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 features such as *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. 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*. The feature extraction developed in this work can be used for automatic labeling of navigational situations.

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 like humans use. This list of terms includes *Walls*, *Doors*, *Corners*, and *Corridors*. By defining semantic terms in a way that humans might use and that a computer can understand, it is much easier for humans to communicate with machines. 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. Therefore the same extraction processes can be utilized by any mobile 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.

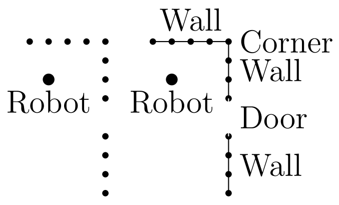


Figure 1.1: Human interpretation of a laser scan image

The assumption for this thesis is that the robot traverses an office building. In this environment, the objects detected are invariant to height. There are no objects in the area that are not detected by the laser scan. It also assumes that all *Walls* are flat and have no curvature.

1.3 Laser Scan

This work only takes distance measurements from a laser scanner as input data. The laser scanner that was used and the expected format for the scan image are explained in the subsections 1.3.1 and 1.3.2.

1.3.1 Laser Scanner

The specific laser scanner that was used to record the laser scan images is a "2D LiDAR sensors TiM561"[1]. 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:

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

1.3.2 Laser Scan Image Structure

Scan images are expected as 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 the angular resolution. The angle at any given index i is determined by $\alpha_i = \alpha_{min} + (i \cdot \alpha_{increment})$.

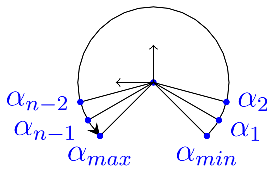


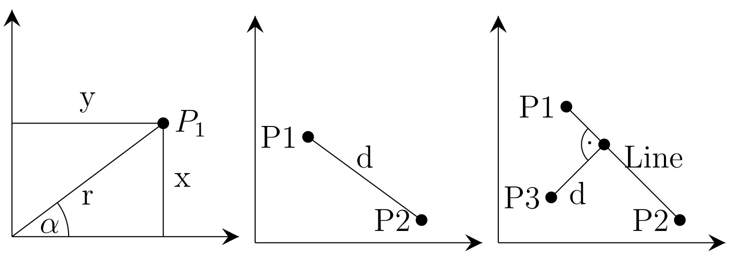
Figure 1.2: Examples of ranges

Figure 1.2 shows the possible range of the laser scanner. The potential cone that represents the scan data is displayed as follows: it goes from α_{min} to α_{max}, incrementing from each angle to the next in steps of $\alpha_{increment}$. Thus $\alpha_i = \alpha_{min} + (i * \alpha_{increment})$ until reaching α_{max} with $\alpha_{max} = \alpha_n = \alpha_{min} + (n * \alpha_{increment})$.

[1] https://cdn.sick.com/media/pdf/6/46/446/dataSheet_TiM561-2050101_1071419_en.pdf

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: Examples for Formulæ

Converting Polar Coordinates to Cartesian Coordinates

Formula (1.2) converts polar coordinates to Cartesian coordinates. This conversion is required as the input of each point is in polar coordinates and the algorithms use the Cartesian notation. 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

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

the x and y coordinates can be extracted using 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 break point detection, which is covered in Chapter 2.

Euclidean Distance between a Line and a Point

Figure 1.3c offers a visualization for

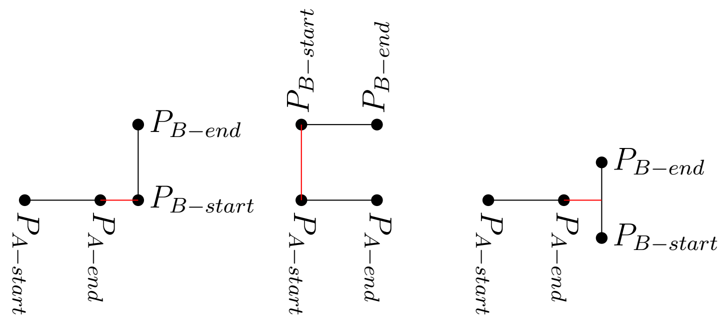
$$d(P_1, P_2, P_3) = \frac{|(p_{2y} - p_{1y}) \cdot p_{3x} - (p_{2x} - p_{1x}) \cdot p_{3y} + p_{2x} \cdot p_{1y} - p_{2y} \cdot p_{1x}|}{d(P_1, P_2)} \quad . \tag{1.4}$$

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

Formula (1.4) is applied to non-intersecting lines A and B in four steps. The first two distances are line A to the start of line B and line A to the end of line B. The second two distances are line B to the start of line A and line B to the end of line A. Of these four distances, the minimum is the result as shown as follows:

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



(a) min dist a (b) min dist b (c) min dist c

Figure 1.4: Examples of minimum distances shown with the red lines

Angle of a Line

A line is defined by two points: the start point and the end point. Using these two points, the Formula

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

extracts the angle by using the two parameter arctan function. The atan2 function[2] determines the angle between the x axis and the line spanned by the origin and a point.

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

To measure the angle between the x axis and a line spanned by any two given points P_1, P_2, the line is first translated so that P_1 is situated in the origin. This is done by subtracting P_1 from P_2. Now atan2 can be used to obtain the angle between P_1P_2 and the x axis.

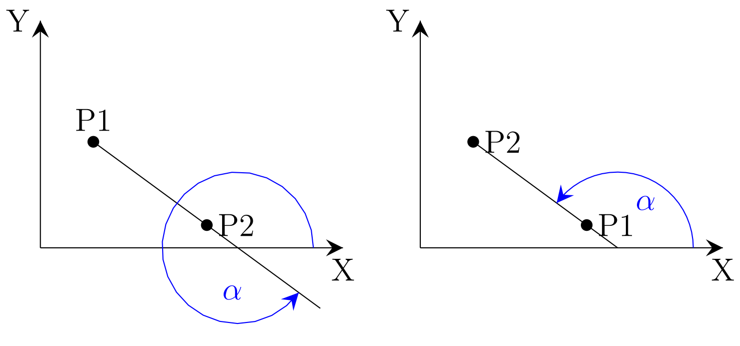


Figure 1.5: Angle of a Line

Angle between two Lines

When two lines are connected, they are defined by three points: P1, P2, and P3. The first line is defined by points P1 and P2 and the second line is defined by points P2 and P3. To determine the angle between these two lines, the difference of angles is taken.

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

Figure 1.6 shows the subtraction process to determine the angle between the two lines. To note $\alpha(A, B, C) \neq \alpha(C, B, A)$

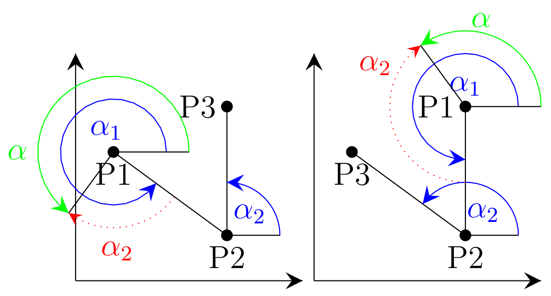


Figure 1.6: Angle α between two connected lines

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 2.4 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

Aghamohammadi et al. 2007 introduced a methodology to compare key points from two consecutive scans. This was an early consideration as the baseline for this thesis, as the generated map seemed to offer reliable *Corner* points. However, the focus of the paper was more on mapping these key points to each other between scans and less on the feature extraction. Additionally, the extracted features did not seem to have the desired flexibility for extracting *Doors*, *Walls* 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. Two consecutive scan range images are compared by calculating the displacement between two scans. This is done by taking a translation and rotation matrix into consideration and attempting to minimize the difference between the two images.

2.2 Procrustes Algorithm

Another method for comparing two consecutive scan images is the Procrustes algorithm. As described by Crosilla 2003, two matrices, which in our case would be the two consecutive scan images, are subjected to similarity transformations until a difference metric has been minimized. For now, the comparison between two scan images exceeds the scope of this thesis.

2.3 Feature Extraction from Laser Scan Data based on Curvature Estimation for Mobile Robotics

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, because the premise of this thesis is that the robot moves in an office hallway 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.

2.4 Line Extraction in 2D Range Images for Mobile Robotics

The work of Borges and Aldon 2004 is used to extract lines from 2D range 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 2.4.1 and 2.4.2. These are used to group segments of connected *Walls* that are split at the *Corners*. This provides the method for extracting *Walls* out of laser scan data. All extraction algorithms used in this thesis are based on said method.

2.4.1 Rupture Detection

The rupture detection follows a simplistic approach. If there are points in the laser scan image that exceed the laser scan maximum or a given threshold d_r, 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 is detected that exceeds the d_r threshold, the points before and after it are tagged with the rupture flag. Examples of rupture points can be seen in Figure 2.1.

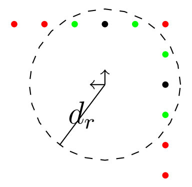


Figure 2.1: The dashed circle indicates the d_r radius around the robot. The points in red are ruptured points. Green points are tagged with the rupture flag. Black points do not have a flag.

In Figure 2.1 the red points are outside of the range of d_r. The green and black points are within the range of d_r. The green points are tagged with the rupture flag.

$$d_n > d_r \tag{2.1}$$

2.4.2 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 scanner, the break point detection analyzes the distances between two consecutive points (P_{n-1}) and (P_n). Break points occur when the distance between the two points is greater than d_b.

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

d_b 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.

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

For example, the distance of d_b 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.

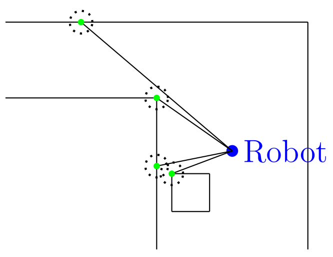


Figure 2.2: Break points are shown in green. d_b is indicated by each dotted circle around each Point.

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

2.4.3 Wall Extraction

After the rupture points 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 2.4.4. After the first segment is completed, the process continues where it left off, working its way in the same manner in order to group the points into connected segments.

2.4.4 Iterative Endpoint Fit

The Iterative Endpoint Fit process takes one continuous set of points and breaks these into individual *Wall* segments. It does this 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.

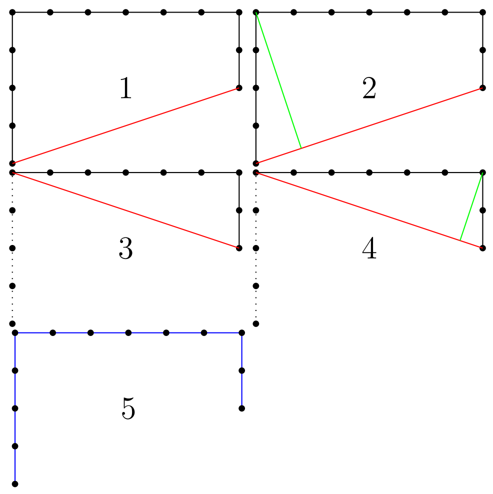


Figure 2.3: Iterative Endpoint Fit process steps

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*, as shown in blue in Step 5.

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 the main contributions of this work.

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 as 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 to note 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*.

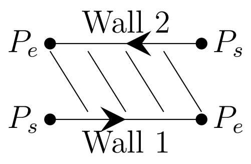


Figure 3.1: Two Walls with their direction indicated by an arrow. The shaded area is navigable

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.

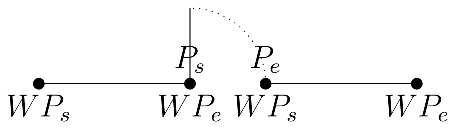


Figure 3.2: Here we have two Walls separated by one open Door

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*.

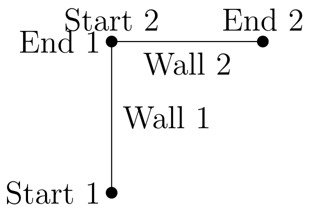


Figure 3.3: A display of what a Corner looks like with its Walls

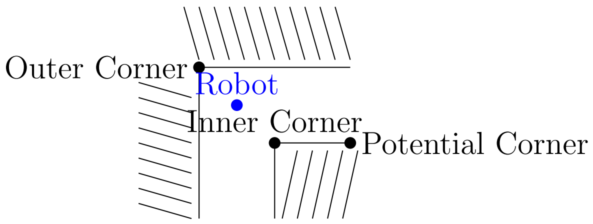


Figure 3.4: Corner Type Examples where the shaded area is occupied

Corridor

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

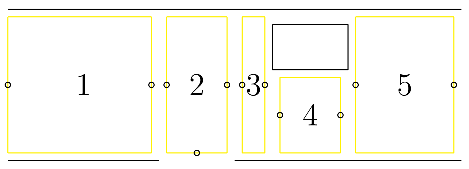


Figure 3.5: Examples of Corridors

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. For the robot to move through the *Corridors*, the distances to these midpoints are relevant. If the midpoint is far away from the robot, the calculation can be excluded, since it is not required at this time. Similarly, if the midpoint is too close to the robot, it should not be added because the proximity can cause difficulties with the detection. Examples of what a robot might detect can be seen in Figure 3.6.

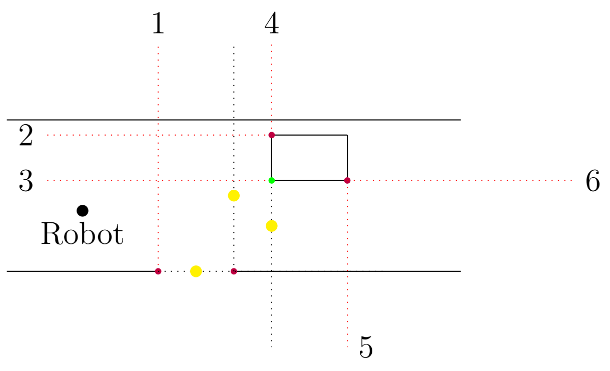


Figure 3.6: The red dotted lines indicate that the search for another Wall did not yield a valid match

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 obvious based only on having two *Walls*. The *Corner* in question could either be facing toward the laser scanner and be an inner *Corner*, or it could 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 two types of *Corners*, full, and potential. A full *Corner* can be subdivided into either an inner *Corner* or outer *Corner*. Examples are shown in Figure 3.4.

- A full *Corner*
  - 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 toward 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 to be a 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

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

  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

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

  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 if the hallway turns off. Thus, the assumption is made that any break or rupture leads to a chance of a potential *Corner*.

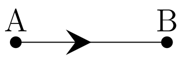


Figure 3.7: A Wall made up by points A and B

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 it. Machine identification of *Corners* and the way they face is not always that 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

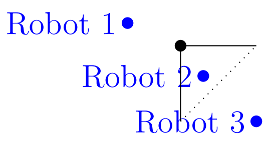


Figure 3.8: Possible Robot Relative Locations

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)

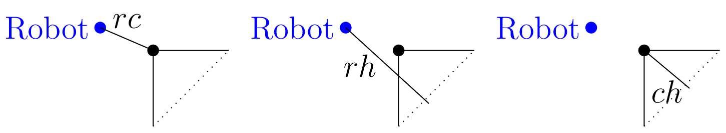


Figure 3.9: All relevant distances

One assumption that can be made here is that if

$$rc < rh$$

the *Corner* is an inner *Corner* and otherwise 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:

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

can be derived. This Formula is 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. Thus the Boolean value 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 *Walls*. 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. A meta wall combines many walls on the same side of a hallway. Thus two parallel meta *Walls* could define a *Corridor*. One reason the definition ultimately changed from the meta *Wall* concept was that comparisons between each *Wall* combination proved more complex than initially assumed. Therefore in this thesis the *Corridor* was defined to be made up of smaller rectangles.

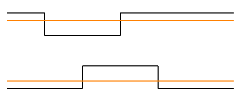


Figure 3.10: Meta Wall

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 Flowchart 4.1. Each function then is explained in subsequent sections in pseudo code.

4.1 Implementation Flowchart

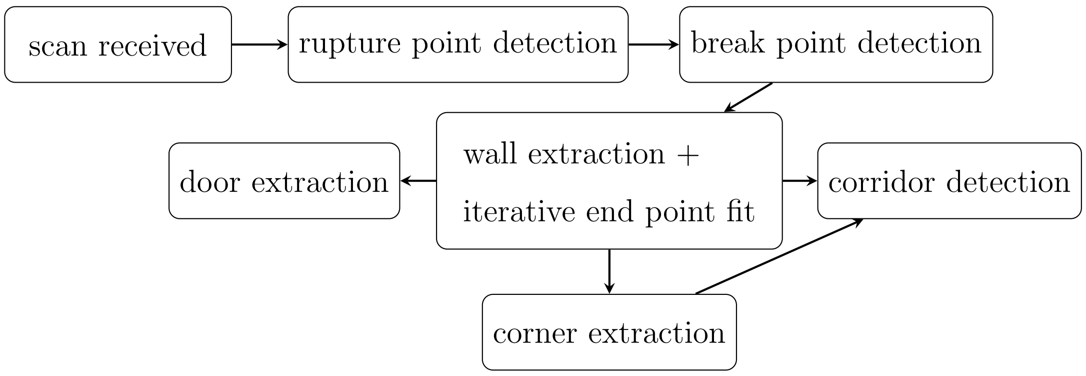


Figure 4.1: Flowchart

The Flowchart illustrates how the result of the wall extraction is used to determine doors, corners and corridors.

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

4.2 Rupture Point Detection Implementation

```
Data: scan from laserscan data
points = empty list ;
forall range, angle in scan do
    if (range != 0.0 meters) and (range < 6.0 meters) then
        convertedPoint = PolarToCartesian(range, angle);
        rupture = False;
        to points add (convertedPoint, (range, angle), rupture);
    else
        set previous rupture = True;
        set next rupture = True;
    end
end
return points
```

Algorithm 1: Rupture Point Detection

The function PolarToCartesian refers to Formula (1.2). It takes a distance and an angle and returns the x and y coordinates.

Each entry into the points list generates a new list. The smaller list is made up by the point in the Cartesian coordinate system, the point in the polar coordinate system, and the rupture flag. The flag indicates if at least one point next to the current point is disregarded.

4.3 Break Point Detection Implementation

```
Data: points from rupture points
for i = 1 to length of points list do
    $distance_b = points[i-1][0] \cdot \frac{sin(\Delta\phi)}{sin(\lambda - \Delta\phi)} + (3 \cdot \sigma)$;
    if d(points[i-1][0], points[i][0]) > $distance_b$ then
        break = True;
        to points[i] add break;
        set points[i-1] break = True;
    else
        break = False;
        to points[i] add break;
    end
end
return points
```

Algorithm 2: Break Point Detection

$distance_b$ is calculated as described in Formula (2.3). The distance between two consecutive points is then determined. If this distance is greater than $distance_b$, the break flag is added as True for the current point. In addition, the previous point's break flag is also set to True. Otherwise the break flag is added and set to False.

4.4 Wall Extraction Implementation

```
Data: points from break points
walls = empty list of Walls;
n = 0;
while $n <$ length of points list do
    start = n;
    n += 1;
    while the n-th point neither has a break or rupture do
        n += 1;
        if $n \geq$ length of points list then
            exit inner while;
        end
    end
    if $n$ - start $> 3$ then
        iterativeEndPointFit(walls, points, start, n);
    end
end
return walls
```

Algorithm 3: Wall Extraction

The wall extraction takes the indices between 0 and the length of the points list. It groups these so the first and last point have either a break point or rupture point. Each of these groups is then checked to ensure it has at least three points. Ideally, a *Wall* will have more points. However a cutoff had to be established somewhere. The resulting groups are then sent into the iterativeEndPointFit for it to split these groups into individual *Wall* segments and place them in walls.

4.5 Iterative Endpoint Fit Implementation

```
Data: walls, points, start, end
if (end - start + 1) $\leq$ 3 then
    return
end
farthestDistance = 0.0;
farthestPoint = -1;
for (n from (start + 1) to end) do
    distanceToLine = d(points[start][0], points[end][0], points[n][0]);
    if distanceToLine > 0.06 meters then
        if distanceToLine > farthestDistance then
            farthestDistance = distanceToLine;
            farthestPoint = n;
        end
    end
end
if farthestPoint == -1 then
    add to walls the wall spanned from points[start] to points[end];
else
    iterativeEndPointFit(walls, points, start, farthestPoint);
    iterativeEndPointFit(walls, points, farthestPoint, end);
end
```

Algorithm 4: Iterative Endpoint Fit

The iterative endpoint fit algorithm is recursive. In the pseudo code 4, the explanation of where the variables are defined was left blank. This is because the first call is from another function. Any subsequent call comes from itself. There are four input variables. The first is the list of *Walls*. This is where the algorithm inserts the *Walls* detected. The second is the list of points that was generated by the break point detection. The third is the index indicating the start of the continuous *Wall* segment to be analyzed. The final is the index indicating the end of the continuous *Wall* segment.

Each call of the algorithm ensures that the continuous *Wall* segment contains at least 3 points otherwise it terminates the call. The process follows the idea explained in section 2.4.4. The index of farthestPoint is the splitting point.

4.6 Door Extraction Implementation

```
Data: walls from wall extraction
doors = empty list of Doors;
forall wall1, wall2 in every combination of two walls from walls do
    distance = d(wall1 end point, wall2 start point);
    if distance is within 0.8 meters and 1.2 meters then
        doorAngle = $\alpha$(wall1 end point, wall2 start point);
        wall1angle = $\alpha$(wall1 start point, wall1 end point);
        wall2angle = $\alpha$(wall2 start point, wall2 end point);
        if doorAngle is within 2 degrees of wall1angle then
            if doorAngle is within 2 degrees of wall2angle then
                add to doors the door spanned from wall1 to wall2;
            end
        end
    end
end
return doors
```

Algorithm 5: Door Extraction

Two walls are compared to see the the endpoint of one is within 0.8 meters and 1.2 meters of the start point of the second. If this is the case, the angle spanned by this end point and start point is compared against the angles of the two walls. Should all three be within 2 degrees of each other, the end point and start point are used to create a door and it is added to the doors list.

4.7 Corner Extraction Implementation

```
Data: walls from wall extraction
corners = empty list of Corners;
forall firstWall in walls do
    forall secondWall in walls do
        if firstWall == secondWall then
            skip to next secondWall;
        end
        if firstWall end point == secondWall start point then
            cornerAngle = $\alpha$(firstWall start point, firstWall end point,
              secondWall end point);
            if 50 < cornerAngle < 310 then
                add to corners the Corner spanned by firstWall and secondWall;
            end
        end
    end
    if firstWall has a break or rupture at the start or end point then
        add to corners the potential Corner spanned by firstWall;
    end
end
return corners
```

Algorithm 6: Corner Extraction

Two walls are compared to see if they share a point. The angle of the two walls that share a point is checked to see if it is within 50 degrees and 310 degrees. Should this be the case, the corner is added to the corner list.

4.8 Corridor Extraction Implementation

```
Data: walls from wall extraction, corners from corner extraction
corridors = empty list of Corridors;
perpendicular = empty list of Walls;
forall corner in corners do
    if d(corner first wall end point, position of the robot) $< 3$ then
        add Wall to perpendicular that spans from the Corner point to the point
          90 degrees and 2 meters from one of the Corner Walls;
    end
end
forall line in perpendicular do
    forall wall in walls do
        if wall was part of the corner that made up line then
            continue to next wall;
        end
        if intersection point between wall and line exists then
            if d(intersection, robot) $< 3$ meters then
                midpoint = the point halfway between the corner from the line
                  and the intersection;
                if (d(midpoint, robot) $> 0.5$ meters) and (d(midpoint, corner) $>$
                  $0.3$ meters) then
                    add midpoint to corridors;
                end
            end
        end
    end
end
return corridors
```

Algorithm 7: Corridor Extraction

For every *Corner*, a check is made at a 90 degree angle to search for a *Wall*. These checks test for intersections with a *Wall*. If such an intersection exists, the distance from this intersection to the robot is measured to see if it is less than 3 meters. If this is the case, the midpoint between the *Corner* and the intersection is chosen as the *Corridor* midpoint. If this point is more than 0.5 meters away from the robot, it is added to the corridors list.

5 Experimental Results

The following images come from a sample ride on a wheelchair recording the laser scan data in an office hallway.

5.1 Office Hallway

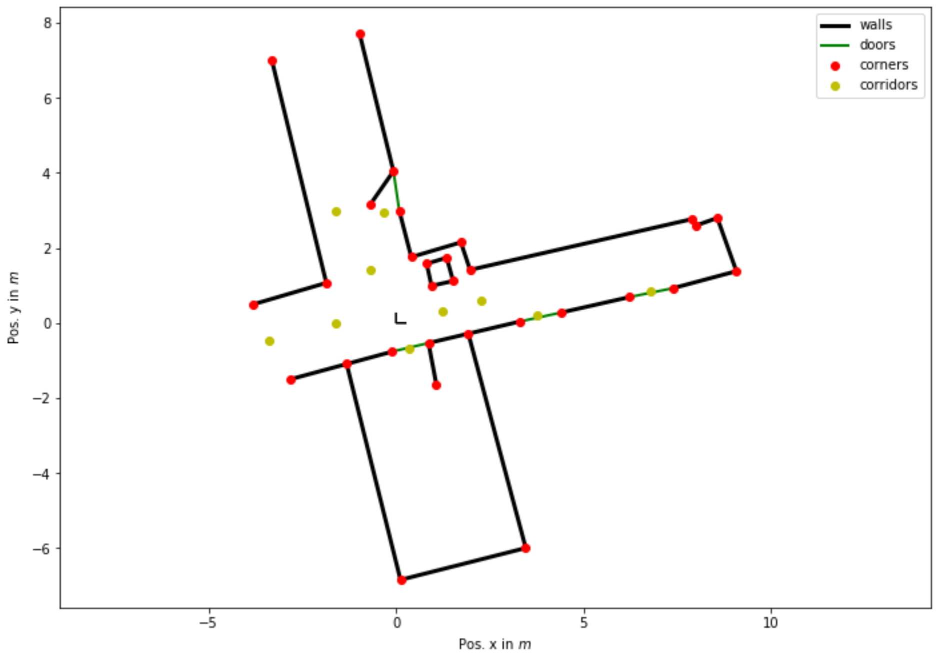


Figure 5.1: Manually annotated features

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 better 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.

5.2 Measurements

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

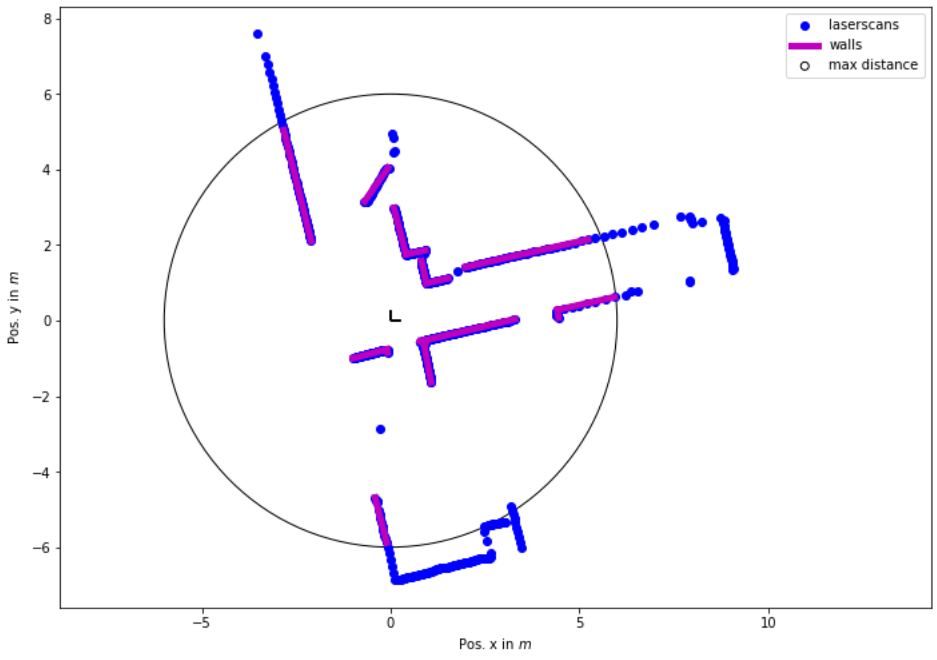


Figure 5.2: *Wall* Detection 1

Figure 5.2 shows the *Wall* detection process in action as presented by Borges and Aldon 2004 in Chapter 2.4. A comparison of the *Walls*, found by the algorithm using the laser scan image, with the floor plan shows that any Wall within range is detected.

[1]The code and the experimental data used to create the image can be found at `https://github.com/Luichang/doors-and-corners-detection`

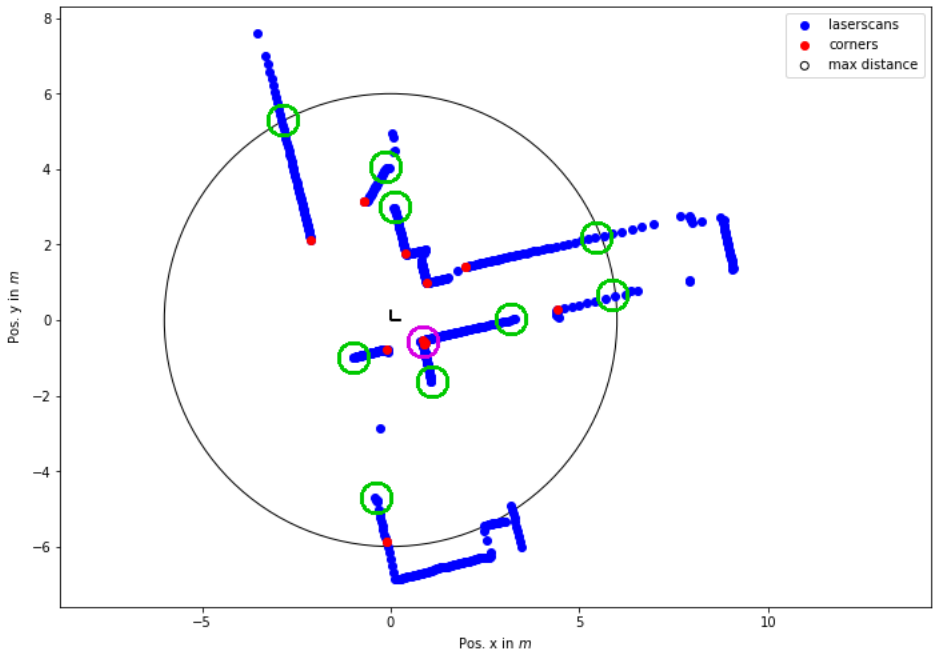


Figure 5.3: Corner Detection 1

Figure 5.3 demonstrates some minor inconsistencies. The detection process found several *Corners* in the same spot. They are 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*.

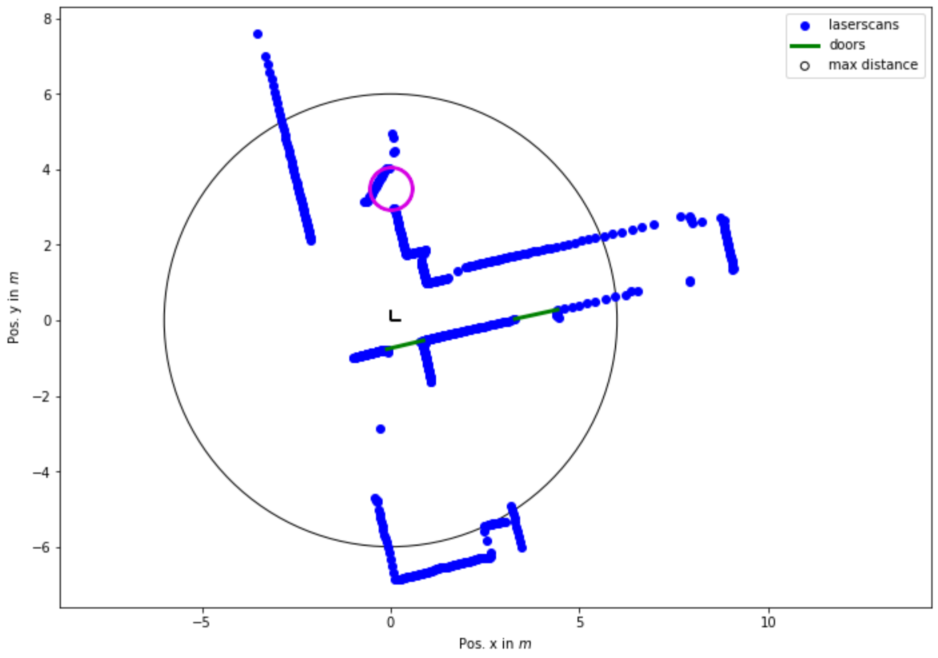


Figure 5.4: *Door* Detection 1

Figure 5.4 correctly identifies two of the three presently open *Doors*. The *Door* that is not detected is indicated by the purple circle. 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*. The way for the door to be detected is for the robot to move to a position where it can see both walls.

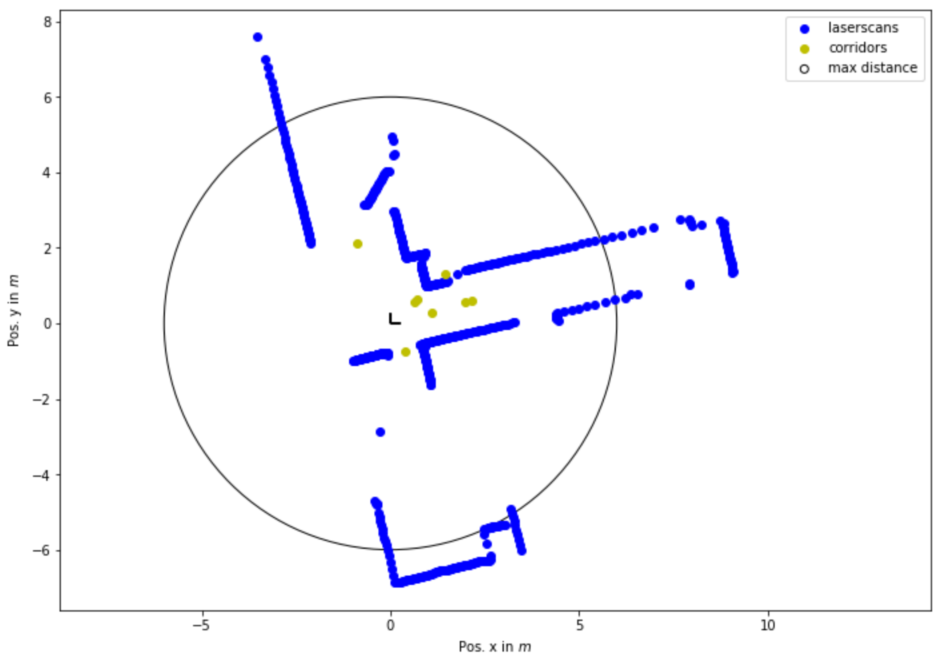


Figure 5.5: Corridor Detection 1

Figure 5.5 shows that the *Corridor* detection can provide both 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.

The evaluation was mainly done by a human evaluating the laser scan data and comparing that with the results of the program. For a quantitative evaluation, a reference dataset with annotated data is needed.

6 Conclusion

This chapter concludes the thesis with a summary and future outlook.

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 data from an indoor environment. 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.

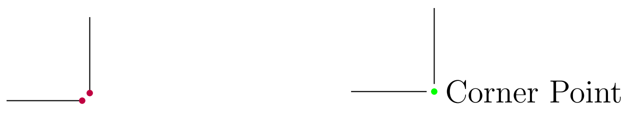
The tests have thus far only been performed on offline data. This is in part due to the fact that these detected features are not yet used to influence the wheelchair. So far the points are not being compared from scan to scan. This is because the detection of *Corners* does not provide perfectly consistent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

6.2 Future Work

Some aspects of the detection were identified, but there was not enough time to implement them fully. Some processes contain parts that can be improved. Future work could include improvements to identify *Wall* breaks and *Corners* better and to modify the width of *Doors*. Some examples follow.

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 *Walls* 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, as shown in Figure 6.1a, both *Walls* can be extended until such an interception exists. This new point, as shown in Figure 6.1b, 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 is really located

Figure 6.1: Corner Detection improvement possibility

6.2.2 Scan Image Recognition

Another aspect in which the detection can be improved in the future is by attempting to remember features from one image to the next. Two possibilities for this have been named, one in section 2.1 and one in section 2.2. With this implemented, information such as the odometry data can be determined. The odometry data obtained this in way would not contain the uncertainty of sensor drift. Instead, it would be able to locate itself relative to its surrounding.

6.2.3 Reference Data Set

After adding a method to localize the position within a map, a manually labeled map will help in the evaluation process. After localizing within this labeled map, the system can check which features it was able to detect and which it missed. This will offer more accurate evaluation data in the future. Once the system is able to recognize enough features, an additional method can add the features automatically to a generated map.

6.3 Real World Applications

With the semantic labeling in place, algorithms can be implemented to allow for commands like "Enter the door" or "turn left at the next corner". This would help wheelchair users have more autonomy and help increase their quality of life by easing the mobility. This could also reduce the reliance on other people. Another way is that the feature detection can be used to automatically annotate large scale laser scan data. This could be then used to evaluate the driving habits of people in wheelchairs. Metrics that can be reviewed may include how an open door is approached. With this a system could potentially prevent collisions with parts of the door by overriding such an input.

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).

[Cro03] Fabio Crosilla. “Procrustes Analysis and Geodetic Sciences”. In: (Jan. 2003). DOI: 10.1007/978-3-662-05296-9_29.

[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.