Universität Heidelberg
Zentrales Institut für Technische Informatik
Lehrstuhl Automation

Bachelor-Arbeit

Geometric Feature Extraction for Indoor Navigation

Name:	Charles Barbret
Matrikelnummer:	3443570
Betreuer:	Prof. Dr. sc. techn. Essameddin Badreddin
Datum der Abgabe:	11. April 2021

Abstract

The intent of this thesis is to create a system that can detect Walls, Doors and Corners in an indoor office environment. The goal is to be able to detect these features automatically and add a semantic aspect to the wheelchair's navigational ability. Concretely in the future, this will mean being able to tell the wheelchair to go down the hallway, turn left at the intersection and go in the second Door on the right. The wheelchair will know exactly what we mean and execute this. This work expands on the work of Borges and Aldon 2004, which is capable of extracting Walls from a laser scan image. By using these Walls, further features can be extracted, such as Doors, Corners, and Corridors.

Contents

List of Symbols

α	angle
d	distance
P_n	Point n made up of the coordinate tuple (p_{nx}, p_{ny}). The reference frame has the scanner at (0,0). Along positive x axis is the direction the scanner is facing.
r	radius of a circle

”Doors and Corners, this is where they get you.”
-Josephus Miller (The Expanse)

1 Introduction

1.1 Motivation

This work aims to set the foundation to improve the ease of indoor navigation of an electrical wheelchair. It does this by using data provided by a laser scanner that is scanning the environment and processing the scan points to return semantic terms that a human can understand. This list of terms includes *Walls*, *Doors*, *Corners*, and *Corridors.* By defining semantic terms in a way that a computer can understand, it is much easier for humans to communicate with the machine. With these terms defined, future works will be able to provide user friendly navigation options. This project aims to aid electric wheelchair navigation. From here on, the generic term 'Robot' will be used because an electric wheelchair is not actually needed. The underlying computer program only relies on the capabilities of a laser scanner.

1.2 Problem Statement

Given consecutive 2D laser-scan images provided by the Robot, a search is conducted for a method to extract semantic features (Walls, Corners, open Doors, Corridors) from 2D laser scan images.

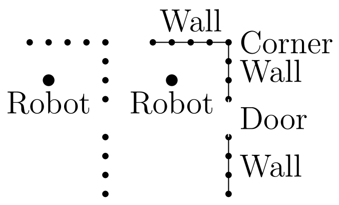


Figure 1.1: Human interpretation of a laser scan image

1.3 Laser Scan

1.3.1 Laser Scanner

The specific laser scanner that was used to record the laser scan images is a "2D LiDAR sensors TiM561". It has a range of up to 10 meters and a refresh rate of 15 hertz. The angle range and angle increment in radians are set at:

$$\begin{aligned} \alpha_{min} &\approx -2.356 \\ \alpha_{max} &\approx 2.356 \\ \alpha_{increment} &\approx 0.006 \quad . \end{aligned} \tag{1.1}$$

1.3.2 Laser Scan image structure

The form which the scan images are expected is a list of distances. The length of this list

$$\lceil \frac{(\alpha_{max} - \alpha_{min})}{\alpha_{increment}} \rceil$$

is defined by the scan range and angular resolution. The angle at any given index i is determined by $\alpha_i = \alpha_{min} + (i * \alpha_{increment}$.

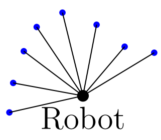
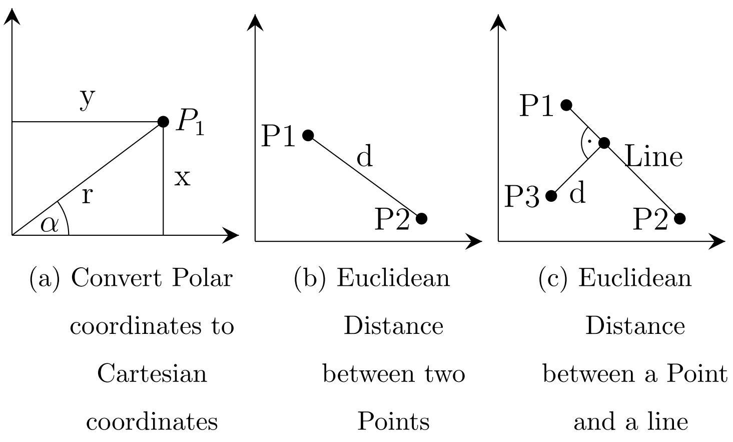


Figure 1.2: Examples of ranges

1.4 Basic Formulæ

In this section the formulæ used throughout this thesis are introduced. These formulæ should provide the reader with the tools to understand this thesis.



(a) Convert Polar coordinates to Cartesian coordinates (b) Euclidean Distance between two Points (c) Euclidean Distance between a Point and a line

Figure 1.3: Basic Set of Formulæ

Converting Polar Coordinates to Cartesian Coordinates

The first formula converts polar coordinates to Cartesian coordinates. This conversion is needed because it is easier to extract features from the Cartesian coordinate system. Figure 1.3a shows an example of how the angle and distance can represent a point in both the polar and cartesian coordinate systems. By using

$$\begin{aligned} x &= r \cdot \cos(\alpha) \\ y &= r \cdot \sin(\alpha) \end{aligned} \tag{1.2}$$

we can extract the x and y coordinates from the angle and the distance to the origin.

Euclidean Distance between Points

Figure 1.3b shows an example of the Euclidean distance between two points. This is determined with

$$d(P_1, P_2) = \sqrt{(p_{2x} - p_{1x})^2 + (p_{2y} - p_{1y})^2} \ . \tag{1.3}$$

An example of the use of this function is in the Breakpoint Detection, which is covered in Chapter 2.

Euclidean Distance between a Line and a Point

Figure 1.3c offers a visualization for

$$d(P_1, P_2, P_3) = \frac{|(p_{2y} - p_{1y}) \cdot p_{3x} - (p_{2x} - p_{1x}) \cdot p_{3y} + p_{2x} \cdot p_{1y} - p_{2y} \cdot p_{1x}|}{d(P_1, P_2)} \ . \tag{1.4}$$

Here, the desired length is the shortest distance from a point to a line. The shortest distance will be perpendicular to the line that is spanned by two points.

Minimum Distance between two Lines

To get the minimum distance between two Lines A and B, the distance between the start and end points of one line is compared with the other line. There are four possible distances, of which the minimum is then returned as follows:

$$\begin{aligned} d_{min}(P_{A-start}, P_{A-end}, P_{B-start}, P_{B-end}) = \min(\ & d(P_{A-start}, P_{A-end}, P_{B-start}), \\ & d(P_{A-start}, P_{A-end}, P_{B-end}), \\ & d(P_{B-start}, P_{B-end}, P_{A-start}), \\ & d(P_{B-start}, P_{B-end}, P_{A-end})) \ . \end{aligned} \tag{1.5}$$

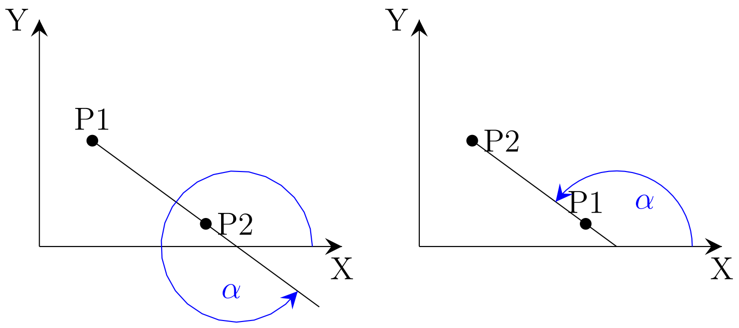


Figure 1.4: Angle between two Points (i.e. angle of a Line)

Angle of a Line

$$\alpha(P_1, P_2) = \text{atan2}(p_{2y} - p_{1y}, p_{2x} - p_{1x}) \tag{1.6}$$

uses the two parameter arctan to determine the angle between two points. The atan2 function [1] determines the angle between 0 and a single point. To compensate for this one point is translated to the origin and the second translated relative to the first. Figure 1.4 shows this process in action.

Angle between two Lines

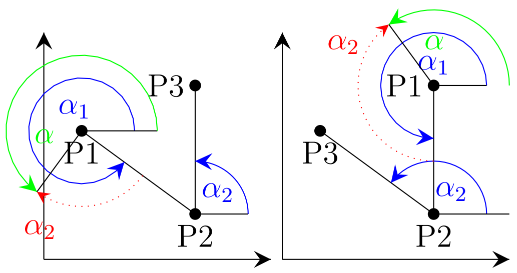


Figure 1.5: Angle α between three Points (i.e. Angle between two lines)

$$\alpha(P_1, P_2, P_3) = \alpha_1(P_1, P_2) - \alpha_2(P_2, P_3) \tag{1.7}$$

involves applying Formula (1.6) twice. This is shown in Figure 1.5. To note $\alpha(A, B, C) \neq \alpha(C, B, A)$

[1]`https://docs.python.org/3/library/math.html#math.atan2`

2 State of the art

To advance in any field one must first see what has already been accomplished and how it was accomplished. Analyzing the works of others and attempting to incorporate their findings into the work for this thesis, several papers were analyzed. However, only the final paper, which is explained in section 2.3 was used for further feature extraction. A brief explanation as to why a paper was not used is offered in each respective section.

2.1 Feature-Based Laser Scan Matching For Accurate and High Speed Mobile Robot Localization

The authors of the paper Aghamohammadi et al. 2007 introduced a methodology to compare key points from two consecutive scans. This was an early consideration as the base line for this thesis, as the generated map seemed to offer reliable Corner points. But ultimately, the key points determined did not seem to have the desired flexibility for extracting Doors, Walls, Corners or Corridors. A future possibility is to use the methods for comparing the consecutive scan points to add a tracking option to the feature extraction proposed in this thesis.

2.2 Feature extraction from laser scan data based on curvature estimation for mobile robotics

The authors of the paper Núñez et al. 2006 used the findings of Borges and Aldon 2004 to expand these findings to work with curved surfaces. These findings are not all that helpful as the premise of this thesis is that the Robot moves in an office hall way with straight wall segments and without curvature. Their findings could be implemented if a circular feature was needed. However standard office hallways usually do not have rounded surfaces. The trade off does not appear to be worth pursuing at this time.

2.3 Line Extraction in 2D Range Images for Mobile Robotics

The work of Borges and Aldon 2004 is used to extract lines from 2D images. These lines can be translated to the Walls that this thesis attempts to extract. As such, the term Walls will be used for their lines while describing their work. The idea is to find certain points, called rupture and break points, which are explained in Sections 2.3.1 and 2.3.2. These are used to group segments of connected Walls that are split at the Corners. This provides the method for extracting Walls. Using the Walls provided by their paper and comparing distances and angles of these this thesis is able to extract the semantic features desired.

2.3.1 Rupture Detection

The idea of the rupture detection is quite simple. If there are points in the laser scan image that exceed the laser scan maximum or a given threshold $d_{rupture\_max}$, there must be a discontinuity in the surrounding area. For example, this can occur when looking down a long hallway or when looking through a Doorway. If a point exceeding the $d_{rupture\_max}$ threshold is detected, the points before and after it are tagged with the rupture flag. Examples of rupture points can be seen in Figure 2.1.

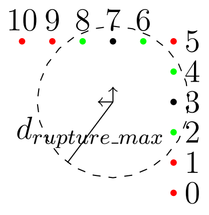


Figure 2.1: The dashed circle indicates the $d_{rupture\_max}$ radius around the Robot. The points in red are ruptured points. Green points are tagged with the rupture flag. Black points do not have a flag.

In Figure 2.1 points 0, 1, 5, 9, and 10 are denoted in red because they are outside of the range of $d_{rupture\_max}$. The result is that rupture flags are assigned as follows: Point 2 is flagged due to points 0 and 1. Points 4 and 6 are flagged due to point 5. Point 8 is flagged due to points 9 and 10.

$$d_n > d_{rupture\_max} \tag{2.1}$$

2.3.2 Break Point Detection

The idea of the break point detection closely resembles that of the rupture point detection. While the rupture point detection analyzes the distance to the Laser scanners, the break point detection analyzes the distances between two consecutive points (P_{n-1}) and (P_n). Break points occurs when the distance between the two points is greater than d_{break_max}.

$$d(P_{n-1}, P_n) > d_{break\_max} \tag{2.2}$$

d_{break_max} is defined in Formula (2.3), where λ corresponds to the worst case of incidence angle of the laser scan ray with respect to a line for which the scan points are still reliable and σ is a parameter for tolerance.

$$d_{break\_max} = P_{n-1} \cdot \frac{sin(\Delta\phi)}{sin(\lambda - \Delta\phi)} + (3 \cdot \sigma) \tag{2.3}$$

For example, the distance of d_{break_max} is exceeded when there are two separate Walls. It is also exceeded when an object, such as a pillar, is obstructing a Wall and is not directly connected to the Wall. Examples of break points are shown in Figure 2.2.

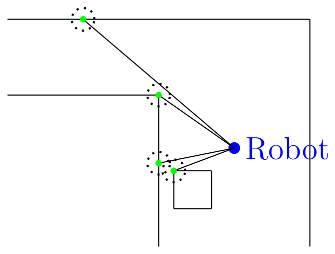


Figure 2.2: Break points are shown in green. d_{break_max} is indicated by each dotted circle around each Point.

If a break point occurs between points P_{n-1} and P_n, both points are tagged with the break point flag.

2.3.3 Wall Extraction

After the rupture and break points have been found, the points can be grouped into continuous Wall segments. These segments are not necessarily individual Walls because the break point detection is not suited for Corner detection. The first step is to group points together until a point either has a rupture or break point associated with it. Once this happens, this segment will be considered a continuous segment. It will be sent into the Iterative Endpoint Fit, which is explained in Section 2.3.4. After the first segment is completed, the process continues where it left off, working its way in the same manner through all available points to group the points into connected segments.

2.3.4 Iterative Endpoint Fit

The Iterative Endpoint Fit process takes one continuous set of points and breaks these into individual Wall segments. It does this is by taking the first and last point and connecting them with a line. Every Point between the first and last point is checked for the maximum distance to the line. If the point that this process finds exceeds a certain length, the segment is split at this point and the process is rerun on both segments until there are only straight lines.

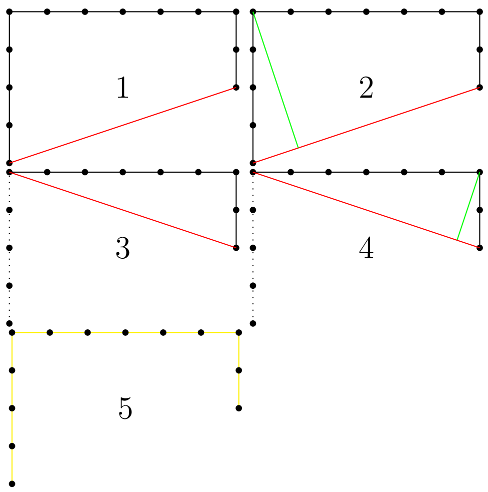


Figure 2.3: Iterative Endpoint Fit process steps

Figure 2.3 depicts the process. It starts with one continuous Wall segment, indicated in black. Step 1 creates the red line between the first and last points. Step 2 shows the maximum distance using the green line. The Corner point is where the process splits the two Walls. This becomes the start point for the dotted Wall in Step 3. The line is dotted because the process recognizes that no point here exceeds the minimum distance to be a Corner. Thus the result is a single Wall segment. The new red line uses the same Corner point. Now it is used as the end point for the red line. Step 4 shows the maximum distance with the line in green and it is split at the Corner. Neither end of this split has a point exceeding the minimum distance. Thus the process terminates and returns all three Walls.

3 Feature Extraction

Chapter 2 was a review of current literature and concepts. It described the method for extracting Walls from a set of laser scan points. Chapter 3 focuses on how these Walls can be used to extract Corners, Doors and Corridors by applying geometric comparisons. The relative location of the Robot to each feature presents difficulties for navigation. The extraction of these features are my contributions.

3.1 Definitions

Before the extraction process is explained, some terms need to be defined. All of the following elements are made up of points and such can be detected by using laser scan data.

Wall

A *Wall*, shown in Figure 3.1, consists of a tuple of two Points. With $W = (P_s, P_e)$. It is important that the Wall (A, B) is not the same as the Wall (B, A). The order in which each Point is added determines the direction of the Wall. The direction of the Wall indicates the side of the Wall the laser scanner is facing. This side is always to the left of the Wall.

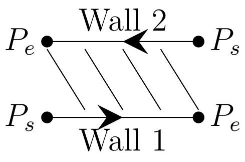


Figure 3.1: Two Walls with their direction indicated by an arrow. The shaded area is navigable

Door

A *Door* is also defined by two points $D = (P_s, P_e)$. The difference is that the start point of the Door comes from the end point of a Wall and the end point comes from the start point of another Wall. This gap will have at least one rupture point associated with it.

Here the assumption is made that a doorway exists only when the Door is open. With this assumption and the check for a gap, only one rupture point is needed, since the Door might still be attached to one side. The gap between the two points is defined to be between 0.8 and 1.2 meters.

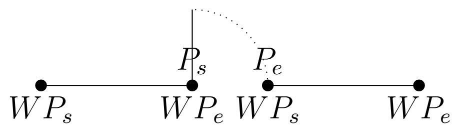


Figure 3.2: Here we have two Walls separated by one open Door

Corner

A *Corner* $C = (W_1, W_2, i)$ is made up of a tuple containing two Walls which share a Point and one integer, as shown in Figure 3.3. The Point both Walls share is the Corner in question. It is always the end Point of the first Wall and the start Point of the second Wall. The integer keeps track of the Corner type. Corner types are outer Corner which is assigned the i value of 0, inner Corner which is assigned the i value of 1, and potential Corner which is assigned the i value of 2, as described in section 3.2.1. Examples of each are shown in Figure 3.4. The Corner type impacts how the Robot can approach the Corner.

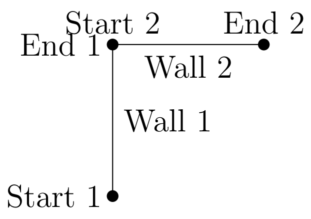


Figure 3.3: A display of what a Corner looks like with its Walls

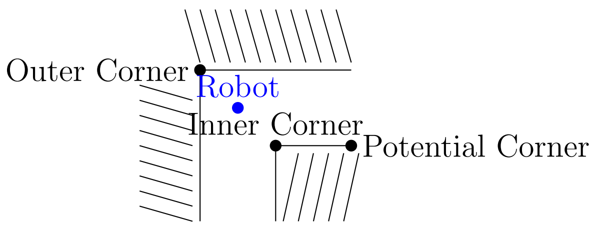


Figure 3.4: Corner Type Examples where the shaded area is occupied

Corridor

A *Corridor* is a rectangle spanned by two parallel Walls and two corners.

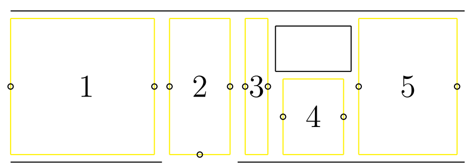


Figure 3.5: Examples of Corridors

A point is used to represent a Corridor entrance or exit. A corridor can have multiple points as indicated by the points in Figure 3.5.

[Vielleicht wird der rest hier gestrichen. Mindestens umgeschrieben.]

represented by a single Point P_c. This point is located between a Corner and a Wall. This indicates the entrance or exit into a given Corridor. Depending on how the midpoints are detected, the same object humans would describe as a Corridor will contain multiple Corridor entrances. A Corridor can have open Doors or objects in its way that creates more Corners. However, by having more of these points in the same Corridor, a path finding system can have an easier time navigating past objects, such as pillars. Figure 3.6 shows in what instances the probing Walls get used and when they yield a Corridor midpoint. The six red lines indicate probing Walls that do not yield a Corridor midpoint. The line marked with 1 does not provide a Corridor midpoint because it is too close to the laser scanner. Lines 2, 3 and 6 do not intersect with a Wall. The Wall in Line 4 is too close to the Corner where the probing Wall originates. Line 5 is too far from the laser scanner and Line 6 does not intersect with a Wall and it is too far from the laser scanner. All other probing Walls yield a midpoint, even the Door.

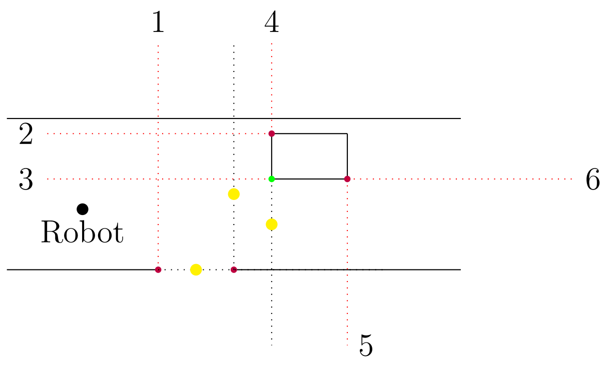


Figure 3.6: The red dotted lines indicate that the search for another Wall yielded no match

3.2 Corner Detection

Once a Corner has been defined, the next step is to detect it. Currently this is done by comparing the two Walls. Should there be a pair ($P_{n\_start} == P_{m\_end}$), the Walls m and n are combined and the Corner point is set as P_{m_end}. Now that the Corner has been identified, it is necessary to determine which side of the Corner the laser scanner can traverse. This information is not directly obvious just based on having two Walls, since the Corner in question could either be facing toward the laser scanner and be an inner Corner or be facing away from the laser scanner and be an outer Corner. To distinguish between the two, the Corner type detection has been implemented.

3.2.1 Corner Types

There are three types of Corners, an inner, an outer and a potential. Examples of which are shown in Figure 3.4

- An outer Corner is assigned the i value of 0. This Corner type points away from the laser scanner.
- An inner Corner is assigned the i value of 1. This Corner type points to the laser scanner.
- A potential Corner is assigned the i value of 2. A potential Corner occurs when a Wall has either a rupture or break point at its start or end point. This rupture or break point is then assigned the potential Corner. This works by creating a dummy Wall, where both start and end point are equal to the rupture or break point in question. This is best done by example. Figure 3.7 shows a Wall going from point A to point B. In the event that point A is to be the potential Corner, a dummy Wall is created where $P_{start} = P_{end} = A$. The Corner then is made up of

  $$p = (\text{dummy Wall}, \text{original Wall}, 2) \ .$$

  Similarly when point B is to be the potential Corner a dummy Wall is created, only now $P_{start} = P_{end} = B$. This Corner is then made up of

  $$p = (\text{original Wall}, \text{dummy Wall}, 2) \ .$$

  By defining the potential Corner this way, the information is preserved that it is a start point or end point. The reason potential Corners are relevant is that there are many situations where a laser scanner will not be able to determine accurately if a Wall continues or whether the hallway turns off. Thus, the assumption is made that any break or rupture leads to a chance of a potential Corner.

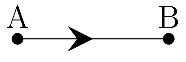


Figure 3.7: A Wall made up by points A and B

3.2.2 Corner Type Detection

A full Corner is either an inner Corner or an outer Corner. Humans can identify which type of Corner they are facing by looking at a Corner. Machine identification of Corners and the way they face is not that always simple. As mentioned in Section 3.1, a Corner is defined by the two connecting Walls and an integer representing the Corner type. When looking at a Corner, the observation can be made that the construct is simply a triangle with the hypotenuse missing. With this in mind, the relative position of the robot to this triangle can be examined. It is necessary for the robot to identify both Walls so that it can determine the presence of a Corner. In the presence of a Corner there are three possibilities for this Corner:

1. An inner Corner
2. An outer Corner where the robot is within the triangle
3. An outer Corner where the robot is outside of the triangle

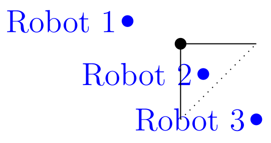


Figure 3.8: Possible Robot Relative Locations

This information must now be translated into mathematical terms. One method is to measure distances. To fully determine if there is an inner or outer Corner, three distances must be calculated.

- the distance from the robot to the corner (rc)
- the shortest distance from the robot to the missing hypotenuse of the triangle (rh)
- the shortest distance from the corner to the missing hypotenuse of the triangle (ch)

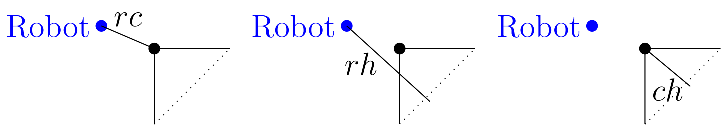


Figure 3.9: All relevant distances

One assumption that can be made here is that if

$$rc < rh$$

the Corner is an inner Corner and other wise it is an outer Corner. This assumption works as long as the Robot is not within the triangle. For this case the additional comparison of

$$rh < ch$$

can be made to see if the Robot is within the triangle. Putting these two comparisons together the Formula

$$Corner\_type = ((rc < rh) \ \& \ !(rh < ch)) \tag{3.1}$$

This Formula can be used to determine the Corner type. The result of Formula (3.1) is a Boolean. In section 3.2.1 an outer Corner was said to hold the integer value of 0 and an inner Corner to hold the value of 1. So the Boolean is directly transferable to the Corner type.

3.3 Corridor Detection

To detect the Corridor midpoint, the Corner types will be used. Starting from every Inner and Potential Corner, two probing Walls are created at a 90 and 180 degree offset to the Corners Walls. These probing Walls have a length of 2.5m and search for any intersection with another Wall. If such an intersection exists, it can be assumed that the midpoint of the Corner and intersection point is a location the Robot can move toward.

In Figure 3.6 the Inner and Potential Corners have probing Walls that exit at 90 and 180 degree angles from the Corners searching for an unconnected Wall.

Meta Wall

An early version of the Corridor detection attempted to create a so-called *Meta Wall* as shown in Figure 3.10. One reason the definition ultimately changed from the Meta Wall concept, was that comparisons between each Wall combination proved more complex than initially assumed.

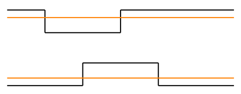


Figure 3.10: Meta Wall

4 Implementation

Having explained the theory behind this thesis, the interesting question becomes how to implement it in a practical manner.[1] The main loop is described in Algorithm 1. Each function then is explained thereafter.

4.1 Pseudo Code

```
forall scan from laserscan measurements do
    rupture_points = Detect_Rupture_Points(scan);
    break_points = Detect_Break_Points(rupture_points);
    Walls = Extract_Lines(break_points);
    Corners = Detect_Corners(Walls);
    Corridors = Detect_Corridor(Corners)
end
```

Algorithm 1: Overview

Rupture Point Detection Implementation

The first step of the rupture point detection is to check if the distance of each point to the Robot is within a valid range. The distance is given by the laser scan measurements. They provide points in the polar coordinate system. Points within the valid range proceed to be converted into the cartesian system using the Formula 1.2. Any point outside of the valid range is not converted. Instead, a rupture flag is added to both of its neighbors. The function returns a list containing all points within the valid range and, if applicable, their rupture flag. Each point is saved in both the cartesian and polar form together with the applicable rupture status.

Break Point Detection Implementation

The break point detection algorithm receives the list of points and rupture flags created in the rupture detection. As described in section 2.3.2 the distances between consecutive points is measured and checked against the d_{break_max}. If two points exceed this distance, the break point flag for both is set to true. After having compared all distances, this new list is passed on.

[1]The code in full can be found under `https://github.com/Luichang/door-and-wall-detection`

Wall Extraction Implementation

The wall extraction algorithm now takes the list provided by the break point detection and splits it at the rupture and break points into sublists. This is done to get continuous Wall segments which can be sent into the iterative endpoint fit function. Unintended splits can occur in the splitting process. They happen when the iterative endpoint fit detects a false corner stemming from noise in the scan data. To compensate for this, the angles between the last two Walls are analyzed at each step. If these are too similar the split is reverted and the two walls become one again. This solution could be further improved by amending values in the iterative endpoint fit. The final list is then returned.

Iterative Endpoint Fit Implementation

The iterative endpoint fit algorithm receives the continuous Wallsegments provided within the Wall extraction implementation.

a list called "list_of_Walls". The separated Walls will be placed, a List of all the Points that were returned in Breakpoint Detection "breakpoints", a start index "start", and an end index "end". From here the distance to the line that is spanned by breakpoints[start], breakpoints[end] to each point between start and end is measured with 1.4. Should the maximum distance of all of these points exceed 6cm this point is likely a Corner. This means that the Iterative Endpoint Fit gets called two more times, once where the start remains the same and the end is the Corner point and once where the end remains the same and the start is the Corner point. Should there be no such point a Wall is created with breakpoints[start] and breakpoints[end] and placed within list_of_Walls.

Door Detection Implementation

The Door extraction algorithm takes the list returned by the Wall Extraction Process as its argument. The distance and angle between each pair of Walls is then determined. The distance is determined with Formula (1.5) and the angle with Formula (1.7). Should the angle be less than 2 degrees or greater than 358 degrees and the distance is between 0.8m and 1.2m the angle of the potential Door is compared to the angles of the two Walls. The angle of the potential Door is calculated using Formula (1.6) where P_1 is equal to the end point of the first Wall and P_2 is equal to the start point of the second Wall. The angle of the two Walls is also determined using Formula (1.6), only as expected with first their start points then their end points. If the angle of the potential Door is within 2 degrees, the potential Door is added to the list of Doors.

Corner Detection Implementation

The Corner detection algorithm takes the list returned by the Wall extraction process as its argument. Each combination of Walls is checked to see if $P_{Wall1-end} == P_{Wall2-start}$. Every pair is then checked to see if the angle between the two is within range of 50 degrees and 310 degrees. Should this be the case, a Corner is set with the two Walls. If a Wall has a rupture or break point, that point is set as a potential Corner.

Corridor Detection Implementation

The Corridor detection algorithm takes both the list returned by the Wall Extraction Process and the list returned by the Corner Detection Process. The first step is to have two perpendicular lines extrude out of each Corner and these lines are added to a list. From here, each line is compared with every Wall for an intersection. If an intersection exists and the distance of this intersection is less than 3 meters to the origin, the algorithm assumes that the line used could be an entrance to a Corridor. This is how the center point of the Corner and the intersection point is determined. If this point is more than 0.3 meters from the Wall and 0.5 meters to the origin, this center point is added as a Corridor point.

5 Experimental Results

5.1 Office Hallway

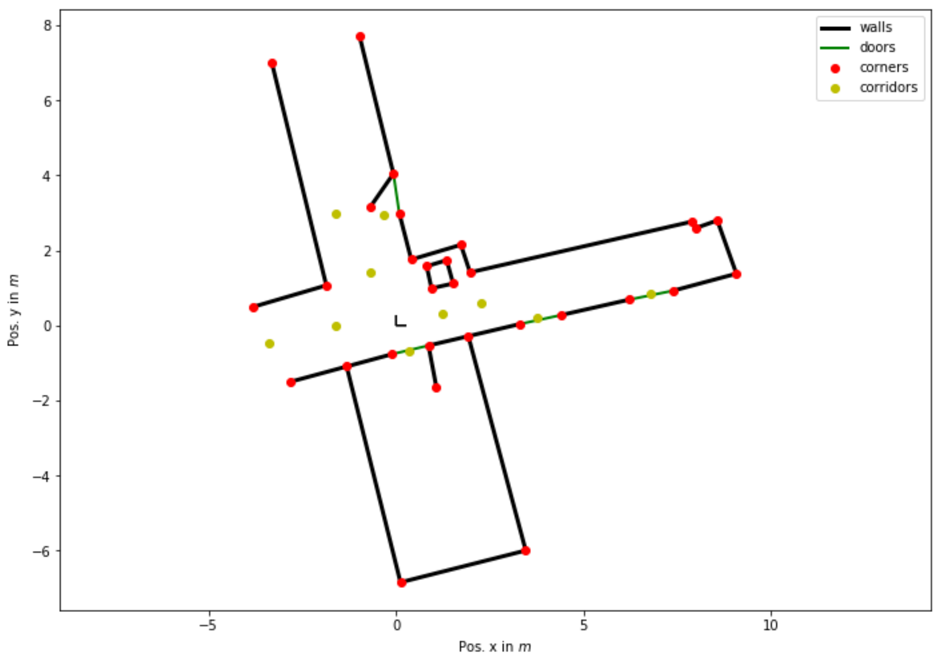


Figure 5.1: Floor Plan 1

Figure 5.1 shows the floor plan with points marked that are provided by humans. Not all features will be visible from every point in the map but all have been noted for a full understanding. The black lines indicate Walls, the green lines indicate open Doors, the red points indicate Corners and finally the yellow points indicate Corridor entrances as the algorithm should be able to determine.

5.2 Measurements

Now that the processes have been described, they need to be analyzed. For this analysis a jupyter notebook created an image for each scan frame indicating the findings of the detection. One specific example has been chosen to be analyzed with a hand crafted layout of that given frame. [1]

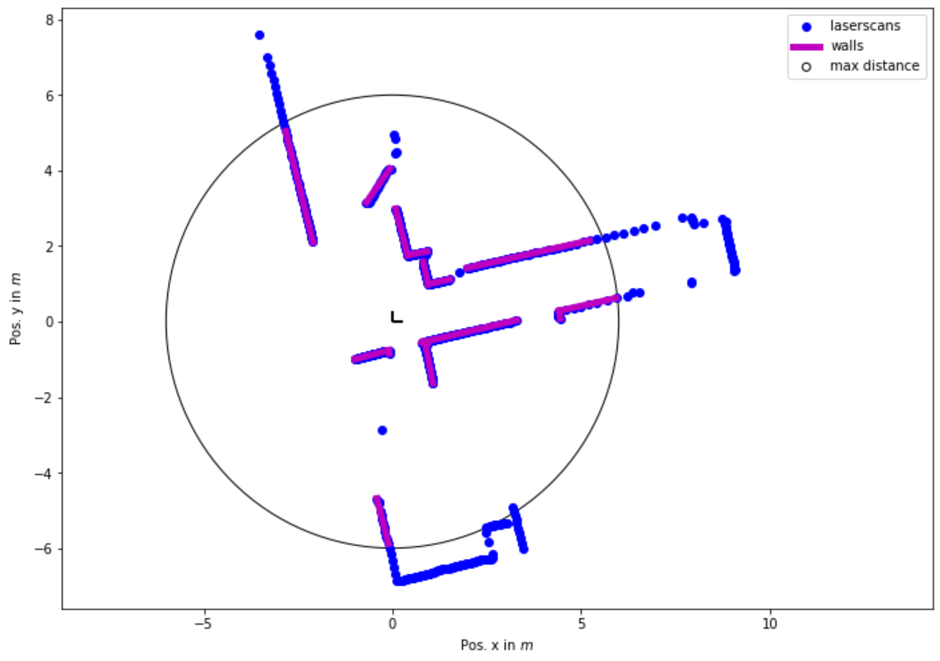


Figure 5.2: Wall Detection 1

Figure 5.2 shows the Wall detection process in action as presented by Borges and Aldon 2004 in Chapter 2.3. Comparing the Walls that the algorithm found with the annotated floor plan, any Wall within range that has the scan points to back it up is detected as a Wall.

[1]The jupyter notebook and bag used to create the image can be found at `https://github.com/Luichang/door-and-wall-detection`

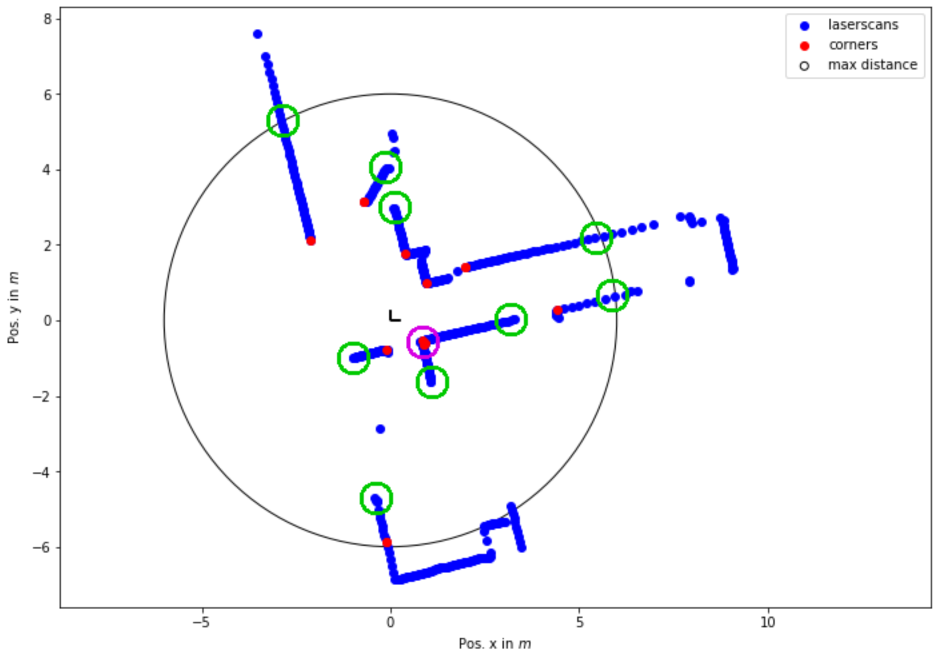


Figure 5.3: Corner Detection 1

Figure 5.3, on the other hand, demonstrates some minor inconsistencies. The detection process found several Corners in the same spot, highlighted by the purple circle. This is likely because the open Door here contains gaps around the Corner area and thus confuses the detection. The points highlighted in green display missing potential Corners. This may be due to the fact that after a Wall is assigned a Corner, the other side is not checked for potential Corners.

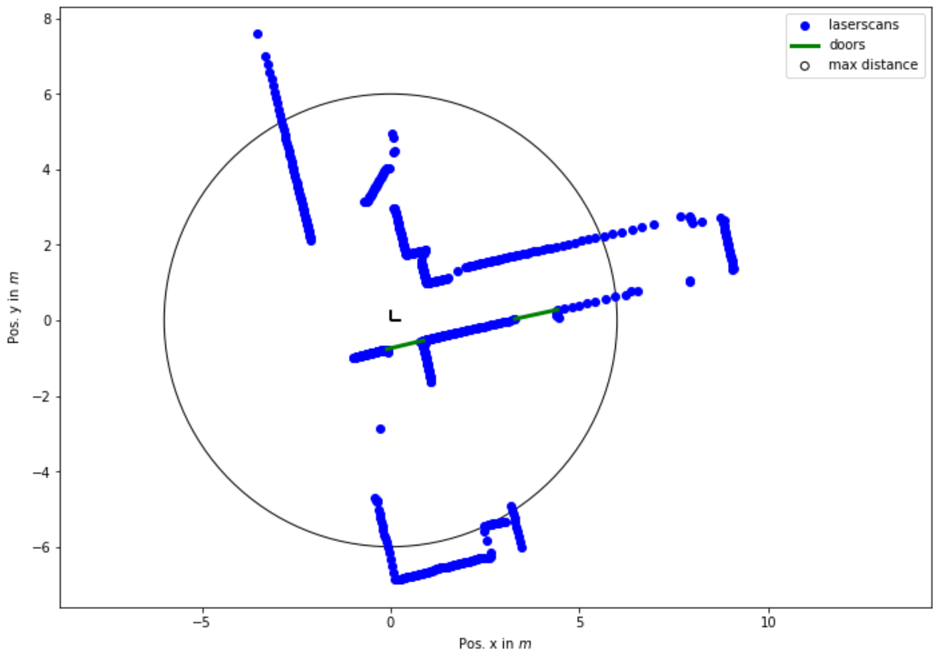


Figure 5.4: Door Detection 1

Figure 5.4 correctly identifies two of the three presently open Doors. The Door that is not detected is at (1,3). The reason this Door is not detected in this frame is because the Wall behind the open Door is obscured. Thus the system can not find a Wall with a similar angle as the Wall on the other side of the Door.

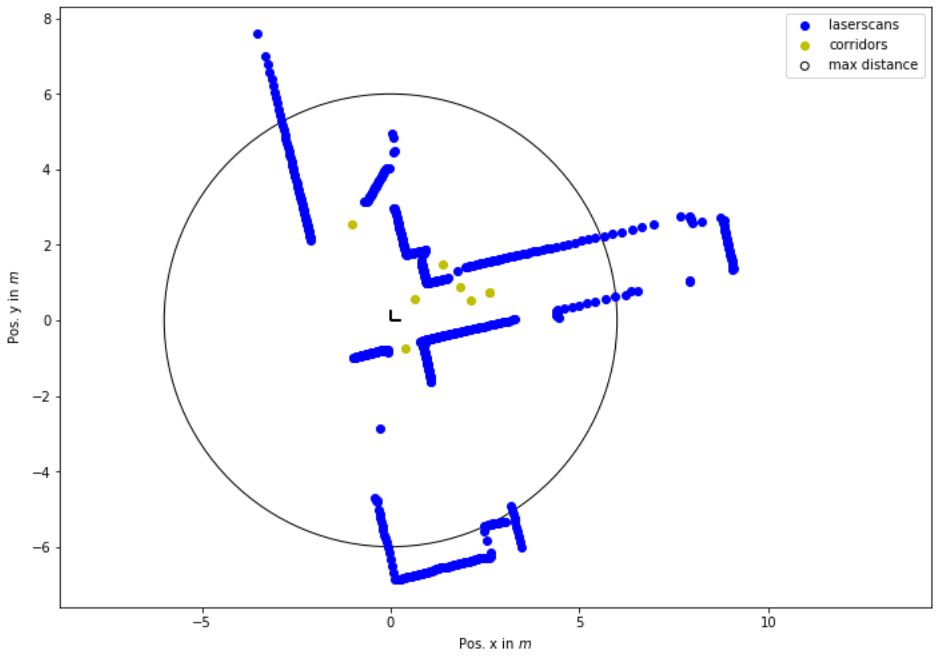


Figure 5.5: Corridor Detection 1

Figure 5.5 shows that the Corridor detection can both provide useful points, such as the Door, and impossible to reach points such as the point behind the pillar.

5.3 Process Evaluation and Analysis of Results

Of all the feature extraction processes, the Wall extraction worked best. The Corner detection proved to have issues with non-smooth objects. The Door detection process seems to work best when the Door is closer to the Robot. The Corridor detection process proved to have the most faults. Most points detected are correct but there still are points detected out of bounds and closer to one side of the Corridor than the other.

6 Conclusion

6.1 Summary

This thesis describes the methodology behind the semantic feature extraction from a 2D range image. It was developed to assist indoor wheelchair navigation, but it can be used for any project that uses laser scan images. For wheelchair users, this can enable greater autonomy of hands off navigation. The result of the work is that at close range the feature detection is very capable of extracting the desired features: open Doors, Corners, Corridor midpoints, and Walls. Detection becomes less reliable as distance increases from the Robot.

The tests have thus far only been performed on offline data. This is in part due to the fact that these detected features are not yet used to influence the wheelchair. The points as of yet are not being compared from scan to scan. This is due to the fact that the detection of Corners does not provide perfectly consistent results.

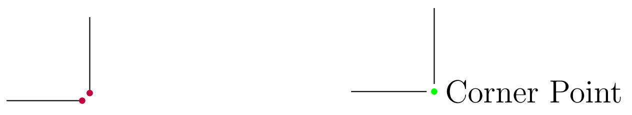
6.2 Future Work

There are some aspects of the detection for which there was not enough time to implement fully or contain parts that can be improved. These include the Corner and Corridor detections.

Future work could include improvements to the code to identify wall breaks and corners better and to modify the width of doors.

6.2.1 Corner Detection

One way the Corner detection could be improved is by analyzing the distance and angle between two Walls. In the event of these being close to each other and around a 90 degree angle a point can be calculated at the intercept point. In the event that they do not directly intercept but are still with in a certain distance both Walls can be extended until such an interception exists. This new point can be used to correct uncertainties that come from the scan image.



(a) How the Wall extraction interprets the scan image

(b) Where the corner probably really lives

Figure 6.1: Corner Detection improvement possibility

6.2.2 Procrustes Algorithm

Another aspect in which the detection can be improved in the future is by attempting to remember features from one image to the next. A possible way to do this is by using the Procrustes Algorithm. This algorithm takes two sets of features and attempts to scale, translate and rotate one set so it can be fit over the other. Should this be made to work with the suggested feature detection of this thesis, information such as odometry data can be extracted. The odometry data obtained this way would not contain the uncertainty of sensor sliding but instead be able to locate itself relatively to its surrounding.

6.3 How this can improve peoples lives?

Bibliography

[Agh+07] Ali Aghamohammadi et al. “Feature-Based Laser Scan Matching For Accurate and High Speed Mobile Robot Localization.” In: Jan. 2007.

[BA04] Geovany Araujo Borges and Marie-josé Aldon. “Line Extraction in 2D Range Images for Mobile Robotics”. In: *Journal of Intelligent and Robotic Systems* (2004).

[Núñ+06] Pedro Núñez et al. “Feature extraction from laser scan data based on curvature estimation for mobile robotics”. In: vol. 2006. June 2006, pp. 1167–1172. DOI: 10.1109/ROBOT.2006.1641867.