Chapter 1

Introduction

1.1 The BeachBot Project

The BeachBot project was a "focus project" at ETH Zurich. During the last two semesters of the bachelor studies the team had the opportunity to develop a mobile and autonomous robot for creating sand drawings on beaches. In total 7 mechanical engineering students, one electrical engineering student and two industrial design students (from the Zrcher Hochschule der Knste) where working on the project.
The result of the project is a 3 wheeled mobile robot that can drive autonomously. The key features are:

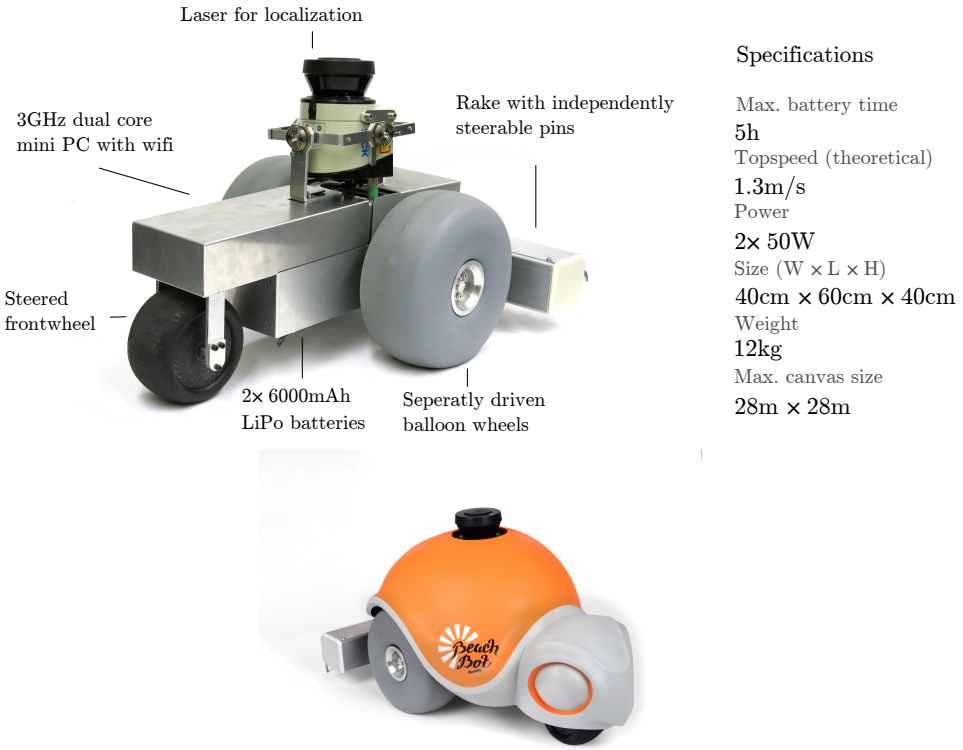
Localization The robot is able to reliably localize itself on the beach, using a laser range finder and 3 or more reflective poles. An localization accuracy of about 3 centimetres was achieved.

Driving speed and turning radius The top speed of the robot is about 0.4 metres per second and it can turn on the spot. Both back wheels are independently steerable. The front wheel is also actuated. This is done to reduce the risk of getting stuck in sand.

Rake The main drawing tool of the robot is a rake. The rake consists of seven pin-pairs which are individually liftable.

Controller The controller uses the output from the localization to steer the robot in a way that it follows a pre-defined trajectory. The trajectory generally is a text file with coordinates.

The robot was tested successfully at the beach.



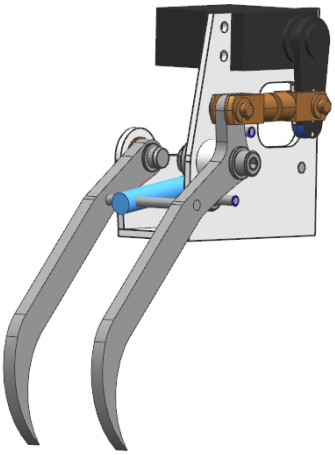


Figure 1.1: Various images of the BeachBot

Chapter 2

Requirements and Inspiration

The BeachBot project itself was inspired by the images of sand artists like Peter Donnelly and Andres Amador, who create large scale sand art at beaches using a rake. Some of the imagery that we found online can be seen in Figure 2.1.
Since our goal was to create drawings similar to those of the artists, we derived our requirements from these sample images.
First of all, and probably the easiest, the BeachBot should be able to draw lines. But even more important is the support of filled areas. While driving lines or curves is straight forward, there is no easy solution how to derive the path for filled areas. Not only should the area be covered to the highest extent, the drawing process should look interesting and artistic to spectators as well (unlike, for example, a printer).
Derived from the pictures is also the requirement that the BeachBot should only be able to draw in two colors. Gradients or differently colored areas are not needed.
Crossing lines or filled areas should be prohibited as good as possible since the balloon wheels and also the front wheels leave visible marks on the raked sand (a humanoid has a huge advantage here, since it can jump over the drawn areas). The effect of driving over the drawing depends on the scale. If the drawing is very big, it does not matter much if part of the image is crossed out.
The input of the path generator should not only be computer readable, but also human editable. Typing in endless lists of coordinates would be tedious in the long run, and having a difficult format to work with would make it hard to collaborate with artists, for example. Therefore the requirement to be able to use a modern graphics editor tool to work with was set. To conform to this requirement we soon decided to use *Inkscape*[1], a popular open source vector graphics editor, as our input creation tool of choice. Further discussion about vector graphics as input and the steps to integrate a standardized vector graphics format into the path generator are explained in **??** and **??** respectively.
During the testing phase it was found out, that some of the generated paths needed some extra care by humans. To be able to easily edit the output of the generator program a graphical user interface should be created to work with the output of the generator program.
In the following sections it will be shown how these requirements were fullfilled and to which extent.

[1] `http://inkscape.org`

(a) Sand drawing by Peter Donnelly. Source: `http://becky-garrett.blogspot.ch/2009/03/sand-dancer-peter-donnelly.html`

(b) Sand drawing by Andres Amador. Source: `http://sftimes.co/?id=25`

(c) Sand drawing by Andres Amador. Source: `http://sftimes.co/?id=25`

Figure 2.1: Various beach drawings by artists

Chapter 3

Path Planning Algorithms

3.1 Algorithm Overview

The output of the path generator should be a single trajectory that completely connects and covers all elements of the drawing.
This happens in three steps: First, the polygons that have to be filled are selected and the fill algorithm is seperately executed for each of the polygons with polygon specific settings. In a second step all of the drawing elements are connected by an open tour and tries to minimize the traveled distance by applying a traveling salesman heuristic. As last step, connections with limited curvature are generated to complete the trajectory.
Each of those steps will be discussed in detail in this section.

3.2 Image Structure

Derived from the requirements, three different elements were identified as part of drawing (also shown in Figure 3.1:

Polyline A line consisting of 2 to n vertices

Polygon A closed line, consisting of 2 to n vertices, where the last segment is a closing one. So that vertex v_{n+1} equals v_0.

Filled Polygon Defined in the same way as the polygon, except that the inner space should be filled by the generated trajectory. Another difference is that the filled polygon can also contain holes, which should not be covered and be excluded from the fill trajectory.

3.3 Polygon Filling

3.3.1 Related Work

Over time several surface coverage algorithms have been developed, and some distinctions can be made.
Coverage algorithms exist for applications like autonomous vaccuum cleaners or autonomous lawn mowers, but also search and rescue robotic applications usually and they usually do not care if they visit the same spot twice. The target for those algorithms is rather to achieve complete coverage of an previously unknown terrain in sensible time. Usually, the complete surface coverage algorithms in are

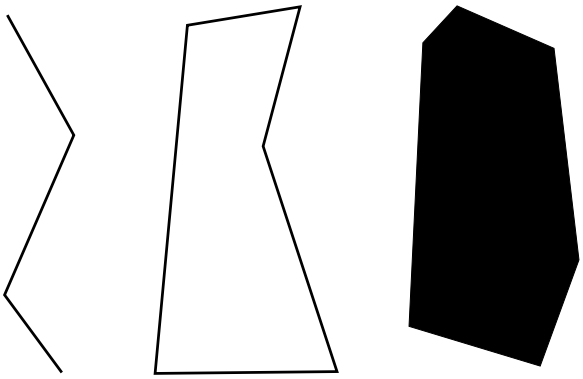


Figure 3.1: Polyline, polygon and filled polygon

also connected with online map generation techniques *SLAM*, whereas the path generation for the BeachBot should happen offline.
However, in agricultural applications some interesting algorithms have been found, which served as inspiration for the explorations presented in this thesis. Especially [1], who hinted at exploiting the straight skeleton algorithm to generate the inset polygons. An optimization strategy is employed to find the shortest trajectory through the field by repeatedly offsetting the remainding shape and traversing all possible ways off filling the shape. The algorithm is relatively computing intensive, what might be justified when using huge agricultural machines but what was not necessary for the goals of this thesis as the benefit of this optimization would be relatively small.
Another field where trajectories have to be generated is in *Computer Aided Machining*. The process of removing layers of material from a block of metal is quite similar (though inverse, usually) to what is achieved in this thesis. Many publications deal with the problem of multi-axis milling machines which are far more complex and are also able to go over the machined surface without problem because they can lift the machnining head – something that is not possible with the autonomous ground robot.
One interesting publication in the CAM field is [2] that presents a method to reduce gaps that are present when simply offsetting a polygon with spirals. The presented method could be a possible future improvement to the spiral fill algorithm of **??**.
The OpenCAM library has also been inspected...

3.3.2 Spiral Filling

Straight Skeleton

To generate a spiral fill for the trajectory, we use the properties of the so called straight skeleton. The straight skeleton is defined as the topolgical skeleton of a polygon that is created by moving the edges inwards at a constant speed and observing the intersections of the vertices. It was first described by [3].
During the creation of the straight skeleton, two events can happen:

- Edge event: An edge shrinks to zero, making its neighboring edges adjacent now.
- Split event: An edge is split, i.e., a reflex vertex runs into this edge, thus splitting the whole polygon. New adjacencies occur between the split edge and each of the two edges incident to the reflex vertex.

(cited from [3]).

3.3.3 Back and Forth Filling

3.4 Path Generation

3.4.1 Traveling Salesman Problem

3.4.2 Adaptation of Traveling Salesman Problem for the Algorithm

3.5 Smooth line connections

Bezir Splines

Spiro Splines

Chapter 4

Implementation

4.1 Input

4.2 SVG Parser

4.3 Tree Container

4.4 Preprocessing

4.5 Implementation of the Algorithms

4.6 Postprocessing

4.7 User Interface

Chapter 5

Conclusion

Appendix A

Appendix

Bibliography

[1] T. Oksanen and A. Visala, "Path planning algorithms for agricultural machines," 2007.

[2] J.-H. Kao and F. B. Prinz, "Optimal motion planning for deposition in layered manufacturing," in *Proceedings of DETC*, vol. 98, pp. 13–16, 1998.

[3] O. Aichholzer, F. Aurenhammer, D. Alberts, and B. Grtner, "A novel type of skeleton for polygons," vol. 1, pp. 752–761, dec 1995.