KUNGL
TEKNISKA
HÖGSKOLAN

Royal Institute of Technology
Dept. of Numerical Analysis and Computer Science

Automatic Map-making of Adventure-pools

by
Erik Magnus Andersson

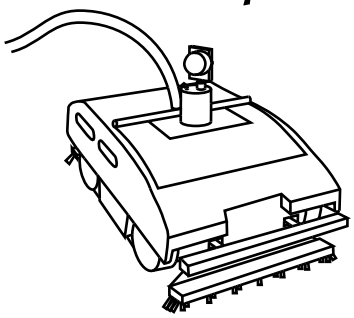
TRITA-NA-E0068

NADA

Nada (Numerisk analys och datalogi)
KTH
100 44 Stockholm

Deptartment of Numerical Analysis
and Computer Science
Royal Institute of Technology
SE-100 44 Stockholm, SWEDEN

Automatic Map-making of Adventure-pools



by
Erik Magnus Andersson

TRITA-NA-E0068

Master's Thesis in Computer Science (20 credits)
at the School of Vehicle Engineering,
Royal Institute of Technology year 2000
Supervisor at Nada was Henrik I Christensen
Examiner was Stefan Carlsson

Abstract

This project is about automatic mapping of irregular so-called adventure type pools. It is a part of a larger project driven by the assigner, Weda, to develop a pool-cleaner robot for these kinds of pools. The robot will be developed from their current line of pool-cleaners made for rectangular pools. Especially the robot will be adapted to the B400 model which is the most widely sold pool-cleaner. The assigners project has been tending towards using map based positioning as a navigation method of the pool-cleaner.

The mapping algorithm first uses a sonar sensor to make local map patches around the pool-cleaner, then it uses a matching algorithm for matching map patches to each other. As the pool-cleaner is exploring the pool, local maps are obtained and then matched and merged to the current map so that eventually a global map is obtained. The exploring of the pool, i.e.the process of obtaining where the next scanning pose shall be, is done manually.

The sonar transducer is mounted on a stepper motor to be able to rotate one revolution and thereby make a polar local map-patch. The sonar is a standard recreational boat sonar, which has been modified by the manufacturer to fit this project. The mapping algorithm has been tested with good results in the pool at Weda and in a larger adventure pool in Fyrishov, Uppsala.

Automatisk kartritning av äventyrsbad

Sammanfatting

Detta projekt handlar om automatisk kartritning av oregelbundna s.k. äventyrsbad. Det är en del i ett större projekt drivet av uppdragsgivaren, Weda, för att utveckla en poolrenarrobot för dessa typer av pooler. Roboten ska utvecklas från de befintliga poolrenarna gjorda för rektangulära pooler. I synnerhet ska systemet anpassas till modell B400 som är den mest sålda poolrenaren. Uppdragsgivarens projekt har lutat åt att använda kartbaserad navigering till poolrenaren.

Kartritningsalgoritmen använder en sonar som sensor för att skapa lokala kartlappar runt poolrenaren, sen används en kartmatchningsalgoritm för att matcha samman olika kartlappar. När poolrenaren utforskar poolen skapas lokala kartor som matchas och slås ihop med den befintliga kartan. När hela poolen är utforskad har således en global karta skapats. Utforskningen, bestämningen av var nästa scanningspose ska vara, sker manuellt.

Sonarens givare är monterad på en stegmotor så att den kan roteras ett varv runt poolrenaren och därmed skapa en lokal kartlapp. Sonaren är en standardsonar för fritidsbåtar, den har modifierats av tillverkaren för att passa detta projekt. Kartritningsalgoritmen har testats med gott resultat i poolen vid Weda och i Fyrishovs äventyrsbad i Uppsala.

Acknowledgements

I'd like to thank Henrik I Christensen, professor at Nada/Cvap department, for his guidance and help. Klas Lange at Weda for making this project possible, and placing pool cleaners and pools to my disposal. Anders Ekenbäck at Weda, for helping me with mechanical matters of the pool cleaner. Dalibor Sedlak at Weda, Alexander Tollet my roommate at Cvap, and Simon Andersson my brother for helping me with electronic and computer technical matters. Roger Phillips and David Annet at Talon Technology for their sonar. The personnel of Fyrishov for letting me use their pool.

Contents

Chapter 1

Introduction

1.1 Pool-cleaning

The keep the pool clean and neat is important for health reasons and for visual reasons. There are often governmental regulations which the baths must live up to, but also in order to attract guests the baths must show clean and hygienic pools.

There is a great variation of guests coming to public baths, which all have a great variation of hygienic ideas. Most baths ask their guests to use the showers before they enter the pool, but in any case there is a lot of dirt coming from the bathers. There is also dust in the air that will end up in the water in the same way as it ends up on the floor in an ordinary room. In outdoor pools the conditions are even worse, there are leaves and dust that are blown into the pool by the wind. From the bathers there is grass, sand and mud that is stuck to the feet that enter the pool. Bacteria growth is another major problem.

For the dirt that is dissolved in the water or not heavy enough to sink to the bottom, there is a usually a water circulation system that filters the water and cleans it with appropriate equipment, but heavier dirt that sinks to the bottom and dirt that is stuck to the pool must be taken care of in a different manner. A pool cleaner scrubs and vacuums the pool bottom to remove the dirt. There are manual pool cleaners that are placed on a rod and that are moved over the surface like an ordinary vacuum cleaner and then there are machines that more or less automatically move around the bottom.

1.2 Autonomous Pool-cleaning

1.2.1 Rectangular Pools

If the pool is of rectangular shape it is quite intuitive to use a machine that moves along one side up and down a number of times and in some manner moves the machine sideways so that the machine covers all of the bottom.

There are many machines of this kind on the market. Weda's machine's strategy is to make a small direction change every time it touches a turning wall, figure 1.1. Pool-cleaners from other manufacturers, for example the Piraya, have a strategy where the machine instead moves one machine width sideways so that

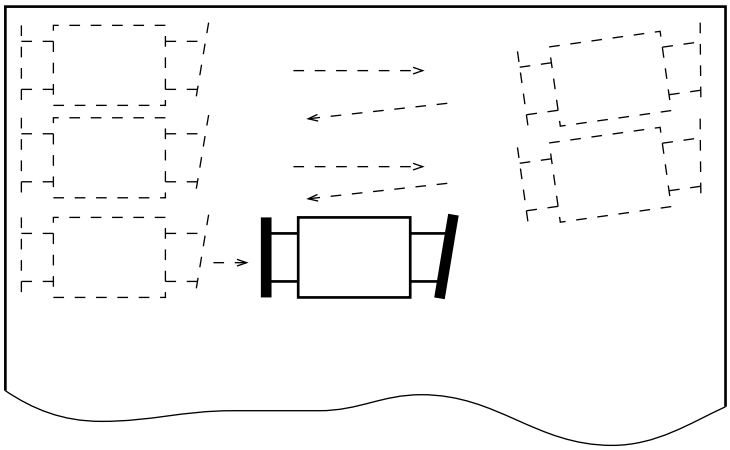


Figure 1.1. Poolcleaning strategy for rectangular pools.

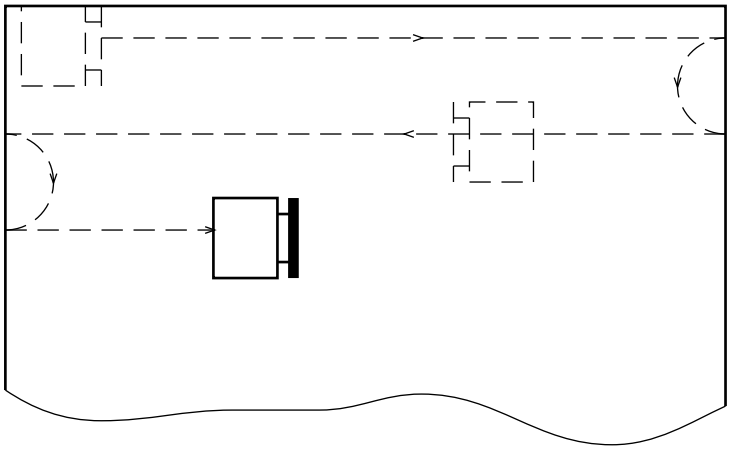


Figure 1.2. The motion pattern used by the Piraya cleaner.

the machine can start on a fresh path as shown in figure 1.2. The method used by Weda is slower and covers certain areas twice, but there are certain advantages. The method used by the Weda machine can use tracks since it does no sharp turns. The Piraya which is the machine representing the other strategy needs to be good at making a sharp turn which requires it to have wheels instead. The tracked propulsion is very good when it comes to pulling the hydrodynamically braked equipment through the water. It goes straight even if there are small bumps on the bottom that otherwise might result in slippage, and in addition there is always a significant pull from the floating cable that could easily make the machine move off its course when the grip is lost for a moment. The tracked vehicle can also move faster since it can handle the higher forces of the increased breaking forces from the water. It is not only a question of the hydrodynamic forces resulting from the motion of the machine, it is also the water motion from the water circulation system that both pulls the machine itself and the floating

cable. Entanglements of the floating cable that inevitably occurs further brakes the machine when it must be pulled through the water.

1.2.2 Irregular Pools

For irregular pools it is not obvious that there is a universal strategy for cleaning. But since there are many pool-cleaners on the market for rectangular pools one often, at least in Fyrishov (chapter 7.3), uses such a machine in the irregular pool and lets it work over the night. The machine will of course go completely unpredictably in the pool, but hopefully its random motion will cover the pool bottom without leaving any spots uncovered. What happens is that the machine often gets stuck with the cable entangled around an island for instance. The most remote parts of the pool, areas where there is a narrow channel leading to, are seldom cleaned. To ensure that the entire pool is cleaned one then has to manually steer the pool-cleaner, a task that takes a very long time and is very boring. For a large pool like the adventure-bath in Fyrishov, it can take five hours if the whole pool is to be covered.

1.2.3 Pool-cleaners at Weda

Considering the Pool-cleaners from Weda there are some interesting features which play a role when one considers how these machines should be developed into a robot. The B400 is the machine that is most widely sold from Weda, and the one that is most interesting to develop into a robot. Then there is the B600, a larger model, and the W70, a newly developed model for use in larger pools.

The B400 and B600 have asynchronous motors while the W70 has DC-motors. The DC motor can be stopped and started slowly while the asynchronous motor starts and stops abruptly. This means that the asynchronous motor cannot be as softly maneuvered, and it slips a bit when started and stopped.

The B400 uses one motor for both propulsion and vacuuming, the B600 has one motor for propulsion and one motor for the vacuuming, and the W70 has one motor for each track and also two motors for the brushing and a fifth motor for the vacuuming.

Each track of the B400 and B600 is connected to a brake that constantly drags the track. By disengaging the propulsion of the track with an electromagnetic clutch the track stops and the machine is made to turn. The corresponding track starts and stops abruptly with tracks sliding for short moments. Short moments of increased friction to the bottom can also make the dragging of the track come loose and the disengaged track starts to rotate for short moments. On the W70 on the other hand a track is softly slowed down and the machine turns in a more controlled way.

1.2.4 Earlier work

A number of other M. Sc. projects have considered control of pool-cleaners. These have all considered the problem of autonomous cleaning of rectangular pools (figure 1.3), [Zunino], [Macmillan] and [Ghanbarzadegan].

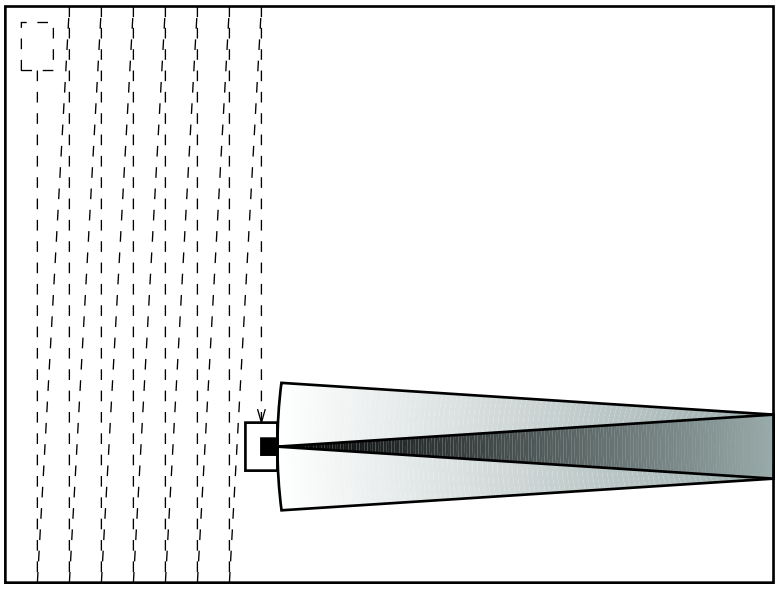


Figure 1.3. Pool cleaner with sonar for rectangular pools.

1.2.5 Problems concerning Autonomous Pool-cleaning.

Material The material pools are made of differs a lot across different parts of the world. The most common in Sweden is clinker, which are ceramic bricks with a rugged surface. But also quite common is the glazed tile [Swedish: kakel], this is the kind that is used in the testing pool at Weda. But in other parts of the world other materials are more common, In the USA there can be concrete in the pools, there can also be rubber carpets covering a dug hole. In Japan there are pools made of stainless steel.

We can surely realize the difference between stainless steel, concrete or rubber when it comes to how the tracked cleaner turns and slips when a track is stopped or braked. The B400 for instance slides a few centimeters, and changes direction slightly when it starts or stops in the Weda test pool.

Common Peculiarities A molding is often placed around the pool so that swimmers can rest even if they are in the deep half of the pool, see figure 1.4. In the USA the edge between bottom and wall is often rounded, which is important to know if you're going to let the cleaner touch the walls, see figure 1.5. In Fyrishov there is in both their indoor and outdoor pool a seat along the side in certain parts of the pool, figure 1.6. In the indoor pool this seat is equipped with whirlpool facilities. In the outdoor pool the seat is more for sitting in the sun. These seats must be taken into consideration if a sonar is to be used to measure distances to walls. In Fyrishov there are also in some walls, stone formations where stones are cast into the structure, figure 1.7. These stone formations pose a challenge as it is difficult to model these objects. At the same time the formations provide an excellent sonar reflector for localization. Below one of the stairs in Fyrishov there is a kind of plateau, figure 1.8.

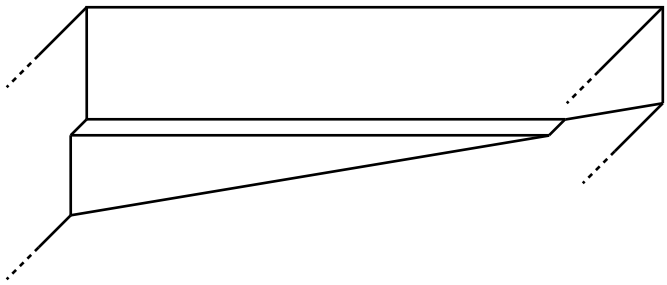


Figure 1.4. Moulding.

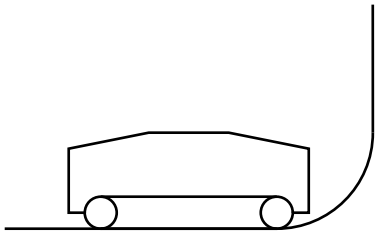


Figure 1.5. Rounded transition from floor to wall.

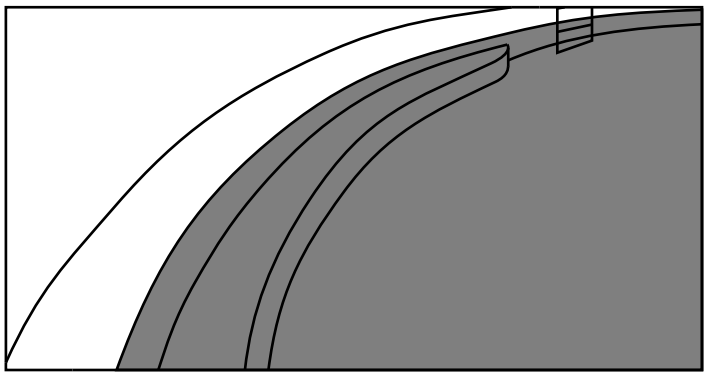


Figure 1.6. Whirlpool bench.

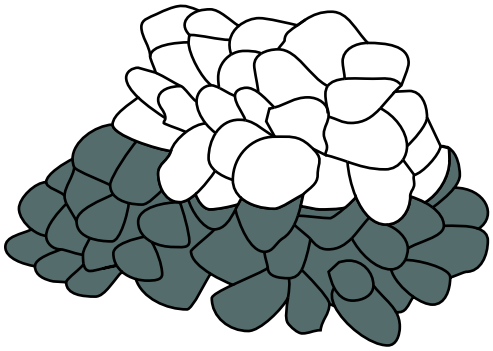


Figure 1.7. Rock formations.

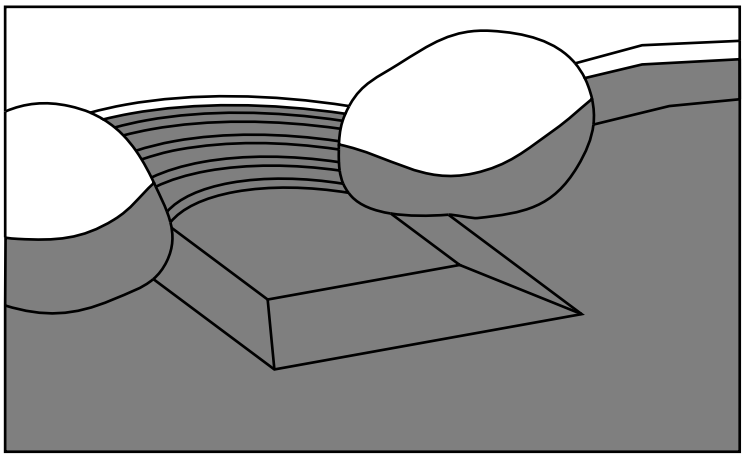


Figure 1.8. Bottom Plateau.

1.3 Map-based Positioning

In map-based positioning one uses a local map which is compared to a global map to obtain the position and orientation of the robot. The local map is obtained by the sensors of the robot, and a map matching algorithm must be used to match the maps. The global map can either be constructed by the robot itself by matching the local maps as the robot is exploring the pool or it can be a preexisting map that is made from drawings of the pool [Borenstein].

If one is going to use a preexisting map one still has to develop a local mapping method using sensors and a map matching algorithm to go with it. So automatic map-making includes major parts that are necessary in map-based positioning.

Sonar is used as a sensor for making the map because it is known to be the best sensor in water. Other sensors for map-making like laser or other electromagnetic waves generally decrease their performance in water while the ultrasound of the sonar is much better propagated in water than in air. It is no coincidence that submarines almost exclusively use sonar to sense the surroundings, also dolphins and whales use ultrasound for their long range sensing of their surroundings.

To prepare for this project the following books and reports have been read;

- J. Borenstein, H.R. Everett and L. Feng, 1995, Navigating Mobile Robots, Sensors and Techniques.
- H.R. Everett, 1994, Sensors for Mobile Robots.
- G. Zunino and M. Simoncelli, 1999, Autonomous pool cleaning.
- A. Macmillan and K. A. Norlin, 1998, Teknik under ytan, automatisering av bassängrenare.

Part I

Theory

Chapter 2

Sound and Waves

The choice of a sonar depends on a number of things. This chapter will present some theory and heuristics to help. In the end it is often the price tag that has one of the highest weighting factors when it comes to choosing the wants to look at whatever is mass produced. Unfortunately these products have a construction that is optimized for another purpose, and it is usually not at all like the theoretical product according to your calculations.

The theoretical model will however still provide valuable input to the selection of a suitable sensor.

Here the damping of the sound, the reflection, and the beam width of the sonar are considered.

2.1 Propagation of Wavefronts

When one is going to predict how a wavefront propagates it is useful to consider Huygens' principle. It explains that every point on a wave front acts as a source of secondary wavelets, so called elementary waves. At a later time, the envelope of the leading edges of the wavelets forms the new wavefront [Bodn]. It can among other things be used to show that free plane waves remain plane and spherical waves spherical during propagation, see figure 2.1.

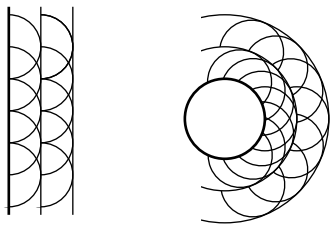


Figure 2.1. Huygens principle: The first picture shows how the envelope of the elementary waves forms plane wave fronts and the second picture the corresponding case for spherical waves.

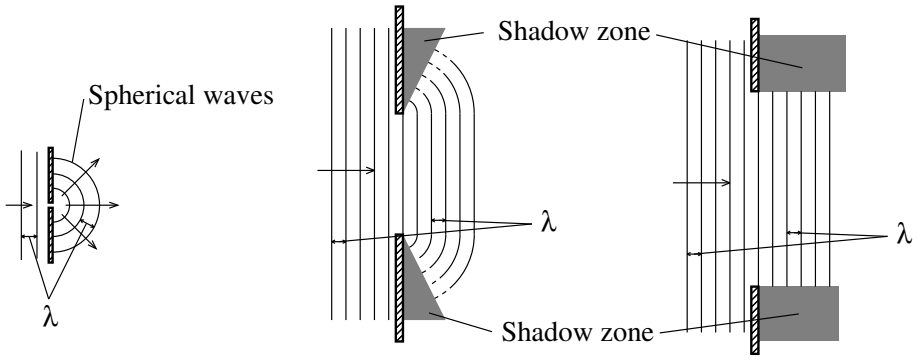


Figure 2.2. For a small hole according to the wavelength the waves will pass to form spherical waves, if the hole increases relative to the wavelength the passed waves will more and more form the shape of the hole.

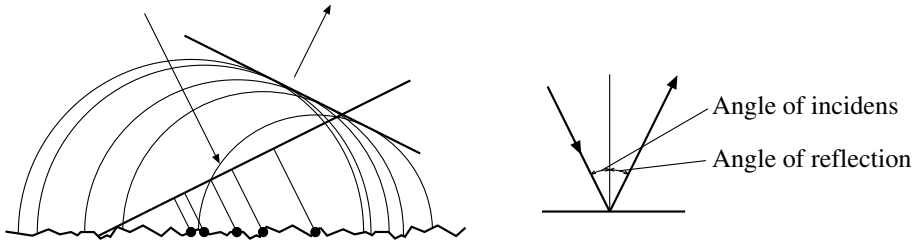


Figure 2.3. With help of Huygens principle, one can show that angle of incidence equals angle of reflection.

2.1.1 Beam Width

Another phenomenon that can be explained with Huygens principle is diffraction, which is waves bending when they pass a hole.

If the frequency is low and the wave length is long compared to the hole size the wave after the passage will bend and be spherical, figure 2.2 a), and if the frequency is high the waves will continue straight forward, figure 2.2 b) and c). This implies that is not possible to form a narrow beam from a small transducer if the frequency is not high enough.

2.1.2 Reflection

When the incoming wave reflects off a surface one can draw elementary waves from the surface and see how the envelope of the wavefronts form the reflected wave. With simple geometry one can see that the reflected wave is symmetric around the normal of the surface, that is the angle of incidence is equal to the angle of reflection, as known from ray theory, figure 2.3 [Benson]. On the other hand when the surface roughness gets large enough relative to the wavelength, the difference in traveled path compared to the wave length will increase and the reflected beam will be canceled out. Instead every little roughness of the surface will reflect the beam according to ray theory, and the result will be that

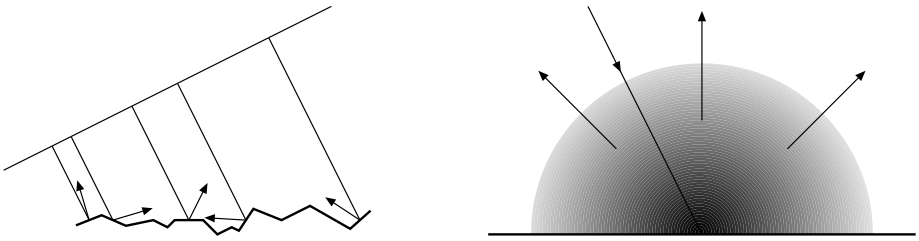


Figure 2.4. Diffuse reflection.

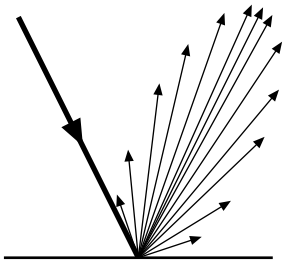


Figure 2.5. Combination of diffuse and specular reflection.

the reflection is equally strong in all directions, figure 2.4. For planar surfaces the reflection is said to be specular and for rough surfaces the reflection is said to be diffuse.

For most practical materials the reflection is a combination of specular and diffuse reflections. To get some insight into this phenomenon one can consider how light behaves instead and think of different materials, for example a mirror, a piece of paper, and a kitchen sink, made of stainless steel. How would these reflect light like for example a spotlight. Imagine a dark room with walls covered with these materials, and having to orient oneself with a spotlight. The mirror would of course reflect in a specular manner and it would be impossible to see a reflection unless the beam is aimed perpendicular to the wall. The white paper would reflect in a diffuse manner, and one would get a reflection, seeing were the beam hits the wall independent of beam direction. The stainless steel would represent the intermediate, where the reflection is very strong when one aims the beam right to a perpendicular wall, but when the angle increases there would be decreasing reflections from small scratches in the metal and one would still see the wall even if it would be a weak reflection.

2.2 Damping

Sound can be composed of both longitudinal waves and transverse waves. Transverse waves require the medium to be able to hold shearing forces, and since fluids can't do that they will only be able to transmit longitudinal waves. This means that the energy of the sound wave in fluids consists of pressure oscillations only. These pressure changes are what in thermodynamics are called state changes, and they will be done differently depending on if there is heat flowing

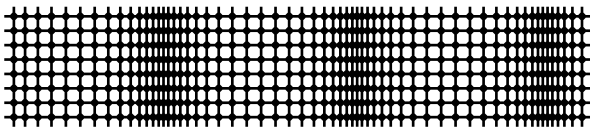


Figure 2.6. Sound in fluids are oscillation of pressure.

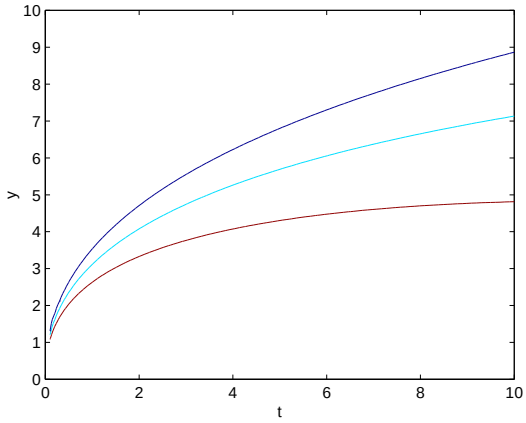


Figure 2.7. Spreading of heat, for some constant temperatures. Obtained from the heat equation with a Dirac function used as initial value. The solution is $u(x,t) = \frac{1}{\sqrt{4\pi\kappa t}} e^{-\frac{(x-x_0)^2}{4\kappa t}}$, which here is plotted for some different temperatures u.

between the state changes or not. For low frequencies there is a long distance between compressed and expanded areas so it will take a long time for the heat to flow but analogously there is also a longer time available which allows for more heat to flow, and the opposite occurs for high frequencies where there is short time for the heat to flow, but on the other hand having a shorter distance to flow. The heat equation [Petersson], gives some insight.

$$\frac{\partial^2 u}{\partial^2 x} = \frac{1}{\kappa}\frac{\partial u}{\partial t} - \frac{q}{\lambda}$$

where u is temperature, x is the spatial coordinate, t is time, κ is thermal diffusivity, q is source strength, and λ is heat conductivity. If one solves the heat equation for a Dirac function as initial condition one sees that heat flow speed is not constant as in the case of sound propagation. But instead for short distances the heat flow speed is closing to infinity and for long distances it is closing to zero, see figure 2.7. This means that for long distances the heat flows slow and consequently on short distance the heat flows fast. This will make two pure cases of how sound propagates through the medium, either adiabatically where no heat is exchanged or isothermally where the heat conduction is total [Bodn]. Adiabatic sound is found for low frequencies and isothermal is found for high frequencies. At intermediate frequencies the state changes are something in between adiabatic and isothermal, especially a compression would not follow the same path as a rarefaction. As long as the state changes follow the same

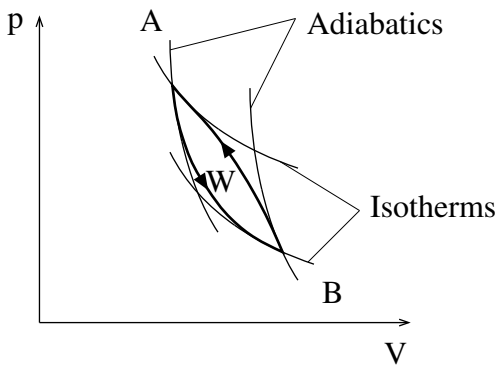


Figure 2.8. For adiabatic sound the state changes follows the adiabatics and for isothermal sound the state changes follows the isothermals. For frequencies where the state changes are between the adiabatic case and the isothermal the state changes will initially be isothermal and then turn to be adiabatic, so that the compression and expansions don't follow the same path and there is an energy loss in every cycle [Bodn] and [Adkins].

path there is no energy loss, but when the changes follow a circular path there will be energy lost in every cycle, according to

$$W = \oint V dp$$

Where W is work, V is volume and p is pressure, [Adkins].

For low frequencies the reversible adiabatic state changes make the sound travel very far. When the frequency is increased there is energy lost and the maximum range is quickly reduced. When the frequency is increased so much that the sound can be counted to be isothermal, then theoretically the energy losses per cycle is reduced, but since the number of cycles also increase, the energy loss per cycle will quickly dampen the signal even if there is only a very small loss per cycle.

The table 2.1 is from Kongsberg Simrad AS [Simrad EA 500], and shows for some different frequencies what the maximum distance would be, when using a sonar for measuring the depth from a boat. This table should of course be read with caution when we try to choose a frequency for the pool cleaner project. Sea bottom can probably be considered to reflect sound worse than hard pool wall. A boat also moves at a much higher speed, where turbulence and possibly air bubbles can reduce the transducer performance. But nevertheless the figure clearly shows how high frequency waves are damped quickly, and the maximum distance of the transducer is decreased. The result of considering the damping is that there is an upper limit on how high frequency one can use.

2.3 Summary

The wave propagation is better adapted for use as a map making sensor when the frequency is increased, since the beam is narrower and the reflections are

Transducer	Type	Power	Max. Depth [m]
12 kHz	12-16	2 kW	13000
18 kHz	11-18	2 kW	8100
27 kHz	38-7	2 kW	4500
38 kHz	49-26	2 kW	3200
49 kHz	27-26/21	2 kW	1800
120 kHz	120-25	1 kW	630
200 kHz	200-28	1 kW	500
710 kHz	710-36	50 W	70

Table 2.1. Max distance of a transducer depends of the frequency, from a product specification from Kongsberg Simrad [Simrad EA 500].

more independent of the incidence angle to the wall. But since the damping is increased with frequency there is a limit for how high a frequency one can use.

Table 2.1 from Kongsberg Simrad shows that the damping quickly reduces the max distance when the frequency is increased, but one would be able to use a higher frequency than recreation boat sonars that usually use 200 kHz [Ghanbarzadegan]. The sonar producer Benthos Inc. suggested 500 kHz to 1MHz as best suitable for measuring walls in a pool environment.

Chapter 3

Signal Analysis

The sonar manufacturer Talon technology made some modifications to one of their sonar models, to fit this project. This chapter describes how the signal that is supplied by that sonar is handled. The modifications are made to a D41 sonar instrument, which is described further in chapter 5.2.

3.1 Measuring Distances with Sonar

The first part of the signal analysis is made by the sonar instrument. The vibrations in the water are transformed into an electric signal by the transducer, and then the envelope of the signal is supplied in dB i.e. the signal is supplied with logarithmic output.

The sonar emits a short sound, a chirp, which spreads in a cone from the transducer. Then objects in this sound cone will reflect it and by measuring the time it took for the sound to come back one can calculate the distance, as seen in figure 3.3. The front flank of the chirp, and specifically the inflection point on the flank, is used to measure from. To detect this flank the derivative and the second derivative is calculated. The inflection is found where the second derivative goes from positive to negative, and to make sure that it is not a disturbance the derivative must be above a certain threshold, compare to figure 3.4. The threshold varies with time as explained later. One might consider using the back flank of the chirp, but it is not as sharp as the front flank since the transducer rings, like a bell, when the sonar goes silent. If the echo returning is very weak and the square shape is lost, one could instead look for where the sound is above a certain level, but this would not have the same precision as looking for the flanks.

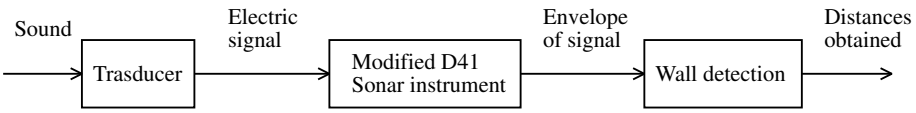


Figure 3.1. Block scheme.

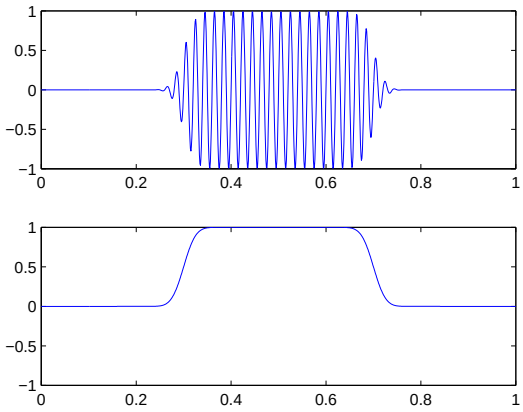


Figure 3.2. The modified D41 sonar instrument supplies the envelope of the signal.

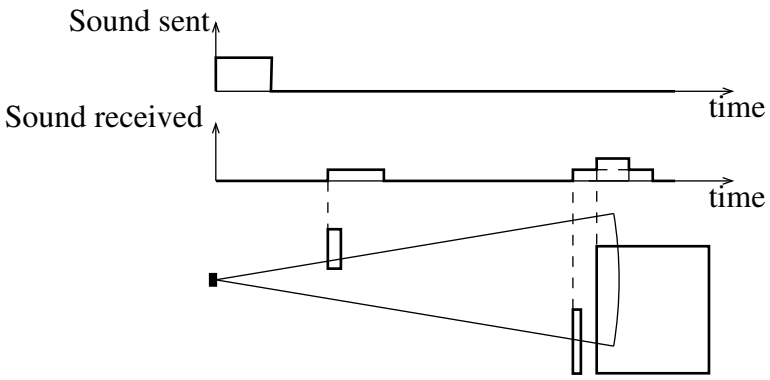


Figure 3.3. Objects in the sound cone.

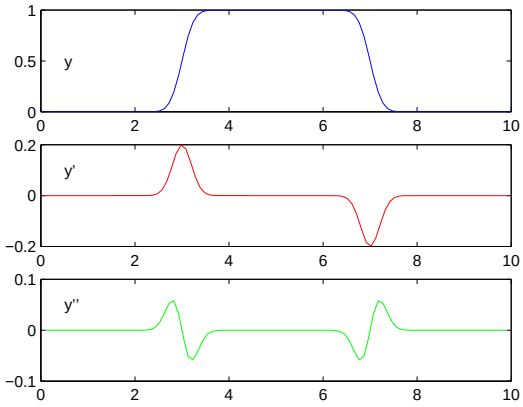


Figure 3.4. Echo of chirp, its derivative and second derivative.

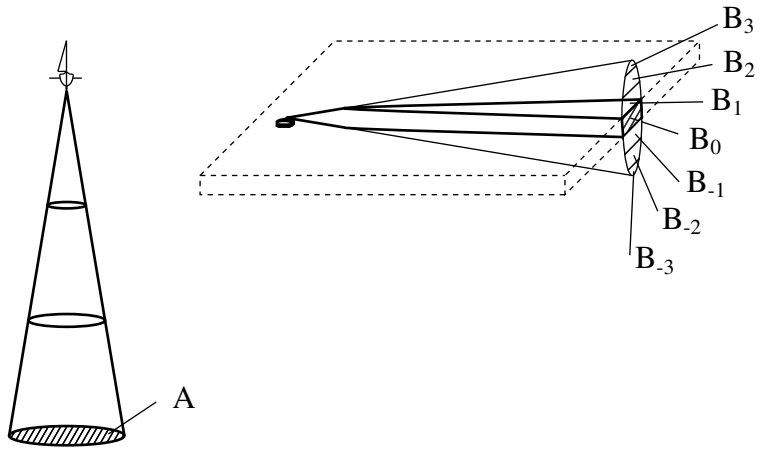


Figure 3.5. When the sonar is used in a pool there are multiple mirrors of the wall B_1, B_2 ... and B_{-1}, B_{-2} ... from where the echo is weakened due to the multiple reflection to surface and bottom.

3.2 Threshold

The sound is damped with increasing distance as described earlier. The sonar instrument amplifies the signal on the assumption that the sonar is used in free space water as when mounted on a boat (figure 3.5). But when the sonar is used in a pool the sound does not spread in free space water but instead is reflected against the surface and the bottom. The wall ahead would have multiple mirrors above and below it. The bottom and surface are however not perfect mirrors but absorb a certain amount of the wave energy, so a wall in a pool decreases the echo strength quicker with respect to the distance than it would in free space.

Another reason for the threshold to vary over time is that the sound card used to sample the the signal, is not a perfect AD converter but instead is made for sounds that are audible for a human. This specifically means that the sound card neglects low frequencies and high pass filters the signal. It is not very serious since it is the derivative and second derivative that are used to detect the distances.

The threshold has been determined empirically and is shown in figure 3.7.

3.3 Filtering

To calculate the derivative and second derivative, difference equations are used which take into account the neighbouring points. These equations are very sensitive to noise and the signal also has to be smoothened. The smoothening is achieved by a weighted average of the closest surrounding points. The weights correspond to a discretized gauss curve, and are obtained by choosing a line of Pascal's triangle, also called binomial coefficients.

$$a_k = \left(\begin{array}{c} r \\ k \end{array} \right), \; k = 1 \ldots r$$

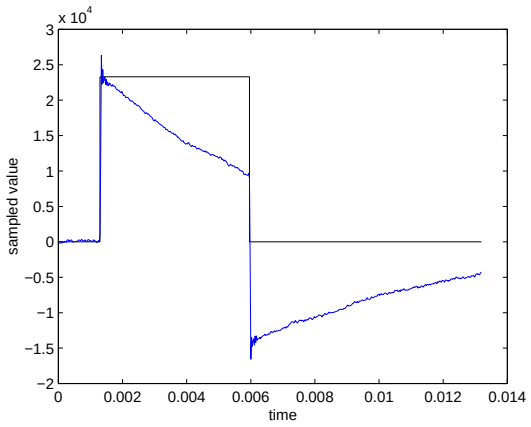


Figure 3.6. A square pulse is sampled to show the sound cards high pass qualities.

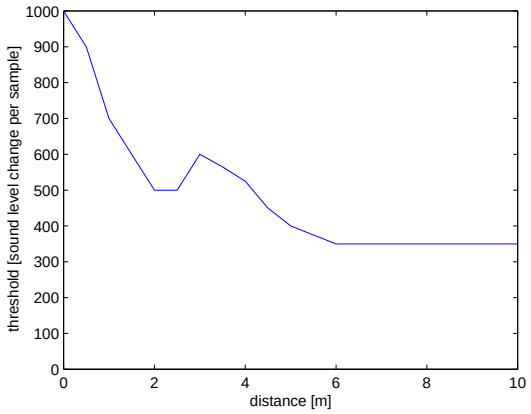


Figure 3.7. The threshold is a function of time, or for better intuition a function of distance.

Where r is an odd number describing the width of the window used for calculating the new value u_n. These weights are normalized so that their sum equals one, and will thereby correspond to a discretized gauss curve.

$$a'_k = \frac{a_k}{\sum_{i=1}^{r} a_i}$$

The new filtered value is then calculated with the following function.

$$u_n = \sum_{i=1}^{r} a'_i y_{n-(\frac{r-1}{2})+i}$$

3.4 Summary

The sonar instrument supplies the envelope as a logarithmic output. The signal is also dynamically adjusted through the receive cycle to increase the gain with time after the start of the receive cycle. This is to compensate for weakening echo due to damping in the water. The flanks of the returning pulse are detected by searching for where the derivative is above a certain threshold, and at the same time the second derivative goes from positive to negative. The threshold varies over time and has been determined empirically.

Chapter 4

Map-making

4.1 Robust Curve-fitting

With a set of data one often wants to summarize it by fitting it to a model that depends on adjustable parameters.

$$y(x) = y(x; a_1 ... a_M)$$

To do this one generally uses an idea where one has a figure-of-merit function that measures the agreement of the data to the model. This function is then optimized and the solution is taken to be the best curve. The merit function is conventionally arranged so that a small value corresponds to a close fit.

$$\min f(a_1 ... a_M, y)$$

Through the decades scientists have been in love with the normal distribution and the assumption that errors are normally distributed. But for our purpose, that of matching a curve to a sonar scan, it is obvious that the measurement error is not normally distributed, since there are outliers that are far away from the wall that we are measuring. If one assumes that the error is normally distributed, and uses the least square method that is connected to it, then these outliers corrupt the whole curve and there will be a very bad match, see figure 4.1. The reason for the bad match is that the least square method uses the square distance from the point to the curve as a penalty in the figure of merit function. This means that the outliers get a very high weight.

M-estimates As a generalization of the least square method, one can use M-estimates which allow any distribution to be used. An M-estimate are the maximum likelihood estimate of the data with the assumed distribution.

The figure-of-merit function, here called ϵ is taken to be the logarithm of the likelihood function with a minus to follow the minimum convention

$$\epsilon = -\ln L(\theta) = -\sum_{i=1}^{n} \ln f(x_i - \theta) = \sum_{i=1}^{n} \rho(x_i - \theta)$$

where $\rho(x) = -\ln f(x)$ and $f(x - \theta)$ is the density function of the distribution.

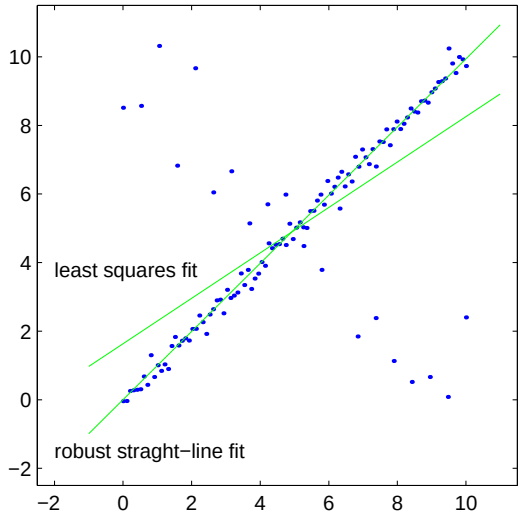


Figure 4.1. Outliers makes the least square method fail.

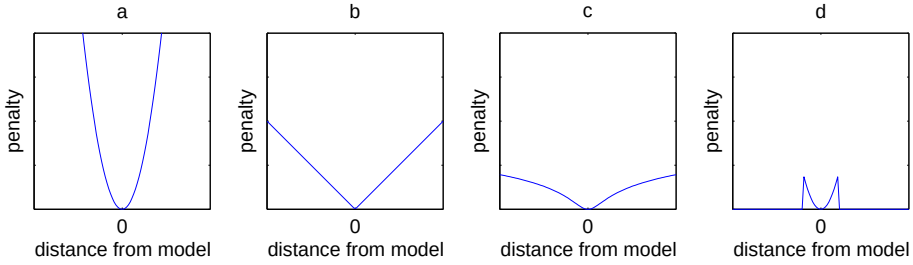


Figure 4.2. Examples of penalty functions $\rho(x)$. a) Least square deviation, b) mean absolute deviation, c) penalty for Lorentzian distribution, d) Penalty used when matching local map to global map.

If one assumes the normal distribution, then $\rho(x) = \frac{x^2}{2} + c$, and the figure-of-merit function would correspond to the least square method. But one can use other distributions that allow more outliers and thereby correspond to the actual data. For a double exponential distribution, $\rho(x) = |x| + c$, and the figure-of-merit function corresponds to the mean absolute deviation. Other distributions can also be used which give even lower penalty to the outliers[Launer], see figure 4.2.

4.2 Local Map

The local map is made by rotating the sonar 360 degrees around the pool, and there are one hundred measurements taken which make up a discrete map of the closest surroundings. If one uses higher resolution it takes a longer time to collect the data, and lower resolution gives poorer maps. A resolution of

two hundred measurements per revolution was initially used but it was skipped because of the long time it took to collect the data. All plots in this report use a resolution of one hundred measurements per revolution. The stepper motor and sonar unit is mounted on the in front of the floating cable of the pool cleaner. As shown later there is no disturbances from the cable.

As a consequence of using a recreational boat sonar, which uses 200 kHz instead of using a sonar adapted for pool walls which in should use a frequency of between 500 kHz and 1 MHz as argued earlier, there are multiple echoes as shown in figure 4.3 and 4.4. If there is a straight wall it will function as a plane mirror and the secondary echos will form a mirrored map on the other side of the wall, figure 4.3. This mirrored map will among other things explain, chapter 5.1, why we needed a modified sonar since the original sonar easily followed the mirrored wall instead. If there are close walls on both sides of the pool-cleaner there can be multiple mirrors. In the pool at Weda for example there were up to six multiple echos in the lengthwise direction of the pool, figure 4.4. In a narrow channel in Fyrishov there were even more.

This problem is not like outliers that can be handled with robust curve fitting methods, but instead must be handled a bit differently. The first echo is chosen and the other ones are neglected. A negative result of this is that outliers at shorter range than the wall will make any measurements from the true wall be neglected. One cannot use the strongest echo because the secondary echo sometimes is stronger than the first, as described in figure 4.5.

After the first echo has been chosen the remaining outliers are removed by curve fitting where a curve is fitted to a small number of points.

4.3 Map-matching

In map matching one has two maps that are going to be matched to each other so that their mutual position and orientation is determined. If one of the maps is a local map recently taken from the robot one also obtains the position and orientation of the robot. The matching is done by choosing a common feature of both maps, and making the assumption that they are from the same physical feature and laying the maps over each other according to the features. Then one calculates how good the match is by using a figure-of-merit function as in curve fitting where one map is a model and the other is the data. The global map is chosen to be the model and the local to be the data, and the data points distance to the closest model point are given a penalty according to figure 4.2d, that is data points where the distance to the model are over a threshold are not counted. Since the choice of features are not likely to be from the same physical feature, the procedure is repeated with randomly selected features and when enough calculations are made one should be able to choose a set of features that has low enough value of the figure-of-merit function and enough number of points within the threshold of the penalty function. This procedure is also called voting or RANSAC.

The features can be of different kinds, but here a very simple one is used which is the distance between two points. In other words random points from the local map p_{l_1} and $p_{l_2}$and from the global map p_{g_1} and p_{g_2} are chosen. If their mutual distance is approximately common, then one has a set of features, and the calculation of the figure-of-merit function can start.

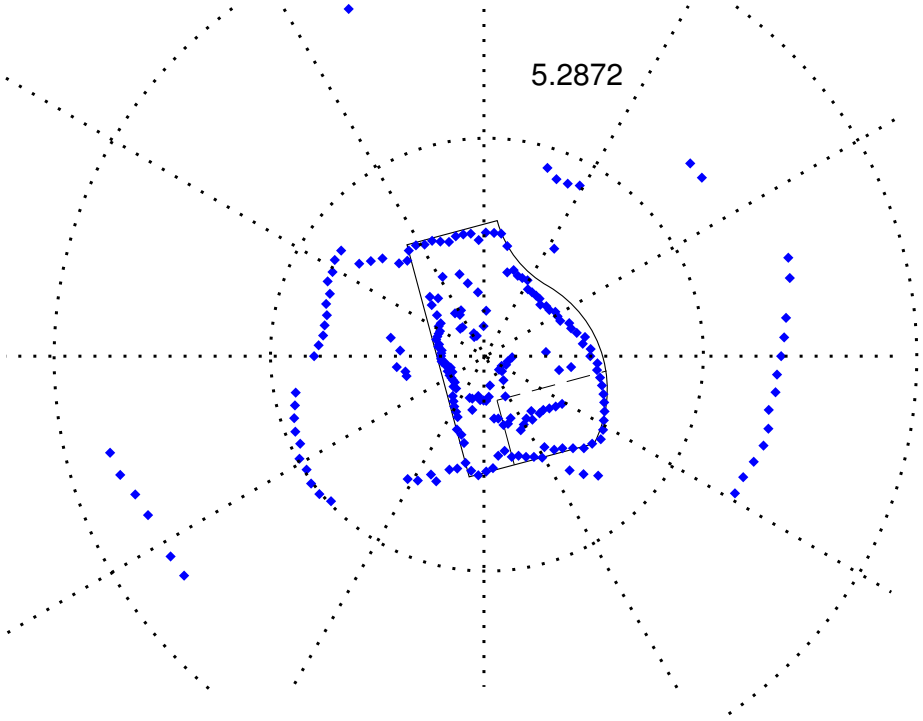


Figure 4.3. Mirrored echos and echos from pool floor with D41 in pool at Weda. A cad-map is included for comparison.

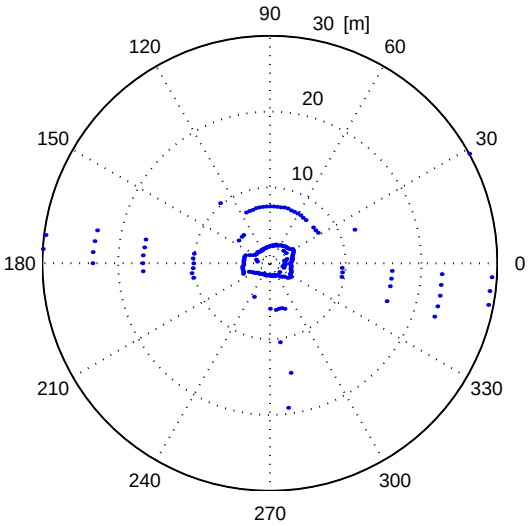


Figure 4.4. Multiple mirrored echos at Weda.

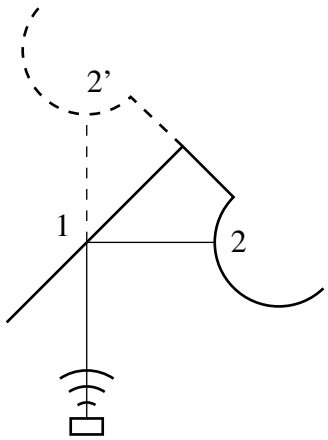


Figure 4.5. Weak echo from 1 due to high angle of incidence, but strong echo from the mirrored wall 2'. As a heuristic example, having figure 2.5 in mind, lets say that the direct echo from wall 1 is 20% of the incoming echo, and the reflection in the direction which corresponds to ray theory is 80% then the reflection losses from the combination sonar-1-2-1-sonar is $0.8 * 0.8 * 0.8 = 0.512 = 51\%$, compared to the direct echo of 20%. This shows why the secondary echo sometimes is stronger than the first echo.

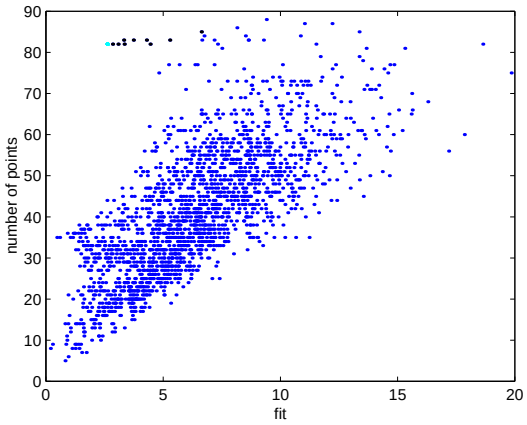


Figure 4.6. If one plots the matching attempts in a graph with the figure-of-merit value on the x-axis and the number of points within the threshold of the penalty function on the y-axis one one would find the best fits in the upper left corner.

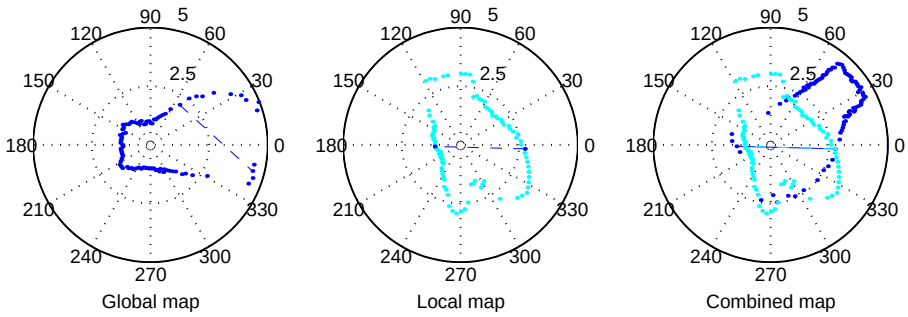


Figure 4.7. Two points from the reference map and two points from the new map, with the same mutual distance. The maps are put together according to the points.

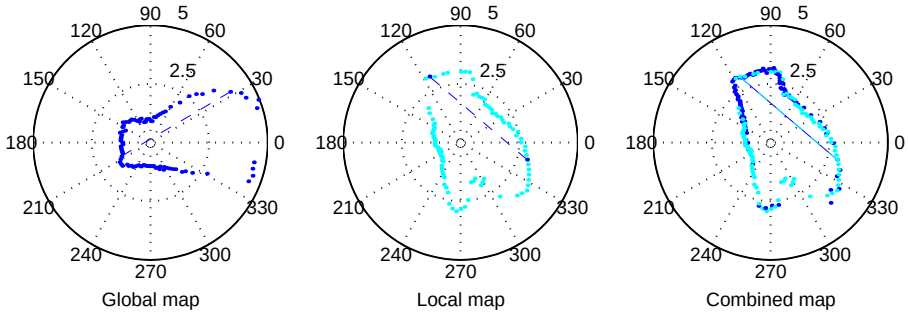


Figure 4.8. After a number of iterations one will be able to choose a good match.

When two thousand sets of features have been matched the procedure is stopped and a best match is selected. The best match is found in the upper left corner of figure 4.6. The actual calculation used is the argument that minimizes

$$\min_i \sqrt{(n_i - n_{tot})^2 + (c \cdot fit_i)^2}$$

where n_i and fit_i are the values from the matching attempts, n_{tot} is the total number of points in the local map and c is a weighting factor between the two axes, empirically $c = 3$ has been used.

The time for matching two local maps is approximately one second, then for every added local map the match time increases with approximately one second.

4.4 Summary

A local map is made by rotating the sonar 360 degrees and then by robust curve fitting outliers can be removed. Then the matching process can match two maps to each other and thereby get their mutual position and orientation to each other. Local maps are merged to build a global map.

Part II

Experimental work

Chapter 5

The Sonar

This project employs a sonar sensor from Talon Technology. It is a depth measurement sonar for recreational boats. Two different versions have been studied. The first one is a standard system which gives the depth, or in our case distance, both on a built-in display and on a NMEA 0183 serial bus. The second version is a modified sensor that delivers a raw sensor signal.

The sonars consists of two parts: the sonar instrument, which is the part where the interpretation takes place, and then there is the transducer that transforms electric signals to vibrations in the water and the reverse, picking up vibrations and transforming them into electric signals. The transducer is often a piezoelectric element. In this chapter the two sonar instruments used and the transducers are described.

5.1 Navman D100

Initially the D100 sonar instrument (figure 5.1) was tested with its built in echo interpretation, and using the NMEA serial line connected to the computer. The built-in interpretation has been made for usage in boats for depth measuring and there are cases when the built in algorithm gives severely incorrect results when it is used in the pool to measure the distance to the walls. The built-in interpretation system tries to find an echo at approximately the same distance

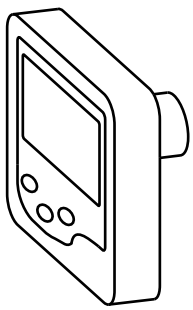


Figure 5.1. D100.

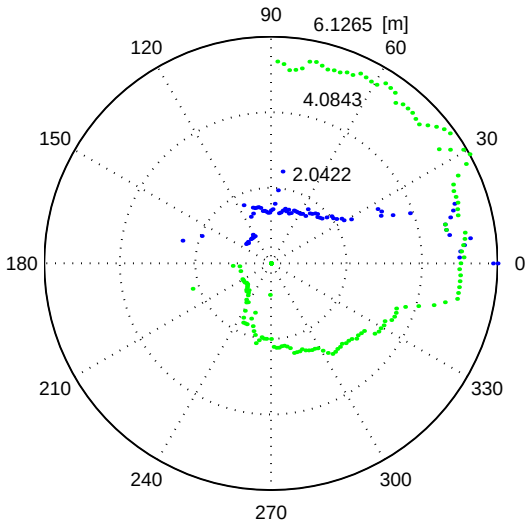


Figure 5.2. Original Navman D100 on a rotating step-motor in the pool at Weda. The transducer rotates 450 degrees, when the second lap starts the sonar follows the mirrored wall instead of the true wall. Compare to the raw scan of figure 5.7 and figure 4.3. Different colors are used from 180 degrees and upward to distinguish overlapping parts.

as the previous interpretation. It only follows one potential wall and neglects other potential walls that might give stronger echoes. The potential walls can be, besides the true wall, mirrored walls or reflections from the bricks in the bottom.

In the raw scan of figure 5.7 in the corner of the long side and the right short side one can see that as the sonar rotates leftward, the built-in interpretation algorithm suddenly gets two walls that it can follow, either the long wall or the mirrored wall of the short side. The interpretation algorithm does not necessarily follow the true wall as seen in figure 5.2. When the sonar passes the same corner on the second lap the mirrored wall is followed instead.

A similar error occurs when the sonar sweeps by an island, figure 5.3, where the interpretation algorithm starts following the bricks in the bottom instead of making a large leap to the much stronger echo of the much more distant true wall.

In bottom measurement conditions it is only the bottom that gives constant echos. All other disturbances are from fishes and floating objects and pass through quickly as the boat moves on. But in the pool there are many constant things especially the floor, and mirrored walls. The wall that we actually are looking for, might not be constant since there are islands etcetera that make sharp jumps as the transducer passes by.

5.2 Modified Navman D41

A modified D41 Navman depth sounder (figure 5.5) has been provided by Talon. The system provides the raw sound level expressed in dB, e.g. the logarithm of

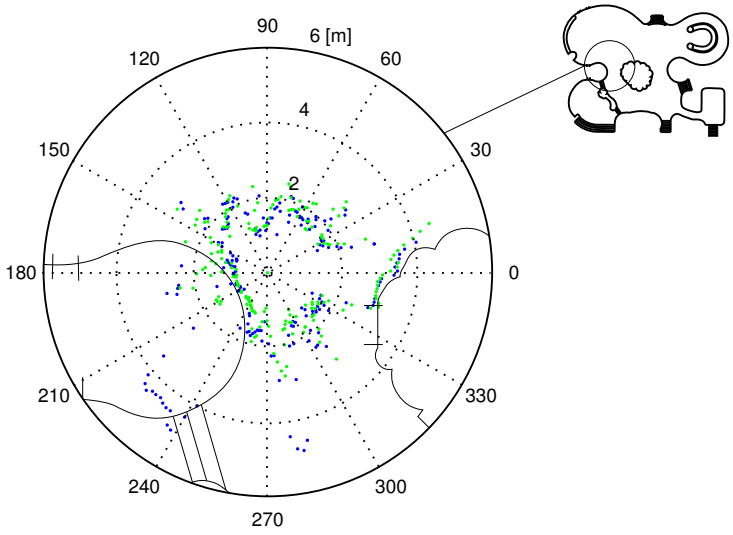


Figure 5.3. Original Navman D100 sonar in Fyrishov. The sonar rotates 720 degrees. When the sonar starts at 0 it gives the correct measurements, but once the transducer beam passes the island it has the option of following the true wall (which is very distant) or to follow the floor at a distance of approximately 2 meters. The closest walls are being recognized correctly but when the interpretation algorithm should make the jump to a distant wall it locks on the bottom instead.

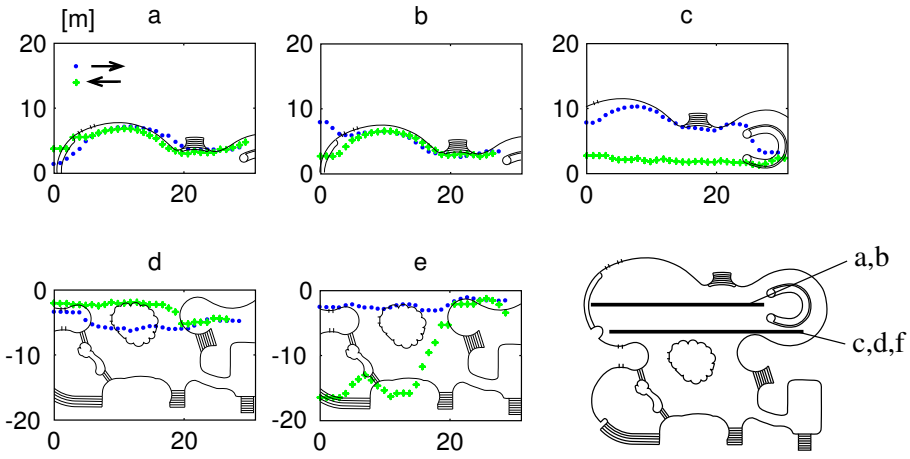


Figure 5.4. D100 in Fyrishov. From figure c one can again see that once the sonar has been on the horse-shoe-shaped island the interpretation algorithm locks on the floor instead of the true wall.

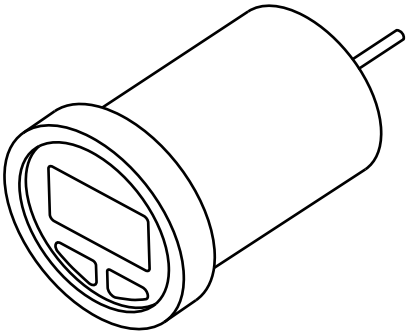


Figure 5.5. D41.

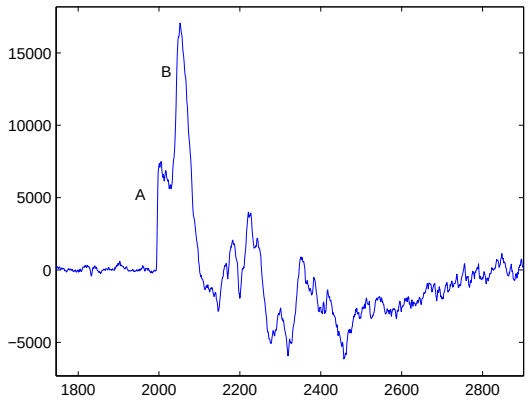


Figure 5.6. At the sharp uprising edge A, the sonar goes mute and the listening is turned on, the signal rises immediately, and the small noisy peaks come from the the floor bricks of the pool. At the sharp uprising edge B, is the actual wall. The next peaks that come from echoes bouncing around in the pool, not interesting for us.

the amplitude. The signal is also amplified with respect to time to compensate for damping for longer distances. The modified sonar instrument also has a trigger reference indicating the start of a receive cycle. Both the signals, the trigger signal and the signal strength, are supplied through two BNC connectors added to the rear of the instrument. The signal is then digitized by the sound card of the computer, figure 5.6. An example of the resulting scan made by the instrument is shown in figure 5.7, where the transducer has been rotated in the pool at Weda.

5.3 The Transducers

From Talon technology there are two kinds of transducers that have been tested. One is a wedge-shaped transducer that is supposed to be mounted in the stern of the boat so that its stream lined wedge goes below the hull in the free flowing water. This is the transducer that comes as standard when one orders the sonar,

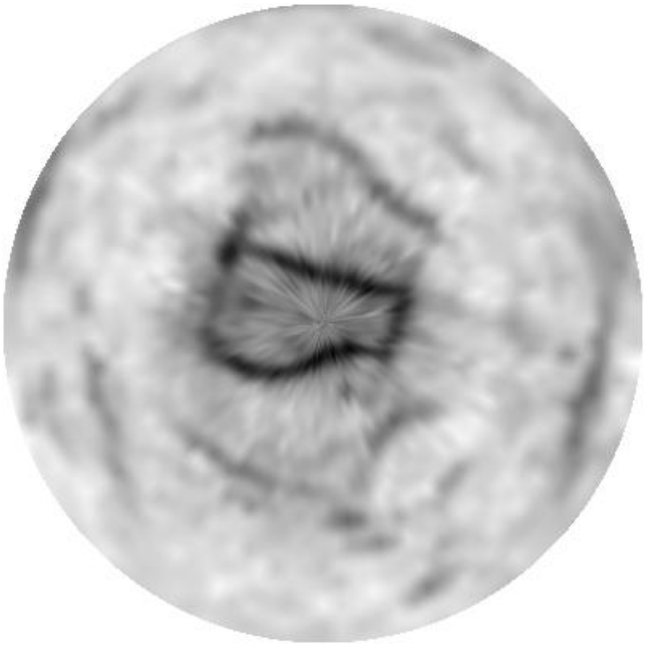


Figure 5.7. Raw scan of pool at Weda. A blacker color represents a sharper derivative of the sound level, of in other words the front flank of the echos.

see figure 5.8. Then there is a puck transducer that has one side sloped to fit the bottom of the boat. It is supposed to be mounted inside the hull with the sonar beam going through it. It is a little bit bigger and heavier and was originally ordered in one of the earlier projects, see figure 5.9. Both of them operate at a frequency of 200 kHz.

5.4 Summary

The sonars from Talon supply the depth in digital form using the serial NMEA-standard. The built-in procedure for calculating the depth is adapted for depth measurement from a boat, and does not work well in a pool.

With a small modification to the sonar, made by Talon, the raw sound level of the sonar working frequency is supplied. This is the sonar that has been used in this project.

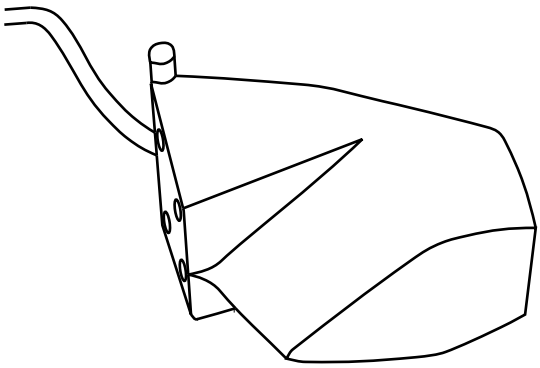


Figure 5.8. The wedge shaped transducer.

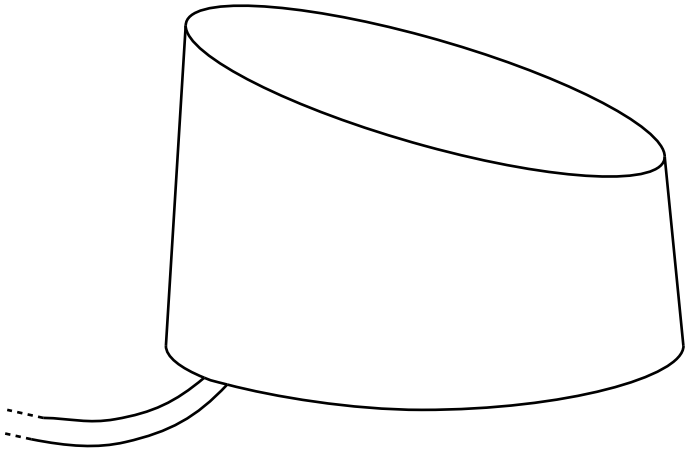


Figure 5.9. Puck transducer.

Chapter 6

The Robot

The robot consists of a B400 pool cleaner, the sonar, a step-motor, and a notebook running Linux. The sonar which is modified to supply the received sound level is connected to the sound input of the sound card of the computer. The stepper motor is connected to a motor control card, which in its turn is connected to the computers serial-port. The pool cleaner is connected to its trolley which is connected to the parallel-port via a card which converts the five volt digital system of the computer to the twelve volt system of the pool cleaner, see figure 6.1.

6.1 Step-motor

The transducer is mounted on a stepper motor to allow a 360 degree scan. This is used to generate local maps. The motor is mounted in a waterproof case. The motor's resolution is 1.8 degrees which means 200 steps per revolution. The control board has the possibility to generate half steps, which means a resolution of 0.9 degrees or 400 steps per revolution. The control board is called Simple Step Control Board, SSCB, and is made by Magna Associates Inc. and it has commands for setting the motor position, setting stepper speed and setting acceleration and deceleration etc. A manual can be downloaded from (Simplestep `http://www.simplestep.com`). The step-motor control board is connected to the serial-port.

6.2 Card to Parallel-port

Between the computer and the pool cleaner there is a card that converts the signals from the twelve volt system of the pool cleaner to the five volt system of the computer.

The control is through an open-collector circuit that is active high. This circuit allows easy switching between automatic and manual control.

When it comes to the out-signal from the pool cleaner there are two of them to indicate if the front or the rear bumper is engaged. These are made to go from twelve to five volt using an optocoupler.

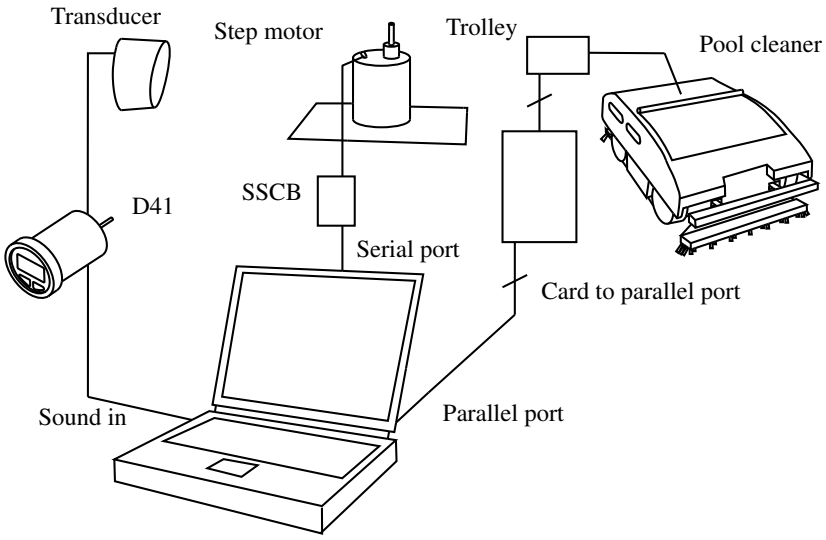


Figure 6.1. Connections.

6.3 Mobility

The traction system for the cleaner is a pair of tracks. The turning of the vehicle is thus through skid steering. The static and dynamic friction are quite different which complicates the control. During turning one track is stopped and the other is running. As the static friction often is larger than the dynamic friction it is difficult to achieve a stable control law. In water the difference between static and dynamic friction is even larger than in air, which further complicates the problem. The cleaner might in fact refuse to turn at all if the stopped track has enough friction due to dirt and roughness of the tiles in the pool. From revolution time measurements in Fyrishov the cleaner was prevented to rotate by these irregularities[1], see figure 6.2. If the cleaner were manually operated this wouldn't appear to be a problem because then one would only push the button a couple of times and in the meantime the pool cleaner would have moved in a straight line just long enough to move away from the locked position.

The result from measuring the revolution time is that it varies significantly. In Weda the result was approximately 15 seconds per revolution while in Fyrishov the problem of irregularity of the floor really became significant. If the pool-cleaner at all rotated a whole revolution it was done in approximately 9 seconds, but mostly the machine was trapped at a fraction of a revolution, according to figure 6.2. The speed of the machine was also measured and it was approximately 0.26 m/s.

[1]If a similar vehicle were considered on land it would probably behave differently, because the driving track would have considerable friction. If at all the power of the engine was enough to still rotate the track, it would probably push the whole machine away from the blocking brick or the whole machine would shiver until the bottom irregularity was overcome. But since water is a very good lubricant once the track has started to slide the pool-cleaner will instead be steadily locked in its position.

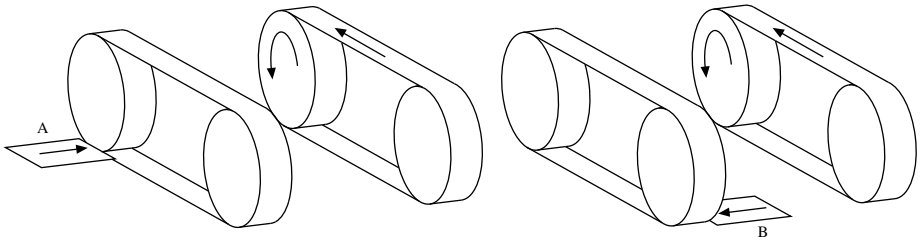


Figure 6.2. Irregularities in the floor can prevent the pool cleaner from rotating. The properties of water makes it a very good lubricant between a surface in motion and the bottom, while a stopped surface can still have a very good grip.

Pool respective pool-cleaner	Braked track	Brake time [s]	Measured angle [degrees]	Estimated revolution time [s]
Weda	left	15.0	360	15.0
Weda	right	15.0	370	14.6
Weda	left	14.8	370	14.4
Weda	right	14.8	380	14.0
Weda	left	14.5	330	15.8
Weda	right	14.5	340	15.4
Fyrishov	left	9	360	9
Fyrishov	right	9	360	9
Fyrishov	left	9	40	∞
Fyrishov	right	9	30	∞
Fyrishov	left	9	180	∞
Fyrishov	right	9	100	∞

Table 6.1. Revolution time of the pool-cleaner, The test is done so that a track is braked a fixed time, and then the rotation angle is measured.

Chapter 7

Map-making Tests

The algorithms described in the report have been tested under two different conditions. For day-to-day evaluation a test pool at Weda in Södertälje has been used. This is a small pool that has a maximum dimension of 6 m. For exhaustive tests the pool at Fyrishov in Uppsala has been used. This is the largest adventure type pool in Sweden (shown in figure 7.5). This pool has a highly irregular shape and several islands and other objects that pose an interesting challenge for automatic cleaning.

7.1 Test procedure

The pool cleaner is put in the pool with all equipment connected according to figure 6.1, and the transducer is aimed forward.

The initial scan of the pool is used as an initial map. The robot is then driven to different locations under semi-automatic control. At different points a scan is acquired and an automatic match and map-update is carried out. Through use of a number of intermediate maps a full map of the pool is generated. As part of the map matching an update of the map and the vehicle position is generated.

The system is implemented in Matlab, with appropriate procedures for interfacing to the platform. During each iteration a map is drawn and the operator points to the next location. Subsequently the vehicle moves to this location. This process is repeated until a full map has been acquired.

7.2 The Pool at Weda

The pool at Weda is a pool that is used for testing pool cleaners and other equipment that is produced. It is six meters long and 1.1 meters deep and has one side a bit curved. There is a plateau in the pool by one of the short sides which has one side sloping so that a pool cleaner can move up on it.

When the sonar is aimed in the direction of the plateau the slope leaves very weak echos. From many measurements neither the slope or the wall behind it is seen, i.e. their echo doesn't reach over the threshold.

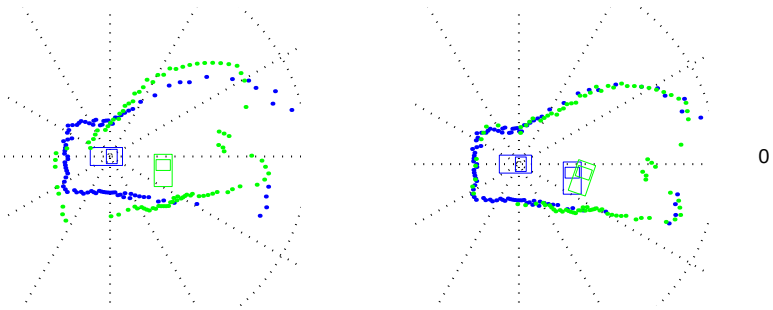


Figure 7.1. This figure shows how the robot's pose is updated from the map matching.

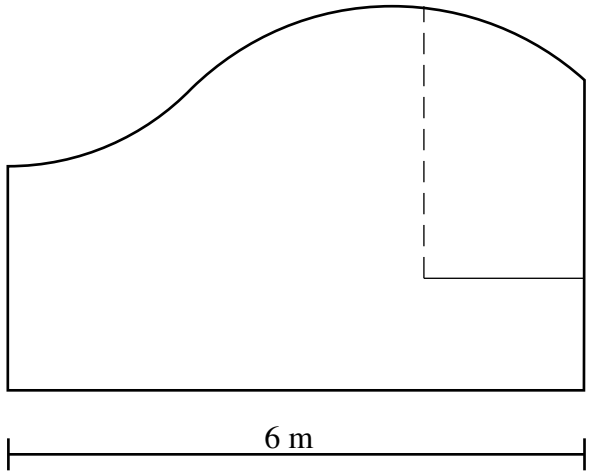


Figure 7.2. The pool at Weda.

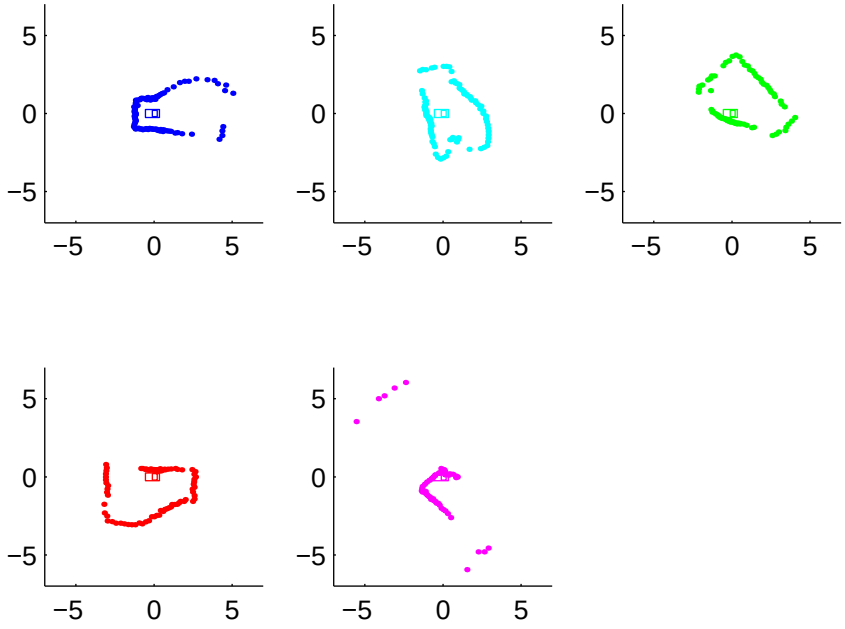


Figure 7.3. The local map obtained from a test run in the pool at Weda.

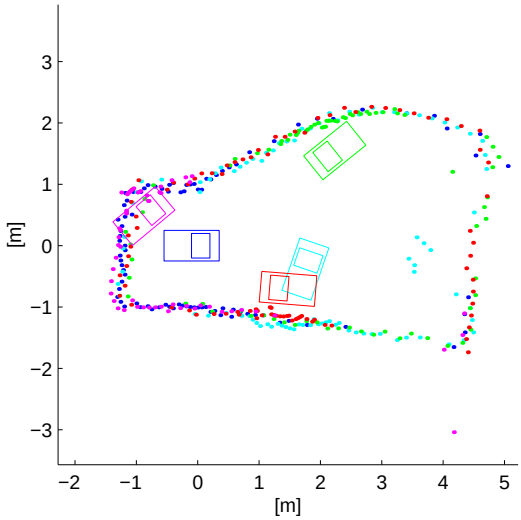


Figure 7.4. The result from subsequent matching and adding of the five scans.

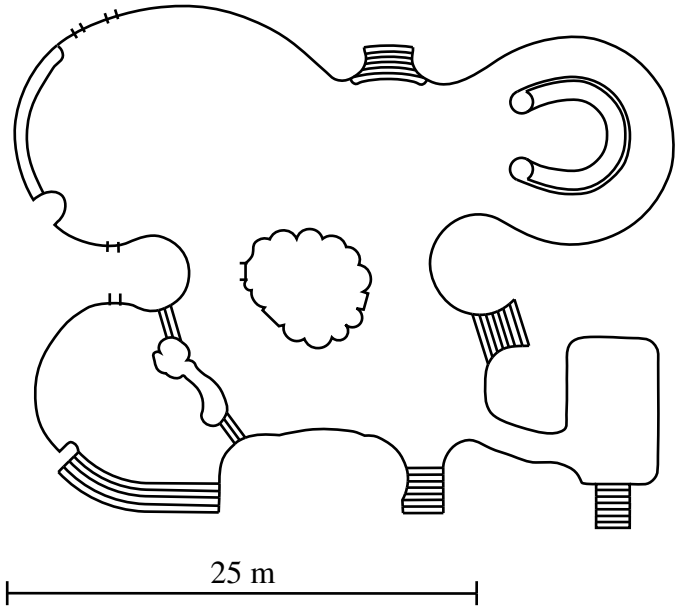


Figure 7.5. Fyrishov.

7.3 The Adventure-pool at Fyrishov

Fyrishov is the largest adventure-bath in Sweden. Except for this pool that is shown in the figure and that the machine was tested in, there is another just as big irregular outdoor pool. And furthermore there is a fifty meter square pool also indoors. The test results from Fyrishov are shown in figure 7.6 and figure 7.7.

7.4 Evaluation

When only the first echo from every measurement is chosen one misses whatever is behind the first object in the sound cone. If there is a narrow passage where the pool cleaner could move into then this passage could be missed if the sound beam is wider than the passage. If mixed readings are detected, i.e. measurements where part of the lobe is on a nearby object and another part hits a distant object then the readings will be corrupted. In general only the narrow object is detected. This implies that islands will be mapped wider than they are and passages will be mapped narrower than they are.

When building the map it is important that there is enough common area in both the map obtained so far and the new local map so that the matching algorithm has enough features to allow generation of a good fit. In other words the robot cannot move too far between two scans.

When it comes to outliers it is interesting to note that the floating cable and toys that are floating on the surface of Fyrishov, have not resulted in additional echoes which could have resulted in outliers.

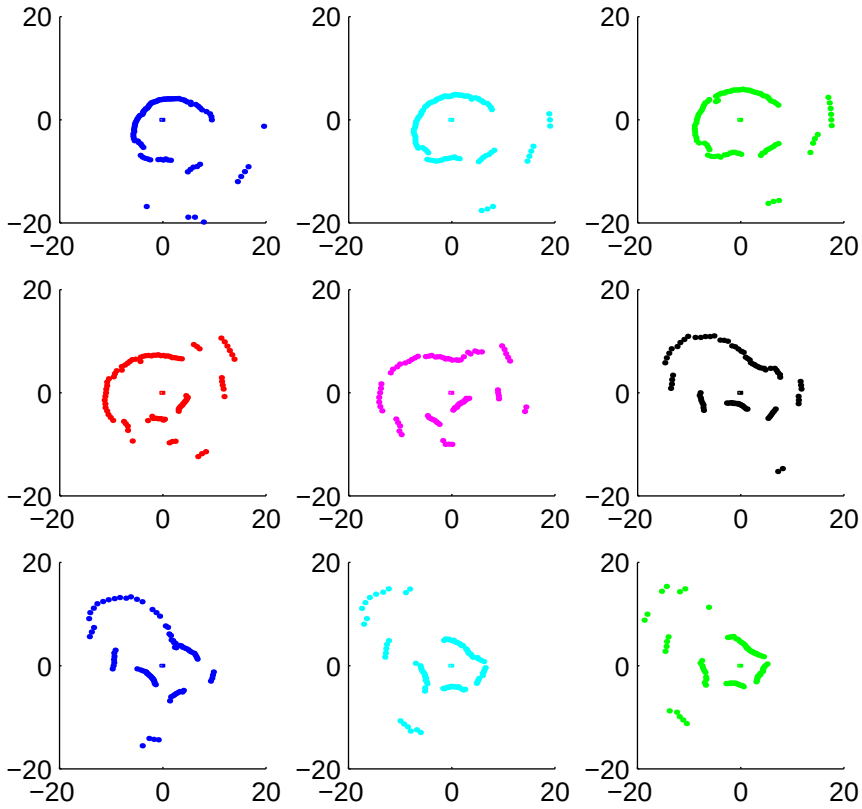


Figure 7.6. The local map obtained from a test run in Fyrishov.

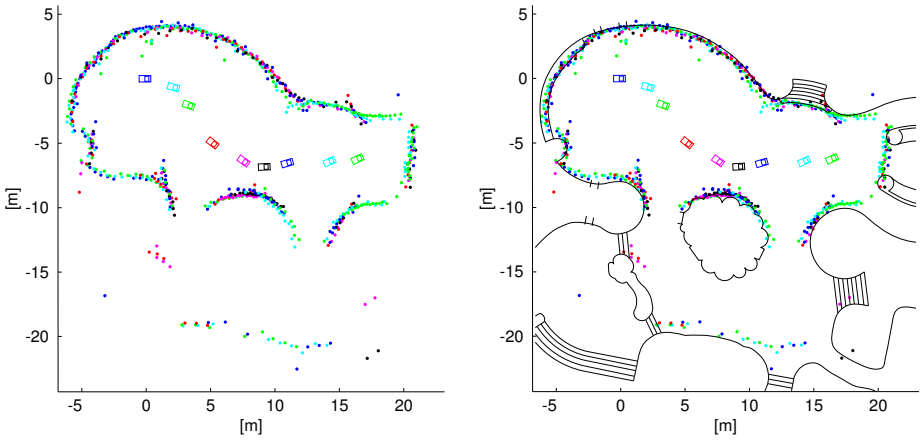


Figure 7.7. The result from subsequent matching and adding of the nine scans in Fyrishov. In the right figure a CAD-map is included for comparison.

7.5 Summary

The algorithm for map making has been tested in the pool at Weda and in an adventure pool in Fyrishov. The robot is driven to different locations in the pool under semi-automatic control. At each point a scan is acquired and a match and map-update is performed. The matching procedure is robust, all matching to the intermediate maps agree with the CAD-maps of the pools.

Chapter 8

Future Work

Mobility The mobility of the pool cleaner will limit the possible ways of how the poolcleaning strategy can be carried out. As described before the pool cleaner can get stuck when it tries to rotate, because of irregularities that give the locked track much higher friction than the rotating track. In almost the same way the pool cleaner can be prevented from rotating when it is close to a wall since the overhang of the pool cleaner touches the wall and the rotating track will just spin[1], see figure 8.1. If the wall is straight and the machine is manually operated this will not appear to be a big problem because then one would just push the turning button a couple of times so that the pool cleaner will move forward and get an increasing angle to the wall and then the pool cleaner would soon come loose. But if the wall is concave one would have a problem because whenever the pool cleaner has moved forward and gets a somewhat small angle to the wall the concavity of the wall would reduce that angle and the machine would again be pushed against the wall. Especially this happened in Fyrishov where there was a pool ladder and an extension blocking the way in the front of and behind the machine (figure 8.1).

The fact that the pool cleaner turns poorly and that it can get stuck next to the wall has given me an idea for how the cleaning strategy should be obtained. That is the pool cleaner should move radially in the cavities of the pool so

[1]If a similar vehicle were considered on land it would probably behave differently, because then the driving track would have considerable friction. If the power of the engine was enough to still rotate the track, it would probably push the whole machine away from the wall. But since water is a very good lubricant, once the track has started to slide the pool-cleaner will not be pushed away from the wall but is instead steadily locked in its position.

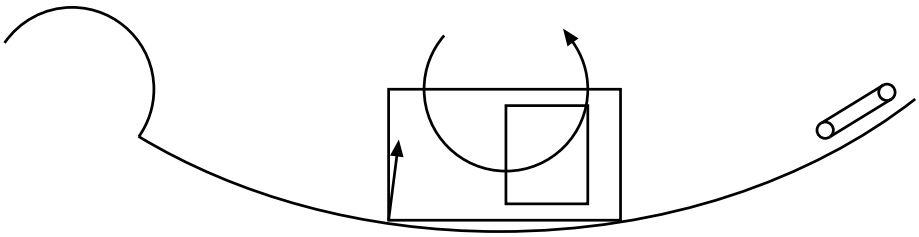


Figure 8.1. B400 trapped.

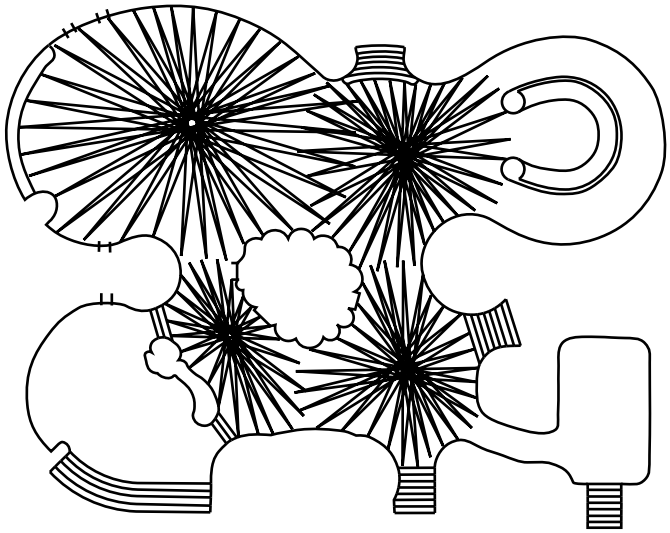


Figure 8.2. Radial cleaning strategy.

that it only moves in a straight path, which it is good at, and also so that it always meets the wall front-end or rear-end first avoiding the risk of ending up parallel to the wall where it can get stuck. This is not as optimal as one would be able to move the pool cleaner if one was steering it by hand, since certain areas are covered more than once. But the time factor is not usually the biggest problem for the baths. The pool cleaner is put in the pool in the evening and it has the whole night until the next days opening to clean the pool. A robust systematic strategy without making large hardware changes to the machine is more interesting.

Sensing The 200 kHz sonar is not the best frequency to use in the pool environment. Between 500 kHz and 1 MHz is probably better. Higher frequency would mean that it is possible to have a narrower beam, to get more diffuse echos which is better for map making, and to reduce the mirror effect of the walls. Yet the frequency must not be increased to much because then damping will become significant, which will reduce the maximum length of the sonar.

Matching Very simple features are used in the matching and one could instead use more complicated ones. There is here a trade-off between complexity of preprocessing versus matching.

Exploration There are different suspicious areas that need to be investigated further. First when there is no echo from a certain direction it can be because the wall is too far away or that the angle of incidence of the beam is too large. Secondly if the wall is very far away the beam will have become wider and there can be narrow channels that are missed. Thirdly if there are islands or half-islands or other extensions in the pool that are blocking the sight of the beam, the pool cleaner needs to move around the blockage to map the occluded

area. These occlusions are detected where there are sharp jumps in the sonar measurements.

Chapter 9

Conclusions

Problem The problem of automatic map-making is to acquire sensors and to develop a map-matching algorithm. In map-based positioning a local map is created which is compared to a global map to obtain the robot's position and orientation. The global map can either be a preexisting map or it can be obtained automatically by the robot itself, as it matches local maps to each other.

It is not a stupid idea to aim directly on automatic map-making instead of using preexisting maps since sensors and map-matching algorithms are parts that need to be developed anyway.

Solution A sonar made for recreational boats which has been modified by the manufacturer to fit this project is used. It has been modified in such way that it supplies the received sound level in the sonar working-frequency band. The sonar transducer is then rotated by a stepper motor to make measurements around the pool cleaner, which then are used to create a local map.

The map matching procedure, is done by randomly selecting features from each map which are then fitted together. For each choice of features the correspondence between the maps is calculated through voting. When enough match attempts are made one chooses the one with the best correspondence. Building the global map is done by merging the local maps from different positions in the pool.

Lessons Learned Some lessons learned in this project are that the walls can be detected efficiently by detecting the flanks of the sonar signal and that through voting based matching one is able to match the maps robustly. A third lesson is that a detailed control model is needed for the vehicle.

Bibliography

[Borenstein] J. Borenstein, H. R. Everett, L. Feng, *Navigating Mobile Robots*, 1995

[Bodn] H Bodn, U Carlsson, R Glav, HP Wallin, M bom, *Ljud och vibrationslära kompendium*, 1995

[Adkins] C. J. Adkins, *Equilibrium Thermodynamics*, 1968

[Petersson] Jan Petersson, *Fourieranalys*

[Benson] Harris Benson, *University physics*, 1991

[Press] William H. Press, Saul S. Teukolsky, William T. Vetterling, Brian P. Flannery, *Numerical Recipes in C*, 1988

[Zunino] Guido Zunino, Maurizio Simoncelli, *Autonomous pool cleaning*, 1999

[Macmillan] Angus Macmillan, Karl-Anders Norlin, *Teknik under ytan*, 1998

[Ghanbarzadegan] Payam Ghanbarzadegan, *Sensor-Based Navigating Pool-Cleaning Robot*, 1998

[Simrad EA 500] Kongsberg Simrad AS, Product specification `http://www.kongsberg-simrad.com/Products/Hydrographic_Survey/Singlebeam/160768a.pdf` (available Oct 2000)

[Launer] Robert L. Launer, Graham N. Wilkinson, *Robustness in Statistics*, 1979

Appendix A

Source Code

A.1 Introduction

The software is written in Matlab, C and C++, wherever a time consuming algorithm must be done, C or C++ is used, also for the hardware connections to the parallel-port, serial-port and the sound card, C or C++ is used. The software consists of the following files

mapT.m Matlab script that makes a global map, the main program

scan.m Matlab function that makes a local map

- **dspdip.cc** Program that waits for the next receive-cycle and records it
- **sttt.dat** Data file used by dspdip.cc
- **AbD.m** Matlab function

tset57600.c Program used with the serial-port

sleepm.c Program to sleep for some time, used when moving the pool cleaner

suarea.m Matlab script

pport.c Program for reading and writing to the parallel-port.

- **pttt.dat** Data file used by pport.c

fitT.m, Matlab script, for matching two maps, it makes use of twor.cc, i1.dat, i2.dat, and o.dat.

- **twor.cc** Program for matching two maps
- **i1.dat** Data file, map number one
- **i2.dat** Data file, map number two
- **o.dat** Data file, match result

pcleaner.m, Matlab function that plots a pool cleaner at a certain pose and certain color.

onemap.m, Matlab script that plots the global map.

A.1.1 Main Program, mapT.m

The main program first initializes the step-motor card, then it makes a first local map to constitute the current map. Then a loop is started where new local maps are obtained and matched to the current map, and then they are merged so that the current map grows for every new local map.

A.1.2 Scanning, scan.m

The scanning is made in the Matlab function scan.m. It also makes use of dspdip.cc sttt.dat and AbD.m. The transducer is stepped for a hundred steps around the pool cleaner and for every direction a receive-cycle is recorded. Then all recordings are smoothened and a distance is obtained from every recording, and at last outliers are removed.

A.1.3 Matching, fitT.m and twor.cc

The matching is done in fitT.m. It also uses twor.cc, i1.dat, i2.dat and o.dat. For two thousand times common features are chosen from the maps. The maps are matched according to the features and two values are calculated which represent how good the match was. These values are the fit-value, sum of closest points weighted according to the penalty function in 4.2 d, and the number-of-points-below threshold in the penalty function. Then a good match is chosen from a combination of the values, as described in figure 4.6.

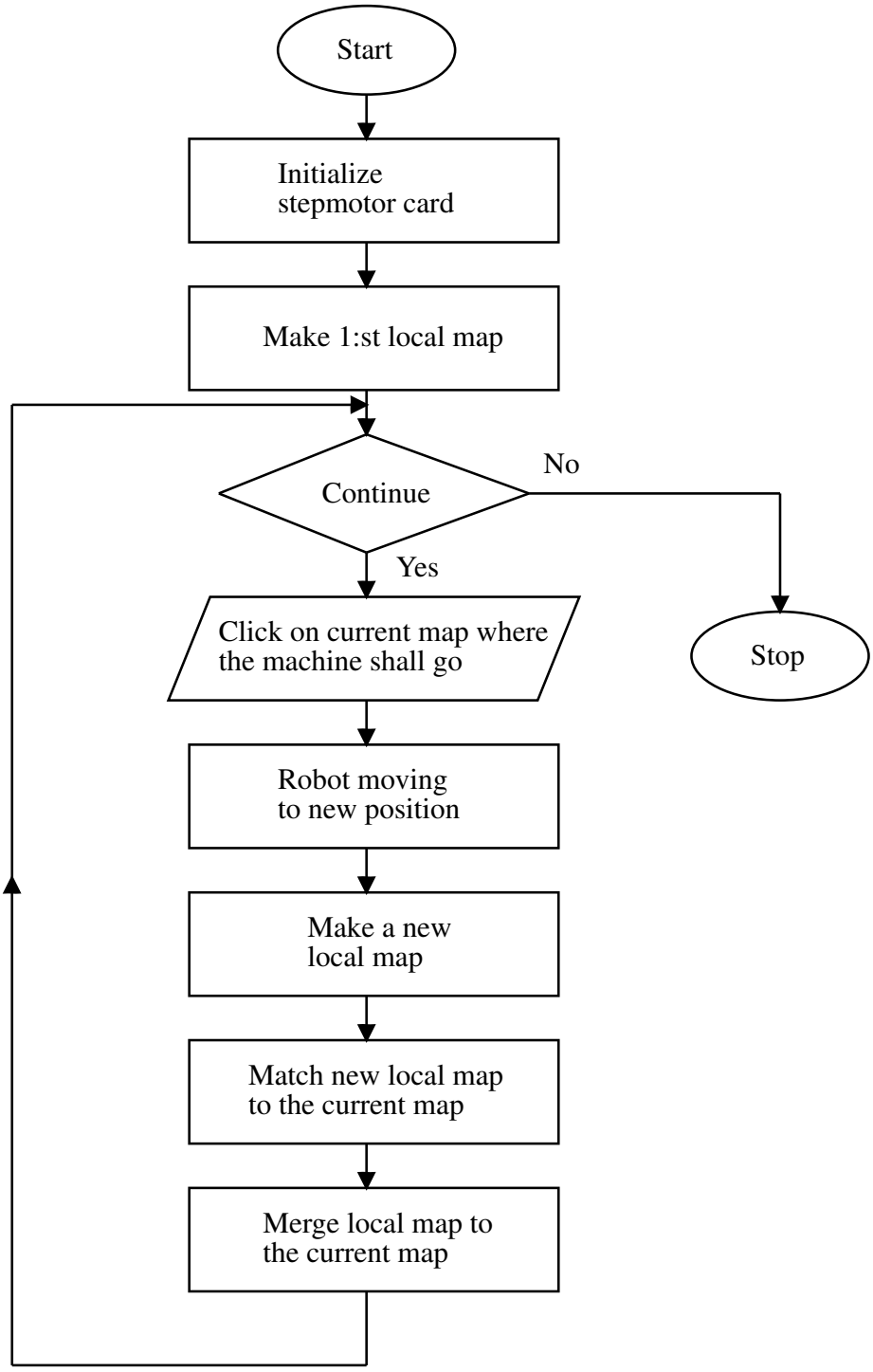


Figure A.1. Flowchart of main program.

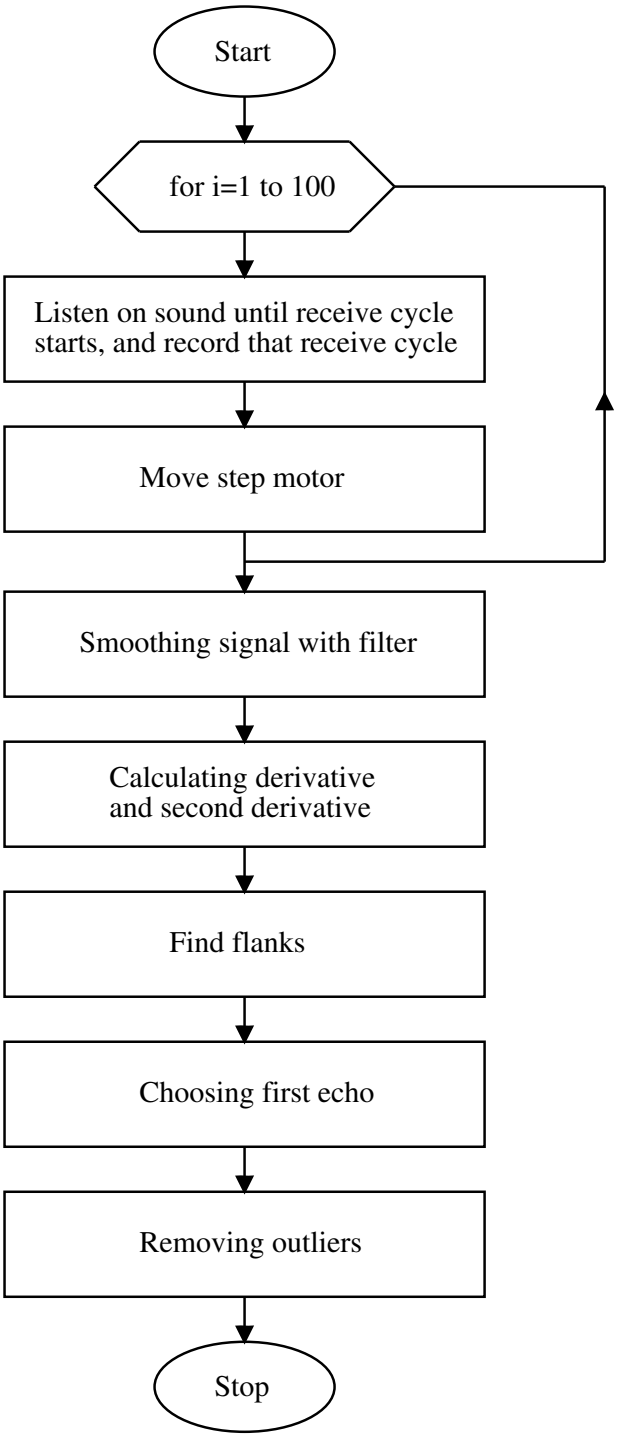


Figure A.2. Flowchart of scanning.

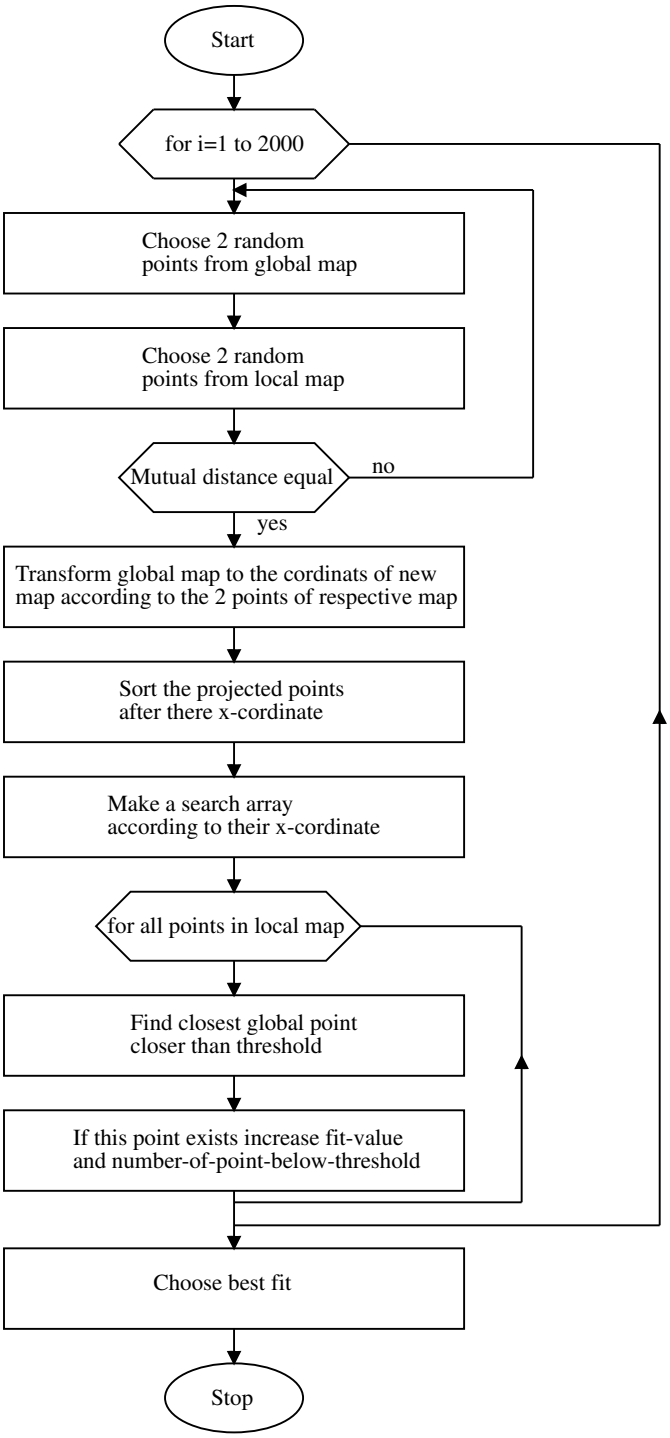


Figure A.3. Flowchart of matching.

A.2 mapT.m

```
%
% This is the main program and it matches subsequent local maps
% to form a global map.
%

clear all;close(findobj('Type','figure'));format compact;
fclose('all');
global Dfrfile Efrdsp Estep Emeandev BIGM BIGM2 fds Caxis

%
% Some debugger constansts and enable constants
%

Dfrfile=2;Dodi=0;Emeandev=0;Egmap=0;Dmsc=0;Dicp=0;%Eumap=1;
Efrdsp=1;Estep=0;Emove=0;Emark=0;Eplod=1;Estwice=0;
Efyrisbug=1;
    % D in the beginning of constant name means Debugger and E
    % means Enable. Of the two lines above, the first shall be
    % zero and the second shall be one, in general, when the
    % program is running sharp.

if Dfrfile
    load wod2;svf(1).uw=[];svf(1).uwl=[];svf(1).dw=[];
        svf(1).dwl=[];svft=svf;

    if 0
    load frot1; svf(1).uw=[];svf(1).uwl=[];svf(1).dw=[];
    svf(1).dwl=[];svft=[svft svf(1) svf(3)];
    load fod1; svf(1).uw=[];svf(1).uwl=[];svf(1).dw=[];
    svf(1).dwl=[];svft=[svft svf(1)];
    load fod2; svf(1).uw=[];svf(1).uwl=[];svf(1).dw=[];
    svf(1).dwl=[];svft=[svft svf(2:3)];
    end;

    svf=svft;
    od_seq=odf;
end;

Ctmp=10;
Caxis=[-Ctmp Ctmp -Ctmp Ctmp];
    % Used to set the cordinate axis of the plots

d2r=pi/180;
r2d=180/pi;

Cleft='1';Cright='2';Cforward='4';Cstop='8';Cbackward='10';
Cfbump=16;Cbbump=32;
if Efyrisbug
```

```
    Cleft='1';Cright='2';Cforward='10';Cstop='8';Cbackward='4';
    Cfbump=16;Cbbump=32;
end;

v=.26; %m/s
vl=2*pi/14.65; %rad/s
vr=2*pi/14.48; %rad/s

Cfbump=32;Cbbump=16;

BIGM=30;BIGM2=10*BIGM;
Fs=8192;
thhld=NaN; %obsolete
G=0:4:399;

fields={'sS','reads','fi','ro','t','G','BIGM','BIGM2','Fs', ...
    'V','thhld','uw','uwl','dw','dwl'};
empty={[],[],[],[],[],G,BIGM,BIGM2,Fs,[],thhld,[],[],[],[]};


ttmp=linspace(0,700);
angtmp=8.17e-6*ttmp.^2+.0022*ttmp;
ttmp1=linspace(2*ttmp(end)-ttmp(end-1),20000);
angtmp1=linspace(2*angtmp(end)-angtmp(end-1),720);
angNtime=[angtmp angtmp1;ttmp ttmp1]; % obsolete

%
% Initialize stepmotor card
%

fds=fopen('/dev/ttyS0','r+');
!tset57600
if Estep
fwrite(fds,['D0N' 13],'char'); eval(['!sleepm ', num2str(50)]);
fwrite(fds,['D0M0' 13],'char'); eval(['!sleepm ', num2str(50)]);
fwrite(fds,['D0B5000' 13],'char');
    eval(['!sleepm ', num2str(50)]);
fwrite(fds,['D0E45100' 13],'char');
    eval(['!sleepm ', num2str(50)]);
fwrite(fds,['D0P1' 13],'char');eval(['!sleepm ', num2str(50)]);
fwrite(fds,['D0S100' 13],'char');eval(['!sleepm ', num2str(50)]);
end;

%
% Make first scan
%

%sv=cell2struct(empty,fields);
if Dfrfile svV=scan(svf(1));
else svV=scan(cell2struct(empty,fields));end;
```

```
%
% Remove points that represent infinity
%

%M=svV;
rotmp=svV(1).ro;
fitmp=svV(1).fi;
[indtmp]=find(rotmp>=BIGM);
    rotmp(indtmp)=[];
    fitmp(indtmp)=[];

%
% Store in the current map
%

[Umap.x,Umap.y]=pol2cart(fitmp,rotmp);
Umap.l=ones(size(Umap.x));
Umap.sc=ones(size(Umap.x));
Umap.r=rotmp;

is=1;
suarea
%a_od=[];
%a_c=[];
%atotal=[0 0 0]';

ang_t=0;
ang_e=0;
    %total angle since start, to avoid entanglements of the cable
pos_v=[0 0 0]';
od_g=[0 0 0]';

while 1
    if Emark
    disp('Mark new poses cordinates (or hit any key to stop): ');
    [ai(1),ai(2),button]=ginput(1);
    if button~=1 fclose all;break;end;
    hold on;plot(ai(1),ai(2),'co');hold off;
    disp('Mark new poses angle: ');
    [tmpx,tmpy]=ginput(1);
    hold on;plot([ai(1) tmpx],[ai(2) tmpy],'b');hold off;drawnow
    [ai(3),trash]=cart2pol(tmpx-ai(1),tmpy-ai(2));
    if Egmap ai=ai-pos_v(:,end)';end;
    else ai=[0 0 0];
    end;

    is=is+1;

    if Emove
```

```
%
% Calculate how the pool cleaner shall move. Then make first
% turn, 'fi1', move straight and then make second turn,
% 'fi2'.
%

[fi1,ro1]=cart2pol(ai(1),ai(2));
if ang_t(end)+ai(3)+fi1>4*pi fi1=fi1-2*pi;
elseif ang_t(end)+ai(3)+fi1<-4*pi fi1=fi1+2*pi;
end;

fi2=ai(3)-fi1;
if ang_t(end)+ai(3)+fi2>4*pi fi2=fi2-2*pi;
elseif ang_t(end)+ai(3)+fi2<-4*pi fi2=fi2+2*pi;
end;

eval(['!pport w ',Cforward,' 'num2str(5000)]);
if fi1>0
    eval(['!pport w ' Cleft]);
    eval(['!sleepm ', num2str(1000*fi1/vl)]);
elseif fi1<0
    eval(['!pport w ' Cright]);
    eval(['!sleepm ', num2str(-1000*fi1/vr)]);
end;

eval(['!pport w ',Cforward,' ',num2str(1000)]);
eval(['!sleepm ', num2str(1000*ro1/v)]);

eval(['!pport w ',Cforward,' 'num2str(5000)]);
if fi2>0
    eval(['!pport w ' Cleft]);
    eval(['!sleepm ', num2str(1000*fi2/vl)]);
elseif fi2<0
    eval(['!pport w ' Cright]);
    eval(['!sleepm ', num2str(-1000*fi2/vr)]);
end;

eval(['!pport w ',Cstop,' 'num2str(1000)]);

end;%Emove

%
% Take a new scan
%

if Dfrfile  svV=[svV scan(svf(is))];
else svV=[svV scan(cell2struct(empty,fields))];end;

%
```

```
% Decide what the reference map will be
%

indtmp=find(Umap.l>0);
if is==2 indtmp=find(Umap.l>0);end;
xRef=Umap.x(indtmp);
yRef=Umap.y(indtmp);
if Estwice
    indtmp=find(Umap.l>1);
    if is==2 indtmp=find(Umap.l>0);end;
    xRef=Umap.x(indtmp);
    yRef=Umap.y(indtmp);
end;

fiNew=svV(is).fi;
roNew=svV(is).ro;

if Dmsc  %Match another scan, for Debugging purpose
    iref=input('Give iref: ');
    inew=input('Give inew: ');

    fiRef=svf(iref).fi;
    roRef=svf(iref).ro;

    fiNew=svf(inew).fi;
    roNew=svf(inew).ro;
end;

%
% Remove map points which represent infinity
%

if Dmsc
[indtmp]=find(roRef>=BIGM);
roRef(indtmp)=[];
fiRef(indtmp)=[];
[xRef,yRef]=pol2cart(fiRef,roRef);
end;

[indtmp]=find(roNew>=BIGM);
roNew(indtmp)=[];
fiNew(indtmp)=[];
[xNew,yNew]=pol2cart(fiNew,roNew);

%
% Sort map points after their polar angle
%

[fiRef,roRef]=cart2pol(xRef,yRef);
[fiRef,indtmp]=sort(fiRef);
```

```
roRef=roRef(indtmp);
xRef=xRef(indtmp);
yRef=yRef(indtmp);

[xNew,yNew]=pol2cart(fiNew,roNew);
[fiNew,roNew]=cart2pol(xNew,yNew);
[fiNew,indtmp]=sort(fiNew);
roNew=roNew(indtmp);
xNew=xNew(indtmp);
yNew=yNew(indtmp);

%
% Calculate odometric startguess, 'od_m', from 'od_seq'
%

th=pos_v(3,is-1);
R=[cos(th) -sin(th);sin(th) cos(th)];
T=pos_v(1:2,is-1);
%tmp=R*od_seq(1:2,is-1)+T;
%tmp=[tmp;th+od_seq(3,is-1)];
%od_g=[od_g tmp];
od_g=[od_g  [R*od_seq(1:2,is-1)+T ; th+od_seq(3,is-1)]];
od_m=od_g(:,is);
if Dmsc
    th=-pos_v(3,iref);
    R=[cos(th) -sin(th);sin(th) cos(th)];
    T=-pos_v(1:2,iref);
    od_m=[R*od_g(1:2,inew)+T ; th+od_g(3,inew)];
end;

%
% Match procedure
%

if Dicp
    fiticp;
else
    fitT;
end;

%ABBprim;

pos_v=[pos_v pos_n];

%if ~Emark waitforbuttonpress;end;
%hold on; plot(pos_v(1,end-1)+ai(1),...
%pos_v(2,end-1)+ai(2),'o');
%tmpa3=sum(a_c(3,:),2);
%tmpa3=pos_v(3,is-1)+atmp(3);
%R=[cos(tmpa3) -sin(tmpa3); sin(tmpa3) cos(tmpa3)];
```

```
    %pos_v=[pos_v(1:2,:) pos_v(1:2,end)+R*atmp(1:2)
    %    pos_v(3,:) pos_v(3,end)+atmp(3)];
    %atotal=atotal+sum(a_c,2);

    %
    % Keep track of how much the poolcleaner has rotated
    %

    tmp=(pos_v(3,is-1)+od_seq(3,is-1))-pos_v(3,is);
    if tmp<-180*d2r
        tmp=tmp+360*d2r;
    elseif tmp>180*d2r
        tmp=tmp-360*d2r;
    end;
    ang_e=[ang_e tmp];
    ang_t=[ang_t ang_t(end)+od_seq(3,is-1)-tmp];

    %
    % Plot suspicious areas
    %

    suarea;

    %
    % Merge the new map to the current map
    %

    onemap;

    %
    % Plot the global map, also plot the pool-cleaner poses
    %

    clf; axis equal;hold on;
    cstr=['b','c','g','r','m','k'];
    for i=1:is
        tmpind=find(Umap.sc==i);
        pcleaner(pos_v(:,i),cstr(mod(i-1,length(cstr))+1));
        if Eplod
            pcleaner(od_g(:,i), ...
                [cstr(mod(i-1,length(cstr))+1) '--']);
        end;
        plot(Umap.x(tmpind),Umap.y(tmpind),...
            ['.' cstr(mod(i-1,length(cstr))+1)]);hold on
    end;
    axis('equal');axis(Caxis);hold off;waitforbuttonpress


end;
```

A.3 scan.m

```
function sv=scan(sv);
%
% The function scan makes a local map and stores it in sv.
%
global Dfrfile Efrdsp Estep Emeandev fds


lowestb=20;
offsetb=20;

v=.3;
Fs=sv.Fs;thhld=sv.thhld;
G=sv.G;p=1:length(G);Deg=.9*G;rad=Deg/180*pi;back=0;

d=zeros(size(G));

maxRange=40; %maximum range in meters
maxl=round(maxRange*2/1500*Fs);
    %number of samples stored from each echo

%
% Setting the time varying threshold used to detect the walls
%

%ytmp=[1000 700 600 500 450 400 350 350];
%ytmp=[1000 800 700 600 525 400 350 350];
%x=0:30;
%y=x;
ytmp=[1000 900 700 600 500 500 600 565 525 450 400 375 ...
    350 350 350 350];
x=0:.5:30;
y=x;
y(1:length(ytmp))=ytmp;
y(length(ytmp)+1:end)=ytmp(end);
thhlx=[0 x/750*8192+offsetb];
thhly=[y(1) y];


%
% If the poolcleaner is not in the pool, Dfrfile=2 makes the
% scaning to be skipped, and one uses a stored result. If one
```

```
% sets Dfrfile=1 the stepmotor is rotated but the sound is still
% read from stored values.
%

if Dfrfile==0 | Dfrfile==1

tic;

sound(sin(1:2:400));

%
% Make a scan
%

for i=p

    eval(['!sleepm ', num2str(1000*.1)]);

    t(i)=toc;

    if Dfrfile==0 & Efrdsp==0
        eval(['!rm sttt.dat']);
        eval ...
           (['!vrec -r -s ' num2str(Fs) ' -b 16 -t 1 sttt.dat']);
    elseif Dfrfile==0 & Efrdsp==1
        eval(['!dspdip >sttt.dat']);
    elseif Dfrfile==1 & Efrdsp==1
        fptmp=fopen('dsp.dat','w');
        fwrite(fptmp,-sv.sS(i,:),'int16');
        fclose(fptmp);
        eval(['!dspdip >sttt.dat']);
    end;

    if Estep==1
        if i~=p(end)
            stepstr=['D0M' num2str(G(i+1)) 13];
                %start rotete stepmotor already now, for next
                %sound read.
            fwrite(fds,stepstr,'char');
            a=fread(fds,[1 3],'char');
            %setstr(a)
        else
            stepstr=['D0M' num2str(0) 13];
            fwrite(fds,stepstr,'char');

        end;
    end;

    fid=fopen('sttt.dat','r+');
    s=fread(fid,inf,'int16');
```

```
    %s=sstmp;
    fclose(fid);
    s=-s';

    if  Efrdsp==1
        size(s);
        reads(i,:)=s;
        sS(i,:)=s;
    elseif Efrdsp==0
        sS(i,:)=s;
        %stmp=s;
        %ind=find(stmp>10000);
        %startind=min(ind);
        %if startind<.03*Fs
        %    ioi=find(ind>.05*Fs);
        %    startind=min(ind(ioi));
        %end;

        [tmp,tmpind]=min([zeros(1,19) s(20:end-maxl-2)]);
                %values approx
        tmpind1=find(s(tmpind:tmpind+20)<.85*tmp);

        startind=tmpind+max(tmpind1)-1-round(0.002*Fs);


        %startind=startind-round(0.002*Fs);
        reads(i,:)=s(startind:startind+maxl-1);
        t(i)=t(i)+startind/Fs;
    end;


    i=i+1;
end; %make of scan
clear sStmp;

sound(sin(1:2:400));
else reads=sv.reads; t=sv.t; sS=[];
end; %Data from file or not (Dfrfile==0 |Dfrfile==1)

%
% Smoothening the signals
%

readst=reads;

order=22;  % must be even value
coeff=diag(rot90(pascal(order+1)))';
B=ones(size(readst,2),1)*coeff/sum(coeff);
A=spdiags(B,(-order/2):(order/2),size(readst,2),size(readst,2));
```

```
tmp=A(1:order/2 ,1:order)*sum(coeff);
A(1:order/2 ,1:order)=tmp./(sum(tmp,2)*ones(1,order));
tmp=A(end-order/2+1:end,end-order+1:end)*sum(coeff);
A(end-order/2+1:end,end-order+1:end)=...
    tmp./(sum(tmp,2)*ones(1,order));

readsf=zeros(size(readst));
for i1=1:size(readst,1)

    readsf(i1,:)=(A*readst(i1,:)')';
end

%
% Calculate the derivative and second derivative
%

readsd1=diff(readsf,1,2);
readsd1of=readsd1(:,20:end);
readsd2=diff(readsf,2,2);

%
% Extracting length measurments from the signals
%

tmp=diff(sign(readsd2),1,2)  <  0 ;
%readst1=readsd1>thhld & ...
%    [zeros(size(tmp(:,1))) tmp zeros(size(tmp(:,1)))];
tmp1=ones(size(readsd1(:,1)))*...
    interp1(thhlx,thhly,1:size(readsd1,2));
readst1=readsd1>tmp1 &...
   [zeros(size(tmp(:,1))) tmp zeros(size(tmp(:,1)))];
readst1of=readst1(:,20:end);


[J,I]=find(readst1of');
It=I;
Jt=J;
if isempty(t) t=ones(size(G));end;
tt=t(I)';
Gt=G(I)';
radt=rad(I)';

%
% Selecting the first echo
%

tmpi=[];
for i1=1:size(I,1)-1
    if I(i1+1)==I(i1)
        tmpi=[tmpi i1+1];
```

```
    end;
end;
I(tmpi)=[];
J(tmpi)=[];
%t(tmpi)=[];
radt(tmpi)=[];

%
% Make a plot for debugging purpose
%

if 0
readst=readsd1;
imagesc([readst zeros(size(readst(:,1))); ...
   zeros(size(readst(1,:))) 0]);
%xlabel('meter');ylabel('meter');
%shading flat
%view(2);
zoom on;
hold on; plot3(Jt+19.5,It+.5,35000*ones(size(It)),'k*');
%plot3(J+19.5,I+.5,35001*ones(size(I)),'ko');
%plot3(J+19.5,I+.5,35001*ones(size(I)),'g*');
hold off;
waitforbuttonpress
end;


tmp=readsd1(:,20:end);
V=tmp(sub2ind(size(tmp),I,J));



Vn=V;
In=I;
Jn=J*750/8192;
%polar(radt,Jn,'.');
%waitforbuttonpress

%
% Remove outliers, in directions where no echo is found there
% a distance of BIGM is used.
%

BIGM=sv.BIGM;   %30;
infind=1:length(rad);
infind(In)=[];
radt1=[radt;rad(infind)'];
Jn1=[Jn;BIGM*ones(size(rad(infind)))'];
Vn1=[Vn;ones(size(rad(infind)))'];
[radt1,indtmp]=sort(radt1);
```

```
Jn1=Jn1(indtmp);
Vn1=Vn1(indtmp);

%radt1=radt;
%Jn1=Jn;
%Vn1=Vn;
if ~Emeandev
for i=1:length(radt1)
    if i==1
        if Jn1(i)<Jn1(i+1)*.9 & Jn1(i)<Jn1(end)*.9 Jm(i)=BIGM;
        else Jm(i)=Jn1(i);end;
    elseif i==length(Jn1)
        if Jn1(i)<Jn1(i-1)*.9 & Jn1(i)<Jn1(1)*.9 Jm(i)=BIGM;
        else Jm(i)=Jn1(i);end;
    else
        if Jn1(i)<Jn1(i-1)*.9 & Jn1(i)<Jn1(i+1)*.9 Jm(i)=BIGM;
        else Jm(i)=Jn1(i);end;
    end;
end;
end;
if Emeandev
C=20;
disp('meandeviation');
astart=[1/6 0 0]';
a=astart;
l=12*pi/180;
for i=1:length(rad)
    ind=find(radt1<rad(i)+l & radt1>=rad(i)-l);
    ind2=find(radt1<rad(i)+l+2*pi & radt1>=rad(i)-l+2*pi);
    ind3=find(radt1<rad(i)+l-2*pi & radt1>=rad(i)-l-2*pi);


    %radnn=radt1(ind)-rad(i);
    %Jnn=Jn1(ind);
    %Vnn=Vn1(ind);
    %plot(radnn,Jnn,'.g')

    radnn=[radt1(ind2)-2*pi ; radt1(ind) ; radt1(ind3)-2*pi]- ...
        rad(i);
    Jnn=Jn1([ind2 ; ind ; ind3]);
    %Jnn=1./Jnn;
    Vnn=Vn1([ind2 ; ind ; ind3]);

    A=[ones(size(radnn)) radnn radnn.^2 ];

    b=Jnn;

    k=Vnn;

    if i==1 a=astart; end;
```

```
clear OPTIONS
OPTIONS(2)=1e-2;
OPTIONS(3)=1e-2;
OPTIONS(7)=1;
OPTIONS(14)=10000;

if ~isempty(A)
    %[a,options]=fminu( 'AbD',a,OPTIONS,'AbDG', ...
    %    A,b,ones(size(k)),C );
    [a,options]=fmins( 'AbD',a,OPTIONS,[], ...
        A,b,ones(size(k)),C );
    a=a+[.1 .1 .1]';
    [a,options]=fmins( 'AbD',a,OPTIONS,[], ...
        A,b,ones(size(k)),C );
    %OPTIONS(3)=1e-4;
    a=a+[.0001 .0001 .0001]';
    [a,options]=fmins( 'AbD',a,OPTIONS,[], ...
        A,b,ones(size(k)),C );

    Jm(i)=a(1) ;%+a(2)*rad(i) +a(3)*(rad(i)).^2;
    %Jm(i)=1/Jm(i);
    if Jm(i)<0.4

        a=[Jn1(i) 0 0]';
        %[a,options]=fminu('AbD',a,OPTIONS,...
        %        'AbDG',A,b,ones(size(k)),C );
        [a,options]=fmins( 'AbD',a,OPTIONS,[], ...
            A,b,ones(size(k)),C );
        a=a+[.1 .1 .1]';
        [a,options]=fmins( 'AbD',a,OPTIONS,[], ...
            A,b,ones(size(k)),C );
        a=a+[.0001 .0001 .0001]';
        [a,options]=fmins( 'AbD',a,OPTIONS,[], ...
            A,b,ones(size(k)),C );

        Jm(i)=a(1);%+a(2)*rad(i) +a(3)*rad(i).^2;
    end;
    if  abs(a(3))>100 | abs(a(2))>20
        %disp('********************');a(2),a(3)
        if i==1
            if Jn1(i)<Jn1(i+1) & Jn1(i)<Jn1(end)
                a=[BIGM 0 0]';
            else a=[Jn1(i) 0 0]';end;
        elseif i==length(Jn1)
            if Jn1(i)<Jn1(i-1) & Jn1(i)<Jn1(1)
                a=[BIGM 0 0]';
            else a=[Jn1(i) 0 0]';end;
        else
            if Jn1(i)<Jn1(i-1) & Jn1(i)<Jn1(i+1)
```

```
                    a=[BIGM 0 0]';
                else a=[Jn1(i) 0 0]';end;
            end;

            %a=[BIGM 0 0]';

            Jm(i)=a(1);%+a(2)*rad(i) +a(3)*rad(i).^2;
        end;

        x=linspace(-l,+l);

        y=a(1) +a(2)*x +a(3)*x.^2;

        %y=1./y;

        %hold off;plot(radt1,Jn1,'.');hold on
        %plot(rad(i),Jm(i),'sc');
        %plot(rad(i)+x,y,'m');hold off;drawnow;

        %eval('i');format long;eval('a''');format bank;
        %eval('options(8)');eval('options(10)');format short;
        %waitforbuttonpress
        %scatter(radt1,Jn1,Vn1/10,Vn,'filled')
        axis equal
    end;
end;

ro=Jm;
fi=rad;
end; %Emeandev

ro=Jm;
fi=radt1';

%infind=1:length(rad);
%infind(In)=[];

%ro(infind)=[];
%fi(infind)=[];

hold off;polar(radt1,Jn1,'.');hold on
polar(fi,ro,'m+');hold off;zoom on;
%axis equal
%disp('scan ready');
if Emeandev sound(sin(1:2:400)); end;
waitforbuttonpress
%clear rot fit;V=who;fit=fi;rot=ro;clear(V{:});ro=rot;fi=fit;
%clear V fit rot;save rotv;
BIGM2=sv.BIGM2; %10*BIGM;
ro(find(ro>BIGM-1))=BIGM2;
```

```
%radlsq

sv.sS=sS;sv.reads=reads;
sv.fi=fi;sv.ro=ro;
return;
```

A.4 dspdip.cc

```
/* This program read small parts of the dsp- (sound-) input and
detects when the recieve-cycle starts. When it has started the
input is passed on to a output file. It is called from the
command line with 'dspdip'  */

#include <unistd.h>
#include <fcntl.h>
#include <sys/soundcard.h>
#include <sys/ioctl.h>
#include <iostream.h>
#include <string>
#include <stdlib.h>
#include <stdio.h>

#define DSP_NAME    "/dev/dsp"
//#define DSP_NAME    "dsp.dat"
#define DSP_MAXCOUNT 8192 /* # of samples to be recorded */
#define BUFFERSIZE 256
#define FS 8192
//#define PLATN 15
#define OUTFILESIZE 512

extern __inline__ unsigned long long int rdtsc()
    {
        unsigned long long int x;
        __asm__ volatile (".byte 0x0f, 0x31" : "=A" (x));
        return x;
    }

int dspfd;
int Fs1=FS,Fs=FS; //Fs1=60000,Fs=60000;
int res1=16,res=16;

//tms ntv,tv;
//clock_t ct;

int main(int argc, char *argv[])
{
    //string bufferlong;
    unsigned char buffer[BUFFERSIZE],bufferlong[DSP_MAXCOUNT];
```

```
int i , l,wn=0,tmp,foundedge=0,platN=0,foundplat=0,
    foundbottom=0;
int minpointB=0,startpointP,startpointB,foundminpointP=0,
    passOnData=0;
int minpointP=0,edgeN,lenbufferlong,buffval,startpoint,readB,
    intro=1,left;
int founddip=0;
int max_count = DSP_MAXCOUNT;
int next_count;


if ((dspfd=open(DSP_NAME, O_RDONLY,0))==-1)
    cout<<"open faild"<<endl;

if (ioctl(dspfd, SOUND_PCM_WRITE_BITS, &res)==-1) cerr<<
    "change bits failed"<<endl;
//cerr<<res<<endl;
if (ioctl(dspfd,SOUND_PCM_WRITE_RATE, &Fs)==-1) cerr<<
    "change samplerate failed"<<endl;
//cerr<<Fs<<endl;

next_count = sizeof(buffer);
//cerr<<"sizeof(buffer)"<<next_count<<endl;
//cerr<<rdtsc()<<endl;

readB=0;

//
// Read small parts and process the it
//

while (next_count > 0 &&
    (l=read(dspfd, buffer, next_count)) > 0) {
    for (i=0;i<l;i++) {bufferlong[readB+i]=buffer[i];}
    //cerr<<"readB: "<<readB<<endl;
    readB+=l;



    if (passOnData==0) {
        //cerr<<buffer[0]<<endl;
        for (i=0;i<l;i=i+2) {
            buffval=256*bufferlong[readB-l+i+1]
                +bufferlong[readB-l+i];
            if (buffval>32767) buffval=buffval-65536;
            buffval=-buffval;
            if (i==32) intro=0; //(0.002*8192)*2
            if (intro==1) buffval=0;

            if (buffval<-7500) {founddip++;}
```

```
            else {
                if (founddip>2) {
                    //"rising edge after dip:"<<
                    //    (readB-l+i+2)/2<<endl;
                    edgeN=readB-l+i-2;
                    cerr<<"edgeN: "<<(edgeN+2)/2<<endl;
                    passOnData=1;
                    //if (write(1, &buffer[i], l-i)==-1) {
                    //    cerr<<"error writing"<<endl;}
                    if (write(1,&bufferlong[edgeN-32],
                        readB-edgeN+32)==-1)
                        {cerr<<"error writing"<<endl;}
                      wn=readB-edgeN+32;
                      //cerr<<"wn: "<<wn<<endl;
                    break;
                 }
                else founddip=0;
             }
         } //end of for sling


     } //end of passOnData==0

    else if (passOnData==1) {

        left=OUTFILESIZE-wn;
        if (left>l) {
            if (write(1, buffer, l)==-1)
                cerr<<"error writing"<<endl;
            wn=wn+l;
            //cerr<<"l: "<<l<<" wn: "<<wn<<endl;
         }
        else {
            if (write(1, buffer, left)==-1)
                cerr<<"error writing"<<endl;
            //cerr<<"wn: "<<wn+left<<" exiting"<<endl;
            exit(0);
         }
     }

    max_count -= l;
    next_count = max_count; /* # of bytes left */
    if (next_count > sizeof(buffer))
        next_count = sizeof(buffer);
    //ct=times(&tv);
 }

if (l == -1)exit(-1);

exit(0);
```

```
}
```

A.5 AbD.m

```
function X=AbD(a,A,b,k,C)
%
% The function AbD is the figure-of-merit-function that is going
% to be optimized when one makes a robust fit of a selected
% number of data. It is used to remove outliers in the local map.
%

X=sum( abs(k.*(b-A*a)) ); %mean deviation

%X=sum( log(1+.5*(k.*(b-A*a)).^2) ); %Lorenzian

%X=sum( log(cosh(C*((b-A*a)))) );

%X=sum( (k.*(b-A*a)).^2 );%least square
```

A.6 tset57600.c

```
/* This program is called from the command line and it sets some
parameters of the serial connection to be adapted to the
stepmotor control card. */

#include <stdio.h>
#include <sys/ioctl.h>
#include <sys/termios.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>

#define BAUDRATE B57600
#define DEVICE "/dev/ttyS0"
#define _POSIX_SOURCE 1 /* POSIX compliant source */

int main(int argc, char *argv[]) {
int  id,tmp;
char s[12];
char c;
struct termios t,newtio,oldtio;



if ((id = open(DEVICE, O_RDWR | O_NOCTTY))<0)
    perror("couldn't open device");
/*printf("id%i\n",id);*/
```

```
tcgetattr(id,&oldtio); /* save current modem settings */
bzero(&newtio, sizeof(newtio));
    /* clear struct for new port settings */

newtio.c_cflag = BAUDRATE | CS8 | CLOCAL | CREAD;
newtio.c_iflag = IGNPAR;
newtio.c_oflag =OPOST | ONLCR;
newtio.c_lflag = 0;
/* blocking read until 1 char arrives */
 newtio.c_cc[VMIN]=0;
 newtio.c_cc[VTIME]=1;
/* now clean the modem line and activate the settings for
    modem */
 //tcflush(id, TCIFLUSH);
 tcsetattr(id,TCSANOW,&newtio);




}
```

A.7 sleepm.c

```
/* This program is called from the command line and it makes the
computer sleep for a time that is specified by its argument. It
is called by 'sleepm t' where t is the time in milliseconds. */

#include <stdio.h>
#include <unistd.h>
#include <asm/io.h>
#include <fcntl.h>
#include <string.h>
#include <setjmp.h>



jmp_buf ebuf;

int main(int argc, char *argv[]) {

int time;

//printf("hallo");

if (setjmp(ebuf)) {
    printf("usage: sleepm #time\n\
    #time = time, in useconds, wont be less than 1\n\
    Ex: sleepm 5000\n");
    return -1;
```

```
 }


if  (argc<2) longjmp(ebuf,1);
if  (argc>2) longjmp(ebuf,1);
if (!isdigit(argv[1][0])) longjmp(ebuf,1);

sscanf(argv[1],"%d",&time);

//printf("time: %d\n",time);

usleep(1000*time);


}
```

A.8 pport.c

```
/* This program is used to write or read to the parallel port. It
is called from the command line. */
//#define DEBUG
//#define DEBUG1

#include <stdio.h>
#include <unistd.h>
#include <asm/io.h>
#include <fcntl.h>
#include <string.h>
#include <setjmp.h>

#define BASEPORT 0x378 /* lp1 */

jmp_buf ebuf;

int main(int argc, char *argv[])  {

char action[3]={'\0','\0','\0'};
unsigned char value;
int time;


if (setjmp(ebuf)) {
    printf("usage: pport #action [#value] [#time]\n\
    #action = r or w, r=read, w=write \n\
    #value = hexvalue of byte to be written \n\
    #time = time, a decimal value, in microseconds, wont be less\
 than 1000\n\
    Ex: pport w 7f 50000\n");
    return -1;
```

```
 }

if  (argc<2) longjmp(ebuf,1);
if (argv[1][0]=='r') action[0]='r';
else if (argv[1][0]=='w') {
    if  (argc<3) longjmp(ebuf,1);
    if (!isxdigit(argv[2][0])) longjmp(ebuf,1);
    sscanf(argv[2],"%x",&value);

    if (argc==3) action[0]='w';
    else if (argc==4) {
        strcpy(action,"wt");
        if (!isdigit(argv[3][0])) longjmp(ebuf,1);
        sscanf(argv[3],"%d",&time);
      }
 }
else longjmp(ebuf,1);

//
// Comand line has now been interpreted, the variables 'action',
// 'value' and 'time' has been set
//

/* Get access to the ports */
if (ioperm(BASEPORT, 3, 1)) {perror("ioperm"); exit(1);}

outb(0x00, BASEPORT+2); //writeonly
//usleep(100000);

//printf("cr: %x\n", inb(BASEPORT+2));
//usleep(100000);

//
// If write, set the data signals (D0-7) of the port
//

if (action[0]=='w') {
#ifdef DEBUG1
    printf("value to write (decimal): %d\n", value);
    printf("value to write: 0x%x\n", value);
#endif
    //printf("value: %c\n", value);

    outb(value, BASEPORT);
    if (argc==4) {
        if (time<1000) time=1000;
#ifdef DEBUG1
        printf("time: %d us\n", time);
#endif
        usleep(100000);
```

```
        outb(0x00, BASEPORT);
     }
 }

#ifdef DEBUG
printf("d: %x\n", inb(BASEPORT));
#endif
//outb(0x00, BASEPORT+1);
//usleep(100000);

//
// If read, read from the status port (BASE+1) and display the
// result
//

#ifdef DEBUG
printf("status: ");
if (action[0]=='r') printf("status: %c\n", inb(BASEPORT + 1));
#endif
if (action[0]=='r') printf("%c", inb(BASEPORT + 1));


/* We don't need the ports anymore */
if (ioperm(BASEPORT, 3, 0)) {perror("ioperm"); exit(1);}

exit(0);
 }
```

A.9 fitT.m

```
%
% This script matches the two maps. It calls twor.cc. It then
% selects the best match from the basis of the result from
% twor.cc.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fac=1;

%
% Write maps to files i1.dat and i2.dat
%

fdi1=fopen('i1.dat','w');
for i=1:length(xRef)
    fprintf(fdi1,'%g %g ',fac*xRef(i),fac*yRef(i));
end;
fclose(fdi1);

fdi2=fopen('i2.dat','w');
```

```
for i=1:length(xNew)
    fprintf(fdi2,'%g %g ',fac*xNew(i),fac*yNew(i));
end;
fclose(fdi2);

%
% Make a great number of match atempts, by calling twor
%

%disp('wait');
clf;text(.5,.5,'wait (~5s)');drawnow;
tic;
eval('!twor i1.dat i2.dat');
toc

%
% Reading the result of the match atempts
%

fdo=fopen('o.dat','r');
tmp=fscanf(fdo,'%g');
fclose(fdo);
pr1V=tmp(1:6:end);
pr2V=tmp(2:6:end);
pn1V=tmp(3:6:end);
pn2V=tmp(4:6:end);
EV=tmp(5:6:end);
ptsV=tmp(6:6:end);

%
% Selecting best match
%

%cand=find(ptsV-1*EV>26);
%cand=find(ptsV>90 & EV<11);

%[trash, cand1]=sort(ptsV-5*EV);
%cand=cand1(end:-1:1);

[trash, cand]=sort(sqrt((ptsV-length(xNew)).^2+(3/fac*EV).^2));
hold off;plot(EV,ptsV,'.');%axis equal;
for i=1:10
    hold on;plot(EV(cand(i)),ptsV(cand(i)),'.k');
    plot(EV(cand(1)),ptsV(cand(1)),'.c');hold off;
    %waitforbuttonpress;
end;
waitforbuttonpress;

if 0
for i=1:10
```

```
    n=cand(i);
    pr=[pr1V(n)+1 pr2V(n)+1];
    pn=[pn1V(n)+1 pn2V(n)+1];
    th=atan2(yNew(pn(2))-yNew(pn(1)),xNew(pn(2))-xNew(pn(1)))-...
        atan2(yRef(pr(2))-yRef(pr(1)),xRef(pr(2))-xRef(pr(1)));

    [fitmp,ropj]=cart2pol(xRef-xRef(pr(1)),yRef-yRef(pr(1)));
    [xtmp,ytmp]=pol2cart(fitmp+th,ropj);
    xpj=xtmp+xNew(pn(1));
    ypj=ytmp+yNew(pn(1));
    [fipj,ropj]=cart2pol(xpj,ypj);
    hold off;polar(fipj,ropj,'.');hold on;
    polar(fiNew,roNew,'.c');
    polar(fiNew(pn),roNew(pn),'c');plot(xpj(pr),ypj(pr),'b--');
    hold off;

    eval('i');waitforbuttonpress
end;
end;

n=cand(1);
pr=[pr1V(n)+1 pr2V(n)+1];
pn=[pn1V(n)+1 pn2V(n)+1];

%
% Calculate the correct pose, pos_n, with respect to reference
% map.
%

th=atan2(yRef(pr(2))-yRef(pr(1)),xRef(pr(2))-xRef(pr(1)))-...
atan2(yNew(pn(2))-yNew(pn(1)),xNew(pn(2))-xNew(pn(1)));

R=[cos(th) -sin(th);sin(th) cos(th)];

T=-R*[xNew(pn(1));yNew(pn(1))]+[xRef(pr(1));yRef(pr(1))];

tmp=R*[0 1;0 0]+repmat(T,1,2);

pos_n=[tmp(1,1) tmp(2,1) ...
    atan2(tmp(2,2)-tmp(2,1),tmp(1,2)-tmp(1,1)) ]';
```

A.10 twor.cc

```
/* This program supplies a basis to determine the best match of
two maps. It is called by the command line 'twor i1.dat i2.dat'
where i1.dat and i2.dat is the two maps, and it returns the file
o.dat which contains a list of match attempts and there
respective fit values. The map files i1.dat and i2.dat are lists
of points which are represented in x- and y-cordinates. The
```

```
i1.dat is prefeably the largest map to make the algorithm as fast
as possible.*/


//#define MAXRANGE 2000
//#define WDIST 7.5
#define MAXRANGE 200
#define WDIST .75
#define ITERATIONS 2000
#define LONGN -56986

#include <unistd.h>
#include <fcntl.h>
#include <sys/soundcard.h>
#include <sys/ioctl.h>
#include <iostream.h>
#include <fstream.h>
#include <string.h>

#include <stdlib.h>

#include <stdio.h>
#include <math.h>
#include <time.h>


#define NUM_POINTS 2000
#define PI 3.14159
#define FLOOR_MAX 0.75

int num_pointsr,num_pointsn;    //the # of points in smaller file
int closeindx[MAXRANGE],closeindy[MAXRANGE];
static long longn;
long *longp;
int BIG_NUMBER = 20;
    //distances from corresponding points must be less than or
    //equal to BIG_NUMBER.

int M_BIG_NUMBER = 10;
    //BIG_NUMBER is for closest point and M_BIG_NUMBER is for
    //matching point

float Bound = 45 * (2 * PI / 360);
    //the bound for the matching point algorithm

float rotation = 45 * (2 * PI / 360);
    //keeps track of last iteration's rotation

float ER;           //ratio used to to judge how good match is
int time1 = 0;
```

```
int num;            //number of corresponding points found.


float B_omega = 20 * (2 * PI  / 360);
    //range for which to check for corresponding closest_points.

float omega_total = 0;   //keeps track of total movements
float T_x_total = 0;
float T_y_total = 0;

short mpr_cycle = 1;
float force_bound[8];


struct point
{
  float x;
  float y;
};

struct point P_orig[NUM_POINTS], P_ref[NUM_POINTS],
    P_new[NUM_POINTS], P_prime[NUM_POINTS],
    P_close_new[NUM_POINTS], P_pj[NUM_POINTS];

struct polar
{
  float R;
  float theta;
};

struct polar P_new_polar[NUM_POINTS], P_ref_polar[NUM_POINTS];

float ran1(long *idum);
void read_data(int argc, char *argv[]);
int compx(const void *, const void *);
int compy(const void *, const void *);

float odometry_x, odometry_y, odometry_rotation;

float curr_transl_x, curr_transl_y, curr_rotation;

FILE *fptr_1, *fptr_2;
FILE *fptr_3, *fptr_4;
ofstream ofile("o.dat");


/*************************************************************/
//
// Main program
```

```
//
/*************************************************************/

main (int argc, char *argv[])
{
 float E_dist, E_dist1, E_dist2, temp, omega_2, T_x_2, T_y_2,
     T_x_1, T_y_1, omega_1;
 float plot_x, plot_y;
 float floor;
 int (*funkp)(const void *,const void *);
 int i1, x, y, offsetx,offsety;
 float dist,disto,ftmp;
 struct point P_ox[NUM_POINTS],P_oy[NUM_POINTS];
 int shind[NUM_POINTS],tmpind,downch,pts,nextlow,nextup,up;
 float E,shortest[NUM_POINTS],tmpxl,tmpxla,tmpyl,tmpdist,xl,yl,
     cnextlow,cnextup;
 int i;

 FILE *fCheck_num1, *fCheck_num2 , *fptr_out;
     //variables are used while setting num_points
 int check_num1, check_num2;
 float dummy[2];



 int pr1,pr2,pn1,pn2;
 float th;
 float R[2][2],T[2];
 int ir;
 if (ofile==0) cerr<<"Couldnt open output file"<<endl;

/*****************************************************************
This code is used in order to set num_points which varies*/

 check_num1 = check_num2 = 0;

 fCheck_num1 = fopen(argv[1], "r");
 while(fscanf(fCheck_num1,"%g %g" , &dummy[0], &dummy[1]) != EOF)
   check_num1++;
 fclose(fCheck_num1);
 //printf("check_num1 = %d\n", check_num1);

 fCheck_num2 = fopen(argv[2], "r");
 while(fscanf(fCheck_num2,"%g %g", &dummy[0], &dummy[1]) != EOF)
   check_num2++;
 fclose(fCheck_num2);

 //printf("check_num2 = %d\n", check_num2);

 num_pointsr=check_num1;
```

```
 num_pointsn=check_num2;

 if ((num_pointsr == 0) | (num_pointsn == 0)) exit(-1);
 //if(check_num1 > check_num2)
 //  num_points = check_num2;
 //else
 //  num_points = check_num1;
 //if (num_points == 0) exit(-1);
 cout<<num_pointsr<<" "<<num_pointsn<<endl;


 read_data(argc, argv);


//
// Finding common features of the two maps
//

 longn=LONGN;
 longp=&longn;
 for(ir=0;ir<ITERATIONS;ir++) {
 //for(ir=0;ir<1;ir++) {
 while (1){
     pr1=(int)(num_pointsr*ran1(longp));
     pr2=(int)(num_pointsr*ran1(longp));
     while (pr1==pr2) {pr2=(int)(num_pointsr*ran1(longp));}
     pn1=(int)(num_pointsn*ran1(longp));
     pn2=(int)(num_pointsn*ran1(longp));
     while (pn1==pn2) pn2=(int)(num_pointsn*ran1(longp));
     if (fabs(hypot(P_new[pn2].y-P_new[pn1].y,
         P_new[pn2].x-P_new[pn1].x)-
         hypot(P_ref[pr2].y-P_ref[pr1].y,
         P_ref[pr2].x-P_ref[pr1].x))<WDIST) break;
   }
 //if (fabs(hypot(P_new[pn2].y-P_new[pn1].y,
 //    P_new[pn2].x-P_new[pn1].x)-
 //hypot(P_ref[pr2].y-P_ref[pr1].y,P_ref[pr2].x-P_ref[pr1].x))
 //    >WDIST) continue;

 //cout<<rand()<<endl;
 //pr1=330; pr2=293; pn1=47; pn2=33; cout<<pr1<<" "<<pr2<<" "
 //    <<pn1<<" "<<pn2<<endl;

//
// Projecting the reference (global) map according to the
// features
//

 //prth=atan2(P_ref.y[pr2]-P_ref.y[pr1],
 //    P_ref.x[pr2]-P_ref.x[pr1]);
```

```
 //pnth=atan2(P_new.y[pr2]-P_new.y[pr1],
 //    P_new.x[pr2]-P_new.x[pr1]);
 th=atan2(P_new[pn2].y-P_new[pn1].y,P_new[pn2].x-P_new[pn1].x)-
     atan2(P_ref[pr2].y-P_ref[pr1].y,P_ref[pr2].x-P_ref[pr1].x);


 R[0][0]=cos(th);R[0][1]=-sin(th);
 R[1][0]=-R[0][1];R[1][1]=R[0][0];

 T[0]= (-R[0][0]*P_ref[pr1].x-R[0][1]*P_ref[pr1].y)+P_new[pn1].x;

 T[1]= (-R[1][0]*P_ref[pr1].x-R[1][1]*P_ref[pr1].y)+P_new[pn1].y;
 //printf("th: %g\n",th);
 //printf("R: %g %g %g %g\n",R[0][0],R[0][1],R[1][0],R[1][1]);
 //printf("T: %g %g\n",T[0],T[1]);
 for(i=0;i<num_pointsr;i++) {
   P_pj[i].x=R[0][0]*P_ref[i].x+R[0][1]*P_ref[i].y+T[0];
   P_pj[i].y=R[1][0]*P_ref[i].x+R[1][1]*P_ref[i].y+T[1];

   //fitmp=atan2(P_ref.y[i]-P_ref.y(pr1),
   //    P_ref.x[i]-P_ref.x[pr1]);
   //ropj=sqrt(pow(P_ref.x[i]-P_ref.x[pr1],2)
   //    +pow(P_ref.y[i]-P_ref.y(pr1)));
   P_ox[i].x=P_pj[i].x;
   P_ox[i].y=P_pj[i].y;
   P_oy[i].x=P_pj[i].x;
   P_oy[i].y=P_pj[i].y;
  }

//
// Sorting the projected points according to their x-cordinate
//

 funkp=&compx;
 qsort(P_ox,num_pointsr,sizeof(*P_pj),*funkp);

 /*
 fptr_3 = fopen("i3.dat","w");
 for (i=0;i<num_pointsr;i++) {
     fprintf(fptr_3,"%g %g\n", P_ox[i].x, P_ox[i].y);
  }
 fclose(fptr_3);

 fptr_4 = fopen("i4.dat","w");
 for (i=0;i<num_pointsn;i++) {
     fprintf(fptr_4,"%g %g\n", P_new[i].x, P_new[i].y);
  }
 fclose(fptr_4);
 */
```

```
//
// Createing sercharray
//

 offsetx=-MAXRANGE/2;
 if (P_ox[num_pointsr-1].x>MAXRANGE/2 | P_ox[0].x<-MAXRANGE/2 ) {
     cerr<<"Error: echos not within MAXRANGE"<<endl;
     exit(-1);
  }
 i1=0;disto=MAXRANGE;
 //cout<<"P_ox[0].x: "<<P_ox[0].x<<" offsetx: "<<offsetx<<endl;

 for(x=0;x<MAXRANGE;x++) {
 //for(x=100;x<101;x++) {
     while (1) {
        dist=fabs(P_ox[i1].x-x-offsetx);
        //cout<<i1<<" P_ox["<<i1<<"].x "<<P_ox[i1].x<<" "<<
        //    x+offsetx<<" disto "<<disto<<" dist "<<dist<<endl;

        if (dist<=disto  & i1<num_pointsr) {disto=dist; i1++;}
        else {closeindx[x]=i1-1;break;}

       }
     //if (x<150) { cout<<"x+offsetx: "<<x+offsetx<<" closeindx:
     //"<<closeindx[x]<<" disto: "<<disto<<" dist:"<<dist<<
     //    " P_ox[.].x: "<<P_ox[closeindx[x]].x<<endl;
     //}
     i1--;
     disto=MAXRANGE;
  }
 //for(x=0;x<MAXRANGE;x++) {cout<<closeindx[x]<<" ";}



 //cout<<"offsetx "<<offsetx<<" closeindx ";
 //for(i=0;i<MAXRANGE;i++)
 //    cout<<i+offsetx<<":"<<closeindx[i]<<" ";
 //cout<<endl;

 //
 // for all points in new map (local map) the closest reference
 // point is found
 //

 E=0;
 pts=0;

 for(i=0;i<num_pointsn;i++) {
 //for(i=42;i<43;i++) {
     shortest[i]=MAXRANGE;
```

```
    shind[i]=-1;
    tmpind=closeindx[(int)rint(P_new[i].x)-offsetx];
    //cout<<i<<" P_new["<<i<<"].x "<<P_new[i].x<<" tmpind "
    //    <<tmpind<<" rint(P_new["<<i<<"].x) "<<rint(P_new[i].x)
    //    <<endl;
    downch=0;

    while (1) {
        tmpxl=P_ox[tmpind].x-P_new[i].x;
        //cout<<"P_ox["<<tmpind<<"].x "<<P_ox[tmpind].x<<
        //    " P_new "<<P_new[i].x<<" tmpxl "<<tmpxl<<endl;
        if ((tmpxla=fabs(tmpxl))<WDIST) {
            if ((tmpyl=fabs(P_ox[tmpind].y-P_new[i].y))<WDIST) {
                //cout<<"tmpdist "<<
                //   sqrt(tmpxl*tmpxl+tmpyl*tmpyl)<<endl;
                if ((tmpdist=sqrt(tmpxl*tmpxl+tmpyl*tmpyl))<
                    shortest[i] & tmpdist<WDIST) {
                    shortest[i]=tmpdist;
                    shind[i]=tmpind;
                 }
             }
         }
        if (tmpxl<-WDIST | tmpind==0 |tmpxl<-shortest[i]) {
            downch=1;
            tmpind=closeindx[(int)rint(P_new[i].x)-offsetx];
         }
        if (tmpxl>WDIST | tmpind==num_pointsr-1 |
            tmpxl>shortest[i]) {
            if (shind[i]!=-1) {
                E+=shortest[i]*shortest[i];
                pts++;
             }
            break;
         }
        if (downch) tmpind++; else tmpind--;
      }
    //cout<<"i "<<i<<" shind "<<shind[i]<<" shortest "<<
    //    shortest[i]<<endl;
  }

 //cout<<endl;
 //for(i=0;i<num_pointsn;i++) cout<<shind[i]<<" "; cout<<endl;
 //for(i=0;i<num_pointsn;i++) cout<<shortest[i]<<" ";cout<<endl;

 //cout<<"E "<<E<<" pts "<<pts<<endl;

//
// Writing the features used and two values which are used to
// select best match
//
```

```
 ofile<<pr1<<" "<<pr2<<" "<<pn1<<" "<<pn2<<" "<<E<<" "<<pts<<
     endl;

 }
}

/*****************************************************************/
//
// compare function, used to create search array of projected
// points
//
/*****************************************************************/
int compx(const void *p1, const void *p2) {
    if ((*(point *)p1).x>(*(point *)p2).x) return 1;
    if ((*(point *)p1).x<(*(point *)p2).x) return -1;
    return 0;
 }

/*****************************************************************/
//
// Random number generator
//
/*****************************************************************/
#define IA 16807
#define IM 2147483647
#define AM (1.0/IM)
#define IQ 127773
#define IR 2836
#define NTAB 32
#define NDIV (1+(IM-1)/NTAB)
#define EPS 1.2e-7
#define RNMX (1.0-EPS)
float ran1(long *idum)
/*Minimal" random number generator of Park and Miller with
Bays-Durham shu e and added safeguards. Returns a uniform random
deviate between 0.0 and 1.0 (exclusive of the endpoint values).
Call with idum a negative integer to initialize; thereafter, do
not alter idum between successive deviates in a sequence. RNMX
should approximate the largest foating value that is less than 1.
*/

{
int j;
long k;
static long iy=0;
static long iv[NTAB];
float temp;

if (*idum <= 0 || !iy) {             //Initialize.
```

```
    if (-(*idum) < 1) *idum=1;     //Be sure to prevent idum = 0.
    else *idum = -(*idum);
    for (j=NTAB+7;j>=0;j--) {
        //Load the shuffle table (after 8 warm-ups).
        k=(*idum)/IQ;
        *idum=IA*(*idum-k*IQ)-IR*k;
        if (*idum < 0) *idum += IM;
        if (j < NTAB) iv[j] = *idum;
     }
    iy=iv[0];
 }
k=(*idum)/IQ;                  //Start here when not initializing.
*idum=IA*(*idum-k*IQ)-IR*k;
    //Compute idum=(IA*idum) % IM without over- ows by Schrage's
    //method.

if (*idum < 0) *idum += IM;
j=iy/NDIV;                      //Will be in the range 0..NTAB-1.
iy=iv[j];
    //Output previously stored value and re ll the shu e table.

iv[j] = *idum;
if ((temp=AM*iy) > RNMX) return RNMX;
    //Because users don't expect endpoint values.

else return temp;
}


/*************************************************************/
//
// Reads in the data
//
/*************************************************************/
void read_data(int argc, char *argv[])
{
 int i;
 //FILE *fptr_3, *fptr_4;

                                              //store odometry data
 //odometry_x = (float) atof(argv[3]);
 //odometry_y = (float) atof(argv[4]);
 //odometry_rotation = 2 * PI *
 //    ((float) atof(argv[5]) / (float) 360);//in radians

 odometry_x = (float) 0;
 odometry_y = (float) 0;
 odometry_rotation = 0;//in radians

 fptr_1 = fopen(argv[1],"r");
```

```
 fptr_2 = fopen(argv[2],"r");               //read in (x,y) data


 //fptr_3 = fopen("i3.dat","w");
 //fptr_4 = fopen("i4.dat","w");

 for (i=0; i<num_pointsr; i++) {
  fscanf(fptr_1,"%g %g", &P_ref[i].x, &P_ref[i].y);

  }
 for (i=0; i<num_pointsn; i++) {
  fscanf(fptr_2,"%g %g", &P_new[i].x, &P_new[i].y);
  }


 //cout<<"xNew: "<<P_new[i].x<<" yNew: "<<P_new[i].y<<endl;
 //exit(0);

 //fprintf(fptr_3,"%g %g\n",
 //    P_ref_polar[i].theta, P_ref_polar[i].R);
 //fprintf(fptr_4,"%g %g\n",
 //    P_new_polar[i].theta, P_new_polar[i].R);


 fclose(fptr_1);
 fclose(fptr_2);
 //fclose(fptr_3);
 //fclose(fptr_4);

}
```

A.11 pcleaner.m

```
function X=pcleaner(a,c)
%
% Plots a poolcleaner at the pose a, with the color c.
%

x=[.35 .35 -.55 -.55 .35   .20 .20 -.1 -.1 .2];
y=[-.25 .25 .25 -.25 -.25   -.2 .2 .2 -.2 -.2];

th=a(3);
```

```
R=[cos(th) -sin(th); sin(th) cos(th)];
T=a(1:2);

tmp=R*[x;y]+repmat(T,1,size(x,2));
xpj=tmp(1,:);
ypj=tmp(2,:);
plot(xpj(1:5),ypj(1:5),c);
plot(xpj(6:10),ypj(6:10),c);
```

A.12 onemap.m

```
%
% This script merges the maps
%

%
% Project the points of the new map to the cordinate system of
% the reference map.
%

th=pos_v(3,is);
R=[cos(th) -sin(th); sin(th) cos(th)];
T=pos_v(1:2,is);
%[xtmp,ytmp]=pol2cart(svV(is).fi,svV(is).ro);
[xtmp,ytmp]=pol2cart(fiNew,roNew);
Ppj=R*[xtmp;ytmp]+repmat(T,1,length(xtmp));
xpj=Ppj(1,:);
ypj=Ppj(2,:);

ind1=find(Umap.l==1);
rm=[];
for i=1:length(ind1)
    indtmp=find(sqrt((Umap.x(ind1(i))-xpj).^2+ ...
       (Umap.y(ind1(i))-ypj).^2)<.75);
    if ~isempty(indtmp) Umap.l(ind1(i))=2;
    %else rm=[rm ind1(i)];
    end;
end;

%hold off;plot(Umap.x,Umap.y,'b.');hold on
%plot(xpj,ypj,'c.');
%axis('equal');axis(Caxis);hold off;waitforbuttonpress

rm=[];
rmU=[];
for i=1:is-1
    n=(pos_v(1:2,is)-pos_v(1:2,i))/2;
    pt=pos_v(1:2,i)+n;
```

```
    lev=n'*pt;

    rmUtmp=find(n'*[Umap.x ;Umap.y] >lev &...
        Umap.sc==i);
    rmtmp=find(n'*[xpj ;ypj] <=lev);
    rmU=[rmU rmUtmp];
    rm=[rm rmtmp];
end;
roNewtmp=roNew;
%Umap.x(rmU)=[];Umap.y(rmU)=[];Umap.l(rmU)=[];
%Umap.r(rmU)=[];Umap.sc(rmU)=[];
%xpj(rm)=[];ypj(rm)=[];roNewtmp(rm)=[];

%
% Merge the maps
%

Umap.x=[Umap.x xpj];Umap.y=[Umap.y ypj];
Umap.l=[Umap.l ones(size(xpj))];
Umap.r=[Umap.r roNewtmp];Umap.sc=[Umap.sc is*ones(size(xpj))];

%
% Resort the map.
%

[tmpfi,tmpro]=cart2pol(Umap.x,Umap.y);
[tmpfi,tmpi]=sort(tmpfi);
tmpro=tmpro(tmpi);
[Umap.x,Umap.y]=pol2cart(tmpfi,tmpro);
Umap.l=Umap.l(tmpi);
Umap.r=Umap.r(tmpi);
Umap.sc=Umap.sc(tmpi);
```