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 to be able to automatically detect navigation situations and add a semantic aspect to the wheelchairs' driving ability. Concretely, this means eventually to be able to tell the wheelchair to go down the hallway, turn left at the intersection and go in the second Door on the right and have the wheelchair know exactly what we mean and execute this. This work expands on the work of [line extraction] 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 between two Points
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. That will make navigating a wheelchair as easy as talking to another person. While this thesis 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.

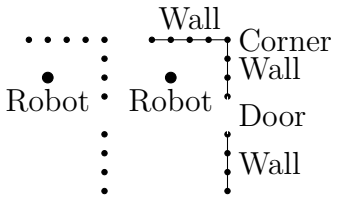


Figure 1.1: Human interpretation of a laser scan image

1.3 Laser Scan Message in ROS

The ROS (Robot Operating System) environment carries many predefined "messages". These messages bridge the gap between the physical space and the data moldable by

computer code. For this thesis, the LaserScan message is used as input. This LaserScan message is comprised of three angle values and a list of ranges. It also contains more data which was not used in this thesis. The three angles are made up of the minimum angle, maximum angle and the angle increment. For the laser scanner used throughout the thesis the values are

$$\begin{aligned} \alpha_{min} &= -2.356194496154785 \\ \alpha_{max} &= 2.3557233810424805 \\ \alpha_{increment} &= 0.00581718236207962 \ . \end{aligned} \tag{1.1}$$

These angles are provided in radians. The ranges list has the length of $\lceil \frac{(\alpha_{max} - \alpha_{min})}{\alpha_{increment}} \rceil$. The first index 0 corresponds to α_{min} and every successive index n is determined by

$$\alpha_n = \alpha_{min} + (n * \alpha_{increment}) \ . \tag{1.2}$$

The value at ranges[n] is the distance from the Laser Scanner to an object.

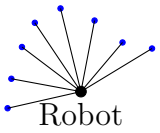
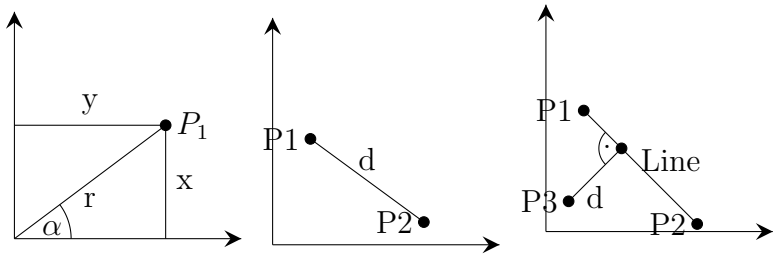


Figure 1.2: Examples of ranges

1.4 Basic Formulæ

In this section the formulæused throughout this thesis are introduced. These formulæshould 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 cattesian coordinate system. Figure 1.3a shows an example of how the angle and distance can represent a point in both the polar and coordinate systems. By using

$$\begin{aligned} x &= r \cdot \cos(\alpha) \\ y &= r \cdot \sin(\alpha) \end{aligned} \tag{1.3}$$

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

$$d(P_1, P_2) = \sqrt{(p_{2x} - p_{1x})^2 + (p_{2y} - p_{1y})^2} \ . \tag{1.4}$$

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

$$d(P_1, P_2, P_3) = \frac{|(p_{2y} - p_{1y}) \cdot p_{3x} - (p_{2x} - p_{1x}) \cdot p_{3y} + p_{2x} \cdot p_{1y} - p_{2y} \cdot p_{1x}|}{\sqrt{(p_{2y} - p_{1y})^2 + (p_{2x} - p_{1x})^2}} \ . \tag{1.5}$$

Here the desired distance is the shortest distance from a point to a Wall. The shortest distance will be perpendicular to the line that is spanned by the Wall start and end.

Minimum Distance between two Lines

To get the minimum distance between two Lines, the distance between the start and end points of one line is compared with the other line. Of these four distances, the minimum is then returned as follows:

$$\begin{aligned} d_{min}(P_{n-start}, P_{n-end}, P_{n+1-start}, P_{n+1-end}) = \min(\quad & d(P_{n-start}, P_{n-end}, P_{n+1-start}), \\ & d(P_{n-start}, P_{n-end}, P_{n+1-end}), \\ & d(P_{n+1-start}, P_{n+1-end}, P_{n-start}), \\ & d(P_{n+1-start}, P_{n+1-end}, P_{n-end})) \ . \end{aligned} \tag{1.6}$$

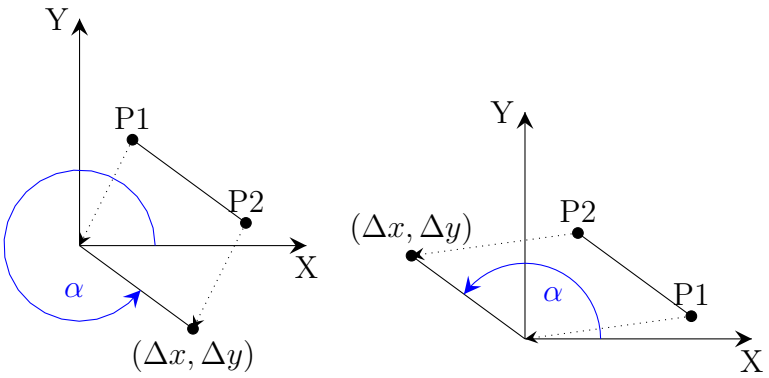


Figure 1.4: Angle between two Points (i.e. angle of a Line)

Angle of a Line

$$\alpha(P_1, P_2) = \text{atan2}(\Delta y, \Delta x) = \text{atan2}(p_{2y} - p_{1y}, p_{2x} - p_{1x}) \tag{1.7}$$

uses the two parameter arctan to determine the angle between two points. The atan2 function <add reference to math.atan2 from python> 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

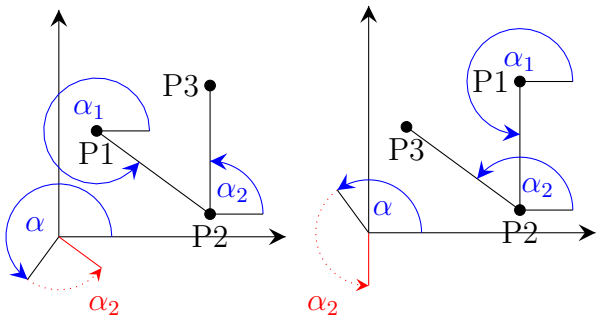


Figure 1.5: Angle α between three Points (i.e. Angle between two lines)

$$\alpha(P_1, P_2, P_3) = \alpha_1(P_1, P_2) - \alpha_2(P_2, P_3) \tag{1.8}$$

involves applying Formula (1.7) twice. This is shown in Figure 1.5. As with the angle of a Line, the order of points matters.

2 Line Extraction in 2D Range Images for Mobile Robotics

The work of BORGES and ALDON 2004 is used to extract Walls from 2D images. The idea is to find certain points, called rupture and break points, which are explained in Sections 2.1.1 and 2.1.2. These are used to group segments of connected Walls that are split at the Corners. This provides the method for extracting Walls.

2.1 Rupture Detection

The idea of the rupture Detection is quite simple. If there are Points of the laser scan 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.

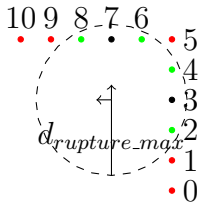


Figure 2.1: The dashed circle indicates the $d_{rupture\_max}$ radius around the Robot. The points in red are ruptured points. Green points are tagged with the rupture flag. Black points do not have a flag.

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 Point 2 is tagged with the rupture flag because of Points 0 and 1 being outside of the range. Points 4 and 6 are tagged with the rupture flag due to Point 5 also being outside of the range. Point 8 is tagged with the rupture Flag due to Points 9 and 10 being outside of the range.

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

2.2 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). A break point 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. Examples of break points are shown in Figure 2.2.

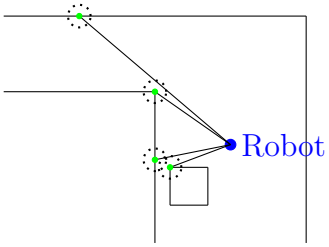


Figure 2.2: Break points are shown in green. d_{break_max} is indicated by each dotted circle around each Point.

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

2.3 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 2.4. 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.

2.4 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 these 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.

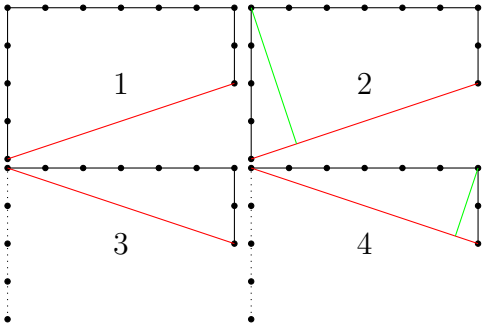


Figure 2.3: Iterative Endpoint Fit process steps

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 reason that it is dotted is that the process recognizes that no point here exceeds the minimum distance to be a Corner. Thus the segment 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 do has a point exceeding the minimum distance. Thus the process terminates and returns all three Walls.

3 Feature Extraction

Chapter 2 has been about the work this thesis builds upon. 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. Difficulties are presented by the relative location of the Robot to each feature. The extraction of these Features are contributions by the author of this thesis.

3.1 Definitions

Before the extraction process is explained, some terms need to be defined first.

- A *Wall*, shown in Figure 3.1, consists of a tuple of two Points. With $W = (P_s, P_e)$. A thing to note here, 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.

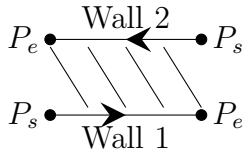


Figure 3.1: Two Walls with their direction indicated by an arrow. The shaded area is navigable

- While under the hood a Door may share the structure of a Wall, that being $D = (P_s, P_e)$, the way it is detected is by finding a gap between two Walls. 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. Because of this assumption and the check for a gap we only need one rupture point, as the Door might still be attached to one side. The gap between the two points is between 0.8 and 1.2 meters.

- A *Corner* is made up of a tuple containing two Walls, which share a Point, and one integer $C = (W_1, W_2, i)$ 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

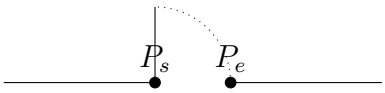


Figure 3.2: Here we have two Walls separated by one open Door

Point of the second Wall. The integer keeps track of the Corner type. Possible Corner types are inner Corner, outer Corner and potential Corner, examples of each show up in Figure 3.4. The Corner type impacts how the Robot can approach the Corner.

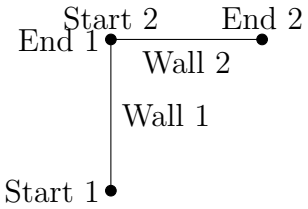


Figure 3.3: A display of what a Corner looks like with its Walls

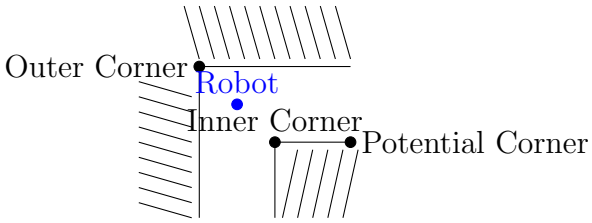


Figure 3.4: Corner Type Examples

- Finally a *Corridor* is represented by a single Point P_c. This point is located between a Corner and a Wall, which indicates the entrance or exit into a given Corridor.

  Due to how the midpoints are detected, the same object humans would describe as a Corridor will contain multiple different Corridor entrances. This is due to the fact that a Corridor can have open Doors or objects in its way that create 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 3.5 shows in what instances the probing Walls get used and when they yield a Corridor midpoint. The 6 red lines indicate probing Walls that do not yield a Corridor midpoint. The line marked with 1 does not provide a Corridor midpoint as 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 from where the probing Wall originates. 5 is too far from the laser scanner and 6 both does not intersect with a Wall and is too far from the laser scanner. All other probing Walls yield a midpoint, even the Door.

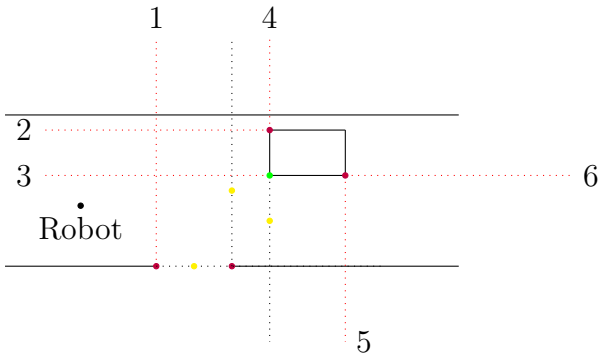


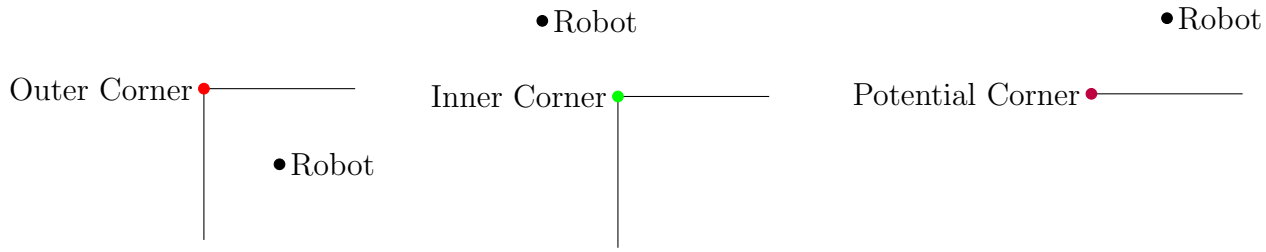
Figure 3.5: The red dotted lines indicate that the search for another Wall yielded for one reason or another no match

3.2 Corner Detection

Now that a Corner has been defined, the next step is detecting them. Currently this is done by comparing the two Walls. Should there be a pair $P_{nstart} == P_{mend}$ the Walls m and n are combined and the Corner point is set as P_{mend}. Now that the Corner has been identified we need to determine on which side of the Corner the laser scanner can traverse. This information is not directly obvious just based on having two Walls, as the Corner in question could either be facing to 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

- Outer Corner is assigned 0. This Corner type points away from the laser scanner as shown in Figure 3.6a.



(a) Outer Corner (b) Inner Corner (c) Potential Corner

Figure 3.6: The Corner Types

- Inner Corner is assigned 1. This Corner type points to the laser scanner as shown in Figure 3.6b

- Potential Corner is assigned 2. A potential Corner occurs when a Wall has either a rupture or breakpoint at its start or end point. This rupture or breakpoint 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 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 (dummy Wall, original Wall, 2). Similarly when 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 (original Wall, dummy Wall, 2). By defining the potential Corner this way the information of it being the start or end point of the Wall is preserved. The reason potential Corners are relevant is that there are many situations where a laser scanner will not be able to accurately determine if a Wall continues or if the hallway turns off. Thus the assumption is made, that any break or rupture leads to a chance of there being a Corner, a potential Corner.

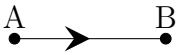


Figure 3.7: A Wall made up by points A and B

3.2.2 Corner Type Detection

When we look at a Corner there are two possibilities for the orientation, an inner Corner and an outer Corner.

As humans we can identify the type by standing in front of the Corner and looking at it. But when we are dealing with machines it is not always that simple. As mentioned above a Corner is defined by the two connecting Walls and an integer, representing the Corner type. When looking at a Corner one can make the observation, that the construct is simply a triangle with the hypotenuse missing. With this in mind we can examine the relative position of the robot to this triangle. When considering that the robot needs to see both Walls to properly determine that it is looking at a Corner we end up with 3 distinct possibilities:

1. We have an inner Corner
2. We have an outer Corner where the robot is within the triangle
3. We have an outer Corner where the robot is outside of the triangle

With this in mind we now need to figure out how to convey this to the robot. One way we can do this is by measuring distances. To fully determine if we are dealing with an inner or outer Corner we will need three distances.

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

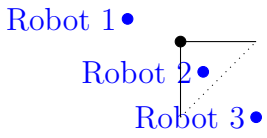


Figure 3.8: Possible Robot Relative Locations

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

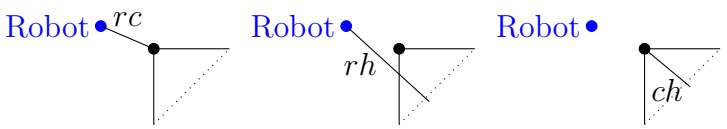


Figure 3.9: All relevant distances

Once we have these three distances we can start comparing these against each other to determine, whether or not the Corner is an inner or an outer. With three variables there are three comparisons to be made:

- the distance from robot to Corner is less than the distance from robot to the missing hypotenuse ($rc < rh$)
- the distance from robot to Corner is less than the distance from Corner to the missing hypotenuse ($rc < ch$)
- the distance from the robot to the missing hypotenuse is less than the distance from the Corner to the missing hypotenuse ($rh < ch$)

By going through each possible combination and creating a visualization one is left with the results of Figure 3.10.

As you may have noticed, the cases (True, False, True) and (False, True, False) did not appear. This is because we are dealing with a 2D plane. If we were on a cylinder or other non flat plane there might be three Points where $rc < rh < ch > rc$ or $rc > rh > ch < rc$, however the space we live in does not have Points which could satisfy these conditions.

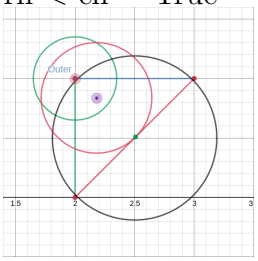
Having determined under which conditions we have an inner Corner or not we can translate this into a Karnaugh map. The result of which is shown in Table 3.1 where a refers to $(rc < rh)$, b refers to $(rc < ch)$ and finally c refers to $(rh < ch)$.

Extracting the information provided by the Karnaugh map

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

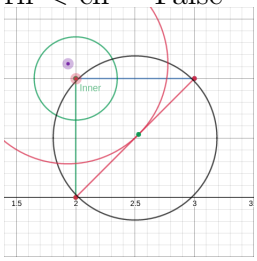
is left. We can now use this Formula to determine the Corner type. The result of Formula (3.1) is a Boolean. In 3.2.1 an Outer Corner was said to hold the integer value of 0 and an Inner Corner to hold the value of 1. So the Boolean is directly transferable to the Corner type.

rc < rh = True
rc < ch = True
rh < ch = True



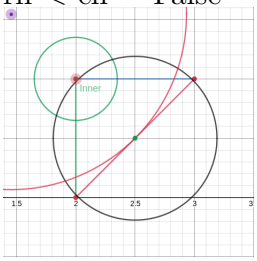
(a) Outer Corner

rc < rh = True
rc < ch = True
rh < ch = False



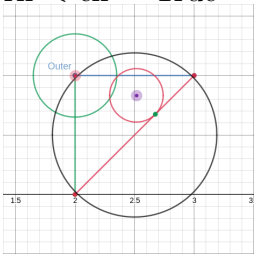
(b) Inner Corner

rc < rh = True
rc < ch = False
rh < ch = False



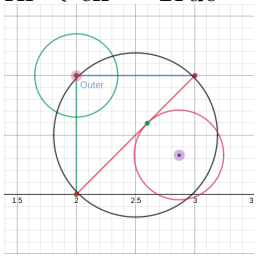
(c) Inner Corner

rc < rh = False
rc < ch = True
rh < ch = True



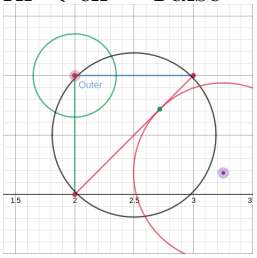
(d) Outer Corner

rc < rh = False
rc < ch = False
rh < ch = True



(e) Outer Corner

rc < rh = False
rc < ch = False
rh < ch = False



(f) Outer Corner

Figure 3.10: Corner Labels

	b		$\overline{b}$	
	$\overline{c}$	c		$\overline{c}$
a	1	0	0	1
$\overline{a}$	0	0	0	0

Table 3.1: Karnaugh map

3.3 Corridor Detection

To detect the Corridor midpoint the Corner types will be made use of. 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. Should such an intersection exist we can assume that the midpoint of the Corner and intersection point is a location the laser scanner can traverse to.

Looking again at Figure 3.5 the inner and potential Corners have probing Walls exiting at 90 and 180 degree angles to the Walls searching for a Wall.

Meta Wall

An early version of the Corridor detection attempted to create a so called *Meta Wall* as shown in Figure 3.11. One reason the definition ultimately moved away from this sort of a Meta Wall idea, was that comparing each Wall with each other proved more complex than initially assumed.

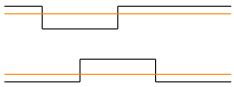


Figure 3.11: Meta Wall

3.4 Implementation Overview

Having explained the theory behind this Thesis, the interesting question becomes how to implement it in a practical manner.

3.4.1 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 way the Rupture Point Detection was implemented was that it started by taking the polar coordinates returned from the laser scan measurements and checking, each individual point if it has a valid length. If this is the case it converts this to Cartesian coordinates using Formula (1.3) and adds a Flag to indicate that this point was not a rupture point. After having converted all points to Cartesian each point is analyzed for a positive rupture flag. Any point that has one of these does not get added for further analysis and the two points before and after it receive the final rupture flag. The final list is then made up of the Point in the Cartesian system, the point in the polar system and the Rupture Flag, which indicates if next to it a rupture occurred.

Breakpoint Detection Implementation

The Breakpoint Detection receives the List of Points and Rupture Flags created in the Rupture Detection. As described in the breakpoint detection the distances between consecutive points is measured and checked against the d_{break_max}. If two Points exceed this distance the breakpoint flag for both is set to true. After having compared all distances this new List is passed on.

Line Extraction Implementation

The Line Extraction now takes the List provided by the Breakpoint Detection. From this list sublists are gathered. These sublists look split at break and rupture points. The effect that this type of splitting has is, each sublist is made up of points that belong to a continuus Wall segment. Each of these continuus Wall segments is then sent to the Iterative Endpoint Fit. After the splitting at the Corners the last two Walls are analyzed for unintended splits, which may occur, when the angle between two Wall segments is too small. In such a case the two Wall segments are fused back together. The final list is then returned.

Iterative Endpoint Fit Implementation

The Iterative Endpoint Fit receives a List "list_of_Walls" wherein 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 1.5. 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 takes the list returned by the Line Extraction Process as its argument. The distance and angle between each pair of Walls is then determined. The distance is determined with Formula (1.6) and the angle with Formula (1.8). 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 (1.7) 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 (1.7), only as expected with first their start points then their end points. If the potential Doors angle is within two degrees the potential Door is added to the list of Doors.

Corner Detection Implementation

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

Corridor Detection Implementation

The Corridor Detection takes both the list returned by the Line 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 added to a list. From here each line is compared with every Wall for an intersection. Should an intersection exist and the distance of this intersection be less than 3m to the origin the algorithm assumes that the line used could be an entrance to a Corridor, so the center point of the Corner and the intersection point is determined. Should this point be more than 0.3m from the Wall and 0.5m to the origin this center point is added as a Corridor point.

4 Experimental Results

4.1 Office Hallway

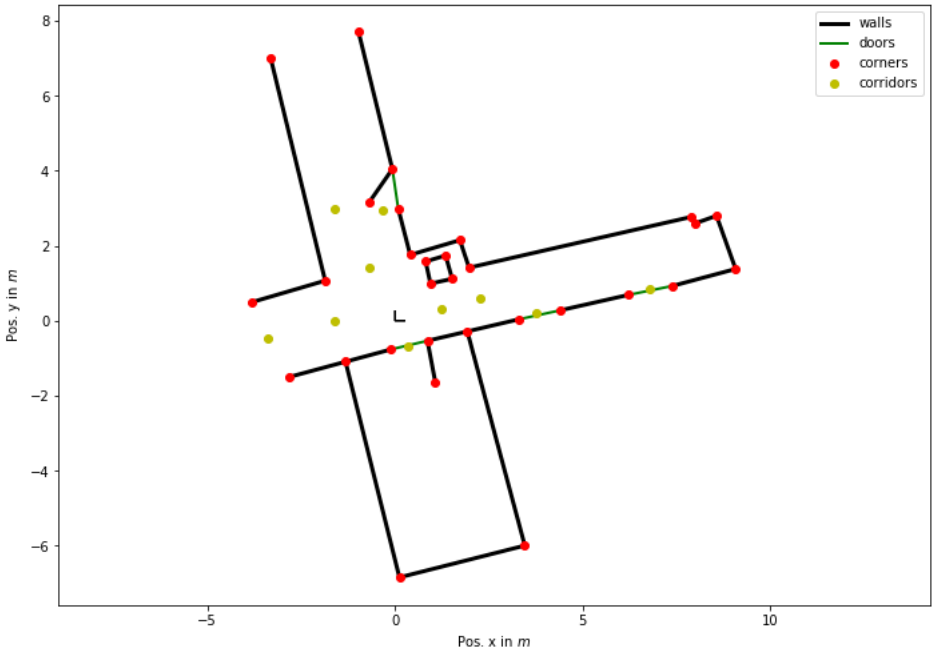


Figure 4.1: Floor Plan 1

Figure 4.1 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 might/should be able to tell.

4.2 Measurements

Now that the processes have been described they need to be analyzed.

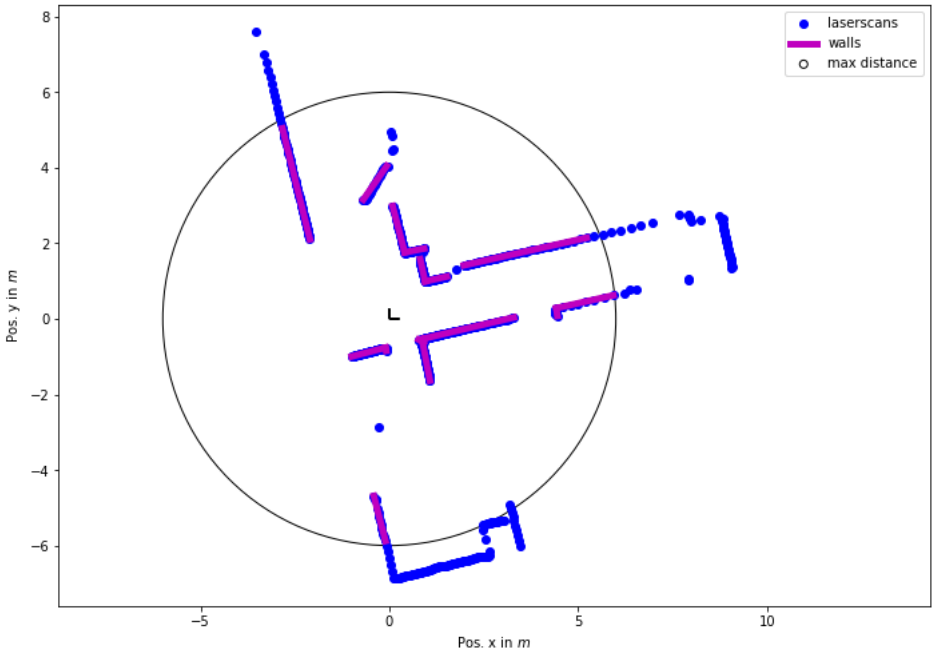


Figure 4.2: Wall Detection 1

Figure 4.2 shows the Wall detection process in action as presented by ¡Paper¿ in Chapter 2.1. 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.

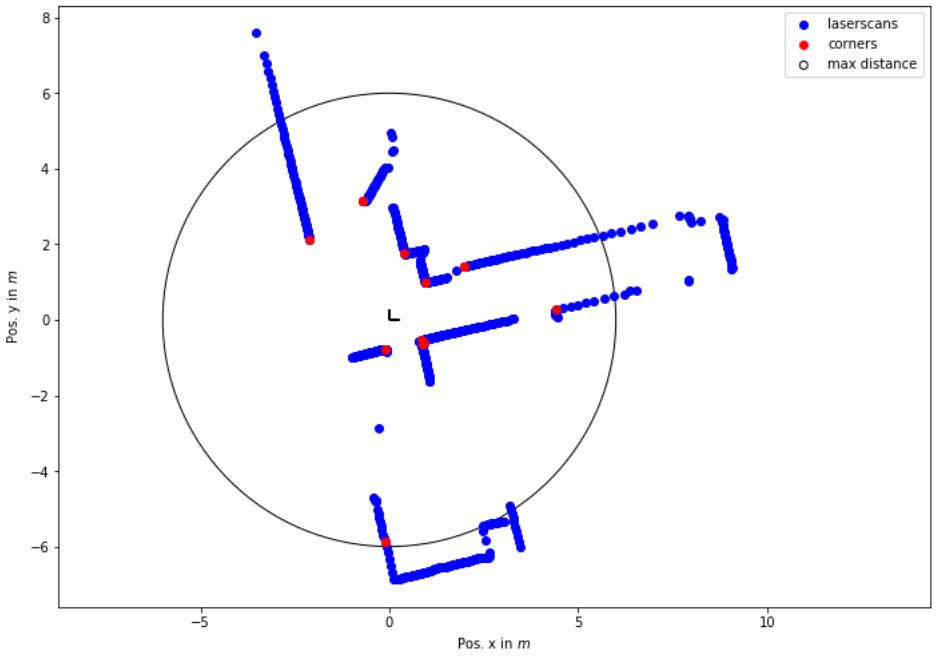


Figure 4.3: Corner Detection 1

Figure 4.3 on the other hand demonstrates some minor inconsistencies. Around (2,-0.5) the detection process has found several Corners, despite there only being one. This is likely due to the fact, that the here open Door contains gaps around the Corner area and thus confuses the detection. Another Corner that has been falsely detected is at (0,-6). Here the Wall is a single straight line.

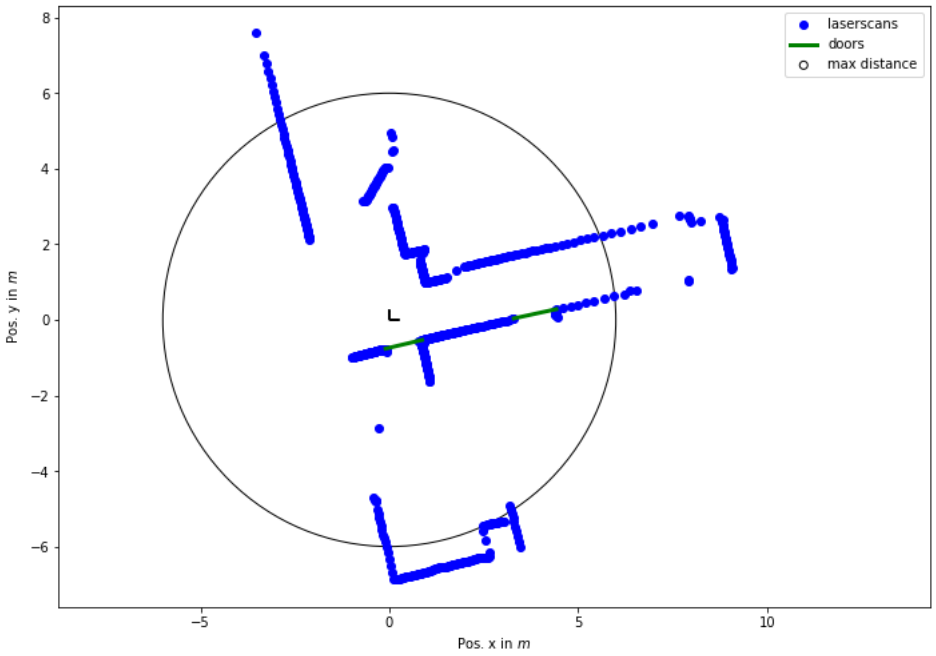


Figure 4.4: Door Detection 1

Figure 4.4 correctly identifies two of the three presently open Doors. The Door that is not detected is at (1,3). The reason for this Door not being detected in this frame is, that 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.

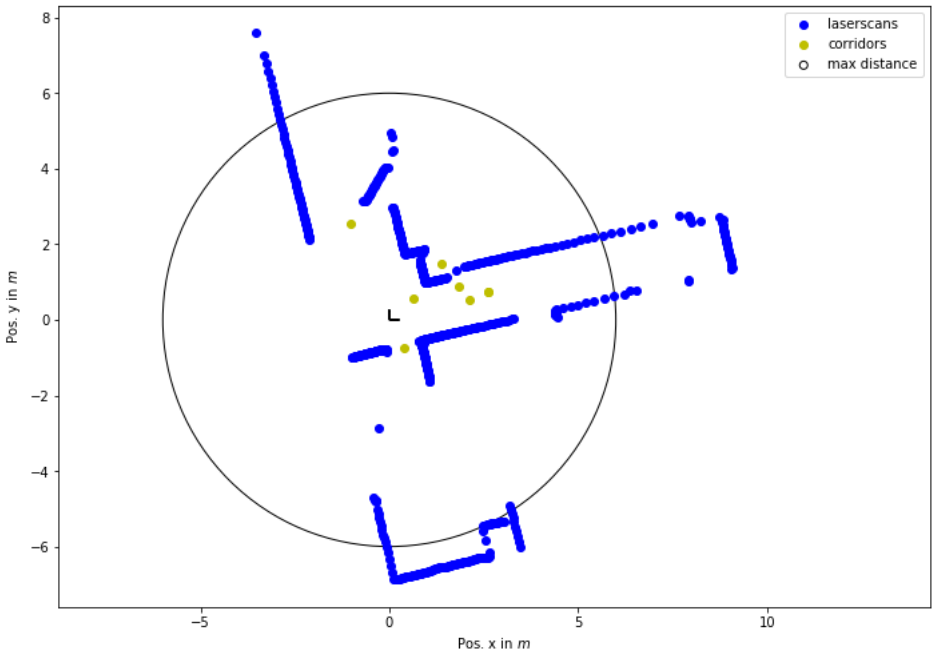


Figure 4.5: Corridor Detection 1

Figure 4.5 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 Conclusion

5.1 Outlook

How well did it work? What were some problems I faced? How can this be used in the future?

5.2 Future Work

What parts can be improved in the future using what things?

5.2.1 Corner Detection

An improvement on this system could involve analyzing the distances between start and endpoints. If the two are within a certain distance the Corner point can be set in an area that makes sense between the two Walls. This could be useful in the event that the laser scanner did not perfectly capture the Corner and the two resulting points are too far from each other to be grouped by the program.

5.2.2 Procrustes

The Procrustes algorithm could be a start for looking into recognizing Corners. This could be useful to determine relative position at each time step. This in theory can be more precise than just using the odometry information, as we base the position on features in the environment.

References

Appendix