Geometric Feature Extraction for Indoor Navigation

Charles Barbret

April 2021

Abstract

The intent is to create a system that can detect Walls, Doors and Corners. We want to be able to detect these features to be able to add a semantic aspect to the wheelchairs driving ability. Concretely, this means we wish to eventually 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.

Contents

1 Introduction **4**
1.1 Motivation 4
1.2 Definitions and Symbols 4
1.2.1 Definitions 4
1.2.2 Symbols 5
1.3 Basic Formulas 6

2 State of the art **8**
2.1 Line Extraction in 2D Range Images for Mobile Robotics overview 8
2.2 Formulas 8
2.2.1 Rupture Detection 9
2.2.2 Breakpoint Detection 9
2.2.3 Line Extraction 9
2.2.4 Iterative Endpoint Fit 10
2.3 Procrustes 10

3 Developments and Additions **11**
3.1 Corner Type 11

4 Experiments **13**
4.1 Laserscanner 13
4.2 Office Hallway 13

5 Results **14**

6 Conclusion **14**

1 Introduction

1.1 Motivation

This work aims to set the foundation of machine assisted indoor navigation for electric wheelchair users. It does this by scanning the environment with a laser scanner and converting the scan points into semantic terms that a human can work with. These semantic terms include a Wall, a Door and a Corner. By having semantic terms be something a computer can understand it is much easier for a human to tell the machine With these defined as semantic terms, future works will be able to take these terms and provide end user friendly navigation options. To include the option to tell the wheelchair where to go and have it understand that, as if speaking to a human. While the thesis aims to aid electric wheelchair navigation any example and clarification of situation will use the term Robot, as nothing mentioned in this requires an electric wheelchair.

1.2 Definitions and Symbols

The following terms will be defined in this part and referenced throughout the rest of the work.

1.2.1 Definitions

- In this work a Point refers to a simple tuple of two floating point values.
- A Wall consists of a tuple of two Points. 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.

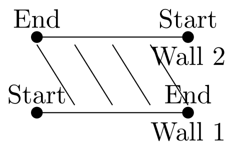

Figure 1: Here we have 2 Walls each made up of a Start Point and an End Point. The order determines to which side we can see and traverse

- Similar to a Wall, a Door too is made up of a tuple of two Points with the added restriction, that the distance be between 0,5 and 1 Meters. (Maybe only count open doors in which case the door becomes a tuple of Points indicating a gap.)
- A Corner is made up of a tuple containing two Walls, which share a Point, and one integer. 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. Possible Corner types

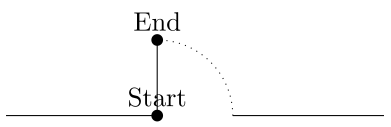

Figure 2: Here we have 2 Walls separated by an open Door

are inner Corner, outer Corner and potential Corner. The corner type impacts how the Robot can approach the Corner.

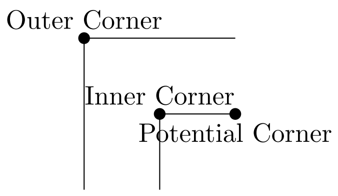

Figure 3: Corner Type Examples

- Corridor := {[List[Wall] a, List[Wall] b]}

1.2.2 Symbols

Pn_x := the x coordinate of the nth Point
Pn_y := the y coordinate of the nth Point

$\underline{\alpha}$ will be used as an angle
r will be used as a radius of a circle
d will be used as the distance between two Points
Robot will be used as a device that can move around and has a Laser scanner attached

1.3 Basic Formulas

In this chapter we introduce Formulas used throughout this work. These Formulas should provide the reader with the tools to fully understand and be able to work with this work.

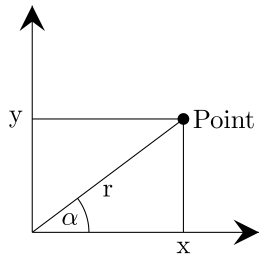

Figure 4: Convert Polar coordinates to Cartesian coordinates

$$\begin{aligned} x &= r * cos(\alpha) \\ y &= r * sin(\alpha) \end{aligned} \tag{1}$$

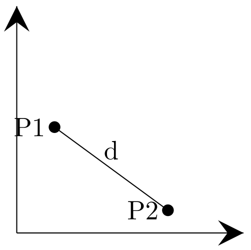

Figure 5: Euclidean Distance between two Points

$$d = \sqrt{(P2_x - P1_x)^2 + (P2_y - P1_y)^2} \tag{2}$$

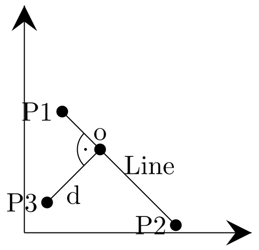

Figure 6: Euclidean Distance between a Point and a line

$$d = \frac{|(P2_y - P1_y) * P3_x - (P2_x - P1_x) * P3_y + P2_x * P1_y - P2_y * P1_x|}{\sqrt{(P2_y - P1_y)^2 + (P2_x - P1_x)^2}} \quad (3)$$

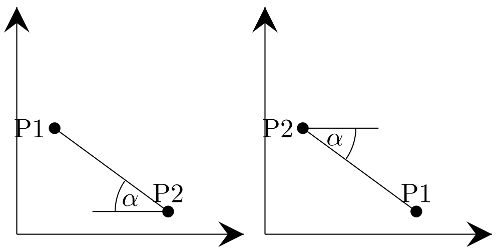

Figure 7: Angle between two Point (aka Angle of a line)

$$\alpha = atan2(P2_y - P1_y, P2_x, P1_x) \quad (4)$$

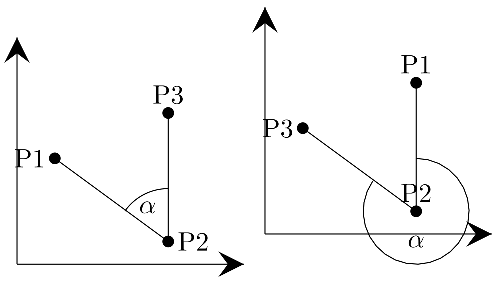

Figure 8: Angle between three Points (aka Angle between two lines)

2 State of the art

2.1 Line Extraction in 2D Range Images for Mobile Robotics overview

The work primarily chose to work with the findings and methods from BORGES and ALDON 2004. This paper focuses on a few things. First off it finds Points that are not connected to its neighbors. What this means is, should there be a larger gap between Point this most likely indicates an occurrence of a break point or a rupture point. In this case a rupture point means a point that is too far away from the scan origin. In a hallway setting this could be when the Wall at the end of the hallway is too far away or when looking through an open Door that leads outside.

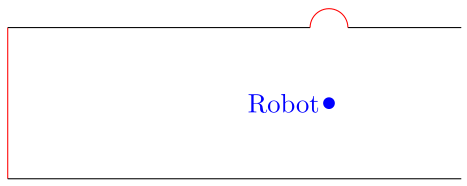

Figure 9: Rupture Points shown in red

A break point on the other hand is defined when there is a larger gap between two Points. This is used to indicate a discontinuous surface. This could happen when an object such as a pillar blocks the line of sight from the robot to the Wall or at intersections.

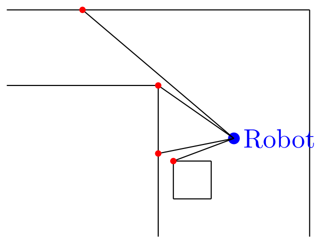

Figure 10: Breakpoints shown in red

After having found the rupture and breakpoints the connected objects need to be split into individual lines. This comes due to the fact that unlike with another object blocking the Wall, when the scan points go around a Corner they tend to be close enough to tell the system, it is the same object.

2.2 Formulas

In this part each formula used in the program is explained.

2.2.1 Rupture Detection

The idea of the Rupture Detection is quite simple. If there are Points of the laser scan that are too far from the Robot we know there is a discontinuity in our surrounding area. This can occur for example, when looking down a long hallway or when looking through a doorway. If this occurs we want to ensure that the system recognizes that any object it is looking at starts or ends with the rupture points.

The execution of this is simply to check the distance of any given Point. If this distance is 0.0 or infinity (depending on your laser scanners settings) or larger than a given maximum distance we set a flag at the previously acceptable Point and the next acceptable Point, while not adding any of the Points that are considered Rupture points.

2.2.2 Breakpoint Detection

The idea of the Breakpoint Detection closely resembles that of the Rupture Detection. While looking at each Point, the distances between the two are measured. In the event, that a pair is farther away than a given threshold, we have a Breakpoint. This happens for example when an object is in front of a Wall, but not connected to the Wall.

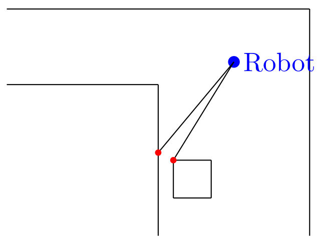

Figure 11: Breakpoints

To do this we first determine a maximum distance D_max by [...]. Now we see if the distance between the last Point and the current Point is less than this D_max. If the distance is greater we set Breakpoint flags to True on both of these Points.

2.2.3 Line Extraction

The line extraction focuses on taking continuous segments and split them at Corners into smaller segments. The first step of this is to group the first Points together until a Point either has a Rupture or Breakpoint associated with it. Once this happens this segment will be considered a continuous segment and sent into the Iterative Endpoint Fit, which is explained next. Now that the first segment is taken care of we continue where we left off, working our way in the same way through all the available Points to group the Points into connected segments that tend to be Walls.

2.2.4 Iterative Endpoint Fit

The Iterative Endpoint Fit process takes one Wall segment and breaks is up at the Corners, should these be in the segment. The way it does this is by taking the first and last Point and connecting these with a line. Now it goes through all Points in between the first and last Point and checks which Point is the farthest away to this line. Should the Point that this process finds exceed a certain length we split the segment at this Point and rerun the process on both segments, until we end up with only straight lines.

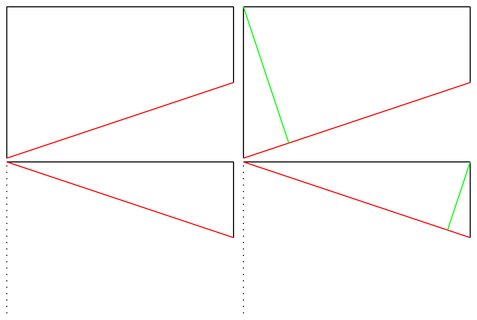

Figure 12: Iterative Endpoint Fit process

2.3 Procrustes

The Procrutes algorithm aims to take two consecutive scans and match them to each other. It tries to find a translation and rotation matrix that can be applied to one scan to set all Point in the reference frame of the second scan.

The algorithm does this by first translating the centers of both scan sets to the origin (0,0). From here both sets are normalized. Finally the first set is rotated to best fit the second set. If the difference is within a threshold we have our rotation and translation matrices.

3 Developments and Additions

3.1 Corner Type

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

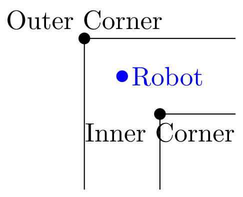

Figure 13: Corner Type Examples

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. Unfortunately to determine whether or not the Corner is an inner Corner we cannot calculate the angle between the two Walls, as we always will get the smaller of the two angles. When looking at a Corner one can make the observation, that the construct is simply a triangle with one of its connecting lines 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
- the shortest distance from the robot to the missing leg of the triangle

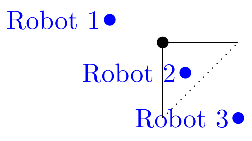

Figure 14: Possible Robot Relative Locations

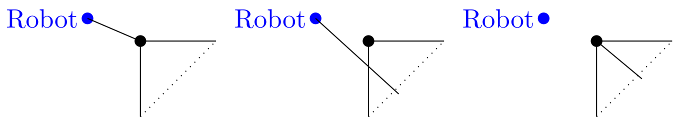

Figure 15: All relevant distances

- the shortest distance from the corner to the missing leg of the triangle

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:

- is the distance from robot to corner less than the distance from robot to the missing leg (r-c ¡ r-l)
- is the distance from robot to corner less than the distance from corner to the missing leg (r-c ¡ c-l)
- is the distance from the robot to the missing leg less than the distance from the corner to the missing leg (r-l ¡ c-l)

If we go through each possible combination and manually check if we are dealing with an inner or an outer corner we find the following:

r-c < r-l = True
r-c < c-l = True
r-l < c-l = True

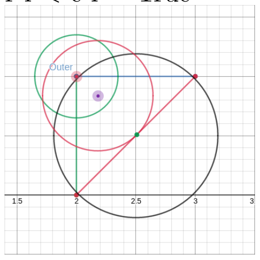

Figure 16: Outer Corner

r-c < r-l = True
r-c < c-l = True
r-l < c-l = False

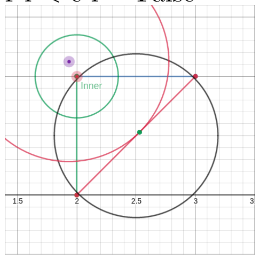

Figure 17: Inner Corner

r-c < r-l = True
r-c < c-l = False
r-l < c-l = False

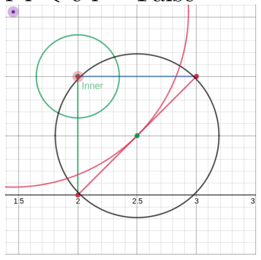

Figure 18: Inner Corner

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 r-c < r-l < c-l > r-c or r-c > r-l > c-l < r-c, 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 and when not we can translate this into a Karnaugh map. The result of which would be:

r-c < r-l = False
r-c < c-l = True
r-l < c-l = True

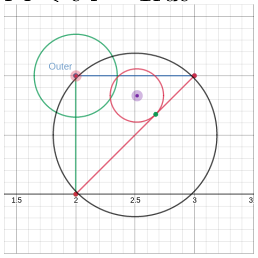

Figure 19: Outer Corner

r-c < r-l = False
r-c < c-l = False
r-l < c-l = True

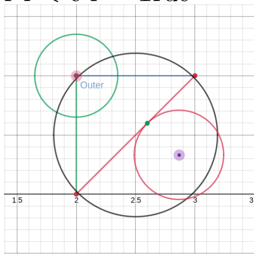

Figure 20: Outer Corner

r-c < r-l = False
r-c < c-l = False
r-l < c-l = False

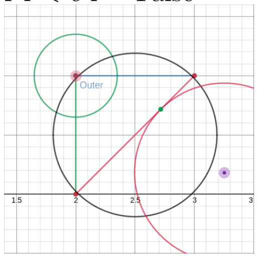

Figure 21: Outer Corner

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

Table 1: Karnaugh map

4 Experiments

4.1 Laserscanner

TODO describe the laserscanner here

4.2 Office Hallway

TODO describe the office hallway in which the scans were taken. Maybe add some pictures

5 Results

TODO describe how well the door, wall, corner, hallway detection works. True/False Positives/Negatives

6 Conclusion

References

Appendix