Autonomous pool cleaning

G. Zunino[1] M. Simoncelli[2]

Centre for Autonomous Systems
Kungliga Tekniska Högskolan
Stockholm, Sweden
January 1999

[1]University of Genoa, Italy.
[2]University of Genoa, Italy.

Contents

List of Figures

List of Tables

Acknowledgments

Many thanks to Henrik Christensen, professor at NADA/CVAP department, for giving us the possibility of joining the KTH and for his support and advice in the development of this project; to the NADA/CVAP staff for their assistance in the development of the program; to WEDA, who have contributed by suggestions and ideas, especially to Klas Lange for his availability and inspiration, to Anders Ekenbäck for his support and assistance during our tests.

Abstract

This thesis describes six months of studies and practical works to develop an autonomous pool cleaning robot. The report is divided into three parts which contain respectively: the system description: robot and sonar system; the theoretical work: considerations and studies about the physical model of the robot, with the relevant simplifications, and about the input data processing; the experimental work: considerations coming up from the several tests performed, control software implementation and the final results.

Chapter 1

Introduction

In this thesis the purpose is to create a sensor-based navigation system in order to make an underwater pool cleaning robot able to perform cleaning in an autonomous way. There are two major steps in this project. The former is a theoretical study and a practical work regarding the problem of a regular swimming pool (with a rectangular shape). The latter is a consideration on the problem of non-regular swimming pools. Only the first step will be considered in this work.

The project is set up by a cooperation between the Royal Institute of Technology (KTH-Stockholm) and WEDA PoolCleaner. WEDA is a company that produces manual and autonomous pool cleaners for daily cleaning of swimming pools. It is located in Södertälje, near Stockholm, and it is one of the three major suppliers of pool cleaners in the world.

1.1 Problem

A problem connected with swimming pools is the one of cleaning. It is necessary to maintain water on a high cleanness level, especially from the hygienic point of view. Anyway this is not all: it is also necessary to consider various types of deposits on the bottom and on the walls of the pool. This cannot be solved by exchanging water or adding chemicals.

The level and the type of dirtiness in a pool depends, generally, on the way the swimming pool is used, and also on the surrounding all around it. It is intuitive there is a huge difference between an in-door pool and an out-door one.

The problem we are going to consider in this project deals the cleaning of the bottom of the pool, both for the out-door and the in-door case.

1.2 Todays Robot

Nowadays there are many products on the world market, both manual and automatic, capable of cleaning the bottom and the walls of the pools. The basic concept used by the greater part of these machines is similar to the one used by a common vacuum cleaner. A pump sucks up water, and dirt particles, which are picked up and collected by a filter bag.

There are many machines manually piloted, such us with a long stick, and with the use of a remote control, connected with a long cable. The company WEDA constructs both types of pool cleaners for commercial pools. The automatic machines can perform an automatic cleaning of rectangular pools; but some problems can occur: one first consideration is about the correction of slippage of the machine in the pool. Another consideration is about the fact that most newer pools have a much more complicated shape. All this call for more advanced techniques, including automatic mapping of the pool and subsequent planning of a cleaning pattern.

1.3 The project

1.3.1 Problem specification

The purpose of this joint project is to create an autonomous system able to clean the bottom of a swimming pool without requiring either a guide to follow or teleoperator control. The objective is a machine which cleans the pool in the smallest possible time and in a satisfactory way (e.g. hygienic speaking), and with a limitation in price, since it has to be a product to sell.

As the work environment is totally static, it could be possible to have a pre-stored map of the world. But this could be useful only if all the swimming pools had both the same shape and the same dimensions. In reality there are lots of different types of pools (i.e. the private ones). Therefore sensing is needed, especially with non-regular pools.

With the mapping (and sensing) system it is then possible to perform automatic correction of slippage in the pool and it is also possible to perform more advanced cleaning.

In general there are three steps involved in the process of creating knowledge about the environment: data acquisition, interpretation and fusion of data from different sensory systems. The first step suffers from problems with accurate measurements, e.g. in the case with the sonar multiple reflections or cross-talk may cause problems. The last two steps are most certainly non-trivial since the interpretation of the sensory data needs some higher level understanding of how the sensors react to different environmental configurations. There could also be a need to have a more accurate definition of what kind of information is required and what should be discarded.

1.3.2 Solutions

At this preliminary stage, there are many possible solutions as far as the problem of the sensors and of the control strategy. The sensor issue gives rise to a wide discussion: there are a lot of sensors on the market, which are able to get a kind of information useful to different applications [3], e.g. optical transducers and sensors, inductive and capacitive transducers, ultrasound transducers, electrical micro switch, etc.

Before to study all the possibilities we can have, it is necessary to consider the work conditions and the work environment. The robot works always in an underwater environment, in which the visibility conditions are not the best, and the chemical composition of the water may represent a problem to an electrical instrument. The temperature varies from pool to pool, e.g. it can be different from an in-door pool to an out-door one (during the cleaning section the temperature of the water is not always controlled). Another important point is the final price of the machine. The cost of the definitive product should be as low as possible, since the pool cleaner will be sold, and it should be competitive on the whole world market.

It is now clearer that optical sensors like cameras and laser ranging devices are not recommended to this application: the visibility in the water sometimes requires use of artificial light; and in general optical systems are rather expensive and they are beyond the cost of this project. Inductive sensors require a magnetic field at the edges, to enable the sensor to detect the field and to indicate a change in the pattern. That can be arranged by placing magnetic bars around the pool, which may not appear very practical in every situation. The magnetic field is also disturbed by other magnetic sources, like magnets in pumps, etc. We can also have capacitive sensors, which require that the sensor is effected by a medium, of at least a specific strength, so that a change in the pressure admits a change to the sensor. The transducer reacts to all kinds of hard enough materials in its way. If the water render this pressure, water can cause that change. Ultrasound transducers are common in industrial applications, among other things for non-destructive testing. They are even applicable in presence of smut and dirtiness. Some ultrasound measurement systems require enclosing for use in water, to make them waterproof and, considering that air pressure under water varies by depth, insensitive to change in pressure. Electrical micro switches was among other options that were considered. A micro switch requires physical contact between the transducer and the edge.

Interesting and more accurate alternatives were the use of GPS, IR (infrared equipment), laser equipment, compass, gyro and sonar. Most of these options do not apply well in practice. GPS operates in water not deeper than few decimeters, and often swimming pools are in an in-door environment; that is GPS is not very good for this application. IR is an optical sensor widely used in industrial applications. However it has a range of a few me-

ters and it cannot be anticipated to act in a satisfactory way for this work. Laser faces the problem of attenuation under water, which implies that we can only use it with relative short ranges or we can use a powerful laser, that is much too expensive to be employed. As far as the use of an electronic compass, magnetic sources that may be all around the pool easily disturb it. And anyway its precision may not always be enough for an application in a little swimming pool. Sonar is frequently used in marine measurements and has been so for quite some time. It is the sensor type that is least sensitive to interference from particles and smut in the water. Submarines do apply this technique for measurement and navigation in the ocean. Another application for sonar is in fishing boats, both for depth measurements and to detect shoals of fishes. Some advanced kinds, i.e. multi-lobe sonar with several rays, are also used in forward measurement in the direction of the boats. An important advantage of this sensor is that there are simple kinds of sonars (i.e. sonar used in fishing boats) which are also cheap.

With regard to the control strategy, it depends on the particular problem we are dealing with. It is rather simple, from a conceptual point of view, for the cleaning of a rectangular and quite regular swimming pool. On the other hand it could be arduous and complex for a pool with a general shape. For this kind of problem it is first of all difficult to complete the map of the environment in a satisfactory way. It is intuitive to understand the difficulties which may appear in a strange pool, e.g. a non planar bottom, an island in the middle of the pool, circular or various shaped walls, etc.

1.3.3 Our work

Making a reference to the regular pool problem, as far as the sensors, it has been used the easier way. Since the subject to be measured is the distance in the pool, from the machine to the edge, the sonar system was chosen for this application. If a new sensor with specific characteristic was to be developed, costs would increase. Even though sonar may not be the best sensor available on the market, it has some advantages: as we noted before, it is very inexpensive compared to other range sensing equipment; the interpretation is fairly simple in contradiction to vision systems and the amount of data to analyze is small; the hardware needed is also easy to operate from a computer through a standard interface.

The particular system chosen is a simple kind of sonar used in fishing boats, which consists of a transducer and a receiver. Considerations about the sonar will follow in the sonar chapter.

The machine is also supplied by a couple of bumpers with electrical micro switches, which are activated by physical contact with an edge (a wall).

Another important point in this application is the filtering of data from the sonar system. It is necessary to avoid problems of disturbance of various type. The control strategy was set up in such a way to follow a pre-calculated

navigation pattern. A scheme of the complete system is represented in the figure 1.1.

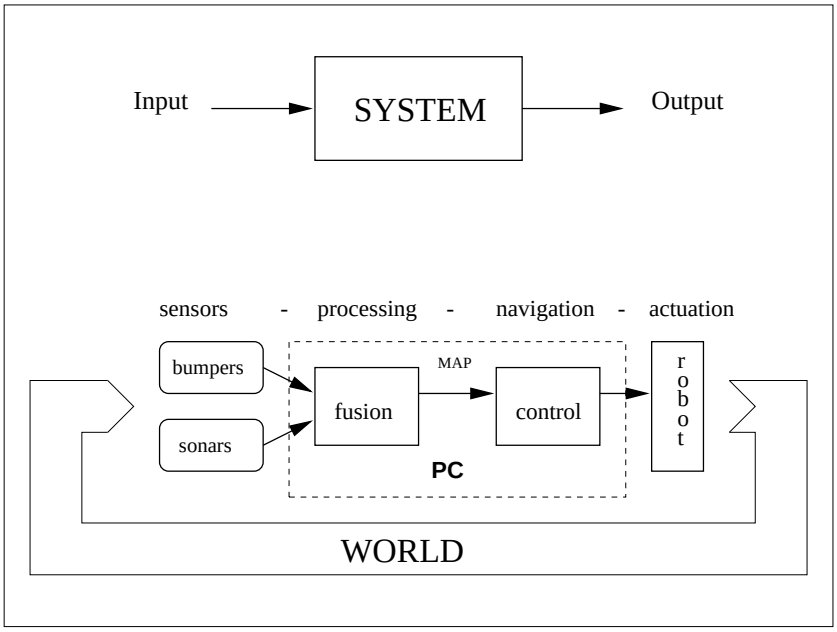

Figure 1.1: System scheme.

This project is composed of three major parts. The first one is to describe the whole system, to understand the way it works and in which way the measurements made by the sonar can be used. The second step is to create a complete control strategy which makes the pool cleaner able to perform autonomous cleaning of pools with a rectangular shape. The last part is a more interesting and at the same time complex work about non-regular pools. The experimental work will be concentrated on the first (in order to understand better the robot and its behavior in the work environment) and the second steps only, since the third one needs a greater amount of time than we have. About this it is also useful to consider the use of other sensors in addiction to sonar, because of the incapability of the pool cleaner to make a detailed map of a generic non-regular pool (sonar is useful but one must not expect more from it than it physically can provide) and in order to be sure about the right way it has to follow (there is not a feedback on the motion of the trucks). This consideration is not definitive but gives reason for further discussions.

1.3.4 Outline of the thesis

- Part One: The Pool-Cleaner.

  This part describes the system, the sonar and gives some basic information about the characterization of propagation of sound in an underwater environment.

- Part two: Theoretical work.

  The theory of the system is here presented, and a dynamic model of the pool-cleaner is studied. This part describes also the theory of the control strategies we use, and a simulation part regarding the behavior of the robot in the work-section is presented too.

- Part three: Experimental work.

  In this part we present the software routines with a brief description about the cleaning strategy. Some results about sonar tests are given, together with an evaluation on the real steering angle. After that the implementation of the software is presented, with some results about cleaning sections.

Part I

The Pool-Cleaner

Chapter 2

The robot

2.1 Introduction

In the following pages we will give a brief presentation of the system, both the machine and the sonar. First we present a description of the pool cleaner as it is at present on the market, and in which way it works.

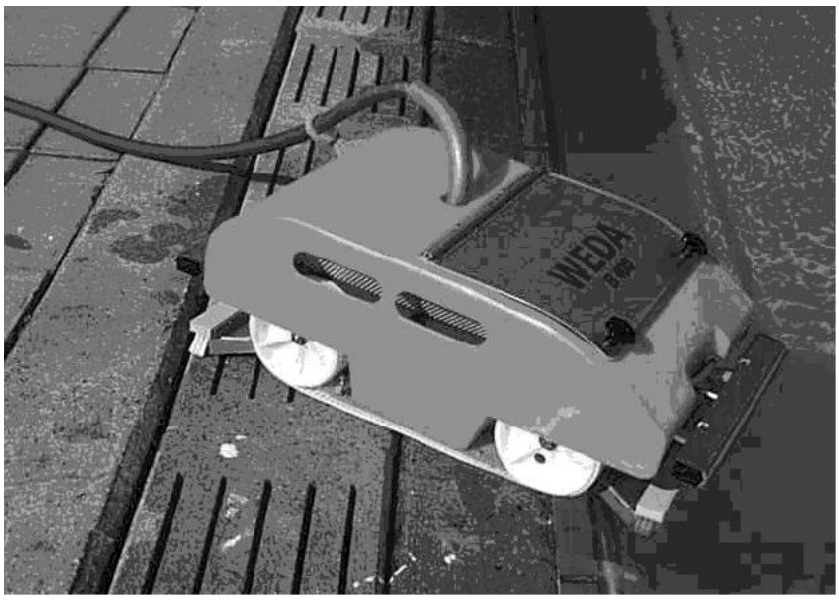

Figure 2.1: WEDA B400 Pool-cleaner.

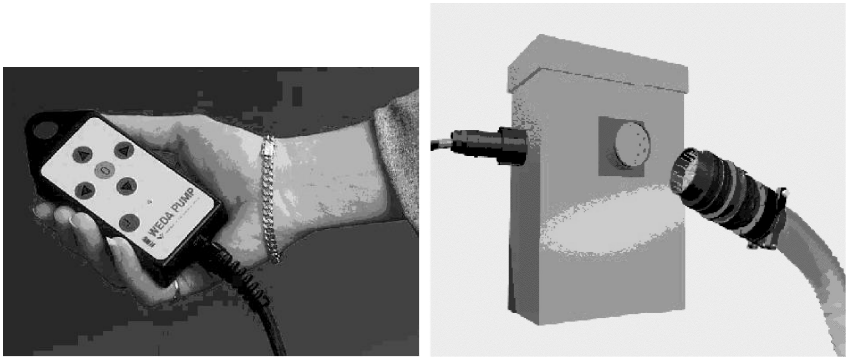

Figure 2.2: WEDA B400 accessories.

2.2 WEDA B400

WEDA B400 is the name of the commercial product, which is already used worldwide. This automatic pool cleaner has on-board pump with an associated filter bag through which water is passed. A set of motorized brushes picks up algae and smut, which then are picked up and collected by a filter bag. To ensure cleaning of the entire swimming pool the cleaner performs a lawn mover type of sweep of the pool.

When the machine is to be applied, normally it starts at one end of the pool, along with the long side and starts moving toward the opposite end. There are two bumpers fastened to each end. By approaching the ends, the bumpers send signals, which change the direction of the motion. One bumper is attached straight, to go in a straight path, while the other is slightly set at an angle, in such a way to enable the vehicle to move along a skew path. We can think to this motion as to a zig-zag pattern.

The machine is connected to the power-supply by a long floating cable, which also includes cables for the signals coming from the bumpers, and for the motion control. The pool cleaner is equipped with a remote control for easy control of cleaning hard-to-reach areas or high speed cleaning. The pictures in the figure 2.2 illustrate the floating cable connection and the remote control.

The WEDA B400 has the following dimensions: the height is 30 cm, the width is 48 cm and the length 90 cm. Its weight in air is 35 kg, and it is about 12 kg in water. It can move with a speed of 0.3 m/sec.

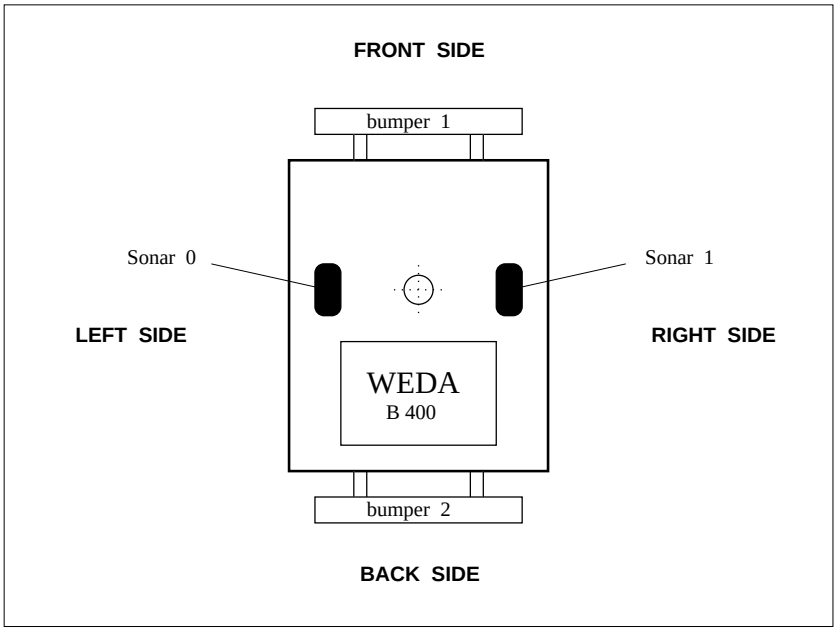

Figure 2.3: WEDA B400 scheme.

2.3 The scheme of the robot

It is now useful to have some assumptions on the whole system, in order to facilitate the work. We will consider here the sonar too, even if we describe it in next chapter; in the final system there is a couple of sonars, which are fastened at the left and right sides of the machine. In this way our assumptions are as in the figure 2.3.

Such a kind of characterization is needed because of the necessity to have in each instant a clear vision of the working condition of the robot from and for the software. It is needed, for example, to know from which bumper a signal comes, since there are two different kinds of motion regarding the different way the robot covers.

Chapter 3

The sonar system

3.1 Introduction

Here a description of the sonar system is given. A set of reasons why sonar has been chosen for this application were listed before. Now, since what we do is just to use a system which is constructed in order to work in the sea, in a completely different environment (e.g. the swimming pool), it is useful to try to understand under which kind of conditions the measures we get are reliable. For this purpose it will be useful to consider something of the propagation of sound in the water. In the last section of this chapter we will describe our results.

3.2 Principles of underwater sound

Of all the forms of radiation known, sound travels through the water the best. In the water both light and radio waves are attenuated much more than that form of mechanical energy known as sound.

Sound consists of a regular motion of the molecules of an elastic substance. Because the material is elastic, a motion of the particles of the material communicates to adjacent particles. A sound way is thereby propagated outward from the source at a velocity equal to the velocity of sound.

The key factors which describe the propagation, physically, are particle velocity and the local pressure, which is responsible for creating the particle displacement. A mathematical study of the physics of sound leads to the formulation of "wave equations", which are differential equations interrelating particle velocity and pressure. These equations incorporate as a "constant of proportionality" a quantity that determines the rate at which a disturbance propagates through the medium. This quantity is the speed of sound and is an important characteristic of all physical media which sustain sound propagation and of all engineering material used in equipments for generation or detection of sound.

3.2.1 Sound speed

The speed of sound, c is determined [2] by three physical properties of the medium, namely its specific temperature at constant pressure, γ, its density, ρ and its isothermal bulk modulus of elasticity, B. The inter-relationship between these quantities is given by the following equation, which is attributed at Newton:

$$c = \sqrt{\frac{\gamma B}{\rho}} \tag{3.1}$$

In distilled water at 20^oC and at standard atmospheric pressure, the physicist measures γ= 1.004 and ρ=998Kgm^{-3}. We thus calculate the speed of sound to be $1481ms^{-1}$. In practise, constraints in measurement accuracy of γ, B and ρ limit the value of this equation. In fact it is sufficient to note that all three variables are quantities which depend upon temperature T, pressure P and chemical composition.Consequently, it has been argued that the speed of sound may be expressed as some suitable function of temperature, pressure, and chemical composition.

$$c = f(T, P, C) \tag{3.2}$$

As the present project is carried out in a controlled environment it is possible to consider the pressure as a constant, and the water temperature restricted to two values according to the fact the pool is indoor or outdoor. Early measurements on distilled water at atmospheric pressure performed at the US Bureau of standards were used to produce the fourth-order polynomial

$$\begin{aligned} c &= 1402.736 + T(5.03358 + T(-0.0579506 + T(3.31636 \cdot 10^{-4} + \\ &\quad T(-1.45262 \cdot 10^{-6} + 3.0449 \cdot 10^{-9})))) \end{aligned} \tag{3.3}$$

where T is in degrees Celsius.

3.2.2 Propagation theory

The propagation of sound in an elastic medium can be described mathematically by solutions of the wave equation using the appropriate boundary and medium conditions for a particular problem. The wave equation is a partial differential equation relating the acoustic pressure p to the coordinates x, y, z and time t, and may be written as

$$\frac{\partial^2 p}{\partial t^2} = c^2 \left(\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} + \frac{\partial^2 p}{\partial z^2} \right) \tag{3.4}$$

where c, as we saw before, is the sound velocity.

There are two theoretical approaches to a solution of the wave equation. One is called normal-mode theory, in which the propagation is described in terms of characteristic functions called normal modes, each of which is a solution of the equation. The normal modes are combined additively to satisfy the boundary and source conditions of interest; the result is a complicated mathematical function which gives information on the distribution of the energy of the source in space and time. The other form of solution is the ray theory, and the body of results and conclusions are called ray acoustics. The essence of ray theory is the postulate of way-fronts [1], along which the phase or time function of the solution is constant, and the existence of rays that describe where in space the sound emanating from the source is being sent. Like its analog in optics, ray acoustics presents a picture of the propagation in the form of the ray diagram. The normal-mode theory gives little insight compared to the ray one, but ray theory has certain shortcomings and does not provide a good solution under conditions where the radius of curvature of the rays or the pressure amplitude changes appreciably over the distance of one wavelength [1]. Even if the former gives a formally complete solution and it is valid at all frequencies, by using the latter, sound distribution is easily visualized. Considering, for the sake of simplicity, the ray theory, we can see something about reflection [1].

Specular reflection. For objects of radii of curvature large compared to a wavelength, the echo originates principally by specular reflection, in which those portions of the target in the neighborhood of the point at which sound is normally incident give rise to a coherent reflected echo. We can see in the figure 3.1 some easy geometry of specular reflection.

In the figure we see a convex target of large radius of curvature, and a regular one: targets with sound normally incident and not. The concave target is a similar case. The pool-edge problem could be led back to these cases.

3.2.3 Sound in shallow water

Generally, in the underwater sound propagation context, shallow means a water depth in which sound is propagated to a distance by repeated reflections from both surface and bottom. In water that is acoustically shadow, the acoustic characteristic of both surface and bottom are important determinants of the sound field. Acoustically speaking, shallow water may be said to mean propagation to distances at least several times the water depth, under conditions where both boundaries have an effect on transmission. Water that is shallow forms a sound channel between the surface and the bottom, in which sound is trapped between the upper and the lower channel boundaries.

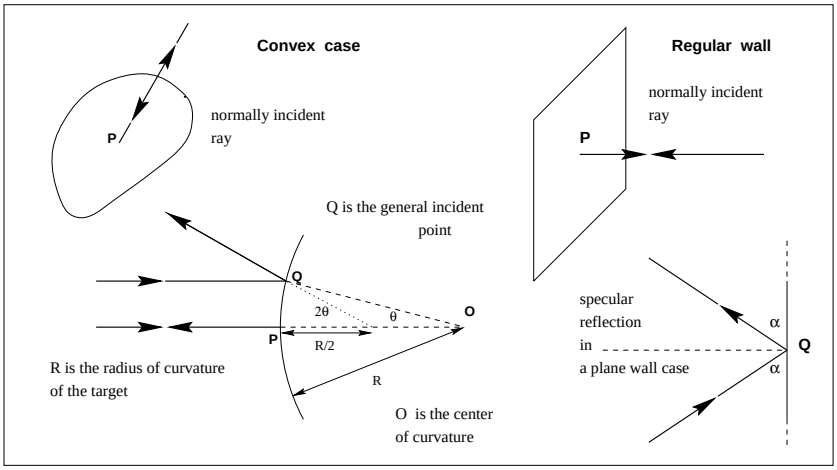

Figure 3.1: Sound wave reflection.

We use the ray theory here too; the normal-mode theory is more appropriate at long ranges [1] because of the greater attenuation with distance. Instead, ray theory is more convenient to use at short ranges, and it may work for the pool problem. It represents the sound field as a sum of ray contributions, each ray emanating from the source or its image in the surface and the bottom. There is a process by which an infinite string of images is built up, with higher-order images tending to became insignificant because of weakening of repeated reflections and by greater distance. Therefore, at short ranges the higher-order images rapidly die out because of reflection losses and an increasingly great distance from the field point. Accordingly, only a few images need ordinarily to be summed at short ranges.

3.2.4 Basic considerations on sonars

In a fluid, the particle motion is to-and-from, parallel to the direction of propagation; because the fluid is compressible, this motion causes changes in pressure which can be detected by a pressure-sensitive hydrophone. These uses of underwater sound constitute the engineering science of sonar, and the system employing underwater sound in one way or another are sonar systems.

Sonar systems are said to be active when sound is purposely generated by one of the system components called the projector. The sound waves generated by the projector travel through the water to the target (the edge of the pool in our case), and are returned as sonar echoes to a hydrophone,

which converts sound into electricity. Active sonar systems are said to echo-range on their targets. Instead passive or listening sonar systems use sound radiated by the target; here only one-way transmission through the water is involved, but it is not our case.

Another consideration is about monostatic and bistatic sonars. In the monostatic case the source and the receiver are coincident, and the acoustic return of the target is back toward the source; instead, in the bistatic case a separated source and receiver are employed. As we will see, we use a monostatic system.

The many phenomena peculiar to underwater sound produce a variety of quantitative effects on the sonar equipment; these diverse effects can be conveniently and logically grouped together in a small number of units called the sonar parameters, which, in turn, are related by the sonar equations. These equations are the working relationships that tie together the effects of the medium, of the target and of the equipment. The equations are founded on a basic equality between the desired and undesired portions of the received signal at the instant when some function of the sonar set is just performed. This function involves the reception of acoustic energy occurring in a natural acoustic background. Of the total acoustic field at the receiver, a portion may be said to be desired and is called the signal. The remainder of the acoustic field is undesired and may be called the background.

In sonar the background is either noise, the essentially steady state portion not due to one's own echo ranging, or reverberation, the slowly decaying portion of the background representing the return of one's own acoustic output by scatters in the water. The aim is obviously to have a good signal-to-background ratio.

3.3 Navman Echo-Sound Sonar

To measure the distance between the robot and the pool's wall it has been chosen to use a NAVMAN 100 sonar. This sonar is projected for marine application to display the sea depth. The NAVMAN 100 display is showed in the figure 3.2.

This instrument is linked through a coaxial cable with the sonar transducer and it can be connected to a personal computer through a serial port. The protocol used is the NMEA standard. It is an uniform interface standard for digital data exchange between electronic devices. The National Marine Electronics Association has developed a standardized data protocol that permits high speed communications among shipboard electronic devices. The distance range is from 2 to 150 meters and for distances shorter than 20 meters the resolution is 10 centimeters. The transducer used is the 2" IN-HULL PUCK TRANSDUCER, it has an open angle of 9 degrees and a working frequency of 200 kHz.

Figure 3.2: NAVMAN 100 Depth series.

3.4 Conclusions

Now, using a few principles of underwater sound propagation, it will be useful to have some consideration about the acoustic environment of the swimming pools.

As a matter of fact, the problem in the pool is easier to deal with and more restricted than the one in the sea. It is enough to consider the regularity of the walls as a target, compared to the shape of the bottom of the sea for instance, or , even worse, to the shape of a general submerged target (i.e. military targets).

Moreover, as far as underwater sound propagation, there are lots of studies and results, especially because of the military research in this field, but we have much less about pools. Thus we will try to evaluate the possible problems we can have in a swimming pool, and to understand which are the physical restrictions of the sonar system used in this application.

3.4.1 Temperature problem

This problem regards the variation in the variable c, speed of sound, depending on a change in temperature. In the sea, different streams produce layers of water with different temperatures, and this may represent a source of noise in the sonar signal. In the pool we will not expect a big variation in temperature, since the global environment is in any case small, and the temperature is often controlled; some changes may appear in out-door pools, but they are restricted to a few tenths of degree. It is now clear the only variation in temperature we should put in evidence is the one there is

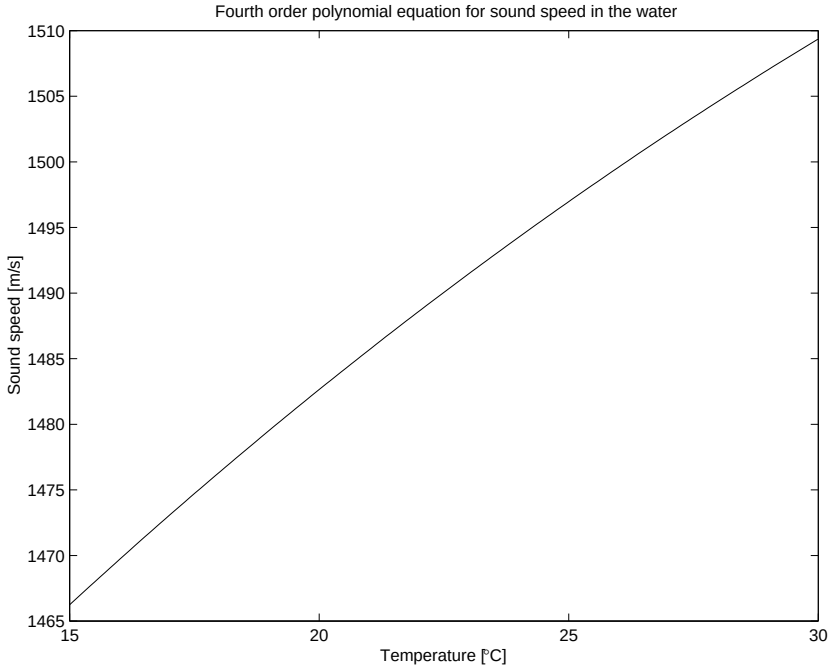

Figure 3.3: Sound speed-temperature relation.

among different pools. Even considering a 15 degrees variation, c changes a little. The velocity of sound increases with temperature and approximate coefficient for the rate of change [1] is

$$\frac{\Delta c}{\Delta T} = (5 \cdot 0.3048) \qquad m/(s)(^oF) \tag{3.5}$$

calculated for temperature near 70 degrees Fahrenheit, that is about 21 degrees Celsius. Considering a range between 15 and 30 (Celsius), e.g. between 59 and 86 degrees Fahrenheit, we find a variation in c of about 40 m/s. We can see even using (3.3) the difference may be around this value, not very much considering we work over short distances, that is we can neglect it; the graphic in figure 3.3 shows the polynomial function (3.3).

3.4.2 Water composition problem

Water composition may represent a problem because of the presence of particles, i.e. air bubbles. Free air bubbles are quite small, since the larger bubbles tend to rise quickly to the surface. Nevertheless, because air has a

markedly different density and compressibility than water, and because of the resonant characteristic of bubbles [1], the suspended air has a profound effect upon underwater sound. But even this fact is reduced to the minimum in a pool, especially because the environment is supposed to be almost static (the only motion is created by the robot itself).

3.4.3 Reflection and scattering by the edges

Reflections on the walls. The first consideration regarding the walls is about their physical characteristics. The walls are generally much more regular compared to the bottom of the sea, which uses to be the target of the sonar we use. Besides, the material which the walls are usually composed with presents acoustic characteristics quite different from the ones in water.

One second consideration regards which kind of reflection we may deal in this case; for the sake of simplicity, it is possible to use the specular reflection theory (3.2.2). The best condition in order to receive one signal with the higher signal-to-noise ratio is to use both the acoustic source and the transducer perpendicular to the wall. But during a work section, the machine often gets measures when it is not exactly perpendicular to the wall, instead it may have a little inclination angle. This fact does not represent a problem if the value of the inclination angle remains restricted to a few degrees, since the open angle of the sound beam generated by the sonar has a value of 9 degrees. In this application is much better to have a reduced open angle, but prices would grow up; the sonar we use needs a greater open angle (it has to detect distances from the bottom of the sea even when the surface is rough, and oscillations of the boat are not avoidable).

Reflections on the bottom and on the surface. Similar problems may occur for the bottom of the pool and the surface of the water. As far as the bottom, the case is practically the same of the one of the walls. Instead, as far as the surface of the water, it is both a reflector and a scatterer of sound, and has a profound effect on propagation [1]. If the water surface is perfectly smooth, it forms an almost perfect reflector of sound; the intensity of sound reflected is very nearly equal to that incident upon it.

Much more complicated is the situation if the water is rough, and the loss on reflection is found to be no longer zero, but this is not our case, because of the almost static condition of the water in the pool.

Shallow water problem. The considerations about the bottom and the surface call for the typical problem of shallow water. In a swimming pool, the depth is generally not larger than 3 meters, but the length may be of several meters, i.e. 25 or 50 meters. It means that contributions to the signal coming from images of the source both on the surface and on the bottom are really small for short distances, and may increase for long distances.

Experimental work have effectively confirmed these observations, putting in evidence growing difficulties in getting accurate measures with the increase of distance to the wall, especially up to 18 meters.

Another problem regards the fact that up to 20 meters the precision of the sonar is 1 m., instead of 0.1 m. (for shorter distances). In order to avoid this problem it is needed to get measures from the shortest side of the pool, obviously where and when possible.

3.4.4 The noise background of the pool

Experimental work has evidenced other possible acoustic sources which may represent noise for the sonar; this noise background of the pool comes from the external ambient. One example may be represented by the pump for the recharge of the water; anyway, in this case, the signal-to-noise ratio remains above acceptable levels.

More serious problems were noticed to come from other instruments used in the nearby of the water; in this case the signal-to-noise ratio was unacceptable, and any filtering techniques was useless.

Part II

Theoretical work

Chapter 4

Dynamic Model

4.1 Introduction

Autonomous vehicles are machines that are capable of intelligent motion and action without requiring either a guide to follow or teleoperator control. With the purpose to create a control strategy for an autonomous robot it is necessary to study the behavior of the physical system, which can be described by the evolution of a set of variables, called state variables.

The physical system we are considering, that is the pool-cleaner, is a kind of tracked robot able to move on the bottom of pools; hence it is a mobile base capable of motion on a planar or near-planar surface, a rolling machine that can turn and move forward or backward.

In the attempt to explain and describe the behavior of the dynamic model of the robot we find it necessary to separate the real situation from a simplified one. That is we need a simplified model since the real one is arduous to deal. In fact we have to consider lots of environmental variables, many of which vary in an unforeseeable way. The first simplification we need derives straight from the complex tracked system. The second one from the fact that the robot operates in an underwater environment which causes several disturbances. Let's consider later the former, and tell something about the latter.

Underwater the weight of the robot diminishes making traction not secure, therefore path following cannot be ensured if we only have internal methods for the position estimation. Moreover we should also consider the pumps and the brushes. Since the electrical engines control both the pumps and the wheels, when we stop the machine in order to have an inversion of direction, we also stop for a little time the pump, creating a water flux at the bottom of the robot. This can make the robot drift in orientation (at this stage the variation in position is not so relevant compared to the variation in orientation).

As far as the tracked robot problem, it is clearly the difficult to find an

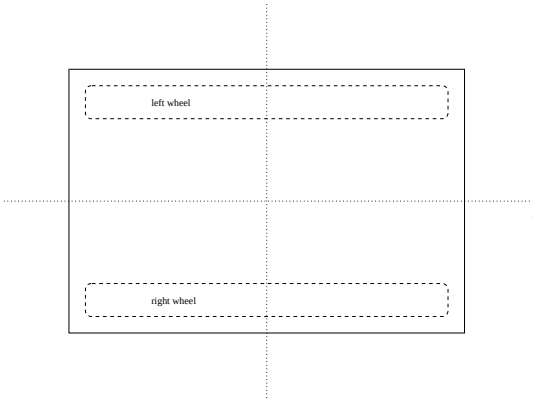

Figure 4.1: Robot plan view.

easy and satisfactory dynamic model. The crawler tracks have a great area of contact with the ground and the fixed point around which the machine turns when one track is stopped changes all the times depending upon various factors. This fact means that we cannot be able to find a perfect law to explain and describe the correct evolution in time of position and orientation of the pool-cleaner. Another important consideration regards the fact that we deal the robot on control regime; it means that acceleration and deceleration of the machine are neglected. We do not care what happen when the vehicle inverts the motion way (i.e., it touches a wall with either its front part or its back).

All these considerations have motivated formulation of a simplified dynamic model capable of giving a satisfactory description of the robot and its behaviors in the working environment.

4.2 The Model

The first step is identification of the type of vehicle used. It can be thought of as similar to a military tank, a vehicle driven by two independent, parallel motorized wheels. This robot can be characterized as nonholonomic, or is said to have nonholonomic constraints. Such constraints can have several effects on autonomous vehicles. A vehicle is limited in how it can move. It has fewer local degrees of freedom than the space it operates in. In other words, while a position and orientation (i.e., a posture) may be achievable, a current configuration of the vehicle might limit its ability to reach the posture without careful maneuvering or judicious use of its local degrees

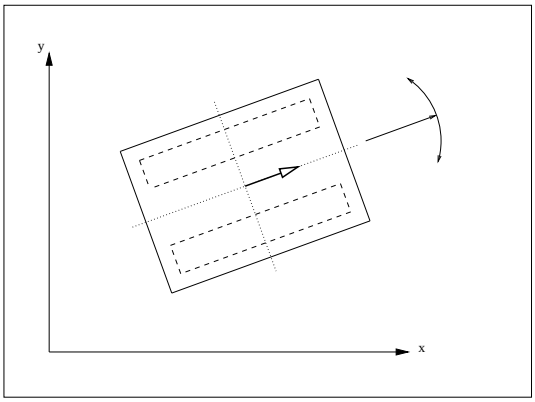

Figure 4.2: Robot plan view.

of freedom. Another important fact is that variables describing the vehicle position and orientation are dependent on an each other. For example, it cannot alter its orientation without changing its position (the two "wheels" of the robot are not able to move on opposite directions, that is it cannot rotate in place). Finally, variables describing vehicle position and orientation depend also on a history of previous values.

At this point, after a general classification of the pool-cleaner in a particular class of vehicles, its equations of motion are given. The robot is constrained to move in the direction that it is pointing. This is the nonholonomic constraint. The vehicle type is shown in figure 4.2.

We consider the robot as a point mass in space. This is the midpoint of the axis connecting the wheels (or crawler tracks) center.

For describing in a proper way the equation of motion of the robot, it is necessary to have information about both the instantaneous position and orientation in space of the robot. Otherwise we have to consider a triple of variables. We can make reference to a 2-D Cartesian coordinates system, in fact at this stage the robot moves onto the bottom of a "regular" swimming pool (i.e., a planar surface). In this way the x_p and y_p variables (that are function of both the time and the orientation) give us the evolution in time of the vehicle position regarding respectively the x and y axis.

As far as the orientation, we consider an angle ϑ with respect to the x-axis (it is a function of time and what we directly control with our software: in fact it is easy to prove the two angles in figure 4.3 are the same).

The value of the angle is given in radians. At this point we introduce $\vec{e}$, a vector with an unitarian module (a versor) characterized from the triple

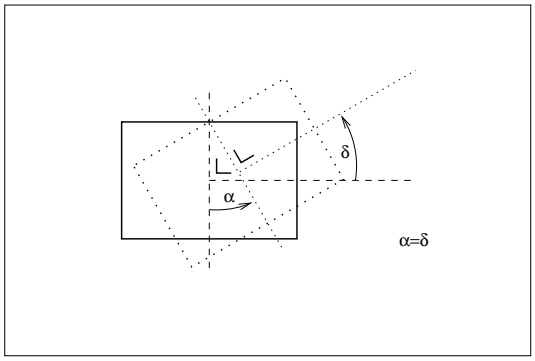

Figure 4.3: The two angles are the same.

of variables just described.

$$\vec{e} = (x_p, y_p, \vartheta)$$

Now a more accurate analysis is needed. We have already told the robot is considered to be on control regime. Hence it moves, and the motion direction is long x. The meaning of the turning angle (i.e., its value and its sign) depends on the position variation of the two wheels, and of course on time. Precisely the angular variation is directly proportional to the differential speed of the wheels. By assumption, the angle will be positive if the turning motion is in a growing-y direction for the +x motion, and in a decreasing-y for the -x motion. (In the -x motion path we consider the covered way as negative).

The introduction in the dynamic model of the total speed of the vehicle is useful. Considering the relative speed of the two wheels, we can assume the absolute value of the total speed is the average of the over cited variables. Now, for the sake of thoroughness, let the module of $\vec{e}$ be not unity but exactly the robot speed. We will call this new vector $\vec{v}$ (see figure 4.5).

$$\vec{v} = (x_p, y_p, \vartheta)$$

Now, if W is the robot width, and if v_r and v_l are respectively the speed of the right and left wheel, we can write:

$$\dot{x}_p = v \cos \vartheta \tag{4.1}$$

$$\dot{y}_p = v \sin \vartheta \tag{4.2}$$

$$\dot{\vartheta} = \frac{v_r - v_l}{W} \tag{4.3}$$

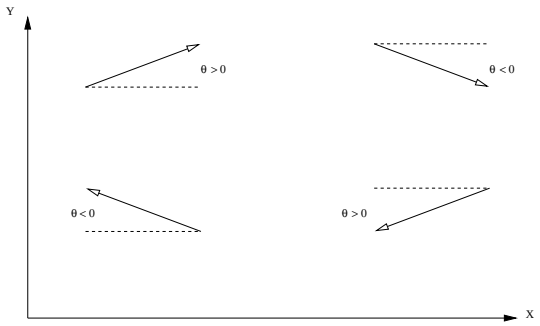

Figure 4.4: The value of the angle is positive or negative according to the direction of the motion of the robot.

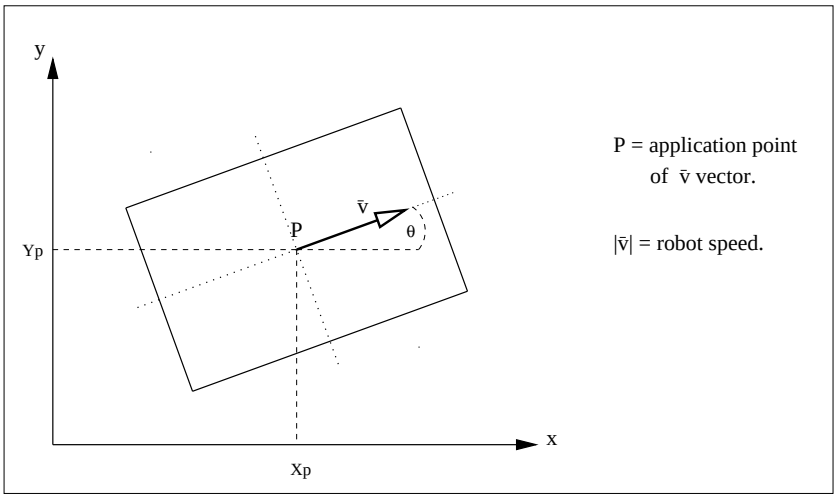

Figure 4.5: Representation of the robot with the vector.

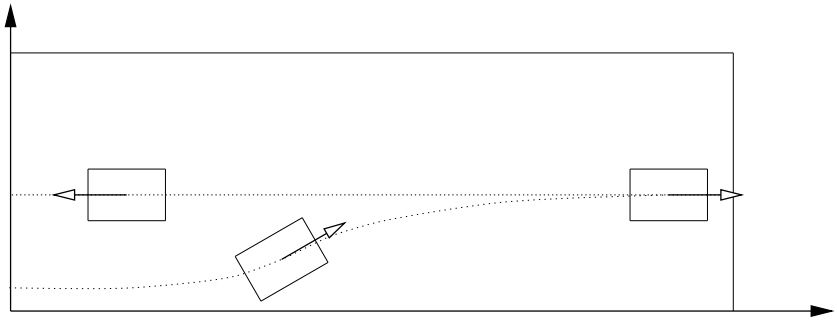

Figure 4.6: Path example.

where:

$$v = \frac{v_r + v_l}{2}$$

As far as the relation between the orientation angle and control signal (the time we stop one wheel), we will see later that in reality the situation is more complex and only experimentally obtainable. In fact we should also consider factors as the ground conditions (the pool-cleaner has never the same adherence condition when moving onto the bottom of the pool). This problem will be addressed soon in the control chapter.

4.3 Discrete representation

Now that we have a representation of the physical system, knowing the motion equations, we will try to adapt it on the control problem. Other wards, it is useful to deal with the problem in discrete terms. In fact the control routines act after the sonar data are received (i.e., every 500 ms). For this reason the motion path can be thought of as a sequence of short straight lines, each one differing from the precedent one by a small angle. Otherwise we would like the robot to travel in a straight line for the distance covered between two iterations, after that it had turned (if necessary) just a little. The steering angle is determinated by the control strategy.

4.3.1 Real system

At any iteration, information on the pose of the robot is given by the triple (x_p, y_p, ϑ). If now we consider two iterations (i) and (i+1), and a steering motion first and a linear one later on, the new pose of the machine will be

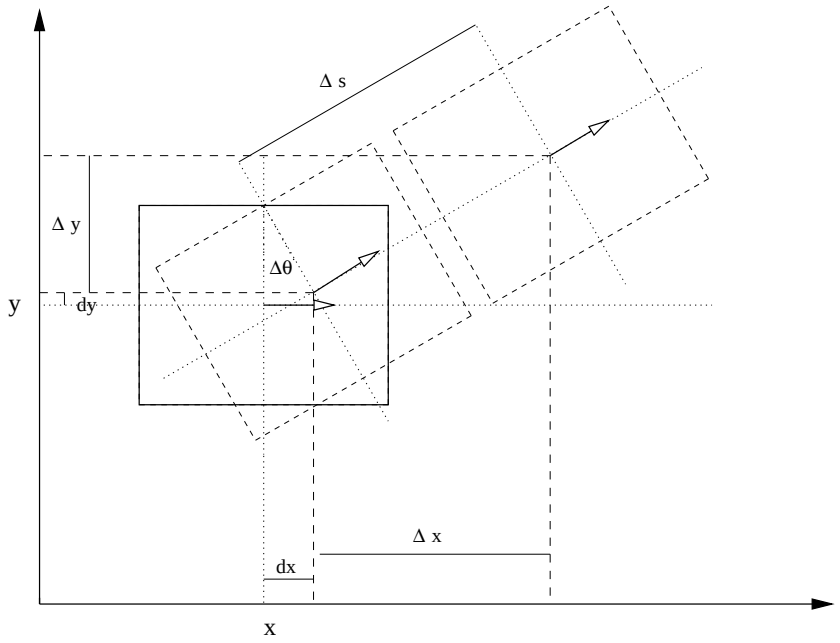

Figure 4.7: Real robot motion: the variation in position depends on a steering movement first, and on a straight one after. During the steering motion, the position of the robot changes from (x,y) to (x+dx,y+dy); after that, a straight motion bring the robot in the final position (x+dx+Δx, y+dy+Δy).

described by the triple (x'_p, y'_p, ϑ'). Hence we have the following transformation:

$$(x_p, y_p, \vartheta) \longrightarrow (x'_p, y'_p, \vartheta')$$

We consider the global coordinate system. We assume the change in orientation depends only on the steering motion, while for the position it is necessary to consider both kind of movements, as we can see in the figure 4.7.

Starting now we will call the position coordinates x and y. We will call Δs the straight path of the robot and $\Delta\vartheta$ the small angle of the steering motion.

Relating to the figure 4.8 it is easy to see that the two different kind of movement give rise to the following variation in position:

- steering motion:

$$dx = \frac{W \sin \Delta\vartheta}{2} \tag{4.4}$$

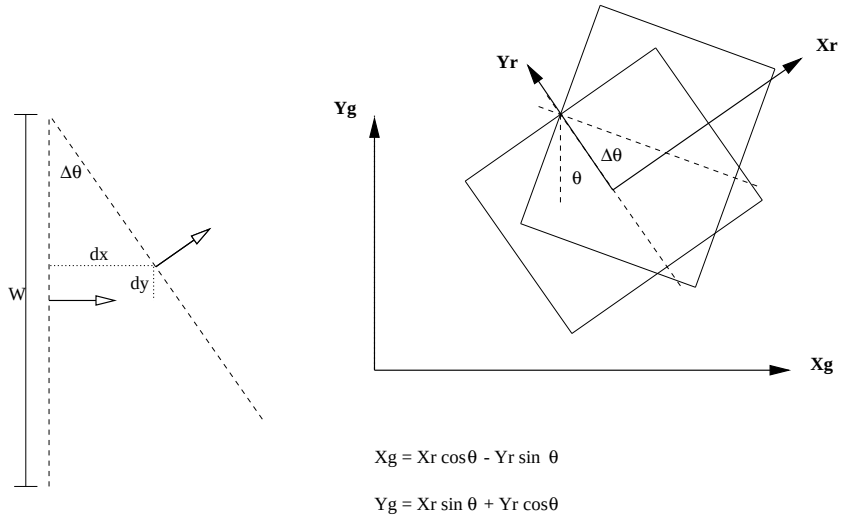

Figure 4.8: Particular of the steering motion: variation in position regarding the robot coordinate system, and the global coordinate system.

$$dy = \frac{W(1-\cos\Delta\vartheta)}{2} \tag{4.5}$$

This is the (x,y)-motion regarding the robot coordinate system; now (see figure 4.8), with a rotation matrix we can find the movement description in the global coordinate system.

$$dx = \left(\frac{W\sin\Delta\vartheta}{2}\right)\cos\vartheta - \left(\frac{W(1-\cos\Delta\vartheta)}{2}\right)\sin\vartheta \tag{4.6}$$

$$dy = \left(\frac{W\sin\Delta\vartheta}{2}\right)\sin\vartheta + \left(\frac{W(1-\cos\Delta\vartheta)}{2}\right)\cos\vartheta \tag{4.7}$$

- straight motion:

$$\Delta x = \Delta s\cos(\vartheta+\Delta\vartheta) \tag{4.8}$$

$$\Delta y = \Delta s\sin(\vartheta+\Delta\vartheta) \tag{4.9}$$

At this point we can easily calculate the change in position of the robot between two iterations. For this purpose we consider the following transformation:

$$(x_i, y_i, \vartheta_i) \longrightarrow (x_{i+1}, y_{i+1}, \vartheta_{i+1})$$

The discrete motion equations are:

$$
\begin{aligned}
x_{i+1} &= x_i \\
&\quad + \left(\frac{W \sin \Delta\vartheta_i}{2}\right) \cos \vartheta_i - \left(\frac{W(1 - \cos \Delta\vartheta_i)}{2}\right) \sin \vartheta_i \\
&\quad + \Delta s_i \cos\left(\vartheta_i + \Delta\vartheta_i\right) && (4.10) \\
y_{i+1} &= y_i \\
&\quad + \left(\frac{W \sin \Delta\vartheta_i}{2}\right) \sin \vartheta_i + \left(\frac{W(1 - \cos \Delta\vartheta_i)}{2}\right) \cos \vartheta_i \\
&\quad + \Delta s_i \sin\left(\vartheta_i + \Delta\vartheta_i\right) && (4.11) \\
\vartheta_{i+1} &= \vartheta_i + \Delta\vartheta_i && (4.12)
\end{aligned}
$$

Now, taking care of the control signals too (i.e., speed of the two wheels and steering time), with the assumption that $T_{c,i}$ is the time we stop a wheel in the (i) step (for the $\Delta\vartheta_i$ angle), and that $V_{r,i}$, $V_{l,i}$ are

$$
\begin{aligned}
V_{r,i} &= v_{r,i} R_i && (4.13) \\
V_{l,i} &= v_{l,i} L_i && (4.14)
\end{aligned}
$$

where R and L can be either 0 or 1 (it is 0 when we stop the wheel), we can write:

$$
\begin{aligned}
\Delta s_i &= v(\Delta T - T_{c,i}) && (4.15) \\
\Delta\vartheta_i &= \frac{(V_{r,i} - V_{l,i}) T_{c,i}}{W} && (4.16)
\end{aligned}
$$

where ΔT is constant and its value is 500[ms].

Now, in order to find the information on the robot after N steps, it is possible to write:

- at the first step

$$
\begin{aligned}
x_1 &= f(x_0, \vartheta_0, \Delta\vartheta_0, \Delta s_0) && (4.17) \\
y_1 &= f(y_0, \vartheta_0, \Delta\vartheta_0, \Delta s_0) && (4.18) \\
\vartheta_1 &= f(\vartheta_0, \Delta\vartheta_0) && (4.19)
\end{aligned}
$$

- after N steps

$$
\begin{aligned}
x_N &= f(x_0, \vartheta_0, \Delta\vartheta_0, ..., \Delta\vartheta_{N-1}, \Delta s_0, ..., \Delta s_{N-1}) && (4.20) \\
y_N &= f(y_0, \vartheta_0, \Delta\vartheta_0, ..., \Delta\vartheta_{N-1}, \Delta s_0, ..., \Delta s_{N-1}) && (4.21) \\
\vartheta_N &= f(\vartheta_0, \Delta\vartheta_0, ..., \Delta\vartheta_{N-1}) && (4.22)
\end{aligned}
$$

At this point, using dx, dy and Δx, Δy we can write:

$$x_N = x_0 + \sum_{i=0}^{N-1} (dx_i + \Delta x_i) \tag{4.23}$$

$$y_N = y_0 + \sum_{i=0}^{N-1} (dy_i + \Delta y_i) \tag{4.24}$$

$$\vartheta_N = \vartheta_0 + \sum_{i=0}^{N-1} \Delta\vartheta_i \tag{4.25}$$

4.3.2 Simplified system

The equation system we wrote is not linear and arduous to deal with for our purpose. For this reason, and for the sake of simplicity, we can try to linearize the system, making some simplifications.

First we will treat the steering motion, later on the straight one. We want to consider only the modification in orientation first, and only the variation of the position after that.

Since this representation presupposes the possibility to modify the angle without changing the (x,y) point even if it is not true, it is necessary to understand if this model is effectively realizable, and in which cases. In reality, as we know, the steering angle is not null only if the speed of one wheel differs from the other. This fact implies even that the covered path of the two wheels is not the same; consequently it is useful to understand if dx and dy are negligible. Considering small angles, we can write:

$$dx = \left(\frac{W \sin\Delta\vartheta}{2}\right)\cos\vartheta - \left(\frac{W(1-\cos\Delta\vartheta)}{2}\right)\sin\vartheta \simeq 0 \tag{4.26}$$

$$dy = \left(\frac{W \sin\Delta\vartheta}{2}\right)\sin\vartheta + \left(\frac{W(1-\cos\Delta\vartheta)}{2}\right)\cos\vartheta \simeq 0 \tag{4.27}$$

since we have:

$$\begin{aligned} \sin\varphi &\simeq 0 \\ \cos\varphi &\simeq 1 \end{aligned}$$

for $\varphi \to 0$.

All these considerations are justified by the fact that in reality, for the control strategies we use, the steering angle is never more than 12 degrees in absolute value. This means that we have dx and dy really small (i.e., they could be respectively 0.04m and 0.005m).

As far as the straight line movement, we assume it is constant. In fact, in reality we do not make control actions each steps, and even if we do them,

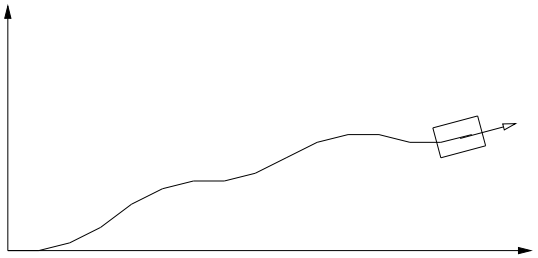

Figure 4.9: Discrete path example.

the steering time is small compared to ΔT. For this reason we can consider

$$\Delta s \simeq v \Delta T$$

and we will use it even in the control strategies.

In figure 4.9 we can see the path of the robot, broken into short segments.

The simplified robot equations are:

$$\begin{aligned} x_{i+1} &= x_i + \Delta s \cos(\vartheta_i + \Delta\vartheta_i) && (4.28)\\ y_{i+1} &= y_i + \Delta s \sin(\vartheta_i + \Delta\vartheta_i) && (4.29)\\ \vartheta_{i+1} &= \vartheta_i + \Delta\vartheta_i && (4.30) \end{aligned}$$

Now we can do something more. Since the system is still non-linear, we can try to linearize it around the working space of the control (i.e., the little variations of ϑ around 0). We will use a Taylor expansion (McLourin, since $x_0 = 0$) until the first term; in this way we obtain:

$$\begin{aligned} x_{i+1} &= x_i + \Delta s && (4.31)\\ y_{i+1} &= y_i + \Delta s(\vartheta_i + \Delta\vartheta_i) && (4.32)\\ \vartheta_{i+1} &= \vartheta_i + \Delta\vartheta_i && (4.33) \end{aligned}$$

Now, considering the control signal as

$$\begin{aligned} u_r &= V_r T_c && (4.34)\\ u_l &= V_l T_c && (4.35) \end{aligned}$$

we can write:

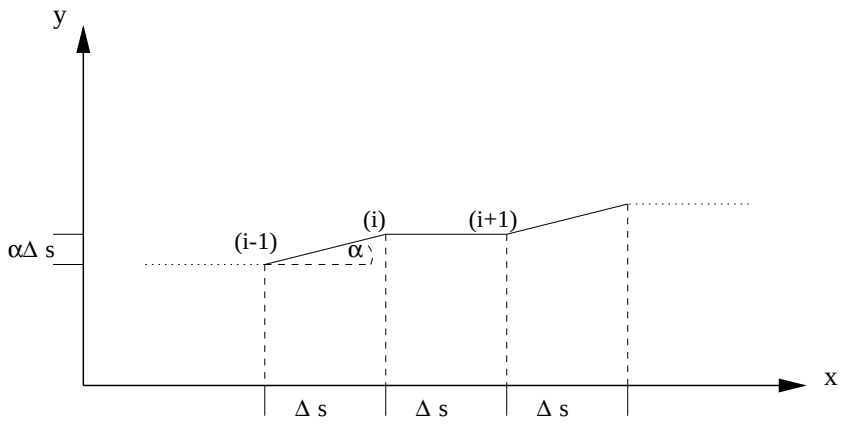

Figure 4.10: Particular of the discrete simplified motion: the variation in the x-direction is considered constant, the one in the y-direction depends on the steering angle.

$$x_{i+1} = x_i + \Delta s \tag{4.36}$$

$$y_{i+1} = y_i + \Delta s \vartheta_i + \frac{\Delta s (u_r - u_l)}{W} \tag{4.37}$$

$$\vartheta_{i+1} = \vartheta_i + \frac{(u_r - u_l)}{W} \tag{4.38}$$

The equations we derived are obviously only approximate, and rely on the assumption of small movements.

We can introduce the control vector $\vec{u}$ with the following definition:

$$\vec{u} \stackrel{\text{def}}{=} \begin{pmatrix} u_r \\ u_l \end{pmatrix} \tag{4.39}$$

Now, in order to represent the system with a vector equation, we define also the state vector as

$$\vec{x} \stackrel{\text{def}}{=} \begin{pmatrix} x \\ y \\ \vartheta \end{pmatrix} \tag{4.40}$$

The dynamic system can be described by the following:

$$\vec{x}_{i+1} = A\vec{x}_i + B\vec{u}_i + C \quad i = 0, 1, ... \tag{4.41}$$

The A, B and C matrix are:

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & \Delta s \\ 0 & 0 & 1 \end{pmatrix} \tag{4.42}$$

$$B = \begin{pmatrix} 0 & 0 \\ \Delta s/W & -\Delta s/W \\ 1/W & -1/W \end{pmatrix} \tag{4.43}$$

$$C = \begin{pmatrix} \Delta s \\ 0 \\ 0 \end{pmatrix} \tag{4.44}$$

Now, in order to find the global position of the pool-cleaner after N iterations, with the assumption that it starts its mission from the origin of the global Cartesian coordinate system, with a zero starting angle (i.e., $x_0 = y_0 = \vartheta_0 = 0$), we can write:

$$x_N = \sum_{i=0}^{N-1} \Delta s \tag{4.45}$$

$$y_N = \sum_{i=0}^{N-1} [\Delta s(\vartheta_i + \Delta\vartheta_i)] \tag{4.46}$$

$$\vartheta_N = \sum_{i=0}^{N-1} \Delta\vartheta_i \tag{4.47}$$

With these equations we can compute the robot pose even after forward and backward motions. In fact we have to remember that Δs is positive when the robot moves in the +x direction, and negative for the -x direction; we also have a similar convention for the angle.

Chapter 5

Control system

5.1 Introduction

Our objective is to have a tracked-vehicle robot cover a rectangular area. The only other human input will be the variance of the measurement error that can expected. This process error will represent the environmental uncertainty (i.e., wheel slippage, uneven terrain). We will assume that these variances are constant throughout the entire area. To solve this problem, the signals are processed in a Kalman filter. Robot position measurements are assumed to be provided by a couple of underwater sonar that measure the distance of the robot from the walls. Figure 5.1 gives a representation of the inputs and the outputs for the various subsystems involved in this project.

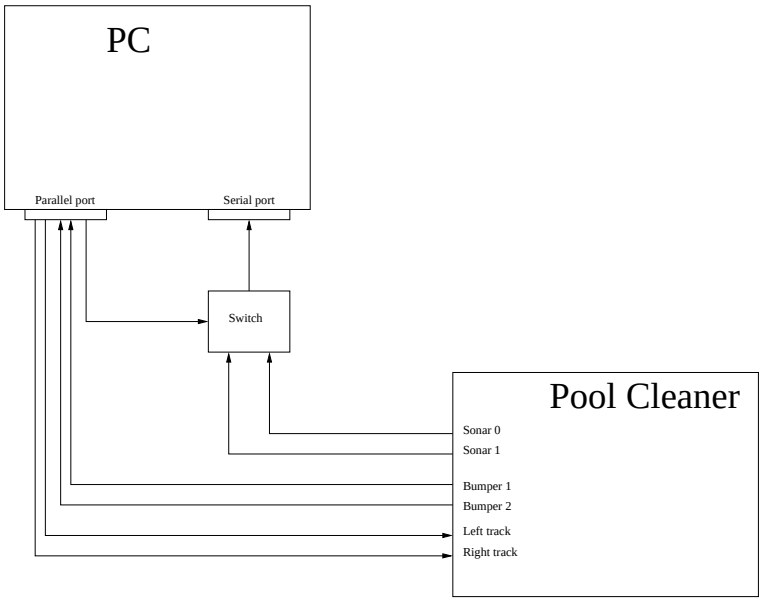

Figure 5.1: Control system scheme.

This project has three main development stages :

- General algorithm development of the control system.
- Algorithm modification with introduction of process disturbance.
- Algorithm modification with introduction of both process disturbance and measurement noise.

5.2 General algorithm development of control system

The algorithm will treat the system using a "lawn-mower" approach. The system block scheme is shown in figure 5.2.

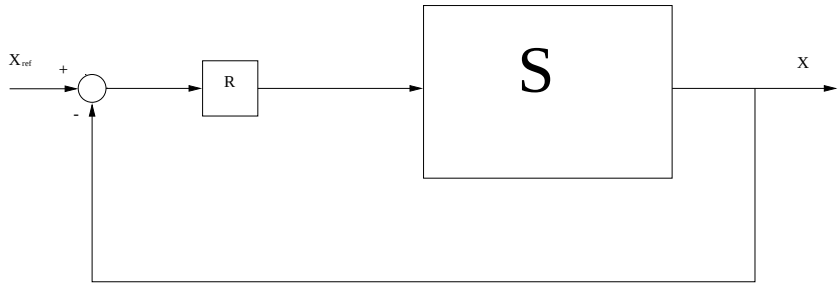

Figure 5.2: Control system block scheme.

The figure 5.3 shows how the rectangular area will be covered by the pool-cleaner.

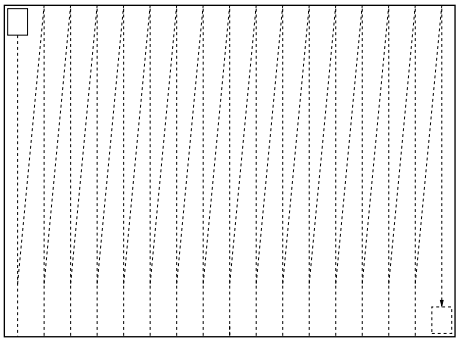

Figure 5.3: Cleaning path.

The robot will proceed initially in the upward $+x$ direction where $x_{i+1} > x_i$. When the robot reaches the wall, it reverses its run (i.e., $-x$ movement) and follows an oblique path that increase its y position. This will continue until the entire field is completely covered. This is the simplest algorithm, compatible with the robot structure for covering the bottom of the rectangular pool.

5.3 Introduction of process disturbance

Unfortunately the underwater pool's environment is very uncertain and it is difficult to find a correct dynamic model for a tracked vehicle in this environment. Because of these problems it is possible to add in the system noise ξ the linearization errors and the other environmental uncertainty. The modified system is shown in figure 5.4.

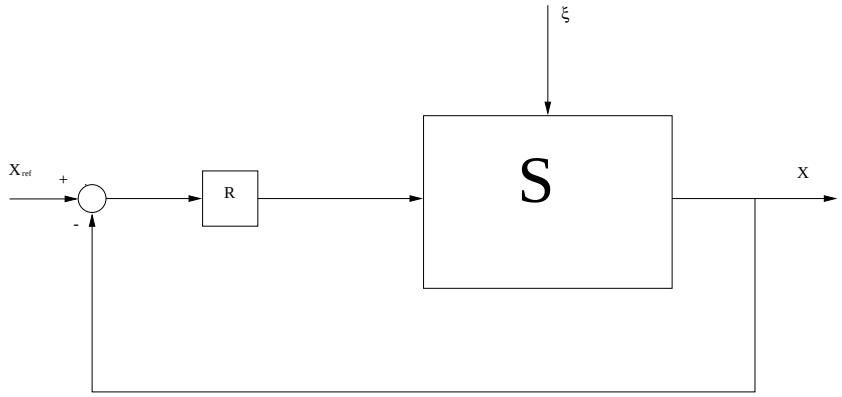

Figure 5.4: Control system with noise.

Each of the paths will require a reference value (e.g. the desired path as defined by the lawn-mower pattern) that the control system will monitor and correct if necessary. Obviously, on each side of the reference value will be a minor correction band. This band will be evenly spaced on each side of the reference value of $\pm e$. Inside this correction band the pool cleaner proceeds forward without correction. If the robot drifts outside this band a correction is required. Figure 5.5 represents a graphical representation of the relationship between the correction landings to Yref.

The target of our control system is to find the condition necessary and sufficient to satisfy the following conditions:

$$\lim_{t\to\infty}(y - y_{ref}) = 0$$

$$\lim_{t\to\infty}(\vartheta - \vartheta_{ref}) = 0$$

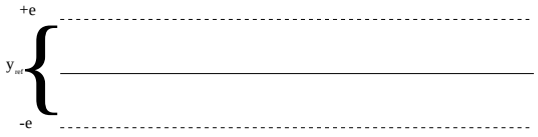

Figure 5.5: Error threshold.

5.3.1 Sufficiency condition

Now we will prove the sufficiency condition for $\exists \lim_{t\to\infty}(y - y_{ref}) = 0$ and for $\exists \lim_{t\to\infty}(\vartheta - \vartheta_{ref}) = 0$. To reach the target we defined a Liapunov function

$$V = \frac{1}{2}\left((\vec{x} - \vec{x}_{ref})^t(\vec{x} - \vec{x}_{ref})\right) \tag{5.1}$$

This function is defined positive and $V = 0 \Leftrightarrow (\vec{x} - \vec{x}_{ref}) = 0$ We consider this function as a candidate Liapunov function. It is possible to write the function ( ) in the form :

$$V = \frac{1}{2}\left((y - y_{ref})^2 + (\vartheta - \vartheta_{ref})^2\right) \tag{5.2}$$

And consider separately :

$$V_y = \frac{1}{2}(y - y_{ref})^2 \tag{5.3}$$

$$V_\vartheta = \frac{1}{2}(\vartheta - \vartheta_{ref})^2 \tag{5.4}$$

Now that we have to investigate $\dot{V}$ we should render :

$$\dot{V} = \begin{cases} 0 & if(\vec{x} - \vec{x}_{ref}) = 0 \\ < 0 & otherwise \end{cases} \tag{5.5}$$

Let start to consider $\dot{V}_y$:

$$\dot{V}_y = \dot{y}(y - y_{ref}) \tag{5.6}$$

We should render :

$$\dot{V}_y = \begin{cases} 0 & if(y - y_{ref}) = 0 \\ < 0 & otherwise \end{cases} \tag{5.7}$$

In accord with the equation (5.3) if we put :

$$\dot{y} = -\gamma_y(y - y_{ref}) \tag{5.8}$$

the condition (5.7) is satisfied.
Let us now consider $\dot{V}_\vartheta$:

$$\dot{V}_\vartheta = \dot{\vartheta}(\vartheta - \vartheta_{ref}) \tag{5.9}$$

We should render :

$$\dot{V}_\vartheta = \begin{cases} 0 & if(\vartheta - \vartheta_{ref}) = 0 \\ < 0 & otherwise \end{cases} \tag{5.10}$$

In accord with the equation (5.4) we can put :

$$\dot{\vartheta} = -\gamma_\vartheta(\vartheta - \vartheta_{ref}) \tag{5.11}$$

With this choice we satisfy the condition (5.10). With these condition it's guaranteed that the (5.1) is decreasing, and its decrease will stop asymptotically in $V = 0$. Our control law will assume the following shape:

$$correction = -\left[\gamma_y(y - y_{ref}) + \gamma_\vartheta(\vartheta - \vartheta_{ref})\right] \tag{5.12}$$

The equation (5.12) returns the value for how long the left or the right track respectively if its value is positive or negative, should be stopped. In according with simulated experiment it has been chosen this kind of control:

$$correction = -\gamma\left[(y - y_{ref}) + 7(\vartheta - \vartheta_{ref})\right] \tag{5.13}$$

Where γ is the variable gain given by:

$$\gamma = 200|MaxErr - |y - y_{ref}|| + 1 \tag{5.14}$$

Where $MaxErr$ is the maximum error fixed as 2 meters. In this way when the vehicle is far from the reference distance the gain is smaller in order to avoid the corrections make the system to become instable. The figure 5.6 shows the simulated results of this control algorithm. This simulation has been done with 160 iterations, it means that with a speed of 0.3 meters per second and one data each 500 milliseconds we are considering a 25 meters long pool we can see that, with an initial error of .03 meters, after about 30 iterations (4.5 meters) the error becomes negligible. With a bigger error the simulation returns the results showed by figure 5.7. It is possible to see that also in this situation the error becomes negligible after few meters.

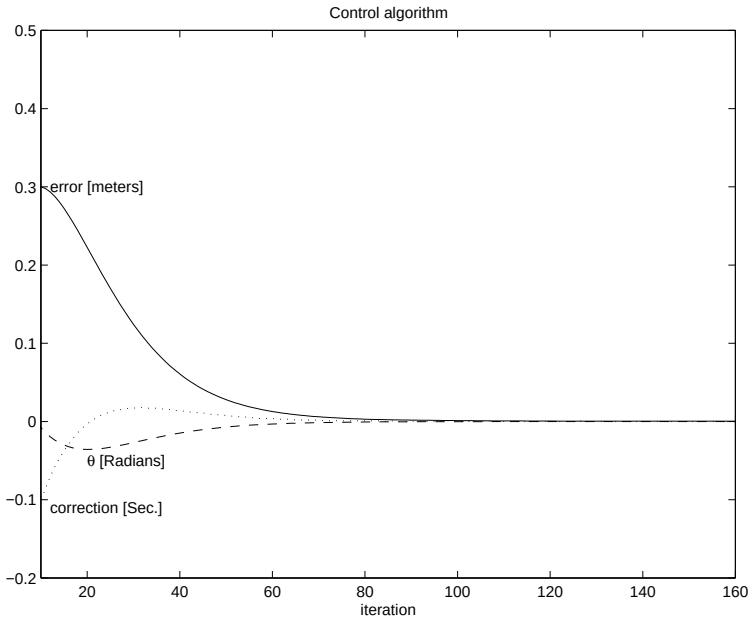

Figure 5.6: Control system simulation. Initial error 0.3 meters

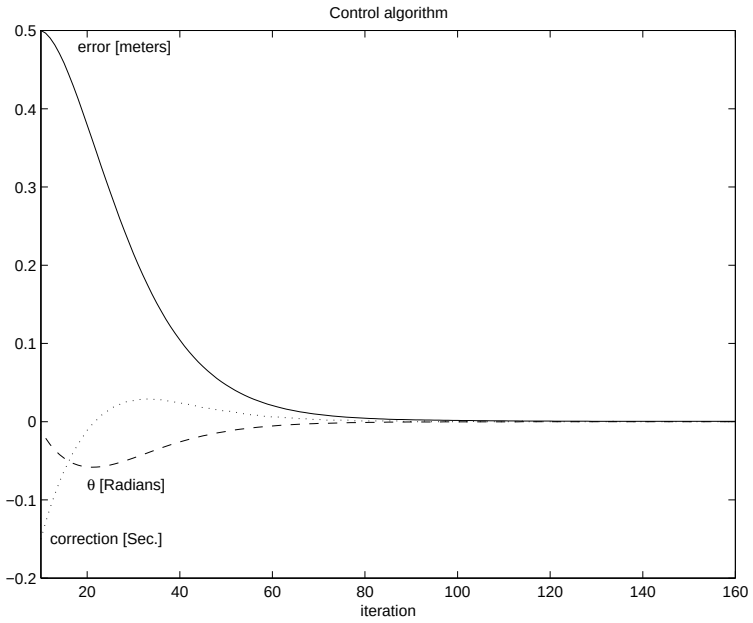

Figure 5.7: Control system with noise and measurement noise.

5.4 Introduction of measurement noise

While this addresses the situation of dynamics uncertainty it assumes that the position can be measured accurately by external sensors. Unfortunately in real world situations and especially in an underwater environment, position measurements are not exact and measurement errors must be considered in the robot control system. Our system will utilize two sonars to measure the y distance. It is necessary to use two sonars because the minimum distance measured reliable by the sonar is about two meters and when the robot works close to a wall the control system uses the distance to the opposite wall. The x position is not relevant in this application we need only to know when the pool-cleaner reaches the wall and it is possible to detect this with two micro-switches mounted on the two bumpers of the robot. A Kalman filter [4] will use these measurements and the robots dynamic model to provide estimates of the complete state vector $(x\ y\ \vartheta)^t$. Because the robots dynamic motion equations are non-linear, we need a linearization around the previous state estimate to use a Kalman Filter (KF) [4]. It is possible to make this approximation because we only consider small angle variations, and we can consider the linear Taylor approximation of the system function. The filter used in conjunction with external and internal position estimates will minimize the internal navigation output errors in a feedback configuration, as seen in figure 5.8.

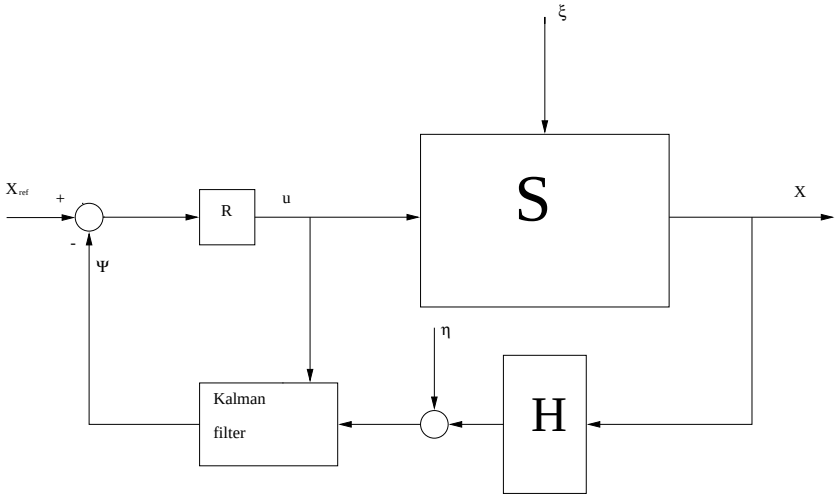

Figure 5.8: Control system with noise and measurement noise.

Most of the problems of the control of the pool-cleaner appear when the

pool-cleaner is close to the wall, in fact it is be possible that because of the reflection of the sonar beam the measure is corrupted. Under these conditions the sonar measure cannot be trusted, and it is necessary to proceed without feedback. Moreover, when the robot reverses its run, it is possible to have a perturbation of the heading because of the stop of the aspiration pump and the brush. During this period the vehicle is floating, which introduces uncertainty. This is a big problem because it is necessary to proceed for a few seconds without knowledge of any state variable. Figures 5.9 and 5.10 illustrate these kind of problems.

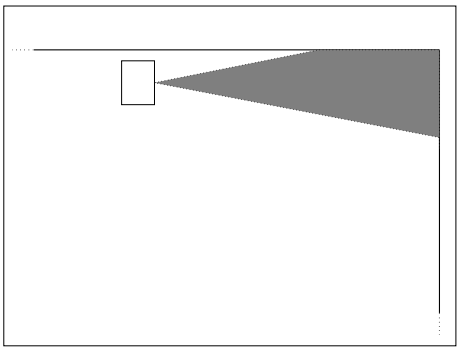

Figure 5.9: Sonar beam reflection.

If we denote the state vector by $\vec{x}$ and denote the measurement vector by $\vec{y}$, the dynamic system (in the discrete form) can be described by

$$\vec{x}_{i+1} = \vec{h}_i(\vec{x}_i, \vec{u}_i) + \vec{\xi}_i \quad i = 0, 1, ... \tag{5.15}$$

$$\vec{f}(\vec{y}_i, \vec{x}_i) = \vec{0} \quad i = 0, 1, ... \tag{5.16}$$

Now we are considering a linearized system, therefore we can use the following equations (see Chapter 5).

$$\vec{x}_{i+1} = A\vec{x}_i + B\vec{u}_i + \vec{\xi}_i \quad i = 0, 1, ... \tag{5.17}$$

$$\vec{y}_i = H\vec{x}_i + \eta_i \quad i = 0, 1, ... \tag{5.18}$$

Where $\vec{x}$ is the state vector $\vec{x} = \begin{pmatrix} y \\ \vartheta \end{pmatrix}$, we don't consider the other Cartesian variable x because it is not controllable by our system. So we can simplify the calculations. A is the discrete system matrix, and it is defined as:

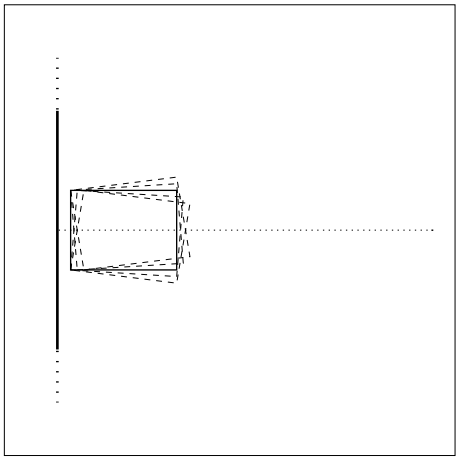

Figure 5.10: Starting angle uncertainly.

$$A = \begin{pmatrix} 1 & \Delta s \\ 0 & 1 \end{pmatrix} \tag{5.19}$$

B is defined as:

$$B = \begin{pmatrix} \Delta s/W & -\Delta s/W \\ 1/W & -1/W \end{pmatrix} \tag{5.20}$$

$\vec{\xi}$ is the vector of random disturbance of the dynamic system and it can be modeled as white noise.

$$E[\vec{\xi_i}] = \vec{0} \tag{5.21}$$

and

$$E[\vec{\xi_i}\vec{\xi_i^t}] = Q_i \tag{5.22}$$

In practise, the system noise covariance Q_i is usually determinated on the basis of experience and intuition (i.e., it is guessed). The measurement vector $\vec{y_i}$ doesn't contain the complete state vector, but only the measurement from the sonar. We assume the measurement system is disturbed by

trial n. *n.*	*angle* 10^{-2} *radians*	*Y error* *m.*
1	-1	.1
2	-1.8	.07
3	0	-.2
4	.5	.23
5	0	-.05
6	-2	-.1
...	...	...

Table 5.1: Experimental trials.

additive white noise η_i with zero mean value and a variance $\sigma = .3$. The matrix H is the measure channel matrix.

$$H = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \tag{5.23}$$

It was outlined earlier how a big problem of the control of the pool-cleaner is the uncertainly of the first state $\vec{x}_0$, because when the robot reverse its run there is no knowledge about its heading and it is also impossible to use the first measures. We can characterize the first state with a Gaussian probability:

$$p(\vec{x}_0) \quad = \quad e^{-\frac{1}{2}\|\vec{x}_0 - \vec{\alpha}\|^2 \Sigma^{-1}} \tag{5.24}$$

Where $\vec{\alpha}$ is the mean value of the state vector and Σ is the covariance matrix. The mean value of the state vector has been derived (determined) through a lot of experiments as showed in the following table. The result is a y variable with a mean value centered on the reference value yref, and a variance of .2. Experimentally the angle had a negative mean value because of the pump which revolves anti-clockwise. The probability distribution function can be approximated by a Gaussian which has mean value of $-1.74 \cdot 10^{-2}$ radians and a variance of $3.48 \cdot 10^{-2}$ as showed in the figure 5.11.

With this information it is possible to use the Kalman filtering theory [4] to obtain a state estimate. The Kalman filter supplies :

$$\vec{\psi_i} \quad = \quad E[\vec{x}_i / I_i] \tag{5.25}$$

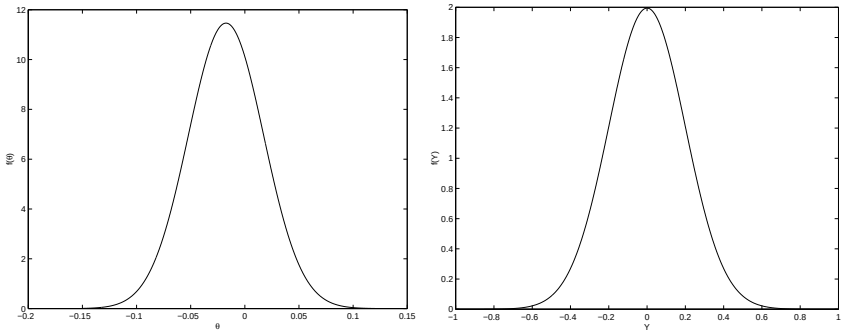

Figure 5.11: Angle and position probability distribution.

and

$$\Sigma_i \quad = \quad cov[\vec{x}_i / I_i] \tag{5.26}$$

Where I_i is the informative whole. It is demonstrable that:

$$\vec{\psi_0} \quad = \quad \underbrace{\vec{x}_0}_{Predicted} + K_0(y_0 - \underbrace{y_0}_{Predicted}) \tag{5.27}$$

$$\vec{\psi_0} \quad = \quad \vec{\alpha} + K_0(y_0 - H\vec{\alpha}) \tag{5.28}$$

Where K_0 is the Kalman gain matrix. The estimate of the mean value of the state is composed by two terms, the first one is the prediction, and the second one is the innovation.

$$\vec{\psi_0} \quad = \quad \underbrace{\vec{\alpha}}_{Prediction} + \underbrace{K_0(y_0 - H\vec{\alpha})}_{Innovation} \tag{5.29}$$

Let us now consider the covariance:

$$\Sigma_0^{-1} \quad = \quad \underbrace{cov(\vec{x}_0)^{-1}}_{Predicted} + H^t R^{-1} H \tag{5.30}$$

It is demonstrable that:

$$cov(\vec{x}_0) = \Sigma \tag{5.31}$$

Therefore :

$$\Sigma_0^{-1} = \Sigma^{-1} + H^t R^{-1} H \tag{5.32}$$

The Kalman gain matrix K_0 is given by :

$$K_0 = \Sigma_0 H^t R^{-1} \tag{5.33}$$

Let us now determine a recurrent formula to propagate ψ_i and Σ_i It is easily demonstrable that generally $P(\vec{x}_i|I_i) = N(\vec{\psi}_i, \Sigma_i)$. It is possible to proceed by induction :

$$\vec{\psi}_{i+1} = \underbrace{\vec{x_i}}_{Predicted} + K_{i+1}(y_{i+1} - \underbrace{y_{i+1}}_{Predicted}) \tag{5.34}$$

$$\vec{\psi}_{i+1} = E(\vec{x}_{i+1}/I_i) + K_{i+1}(y_{i+1} - E(y_{i+1}/I_i) \tag{5.35}$$

Where :

$$E(\vec{x}_{i+1}/I_i) = E(A\vec{x}_i + B\vec{u}_i + \vec{\xi}_i/I_i) \tag{5.36}$$

$$E(\vec{x}_{i+1}/I_i) = A\vec{\psi}_i + B\vec{u}_i \tag{5.37}$$

As $E(\vec{\xi}_i) = \vec{0}$.
And :

$$E(y_{i+1}/I_i) = E((H\vec{x}_{i+1} + \vec{\eta}_{i+1})/I_i) \tag{5.38}$$

$$E(y_{i+1}/I_i) = H(A\vec{\psi}_i + B\vec{u}_i) \tag{5.39}$$

As $E(\vec{\eta}_i) = \vec{0}$.
Let us now consider the propagation of the covariance Σ_i:

$$\Sigma_{i+1}^{-1} = \underbrace{cov(\vec{x}_{i+1}/I_i)^{-1}}_{Predicted} + H^t R^{-1} H \tag{5.40}$$

Where :

$$cov(\vec{x}_{i+1}/I_i) = cov(A\vec{x}_i + B\vec{u}_i + \vec{\xi}_i/I_i) \tag{5.41}$$

$$cov(\vec{x}_{i+1}/I_i) = cov(A\vec{x}_i/I_i) + Q \tag{5.42}$$

As $cov(\vec{\eta}) = Q$, and :

$$cov(A\vec{x}_i/I_i) = E\left(A(\vec{x}_i - \vec{\psi}_i)(\vec{x}_i - \vec{\psi}_i)^t A^t\right) \quad (5.43)$$

$$cov(A\vec{x}_i/I_i) = A\Sigma - iA^t \quad (5.44)$$

We finally obtain these equations :

$$\vec{\psi}_{i+1} = A\vec{\psi}_i + B\vec{u}_i + K_{i+1}\left[y_{i+1} - H(A\vec{\psi}_i + B\vec{u}_i)\right] \quad (5.45)$$

$$\vec{\psi}_0 = \vec{\alpha} + K_0(y_0 - H\vec{\alpha}) \quad (5.46)$$

$$\Sigma_{i+1}^{-1} = (A\Sigma_i A^t + Q)^{-1} + H^t R^{-1} H \quad (5.47)$$

$$\Sigma_0^{-1} = \Sigma^{-1} + H^t R^{-1} H \quad (5.48)$$

$$K_i = \Sigma_i H^t R^{-1} \quad (5.49)$$

As the covariance matrix doesn't depend on line variables, it is possible to calculate it off-line. As shown in the figure 5.12 the elements of the Kalman gain matrix are stabilized after about thirty steps.

It is therefore sufficient to store the first thirty matrices.

$$K_0 = \begin{pmatrix} .4 & 0 \\ 0 & 0 \end{pmatrix}, \quad K_1 = \begin{pmatrix} .4239 & 0 \\ 0.01 & 0 \end{pmatrix}, ..., K_{30} = \begin{pmatrix} .4881 & 0 \\ 0.297 & 0 \end{pmatrix}, \quad ....$$

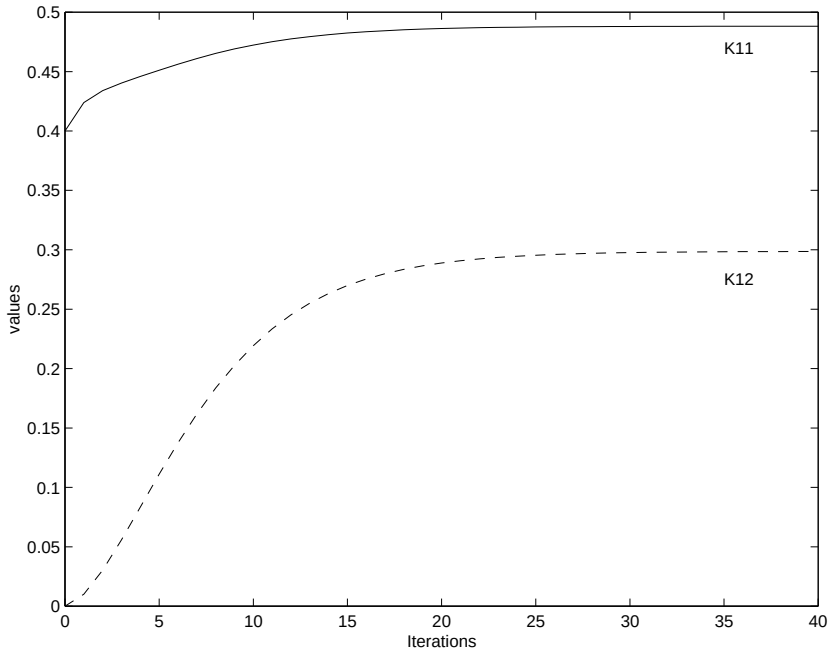

Figure 5.12: Covariance matrix elements K_{11} and K_{12}.

The simulation of our control algorithm adding the measurement noise and the finite precision of the instrument shows the following results. The simulation starts with an error of 0.3 meters, the following figures show respectively the sonar output (6.13), the estimation of the state vector (6.14), the control vector (6.15), and the real position of the vehicle (6.16). The sonar output data have a resolution of ten centimeters, so our system doesn't receive a continuous data, but a quantized value. These values are also corrupted by the noise present in the underwater environment. A typical stream of data input received by our system is shown in the figure (5.14).

The figure 5.14 shows how the Kalman filter derives the state estimation from the sonar measure and the control output. The next figure (5.15) shows the output control of our system, these control values are quantized in a set of values spaced at 50 milliseconds.

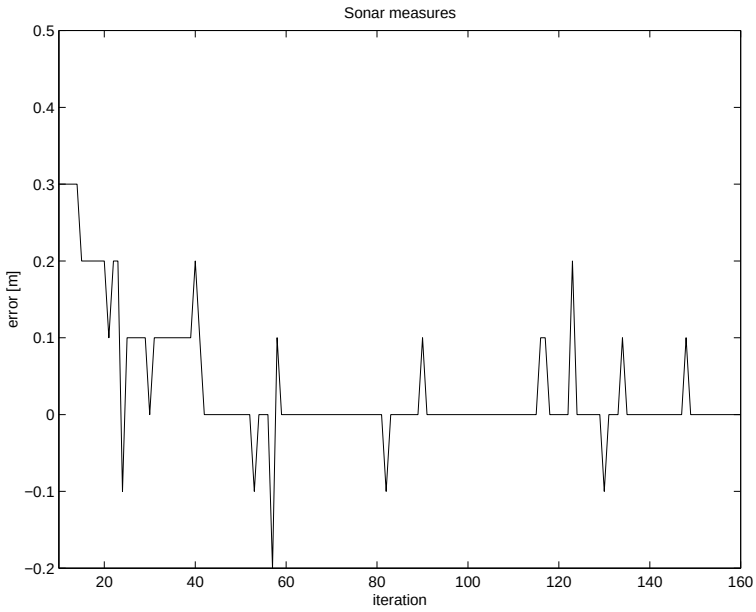

Figure 5.13: Sonar output (initial error 0.3 meters). This figure represents a simulated sonar output with an additive noise.

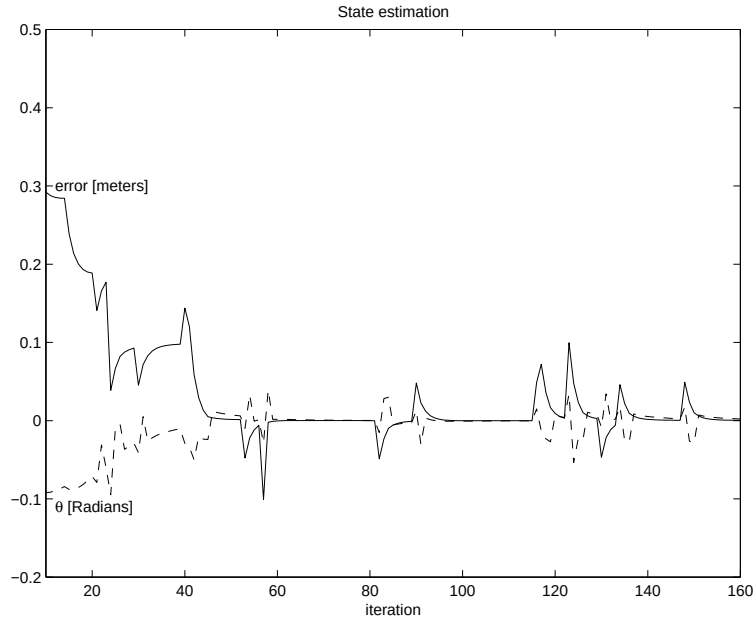

Figure 5.14: Data received from the Kalman filter. The figure shows the estimation of the position and of the angle supplied, with an elaboration of sonar data, by the KF.

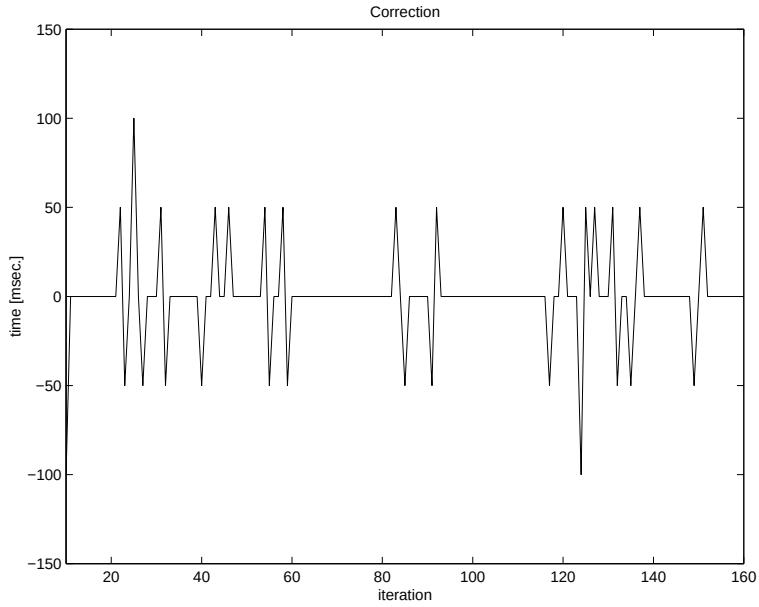

Figure 5.15: Control output sequence.

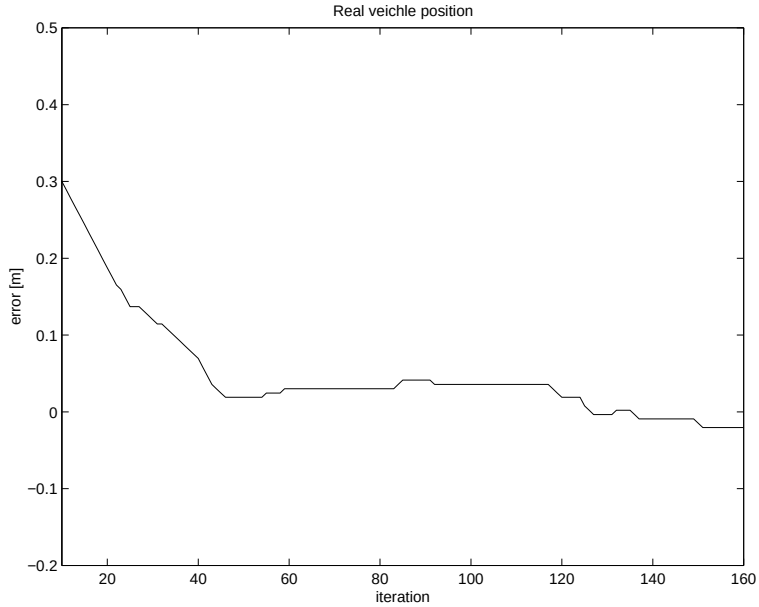

Figure 5.16: Real vehicle position.

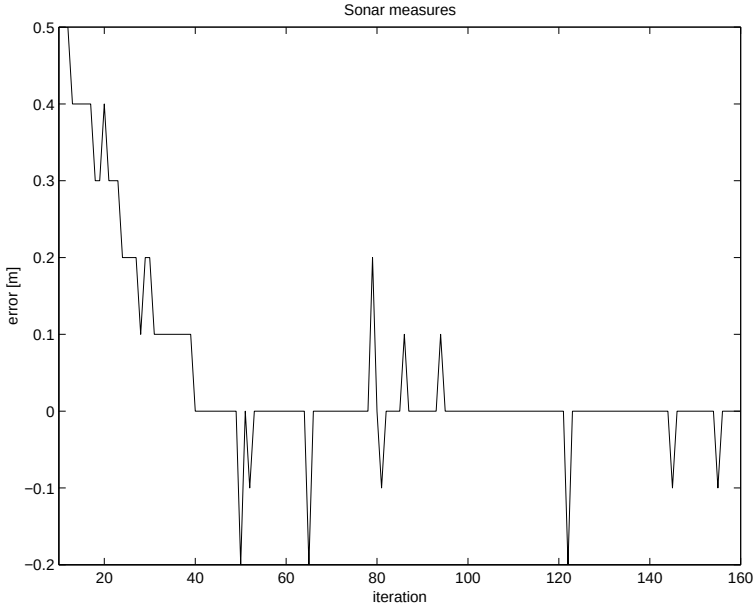

Figure 5.17: Sonar output (initial error 0.5 meters).

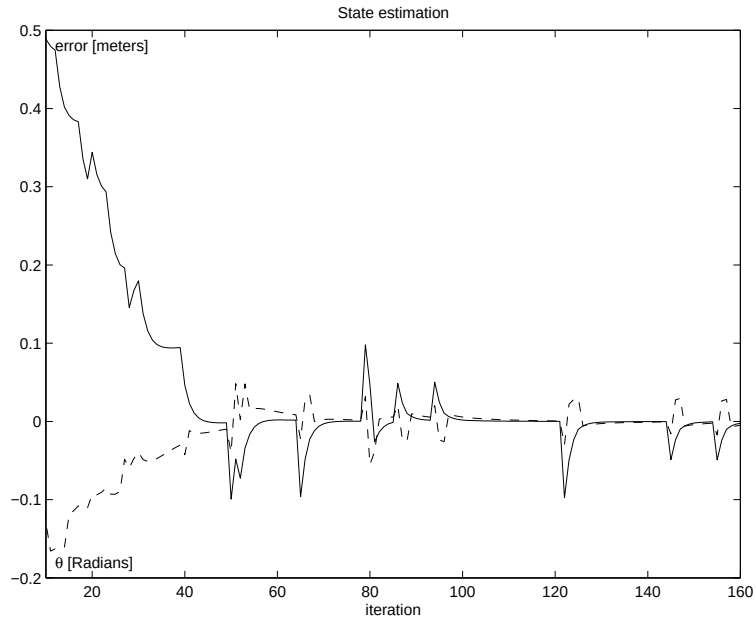

Figure 5.18: Data received from the Kalman filter.

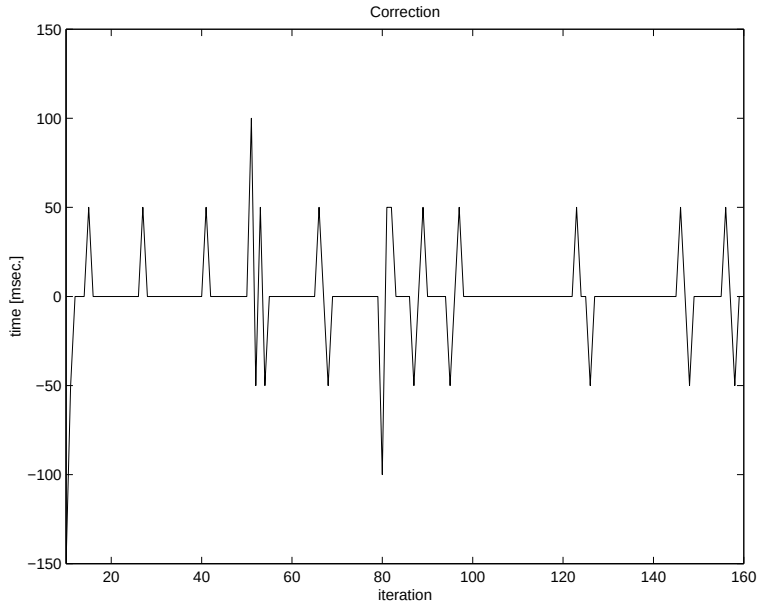

Figure 5.19: Control output sequence.

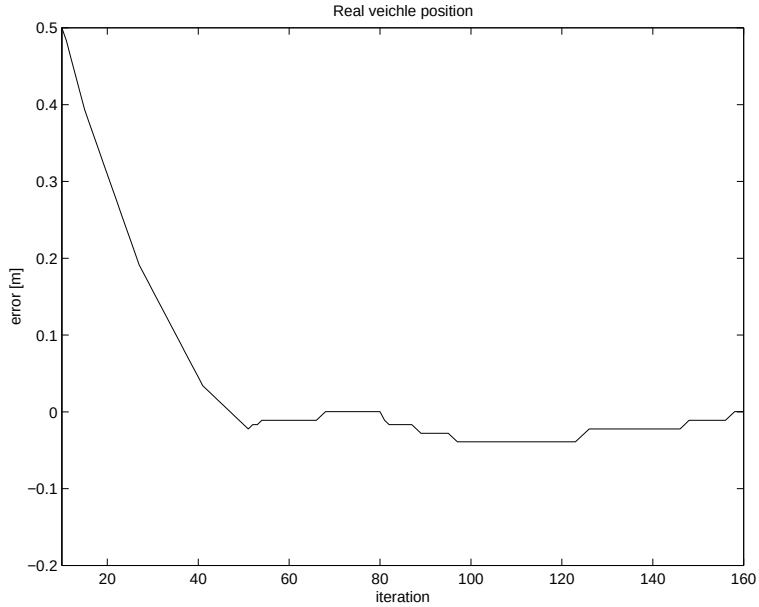

Figure 5.20: Real vehicle position.

Part III

Experimental Work

Chapter 6

The Software

The software for the control of the pool cleaner has been written in C++. It is composed of 7 files.

- mainfil.cpp
- robot.cpp & robot.h
- sensor.cpp & sensor.h
- serial.c & serial.h

In this chapter it will be explained how the software works, and how to use the classes for future upgrades.

6.1 Main program

The main program *mainfil.cpp* (see the appendix A.1 for the code) comprises the initialization of all the variables and the main loop which calls the control routines. The figure 6.1 shows the flow chart of the program.

The main program calls alternately the member function Run_Straight and Run_Cross, of the class PoolCleaner. At the first step it shows the size of the pool and the estimated time needed to cover the entire area. The loop can be interrupted pressing the "q" key, otherwise the program will stop until the whole area will be cleaned.

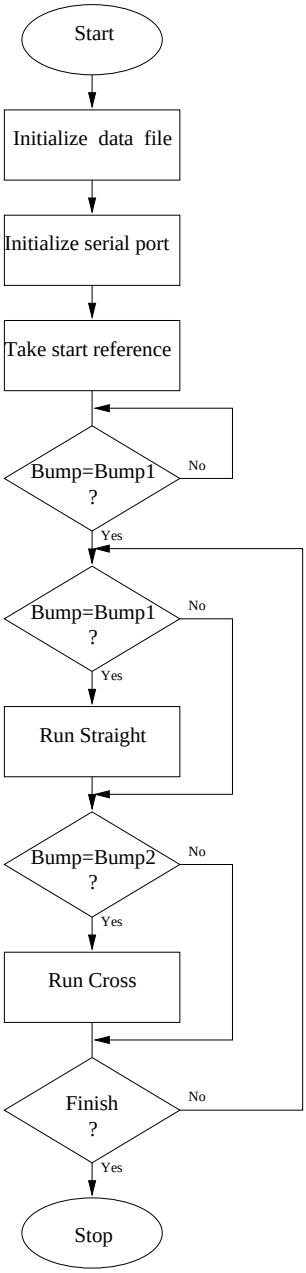

Figure 6.1: Main program flowchart.

6.2 PoolCleaner class

The PoolCleaner class is defined in the files *robot.cpp* and *robot.h* (see appendices A.2 and A.3 for the source code). The object PoolCleaner is composed of the private data *distance* and *reference*, where *distance* is the distance between the two walls perpendicular at the vehicle direction and *reference* is the reference distance to maintain between the pool cleaner and the parallel motion left wall. The class is also composed by the following member function:

- Run_Straight
- Run_Cross
- Distance_Edge
- Distance
- Angle
- Set_Dist_Ref
- Dec_Dist_Ref
- Stop_Left
- Stop_Right

The most important member functions are *Run_Straight* and *Run_Cross* the other functions are used by these.

6.2.1 Run_Straight

This function makes the vehicle to proceed straight holding the distance from the wall. The flowchart of this routine is showed by the figure 6.2. This function requires as input the object Sensor that contain data values from all the transducers of the pool cleaner. After the initialization of the local variables the routine makes a delay of one second to go away from the wall, then it starts a loop until the opposite wall is reached. Inside the loop the procedure makes the necessary operations to annul the position and the angle errors.

This routine takes the error values from the object sensor and makes a correction of heading in accord with this equation:

$$correction \quad = \quad -floor(\lambda(error + 10\delta))$$

Where λ is the variable gain:

$$\lambda = 200\,|MaxErr - |error|| + 1$$

Error and δ are respectively the position error and the angle error, MaxErr is the maximum error set at 2 meters.

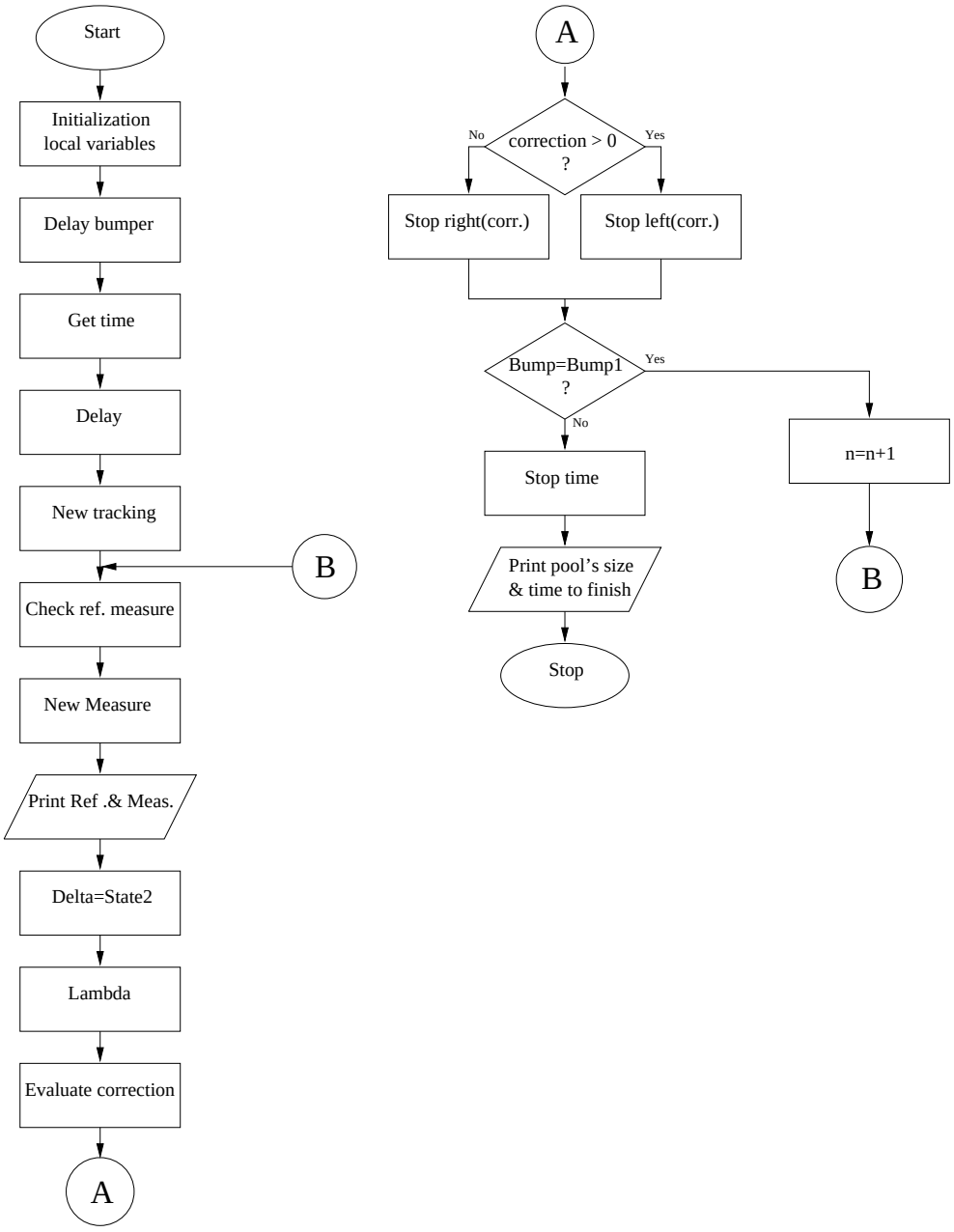

Figure 6.2: Run_Straight flowchart.

6.2.2 Run_Cross

This other routine makes the vehicle to proceed diminishing the distance from the parallel wall. Also this function requires as input the object Sensor. The reference value will change as shown in the figure 6.3.

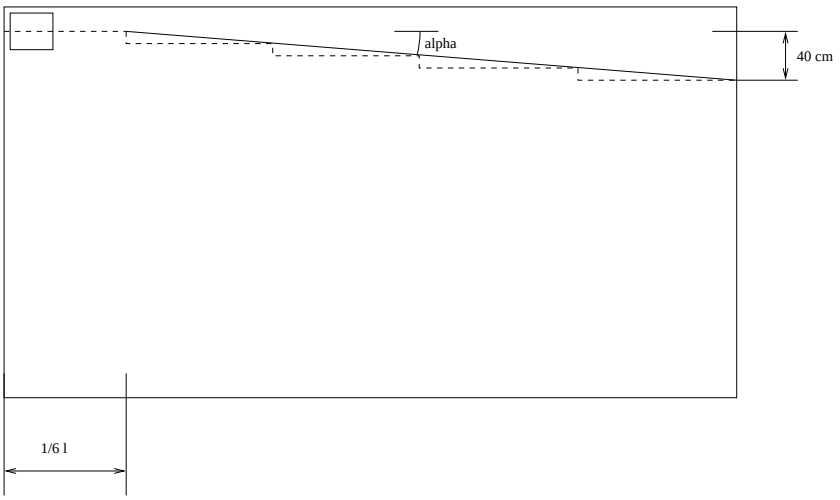

Figure 6.3: Reference value.

This routine leaves the robot proceed without correction for one sixth of the pool length to avoid the reflection of the sonar beam on the perpendicular wall, then it makes the vehicle turn of the angle alpha and starts a loop until the opposite wall is reached. Also inside this loop as like as in the *Run_Straight* procedure, the program corrects the position and the angle errors. When the pool cleaner reaches the wall the program checks the error position, if this error is little enough the reference distance will be decreased, otherwise it will be repeated the precedent path. The figure 6.4 shows the *Run_Cross* flow chart.

6.2.3 Distance_Edge and Distance

The *Distance_Edge* function stores in the private variable *distance* the respective value. It requires as input the time when the vehicle leaves the wall and the time when it reaches the opposite wall. To calculate this distance is applied the basic physics law $x = v \cdot t$. The *Distance* function gives the distance value.

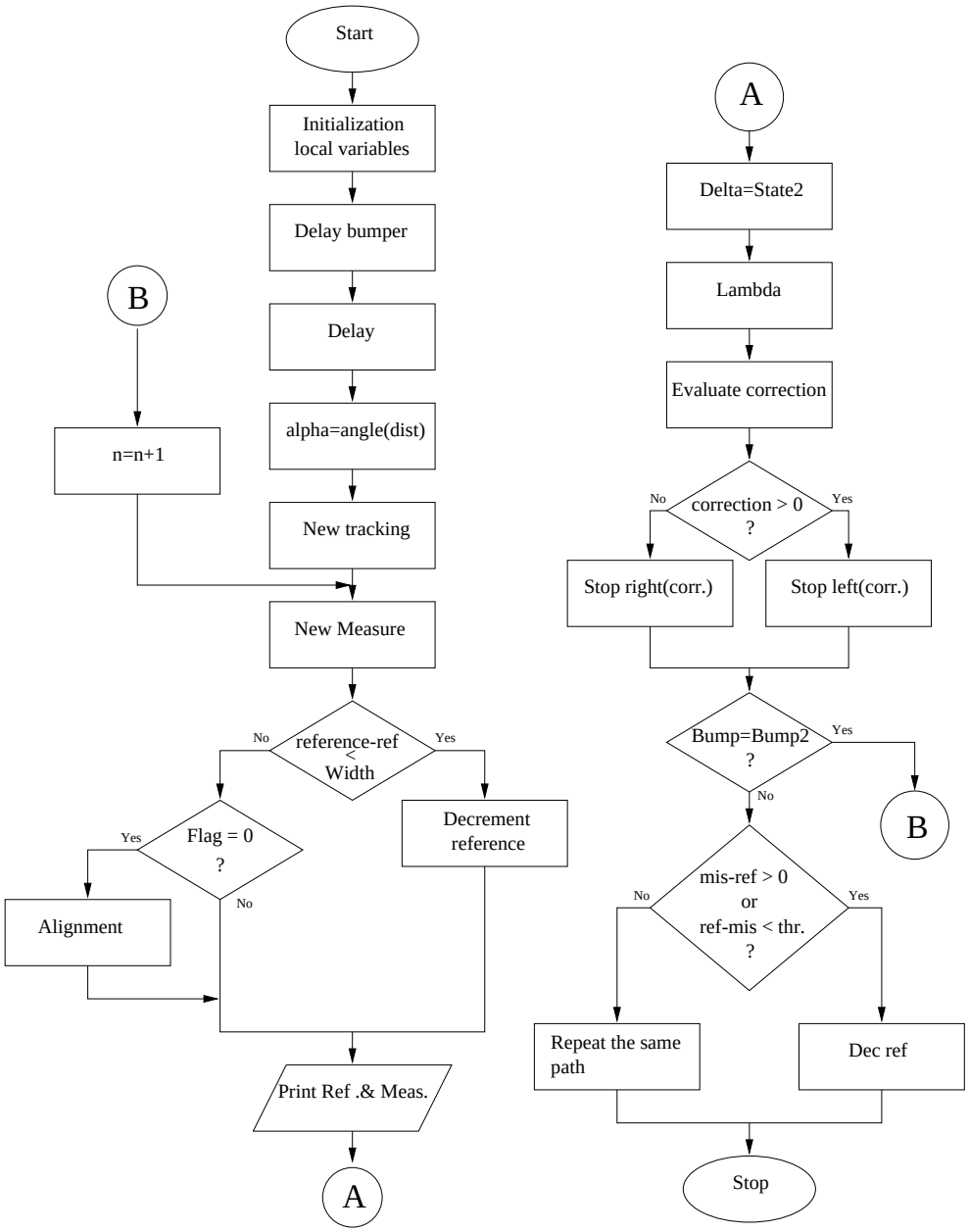

Figure 6.4: Run_Cross flowchart.

6.2.4 Set_Dist_Ref and Dec_Dist_ref

The *Set_Dist_Ref* function stores in the private variable *reference* the respective value. The *Dec_Dist_Ref* function decreases the *reference* value of 40 centimeters, the width of the pool cleaner.

6.2.5 Stop_Left and Stop_Right

These two function stop respectively the left and the right track of the robot for a time of milliseconds required as input. It is possible to regulate the steering rapport between the right and the left track with the constant R_L. These routines use as output port the parallel port.

6.3 Sensor class

The Sensor class is defined in the files *sensor.cpp* and *sensor.h* (see appendices A.4 and A.5 for the source code). The object sensor is composed of the private data *sonar*, *flag*, *width*, *buffer*, *error*, *state1* and *state2*. The variable *sonar* can assume the value 0 or 1 in conformity with which sonar is working; the variable *flag* is a check variable; the variable *width* contains the value of the pool width; the variable *buffer* and *error* are vectors which contain respectively the last BUFSIZE measures and the last BUFSIZE position errors; the variables *state1* and *state2* contain respectively the value of the position state variable and the angle state variable, estimated by the Kalman filter. The class is composed by the following member function:

- Read_Bumper
- Set_Sonar,Check_Measure and Sonar
- Getstring
- Set_Flag, Flag
- New_Sonar_Measure, Store_Err
- Value, Error and Clear_Vec
- Average, Filtering
- Take_Start_ref, Width
- State1, State2
- Get_Time

6.3.1 Read_Bumper

This member function reads from the parallel port which is the last bumper that has touched the wall. It returns the static value *bumper1* if the last bumper to touch the wall is the forward bumper, otherwise it returns *bumper2*. This function is called by the main program as a condition to start the subroutines *Run_Straight* and *Run_Cross* of the Pool cleaner class and also inside these functions.

6.3.2 Set_Sonar, Check_Measure and Sonar

The *Set_Sonar* member function changes the output address in according to the sonar we want to read the data.The *Check_Measure* function changes the input sonar in conformity with the measure carried out , if the measure is lower than MINTHRESHOLD this function calls the *Set_Sonar* member function to switch to the other sensor. The *Sonar* function gives which sonar is working at the moment.

6.3.3 Getstring

The *Getsring* member function returns the the distance measured by the active sonar. It also stores the data in the *sonardat.dat* file.

6.3.4 Set_Flag and Flag

The *Set_Flag* member function sets to 1 the *flag* private variable.This member function is called when the user presses the "q" or the "Q" key to break the program. The *Flag* member function returns the *flag* value.

6.3.5 New_Sonar_Measure and Store_Err

The *New_Sonar_Measure* and *store_Err* stores respectively the filtered measure in the *buffer* vector and the error position in the *error* vector.

6.3.6 Value, Error and Clear_Vec

The *Value* and *Error* functions require as input an integer n, they return respectively the nth element of the *buffer* vector and the nth element of the *error* vector. The member function *Clear_Vec* initializes the *buffer* vector to zero.

6.3.7 Average and Filtering

The *Average* member function returns the average value of the *buffer* elements. The *Filtering* function makes a low-pass filtering of the *buffer* elements.

6.3.8 Take_Start_ref and Width

The *Take_Start_ref* function stores in the *width* variable the pool width. The *Width* function returns the value of the *width* private variable.

6.3.9 State1 and State2

These two member functions return the values of the estimated state variables using the Kalman filtering theory. The *State1* function requires as input the number of the current iteration, the reference position value, and the last measure effected. The *State2* function requires the number of the current iteration, the value of the current control and the last value measured.

6.3.10 Get_time

This member function returns the current time in seconds taken from the system clock.

6.4 Serial routines

The serial routines are defined in the files *serial.c* and *serial.h* (Appendices A.6, A.7). This is a low level software to communicate by the serial port with the sonar device. The protocol used is the NMEA standard. It is an uniform interface standard for digital data exchange between electronic devices. The National Marine Electronics Association has developed a standardized data protocol that permit high speed communications among shipboard electronic devices. This software is written in C language, beside the initialization routines the most important procedure is the *getstring* which is the function called by the main program, it returns the sonar data stored in a buffer. This buffer is loaded by the interrupt routine *serial_isr*.

Chapter 7

Implementation

In this chapter some experiments are presented; after a theoretical approach to the pool cleaner problem, a set of tests are needful. The real environment of the pool is quite far from the world of the simulations, and since the aim of the project is to create a machine which must be sold, the experimental work is the most important and sensitive.

Some results about sonar tests are presented, together with some tests about the steering angle; after that, a set of tests with the machine are described and the results are given too.

7.1 Sonar tests

The experiments with the sonar system have been divided into two steps. The former is about the minimum distance measurable by the transducer; the latter regards the problems which occur when the machine moves along a non-parallel path with respect to the wall (see figure 7.1.a and 7.1.c).

As far as the minimum distance, the technical data tell us it is 0.8 meters, but several tests suggested us to consider it higher. As a matter of fact, for distances shorter than 0.8 meters, the sonar was effectively not able to give measurements at all. But we also found brief problems for measures between 0.8 and 1.3 meters; these measurements were sometimes unreliable, as we can see in the following table.

This fact suggested to use 3.0 meters as the minimum measurement distance, also in order to have a satisfactory safety range. Looking at the problem of the orientation of the sensor, tests have been necessary in order to determine the limit in the angle of inclination with respect to the wall. Since the sound beam of the sonar is about 9 degrees, for little θ, measures remain reliable. Instead, for values of the angle over 15 degrees, it becomes arduous to use the sensor, and this is the reason why we choose this value as the maximum limit for the steering angle of the robot.

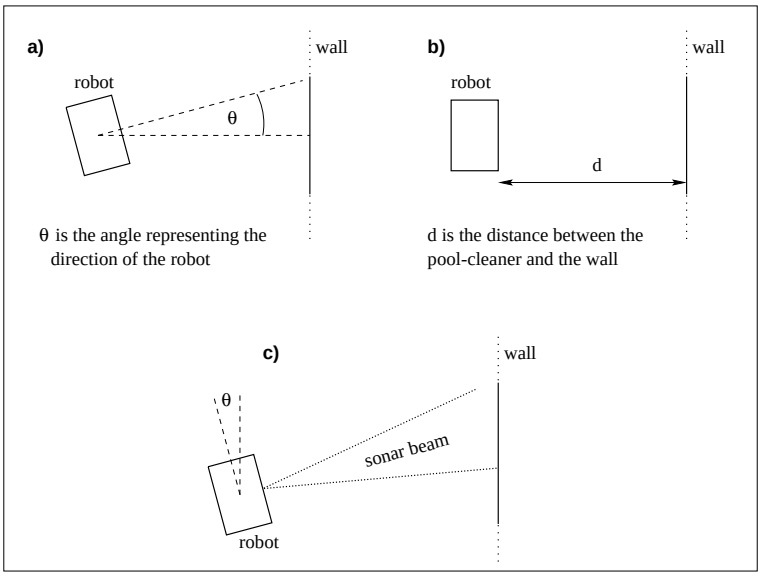

Figure 7.1: Tests with the sonar system: a) and c) regard the problem with the sonar when the robot is not aligned to the wall; b) shows the problem of short distances.

7.2 Real steering angle relation

It has been mentioned that there might be disturbances in the pool cleaner motion, especially as far as the steering angle evaluation. A theoretical relation between angle and steering time has been given too. Considering also experimental work, we found this relation was not so precise because of too much environmental noise acting on the machine. During our first few tests we figured out the vehicle steering did not follow a linear law. It is easy to imagine that for the first few instants the stopped track will slip for a while, then it will start to adhere more and more to the floor enabling the vehicle to turn. Thus we decided to use an experimental relation, more satisfactory, than the linear one, in our real applications. Our first step was to have several tests in order to find steering-angle values for different steering time, then to obtain a relation that approximates the precedent values.

The experimental work brought the following results:

real distance [m]	*sonar measures [m]*	*real distance [m]*	*sonar measures [m]*	*real distance [m]*	*sonar measures [m]*
	-		1		1.1
	-		0.9		1.1
	-		0.9		1.3
	0.9		1		1.2
0.8	1	1	1.2	1.2	1.1
	-		1		1.1
	-		0.9		1
	0.7		-		1.1
	...		...		...
	1		1		1.2
	-		0.9		1.3
	-		-		1.3
	0.9		1.1		1.2
0.9	-	1.1	1.1	1.3	1.2
	0.8		-		1.3
	-		1.1		1.1
	0.7		1		1.3
	...		...		...

Table 7.1: Sonar measures at short distances. The symbol - indicates no readings.

Gathering this data in a graphic we obtain:

Now we can fit these data with a second degrees polynomial function in a least-squares sense, finding an approximate law that links the robot steering with the time one track stops. A second degrees polynomial function is accurate enough for this application. The polynomial function that fit our data is:

$$\theta(t) \quad = \quad 8.17 \cdot 10^{-6} t^2 + .0022t \tag{7.1}$$

Where θ is the steering angle in degrees and t is the time in milliseconds. This polynomial function fits the data as shown in the figure (7.3).

steering time [msec.]	*angle [deg.]*	*angle [Radians]*	*steering direction [L/R]*	*steering time [msec.]*	*angle [deg.]*	*angle [Radians]*	*steering direction [L/R]*
300	1.66	0.0289	L	300	1.78	0.0311	R
300	1.54	0.0268	L	300	1.51	0.0263	R
300	1.80	0.0313	L	300	1.78	0.0311	R
300	1.20	0.0209	L	300	1.23	0.0215	R
300	1.80	0.0313	L	300	1.77	0.0311	R
400	1.70	0.0296	L	400	2.47	0.0431	R
400	2.50	0.0436	L	400	1.78	0.0311	R
400	2.35	0.0409	L	400	2.20	0.0383	R
400	1.80	0.0313	L	400	2.47	0.0431	R
400	1.80	0.0313	L	400	1.78	0.0311	R
500	3.20	0.0558	L	500	2.75	0.0480	R
500	2.65	0.0462	L	500	3.02	0.0527	R
500	2.70	0.0470	L	500	2.75	0.0480	R
500	3.00	0.0523	L	500	3.02	0.0527	R
500	3.00	0.0523	L	500	3.02	0.0527	R
600	3.12	0.0544	L	600	3.71	0.0647	R
600	3.67	0.0640	L	600	5.48	0.0956	R
600	4.44	0.0774	L	600	4.39	0.0766	R
600	4.33	0.0755	L	600	4.67	0.0815	R
600	4.71	0.0821	L	600	4.39	0.0766	R
700	5.55	0.0968	L	700	4.94	0.0862	R
700	4.97	0.0866	L	700	6.59	0.1150	R
700	6.11	0.1065	L	700	6.03	0.1052	R
700	6.25	0.1090	L	700	5.49	0.0956	R
700	5.12	0.0893	L	700	6.03	0.1052	R

Table 7.2: Stop Left and Right track.

7.3 Pool cleaner test

Now we are going to present the results of our tests in a rectangular 25 by 8 meters swimming pool. We will show the data stored by the control software during the cleaning of the pool. For the first test we have positioned the robot in the middle of the pool, and we let it work along the short side. The second test regards a partial cleaning section along the long side. The final test is a complete cleaning section of the 25 by 8 meters pool. In the figures the dashed lines represent the reference path.

7.3.1 Short side test

The data stored are shown in the figure 7.4. The vehicle has proceeded along the shortest side of the swimming pool decreasing its distance to the wall. The pool cleaner started with a distance from the wall of about 13.7 meters. It is noticeable that, between the iteration 300 and 400, the robot has repeated the same path because of a too large position error. Since the length of the pool is 8 meters, the speed of the vehicle is 0.3 meters

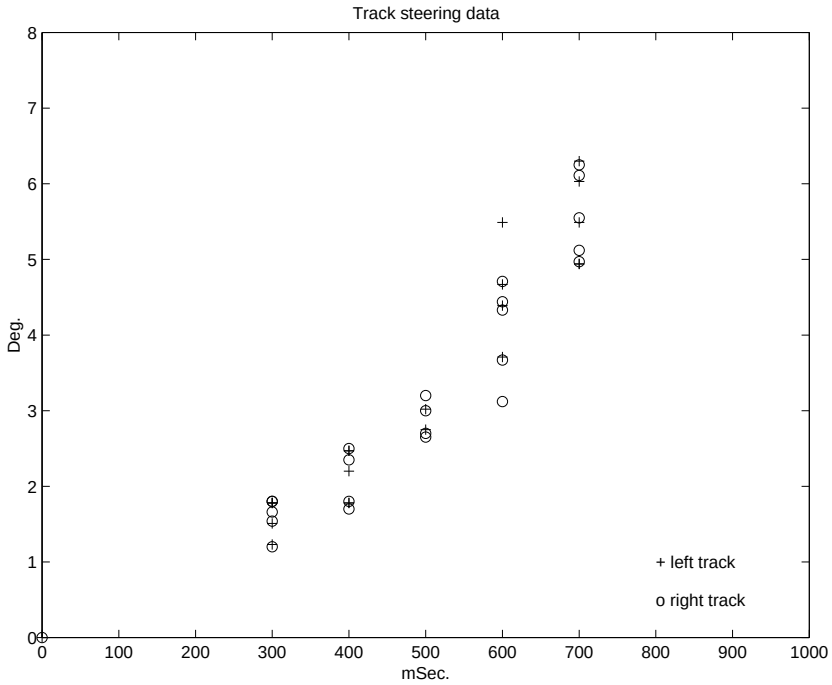

Figure 7.2: Measured data.

per second and the sample time of the sonar is 0.5 seconds, it means that the control software stores about 50 samples during each passage. The final result is that the robot covered more than 96% of the entire surface. The next step is to make the pool cleaner able to proceed along the longest side of the swimming-pool; this is necessary because the sonar doesn't supply the necessary resolution for distances larger than 20 meters. Thus, in order to make the vehicle able to cover the whole surface by itself, this strategy has been used. This situation is more difficult than the previous one, because even a small angle error can produce a very big position error if the control program doesn't correct it readily.

7.3.2 Long side test

The machine started 7 meters far from the target and proceeded until the measurement was about 4 meters. The data stored are represented in the figure 7.5. As we can see, at the beginning of the test a bothersome big oscillation of the system, due to a set of unreliable measures from the sonar, is present. The stabilization of the system is reached after a couple of side-to-side actions, but it is clear this kind of problem is totally unforeseeable; it depends on wrong sonar measures which are not corrigible by any filtering technique.

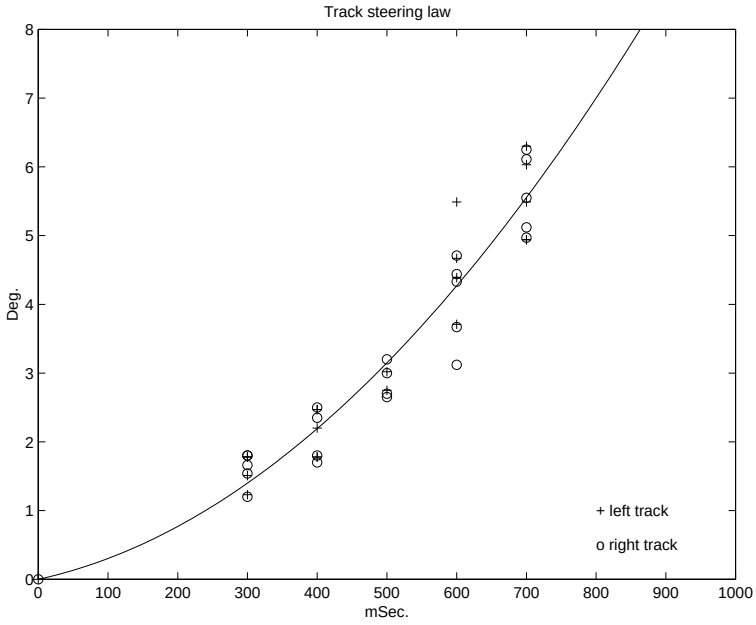

Figure 7.3: Polynomial interpolation.

The fact that the stabilization is reached after so much time, together with the type of oscillations which are present in the rest of the test, are anyway reducible with a new adjustment of the control parameters. The result of this operation is directly used in the final complete test.

7.3.3 Final test

This is a complete cleaning section, in which the pool cleaner starts from one corner of the pool, after it has taken the first reference measure, e.g. the width of the pool (it is 8 meters, but in the figure 7.6 we see about 7.5 meters, because the machine has a width of 0.5 meters); then it proceeds in its work until the measure of the distance to the wall becomes 3.0 meters. At this point the control switches the sonar and the machine goes on until the end of the work using the other sonar. The data stored are represented in the figure 7.6. Making a comparison with the long side partial test, this case shows more limited oscillations around the ideal path. The first observation regards the characteristic of the real covered way between the iterations 2700 and 3000. It is quite different compared with the rest of the real path, and this is due to the change of the sonar: since the measures are not always reliable at the beginning of the working section of the sonar, the control has to diminish the velocity with which the distance to the wall is reduced. This means that the *Run_Cross* function was not called by the control for a

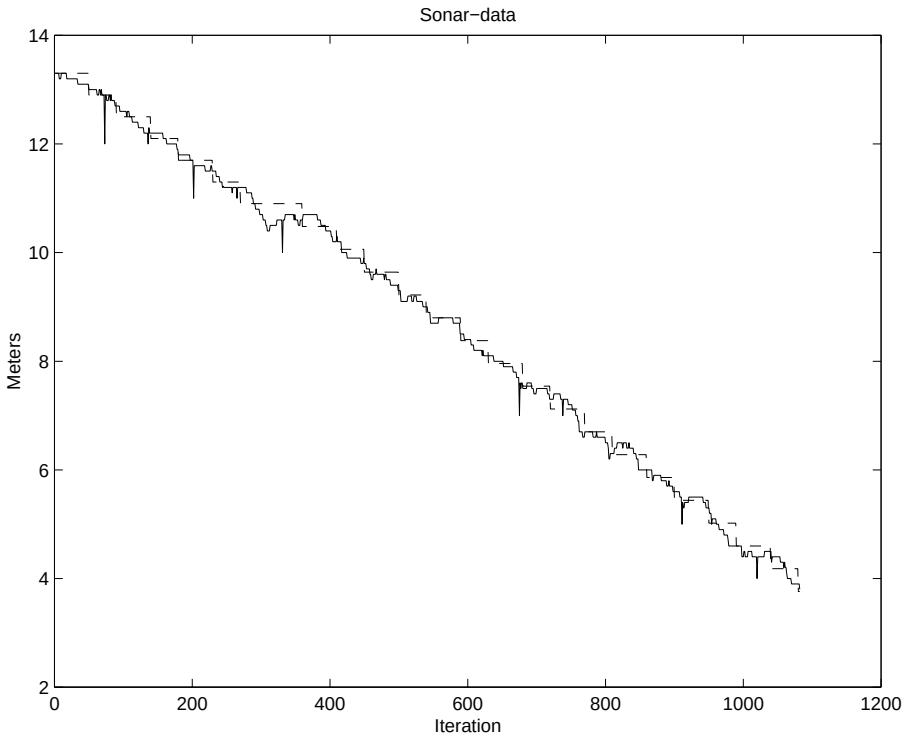

Figure 7.4: Pool-cleaner test along the short side.

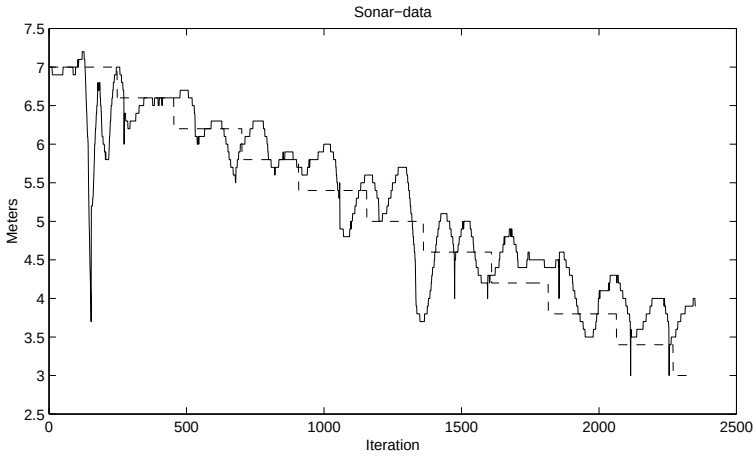

Figure 7.5: Pool-cleaner test along the long side.

couple of for-and-back runnings, but it depends only by the time needed by the sonar for becoming steady.

Another observation regards the real path around the iteration 900: a

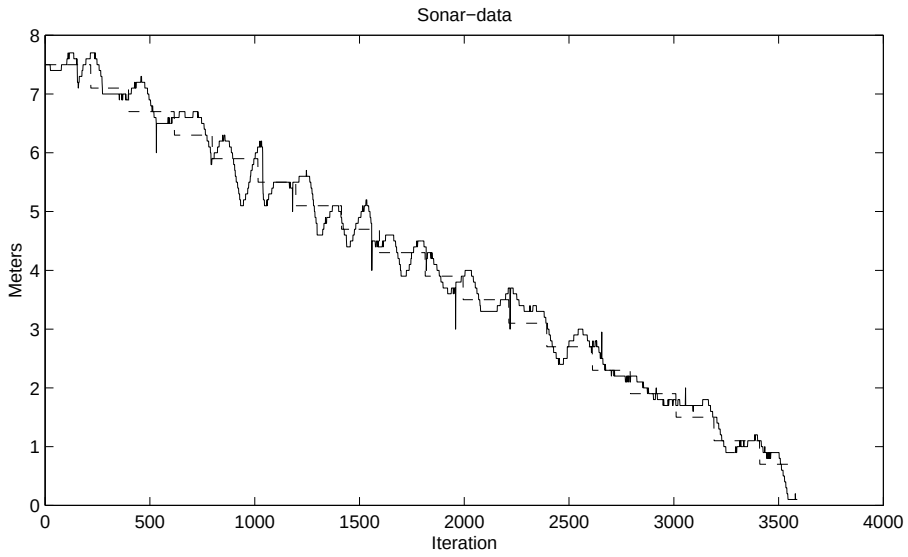

Figure 7.6: Pool-cleaner test of the whole swimming pool.

too fast approach to the new reference distance, caused again by wrong sonar measures, brought the robot far away from the ideal path. This fact induced the control to make a strong correction, but with excellent results: the normal condition was reached in few tens of iterations.

The machine completed the cleaning section of the entire swimming pool in about 3600 iterations, and since each iteration in the graphic is 1 sec., the total time was about 1 hour. The cleaning time calculated at the beginning of the pool cleaner task by the software was about 55 min., and considering that the machine lost some time in the change of the sonar, and other time for the correction of a path error, this estimated time was almost completely respected.

Observations. Now, a consideration on the couple sonar-machine is necessary. The problems we had with this kind of sensor have actually been estimated by our theoretical study on the sonar system and on underwater sound. Obviously, no one should expect much more from these sonars than they physically can provide; their limits are well known, but just this matter puts in evidence the potential resources of the pool cleaner. The control system is able to follow an ideal path and, at the same time, to correct wrong ways chosen because of environmental troubles (wrong sonar data included); how the error of the robot could be restricted, and the real time by which this error could be correct, depend on several matters; lots of tests are still necessary in order to set in a definitive way all the parameters which are at stake.

Chapter 8

Conclusions

8.1 Summary

The objective was to create an autonomous pool cleaning machine in order to put it directly on the world market with a competitive price. The first step of this project was to evaluate the sonars as sensors for this application, and to study both in a theoretical way and in an experimental one their behavior in the swimming pool. After having determined their limits and advantages, we started a theoretical study about the whole system, considering some possible solutions for the integration between the robot and the sensors. The experimental part brought us interesting results; it was divided into some tests, only a few of which are reported in the third part of this thesis. Some different pools were used, but always with the same characteristics.

8.2 Results

In this thesis we have reported on six months of studies and tests. The autonomous pool cleaner is almost ready for commercialization; many of the achieved results in our tests are very promising, especially considering that the bottom of the swimming pool can be almost completely covered in a realistically acceptable time. In order to reach the real goal of this project several other tests are necessary, especially to determine appropiate values for all the variables of the program; here and now the software should be tested in a larger variety of swimming pools, because even a little change in the material with which the ground is composed, or different shape of the walls, may call for changes in several parameters.

Some considerations may be done about the cleaning characteristics: it is possible to set, at the beginning of the work in the pool, the accuracy we want the robot to use for the cleaning section. A high accuracy means that the machine goes in for-and-back diminishing the reference distance to a smaller value compared to the low accuracy strategy. The time needed

to complete the entire cleaning process increases, but with it, the level of cleanness increases too. In our tests the strategy used was mostly the low accuracy one, but this was due to the fact that we always had really a few time for the experiments. Anyway, this means that the total covered area may easily be increased.

An excellent result is about the capability of the robot to estimate the total time it needs in order to complete the whole cleaning section, and especially the fact that it mostly respects it.

A conclusion that can be made is that sonars definitely have an use in this application, as long as they are not expected to come up with information about the environment that they just can not obtain. They are suitable for environments that are not too cluttered with objects having very advanced geometries. It may be arduous to use these sensors in the non-regular pool as we did in the rectangular one; in these more complicated environments, their use may be a complement to other sensors. One of the major problems that arose during the work was that the updating of new sonar readings was far too slow to obtain enough information about the environment, i.e. the detection of a change in the distance was still too slow.

The fact that there must be a restriction on the price means obviously a limitation in the real development of the machine; but just considering the real aim of the project, the creation of a product to be sold, all the results of this work are absolutely satisfactory.

8.3 Future work

We have concentrated our work during this six months on one particular kind of evaluation, adequacy evaluation of on market or near market products. Although we feel that substantial progress has been made towards the overall goal of develop an autonomous pool cleaner, we are aware that much remains to be done. Obvious directions for future work include making the robot reliable as much as possible in rectangular swimming pools and extending the use of the pool cleaner in the non regular pools. The experience acquired during these months, described in this report, should be a good background for future developments.

Appendix A

The source code

In this appendix it will be showed the source code of the control program.

A.1 mainfil.cpp

```
//      file Mainfil.cpp
//      Main code
#include <dos.h>
#include <stdio.h>
#include <stdlib.h>
#include <bios.h>
#include <iostream.h>
#include <conio.h>
#include"robot.h"
#include"sensor.h"

// Declaration of external C functions
extern "C"
        {
        void install();
        int sgetch();
        void cleanup();
        }

void main()
{
char key;
PoolCleaner rob;
Sensor data;

// Initialization
```

```
// variables
data.Init_State();
data.Store();
data.Set_Sonar(0);

install();

cout<<endl<<"Put the pool cleaner to the center of the wall and press a key..."
    <<endl;
while(!kbhit());

cout<<"starting Take_start_ref....."<<endl;

data.Take_start_ref();

cout<<" Ref dist: "<<data.Width()<<endl;

rob.Set_Dist_Ref(data.Width());

cout<<"Put the poolcleaner on the pool edge...."<<endl;

while (data.Read_Bumper()!=bumper1);


cout<<"Start cleaning...."<<endl;

data=rob.Run_Straight(data);

cout<<endl<<"Length pool: "<<rob.Distance()<<endl;
cout<<"Width pool: "<<data.Width()<<endl;


// Execution loop

 while(1)
        {
        if (data.Flag()==1)
                {

                cout<<"User Break !"<<endl;
                break;

                }
```

```
        if (kbhit())
         {

         key=getch();
         if (key=='q' || key=='Q')
                {

                cout<<"User Break !"<<endl;
                break;

                }

         }

        if ( data.Read_Bumper()==bumper1)
                {

                cout<<"Bumper1 "<<endl;
                while((sgetch()) !=-1);
                        data=rob.Run_Straight(data);

                }
        if (data.Read_Bumper()==bumper2)
                {

                cout<<"Bumper2 "<<endl;
                while((sgetch()) !=-1);
                        data=rob.Run_Cross(data);

                }
        }

        cleanup();
}
```

A.2 robot.cpp

```
//      file robot.cpp
//      Definition robot class member function

#include<iostream.h>
#include<dos.h>
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#include<conio.h>
#include"robot.h"
#include"sensor.h"


// ***************************************************************************

void PoolCleaner::Dec_Dist_Ref()
  {

  reference-=WIDTH;

  }

// ***************************************************************************

void PoolCleaner::Set_Dist_Ref(float dist)
  {

  reference=dist;

  }

// ***************************************************************************

void PoolCleaner::Distance_Edge(int startt,int stopt)
  {

  distance=(stopt-startt)*SPEED;

  }

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

```
float PoolCleaner::Angle(float dist)
  {

  float alpha;
  alpha=atan2(WIDTH,dist);
  return alpha;

  }

// ***************************************************************************

void PoolCleaner::Stop_Left(Sensor dat,int mSec)
  {

  if (dat.Sonar()==0)
        {

        cout<<"    Stop left track for "<<mSec<<" mSec."<<endl;

        outport(port_number,stop_left);
        delay(mSec);
        outport(port_number,start);

        }

  if (dat.Sonar()==1)
        {

        cout<<"    Stop left track for "<<mSec<<" mSec."<<endl;

        outport(port_number,stop_left+4);
        delay(mSec);
        outport(port_number,start+4);

        }
  }

// ***************************************************************************


void PoolCleaner::Stop_Right(Sensor dat,int mSec)
  {

  if (dat.Sonar()==0)
```

```
        {

        cout<<"    Stop right track for "<<R_L*mSec<<" mSec."<<endl;

        outport(port_number,stop_right);
        delay(R_L*mSec);
        outport(port_number,start);

        }

  if (dat.Sonar()==1)
        {

        cout<<"    Stop right track for "<<R_L*mSec<<" mSec."<<endl;

        outport(port_number,stop_right+4);
        delay(R_L*mSec);
        outport(port_number,start+4);

        }

  }

// ***************************************************************************

float PoolCleaner::Distance()
  {

  return distance;

  }

// ***************************************************************************

Sensor PoolCleaner::Run_Straight(Sensor dat)
  {


 // Local variables
  int i,n;
  long int startt,stopt;
  float dist;
  float error[2];
  float total=0;
```

```
float lambda;
int timetogo;
float delta=0;
float correction;
div_t x,y;
char key;

dat.Init_State();
correction=0;
n=0;

delay(DELAY);
startt=dat.Get_time();

delay(1000);
dat.Check_Measure(reference);
dat.New_Data_Tracking();

while(dat.Read_Bumper()==bumper1)
      {

      if (kbhit())
              {
              key=getch();
              if(key=='q' || key=='Q')
                      {

                      dat.Set_Flag();
                      return dat;

                      }

      }



      dat.New_Sonar_Measure(n,reference,correction);
      dat.New_Sonar_Measure(n,reference,correction);

      cout.precision(1);
      cout<<"Reference dist: "<<reference<<"   Edge dist: "<<dat.Value(0);
      dat.Store_Err(dat.Value(0)-reference);
      error[0]=dat.Error(0);
      error[1]=dat.Error(BUFSIZE-1);
```

```
if (fabs(error[0])>THRESHOLD)
        {

        delta=dat.State2(n,correction,dat.Value(0));
        cout<<" Vehicle angle estimation :  "<<delta*57.32<<endl;
        lambda=GAIN*fabs(MAXERR-fabs(error[0]))+1;
        correction=-floor(lambda*(error[0]+10*delta));

        if (fabs(correction)<25)

                correction=0;

        else if ((correction > 0) && (correction < 125))
                correction=100;

        else if ((correction > 0) && (correction < 225))
                correction=200;

        else if ((correction > 0) && (correction < 325))
                correction=300;


        else if ((correction < 0) && (correction > -125))
                correction=-100;

        else if ((correction < 0) && (correction > -225))
                correction=-200;

        else if ((correction < 0) && (correction > -325))
                correction=-300;


        if ((((total+correction)>MAXDELAY) && (delta<0)) ||
              (((total+correction)<-MAXDELAY) && (delta>0)))
                total=0;

        if (fabs(total+correction)<MAXDELAY)
                {

                if (correction<0)
                        {

                        Stop_Left(dat,-correction);
```

```
                        }
                    else if (correction>0)
                        {

                        Stop_Right(dat,correction);

                        }

                    else if (correction==0)
                        cout<<"    No Correction needed."<<endl;

                    total +=correction;

                    }
                else cout<<"    No correction (overmax)."<<endl;

                }
            else
                {

                cout<<"    No correction (thresh)."<<endl;

                }
            n++;

            }
            stopt=dat.Get_time();
            Distance_Edge(startt,stopt);


            timetogo=floor((stopt-startt)*2)*(dat.Value(BUFSIZE-1)/WIDTH);

            x=div(timetogo,3600);
            y=div(x.rem,60);
            cout<<"Time to finish: "<<x.quot<<":"<<y.quot<<":"<<y.rem<<endl;
            return dat;

}

// ***************************************************************************

Sensor PoolCleaner::Run_Cross(Sensor dat)
  {
```

```
 int n,i,alignment,flag;

 long int a;

 float dist;
 float error[2];
 float total;
 float alpha;
 float lambda;
 float delta;
 float correction;
 float ref,ref1;

 char key;

 n=0;
 delay(DELAY);

 cout<<"Delay to leave the wall..."<<endl;

 delay(floor((distance/(6*SPEED)*1000)));

 alpha=Angle(distance*5/6);

 cout<<endl<<"  alpha   ... "<<alpha<<endl;

 flag=0;
 correction=0;
 dat.Init_State();


alignment=floor((-.0022+sqrt(.00000484+0.00003268*alpha*57.32))/.00001634);
 cout<<"Alignment...";
 Stop_Left(dat,alignment);

 total=0;
 dat.New_Data_Tracking();
 ref=reference;
 while(dat.Read_Bumper()==bumper2)
       {

               if (kbhit())
               {
```

```
        key=getch();
        if(key=='q' || key=='Q')
                {

                dat.Set_Flag();
                //break;
                return dat;

                }
        }


dat.New_Sonar_Measure(n,reference,correction);
dat.New_Sonar_Measure(n,reference,correction);


if ((reference-ref)<WIDTH)
{

        ref-=(SPEED*sin(alpha));


        dat.Store_Err(dat.Value(0)-ref);

}
else
        {

        if (flag==0)
                {

                Stop_Right(dat,alignment);
                total-=alignment;
                flag=1;

                }

        dat.Store_Err(dat.Value(0)-(reference-WIDTH));

        }


error[0]=dat.Error(0);
error[1]=dat.Error(BUFSIZE-1);
```

```
cout.precision(1);
cout<<"Reference dist: "<<ref<<"    Edge distance: "<<dat.Value(0);


if (fabs(dat.Error(0))>THRESHOLD)
        {

        delta=dat.State2(n,correction,dat.Value(0));
        cout<<" Vehicle angle estimation :  "<<delta*57.32<<endl;
        lambda=GAIN*fabs(MAXERR-fabs(error[0]))+1;
        correction=-floor(lambda*(error[0]+10*(delta-alpha)));


        if (fabs(correction)<25)

                correction=0;

        else if ((correction > 0) && (correction < 125))
                correction=100;

        else if ((correction > 0) && (correction < 225))
                correction=200;

        else if ((correction > 0) && (correction < 325))
                correction=300;


        else if ((correction < 0) && (correction > -125))
                correction=-100;

        else if ((correction < 0) && (correction > -225))
                correction=-200;

        else if ((correction < 0) && (correction > -325))
                correction=-300;


        if ((((total+correction)>MAXDELAY) && (delta<0)) ||
            (((total+correction)<-MAXDELAY) && (delta>0)))

                total=0;

        if (fabs(total+correction)<MAXDELAY)
```

```
                        {

                        if (correction<0)
                                {

                                Stop_Left(dat,-correction);

                                }

                        else if (correction>0)
                                {

                                Stop_Right(dat,correction);

                                }

                        else if (correction==0)
                                cout<<"    No correction needed."<<endl;

                        total +=correction;

                        }
                else cout<<"    No correction(overmax)."<<endl;
                }
        else

                {

                cout<<"    No correction(thresh)."<<endl;

                }
        n++;
        }
  if ((dat.Average()-ref)>0 && (dat.Average()-ref)<WIDTH)

        Dec_Dist_Ref();

  else
        cout<<"Repeating the same path....  "<<endl;

  return dat;

}
// ***************************************************************************
```

A.3 robot.h

```
//      file robot.h
//      Definition robot class

#ifndef ROBOT.H
#define ROBOT.H

#include"sensor.h"

//      Constant definition

#define SPEED           .3      // Speed of Poolcleaner
#define WIDTH           .4      // Width of Poolcleaner
#define DELAY           4000    // Time delay to reverts the run
#define MAXDELAY        10000   // Maximum cumulative delay
#define THRESHOLD       .05     // Error threshold
#define MAXERR          2       // Maximum error
#define GAIN            200     // Static gain
#define R_L             1.5     // Steering rate between right and left track
#define port_number     0x378   // Parallel output address
#define stop            0x03    // Stop both tracks word
#define stop_left       0x02    // Stop left track word
#define stop_right      0x01    // Stop right track word
#define start           0x00    // Run both tracks word


class PoolCleaner
{
//      Definition private data

  float         distance;
  float         reference;

//      Definition member function
  public:

  void          Distance_Edge(int startt,int stopt);
  float         Angle(float dist);
  float         Distance();
  Sensor        Run_Straight(Sensor dat);
  Sensor        Run_Cross(Sensor dat);
  void          Dec_Dist_Ref();
```

```
  void          Set_Dist_Ref(float dist);
  void          Stop_Left(Sensor dat,int mSec);
  void          Stop_Right(Sensor dat,int mSec);

};
#endif
```

A.4 sensor.cpp

```
//      file sensor.cpp
//      Definition sensor class member function

#include <dos.h>
#include <stdio.h>
#include <stdlib.h>
#include <bios.h>
#include <iostream.h>
#include <fstream.h>
#include "sensor.h"
#include <conio.h>

// Declaration of external C functions
extern "C"
{
 int sgetch(void);
 float getstring(void);
}

// ***************************************************************************

float Sensor::Getstring()

  {

  float a;
  ofstream storedata;
  a=getstring();
  if (sonar==1)
        a=width-a;

  storedata.open("sonardat.dat",ios::app,0);
  storedata<<a<<endl<<state2<<endl;
   // NOTA
  storedata.close();
  return a;

}

// ***************************************************************************

void Sensor::Check_Measure(float ref)
```

```
   {

   if ((sonar==0) && (Average()<MINTHRESHOLD))
        {

        Set_Sonar(1);
        outport(port_number,start+4);
        delay(2000);
        Take_start_ref(width);
        }
   }

// ***************************************************************************

void Sensor::Set_Sonar(int i)

   {

   cout<<endl<<"Switching to sonar "<<i<<endl;
   if (i==0)
        {

        sonar=0;
        outport(port_number,0x00);

        }

   if (i==1)
        {

        sonar=1;
        outport(port_number,0x04);

        }

   }

// ***************************************************************************

int Sensor::Sonar()
   {

   return sonar;
```

```
   }

// ***************************************************************************

void Sensor::Store()
   {

   ofstream storedata;
   storedata.open("sonardat.dat",ios::out,0);

   }

// ***************************************************************************

unsigned char Sensor::Read_Bumper()
  {

  unsigned char inbump = NULL;
  static bumper;


  inbump=inp(0x379);
  if (inbump==0xf8)
        bumper=bumper1;
  if (inbump==0x38)
        bumper=bumper2;
  if (bumper==NULL)
        return NULL;

  return bumper;

  }

// ***************************************************************************

long int Sensor::Get_time()
  {

  struct time t;

  gettime(&t);
  ttime=t.ti_sec+t.ti_min*60+t.ti_hour*3600;
  return ttime;
```

```
  }

// ***************************************************************************

float Sensor::Width()
{

        return width;

}

// ***************************************************************************

int Sensor::Flag()
{

        return flag;

}

// ***************************************************************************

void Sensor::Set_Flag()
{

        flag=1;

}

// ***************************************************************************

int Sensor::Take_start_ref()
{


        Clear_vec();
        cout<<endl<<"Aquiring start reference....."<<endl;
        for (int i=0;i <BUFSIZE;i++)
                {

                buffer[i]=Getstring();
                error[i]=0;

                }
```

```
        width=Average();

        return 0;

}

// ***************************************************************************

int Sensor::Take_start_ref(float dist)
{
        do
        {
        for (int i=0;i <BUFSIZE;i++)
                buffer[i]=Getstring();
        } while (Average()<4 && Average()>2);
        width=dist;
        state1=Average();

        return 0;

}

// ***************************************************************************

void Sensor::Clear_vec()
{

        for (int i=0;i <BUFSIZE;i++)
                buffer[i]=0;

}

// ***************************************************************************

float Sensor::Average()
{

        float a=0;
        for (int i=0;i <BUFSIZE;i++)
                a+=buffer[i];

        a=a/BUFSIZE;
```

```
        return(a);

}

// ***************************************************************************

int Sensor::New_Sonar_Measure(int n, float ref, int contr)
{


        for(int i=1; i<BUFSIZE; i++)
                buffer[i]=buffer[i-1];

        buffer[0]=State1(n,ref,contr,Filtering());

        return 0;

}

// ***************************************************************************

float Sensor::State1(int i,float yref,int contr, float mis)
  {
   float a;
   float        kappa1[30]={.4, .4239, .434, .4404, .4459, .4512, .4563,
                            .461, .4653, .4691, .4723, .4751, .4775, .4795,
                            .4811,.4824, .4835, .4844, .4852, .4858, .4862,
                            .4866, .4869, .4872, .4874, .4875, .4877, .4878,
                            .4879, .4879};
       cout<<i<<"  e i "<<endl;

  if ( i==0)
        {

        state1=yref+kappa1[0]*(mis-yref);

        return state1;

        }
  else if (i<30)
        {

        state1=state1+.15*state2+.0000675*contr+kappa1[i]*(mis-
               (state1+.15*state2+.0000675*contr));
```

```
        return state1;
        }
  else
        {
        state1=state1+.15*state2+.0000675*contr+KAPP1*(mis-
               (state1+.15*state2+.0000675*contr));

        return state1;
        }

  }

// ************************************************************************

float Sensor::State2(int i, int contr, float mis)
  {
  float         kappa2[30]={.0, .01, .0303, .0558, .0834, .1111, .1375,
                            .1618, .1836, .2028, .2193, .2333, .2452,
                            .2550, .2632, .2699, .2754, .2799, .2835,
                            .2865, .2889, .2908, .2923, .2936, .2946,
                            .2954, .2961, .2966, .2970, .2973};

  if ( i==0)
        {

        state2=MEANA;

        return state2;

        }
  else if (i<30)
        {

        state2=state2+(.00000817*contr*contr+.0022*contr)*.0174+
               kappa2[i]*(mis-(state1+.15*state2+.0000675*contr));

        return state2;
        }
  else
        {

        state2=state2+(.00000817*contr*contr+.0022*contr)*.0174+
               KAPP2*(mis-(state1+.15*state2+.0000675*contr));
```

```
        return state2;
        }

  }

// ***************************************************************************

float Sensor::Filtering()
{

        float a;

        a=0.3*buffer[1]+0.2*buffer[2]+0.1*buffer[3]+.4*Getstring();
        if (abs(a-buffer[3])>TH_PAR)
                a=buffer[3];

        return a;

}

// ***************************************************************************

float Sensor::Value(int n)
        {

        // Return the n stored sonar data.

        return buffer[n];

        }

// ***************************************************************************

float Sensor::Error(int n)
        {

        // Return the n stored error data.

        return error[n];

        }

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

```
void Sensor::Store_Err(float a)

        {

        for(int i=1; i<BUFSIZE; i++)
                error[i]=error[i-1];

        error[0]=a;

        }

// ***************************************************************************

void Sensor::New_Data_Tracking()

        {

         while(sgetch()== -1);

        }

// ***************************************************************************

void Sensor::Init_State()

        {

        state1=Average();
        state2=MEANA;

        }

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

A.5 sensor.h

```
//      file sensor.h
//      Definition sensor class

#ifndef SENSOR.H
#define SENSOR.H

//      Constant definition

#define THCHANGE        .2
#define BUFSIZE         10      // Size of the data buffer
#define TH_PAR          .3      // Maximum difference between two measures
#define MINTHRESHOLD    3.0     // Minimum distance relevaible

#define bumper1         0x38    // Bumper1 input address
#define bumper2         0xf8    // Bumper2 input address

#define port_number     0x378   // Parallel output address
#define stop            0x03    // Stop both tracks word
#define stop_left       0x02    // Stop left track word
#define stop_right      0x01    // Stop right track word
#define start           0x00    // Run both tracks word

#define KAPP1           .4880   // Kalman filter's gain
#define KAPP2           .2976   // Kalman filter's gain

#define MEANA           -.0174  // Mean value of the departure angle


class Sensor
{
//      Definition of private data

  int           sonar;
  int           flag;

  float         width;
  float         buffer[BUFSIZE];
  float         error[BUFSIZE];
  float         state1;
  float         state2;
```

```
 long int       ttime;

//      Definition member function
  public:

  unsigned char Read_Bumper();

  void          Init_State();
  void          Set_Sonar(int i);
  void          Store();
  void          Clear_vec();
  void          Set_Flag();
  void          Store_Err(float a);
  void          Check_Measure(float ref);
  void          New_Data_Tracking();

  long          int Get_time();

  int           Sonar();
  int           Flag();
  int           Take_start_ref();
  int           Take_start_ref(float dist);
  int           New_Sonar_Measure(int n, float ref, int contr);

  float         Average();
  float         Filtering();
  float         Width();
  float         Getstring();
  float         Value(int n);
  float         Error(int n);
  float         State1(int i, float yref,int contr, float mis);
  float         State2(int i, int contr, float mis);


};
#endif
```

A.6 serial.c

```
//      file serial.c
//      serial port routines
#include<dos.h>
#include<stdio.h>
#include<bios.h>
#include<stdlib.h>

#define COM_PORT        1               // serial port to use
#define COMPARMS 0xc3   // 4800-n-8-1 " 0xe3 comm parameters for 9600-n-8-1 "

#include "serial.h"

void interrupt (*old_vector)();
char    queue[QUELEN];  // input buffer
int     break_detect;   // set true on break
int     char_overrun;   // set true on character overrun
int     echo;           // controls kbd echo to screen
int     translation;    // add lf to incomming cr
int     frame_error;    // count of framing errors
int     input_index;    // input index of serial buffer
int     output_index;   // output index for serial buffer
int     parity_error;   // count of parity errors
int     ring_detect;    // set true on ring detect

//                                  definitions for this program
float getstring(void)
{

      int i=0, flag=0;
      char ch,buffer[10]={'\0'};

      while((ch=sgetch())!='$');
      while(1)
      {
           if((ch>=48 && ch<58) || ch==46)
           {
               flag=1;
               buffer[i++]=ch;
           }
           if(ch==44 &&  flag==1)
               break;
           ch=sgetch();
```

```
      }

      return atof(buffer);
}
void interrupt serial_isr(void)                 // interrupt service routine
        {
        char ch;
        int temp_index;

        enable();                               //  enable interrupts
        ch = inportb(ACE_INT_IDENT_REG);        // examine IIR for cause
        switch(ch)
                {
                case 6:
                        ch = inportb(ACE_LINE_STAT_REG);  // get line status
                        if(ch & PE) parity_error++;
                        if(ch & OE) char_overrun++;
                        if(ch & FE) frame_error++;
                        if(ch & BI) break_detect++;
                        inportb(ACE_DATA_REG);  // read the data reg to empty it
                        break;

                case 4:                         // received data available
                        ch = inportb(ACE_DATA_REG);
                        temp_index = input_index + 1;
                        if (temp_index >= QUELEN) // test for index past buf end
                                temp_index = 0;
                        if(temp_index != output_index)  // test for queue overfl
                                {
                                queue[temp_index] = ch;
                                input_index = temp_index;  // update the index
                                }
                        break;

                case 0:                 //      if modem status interrupt
                        ch = inportb(ACE_MODEM_STAT_REG);
                        if(ch & DCTS)
                                {}      //      add routine for CTS changes
                        if(ch & DDSR)
                                {}      //      add routine for DSR changes
                        if(ch & TERI)
                                ring_detect = 1;
                        if(ch & DDCD)
                                {}      //      routine for DCd changes
```

```
                }
        outportb(PIC_CTL_REG, NON_SPEC_EOI);    // clear the PIC
        }

int sgetch(void)        // read a char from the serial input buffer
{
        int ch;
        int temp_index;

        if(output_index != input_index)
                {
                temp_index = output_index + 1;
                if(temp_index >= QUELEN)
                        temp_index = 0;
                ch = queue[temp_index];         //      get char from buffer
                output_index = temp_index;

                return(ch & 0xff);
                }
        else
                return((int)S_EOF);             //      if no chars available
}

void sputch(char ch)  //        output a char to serial transmitter
        {
        while((inportb(ACE_LINE_STAT_REG) & THRE) != THRE ) //  buffer empty?
          {     ; }             // if not, wait til it is
        outportb(ACE_DATA_REG, ch);
        }

void install(void)  // install the service routine and init the 8250
        {
        char ch;

        bioscom(0, COMPARMS, COM_PORT - 1);  //         set comm. params
        // enable ACA ;modem status, line status, and receive data interrupts
        outportb(ACE_INT_ENB_REG, 0x0d);
        inportb(ACE_DATA_REG);          //      empty the data register
        inportb(ACE_LINE_STAT_REG);     //      clear the line status register

        old_vector = getvect(COM_INT_NUM);
        setvect(COM_INT_NUM, *serial_isr);
```

```
        ch = inportb(PIC_INT_MASK_REG);
        ch &= IRQ_MASK;
        outportb(PIC_INT_MASK_REG, ch);

        outportb(ACE_MODEM_CTL_REG, 0x0b);  //  set RTS, DTR to enable modem
                              //and turn on OUT2 to enable 8250 IRQ to system

  //    printf("\nCOM%d initialized.\n", COM_PORT);
        }

void cleanup(void)  // restore normalcy on exit
        {
        char ch;

        outportb(ACE_INT_ENB_REG, 0);   //      disable all 8250 interrupts
        outportb(ACE_MODEM_CTL_REG, 0);         //      clear RTS, DTR, OUT2
        ch = inportb(PIC_INT_MASK_REG);
        ch |= !IRQ_MASK;
        outportb(PIC_INT_MASK_REG, ch);

        setvect(COM_INT_NUM, old_vector);


        }
```

A.7 serial.h

```
//      file serial.h
//      header for serial port routines

#ifndef SERIAL_H
#define SERIAL_H



               // receive a serial character
void interrupt serial_isr(void);        // serial interrupt service routine
void install(void);                     // installation/initialization
void sputch(char);                      // send a character
void cleanup(void);                     // exit cleanup routine
float getstring(void);

#define QUELEN  32                      // size of serial input buffer

//void interrupt (*old_vector)();       // place to save old int vector
#define S_EOF  -1                       // if no characters are available

//              8259A PIC register addresses and commands

#define PIC_CTL_REG             0x20    // PIC control register
#define PIC_INT_MASK_REG 0x21           // interrupt mask register
#define NON_SPEC_EOI            0x20    // non-specific end-of-interrupt

//              8250 ACE register addresses and bit definitions

#if             COM_PORT == 1           // definitions for COM1

#define ACE_DATA_REG            0x3f8   // data register
#define ACE_INT_ENB_REG         0x3f9   // interrupt enable register
#define ACE_INT_IDENT_REG       0x3fa   // interrupt identification reg.
#define ACE_LINE_CTL_REG        0x3fb   // line control register
#define ACE_MODEM_CTL_REG       0x3fc   // modem control register
#define ACE_LINE_STAT_REG       0x3fd   // line status register
#define ACE_MODEM_STAT_REG      0x3fe   // modem status register
#define COM_INT_NUM             12      // interrupt number for COM1
#define IRQ_MASK                0xef    // IRQ mask for IRQ4(11101111)

#elif   COM_PORT == 2                   // definitions for COM1
```

```
#define ACE_DATA_REG            0x2f8   // data register
#define ACE_INT_ENB_REG         0x2f9   // interrupt enable register
#define ACE_INT_IDENT_REG       0x2fa   // interrupt identification reg.
#define ACE_LINE_CTL_REG        0x2fb   // line control register
#define ACE_MODEM_CTL_REG       0x2fc   // modem control register
#define ACE_LINE_STAT_REG       0x2fd   // line status register
#define ACE_MODEM_STAT_REG      0x2fe   // modem status register
#define COM_INT_NUM             11      // interrupt number for COM1
#define IRQ_MASK                0xf7    // IRQ mask for IRQ4(11110111)

#endif

#define THRE            0x20            // transmit holding register empty
#define CTS             0x10            // clear to send
#define DSR             0x20            // data set ready
#define RI              0x40            // ring indicator
#define DCD             0x80            // data carrier detect
#define PE              4               // parity error
#define FE              8               // framing error
#define OE              2               // overrun error
#define BI              0x10            // break interrupt
#define DCTS            1               // delta clear to send
#define DDSR            2               // delta data set ready
#define TERI            4               // trailing edge ring detect
#define DDCD            8               // delta data carrier detect

#endif

// end of header
```

Appendix B

WEDA Pool-cleaners

This appendix describes all technical data, standard and extra equipment of the B400 and B600 poolcleaner.

	B400	*B600*
Dimensions (l x w x h)[cm]	90 x 48. x 30	86 x 60 x 30
Suction capacity [l/min]	48	60
Weight in air [kg]	35	52
Weight in water [kg]	12	7
Power rating [kW]	1.3	2.5
Speed [m/sec]	0.3	0.2-0.4
Main voltage 3-phase,50/60 Hz [V]	230/400/415	230/400/415
Secondary voltage 3-phase [V]	45	45

Table B.1: Technical data.

	B400	*B600*
Filter bags 125mic	Twin	Twin
Protection bags	-	Twin
Brushes	Fixed	Rotating
Electrical cable [m]	10	10
Cable hand box [m]	20	30

Table B.2: Standard equipment.

	B400	*B600*
Radio control	Yes	Yes
Timer	Yes	Yes
Free stream impeller	Standard	Yes
Filter bags inner 70 mic	Yes	Yes
Filter bags inner 200mic	-	Yes
Hose connection	Yes	Yes
Fixed brush	-	Yes
Single phase	Yes	-
Flexible bumper	-	Yes

Table B.3: Extra equipment.

B600's technical data have been inserted in order to make possible a future upgrade of our control software with this new and more powerful machine.

Appendix C

Navman 100 sonar

This appendix describes all technical data of the Navman 100 sonar system. These are the principal characteristics:

- Displays depth to 150 meters, 450 feet or 70 fathoms.
- Shallow and deep alarms are both audible and visual. Values are stored in memory at power off.
- Variable dampening rates allow user adjustment for optimum performance.
- Adjustable keel offsets can be calibrated to display depth below the keel or depth from the water line.
- Precision of one decimeter between 2-20 meters, 1 meter between 20-150 meters.
- Requested feeding from 12 to 24 volts.
- Data protocol NMEA 0184.
- Transducer 51 mm diameter, beam angle 9 degrees, working frequency 200 kHz.

Appendix D

Connection schemes

This appendix shows how to connect the parts of the system. The sonar is connected to the computer by the RS-232 serial port. In order to connect them, it is necessary to make the following connections:

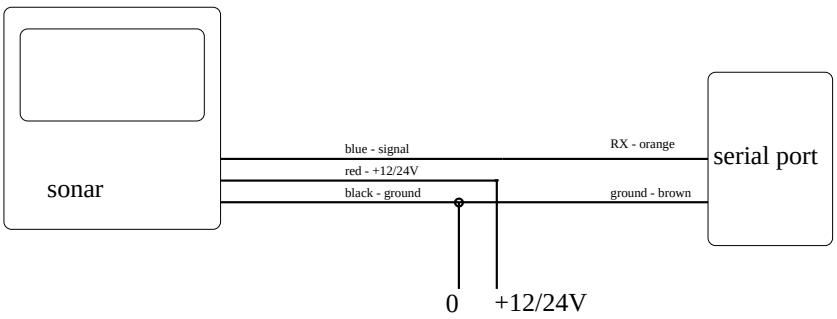

Figure D.1: Serial port connection.

The other connections between the bumpers, the tracks and the PC are made with the parallel port.

Bibliography

[1] Urick, Robert J.: Principles of underwater sound. *McGraw-Hill.* 3d edition/1983.

[2] Leroy, C.C.: Development of simple equations for accurate and more realistic calculation of the speed of sound in sea water. *J.Acoust.Soc.Am. 46:216 (1969).*

[3] Leonard, J.J. and Durrant-Whyte, Hugh F.: Directed sonar sensing for mobile robot navigation. *KAP-Kluwer Academic Publishers, 1992.*

[4] Sorenson, Harold W.: Kalman filtering: theory and application.

[5] Martins de Carvalho, J.L.: Dynamical systems and automatic control.

[6] Marro, G.: Controlli automatici. *Zanichelli, 1992.*