure 3.4: Resulting image of blurring with a Gaussian filter.

3.2.2 Canny Detection

The Canny edge detection method is implemented according to the procedure described in section 2.3 but with one exception: The original method handles an upper and a lower threshold concerning the selection of edges, this implementation only considers the higher threshold, T_H. It is applied on the resulting Gaussian blurred image from the previous section.

The value of T_H is determined by

$$T_H = \frac{CUSUM(H_M) < (A_{non\ edges})}{I} \tag{3.2}$$

where $A_{non\ edges}$ is the number of non edge point pixels detected from the blurred image, H_M is a histogram of the magnitudes calculated by equation 2.4, I is the number of intervals in H_M and CUSUM represents the cumulative mass function.

Edges detected with this implementation when the blurred image in figure 3.4 is used as input are illustrated in figure 3.5.

Figure 3.5: Edges detected by implemented Canny edge detector.

Even though edges are clearly visible to human eye the following HT encounters problems with edges of one pixel width. Therefore, all edges are thickened as illustrated in figure 3.6. Before the thickening is done single edge pixels are removed. The outcome of this Canny edge detector is a binary (black and white) image.

3.2.3 Hough Transform

The implemented HT is slightly modified from the one presented in section 2.4. Extracted lines have been limited to regions where edge points actually

Figure 3.6: Thickened edges.

are detected. These lines constitute HLs and they represent street and building features in an image. Figure 3.7 illustrate the HLs extracted from the thickened edge image in figure 3.6.

Figure 3.7: Extracted Hough lines by implemented Hough transform.

A special case is introduced for the area represented by cells 4 and 5 (see figure 3.3) on which the HT is applied separately. Extracted HLs from this

special case is from this point and onwards called distant HLs. The resulting distant HLs from the same image (figure 3.6) are illustrated in figure 3.8.

Figure 3.8: Extracted Hough lines in distant area.

The ranges for the accumulator array, $A(\theta, r)$, are $0° \leq \theta < 180°$ and $0 \leq r < 2D$, where D is the diagonal size in pixels. A discretization is performed of the ranges where the chosen sampling rates are set to $\Delta\theta = 1\,°$ and $\Delta r = 1$. For each detected edge pixel from the outcome of the implemented Canny edge detector, every $\Delta\theta$ is considered.

The implemented HT require a number of input parameters: A threshold value for $A(\theta,\rho)$ to identify peaks as HLs, the number of peaks accepted, minimum length of a HL and the maximum gap for being classified as a HL. The values on these input parameters differ from HLs to distant HLs due to that distant HLs often are much shorter. If their values had not been changed for the distant area, distant HLs would risk being overlooked, and contrary, if the same settings were used for HLs as for distant HLs that would lead to information overload. One example is paving stones which are rich in edges and much clearer in cells 11-14 (see figure 3.3) than in cells 4 and 5. The input parameters used in this implementation are found in table 3.1.

Table 3.1: Values of input parameters used in implemented HT. y_{max} represent the height of an image in number of pixels.

Type	Threshold	Numb. of Peaks	Min. line length	Max. gap
HL	$y_{max}/3$	50	$y_{max}/10$	5
Distant HL	$y_{max}/30$	50	$y_{max}/30$	4

3.3 Analysis of snapshot properties

The outcome of previous sections handling image processing is a more informative and stripped image. This section describes the approach for interpret this image into useful information. First, an approach for estimating the street outline is introduced. The outline is represented by two suitable HLs from section 3.2.3 on which all other extracted HLs are projected. HLs that overlaps are merged and openings along the reference lines are searched for. Based on found openings the snapshot is classified into a specific type: `intersection` or `straight forward`. If its type is `intersection` an estimation of the distance to that intersection is performed. The snapshot type along with a potential distance estimation is used in the final matching procedure with information from Wisepilot^{TM}.

3.3.1 Extracting Reference Lines

HLs suitable for representing the street outline in a snapshot are searched for. When two such HLs have been found, so called reference lines are extracted which have the same slopes as the found HLs. The street's horizon is optimally located where the two reference lines intersect.

The search procedure starts with the image being divided into 16 regions, according to figure 3.3. Each region is ranked by its probability of holding HLs representing the street outline. Region 1–8 are discarded since the advised smartphone's orientation while snapping a picture (see Appendix A) indicate that the horizon will approximately be located in the center of a snapshot. The remaining regions, 9–16, are all valid for holding reference lines. However, the regions 9, 10, 15 and 16 are prioritized. This prioritization is done after a number of test images showing that those are the most common regions containing the outline of the street when the picture has been snapped as advised.

All extracted HL from section 3.2.3 are each handled, dependent on its place-

ment and parameter values of length r and slope θ, it is either stored or discarded. HLs partly or completely located in section 9, 10, 15 or 16 precede other HLs; lengths are considered secondary.

The algorithm for locating reference lines are applied on both the left and right side of an image. Valid HLs on the left side has a slope within the range $0° \leq \theta < 75°$ and on the right side within $105° \leq \theta < 180°$. Out of the HLs that fulfills the left side condition as well as being located in region 9 or 10, the longest will be chosen to constitute the foundation of the left reference line. The same holds for the right reference line but with the right side condition and region 15 or 16. Scenarios could occur where the foundation of only one reference line is found by the above criterions. If so is the case, that extracted reference line is mirrored to compose the reference line on opposite side. If no HLs fulfill the criterions, the algorithm defaults to reference lines illustrated in figure 3.9.

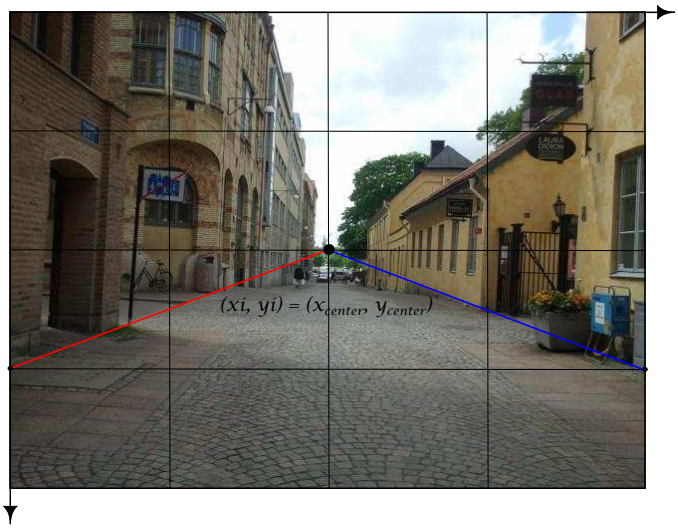


Figure 3.9: Pre-determined reference lines when no Hough lines fulfill the criterions of being valid foundations for reference lines.

It is important to emphasize that the most distinct edges which are represented by the longest HLs not always constitute the outline of a street. Empirically they are more frequently representing building structures like windows. It is, however, not crucial to find the exact street contour in an image — most important is to get an approximate direction and this can be achieved with help of building structures as well, as long as they are parallel to the street of interest. However, extracted reference lines located at actual

ground level are preferable since those results in better distance estimations (described in section 3.3.5).

3.3.2 Project Hough lines onto ground level

The outcome of the procedure for estimating the street outline are two reference lines, one for each side. Extracted HLs which do not constitute the foundation of the two reference lines are still helpful for determining the snapshot type and are therefore included in the analysis. This is achieved by projecting them onto ground level, i.e. the reference lines. For this purpose, the reference lines linear equation parameters, k and m (from equation 2.9) are determined as well as the point, $p_i = [x_i,\ y_i]$, where the two reference lines intersect. These parameter values are set as projection references.

Each HL consist of two points, $p_1 = [x_1,\ y_1]$ and $p_2 = [x_2,\ y_2]$, a slope, θ, and a length, r. All HLs are classified being left or right based on their values of x_1 and x_2. If $x_1 < x_i$ AND $x_2 < x_i$ then it is classified as a left HL or if $x_1 > x_i$ AND $x_2 > x_i$ it is classified as a right HL where x_i represent the x-value where the two reference lines intersect, as is illustrated in figure 3.9. Lines having x-values on both sides of x_i are discarded. Also, a range for θ has been set for the projection. This is to sort out HLs that would not be very helpful but rather interfere with the analysis. HLs with a slope within $5° < \theta < 175°$ are valid for projection. A special case is applied on the distant area represented by regions 4 and 5 in figure 3.3. This is since it is hard to get reliable information from extracted HLs due to the snapshot's poor resolution and the grand real distance to that area. The most apparent information obtained is from roof tops and therefore the range of the distant HLs slopes have been limited to $105° < \theta < 165°$ on the left side and $15° < \theta < 65°$ on the right side.

The projection of HLs is made along the y- or x-axis dependent on their location. For each HL, y_1 and y_2 are compared to the reference lines y-value, y_{RL}, which is calculated by equation 2.9 based on x_1 and x_2. If a HL is located above the reference lines, $y_1 > y_{RL}$ AND $y_2 > y_{RL}$, representing e.g. building structures the projection is performed along the y-axis. If instead a HL is located below the reference lines, $y_1 < y_{RL}$ AND $y_2 < y_{RL}$, the projection is performed along the x-axis since the HL is located at ground level. This is also the case if only one point is found at ground level $y_1 < y_{RL}$ OR $y_2 < y_{RL}$. The projection scenario is illustrated in figure 3.10. Projection along the y-axis is made with help of the linear equation 2.9 while projection along the x-axis is made with help of a shifting of the linear equation according to

$$x_p = \frac{y_p - m}{k}. \tag{3.3}$$

In the ideal case when all valid HLs have been projected onto the reference lines, intersections are represented by openings along the reference lines, see figure 3.11.

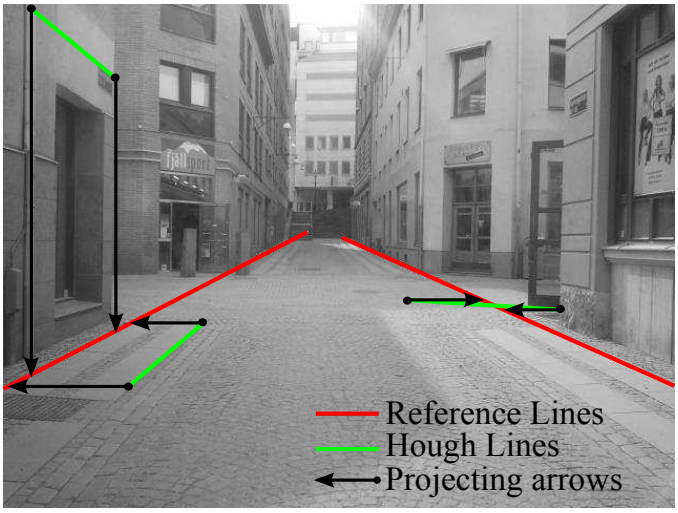


Figure 3.10: Lines projected on to reference lines.

3.3.3 Merge projected Hough lines

The projection is made separately for each valid HL and therefore no openings can be recognized directly. Hence, all projected lines is merged to get a general idea of the situation. The merging procedure loops through all projected lines and checks for overlaps. Overlapped lines are merged resulting in longer and a fewer number of lines from which the street structure can be interpreted. Small gaps between two such lines are disregarded and the lines are seen as one line, as is illustrated in figure 3.12. Large gaps will not be disregarded and it is such openings that is considered to be intersections.

Figure 3.11: Result of line projection.

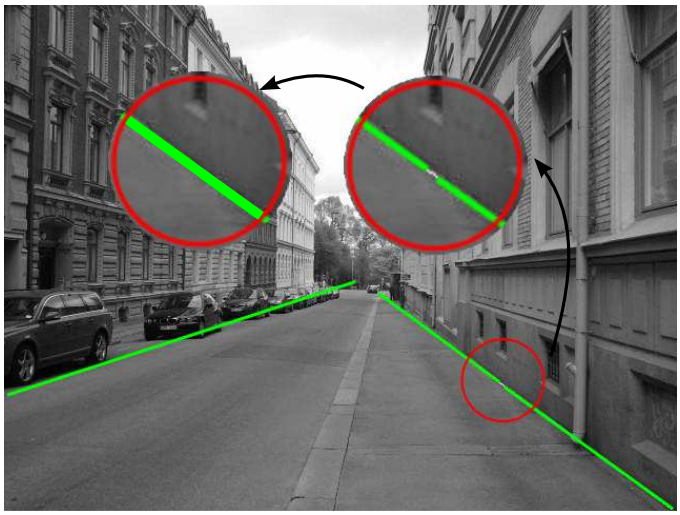


Figure 3.12: Illustration of how the small gapes are disregarded and concatenated.

3.3.4 Classification of snapshot

The snapshot is classified to be matched with map data from Wisepilot^{TM}. There are two kind of classification types, `intersection` and `straight forward`. Figure 3.13 illustrate projected lines from section 3.3.3 that results in

these types. Also, if the type is interpreted to be an `intersection` then an estimation of the distance to the mid point of that intersection is performed as described in section 3.3.5. The resulting type along with a potential distance estimation is provided to the following matching procedure which is described in section 3.4.

The classification is made by checking the number of merged lines on the right and left side respectively. The snapshot is categorized as being `straight forward` if the merged lines have no openings. Otherwise, if there is one opening or more, it is categorized as being an `intersection`. If so is the case, additional information is extracted from this intersection by the algorithm described below.

IF *"there is one left line and more than one right line"*

RETURN the largest opening on the right side as the intersection.

ELSE IF *"there is one right line and more than one left line"*

RETURN the largest opening on the left side as the intersection.

ELSE IF *"there are more than one line on both left and right side"*

IF *"two openings overlap in y-direction"*

these openings are selected as being the intersection

ELSE

the largest opening is selected as being the intersection.

END IF

END IF

The outcome of this classification procedure is the snapshot's categorization type and if the type is `intersection`, the value of the y-pixel placed in the middle of the intersection and the estimated distance length to that intersection are provided as well.

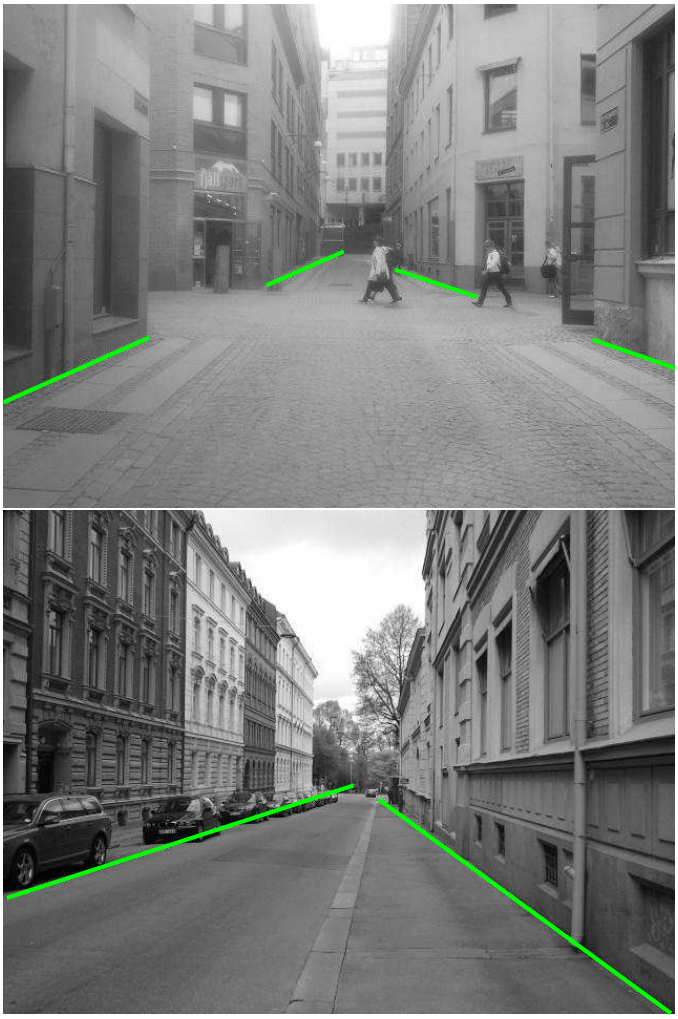


Figure 3.13: Example cases of projected and merged lines. The upper figure is categorized as an `intersection` and the lower figure is categorized as a `straight forward` street.

3.3.5 Distance estimation in image

If the snapshot is interpreted as being of the type `intersection` then an estimation of the distance to that intersection is performed which is achieved with trigonometry calculations. It is the real distance to the detected intersection that is of interest and therefore the distance estimation is performed along the y-axis of a snapshot, see figure 3.3. The snapshot taken is in perspective mode (according to Appendix A), this in turn results in that pixel ranges with high[1] y-values represent shorter real distances than pixel ranges

with low y-values. The part of an image where distance estimation is of interest is along the street which is located in the lower section of the image, from pixel $p_{y_{max}}$ to the center pixel $p_{y_{max}/2}$. For each pixel within this range a distance is calculated, d_p. The procedure is described below.

The height above ground level, h, is estimated to be approximately 1.5 meters, a number based on the mid-length of the Swedish population [9]. The vertical angle of view θ_{view} and the orientation angle θ_{roll} are stored while snapping a picture. The value of θ_{view} will be constant for a specific smartphone while θ_{roll} differs from snapshot to snapshot dependent on the smartphone's orientation as is described in section 3.1. The estimation approach is described below and a visual representation is illustrated in figure 3.14.

1. The angle to the shortest visible distance in a snapshot, $\theta_{p_{y_{max}}}$, is calculated according to,

$$\theta_{p_{y_{max}}} = \theta_{roll} - \frac{\theta_{view}}{2} \tag{3.4}$$

2. Angles, θ_{p_y}, representing each pixel within the range $P = [p_{y_{max/2}}, p_{y_{max}}]$ is calculated by

$$\theta_{p_y} = \theta_{p_{y+1}} + \Delta\theta \tag{3.5}$$

   where $\theta_{p_{y+1}}$ represent the angle to the pixel below pixel p_y and $\Delta\theta$ represent equally large sub-angles for each pixel in range P. $\Delta\theta$ is determined by

$$\Delta\theta = \frac{\theta_{tot}}{y_{max}/2}. \tag{3.6}$$

   where θ_{tot} is the total angle from the bottom to center part of an image, $\theta_{tot} = \theta_{view}/2$ and y_{max} is the total number of pixels along the y-axis of a snapshot.

3. The distance d_p to each pixel p_y in the range P is calculated according to

$$d_p = h \cdot tan(\theta_{p_y}) \tag{3.7}$$

[1]Observe that high y-values are located at the bottom part of a snapshot

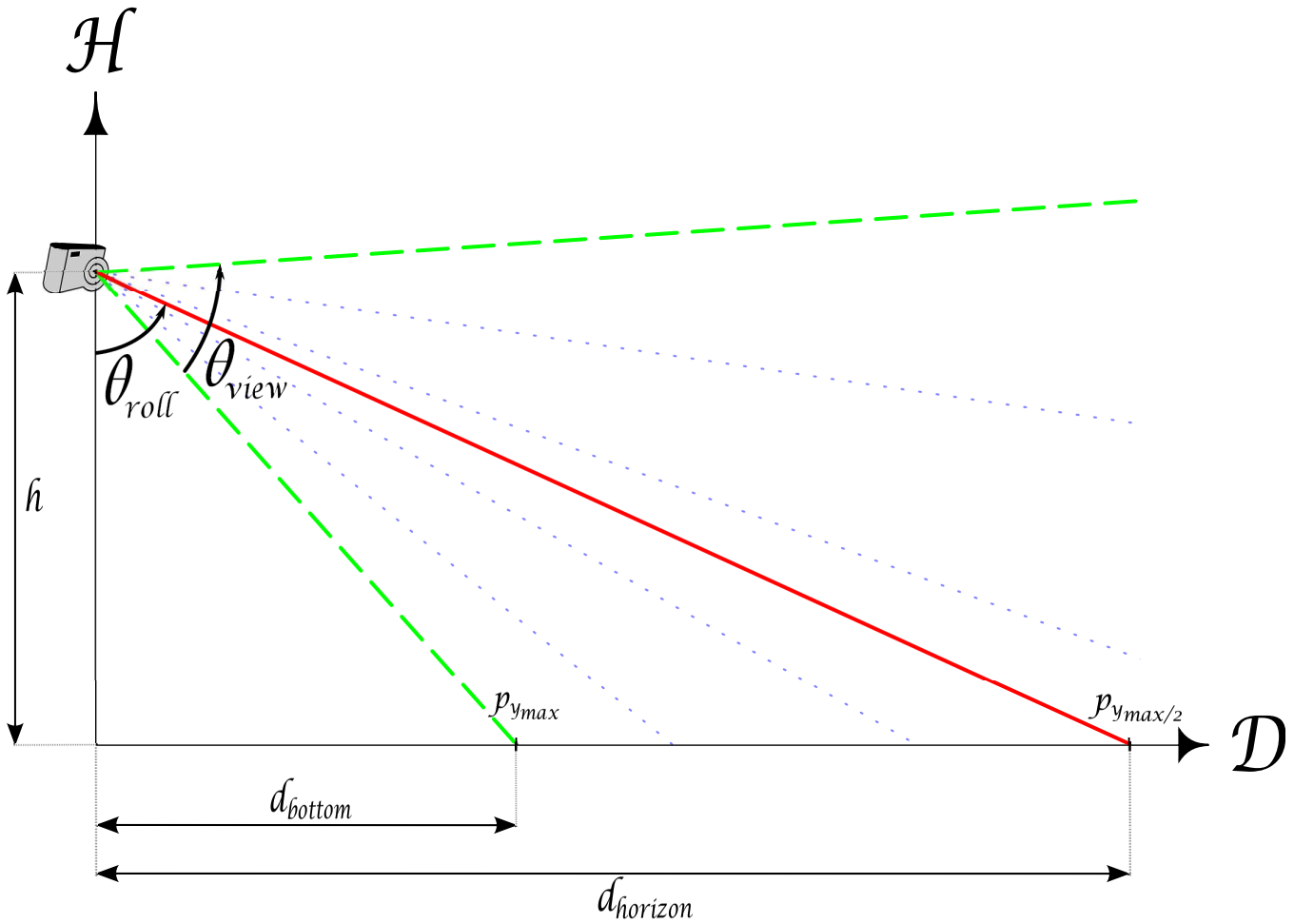


Figure 3.14: Angle of view, θ_{view}, and orientation angle, θ_{roll}, representation in a 2D illustration of the real world. The distance estimation is performed along the x-axis, d_{bottom} represent the distance to the lower part of a snapshot and $d_{horizon}$ represent the distance to the mid part of a snapshot.

3.4 Integration with Wisepilot^{TM}

Wisepilot^{TM} is a navigation system for smartphones developed by Appello Systems AB. Its structure is based on a client, the smartphone, which makes requests to a server for navigation information. The client is required to be online during navigation.

Many of the features in Wisepilot^{TM} are useful for the application of this thesis which therefore is integrated with the system. The features used are parts of the GUI, the search algorithm for finding a GPS position of specified destination, the routing procedure[1] and requests of map data. Map data is provided by NAVTEQ, it is split up in pieces by a Drill Down Server (DDS) which delivers processed data to a server where the data is processed further. Useful types are created and their description is listed in table 3.2.

Table 3.2: Types used in the application from the integration with Wisepilot^{TM}.

Type	Description
`vector map`	Holds `map objects`
`map object`	Objects of e.g. streets, street names, lakes, pounds, houses etc.
`route`	Contains navigation parameters which are delivered when a destination have been chosen
`shade point`	Points defining streets
`way point`	A `shade point` on a `route` with an instruction, e.g. turn left
`position range`	Area which holds `current position`
`current position`	Position delivered by the built-in GPS
`matched position`	Snapped position from `current position` to the closest position on the `route`

The input parameters to the matching are from the image processing procedure, the snapshot's classification type, the center pixel of potential intersection and the estimated distance to the point representing that pixel. The

[1]The navigation route calculated from one location to a specified destination.

input parameters provided by Wisepilot^{TM} are the `current position` of user and their destination target, the calculated `route`, the `matched position` and map data on the `matched position`'s surroundings. The outcome is a navigation instruction. The following sections describes the approach for determine that instruction.

3.4.1 Match intersections with shade points

A `route` is requested by providing the user's position and destination target to Wisepilot^{TM}. This `route` provides information about the pathway and its surroundings. It is the intersections along the `route` that is of interest and therefore streets within the `position range` are examined. These streets are delivered in `vector maps` containing `map objects`. Each `map object` is built of arrays containing GPS positions on the form: starting latitude-longitude position and ending latitude-longitude position. The streets are checked for intersections using linear equations. Found intersections, which are close enough for being visible from the `matched position`, are stored and matched with the next `shade point` on the route. If a `shade point` matches an intersection, the distance to it is calculated and stored. If there is no match the distance is set to far away.

3.4.2 Conditions for determine navigation instruction

In the developed application there are four different instruction types for navigation: turn left, go straight forward, turn right and destination reached. One of these is overlayed on top of the original snapshot as a navigation instruction. Determining this instruction is dependent on a set of conditions which were designed based on a few assumptions. For instance, an intersection was assumed to be visible in a snapshot at a starting distance of 70 meters and ending distance of 100 meters, these values were estimated based on measurements. The span of 30 meters is set due to the uncertainty of what part of an intersection the GPS positions represent.

Before the matching is performed calculations on the distance from received `matched position` to the following intersection on the `route` and on the distance from `matched position` to the following `way point` are required. Also, the following intersection is examined for being a `shade point` or a `way point`.

The matching procedure is explained in detail in Appendix B. Its inputs

are the calculated distance to the following intersection, the calculated distance to the following `way point`, type of intersection (`way point/shade point`), type of snapshot (`intersection/straight forward`) and the difference between the estimated distance in the snapshot and the calculated distance to the intersection.

Chapter 4

Error Sources

Recall that the developed algorithm has not made use of the smartphone's compass and the uncertainty of received GPS position in an urban environment is large. These limitations demand high quality on the analysis of a snapshot and on the matching with Wisepilot^{TM}. Error sources interfering with the algorithm are described in the following sections.

4.1 Disturbances within snapshots

There are several factors that could result in a fault classification of a snapshot. Objects other than buildings can originate HLs which do not represent the street structure and should therefore not be projected onto the reference lines. It is a difficult task to sort out valuable HLs from others and therefore such HLs will be projected as well. Typical error sources in the classification of a snapshot are listed below.

- Objects — edges of interfering obstacles can result in "faulty" HLs, e.g. trees, branches and cars (see figure 4.1).
- Car located at intersection — can originate HLs which might cause a fault classification of snapshot type from `intersection` to `straight forward`.
- Shadows — likely to be detected as edges. They are most misleading when their edges are parallel to the street's direction in a snapshot.
- Faulty detected reference lines — the street outline can be missed completely or interfering objects can interrupt valuable HLs as a cause of less distinct edges.

Figure 4.1: Marked areas which could suffer from problems with branches and cars present.

- Downhill and upward slopes — will result in inaccurate distance estimation. The distance estimation is assumed flat ground. 3D map data cannot be obtained and therefore hills go by undetected. A small slope results in relatively large errors of the distance estimation.
- Estimated height of smartphone — can result in inaccurate distance estimation. There is no way to determine the actual height while snapping a picture and thus it is estimated.
- Narrow crossing streets — can result in faulty snapshot classifications. Even for the human eye it can be difficult to detect a narrow crossing streets and details are even more difficult to detect through a camera view. The further away the crossing street is located, the more difficult it gets to detect it.
- Wide streets — difficulties arise when either the right or left side is not visible in the snapshot.

4.2 Limitations in Android and Wisepilot^{TM}

The implementation of the application is performed with the aim of not changing major parts in Wisepilot^{TM}, but rather adapting it to the existing

code. A such implementation along with smartphone limitations leads to complications which are listed below.

- The `map object` data provided by the DDS is delivered in segments of GPS positions and these do not always overlap at intersections. Therefore, this problem needs to be considered. An ultimate solution would be if the DDS also categorized `shade point`s as being intersections or not and if it is an intersection also specify its type, e.g. T-crossing, 4-way-crossing etc.

- The GPS position is not very accurate, it also becomes worse in an urban environment [10]. This inaccuracy can result in Wisepilot^{TM} generating a faulty `matched position` possibly leading to `shade point`s being marked as passed when they are still located in the up-coming parts of a `route`.

- The compass inaccuracy makes it unusable in this thesis. This complication could be handled if the compass would operate in a 3D mode.

Chapter 5

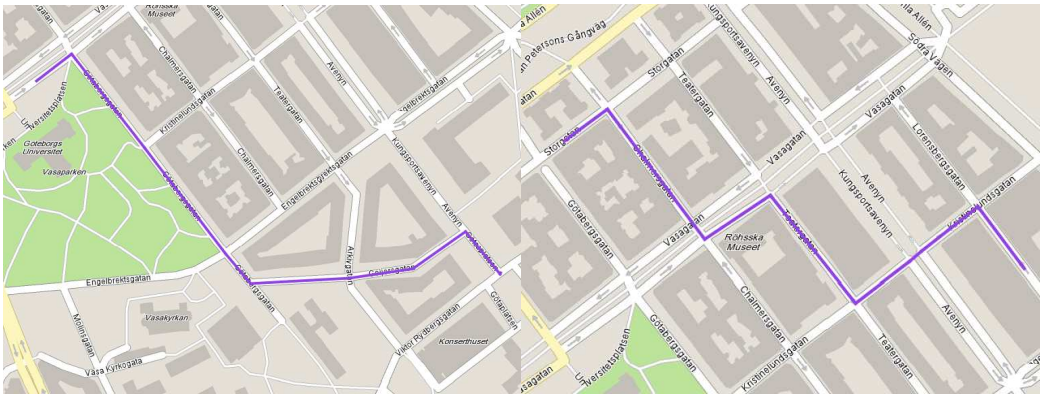
Results

The application was evaluated by reviewing six example routes. Every route is invoked from the inner city of Gothenburg and their paths are illustrated in figure 5.1. For each route a number of navigation instructions are requested. For a reliable result it is necessary that the snapshot is taken along the route. Each instruction is equally considered in the evaluation result. They are classified either being successful or not, where successful indicates a correct guiding directive along the route. No consideration is taken to the placement of the guiding arrow in the snapshot. The result is illustrated in table 5.1.

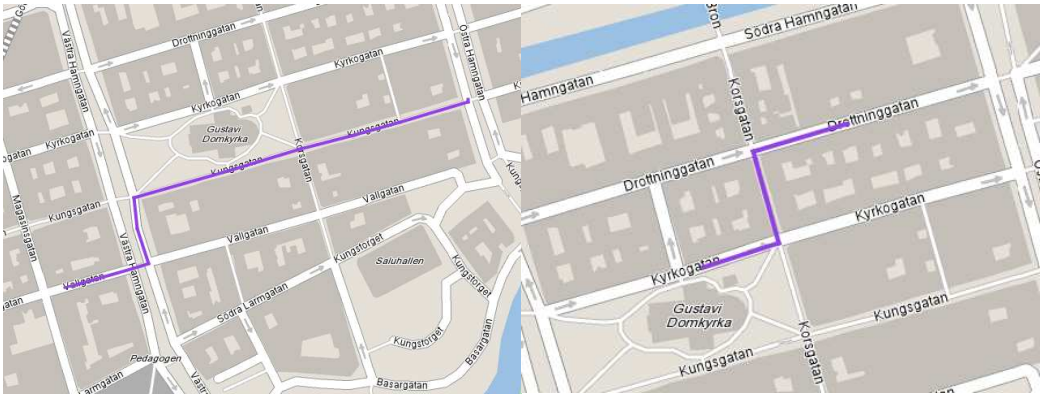
Table 5.1: Navigation instruction results for each followed route 1–6.

Route	Number of snapshots	Successful	[%]
Route 1	7	$^{7}/_{7}$	100.0
Route 2	8	$^{4}/_{8}$	50.0
Route 3	5	$^{5}/_{5}$	100.0
Route 4	3	$^{2}/_{3}$	66.7
Route 5	4	$^{3}/_{4}$	75.0
Route 6	6	$^{4}/_{6}$	66.7
All snapshots	33	$^{25}/_{33}$	75.8

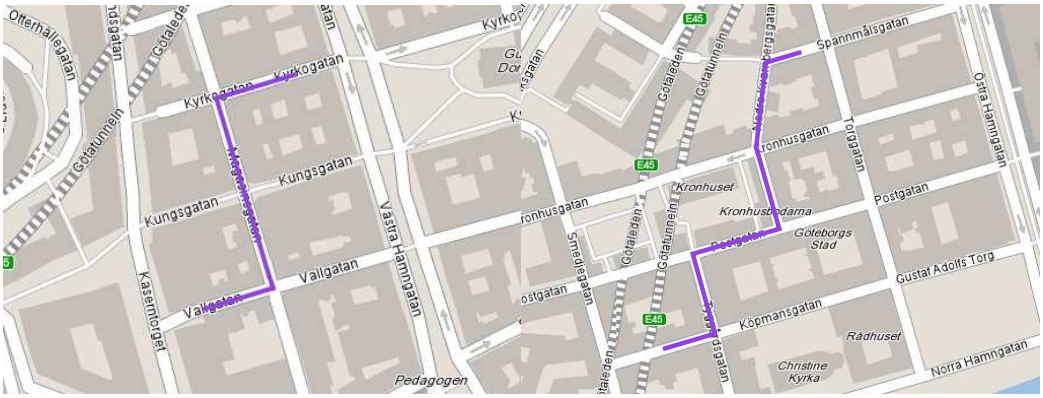
For the not successful cases something in the algorithm has failed during its execution. There are a number of critical steps which can result in complications. These complications can be crucial for the result or overridden by the conditions for determining final navigation instruction. Therefore, a successful outcome can still contain errors in certain steps of the algorithm. Critical



(a) Route 1 (b) Route 2



(c) Route 3 (d) Route 4



(e) Route 5 (f) Route 6

Figure 5.1: Evaluated routes, all located in the inner city of Gothenburg.

steps are listed in tables 5.2 and 5.3 where an analysis has been performed of the snapshots applied from table 5.1.

Table 5.2: Critical steps in final algorithm for determining navigation instructions for all successful snapshots from table 5.1.

Snapshot processing step	Snapshots / Successful[1]	[%]
Correct intersection detected [2]	$^{17}/_{20}$	85.0
Correct classification of snapshot	$^{24}/_{25}$	96.0
Correct display of arrow	$^{23}/_{25}$	92.0
Matching step	**Snapshots / Successful[1]**	**[%]**
Approximately correct GPS position	$^{25}/_{25}$	100.0

Table 5.3: Critical steps in final algorithm for determining navigation instructions for all snapshots with incorrect navigation instructions from table 5.1.

Snapshot processing step	Snapshots / Unsuccessful[3]	[%]
Correct intersection detected [2]	$^{7}/_{8}$	87.5
Correct classification of snapshot	$^{7}/_{8}$	87.5
Matching step	**Snapshots / Unsuccessful[3]**	**[%]**
Approximately correct GPS position	$^{0}/_{25}$	0.0

It is clear that a crucial step for being a successful or unsuccessful navigation instruction is highly dependent on the GPS position. An incorrect GPS position will always result in an unsuccessful navigation instruction. A faulty GPS position is not regarded in this thesis and therefore not evaluated further. Most common error sources attached to snapshot classification are listed in table 5.4, one snapshot can contribute to more than one error cause.

Table 5.4: Evaluation of image processing

Image properties	Part per Snapshot	[%]
Reference lines found at both sides	$^{11}/_{33}$	33.3
Mirrored left/right reference lines used	$^{12}/_{33}$	36.4
Default reference lines used	$^{10}/_{33}$	30.3
Obstacle disturbance	$^{18}/_{33}$	54.5
Shadow disturbance	$^{6}/_{33}$	18.2

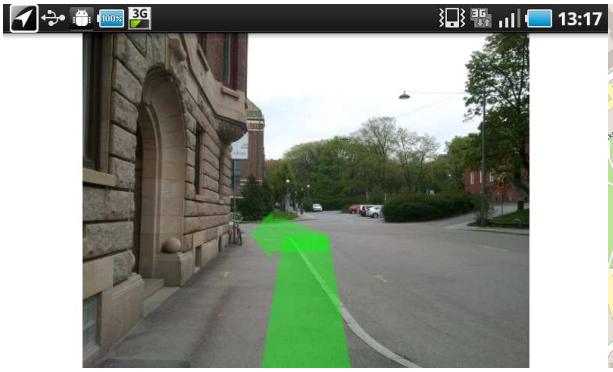
Obstacles and shadows are common in snapshots (as is clear from table 5.4), these disturbs the detection of HLs. HLs in turn are necessary for finding good reference lines. Therefore, finding only one or none reference line is not an unusual occurrence. If one reference line is found, half the street's outline is obtained. The street in a snapshot is preferred to be symmetrical located (see Appendix A) and a mirrored reference line is therefore still representative. However, default reference lines provides significantly lesser information and to determine whether they are trustworthy or not is difficult. In spite of these potential errors the snapshot classification is correct in 31 out of 33 possible cases. This indicates that the snapshot classification is not most critical for being correct when determining navigation instruction, the GPS position on the other hand is.

An example of a typical route is presented by the most successful route from table 5.1, route 1. It is illustrated in figure 5.4 along with locations where navigation instructions have been requested. The outcome of these requests are presented in figures 5.5 – 5.11. The snapshot has been taken just before an intersection, if there is one, for obtaining best result (as is advised in Appendix A).

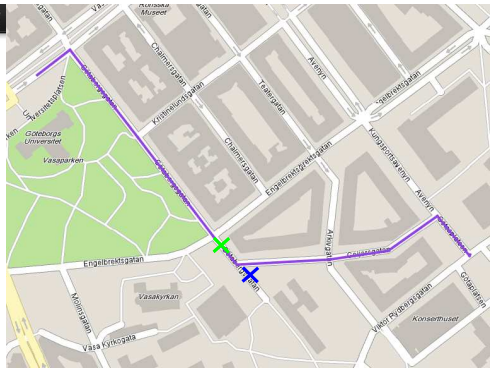
[1]Representing the same snapshots that resulted in successful navigation instructions from table 5.1.

[2]Not every snapshot has detected an intersection, only those who has are considered.

[3]Representing the same snapshots that resulted in incorrect navigation instructions from table 5.1.

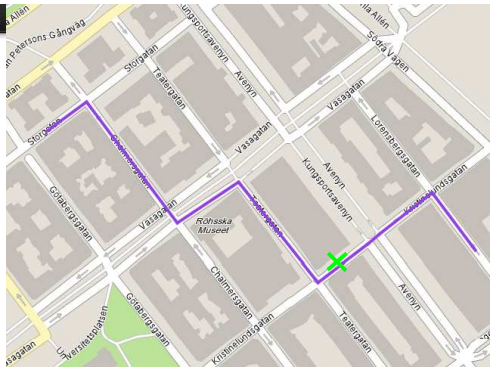


(a) Successful navigation instruction 1



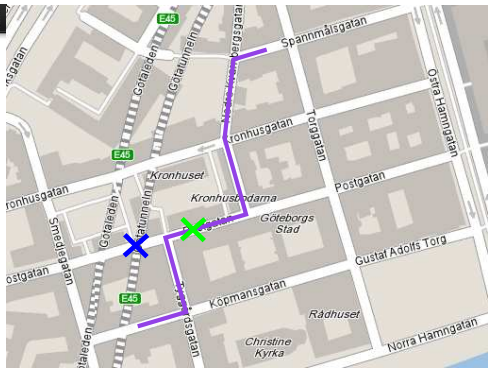
(b) GPS position for instruction (a)

(c) Successful navigation instruction 2



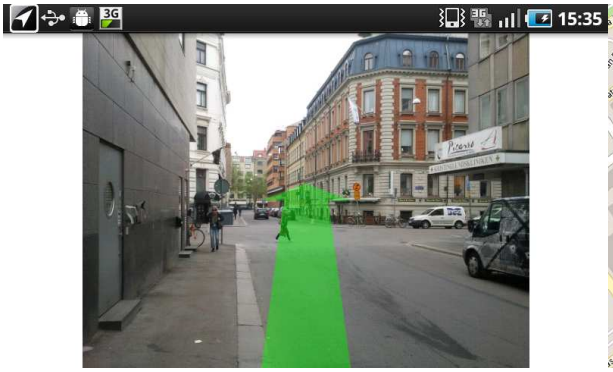
(d) GPS position for instruction (c)

(e) Successful navigation instruction 3

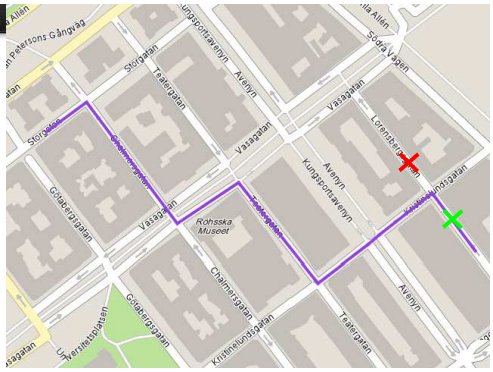


(f) GPS position for instruction (e)

Figure 5.2: Example images of a successful navigation instruction. The blue crosses in images (b), (d) and (f) represent the received GPS position while the green crosses represent the snapped GPS position.

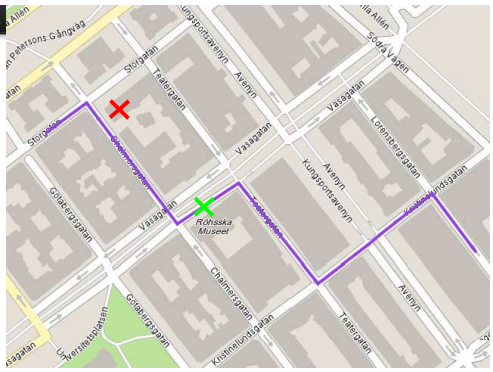


(a) Unsuccessful navigation instruction 1



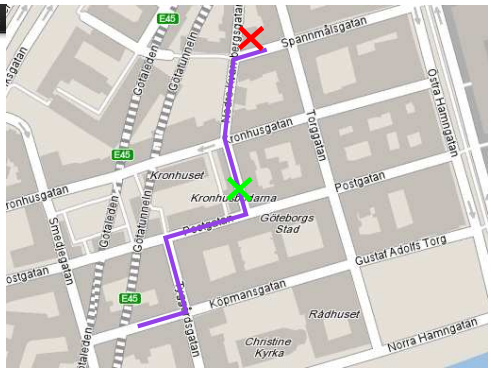
(b) GPS position for instruction (a)

(c) Unsuccessful navigation instruction 2



(d) GPS position for instruction (c)

(e) Unsuccessful navigation instruction 3



(f) GPS position for instruction (e)

Figure 5.3: Example images of an unsuccessful navigation instruction. The red crosses in images (b), (d) and (f) represent the received GPS position while the green crosses represent the snapped GPS position.

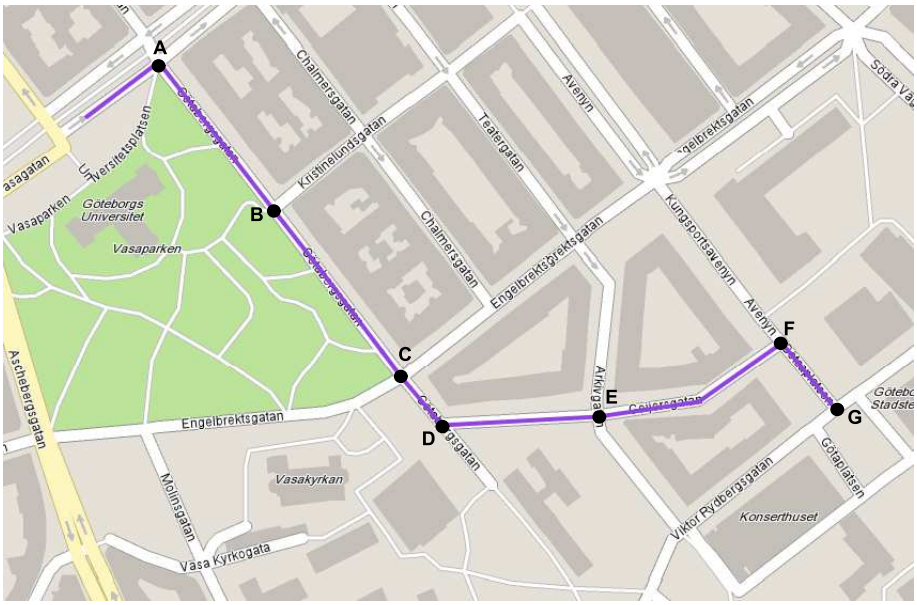


Figure 5.4: Example route with marked locations for navigation instruction requests.

Figure 5.5: Point A – Right turn.

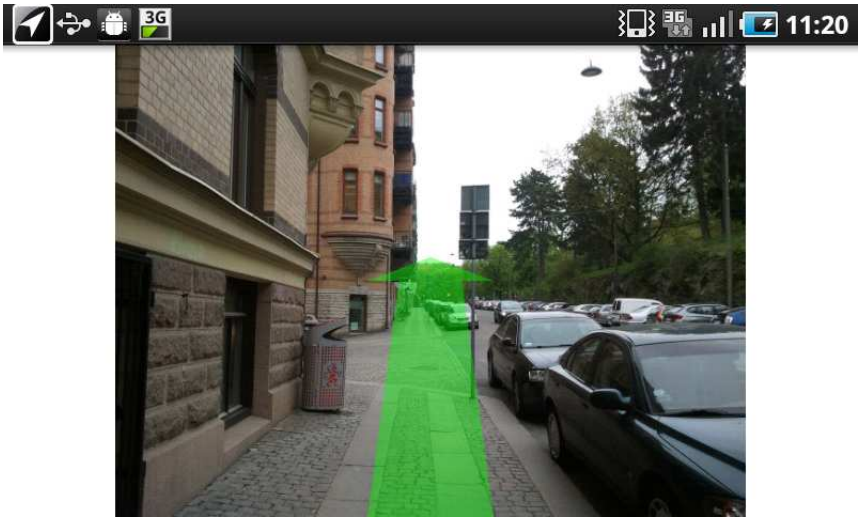


Figure 5.6: Point B – Straight ahead.

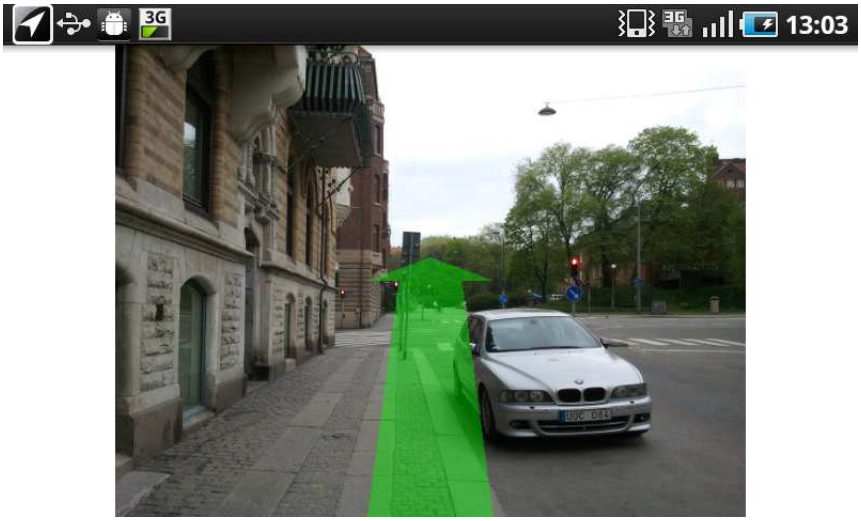


Figure 5.7: Point C – Straight ahead.

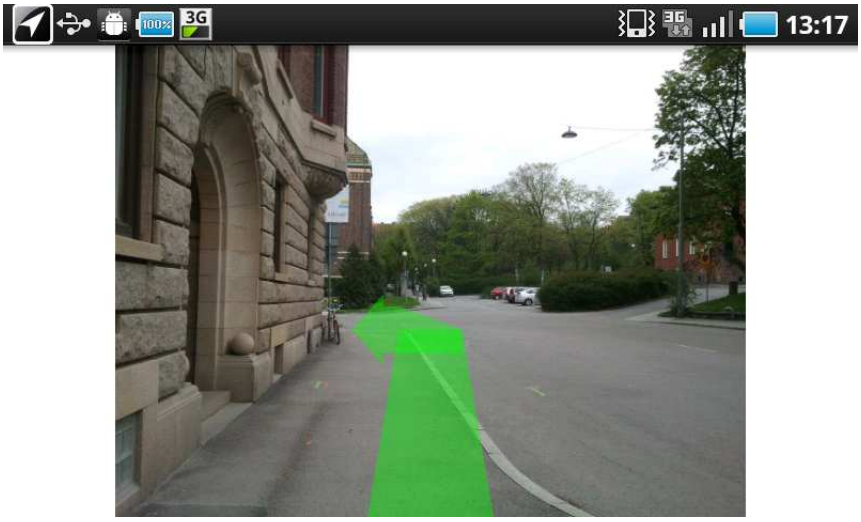


Figure 5.8: Point D – Left turn.

Figure 5.9: Point E – Straight ahead.

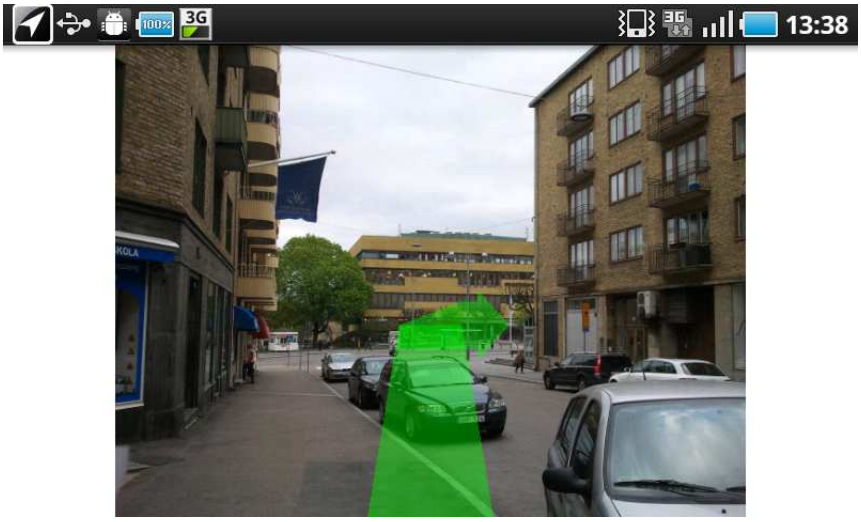


Figure 5.10: Point F – Right turn.

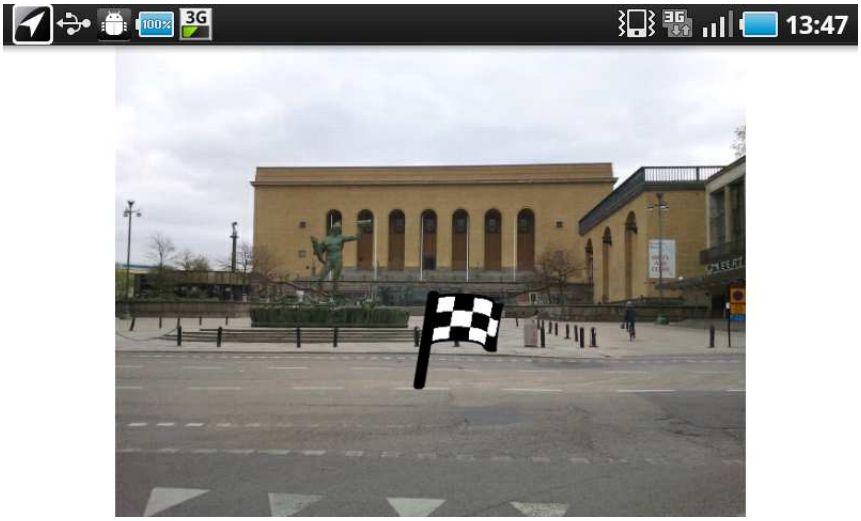


Figure 5.11: Point G – Destination reached.

Chapter 6

Discussion and Future work

Different parts in the developed algorithm are in this section examined and evaluated separately. The methods used for processing the snapshot content are classical image processing methods which are selected because of their simplicity and robustness. Also, they are suitable for extracting information from an urban environment where buildings with sharp edges occurs frequently, like windows, corners and transitions to streets. The Gaussian filter blurs the snapshot removing redundant information before the remaining edges are detected by the implemented Canny edge detector. The resulting edge points are interpreted with help of the Hough Transform (HT) to a more usable set of data, i.e. lines. The reason for choosing this approach is that streets often can be defined by straight lines. The HT is a method based on a ranking system which leads to trustworthy results based on the edge image.

The estimation of the street outline is made with generated reference lines which are based on suitable Hough Lines (HLs), as described in section 3.3.1. Such Hough Lines (HLs) are searched for in specific cells where the outline of the street normally is located. By only searching in those cells, there is a risk that the street outline is estimated wrong since more suitable HLs could still be located in other cells. However, even if the other cells can contain well presented HLs in a certain scenario they can be misleading, i.e. describing buildings, in other scenarios. The method for detecting reference lines can be improved if the extraction procedure of snapshot information is improved.

The snapshot classification is divided into two types, `intersection` and `straight forward`. The intersection type could also be classified as its actual type, however, there are two main reasons why only `intersection` and `straight forward` are considered. Firstly that the procedure for detecting intersections is not stable enough to always detect their cross-streets. This is

due to obstacles and shadows in the snapshot disturbs the edge detection procedure. Secondly it is because that the map data contributed by NAVTEQ provides streets as unjoined segments. For instance, crosswalks would be interpreted as ”false” T-crossings and this applies in both directions. The problem with crosswalks could easily be solved but since the street detecting algorithm is vulnerable against disturbances it is not carried out in this thesis.

The distance estimation to intersections is vulnerable due to a number of factors. For instance, the smartphone's height while snapping a picture is estimated and not measured. While performing the trigonometry calculations, the height is relatively short compared to the distances to intersections. Therefore, it has a critical role and a poor estimation leads to large errors. Also, the street on which a snapshot is taken is not always completely flat. If there is a slope, uphill or downhill, it will cause errors in the distance calculations since the angle to the street cannot be calculated. This problem could be dealt with if 3D map data was available and included an altitude component was included. Furthermore, the number of pixels in the snapshot are few. This is a problem when distances far away are estimated, resulting in that in the center part of the snapshot, one single pixel can represent several meters in the real world. Errors in one pixel might not seem to bad but will cause errors of several meters. This problem could be decreased if the snapshot size is chosen bigger, however, then other problems arise as the need for extended computational power which results in longer processing time. An accurate distance estimation would also be of use when merging projected HLs in section 3.3.3. The merging algorithm discards small gaps and consider them as false openings. The best way of implementing this would be by discarding gaps smaller then a certain distance rather then a certain amount of pixels. The current algorithm takes no consideration to the real distances of the gaps when discarding them.

The performance of the application was evaluated based on reviewing six routes along with the resulting outcome instructions. The routes have mostly been acquired where streets are quite narrow. This is because it is considerably easier to detect reference lines from buildings on both sides of a street when navigating though narrow streets than broad streets. The matching procedure, resulting in a navigation instruction, of snapshot properties with map data uses the intersections along the route as well as the current GPS position. It is clear from the evaluation result that the application is more successful than unsuccessful. From the result it is also clear that the by far largest error source is an inaccurate GPS position. This is due to that the GPS position is the only existing localization parameter and that all match-

ing conditions are based on it. For instance the delivered GPS position is snapped to the closest position on the route, calling it a `matched position`. Hence, if the GPS position is off, so is the `matched position` which is snapped to another part of the route that could be another street from where the user stands. This would mess up the matching between snapshot content and `map objects` leading to errors in navigation instructions. The poor GPS position is difficult to strengthen in an urban environment due to that GPS signals bounces on buildings. However, since the GPS receiver is the only "reliable" locator provided in a smartphone, it is used regardless of its limitations.

For improvements of the matching with Wisepilot^{TM}, usage of a reliable compass orientation of the smartphone while snapping a picture would be of great asset. The reason for not using the built-in compass is due to the lack of a 3D mode. By using a 3D mode compass the user would not have to follow the route in order to get reliable results.

Considering future work, a suggestion is that it will only be possible to snap a picture for navigation with an accurate GPS signal. In the case of having an inaccurate GPS signal the user should have the alternative to see a map of the route. As soon as the smartphone obtain a strong enough signal, the possibility of snapping pictures for guidance should be available again.

Chapter 7

Conclusions

The algorithm used for processing and analyzing the snapshot is in need of improvements for always supplying the matching procedure with reliable information. Improvements on the method for classifying the snapshot into a type would be helpful if the method can be kept robust, if different kinds of intersections were considered it would result in more informative data.

If the direction of which a picture is snapped was considered improvements in the matching procedure would be possible. A reliable built-in compass would have been helpful for this purpose.

Regardless of these needs for improvements, the resulting navigation instructions of the application are more often successful than unsuccessful. However, for a complete evaluation of the applications performance, more data sets have to be considered.

From the results it is clear that the GPS position by far is the largest error source.

Finally, the application as it is implemented today, does not provide enough reliable results to be commercialized.

Bibliography

[1] Open Handset Alliance. Android, overview, 2011. URL: http://www.openhandsetalliance.com/android_overview.html, (Accessed: 18 January 2011)

[2] Android Developers, 2011. Android 3.1 Platform. http://d.android.com/sdk/android-3.1.html (Accessed 2 June 2011 16.13)

[3] U.S department of commerce National Oceanic and Atmospheric Administration, Do you know where you are? - The Global positioning System, 2008. URL: http://oceanservice.noaa.gov/education/kits/geodesy/geo09_gps.html, (Accessed: 20 January 2011)

[4] U.S. Coast Guard Navigation Center, General information on GPS, 2007. URL: http://www.navcen.uscg.gov/?pageName=GPS, (Accessed: 20 January 2011)

[5] John F. Canny. A Computational Approach to Edge Detection, 1986.

[6] Rafael C. Gonzalez, Richard E. Woods. Digital Image Processing Third Edition, 2008.

[7] Linda G. Shapiro and George C. Stockman. Computer Vision, First edition 2001. pp. 332.

[8] Android Developers, 2011. Public class SensorEvent. http://d.android.com/reference/android/hardware/SensorEvent.html(Accessed 20 May 2011 13:45)

[9] Statistiska Centralbyrån. Vikt och längd i befolkningen, 2008. URL: http://www.scb.se/Pages/TableAndChart____47966.aspx, (Accessed: 3 Maj 2011)

[10] Vertongen, P.-P. Hansen, D.W. Location-based Services using Image Search. Applications of Computer Vision, 2008. WACV 2008. IEEE Workshop on. June 2008.

Appendix A

Navigation Instructions

When taking a snapshot for navigation a few things can be considered for enhancing the performance of the system. These are described in the list below.

- The snapshot should be taken in landscape mode, i.e. horizontal position of camera.
- For best result, snap the picture for navigation:
  - as symmetrically as possible
  - just before reaching an intersection
  - when it is assured that street lines are visible in the picture
  - with a minimum number of obstacles present to avoid interference on determined instruction
  - without too much light falling in with the camera lens

Appendix B

Conditions for Matching

On the delivered `route` a number of `shade point`s are examined in the following order from `matched position`: current `shade point`, next `shade point`, previous `shade point`, second next `shade point` and so on. The matching conditions are explained in the pseudo code below.

FOR *"`shade point`s to be examined <= 7 OR condition has been fulfilled"*

IF *"snapshot type is* `intersection`*, next* `shade point` *is a* `way point`*, 2 meters < distance to* `way point` *< 100 meters AND difference between estimated distances and calculated distance < 70 meters"* THEN

RETURN `way point` instruction

ELSE IF *"snapshot type is* `straight forward`*, next* `shade point` *is the destination* `way point` *AND the distance to it is shorter then 70 meters"* THEN

RETURN destination target reached

ELSE IF *"snapshot type is* `straight forward` *AND (distance to next intersection > 100 meters OR distance to next intersection > distance to next* `way point` *)"* THEN

RETURN straight forward instruction

ELSE IF *"snapshot type is* `intersection` *AND (distance to next intersection < 70 meters OR distance to the next* `way point` *> 100 meters)"* THEN

RETURN straight forward instruction

END IF

END FOR

IF *"no condition was fulfilled AND snapshot type is* `intersection` *AND (distance to next intersection < 70 meters OR distance to next* `way point` *> 70 meters)"* THEN

RETURN straight forward instruction

ELSE IF *"no condition was fulfilled"*

RETURN instruction of snapping a new picture

END IF