Institute of Parallel and Distributed Systems
University of Stuttgart
UniversitätsstraSSe 38
D–70569 Stuttgart

Masterthesis Nr. 3310

3D Video Tracking and Localization of Underwater Swarm Robots

Martin Antoni

Course of Study: INFOTECH

Examiner: Prof. Dr. rer. nat. habil. Paul Levi

Supervisor: Dipl.-Ing. Tobias Dipper

Commenced: 01.02.2012

Completed: 02.08.2012

CR-Classification: D.1.0, B.1.4, I.4.1

Abstract

Autonomous underwater vehicles (AUV) are robots, which usually estimate their position by localization the help of internal or external sensors. In this thesis, small swarm robots from the CoCoRo are used as experimental platform. It is often useful, to know the exact position inside the testing area to evaluate swarm algorithms. Controlling the position of the robot should be possible as well.

A software is developed, which is able to track a robot inside an aquarium. Two cameras are install at each side of this aquarium for determining the 3D position which includes the diving depth. Perspective distortions, which come from viewing angle, are compensated with the help of image transformation. With the corrected image, the template matching algorithm with normalized cross-correlation is used to track the robot in the camera image.

A wireless connection is established between the computer and the robot to read out sensor data and to control the motors. Waypoints can be set by the user which the robot follows. The computer uses two independent controllers for rotational and for distance control.

Zusammenfassung

Unbemannte Unterwasserfahrzeuge (AUV) sind Roboter, welche üblicherweise ihre Position durch interne und externe Sensoren bestimmen. In dieser Masterarbeit werden Schwarmroboter von dem CoCoRo Projekt als Testplattform benutzt. Es ist sinnvoll, die genaue Position des Roboters zu wissen. Das macht es einfacher, bestimmte Schwarmalgorithmen zu testen. Weiterhin ist es nützlich, den Roboter steuern zu können.

In dieser Arbeit ist eine Software entwickelt worden, welche die Position des Roboters in einem Aquarium verfolgt. Zwei Kameras sind auf jeder Seite des Aquariums angebracht. Damit ist es möglich, die 3D Position zu bestimmen, welche die Tauchtiefe enthält. Perspektivische Verzerrungen, welche durch den Blickwinkel entstehen, werden mit einer Koordinatentransformation kompensiert. Mit Hilfe des 'Template matching'-Algorithmus und der normalisierten Kreuzkorrelation wird die Position des Roboters im Bild verfolgt.

Eine kabellose Verbindung zwischen dem Computer und dem Roboter überträgt die Sensordaten und wird für die Motorsteuerung genutzt. Die Motorleistung wird durch eine Drehungs- und Abstandsregelung vorgegeben. Durch Wegpunkte kann eine Route gesetzt werden, welche der Roboter abfährt.

Contents

List of Figures

List of Tables

1 Introduction

Autonomous underwater vehicles (AUV) are used in scientific application, like analysing an area of a reef or the sea ground. They can operate alone or with multiple robots. When many robots act together, a swarm is created. Swarm robots have the advantage, that if one fails, the group can continue operating. Multiple defects lead only to a smaller group. When using one big robot, which have the same ability like the whole swarm, defects on the robot are much more harmful.

AUV's navigate with the help of localisation and mapping. This includes estimating their position using sensors. Some robots can operate with a relative position, others needs the absolute position.

The development phase is a very critical phase where quick results are important to get good results in a feasible time. All aids are welcome to the developer. In this thesis, a tool is discussed, which eases the testing and developing phase of autonomous underwater vehicles (AUV).

The CoCoRo [coc] project deals with AUVs which are able to move in a swarm. These robots can estimate their positions with internal and external sensors. Calculations are made with the sensor data and control data from actors. The position can either be an absolute one or a relative position to other robots. To validate the precision of the estimation, the real position has to be known. With this information, parameters influencing the estimation can be optimized. Localisation of AUVs has two main approaches, visual and acoustic[PZ04]. In the visual approach, cameras are installed in the robot to get a picture of the environment. The robot can use stereo cameras[PZ04] to locate itself. Also, pattern recognition can be used for localization[MCN]. These approaches have the problem, that the camera inside the robot is relatively big and processing the video stream needs a lot of computation power. In the CoCoRo project, robots with a size of about 12cm in diameter are used. They have only a small microcontroller for the basic controlling applications. An onboard camera is therefore no option. The acoustic approach uses sound waves. In a small aquarium, a lot of reflections occur. This is very problematic, because the waves add and subtract themselves. The localisation is almost not possible.

1.1 Task

For this thesis, the localisation area is an aquarium. A system is needed to get the position of the robots inside this aquarium. This is done with external cameras. This method is chosen, because it does not disturb the movement of the robot. With two cameras at different positions, it is possible to get the 3D position of the robot. One problem with this approach is refractions. Under water, light travels slower than in air. These refractions have to be considered in the position calculation process. With the knowledge of the corner points of the aquarium, a transformation matrix can be created to compensate the viewing angles distortions.

The cameras are connected to a computer. Processing the video stream from the cameras is needed to locate an object inside a picture. Many algorithms exist for localisation. With the feature based approach[JH], borders and edges are used to recognise an object. This is not sufficient for this thesis, because the robot (see figure 2.2) does not have many features. Other approaches use all pixel values to find an object. In this thesis, the template matching algorithm [KB01] was chosen. With the help of internal sensor data from the robot, it is possible to get the heading of the robot.

To establish a connection between the robot and the computer to exchange data, a wireless system is needed. It is not possible to connect a cable to the robot, because this would disturb its movement. All robots are equipped with a RF module. Transmitting data is done in the robots firmware. On the computer side, an interface is needed to connect the module to the computer. With the aid of waypoints, the robots should be able to drive a defined route without interaction of the user.

1.2 Work description

The task is to get a working solution, where two cameras point onto the aquarium and track the robots inside the aquarium. There has to be control of the position of one robot over the computer software. Therefore, a connection to the robot has to be established. All algorithms that are used are described in this thesis. In the first step, a camera is chosen for the video tracking system. This camera is selected with regards to the price and resolution. In the next step, a program to read out the video stream is developed and the frame rate of the tracking has to be defined by the user. After setting up the cameras, a proper tracking algorithm is selected and implemented. With the help of the tracking system, the control of the robot is developed in the next step. First, the control of the rotation, then the distance control is implemented. After motion control of the robot is established, the selection of waypoints is implemented. Logging feature have to be implemented, to use the tracking data for later evaluation.

2 Hardware

There are two main parts of hardware used in this thesis, the AUV 'Lily' and a computer. For the communication, a wireless system is used. For the video tracking, two cameras are installed. In the following sections, each part is described in detail.

2.1 Robot

The underwater robot that is used in this thesis is a modified toy submarine. Figure 2.1 shows the original model, which can be bought in the internet or other stores. This toy can move backwards, forwards and dive. It also can turn on a spot, so it is possible to move in each direction. For the CoCoRo project, this toy was modified.

Figure 2.1: Submarine Toy

The top was replaced by a custom made head. Motors and the on-off switch on the bottom part were not changed. In the inside, the custom main control board was inserted and the batteries were changed to lithium polymere technology. Several sensors are around the outside of the robot and on the mainboard. In figure 2.2, the final version of the robot is shown, which is used as experimenting platform during this thesis. The pins on the top and on the side of the robot are electric field sensors, which are used in a different project and are not part of this thesis, so they are ignored. The dark dots are photodiodes for the blue light sensor (see section 2.3). This sensor also includes several blue light emitting diode (LED).

Figure 2.2: CoCoRo Robot

2.2 Communication

The robot communicates mainly through two different channels. One channel is the RF module and the other channel is the blue light sensor. In this thesis, the RF module is used most of the time, because it is more flexible than the blue light transmission and omnidirectional. Blue light has the problem, that it doesn't work with high distances between sender and receiver and is directional.On the other hand, it is capable of higher data rates. For the RF module, a connection to the computer is needed, because the module has only a serial peripheral interface (SPI) for setup and data transmission. An interface is developed and described in section 2.2.2.

2.2.1 Blue light transmission

The blue light sensor uses optical data transmission under water with a led as sender and a photodiode as receiver. For the light color of the LED, there are many possibilities. Blue was chosen, because it is less absorbed than other visible light under water[WSPZ97]. A photodiode was chosen, which is sensitive to this color. A multiprotocol transceiver chip (CS8130) is used which can drive two sending LEDs and has one photodiode input. The transceiver has an UART interface to receive and send data. It is also used to change the configuration of the CS8130. CMOS levels (3.3V) are used for transmission, so no level converter is needed to connect this module directly to the main microcontroller. The communication via light uses modulation to enhance the noise margin of the transmission and to reject ambient light and other parasitic light. Amplitude shift keying (ASK) modulation is used with a frequency of $f = 500kHz$. Section 2.3.4 describes, how the blue light is also used for distance measurement.

2.2.2 USB RFM12 Bridge

The second and mainly used channel in this thesis uses wireless transmission with a 433MHz transmission frequency. This is a lower frequency when comparing ZigBee[PB06] or BlueTooth[blu01], which work with a frequency of 2.4GHz. The problem is, that water absorbes the higher frequency more than the lower frequency. So, a lot more transmission power would be needed for the same transmission range. A RFM12 transceiver module is used in the robot because the bandwidth is sufficient for this application and also the price is very low. An additional advantage is that these modules are used by many people in the world and the documentation and tutorials are very good. To communicate with the RF module, the SPI interface is used, which cannot be directly connected to the computer. The interface in the developed bridge is a microcontroller from ST-Microelectronics, a 32-Bit Cortex-M3 STM32F103CB. This chip has an integrated USB interface and a SPI interface. That makes the Cortex fitting perfectly this application.

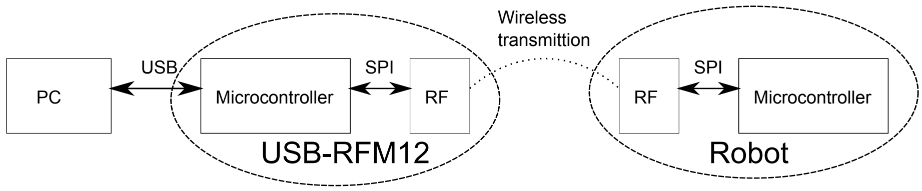

Figure 2.3: USB RFM12 bridge dataflow

Figure 2.3 shows the dataflow of data from the pc to the robot and backwards.

Wireless systems have the problem, that the error rate is higher than wired systems[RK]. To compensate this, the data is wrapped inside a frame, which is controlled by a checksum. Table 2.1 shows a data packet with n bytes.

$preamble$	$preamble$	$length$	$data[0]$	$data[1]$	$\ldots$	$data[n]$	$CRC16$

Table 2.1: Data packet

The preamble consists of bit changes for optimal clock recovery of the receiver. $0x55$ and $0xAA$ are good values. Following the preamble, the data length is appended. This has a advantage against fixed length transmission. If small packets are sent, then no time is wasted while at the same time, longer packets can be sent. At the end of the data packet, a cyclic redundancy check (CRC) is added. Over the whole packet, the CRC16[GG87] sum is counted. The receiver can then detect bit errors and discard the data packet.

Interfacing the computer software with the USB protocol is done with the free LibUsb project. This is an open source implementation of USB functions to read and write to the interface. In fact that the C# programming language is used, a modification of the LibUsb project is used. A library for DotNet exists, it is called LibUSBDotNet[lib]. To improve the transmission distance, a better antenna than the chip antenna is mounted. In figure 2.4, the final version of the bridge hardware is shown. The pin header is used for programming and will be removed in the finished version.

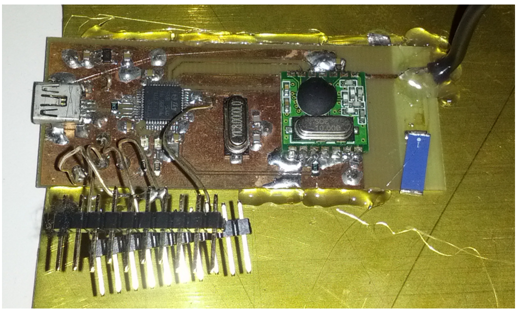

Figure 2.4: USB RFM12 bridge

2.3 Sensors

In table 2.2, all sensors are listed which are installed in or on the robot. The distance sensor is in brackets, because the main part for this sensor is communication. All used sensors are discussed in more detail in the following sections.

Type	Sensor
Position	Accelerometer
	Digital compass
	Electric field sensor (not used)
	Pressure sensor
Status	LEDs (no sensor, but indicator)
Distance	(Blue light sensor)

Table 2.2: Sensors of the robot

2.3.1 Accelerometer

The accelerometer has three axis, x-axis, y-axis and Z-axis. It is a LIS302DLH from STMicroelectronics. It is highly integrated. The acceleration is digitalized with a resolution of 12Bit with the build in analog-to-digital converter (ADC). All three axes can be read out over the I^2C interface.

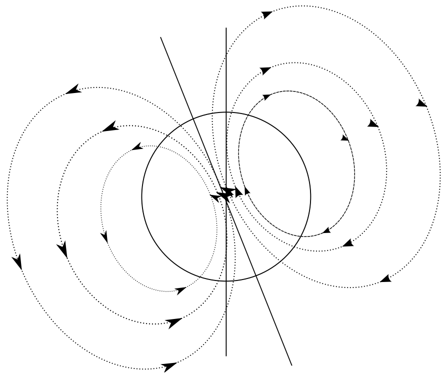

Figure 2.5: Magnetic field of the earth

2.3.2 Digital compass

To get the heading or rotation of the robot, a digital compass in microelectric mechanical system (MEMS) technology is used. This compass has the advantage of being very small and consuming very little electrical power. There are different compass sensors available from manufactures like ST-Microelectronics and Honeywell. The robot has a Honeywell HMC5883L on its mainboard. This sensor is a digital three axis compass which uses mageto-resistive sensor elements to measure the small magnetic field of the earth. Because the sensor has a very high integration factor, it can be installed very simply. A 12 Bit AD converter is on the chip and an I^2C interface is used to read out the data. The vector of the magnetic field can be seen in figure 2.5. The strength depends on the location on the earth. In central Europe, it has about $48 microtesla(\mu T)$. The old unit for the magnetic field is gauss (G). 1G equals $100\mu T$. With the compass, the field can be measured and all three field vector components can be read independently.

To get the robot's heading, some mathematical calculations have to be done. There are some cases, which require additional sensor data from a tilt sensor. In the simple case, the compass is level to the earth's surface. Only the x and y components are needed to get the heading. A coordinate transformation between cartesian and polar coordinates can be done with the following equation.

$$\varphi = \arctan \frac{y}{x} \tag{2.1}$$

where φ is the heading. One problem is that the arctan function cannot differ between the quadrants of the coordinate system. This problem is solved by using the arctan 2 function.

$$\varphi = \arctan 2(y, x) \tag{2.2}$$

This is just an extension to the normal arctan function, where the correct quadrant of the coordinate system is considered. When the compass is not level to the surface, a tilt error occur. This error can be compensated with a second sensor, which measures the tilt. It is possible to get the value of the tilt with the help of an accelerometer. This has the restriction, that no additional acceleration than the earth's gravity exists. Otherwise, errors would occur.

To get the tilting, the earth's gravity is measured. The gravity vector is pointing downwards to the earth surface. It has a strength of $9.81\frac{m}{s^2}$[for96]. The roll angle ϕ and the pitch angle θ can be calculated by using all three axes from the accelerometer. Figure 2.6 shows, to which axis the angles belong to. The roll angle ϕ defines, how much the robot is tilted to the left or right. When tilting to the top or the bottom, the pitch angle θ is different from zero. The yaw angle ψ is the heading. In

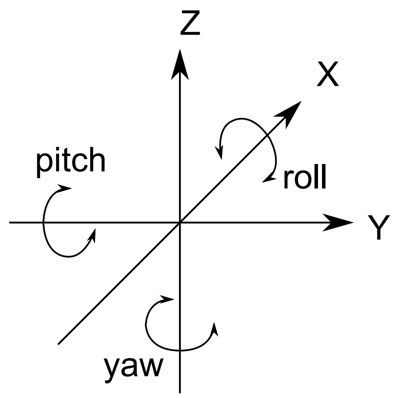

Figure 2.6: Yaw, roll and pitch angle

this thesis, the tilt compensation is applied. All equations given in this section are from an application note[com].

In the tilt compensation algorithm, the pitch and roll angle are calculated first. With these two angles, it is possible to rotate the magnetic field vector back to the earth's surface. This is done by multiplication with rotation matrices.

The gravity vector is defined as

$$\overrightarrow{g} = \begin{pmatrix} g_X \\ g_Y \\ g_Z \end{pmatrix} \tag{2.3}$$

and the magnetic field vector is defined as

$$\overrightarrow{b} = \begin{pmatrix} b_X \\ b_Y \\ b_Z \end{pmatrix} \tag{2.4}$$

The final equation for the roll angle is

$$\phi = \arctan\left(\frac{g_Y}{g_Z}\right) \tag{2.5}$$

and for the pitch angle

$$\theta = \arctan\left(\frac{-g_X}{g_Y \cdot sin\phi + g_Z \cdot cos\phi}\right) \tag{2.6}$$

The pitch angle θ in equation 2.6 depends on the roll angle ϕ. This is due to the fact, that the rotation matrices are applied consecutively. It is also possible to calculate first the pitch and then the roll angle, but then, different solutions are given. To get the heading ψ, the following equation is used.

$$\psi = \arctan\left(\frac{v_Z \cdot sin\phi - v_Y \cdot cos\phi}{v_X \cdot cos\theta + v_Y \cdot sin\theta \cdot sin\phi + v_Z \cdot sin\theta \cdot cos\phi}\right) \tag{2.7}$$

2.3.3 Pressure sensor

The pressure sensor is used to measure the diving depth of the robot. It is an absolute pressure sensor which has a very high resolution of 1 Pascal (Pa). Each time the robot is switched on, the sensor has to be calibrated to ambient pressure. This is done automatically in the firmware of the robot.

2.3.4 Distance measurement

To get the distance of an object or a wall, a light pattern is transmitted over the blue light diode, gets reflected and the backscattered light is received with the photodiode. A light modulation is used, otherwise every ambient light would alter the measurement. The transceiver is set to a 500kHz ASK modulation. The microcontroller sends a data pattern through the CS8130 transceiver. The data pattern $0xAA$ is randomly selected for this purpose. If the reflection is strong enough, then the light received from the photodiode and is send back to the microcontroller over the CS8130. The transceiver has a variable gain amplifier implemented. To change the sensitivity of the photodiode, internal registers of the transceiver have to be reprogrammed. This feature is used for the actual distance measurement. The reception threshold is decreased until a valid data pattern is received. The distance can be calculated as follows:

$$d = \frac{K_{ambient}}{treshold^2} \tag{2.8}$$

$K_{ambient}$ is a constant, which depends on the environment. It includes the reflection coefficient of the object and also the light absorbtion of the water. Because of this fact, a simple lookup table is used.

2.4 Actors

Two electrical motors are mounted on the bottom back of the robot. They have small propeller to operate under water. If both motors are turned on, the robot is going forward or backwards, depending on the polarity. If only one motor is turned on or both motors are turned on in opposite direction, the robot turns. This is called a differential system. To enable diving, the robot has a third motor installed, which drives a pump. The volume of air can be changed inside the robot and the buoyancy adjusted. The volume of the robot changes and it is possible to control the float.

2.5 Aquarium

An aquarium is used to test the camera system and the control of the robots. It has a water depth of about 80cm. It has a width of 138cm and a length of 200cm. All experiments are performed in this environment.

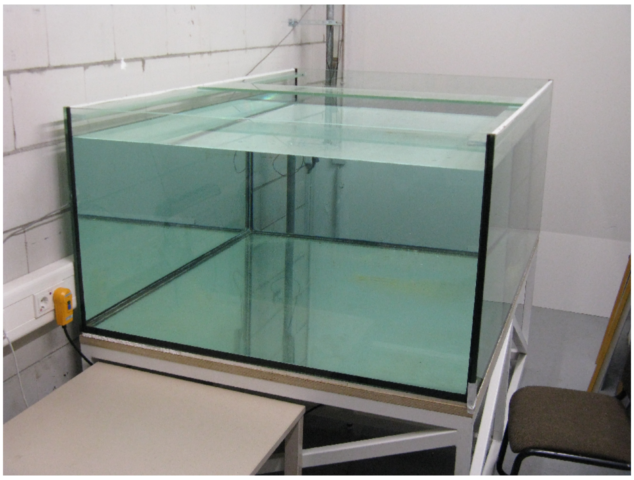

Figure 2.7: Aquarium

2.6 Cameras

The tracking system needs two cameras to get the 3D position the robots. For this project, conventional webcams are used, because they are cheap and have a high enough resolution. The Microsoft LifeCam HD-3000 (figure 2.8) has a HD ready resolution (1280x800), but this resolution can only be used with the proprietary software from microsoft. In the developed software, the resolution is limited to 640x480. This is still high enough to get a position resolution of about 1cm.

Figure 2.8: Microsoft LifeCam HD-3000

The camera has a flexible leg, which can be mounted on the ceiling of the room or at least above the aquarium. Both cameras have to be on the opposing sides of the aquarium to get a precise picture of the robot.

In figure 2.9 and figure 2.10, both sides of the aquarium can be seen. All corners of the aquarium are in the picture. One obvious problem is the viewing angle, which distorts the picture. This can be compensated (see 3.1).

Webcams normally have an auto exposure feature, which means, the camera adapts to ambient light changes. This can lead to problems, if the cameras are used outside a room with unsteady, environmental light. In section 3.3, this problem is discussed.

Figure 2.9: View of camera 1

Figure 2.10: View of camera 2

3 Video Tracking

Video tracking is a method for tracking an object with a video camera. Tracking means that an algorithm extracts the position of an object. When this object moves, the algorithm has to recognize this and calculate the new position. For each frame, the position of the tracked object has to be calculated. This can be done with different time intervals. More pictures per second means a higher motion resolution of the object, but it also has a higher computational effort. Less pictures per second generate less data to be analysed, but the resolution is reduced. There is an optimum between the frame rate (pictures per time interval) and resolution. If the object moves very slow, then low rates can be used. In this thesis, an underwater robot is the object. It has a maximum velocity of about 10cm/s, so a frame rate of 10Hz is sufficient.

To find an object in a picture, many different algorithms exist. In this thesis, the template matching algorithm is chosen. Figure 3.1 shows the processing steps for the video tracking system. The next sections discuss each step in detail. In the first step, the video stream is read in a $10Hz$ frame rate. The following step corrects the distortion, which comes from the camera perspective. In section 3.1, a solution for this is presented. Having a back transformed picture, the position is computed with the template matching algorithm using cross-correlation. In the next step, the output of the normalized cross correlation is searched for its maximum. Once the maximum is found, the data from both cameras is combined and the real position and diving depth of the robot are calculated. The last step is displaying the results on the screen. The real position cannot be found by just one camera, if the robot dives. Both cameras give only a virtual 2D position of the robot, because refraction changes the direction of the light

3.1 Perspective distortion

When a camera looks onto an object, distortions are present. The distortion depends on the viewing angle of the camera and the distance from the camera to the object. Also, the focal length of the camera plays an important role. In this distortion, lines stay lines, but their angle changes. Figure 3.2 shows an example for this kind of distortion. All parallel lines in reality meet in one vanishing point. Also, objects near to the camera look bigger than object in the distance. In this project, both cameras look from a different angle and distance onto the aquarium. So there are

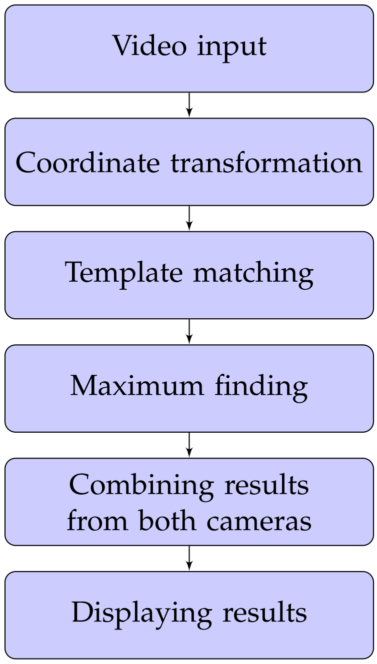

Figure 3.1: Image processing flow

two different distortions. To correct the distortions, a mathematical transformation has to be applied. The general transformation matrix has the form

$$M = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \tag{3.1}$$

The last coefficient i, it is set to 1, it represents a scaling coefficient. The multiplication of the original coordinates with the transformation matrix produces the undistorted coordinates. To multiply, the coordinates have to be extended with one dimension more, which represents a scaling factor.

Now, the equation looks like the following.

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = M \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{3.2}$$

Solving equation 3.2 leads to equation 3.3 and 3.4.

$$x' = \frac{a \cdot x + b \cdot y + c}{g \cdot x + h \cdot y + 1} \tag{3.3}$$

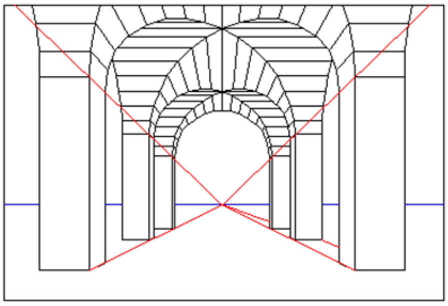

Figure 3.2: Vanishing point

$$y' = \frac{d \cdot x + e \cdot y + f}{g \cdot x + h \cdot y + 1} \tag{3.4}$$

All parameters from the matrix M need to be found. There are eight parameters, so at least eight equations have to be solved to get a unique solution. One corner point has two coordinates, X and Y. Four points are used to get all parameters for the transformation matrix. Changing equation 3.3 and 3.4 removes the fraction and leads to the following equations.

$$x' = a \cdot x + b \cdot y + c - g \cdot x \cdot x' - h \cdot y \cdot x' \tag{3.5}$$

$$y' = d \cdot x + e \cdot y + f - g \cdot x \cdot y' - h \cdot y \cdot y' \tag{3.6}$$

Taking all four corner points into consideration, the following equation system is created.

$$\begin{pmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1 \cdot x'_1 & -y_1 \cdot x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1 \cdot y'_1 & -y_1 \cdot y'_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2 \cdot x'_2 & -y_2 \cdot x'_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2 \cdot y'_2 & -y_2 \cdot y'_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3 \cdot x'_3 & -y_3 \cdot x'_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3 \cdot y'_3 & -y_3 \cdot y'_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4 \cdot x'_4 & -y_4 \cdot x'_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4 \cdot y'_4 & -y_4 \cdot y'_4 \end{pmatrix} \cdot \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{pmatrix} = \begin{pmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ x'_3 \\ y'_3 \\ x'_4 \\ y'_4 \end{pmatrix} \tag{3.7}$$

Solving this equation system can be done with the gaussian elimination algorithm[for96]. The software in section 5.2.1 solves the system numerically.

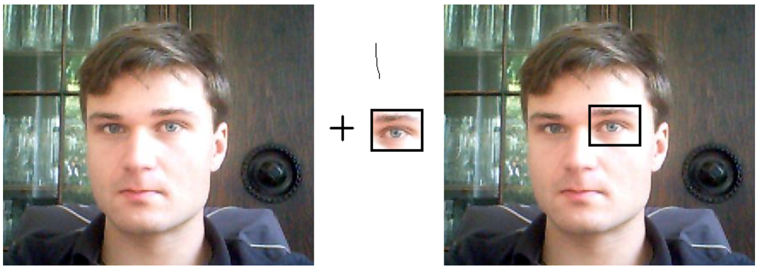

Figure 3.3: Template matching

3.2 Gaussian elimination process

The Gaussian elimination is an approach to solve systems of linear equations. Elimination means, that out of all equations, two are added in such a way, that one variable is eliminated. Applying this to all equations a few times, it is possible to get a solution to one variable. After having one solution, this is substituted back to an equation with two variables and so on. A system of equation has the form

$$A \cdot \overrightarrow{X} = \overrightarrow{B} \tag{3.8}$$

where A is a coefficient matrix and B is a vector. X is the transformation vector which contains the solution to this system. After applying the Gaussian elimination process, the matrix A has all zeros under the first main diagonal. The next matrix is an example, where the elimination was applied.

$$A = \begin{pmatrix} 1 & 2 & -4 \\ 0 & 1 & 3 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.9}$$

In the last line of equation 3.9, all variables except one are zero.

3.3 Template matching

Template matching means, that a template is compared with a picture and the algorithm finds the point, where the template and the picture match the best. Figure 3.3 shows an example. The left picture shows the input, in the center is the template and on the right side, the template is found in the input picture and marked with a frame. The challenge during template matching is to make it robust against picture noise and other changes like brightness and scaling / rotation transformations. There are two main approaches in template matching. One is called feature matching[JH]. Special features like corners, edges or other shapes are marked in the template. These features are then searched in the picture. It is only possible, if the template is not just plain but contains enough usable information.

The other approach is called template based. There, all pixel information from the template is used for comparison. Comparing can be done with difference values (SAD[sad], cross-correlation[KB01]). In this thesis, the cross-correlation is used as template matching approach.

3.4 Cross correlation

The cross-correlation is the summed product of two functions[tim06]. The formula of the continuous cross-correlation is defined as follows:

$$R_{xy}(\tau) = \int_{-\infty}^{\infty} x(t) \cdot y(t-\tau)dt \tag{3.10}$$

τ describes the replacement in time, $x(t)$ and $y(t)$ are the two functions, which are compared to find a match. This is the global case, where both functions reach from $-\infty$ to ∞. The result is a function, which maxima describe points, where a good matching in time exists. For image processing, the cross correlation is not one dimensional. There is not a replacement in time but two different replacements in x and y position. Furthermore, the images are in digital form, so a discrete solution is needed. The next formula shows the 2D discrete cross correlation.

$$R_{fg}(x,y) = \sum_{i=0}^{W-1} \sum_{j=0}^{H-1} f(i,j) \cdot g(i+x, j+y) \tag{3.11}$$

W is the width and H is the height of the template. The function f represents the template and the function g is the picture. Both variables i and j represent the counting variables. For image processing, the cross-correlation is not working very good in some cases. If the contrast or the brightness changes, the matching coefficient R_{fg} gets smaller and the peak from its curve is wider. The precision gets lower. For compensating this, the normalized cross correlation (NCC) is used. The following equation shows the NCC.

$$R_{fg}(x,y) = \frac{1}{W \cdot H} \cdot \sum_{i=0}^{W-1} \sum_{j=0}^{H-1} \frac{(f(i,j) - \overline{f}) \cdot (g(i+x, j+y) - \overline{g})}{\sigma_f \cdot \sigma_g} \tag{3.12}$$

When comparing equation 3.11 to equation 3.12, then the mean of the template $\overline{f}$ and the mean of the picture $\overline{g}$ is subtracted to the functions, which holds the pixel data. If the brightness of the input picture gets higher, the mean of the picture is also higher. That means, subtracting the mean compensated brightness changes. In the denominator of equation 3.12, the standard variation σ is a term. Contrast changes in a picture changed also the standard deviation. But due to the division of the standard deviation in the NCC, contrast changes are also compensated. The mean of the function $f(x, y)$ is defined as

$$\overline{f} = \frac{1}{W \cdot H} \cdot \sum_{i=0}^{W-1} \sum_{j=0}^{H-1} f(i, j) \tag{3.13}$$

The standard deviation σ of the function $f(x, y)$ is defined as

$$\sigma = \sqrt{\frac{1}{W \cdot H} \cdot \sum_{i=0}^{W-1} \sum_{j=0}^{H-1} (f(i, j) - \overline{f})^2} \tag{3.14}$$

From the equation 3.12, it can be seen, that there are many computations, especially when the image size or the template size is high. For tracking applications, it is usually not necessary to cross-correlate the template with the whole input picture. When the user selects the first time the template, the initial position of the object is known. From this point, only a narrow surrounding has to be analysed, because the object does not move very far. This area can be very small, but it has to be big enough, that the object cannot move outside the analysed area during one frame interval. In the software in section 5.1, the area is only 10 by 10 pixels with a template size of 16 by 16, because the robot moves with a maximum of 10 cm/s. The frame rate is 10Hz, so between two frames, only 1cm (1 pixel) is moved.

3.5 Calculation of the 3D position

The two cameras look on the surface of the water to see the robot. But the problem is that the robot can dive. When diving, the cameras see the water surface, not directly the robot itself. When light goes from an optical thin medium to an optical thick medium or the other way around, refraction occurs. Water has a refraction coefficient of 1.33 [for96], while air has approximately 1.0 [for96]. This index is slightly depending on the temperature and more on the wavelength, but for the precision of this work, this can be neglected. Because of the refraction, the real position of the robot is not where it is seen on the surface.

The camera looks at any point of the aquarium with a different angle. With Snell's Law (figure 3.4), the refraction angle can be calculated. The incoming angle is β_1 is refracted to the outgoing angle β_2. When the material n_2 is optical thicker (higher refraction index) than material n_1, the outgoing angle is refracted to the normal. The relation between the two angles β_1 and β_2 is described in Snell's Law[for96].

$$\frac{sin\beta_1}{sin\beta_2} = \frac{n_2}{n_1} \tag{3.15}$$

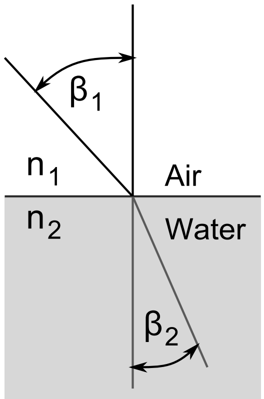

Figure 3.4: Snell's Law

Figure 3.5 shows the aquarium, where all rays are drawn and all vectors are named. The camera on the right is only indicated, but for the calculation process, only one camera is regarded.

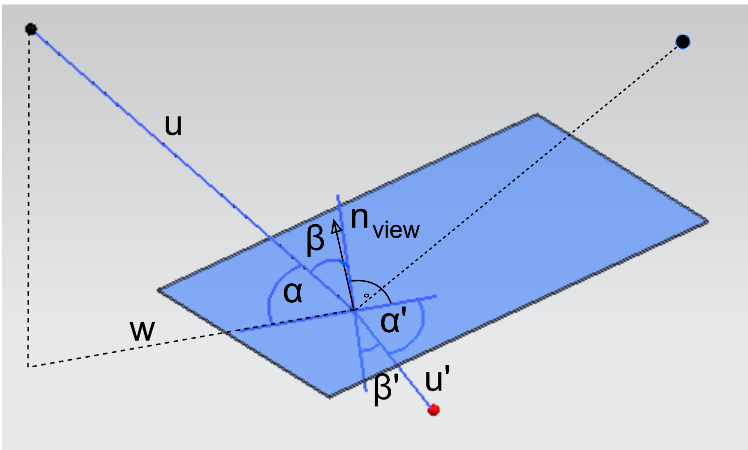

Figure 3.5: Optical path of the cameras in the aquarium

Table 3.1 describes all related angles and vectors to figure 3.5. In the center of the picture where all rays meet is the virtual position of the robot on the water surface. The line u is the ray from the camera to the surface. This ray is refracted to the line u'. All angles which are needed to compute the position are shown. α is the viewing angle from the left camera. For Snell's Law, the angle to the normal is needed, which is marked as β. The refracted angle then is β' and the viewing angle under water is α'. The viewing plane is needed to calculate the refracted ray. This plane is orthogonal to the XY plane (water surface) and contains the lines u, u' and

u	Ray from camera to surface
u'	Ray from surface to robot
w	Projected Ray u to the XY plane
$\overrightarrow{n}$	Normal vector of the viewing plane
α	Viewing angle
α'	Refracted angle to normal
β	Viewing angle to normal
β'	Refracted viewing angle to normal

Table 3.1: Perspective distortion vectors and angles

w.

For the line u, the following equation holds.

$$u = (\overrightarrow{P_{robot\_surface}} - \overrightarrow{P_{camera}}) \cdot j + P_{robot\_surface} \tag{3.16}$$

The line w has the equation 3.17.

$$u' = (\overrightarrow{P_{robot\_surface}} - \overrightarrow{P_{camera\_bottom}}) \cdot k + P_{robot\_surface} \tag{3.17}$$

To get the normal vector $\overrightarrow{n}$, the vector product between the directional vector from equation 3.16 and equation 3.17 has to be done.

$$\overrightarrow{n} = (\overrightarrow{P_{robot\_surface}} - \overrightarrow{P_{camera}}) \times (\overrightarrow{P_{robot\_surface}} - \overrightarrow{P_{camera\_bottom}}) \tag{3.18}$$

With all three equations, the viewing plane equation can be written down.

$$p : \begin{pmatrix} x \\ y \\ z \end{pmatrix} \cdot \overrightarrow{n} = 0 \tag{3.19}$$

To calculate the viewing angle α, the following trigonometric functions are used.

$$sin\alpha = \frac{|\overrightarrow{P_{robot\_surface}} - \overrightarrow{P_{camera}} \cdot \overrightarrow{n}_{XY}|}{|\overrightarrow{P_{robot\_surface}} - \overrightarrow{P_{camera}}| \cdot |\overrightarrow{n}_{XY}|} \tag{3.20}$$

with the XY plane's normal vector:

$$\overrightarrow{n}_{XY} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \tag{3.21}$$

The viewing angle with respect to the normal vector is

$$\beta = 90 - \alpha \tag{3.22}$$

With equation 3.16 and equation 3.19 it is possible to calculate the refracted ray u'. Two conditions hold for this ray: It has to lie in the viewing plane and its angle to the water surface has to be the refracted viewing angle α'.

Condition 1:

$$p : \overrightarrow{P_{robot\_real}} - \overrightarrow{P_{robot\_surface}} \cdot \overrightarrow{n} = 0 \tag{3.23}$$

Condition 2:

$$\beta' = arcsin(\frac{sin(\alpha') \cdot 1}{1.33}) \wedge \beta = 90 - \beta' \tag{3.24}$$

Condition 3.23 and 3.24 give two equations with two unknowns that can be solved. The result is the refracted line u' which has the direction vector

$$\overrightarrow{u'} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{3.25}$$

The last coordinate Z is set to -1. That means, the vector points downward. The value -1 has the advantage, that in a later step the diving depth is calculated faster. The first condition from equation 3.23 leads to:

$$\overrightarrow{u'} \cdot \overrightarrow{n} = 0 \tag{3.26}$$

Condition 2 leads to equation

$$sin\alpha' = \frac{\left| \begin{pmatrix} x \\ y \\ -1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \right|}{\left| \begin{pmatrix} x \\ y \\ -1 \end{pmatrix} \right| \cdot \left| \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \right|} \tag{3.27}$$

Solving equation 3.26 and equation 3.27, the component y has the following solution

$$y = \pm \sqrt{\frac{\frac{1}{sin^2\alpha'}}{1 + (\frac{n_y}{n_x})^2}} \tag{3.28}$$

Solving for x is similar. There are two solutions, which must be chosen correctly for each camera. The left camera looks in positive y direction, so the y-component is positive. The right camera looks in a negative y direction, so the y-component is negative.

All calculations have to be done for both cameras. The result is two line equations which are used for the final step. In this step, a point is found, which has the minimal distance to both lines u_1 and u_2. There are two cases. In the first case, which does not occur very often, both lines intersect in one point. This would be the optimum case where the distance is zero. Due to the fact that all measurements are inaccurate, this case is not considered and is also just a special case of the second one.

For the second case, both lines do not intersect, they are skew lines. But it is possible to find two points on the two lines, which have a minimum distance to each other. Then, the real robot position lies near to those points. Figure 3.6 shows two lines, which do not intersect. The plane p is parallel to the line g and the line h intersects this plane in one point F_h.

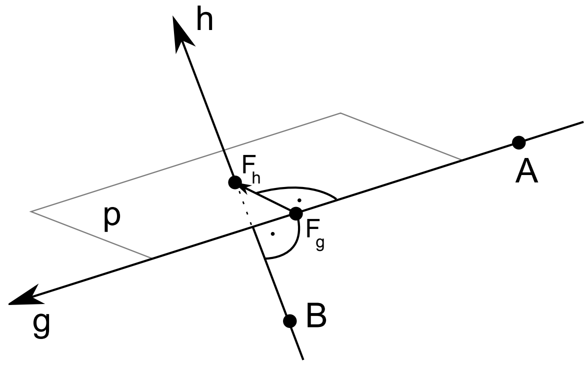

Figure 3.6: Skew lines

The vector n_0 is defined as:

$$\overrightarrow{n_0} = \overrightarrow{F_h} - \overrightarrow{F_g} = \overrightarrow{v} \times \overrightarrow{w} \tag{3.29}$$

The vector n_0 is orthogonal to the line g and the line h. In equation 3.29, the direction vector for line g is $\overrightarrow{v}$ and for line h the vector is $\overrightarrow{w}$. To get the normal vector of the plane p, the vector product between one direction vector and the vector n_0 is done. The plane p has the equation 3.30.

$$p : \overrightarrow{n_0} \times \overrightarrow{v} \cdot (\overrightarrow{x} - \overrightarrow{A}) = 0 \tag{3.30}$$

In the next step, the point F_h is by with intersecting the plane p and the line h.

$$p : \overrightarrow{n_0} \times \overrightarrow{v} \cdot (\overrightarrow{B} + \overrightarrow{w} \cdot r_1 - \overrightarrow{A}) = 0 \tag{3.31}$$

The variable r is proportional to the diving depth. The same procedure is done to the other point F_g with the equation 3.32.

$$p : \overrightarrow{n_0} \times \overrightarrow{w} \cdot (\overrightarrow{A} + \overrightarrow{v} \cdot r_2 - \overrightarrow{B}) = 0 \tag{3.32}$$

Finally, the resulting points and the two diving depths have to be combined. This can be done by taking one result or make an average. The average approach is the better one, because the error is minimized.

4 Robot control

The second task of this work is to have a position control of one robot. This is done by setting waypoints in the software and the computer sends commands to the robot that moves it to the destination point.

The manual control is an easy task. On pressing the forward key, both motors should drive forward, the backward key drives both motors backward and the turning keys drive both motors in opposing direction.

For the computer control there are many approaches. The next table shows three of them. Each of the approaches has advantages and disadvantages. On the following sections, these are discussed.

	Approach 1	**Approach 2**	**Approach 3**
Computer task	Send the current position and a set of waypoints	Send the error to the next waypoint	Sending motor control commands
Robot task	Waypoints and control of the motors	Only control of the motors	-
Video tracking system	Getting the position	Getting the position	Getting the position

Table 4.1: Approaches to control

Approach 1

In the latest development step this approach is the best one. All calculations are done by the robot itself. It gets only its current position and has to calculate all motor control actions. The waypoints are also handled by the robot. The computer sends all points once to the robot and then a start or stop command. If the waypoints are meant to be changed dynamically, the computer can also send updates of waypoints while the robot moves. One big disadvantage is that the control parameters like PD parameters are hardcoded in the firmware of the robot. Changing these parameters is a very time consuming part which includes reprogramming the robot. In the final phase, after sufficient testing / development, this approach can be used. Another disadvantage is the computation complexity. All control mechanisms have to work on a system with limited resources. But due to the modern 32bit high performance microcontrollers, the disadvantage is not that big.

Approach 2

The second approach relocates the waypoint control to the computer. It is now possible to apply quicker changes in the planned route or the waypoints itself. The computer makes the path planning and sends only the error to the next waypoint to the robot. The robot still has to compute the motor control. Disadvantages are just like in approach 1, the motor control parameters are still within the firmware.

Approach 3

All computations are done by the computer. On the robot, only the sensor handling is active. This approach is the most flexible. All parameters can be changed on the fly without reprogramming the robot. Also new control strategies can be implemented without reaching limits of computational power on the microcontroller. Waypoints can be easily set with the help of visualization on the computer screen. A the the advantages are numerous, this last approach is used in this thesis.

In section 4.1, a model for the robot is developed for future use. In this thesis, this model is..

4.1 Hydrodynamic equation

The hydrodynamics of the robot are useful to know, for setting up a model of the robot. Under water, the friction is very much lower than on the surface. The motors turn relatively fast compared to the actual driving speed. Measuring the propelling time, an estimation can be done, how far the robot drives.

4.1.1 Forces under water

To get an equation for the position of the robot, all forces which influence the moving robot have to be known. Table 4.2 shows the most important forces.

Force	**Depends on**
Inertial force	Acceleration and mass
Friction in water	Velocity
Propelling force	Motor speed and velocity

Table 4.2: Forces under water

The inertial force F_I is the same in water like on the surface. When the robot mass is accelerated, this force is working.

The friction in water is more complex. This force depends on the fluid, here water, on the shape of the robot and its squared velocity. There are also some other dependencies like temperature, which changes the density of the water and other factors. But due to simplicity and importance, these factors are not considered. The force is called drag force F_W.

For the propelling F_M force, the velocity is also needed. When the propeller is turning and the robot is not moving, it generates a force. The higher the robot's velocity gets, the smaller gets the relative velocity between the propeller and the robot. But this dependency is so small, it can be ignored.

All three forces have to be in equilibrium and the following equation shows this statement.

$$F_I + F_W = F_M \tag{4.1}$$

To get the position of the robot depending on all these forced, a differential equation is written.

The velocity is defined as

$$v(t) = \frac{d}{dt}(x(t)) = x'(t) \tag{4.2}$$

and the acceleration is defined as

$$a(t) = \frac{d}{dt}(v(t)) = x''(t) \tag{4.3}$$

Inserting 4.3 and 4.2 in equation 4.1 leads to the following

$$x''(t) \cdot m + (x'(t))^2 \cdot w = K \tag{4.4}$$

where m is the mass of the robot, w is the water resistance factor and K is the motor force. The units are $[m] = kg$, $[w] = \frac{N}{m/s}$ and $[K] = N$

4.1.2 Solving movement equation

The equation 4.4 is a second order nonlinear differential equation. To solve this equation, the homogeneous and particular solution has to be found. The homogeneous solution is

$$x_h(t) = \frac{m \cdot \log(C_1 \cdot m + t \cdot w)}{w} + C_2 \tag{4.5}$$

The non homogeneous solution is

$$x_h(t) = \frac{m \cdot \log\left(\cosh\left(\frac{\sqrt{K}\cdot\sqrt{w}\cdot(C_1 \cdot m + t)}{m}\right)\right)}{w} + C_2 \quad (4.6)$$

All coefficients have to be known to enable an exact estimation of the robots position. Some are easy to measure like the mass, others are more complex to calculate. In this case, an empirical method is used. Figure 4.1 shows the movement of the robot.

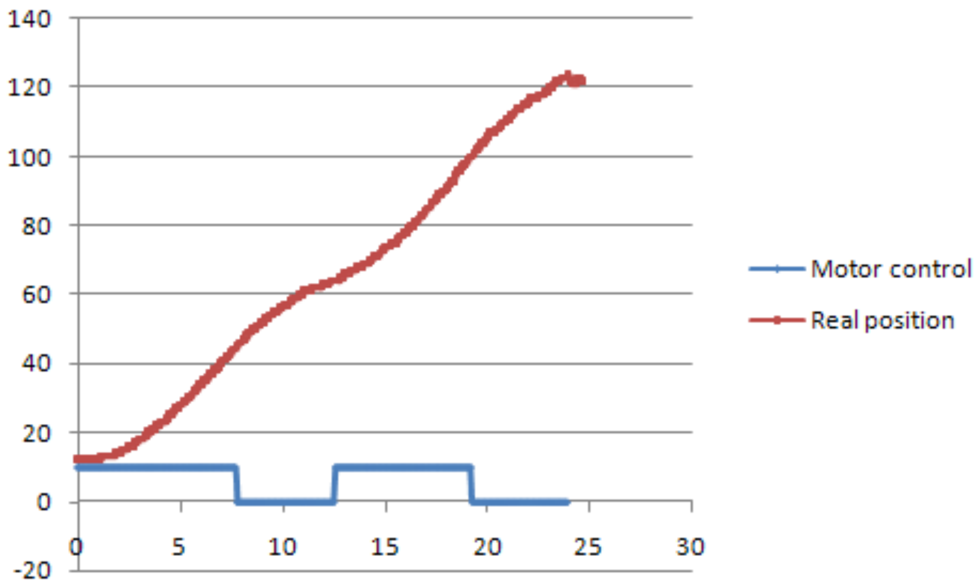

Figure 4.1: Real Position of the robot

The red line shows the x coordinate of the robot. The blue line the motor control.

In the next step, the parameters w, and K have to be varied, that the estimated position matches the real position. The mass m is known with $m_{robot} = 0.43Kg$. Figure 4.2 shows the estimated position of the robot with the blue line.

For better view, the estimated position has an offset of -5. The matching is very good. For future use, more measurements have to be made and the parameters have to be optimized. It is shown, that the motion estimation can be used as good model.

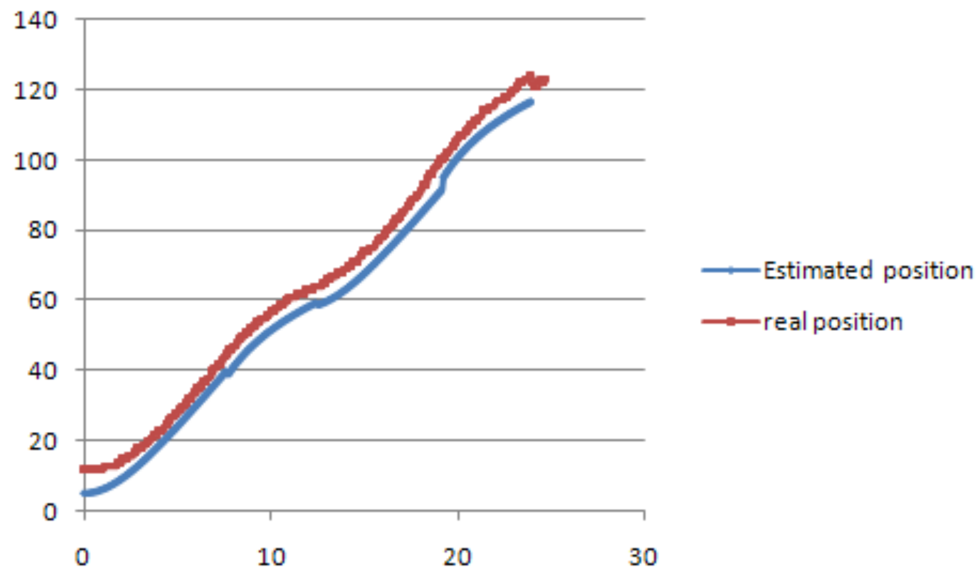

Figure 4.2: Real Position of the robot

4.2 Control strategie

Controlling the robots position is done with two independent controllers. One is for the heading direction and one is for the distance control. When they are combined, it is possible to let the robot drive to each point in the water without user interaction. Vehicle control underwater is quite different from normal robots on the surface. When the propeller stops turning, the robot continues driving or drifting. There is no direct brake, only propelling backwards. Also the turning behavior is different. On the surface a robot can steer directly to a direction. Underwater, the robot turns, but continues to drive straight ahead. The speed of the robot increases, if the motors turn in the correct direction or decrease the speed to zero and go backwards when continuing propelling. This behavior is like an integral one. The propelling time adds to the speed. So it can be said, the robot has an integral behavior in case of moving. In the next parts, different controllers are shown and explained, if they work for controlling underwater robots. All controllers are limited to simple ones, fuzzy control or nonlinear controllers are not described. In the next chart, the control scheme is shown.

On the left side, there are the two variables *xSet* and *ySet*. These come from the waypoint list and define the new target. From the video tracking system, the real coordinates are measured. The difference of the desired target and the current position is the error for both coordinates.

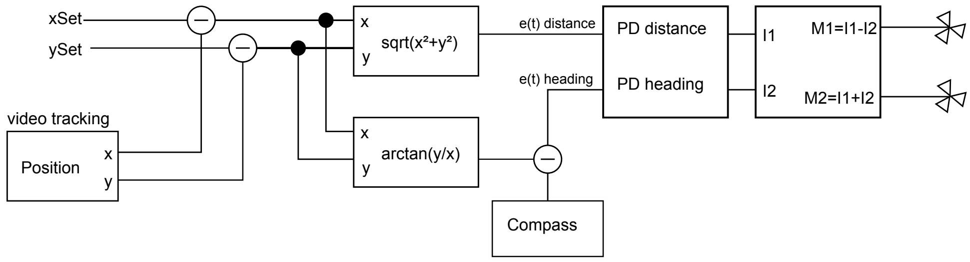

Figure 4.3: Controller overview

There is a branch to the rotational controller and the distance controller. The rotational controller uses the arctan function to determine the angle, in which direction the robot has to drive to reach the target. This angle is compared with the current heading from the compass data. The angle difference is the error for the rotation PD-controller. On the other branch, the distance is calculated. It is error term for the second PD controller. Both outputs from the controllers are connected together and for each motor, a signal is generated.

In the next sections, two controller types are described.

4.2.1 P-controller

The P-controller[reg05] is a very simple controller. The transfer function is

$$u(t) = P \cdot e(t) \tag{4.7}$$

where P is the constant and $e(t)$ is the error. The output of the controller is $u(t)$. For underwater robots, this controller is not sufficient. For example, if the robot drives to a point and gets nearer, the error term $e(t)$ gets smaller. At some point, the robot reaches the final position, so the error is zero. But now, the robot has still some velocity. So it drives further. The error term starts to get bigger, but now in the opposite direction. The propellers drive backwards, the robot comes to a stop and drives back to the destination. This goes on and on, the system is not stable. The problem is the low friction in water. Better results are achieved with a PD controller (see section 4.2.2).

4.2.2 PD-Controller

The PD-controller [reg05] has the transfer function

$$u(t) = D \cdot \frac{d}{dt} \cdot e(t) + P \cdot e(t) \tag{4.8}$$

This is a very good approach for controlling the robot. In the transfer function, the differentiating part can cancel out the integrating part of the water, if the parameters are set correctly. In the example from the P-controller, the robot drives near to the destination point. The error term e(t) gets smaller. When differentiating the error term, a negative value comes out. This negative part is added to the output of the controller. At one point, the output gets negative, so the robot breaks. This braking point is before reaching the target, not after the target. The PD controller can have a steady error. But in water, the velocity of the robot is controlled not the position. Therefore, this steady error does not affect the position control. This statement is only valid, when no current exists in the water. But inside the aquarium, the water is not moving.

5 Software

The software is written in C#. There are many programming languages, but C# was chosen because it is platform independent with the Microsoft .NET framework and the coding style is like C or C++. C is a common programming language and is used for programming the firmware of the robots. Also designing a GUI is very simple. Many libraries are free available on the internet available. For this project, the DirectShow and the LibUsbDotNet library are essentially.

Developing this software includs many steps. First, a list of requirement is created. Table 5.1 shows an overview.

Requirement	**Importance**
Show the robot position for each camera	+
Show the combined real position of the robot	++
Show the robot angle relatively to the aquarium	++
Show the diving depth	++
Mark the robots position inside the pictures to check, if the tracking works correctly	++
Possibility to save all data to a file for later evaluation	++
Camera number selection	++
Show template	+
Import ini file with coefficients	++
Contrast control with slider	+
Template rotation	-
Manual control of the robot	-
Set waypoints for a route of the robot	+

Table 5.1: Reqirements of the main software

All features with a plus or more have to be implemented in the software. Those with a minus are optional. In section 5.1, the user interface is shown and explained

5.1 Main software

The main software is used to control the robot inside the aquarium. In the next sections, a functional description is given.

5.1.1 GUI

The developed software is shown in figure 5.1. Each part of the user interface is discussed in more detail in this section.

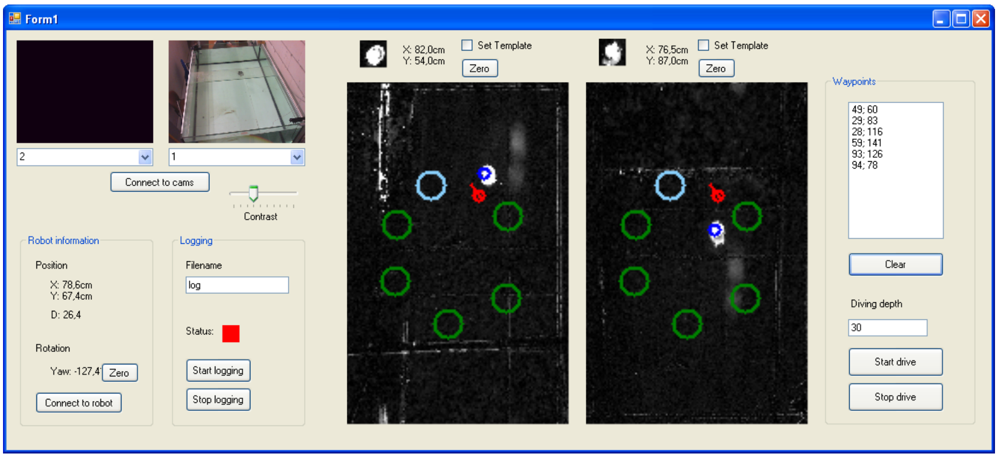

Figure 5.1: Main software GUI

The software consists of only one window, where everything can be controlled.

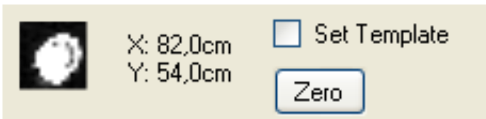

Figure 5.2: Main software - Template settings

Figure 5.2 shows the part of the GUI, where all settings for the template are made for each camera. On the left side of the figure 5.2 is the template shown, which is searched for in the camera picture. To the right of the template, there are the coordinates for the corresponding camera. When the program starts, the coordinates are set to zero.

On the right side is a checkbox to enable template update. If this checkbox is selected, the user can click in the camera picture to select the robot. The template is updated and the checkbox is automatically unselected. The 'zero' button is used to set the background picture, which will be subtracted during picture optimizing (see section 5.1.3). Before taking any measurements with the robot inside the aquarium, the picture has to be zeroed.

Data logging is important for later evaluation. On the top of figure 5.3, a filename has to be set. If the file already exists, then an increasing number is added to the filename. For example, if the file 'log' exists, the new file is called 'log0' and

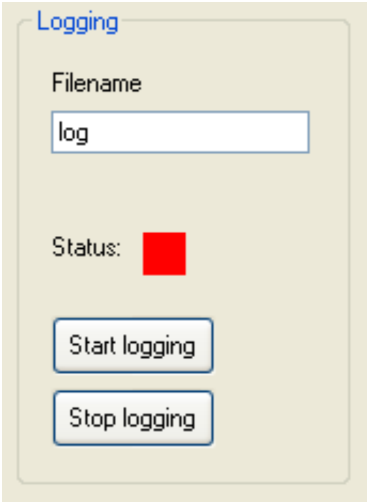

Figure 5.3: Main software - logging settings

the following file will be called 'log1'. The extension of all logging files is '*.csv'. CSV means 'comma separated value'. Instead of using a comma, a semicolon is used. When using the data in Microsoft Excel, a semicolon is the default setting for separation. The format of the log file is shown in table 5.2.

Time, date and column information					
-	d[0,0]	d[0,1]	$\cdots$	d[0,8]	d[0,9]
-	d[1,0]	d[1,1]	$\cdots$	d[1,8]	d[1,9]
			$\vdots$		
-	d[n,0]	d[n,1]	$\cdots$	d[n,8]	d[n,9]

Table 5.2: Logging file format

Beginning with some information about the measurement, the data is appended. The first line is the date and time of the measurement. All following lines describe one sample each. Because of the measuring rate of 10Hz, the file is extended with 10 lines every second. Table 5.3 shows, which value represents one sample. For evaluating purposes, the outputs of the two PD controllers are also logged.

Figure 5.4 shows the part of the GUI for waypoint control. If the user clicks on the cam's picture and the template checkbox is not checked, a waypoint is added to the waypoint list. The list is displayed and also visualized with green circles. The current waypoint, to which the robot drives is light blue.

The 'clear' button deletes all waypoints from the list. This can only be done, while the robot is in the stop state. To start the robot, the user clicks on the 'start

d[x,0]	Camera 1 X-Pos
d[x,1]	Camera 1 Y-Pos
d[x,2]	Camera 2 X-Pos
d[x,3]	Camera 2 Y-Pos
d[x,4]	Combined X-Pos
d[x,5]	Combined Y-Pos
d[x,6]	Diving depth
d[x,7]	Robot heading
d[x,8]	PD-controller rot.
d[x,9]	PD-controller dist.

Table 5.3: Loggine file sample data

drive' button and the robot drives to the first waypoint. Stopping it is done with the 'stop drive' button.

In figure 5.5, the 'robot information' field is shown. On the top, the calculated aquarium coordinates of the robot are shown. Below this, the diving depth is displayed. Further down, the robot's rotation is shown. Left to it, the user can click on the 'zero' button to reset the current angle to zero degrees. This is useful, when using different locations for testing. Zero degrees is in the direction of the x-axis.

On the bottom, the 'Connect to robot' button establishes the connection to the robot. Therefore, the USB library functions are called. Each USB device has a unique VID/PID (vendor ID and product ID)[pro] combination. This has to be given to the functions to find the correct device.

In figure 5.6, one of the two transformed camera pictures is shown. The big green circles are the waypoints. One of the big circles is blue. This is the currently activated waypoint. When the robot is near enough to the waypoint (radius 8cm), the next is automatically activated and the robot continues to the next one.

Clicking in the transformed camera picture has two different functions. One is to set a new waypoint and one is for template selection. Both click functions are described above.

5.1.2 Flowchart

A flowchart gives a better understanding of the flow of the program. In figure 5.7, the initialization and start of the main program is shown.

After the program is started, all arrays have to be initialized and memory is allocated. The dynamic allocation would always create a new array and the memory usage would rise constantly. In C#, the garbage collection could remove the unused space, but this doesn't work perfectly. All other variables, which are not set automatically to zero, are initialized. The program is ready to accept any user input like clicking on a button. Figure 5.7 shows two flowcharts, which work in parallel.

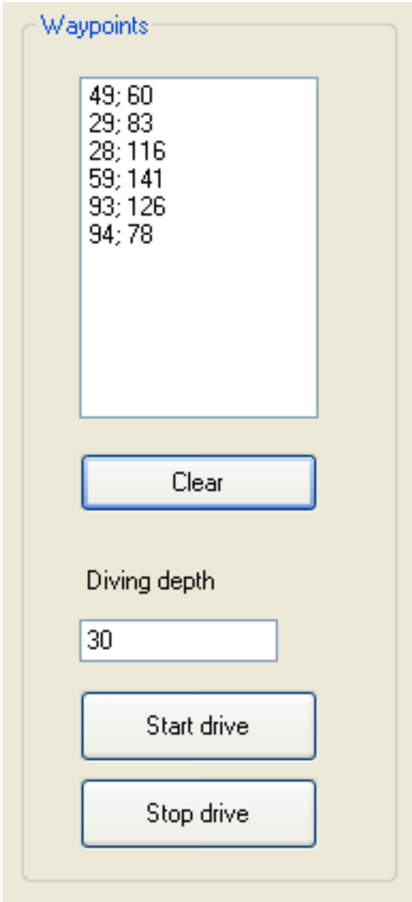

Figure 5.4: Main software - Waypoint settings

Therefore, an event is created, which is triggered by incoming data. This is done asynchronously to the main program.

The left flowchart shows the timer interrupt. In the first step, the data from the camera is read and copied to an array. This is done for both cameras. There is a temporary array, which holds the RGB data from the cam. This is copied and in the same step converted to black and white. The RGB24 format has three bytes, one for the red, one for the green and one for the blue part. The black and white value is calculated with the equation

$$BWvalue = \frac{R+G+B}{3} \tag{5.1}$$

This is the average of all three color parts. In the second step, the transformation matrix is applied to the array (see section 3.1). Then, the cross-correlation is done

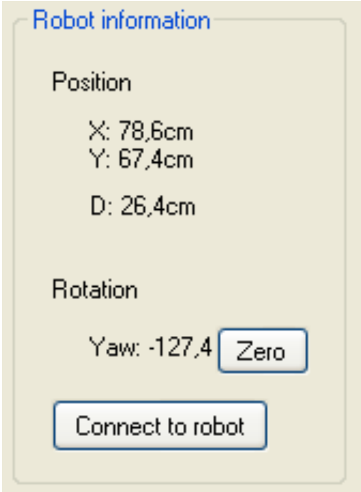

Figure 5.5: Main software - Robot information

and the maximum point is saved. The position is then averaged over four frames to have a smoother value and also for the D part of the distance controller (section 4.2.2, intermediate values are better. In real application, the robot moves about half a centimeter in each frame. Because the resolution is exactly one centimeter, each two frames is a step in the position. This would not lead to a sufficient motor control.

The next step is to display the results. In the end of the function, the motor control function is called. Parallel to this function, the receive data interrupt can occur. Here, the data stream from the USB RFM12 bridge (section 2.2.2) is read. The data consists of all sensor data. From this stream, the data is saved to variables and further processed.

5.1.3 Algorithms

The most important part of this thesis is the template matching. This is done with the help of the cross-correlation. An implementation in software is shown in algorithm 1.

There are two for-loops. Due to efficiency, the cross correlation is only done is a small area (see section 3.4). Line 1 and line 2 show, that the for-loops iterate only over this area. From line 6 to line 12, the two inner for-loops iterate over the template size. In line 8, the first part of the nominator of equation 3.12 is calculated. Because this term is also needed in the denominator, it is saved in a temporary variable. Continuing, in line 9 the sum of the nominator is calculated. Line 10 is only for calculation the standard variance 3.14 of the input picture. Now, the sum part is ended for the current x-y position. Line 14 discards negative correlation

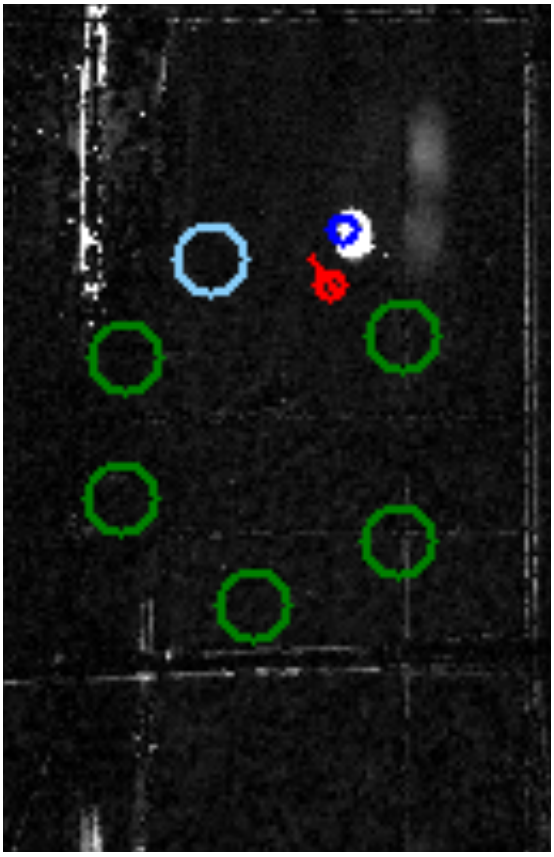

Figure 5.6: Main software - Transformed camera picture

coefficients. When this coefficient gets negative, the picture is inverted.

From line 16 to line 17, the rest of equation 3.12 is calculated. The value 250 in line 16 represents the scaling to a useful value. Because the array (*output*), which holds the correlation data, has the type 'byte', the value reaches from 0 to 255. The output of the cross correlation is a value from 0 to 1. So, multiplying it with 250, almost the full scale is used. Due to rounding errors, a little space to 255 is held.

Line 19 finishes the function. The output array is filled with the correlation coefficient. Within this algorithm, the maximum is continuously compared with the new value. If the new value is bigger, the maximum is updated.

The algorithm 2 for the robot control has to be called each time, a new position is calculated. It is called every 100ms. In line 1 and line 2, the errors for the x and y coordinate are calculated. The $realX$ and $realY$ value comes from the video tracking system and the $setX$ and $setY$ variable is from the waypoint control. In line 4, the heading of the robot is calculated, where the robot should drive and in line 6, the angular error is calculated. In line 7, the error for the distance control is calculated.

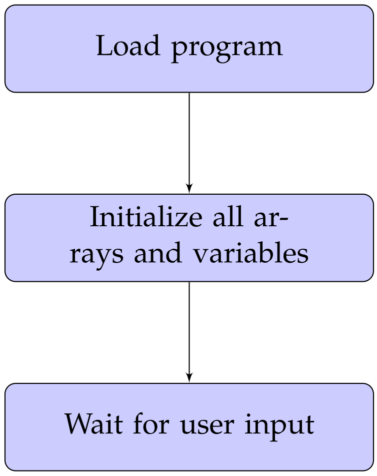

Figure 5.7: Main software flowchart (initialization)

One step later, the PD-control structure is applied (line 9 and line 12). Each is followed by a saturate function. This function is to limit the values to a range, which lies in the motor control range. For the LiLy robot, is is useful to limit the rotational control output to ±80% and the distance control output to ±50%.

The last step is to send the speed information to the motor controller. This is done in line 15.

Algorithm 3 describes, when a new waypoint is selected.

In line 1, the distance from the waypoint to the robot is checked. When the robot is near enough, the new waypoint is selected from the waypoint list. This list is an array, which contains all waypoints. The index is increased and wraps around, when the last element is reached. At this point, the drive route is repeated. This continues until the program is closed or the user clicks on the 'stop drive' button.

Line 4 to 5 sets the global variables $setX$ and $setY$ to the new coordinates. These are used in the robot control part (see algorithm 2).

To have a better tracking result, the pictures from the cameras are optimized. Algorithm 4 shows, how this is done.

The function consists of a loop, which iterates over each pixel of the image. In line 2, the background picture is subtracted. The absolute value is taken, because the byte array does not have negative values.

To increase the contrast, the subtracted result is multiplied with the contrast setting. A value of '1' is the original picture, higher values lead to an increased contrast. Line 3 is needed for contrast adjustment. When increasing the contrast, the pixel value may exceed the maximum for a byte. Due to this fact, the pixel has to be limited to 255. In line 4, the output array is filled with the new value.

Coordinate transformation is implemented in algorithm 5.

Algorithm 1 Normalized cross-correlation

1: **for** $y = area(top)$ to $area(bottom), step + 1$ **do**
2:     **for** $x = area(top)$ to $area(bottom), step + 1$ **do**
3:         calculate mean of input
4:         $SigmaPicture = 0$
5:         $CC = 0$
6:         **for** $v = 1$ to $TemplateHeight, step + 1$ **do**
7:             **for** $u = 1$ to $TemplateHeight, step + 1$ **do**
8:                 $temp = input(x + u, y + v) - \overline{input}$
9:                 $cc+ = temp \cdot (template(u, v) - \overline{template})$
10:                 $SigmaPicture+ = temp^2$
11:             **end for**
12:         **end for**
13:         **if** $CC < 0$ **then**
14:             $CC = 0$
15:         **end if**
16:         $CC = CC \cdot \frac{250}{SigmaPicture \cdot SigmaTemplate}$
17:         $CC = CC \cdot \frac{1}{TemplateHigh \cdot TemplateWidth}$
18:         $output(x + TemplateWidth/2, y + TemplateHeight/2) = CC$
19:     **end for**
20: **end for**

There are two loops, iterating over the width and the height of the output image. In line 3 and 4, the transformed coordinates are calculated. Then, in line 5, the output array is updated pixel wise with the new value. The variables a to h are the transformation coefficients from equation 3.7.

5.2 Additional software

For the determination of the transformation matrix, the corner points of the aquarium in the camera picture have to be known. Eight equations are needed to solved (see section 3.7). For this thesis, a software is developed, where the user can click on the corner points and the transformation matrix is generated. All coefficients are then stored to a file, which is used as a config file for the main software. Figure 5.9 shows the GUI for this software.

The user has to click in the picture to select a point. In the red circle with no. 1, the coordinates are displayed. Also, the user has to select, which corner this point belongs to. In the circle no. 2, already determined corner points are displayed. The left bottom point marks the origin of the aquarium.

Algorithm 2 Robot control

$errorX = setX - realX$
$errorY = setY - realY$

$robotHeadingSet = arctan2(errorY, errorX)$

$errorHeading = robotHeadingSet - robotHeadingReal$
$errorDistance = \sqrt{errorX^2 + errorY^2}$

$u_{rotation} = P \cdot errorHeading + D \cdot (errorHeading - lastErrorHeading)$
$saturate(u_{rotation})$

$u_{distance} = P \cdot errorDistance + D \cdot (errorDistance - lastErrorDistance)$
$saturate(u_{distance})$

$RobotSetMotors(u_{rotation}, u_{distance})$

Algorithm 3 Waypoint control

if $errorDistance < 8$ **then**
    $selectnextwaypointfromwaypointlist$

    $setX = newwaypoint.X$
    $setY = newwaypoint.Y$
end if

After selecting all corner points, the user has to click on the 'generate matrix file' button. In the same location as the program a text file with the name 'matrix.txt' is generated. The file has the format described in table 5.4.

The letters a to h are from the equation system in section 3.7. Each line represents a coefficient. The type of the variable is a double with a comma as separator (german notation). For the main software 5.1 it is needed to have two transformation matrices because the two cameras have different positions. Therefore, this additional software has to be started twice to get both matrices. The user can select the desired camera from a combo box (circle no. 3). Before starting it the second time, the 'matrix.txt' file has to be saved, because it will be overwritten.

In the figure 5.9, the aquarium is standing upside down. This is because the video stream is flipped. That is no problem, if the correct corner points are selected. When applying the transformation matrix, the picture is corrected automatically. Having all corner points selected, the program solves the equation system with the help of Gaussian elimination. This principle is explained in the section 3.2.

Algorithm 4 Picture optimizing

for $i = 1$ to $PictureWidth \cdot PictureHeight$ **do**
    $PixelValue = |input(i) - background(i)| \cdot ContrastSetting$
    $saturate(PixelValue)$
    $output(i) = PixelValue$
end for

Algorithm 5 Coordinate transformation

for $y = 0$ to $ArrayHeight$ **do**
    for $x = 0$ to $ArrayWidth$ **do**
        $x' = \dfrac{a \cdot x + b \cdot y + c}{g \cdot x + h \cdot y + 1}$
        $y' = \dfrac{d \cdot x + e \cdot y + f}{g \cdot x + h \cdot y + 1}$
        $output(x, y) = input(x', y')$
    end for
end for

5.2.1 Softwareimplementation of Gaussian elimination

When using the computer to solve a system with the Gaussian elimination principle, the algorithm 6 can be used.

The algorithm consists of two loops, one goes from line 1 to line 13. This is the elimination process. The second loop from line 14 to line 20 is the back substitution. From line 2 to line 5, the row with the greatest value at the top is found. This is important, because some equations have a leading zero, which will cause problems in a later step. After finding the *maxrow*, this row is swapped with the first one (line 7). In line 9 to line 11, two rows are subtracted, that one variable is eliminated. This is done for all remaining rows. In the next loop iteration of the first main loop, this procedure is repeated.

a
b
c
d
e
f
g
h

Table 5.4: 'matrix.txt' file format

Algorithm 6 Gaussian Elimination

```
for i = 0 to Nstep + 1 do
    for j = 1 to istep + 1 do
        if matrix(j, i) > matrix(maxrow, i) then
            maxrow = j
        end if
    end for
    Swapmatrix(maxrow, *)withmatrix(j, *)
    for j = i + 1 to Nstep + 1 do
        for k = N to istep − 1 do
            matrix(j, k)− = (matrix(i, k) · matrix(j, i)) / matrix(i, i)
        end for
    end for
end for
for j = N − 1 to 0step − 1 do
    temp = 0
    for k = k + 1 to Nstep + 1 do
        temp+ = matrix(j, k) · output(k)
        output(j) = (matrix(j, N) − temp) / matrix(j, j)
    end for
end for
```

The second loop is for back substituting. For applying this algorithm, the matrix A has to contain the vector $\overrightarrow{B}$ from equation 3.8 as additional column. The vector $\overrightarrow{X}$ is the output.

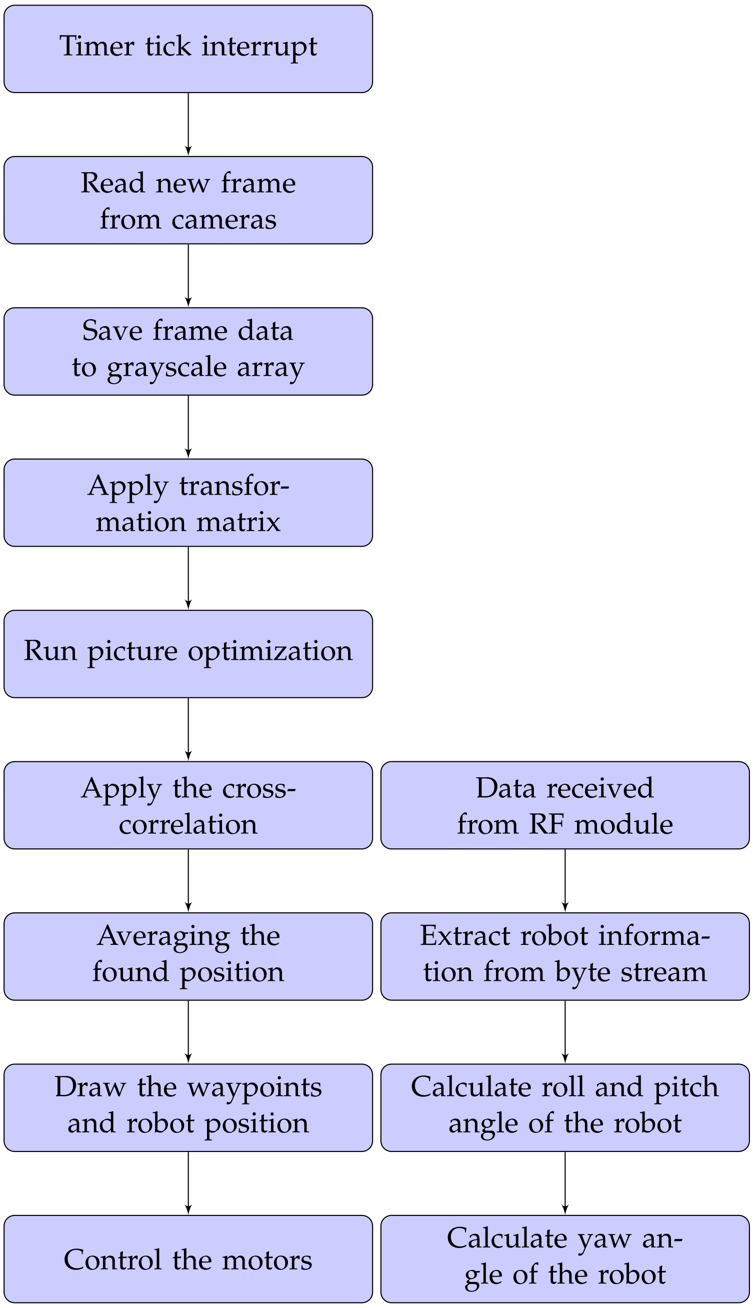

Figure 5.8: Main software flowchart

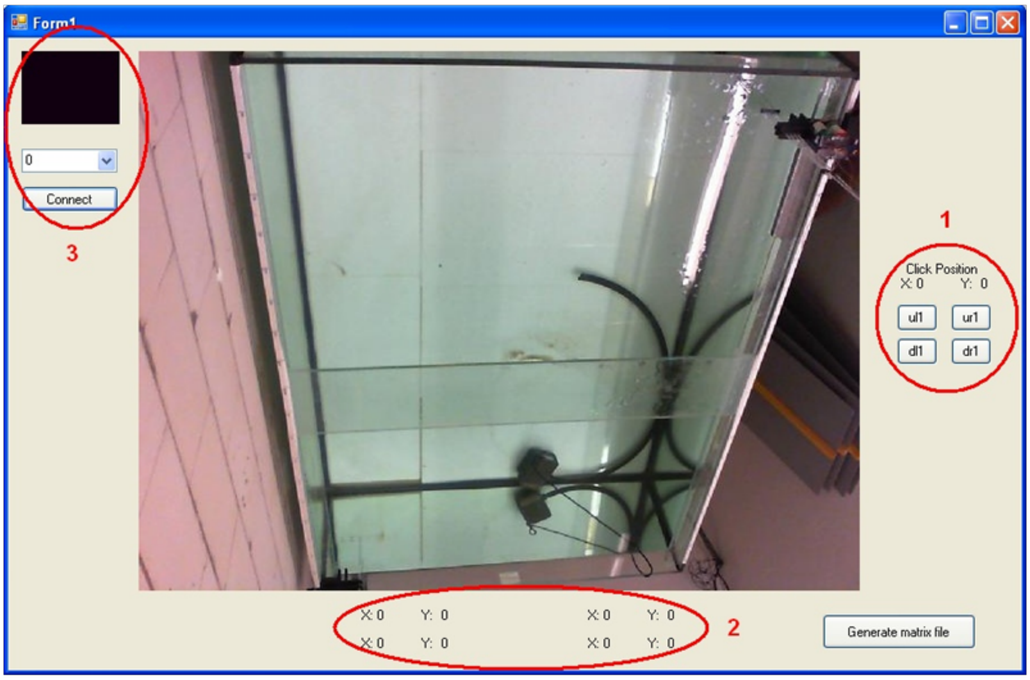

Figure 5.9: Transformation matrix solving GUI

6 Experiments

The software in chapter 5 is tested, if the robot is recognized in all situation. This includes floating on the water surface and diving. After evaluating the tracking system, the robot control is optimized. In section 6.2, the parameters for the controllers are optimized.

6.1 Video tracking

The video tracking system is evaluated by letting the robot drive through a set of waypoints. With the help of the logging file, the track of the robot is drawn into a diagram.

Tracking the robot works well, if the template contrast is high enough from the rest of the surroundings. In the template view (see figure 6.1), the current template is compared with the search window (see figure 5.6). The contrast enhancement of the camera picture is set to a value, which is sufficient, that in all conditions the robot is recognized. Figure 6.1 shows the robot for three different contrast settings. The value 1 means no increase in contrast.

Figure 6.1: Constast setting from left to right: 1, 2, 3

The higher the contrast, the more saturated the robot gets. Higher contrast values are useful, because the cross-correlation works more precise.

Problems can arise, when the robot dives. Figure 6.2 shows an example of shading and reflection problems. The aquarium is made out of glass. When the robot drives near to a wall and is diving, the reflection can be seen in the wall.

The deeper the robot dives, the bigger its shade gets. This brightness change is disturbing the template matching. Due to the fact, that the search area (see section 3.4) is very small, the reflections are usually ignored by the tracking system.

The shades does not influence the template matching process strong, so this is

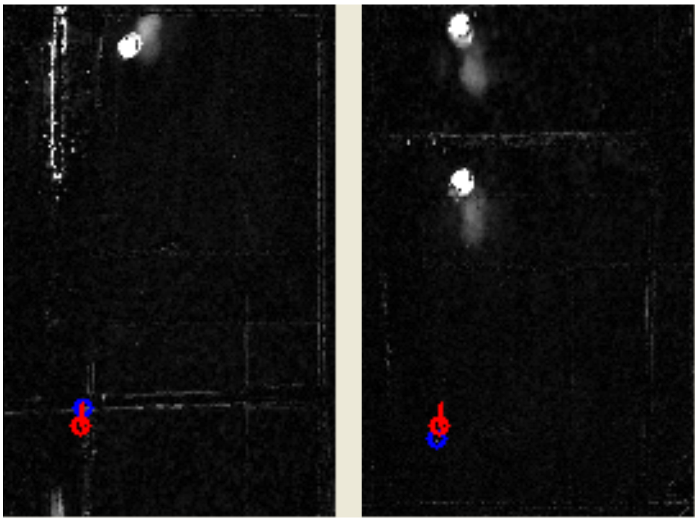

Figure 6.2: Reflections and Shading problems

6.2 Robot control

The results for the robot control were obtained by first evaluating the rotation controller. Therefore, the distance controller was set to zero, so that the robot can not move forward or backward. The setpoint for the rotation controller is toggled between two points, to get the step response for a positive and a negative step. This controller was optimized, that the turn is done fast, but with a small overshoot. The parameters for the PD controller were optimized by hand. In figure 6.3, a plot of the step response for turning the robot is shown.

For low values of the D part, the controller is swinging a lot and needs a long time to settle to its final position. The higher the D part gets, the faster the settling time. Limiting factor is the compass data. Noise on this data is directly amplified from the D-part. When the pump motor of the robot is acting, the compass is disturbed. This leads to changes up to 5^o in heading.

In figure 6.3, different values for the P and D-value have been tested. A value of $D = 1000$ has been taken empirically, where the robots behaviour is sufficient to have a good control.

Figure 6.4 shows the step response for a negative step. Compared to figure 6.3, the robot has to turn further, so during turning, the rotational speed gets higher. With $D = 800$, the PD controller starts again to swing. Therefore, higher values are recommend.

Controlling the heading of the robot works good. The distance controller does not have to work perfectly, because for reaching a waypoint, it is sufficient to get near to it. Due to the PD controller, the robot gets slower before reaching a way-

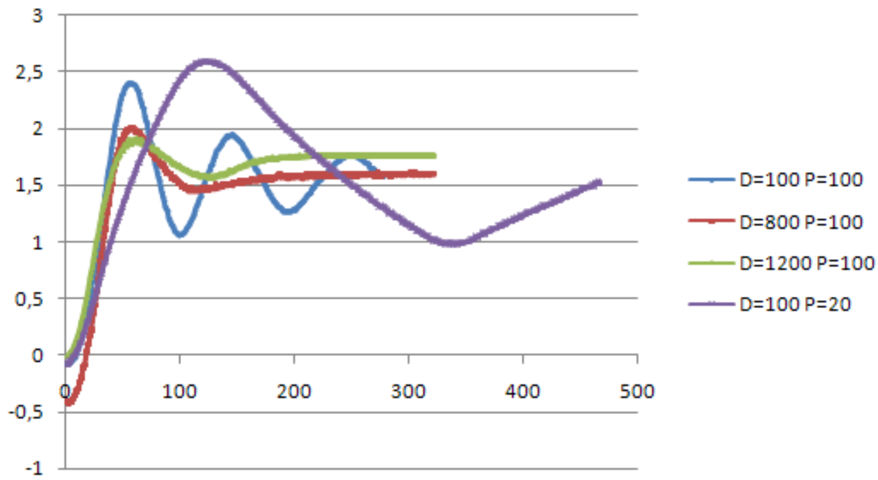

Figure 6.3: Rotation controller up step response

point. This is necessary, because if reaching the next waypoint includes a turn, the robot would overshoot. Its inertial force would let it drift further in the old direction. Figure 6.5 shows a tracked route from the robot. In this case, the robot did not dive.

The waypoints are marked with circles. In the center of figure 6.5 is she starting point of the robot. Each waypoint is crossed nearly in the center.

The next experiment, the diving depth was set to $30cm$. Figure 6.6 shows the travelling path of the robot. The result is much different than the route with no diving. A few time, the robot closely misses a waypoint, it has to turn around and drive in the opposite direction. This takes a lot of time. This problem comes from the side drift of the robot. Due to the low friction of the water, changing the direction needs time.

To evaluate the tracking, the diving depth can be used as measure. The depth controller of the robot holds the diving depth in a range between $1cm$ up and down. Figure 6.7 shows the tracked diving depth for diving $30cm$ deep.

The maximum error is about $8cm$. This error has its origin in the template matching algorithm. Viewing the robot from different angles, the center point of the robot changes. For extreme viewing angles, only small differences in matching lead to a great depth error.

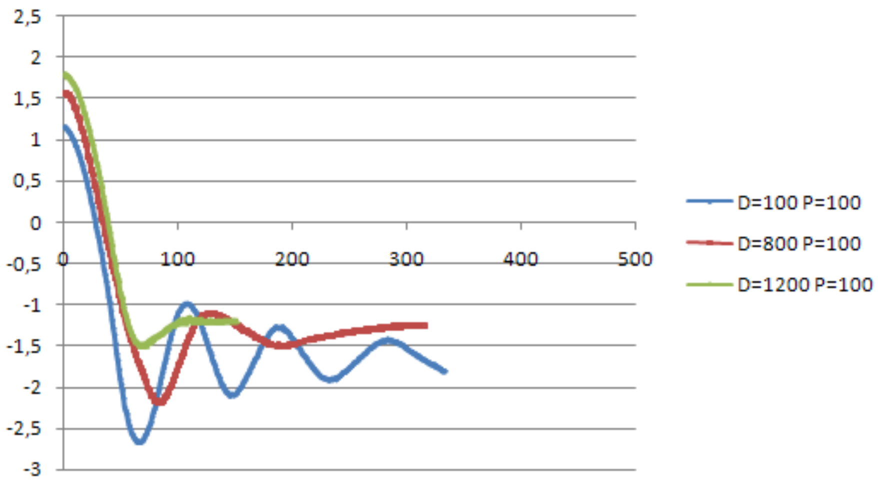

Figure 6.4: Rotation controller down step response

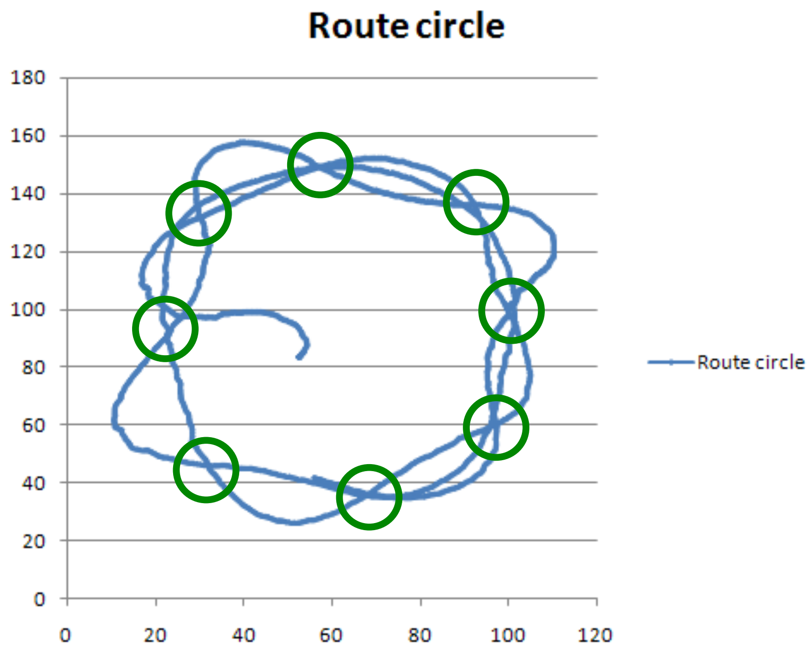

Figure 6.5: Waypoint route on the surface

Figure 6.6: Waypoint route with 30cm diving depth

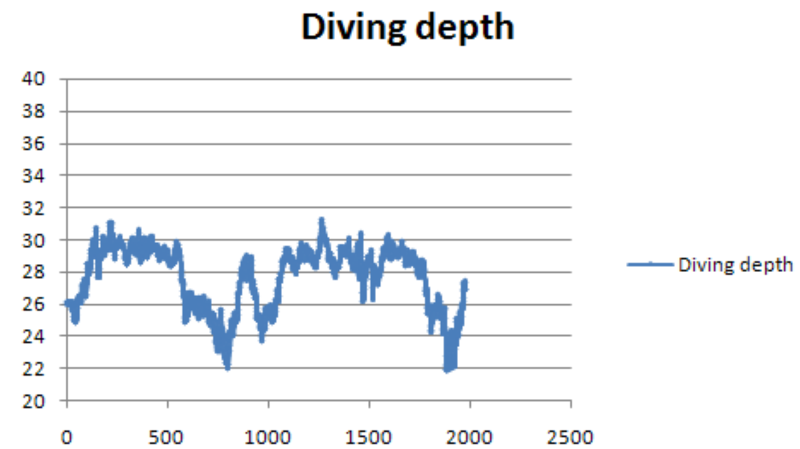

Figure 6.7: Depth calculation

7 Summary

In this thesis, a tracking system for underwater robot was developed and a robot control feature was implemented. The robot can follow a route, which is set with waypoints. All experiments are made with a small robot in a 140x200cm aquarium.

Two webcams are installed on the sides of this aquarium. The cameras are connected to a computer via USB. For the frame rate of the tracking system, 10Hz are chosen, because the robot moves with less than 10cm/s, which represents only a few pixel in the camera picture. Higher frame rates would bring a higher CPU utilisation, but no increase in performance.

A software was developed, which reads out the camera streams. The raw pictures contain perspective distortions. With a matrix transformation, the distortions are compensated. In the following step, the picture is optimized, so that only relevant information are visible. The background is subtracted and the contrast is increased.

Finding the robot in the camera picture is done with template matching. The normalized cross-correlation is used in the template matching algorithm. Contrast and brightness changes in the picture have only a little effect on the cross-correlation, because the normalization compensated these changes.

The position seen by the cameras is only a virtual position when the robot dives. With the help of the second camera and refraction equations, the real position of the robot is determined.

In the second part, a controller for the robot was developed. Two independent controllers, one for the heading and one for the distance to the target, produce a signal, which is combined and send to the motor driver. Both controllers are tested and perform well with only low overshoot.

The tracking system is tested and it works for all robot positions. During diving, the contrast of the robot is high enough, to be found by the template matching. Shades and reflections do not have an effect on the performance of this system. The system works well in the aquarium, where it was tested. It is shown, that the robot can follow a route of waypoint with the help of the tracking software.

7.1 Outlook

For the future, the system should be tested in other areas than a small aquarium. In the room with the aquarium, the ambient light was constant. In other placed, this might not be the case. So, the background subtraction may not work sufficient.

The next point is regarding the tracking of multiple robots. In the developed software, it is not possible to track more than one robot. It is mentioned, that the CPU utilization is so low, that tracking of more than one robot should be possible without performance problems.

Problems can occur, if multiple robots drive at different depth but at the same position. The robots can shade each other and the tracking system can be confused. With the motion estimation, the shading problem could be solved.

The waypoint system is working, but has only limited functionality. Currently, the waypoints only accept 2D positions. The diving depth is constant in the driving route. For the future, each waypoint could be a 3D point.

Bibliography

[blu01] *Bluetooth revealed: the insider's guide to an open specification for global wireless communication*. Prentice Hall PTR Upper Saddle River, 2001.

[coc] Cocoro project.

[com] *AN4248 - Implementing a Tilt-Compensated eCompass using Accelerometer and Magnetometer Sensors*.

[for96] *Formeln und Tabellen*. paetec, 1996.

[GG87] G. Carlyle Stones Georgia Griffiths. The tea-leaf reader algorithm: an efficient implementation of crc-16 and crc-32. Technical report, Compusec, Inc., 1987.

[JH] R.C. Flemmer J.W. Howarth, H.H.C. Bakker. Feature-based object recognition. Technical report, Massey University.

[KB01] Uwe D. Hanebeck Kai Briechle. Template matching using fast normalized correlation. Technical report, TU Muenchen, 2001.

[lib] Libusbdotnet library.

[mat06] *Mathematische Formelsammlung*. Vieweg & Sohn Verlag, 2006.

[MCN] R. Garcia M. Carreras, P. Ridao and T. Nicosevici. Vision-based localization of an underwater robot in a structured environment. Technical report, University of Girona, ?

[PB06] Vince W.C. Chooka Stefano Chessab Alberto Gottab Y. Fun Hua Paolo Barontib, Prashant Pillaia. Wireless sensor networks: A survey on the state of the art and the 802.15.4 and zigbee standards. Technical report, Mobile and Satellite Communication Research Centre, School of Engineering, Design and Technology, University of Bradford, United Kingdom Wireless Networks Laboratory, Istituto di Scienza e Tecnologie dell'Informazione, Pisa, Italy, Department of Computer Science, University of Pisa, Pisa, Italy, 2006.

[PCV07] Matthew Dunbabin Michael Hamilton Daniela Rus Peter Corke, Carrick Detweiler and Iuliu Vasilescu. Experiments with underwater robot localization and tracking. Technical report, CSIRO ICT Centre, Massachusetts Institute of Technology, University of California, 2007.

[pro] USB protocol. http://www.usb.org.

[PZ04] Jason Gu Pifu Zhang, Evangelos E. Milios. Underwater robot localization using artificial visual landmarks. Technical report, Dalhousie University, 2004.

[reg05] *Regelungstechnik 1*. Springer, 2005.

[RK] Kav'e Salamatian Ramin Khalili. Evaluation of packet error rate in wireless networks. Technical report, Universit'e Pierre et Marie Curie.

[sad] Fast sum of absolute differences visual landmark detector. Technical report, Australian National University.

[tim06] *Continuous-Time Signals*. Springer, 2006.

[WSPZ97] Deric Gray W. Scott Pegau and J. Ronald V. Zaneveld. Absorption and attenuation of visible and near-infrared light in water: dependence on temperature and salinity. Technical report, Optical Society of America, 1997.

Declaration

All the work contained within this thesis, except where otherwise acknowledged, was solely the effort of the author. At no stage was any collaboration entered into with any other party.

(Martin Antoni)