THE REALITY GAP OF AUTONOMOUS UNDERWATER VEHICLE MAKO AND SUBSIM

Markus Dittmar

Diplomarbeit Informatik (FH) 2008

Fachbereich Mathematik, Naturwissenschaften und Informatik,
Schwerpunkt Technisch-Wissenschaftliche Anwendungen,
FH-Gießen-Friedberg, Germany

Mobile Robotics Laboratory, Centre for Intelligent Information Processing Systems, School of Electrical, Electronic and Computer Engineering,
The University of Western Australia.

Autonomous Underwater Vehicle
Project Mako
The University of Western Australia
SEEK WISDOM
SubSim

Supervisor **Professor Dr. Wüst**
Co-Supervisor **A. Professor Thomas Bräunl**

Hiermit versichere ich, die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Perth, den 30. November 2008

Markus Dittmar

Acknowledgements

It has been a great challenge for me to work on two projects within in my thesis. There are a number of people I want to mention and thank for their help and support. *Mako* is a two-man project and without their help it would have been impossible to accomplish this thesis.

Firstly, I want to thank my friend Stephan Beisken. He spent many of his free sundays during the winter with me at the pool and most of the time in the pool. At least with hot chocolate and coffee I could keep him alive successfully.

Many thanks goes to the gentlemen in the mechanical and electronic workshops. Without their dedication and ideas many experimental setups would have been impossible.

I would like to give thanks to Thomas Bräunl for giving me the opportunity to conduct my final project at the University of Western Australia. It has been amazing to "play" with the robots from the lab.

Last but not least I want to express my thank to Adrian Boeing for his guidance and support with PAL.

Contents

Nomenclature

Acronyms

ADC	Analog-to-Digital Converter
Ah	Amp-hour
API	Application Programming Interface
AUV	Autonomous Underwater Vehicle
AUVSI	Association for Unmanned Vehicle Systems International
CPU	Central Processor Unit
DOF	Degrees of Freedom
EMI	Electromagnetic Interference
GPU	Graphic Processor Unit
GUI	Graphical User Interface
HDT	Hardware Description Table
LCD	Liquid Crystal Display
LSB	Least Significant Bit
MCM	Mine Counter Measures
PAL	Physical Abstraction Layer
PC	Personal Computer
PID	Proportional Integral Derivative
PVC	Polyvinylchloride
PWM	Pulse Width Modulation
RAM	Random Access Memory
ROM	Read Only Memory
ROV	Remotely Operated Vehicle

Nautical Terminology

Bow	Front side of vehicle
Stern	Back side of vehicle
Starboard	Right side of vehicle
Portside	Left side of vehicle
Surge	Motion in the longitudinal or x direction
Sway	Motion in the lateral or y direction
Heave	Motion in the vertical or z direction

Chapter 1

Introduction

"*Simulation is the process of designing a model of a real system and conducting experiments with this model for the purpose either of understanding the system or of evaluating various strategies (within the limits imposed by a criterion or set of criteria) for the operation of the system*" (R. E. Shannon 1975) [52]

1.1 Autonomous Underwater Vehicle

The CIIPS[1] group at the University of Western Australia in Perth develops submarines, so called "Autonomous Underwater Vehicles" (AUV). An AUV is a fully submersible, self decision-making submarine capable to act on its own [47]. Their manifold application in mining or for military purposes are only two indications for the importance of AUVs [56]. Compared to remotely operated vehicles (ROV) they act on their own and perform predefined tasks. These tasks are mostly in inaccessible or too dangerous areas for a crew. AUVs do not depend on a steady communication like ROVs do. The need of a steady communication harbours the risk of loosing the vehicle when the connection is interrupted. By using a whole array of sophisticated sensor systems AUVs are able to act autonomously via the provided real time data.

The construction of AUVs is still a mechanical and electrical challenge [37]. The environmental conditions and exceptionally physics under water demand cutting-edge technology. Therefore mastering these conditions is one of the most difficult tasks in the development of AUVs.

1.2 Motivation Mako

Project *Mako* is an initiative originated from the increasing demand of robot research, especially on AUVs. With the regard to commercial applications, the initiative wants to enforce research in the field of AUVs [29].

[1]CIIPS: Centre for Intelligent Information Processing Systems - http://ciips.ee.uwa.edu.au

The project *Mako* is mainly geographically restricted to the Australasian region. Its role model is a related project in North America, namely "Association for Unmanned Vehicle Systems International" (AUVSI) [11]. The mission tasks of the project *Mako* initiative can be found in appendix A.

The decision was made to split the competition into two different streams, given the rise to profit of either comparing simulated or physical AUVs. The basic research in the field of sensors and control methods - driven by the project Mako - might result in a better understanding of the essentials and might even offer novel approaches.

1.3 Motivation SubSim

SubSim is a simulation program for AUVs developed by the CIIPS group. Its main goal is to ease research and to provide a base system for testing and debugging. It was developed to allow running programs written for a real robot without any changes in the simulation.

Simulations appeal through their inexpensive and time saving properties. They are customizable to desirable conditions, and parameters can be adjusted with ease. Control programs can be tested without taking the risk of damaging or endangering the AUV.

SubSim is an acronym for "Submarine Simulator" and indicates for what it is build for. Most simulation programs available are only capable to target a certain task and involve a lack of flexibility. Therefore *SubSim* was developed to be as customizable as possible. The simulation emulates the *EyeBot* microcontroller which finds application in many different, distinguishable mobile robots developed by the CIIPS group.

1.4 Thesis Goals

Control optimization is often performed in simulated enviroments which can not be implemented into real autonomous robots. The conditions of the real world are often not simulated appropriatly, and real robots behave different in reality. The problem to transfer a behaviour of a robot into a simulation and vice versa is refered a reality gap [40]. Proper treatment of noise in the simulation however can close the this gap [44]. Brooks writes in [30]:

"*There is a real danger (in fact, a near certainty) that programs which work well on simulated robots will completely fail on real robots because of the differences in real world sensing ... sensors simply do not return clean accurate readings. At best they deliver a fuzzy approximation to what they are apparently measuring, and often they return something completely different.*"

In the opinion of P. Husbands and I. Harvey, mentioned in [43], a simulation has to meet, amongst others, the following three conditions:

- The simulation should be based on large quantities of carefully collected empirical data
- Appropriately profiled noise should be taken into account at all levels
- Noise added in addition to the empirically determined stochastic properties of the robot may help to cope with the inevitable discrepancies of the simulation

The latest release candidate of *SubSim*, version 2.3 , has no available error model, and a program written for *Mako* was never applied in *SubSim*. In an available "TODO" list for *SubSim* one of the past developer left the message:

"*TESTING! TESTING! TESTING! TESTING! It seems that NO ONE has ever checked if behavior of a real eyebot is the same as the one in the simulation*"

This initial situation formed the goals for this thesis: the system of *Mako* and its sensor suite has to be identified. Subsequent, an detailed error model for the simulation *SubSim* has to be designed, developed, implemented and tested. Finally, the behaviour of an available model of *Mako* for *SubSim* has to be compared with its real counterpart.

1.5 Outline of Thesis

In accordance with the order of the actual stages in this project, this thesis provides first an overview about the initial situation and back ground knowledge, and then proceeds to the experiments and model adaption.

Chapter 2 gives an overview about available AUVs and simulators. The initial situation of *Mako* and *SubSim* is described in detail.

Chapter 3 goes into details of sensor for mobile robots. Typical sensor errors are identified. Finally, the available sensor suite for *SubSim* and *Mako* is presented and discussed.

Chapter 4 shows the results of experiments designed and conducted with *Mako* and its sensor suite to identify the system.

In chapter 5 an introduced error model is presented. Results of experiments processed within the simulation are presented and compared with results of chapter 4.

Chapter 6 summarises the overall results and undertakings in this thesis and contains suggestions for future.

Chapter 2

Current AUVs and Simulation Systems

2.1 AUVs - an Overview

Some of the first versions of AUVs were developed in the 1970s e.g at the Massachusetts Institute of Technology [1]. In the beginning, most of the AUVs were designed for military purposes, primarily due to the fact of a higher budget available to them by companies, whom look for more cost effective alternatives. However, this trend began to change, and in the beginning of the 1990s AUVs became attractive for non-military purposes such as the mining and oil industries. With increased reliability and Mores Law, research labs were able to develop smaller and inexpensive AUVs [47]. In the last 20 years, a plenty of AUVs were developed by universities and companies alike. However, in spite of these encouraging trends only a few companies, even today, actually sell these vehicles, e.g. Hydroid Inc. [6], Kongsberg Maritime [7] or Bluefin Robotics [8].

One reason for this hesitant trend is the significant element of risk in the development and deployment of AUVs. Even the offshore industries, which are keen on looking for cost effective possibilities, began using the service of AUVs relatively slowly. The key point to stress in this discussion is the reliability factor. In order to accomplish this goal, intensive research is being carried worldwide especially in the topics of autonomy, object detection, energy sources, navigation and information systems. Countries all over the world run intensive AUV research and development programs, a clear indication of the growing importance and significance of these vehicles.

Typical applications of the AUV in scientific areas [56] include ocean modelling, ocean bottom exploration, ocean bottom sampling, underwater survey and hydrodynamics. Industrial applications involve conducting comprehensive 3D view of behaviour of oceans. Mining and oil industries deploy them for mineral exploration. In the military arena, AUVs found an important application in the Persian Gulf where they were used by the US Navy for mine counter measures (MCM) [32].

2.1.1 Commercial AUVs

The company Hybroid Inc. has been involved in the marketing of a series of AUVs, each of them designed for specific purposes and mission areas. The Remus 100 [6] was been designed to scale depths of up to 100 meters in coastal environments. It was constructed with the intention of being compact and lightweight; its maximum diameter of 19cm, a maximum length of 160cm and weight of 37kg allows economical overnight shipping and ease of deployment and recovery. With the application of numerous missions and mission hours it has proven its reliability and thus has become a cornerstone in the coastal AUV market. It can also be further configured to include a wide array of customer specified sensors like side scan sonar, conductivity, temperature and pressure sensors, but it can also be equipped with video cameras, inertial navigation system or global position systems (GPS). Typical applications are hydrographic surveys, harbor security operations, fishery operations and scientific sampling and mapping.

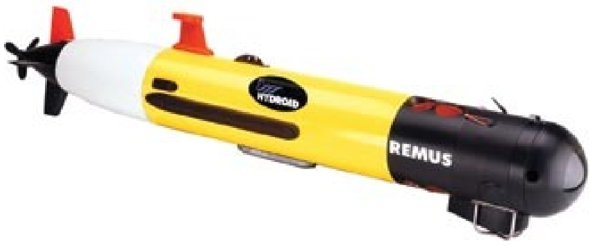


FIGURE 2.1: Remus 100 market by Hydroid Inc. [6].

Another notable AUV is the HUGIN 3000 Model manufactured by Kongsberg Maritime. It is a hybrid vehicle that can be operated in either supervised (ROV) or autonomous mode down to an operating depth of maximum 3000 meters. Its advantages are high speed and long endurance - up to 60 hours at full speed and all payload sensors running. It uses a high sophisticated navigation system. By virtue of its characteristics it is suitable for high-speed seabed mapping and imaging, inspection of underwater engineering structures and pipelines. It has also found myriad applications in the military area like mine counter measures or even anti-submarine warfare.

FIGURE 2.2: Deep water environment AUV, HUGIN 3000 Model [9].

2.1.2 Research AUVs

The AUVSI and ONR's 11th International Autonomous Underwater Vehicle Competition was held in August 2008. To gain some ideas and compare *Mako* with competing AUVs it is worth analysing this years winners. The team from the University of Maryland, USA, secured *Tortuga 2* [10] the first place [11]. Tortuga 2 was designed by Robotics@Maryland, a student-run organization at the University of Maryland. It was created to be highly manoeuvrable, regardless of speed or hydrodynamics, low cost, with an emphasis on simplicity and reliability.

A modular chassis was implemented to achieve an easy design and the possibility for future modifications. Aluminium was the choice of material. All electronics and batteries were located in an acrylic pressure hull, the central component of that AUV.

The electronic system itself was once again implemented in a modular configuration. It consisted of a backplane board and several plug-in cards for easy expandability. Another advantage of the modularity is the reduction of complexity and ease of repair. Plug in cards are e.g. a sensor board and a sonar board. The backplane performs the interconnection between the boards and supply them with power.

Low-level algorithms use the inertial guidance sensors to stabilize the AUV, thus allowing precise basic movements in the water. These sensors are a pressure transducer for depth measurements, and a MEMSense nIM nano Inertial Measurement Unit containing a three axis magnetometer, an accelerometer and an angular rate gyroscope. The accelerometer and the magnetometer are used to gain angular readings, the gyroscope measures and the angular velocity.

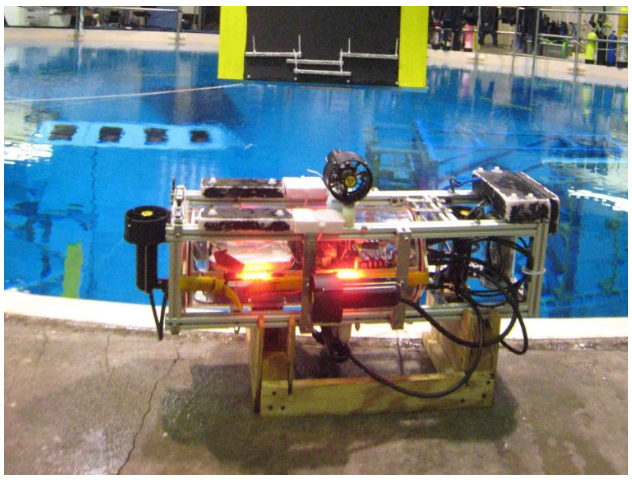


FIGURE 2.3: Tortuga 2 from the University of Maryland, USA [10].

The sonar system consists of a three sonar array to determine the direction of a pinger in a 2D plane. The vision system use two cameras - one faces in front while the other faces downward. The vision software was written in C++ and makes use of the *openCV* image processing library.

Tortuga 2 is not only able to translate fore/aft and up/down, through the placement of the thrusters, it can also translate int port/starboard direction and precisely change its orientation even when translating.

To accomplish its mission tasks, an AI (artificial intelligence) was implemented using the sensor data. The AI was written completely in Python, thus making it unnecessary to recompile the code after changes. The batteries last for two hours during underwater operations.

A good overview over other current AUV research projects and groups can be found at [12].

2.2 Mako Abstract

Mako had to be designed and built up from scratch. This included the mechanical and electrical system as well as the sensor suite. *Mako* was modelled and built by Louis Andrew Gonzalez in 2004 during his final year project [39]. His work included the mechanical and electrical design. Both had to be done simultaneously to prevent mutual exclusion. Several other constraints influenced the choice of the design. Since *Mako* was also constructed with the possibility to participate on the AUVSI competition, the rules from 2004 limited the design of participating AUVs. These limits covered the dimensions and the weight that each AUV entry must fit within a box of 180cm x 90cm x 90cm. The AUVSI rules from 2008 [11] limit the dimensions of a box to 183cm x 91cm x 91cm and additionally limit the weight to a maximum of 50kg.

Besides these constraints the main goals for the design were [29]:

- Ease in machining and construction
- The ability to survive in salt water
- Ease in ensuring watertight integrity
- Providing enough space for the equipment
- Providing static and dynamical stability
- A modular design to allow easy design changes
- Cost effective
- Long lasting batteries

Mako measures 134cm long, 64cm wide and 46cm tall with a total mass of 35kg. An aluminium skeletal frame was chosen due its lightweight characteristics and its resistance to corrosion. It provides high modularity thus allowing easy design changes and the provision of mounting extra external devices. Since the tasks of the AUVSI changes every year, the ability of changing the setup was almost a pre-requisite.

Two PVC cylinders provide enough space for the internal equipment and ensure an ease in terms of watertight integrity. The upper PVC cylinder contains a micro controller, a sonar multiplexer and vision system as well as a couple of internal sensors. Theses devices are mounted on an aluminium tray which can be removed with ease. Thus it is possible to have quick access to the devices (appendix F).The lower cylinder contains the heaviest part, the batteries. By mounting the batteries under the center of buoyancy and thus moving the center of mass as well under the center of buoyancy it is ensured that *Mako* always stays in an upright position. Three sealed lead acid batteries providing a capacity of 31Ah end ensures sufficient operating time.

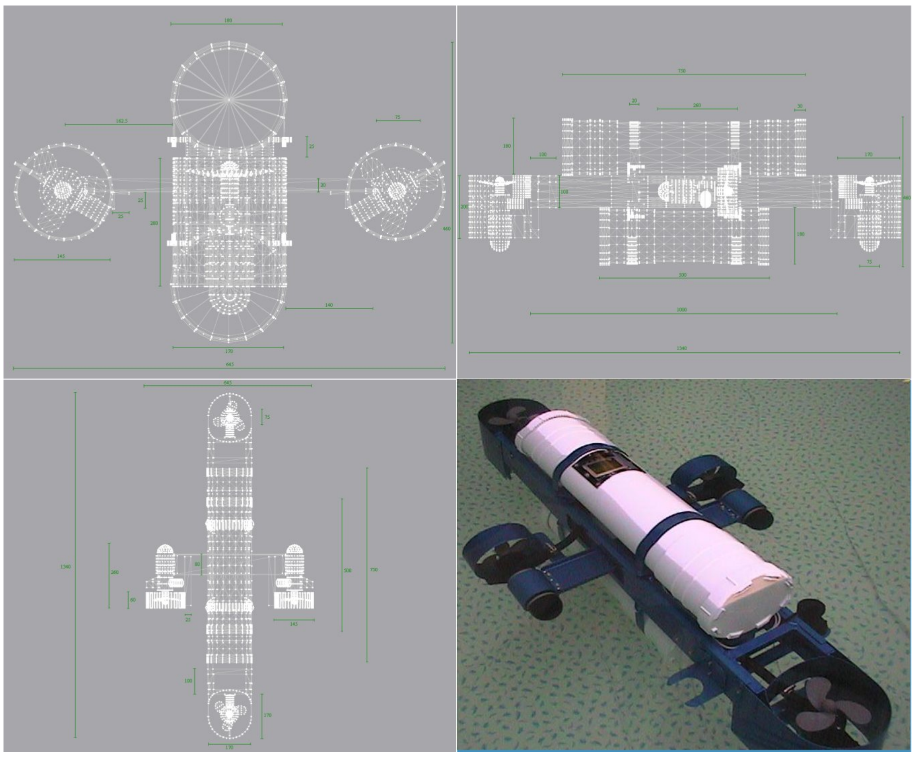


Figure 2.4: Mechanical system schematics of *Mako* [39].

2.2.1 Propulsion and Range of Motion

The propulsion system consists of 4 trolling 12V and 7A DC thrusters of the same kind. Their original application in water ensures watertight integrity and allows forward as well as backward movement. Two thrusters are aligned horizontally, one on the port side, the one on the starport side for fore/aft translation. The other two thrusters are aligned vertically on the bow and stern, and thus facilitates a translation in the up and down directions.

The decision was made not to use a ballast tank and single thruster system with rudders mainly due to the advantage of easier control over the submarine. It also made the design of mechanical systems much easier and thus reduces the complexity in terms of watertight integrity. A disadvantage that arises is a higher consumption of energy during diving tasks, since both, bow and stern thrusters, have to apply a steady force to keep the submarine under water.

The current thruster setup gives *Mako* four controllable degrees of freedom (DOF):

- Movement in x-direction by setting both port and starboard thruster to the same speed and direction (surge)
- Movement along the z-axis by setting both bow and stern motors to the same speed and direction (heave)
- Turning around the z-axis is possible by setting different speeds and/or directions to port and starboard thruster (yaw)
- Turning around the y-axis by setting different speeds and/or directions to bow and stern motors (pitch)

Due to the fact that the center of mass is lower than the center of buoyancy the innate metacentric righting moment passively controls roll and pitch. *Mako* lacks the ability to translate in port or starboard direction but is still able to navigate to all world directions and positions with the four DOF. The speed of the thrusters is set by a pulse width modulation (PWM) with inverted logic. That means setting the modulation to a value of '0' produces maximum thrust.

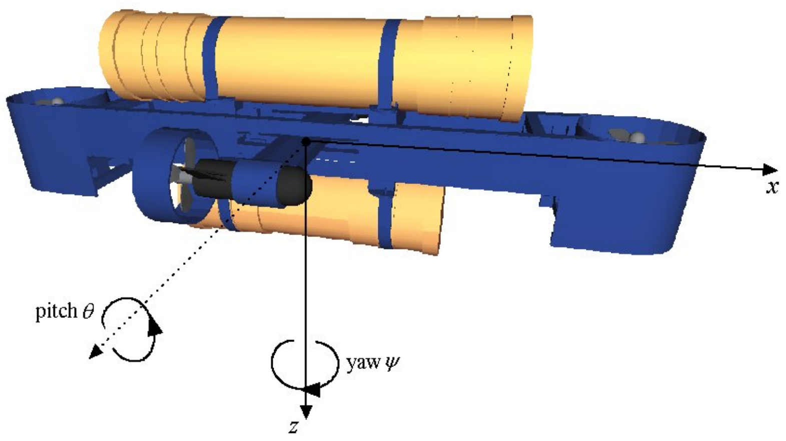


FIGURE 2.5: Range of Motion of *Mako* by the use of its thrusters(4DOF) [39]. modularity

2.2.2 Controllers

The control system is split into two parts. An *EyeBot MK4* microcontroller is responsible for reading sensor data and controlling the thrusters while a *cyrix* mini PC provides the necessary computation power needed by the vision system. It was a logical decision to separate these two independent systems and use the power of distributed computation. To split these tasks implies a reduction in complexity and thus simplifies the writing of applications for the *EyeBot* microcontroller and the vision system.

The *EyeBot* microcontroller runs in C or C++ written and compiled programs to navigate *Mako* through its task, reading data from the sensors over analog, digital or serial ports and setting the speed to the thrusters by changing the pulse width for the PWM. The programs are transmitted serially to the controller, either with a cable or preferably a bluetooth connection. The latter in particular is quite handy if the controller is mounted inside *Mako*.

The specifications of the *EyeBot MK4* microcontroller [28]:

- Operating system: RoBIOS
- 25MHZ 32bit Controller (Motorola 68332) overclocked to 35MHZ
- 512KB ROM for system + user programs, 2048KB of RAM
- 8 digital inputs
- 8 digital outputs
- 8 analog inputs
- 2 serial ports
- Graphics LCD (128x64 pixels)

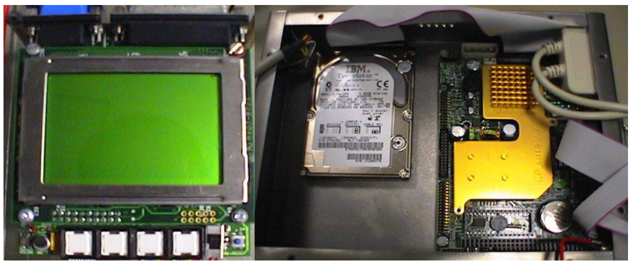


FIGURE 2.6: left: *EyeBot* controller, right: mini PC [39].

The RoBIOS operating system offers a low-level API as well as a powerful high-level API and simplifies developing applications enormously by offering interfaces to almost all available sensors. A detailed documentation can be found under [28]. Each robot using the *EyeBot* controller is coupled with a *hardware description table* (HDT). It is used to define low level attributes for sensors and actuators, and offer the possibility to balance their different behaviours using these sensors or actuators.

The Cyrix mini PC runs with 233MHz and uses 32MB of RAM and thus has sufficient power for computationally intensive image processes. The operating system is the Red Hat Linux distribution using Kernel 2.4 without any graphical user interface (GUI).

2.2.3 Sensor Suite Overview

In recent times there have been a huge variety of sensors for mobile robots [38] [27]. Sensors for mobile robots has been a major topic in the past, and intensive research and development has taken place in this field. However, the most difficult challenge is in choosing the right sensor for the right application. Since the competition tasks of the AUVSI changes every year, it is necessary to change the setup of the AUV accordingly. This section gives only a short overview over the current mounted sensor on *Mako*, and a detailed discussion will be provided in chapter 3.

Mako comes along with a couple of sensors that are essential to accomplish its mission tasks. These sensors are connected directly or indirectly to the *EyeBot* microcontroller and can be accessed by the RoBIOS API. The following sensors are currently mounted:

- Digital compass
- 3 axis accelerometer
- Velocity sensor
- Depth sensor
- Four sonar transducer, respectively one facing towards the front, backwards, left and right

The vision system was implemented by Daniel Loung Huat Lim in 2004 [46]. However, this will not be a part of this thesis primarily due to the fact that the camera is currently non functional. Ivan Neubronner, a senior technician at the electronics workshop of the school of electrical, electronic and computer engineering, was involved with the construction of *Mako*. According to him, the camera just might be not powered. It was not possible to go into the matter any further since the back side of the upper PVC hull, where the electrical power was distributed, is sealed.

2.3 Other Simulations

To get the necessary background and to spot ideas to fill the reality gap in *SubSim* it was useful to have a look into other available simulations for mobile robots. Currently there is a wide variety of different simulators available for robots. Most of these are designed for the Microsoft Windows operating system but using the operating system independent OpengGL graphics library (Open Graphics Library) [31] [13]. They find their applications in military, commercial and scientific areas. Many universities, such as the Naval Postgraduate School in California, USA, or the University of Tokyo, Japan, to name a few, have conducted, their own simulations for their AUVs.

Due to the strong limitation in communication with autonomous and nearby autonomous underwater vehicles, simulations try to minimize the operating risk since the operating environment is extremely unforgiving and bears many, sometimes unexpected risks. By closing the reality gap between offline simulations and real robot, real time simulations reduce developing time and thus developing costs dramatically [24].

Rainer Trieb and Ewald von Puttkamer from the University of Kaiserslautern classify their *3d7- Simulation Environment* [51] into four distinguishable system aspects.

Characteristics of the operating environment : the simulated operating environment must be abstracted from the real environment and mapped to an appropriate representation. Simplicity and realism have to be balanced

Configuration of the sensor system : to configure a sensor system it is necessary to model individual sensors with the possibility of independent sensor configurations. The purpose is to gain realistic output data which can be used for control programs. To achieve this geometrical, physical and stochastical methods are used to describe the behaviour in a real environment

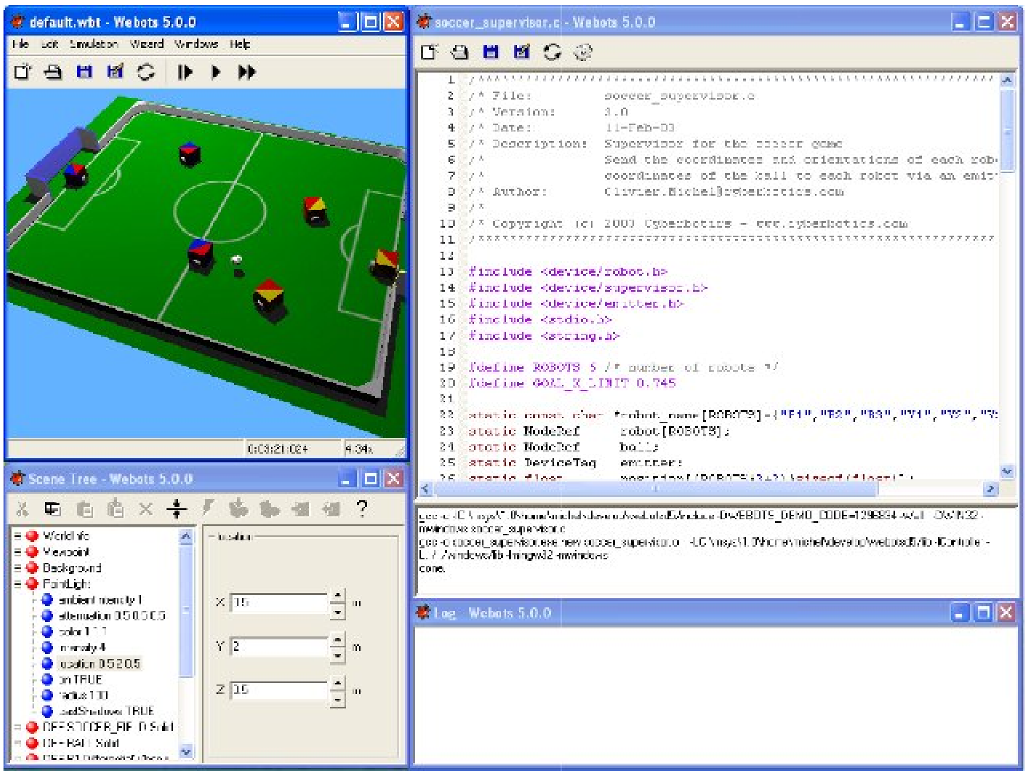
Type of locomotion: the simulation of the locomotion system is necessary to develop control applications for navigation and obstacle avoidance

Partition of the control structure: the implementation of the control program

2.3.1 Webspots 5 Simulation

Webspots 5 is a commercial robot simulation, developed by Cyberbotics [14] and the Swiss Federal Institute of Technology in Lausanne. It finds application in over 450 universities and research centers around the world. Due to the maintenance of over eight years, this simulation has proven to be both reliable and robust. It allows custom defined robots, including the physical and graphical modelling. Its own software library enables the transfer of the controller program, written for the simulation, to several commercial mobile robots, e.g. Lego Mindstorms or Khepera, and comes along with its own IDE. External IDEs are supported as well. Control programs can either be written in C, C++ or even Java.

It runs simultaneously several, different mobile robots in the same environment and allows the communication amongst these. Wheeled, legged and flying robots are currently supported. The available sensors are infrared and ultra sonic sensors, range finder, light sensors, touch sensors, GPS, cameras including stereo view or 360 degree vision systems, position and force sensors for servos and incremental encoders for wheels.



```
/**********************************************************************
/* File:          soccer_supervisor.c
/* Version:       3.0
/* Date:          11-Feb-03
/* Description:   Supervisor for the soccer game
/*                Send the coordinates and orientations of each rob
/*                coordinates of the ball to each robot via an emit
/* Author:        Olivier.Michel@cyberbotics.com
/*
/* Copyright (c) 2003 Cyberbotics - www.cyberbotics.com
/**********************************************************************

#include <device/robot.h>
#include <device/supervisor.h>
#include <device/emitter.h>
#include <stdio.h>
#include <string.h>

#define ROBOTS 5 /* number of robots */
#define GOAL_X_LIMIT 0.745

static const char *robot_name[ROBOTS]={"B1","B2","B3","Y1","Y2","Y
static NodeRef      robot[ROBOTS];
static NodeRef      ball;
static DeviceTag    emitter;
static float        position[(ROBOTS*3+2)*sizeof(float)];
```

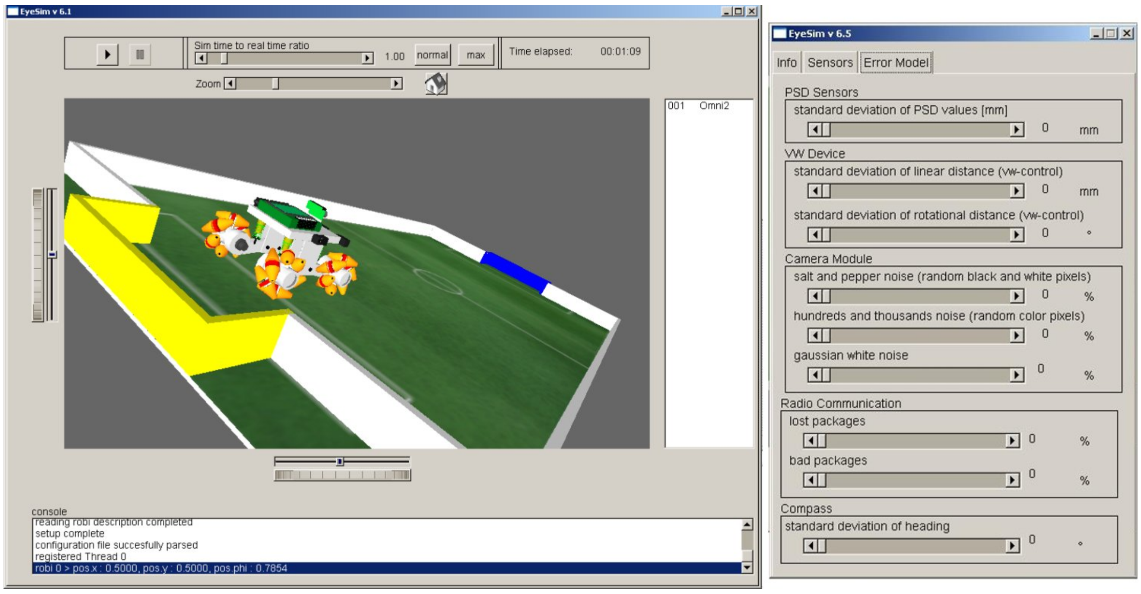
```
gcc -c -I[illegible] -DWEBOTS_DEMO_CODE=1298834 -Wall -DWIN32 -
mwindows soccer_supervisor.c
gcc -o soccer_supervisor.exe new soccer_supervisor.o -L[illegible]/webots5/lib -lController -
L../../windows/lib -lmingw32 -mwindows
done.
```

FIGURE 2.7: Webspots 5 Simulation Software market by Cyberbotics [14].

The software makes use of the OpenGL graphical library and includes its own 3D editor for modelling objects. Each component of an object is considered individually and takes into account different mass distribution, static and kinematic friction coefficients, whereby the Open Dynamics Engine (ODE) takes over the physical simulation. Changing the simulation speed allows simulations either in slow motion or up to 300 times realtime. A supervisor program allows changing the environment during the simulation and comes very handy by computational expensive simulations like neural networking, where a lot of training data is necessary.

2.3.2 EyeSim

Next to *SubSim* a couple of mobile robot simulations were developed at the robotics research group of CIIPS. All of them make use of the RoBIOS operating system and allow developing control programs using the same SDK. *EyeSim* is one such simulation and was developed in 2004 to simulate mobile robots [45]. It is able to simulate all available sensors and actuators supported by the RoBIOS including a virtual camera. The camera allows real time image processing. Like Webspots 5, EyeSim supports multi robot and multi tasking simulations simultaneously and allows a simulated wireless intercommunication between the robots. As other simulations developed at the robotic department, applications are written in C or C++ and apply the RoBIOS API. The completely functions and graphical representation of the *EyeBot's* LCD were implemented as well. Most simulations run control programs and simulation in different applications or processes. EyeSim instead dynamically links the control software during the runtime and executes it in one application.



```
reading robi description completed
setup complete
configuration file succesfully parsed
registered Thread 0
robi 0 > pos.x : 0.5000, pos.y : 0.5000, pos.phi : 0.7854
```

PSD Sensors
standard deviation of PSD values [mm] 0 mm

VW Device
standard deviation of linear distance (vw-control) 0 mm
standard deviation of rotational distance (vw-control) 0 °

Camera Module
salt and pepper noise (random black and white pixels) 0 %
hundreds and thousands noise (random color pixels) 0 %
gaussian white noise 0 %

Radio Communication
lost packages 0 %
bad packages 0 %

Compass
standard deviation of heading 0 °

FIGURE 2.8: left: EyeSim simulator version 6.5, right: the detailed error model

Robots and environments can be customized with different parameter files and presented through a 3D visualization. Again the OpenGL 3D library was used for the 3D drawings. Milkshape[1] shareware was used to provide 3D polygon representations for objects and robots. Much effort was spent on the error model to emulate a realistic environment for the robots. The error model is based on statistical behaviour descriptions and can be applied on sensors, actuators, the wireless communication and the virtual *EyeBot* camera. Next to the error model a debugging mode was implemented for sensor data visualization. It enables, amongst others, the graphical representation of PSD beams (Positional Sensitive Device) and the visualization of the camera frustum, and thus assists the user in terms of developing and debugging.

2.4 Subsim

SubSim was designed and developed in 2004, during the same time as *Mako*, with the goal to be extendable and flexible as possible [29]. Many different projects mostly done by final year project students enhanced the *SubSim* to version 2.3. This section describes the status of *SubSim* at the beginning of the thesis. The implemented extensions and changes will be discussed in chapter 3 and 5.

[1]http://chumbalum.swissquake.ch/

2.4.1 Requirements, Restrictions and Shortcomings

Some basic requirements and important properties for the design were defined before the actual implementation could be done:

- The simulation must be able to simulate an environment for the AUV with all its sensors and special physical underwater properties
- Additional passive objects like buoys, ships or pipes should be able to be placed
- Any *EyeBot* program written for an AUV has to be executable without any changes on its real counterpart
- Therefore the *EyeBot* LCD must be emulated with all its functions
- The user must be able to watch the scene from different perspectives
- The movement of any objects must be visualized
- The user should be able to observe the sensor data
- The user should be able to change the speed of the simulation
- Position and orientation of AUVs should be changeable by the user
- It must be highly extensible through plugins and provide appropriate functions to access the simulator
- Pre-setups through customizable settings files must be provided
- A developer documentation should be provided

Version 2.3 fulfils most of the above mentioned points, however some functionality is still missing. The execution of any *EyeBot* programs is not completely possible. This is mainly due to the reason that some sensors, e.g. the sonar system, used by *Mako* were not part of RoBIOS. Furthermore a sonar system was not implemented in version 2.3. The possibility to observe sensor data was implemented and was working in previous versions, however this was observed to be broken in this version. The functionality to set the position and orientation of AUVs was not implemented. A user documentation to setup the simulation through its settings files as well as a documentation on how to implement a new AUV model are available. However, a full up-to-date documentation for *SubSim* is currently not available.

2.4.2 Definition of Objects

To understand the model design and concept of *SubSim* it is necessary to define and explain its inherent objects. The environment is referred as *World*. It contains other objects called *WorldObjects*. These *WorldObjects* are distinguished as passive and active objects. Passive objects are such as the buoys and ships. Active objects like submarines extend the properties of the world and are referred as *WorldActiveObjects*. Each *WorldActiveObject* is attached with appropriate descriptions for its sensors and actuators and is coupled with a client dynamic link library (DLL), containing the control program, and a HDT.

2.4.3 Software Architecture

SubSim is a single process application. The physics and graphics are computed within one application. Even the client control programs are performed in the same application as it is done in EyeSim. Due to the fact that only one active object can be simulated at a time, to split the physics and graphics to a distributed architecture was not necessary. Current PCs can easily handle the computational work load.

A layered based client/server architecture was implemented to reduce code dependencies. The ability to keep the software architecture abstract allows defining clear interfaces between these layers. The server component consists of the program core, the physics and graphics engine and performs the data processing. Two different APIs provide the access for plugins or client programs. The abstract layer model consists of a presentation, control, application and data layer.

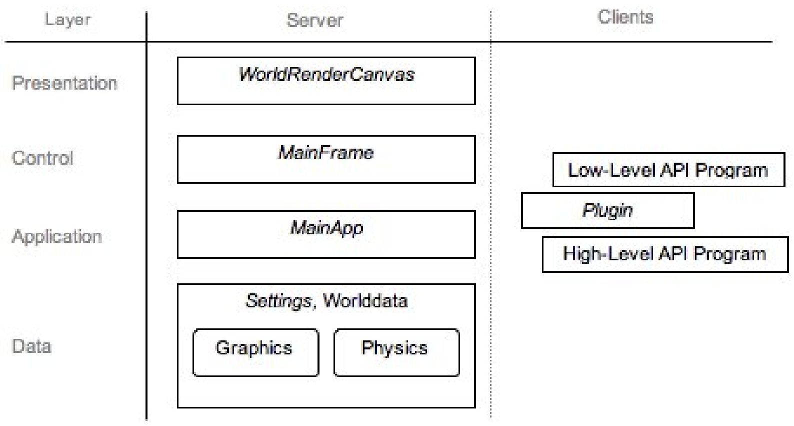
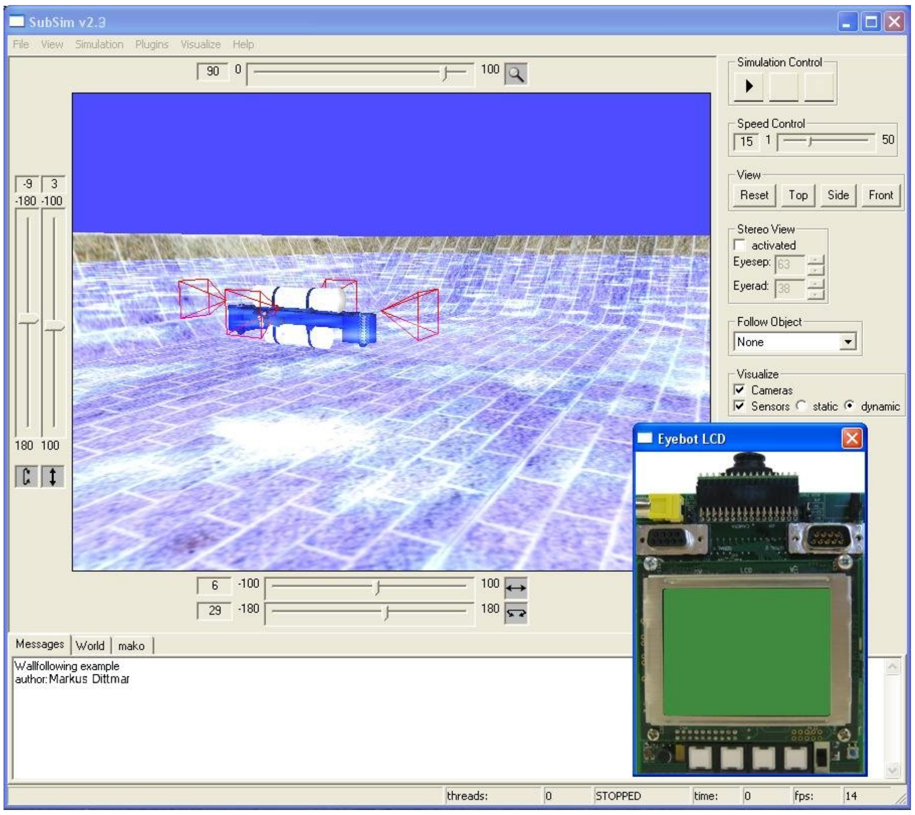


FIGURE 2.9: The layer architecture of *Subsim* [24]

The GUI shown in figure 2.10 covers the presentation and control layer. The presentation layer is implemented by a software canvas which will be explained later in this chapter. It shows the *World* with all its contained world objects. The canvas accepts mouse gestures for changing the visible frustum by translating, rotating and zooming in and out of view.

The control layer is represented by the main frame and offers many functions. However,



Speed Control: 15

Follow Object: None

Visualize: [✓ Cameras] [✓ Sensors] static [✓ dynamic]

Wallfollowing example
author:Markus Dittmar

threads: 0 STOPPED time: 0 fps: 14

FIGURE 2.10: GUI mainframe with its controls and the simulated *EyeBot* LCD of *SubSim*

only the most important functionality will be mentioned. A full description can be found at [24]. It is possible to change the visible frustum with five different slide bars for translating and rotating on the x and the y axes respectively. Zooming is approached by translating forward or backwards along the z-axis (figure 2.11). On the right side are the controls for the simulation. Two buttons are available to start and stop the simulation. A speed panel allows the user to change the simulation speed. Another interesting feature is the possibility to toggle the canvas to a stereo view and this allows a three dimensional view of the scene. The mainframes menu bar contains several entries. The "File" entry allows to load a simulation description file needed to run a simulation or to quit. The Plugins" entry gives the opportunity to place different plugins to extend the simulation. One plugin, the *EyeBot* LCD, is already included and gives developers the chance to see how to develop new plugins. Another important entry is "Visualize" and holds several options in order to visualize sensors and cameras of active objects.

The core of the simulation covers the application layer. It consists of a single object and was thus implemented with a singleton software pattern. Its task is to connect the GUI with the data layer. During the start up of the simulation it initializes the application and the GUI. Comandline parsing allows the loading of a simulation description with the start up of the simulation. The core is also connected to the API plugin component and offers a low and high level API to the function provided by the RoBIOS. These functions are defined in a global static space and makes it impossible to run multiple client programs simultaneously.

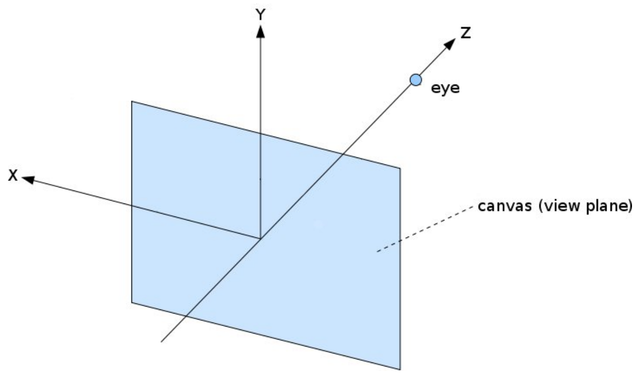


FIGURE 2.11: The right-handed coordinate system and view orientation in *SubSim*

The data layer holds the information loaded from the different extensible markup language (XML) description files. The XML file format was chosen due to its easy readable syntax. The "settings.xml" is loaded at start up of the application and contains the relative paths to other description files, GUI resources and plugin files. A simulation description file is the main configuration file for any simulation and carries a "sub" file extension. This file has to be loaded to start any simulation. Hence each simulation can have only main description file. It describes all objects used by the simulation, including a world description file and a description file for world objects. Furthermore several pre-setups for the simulation can be set e.g. the visualization for sensors, the simulation speed or view settings. A world description file defines the environments, its dimensions and what graphic files in particular are used to render the world. Settings for the water physics and gravity can also be found but are not used at in this version. However, each simulation can use only one world description file. A world object file describes both, the graphical and the physical properties of a world object respectively, which are processed by the physical or graphical engine of *SubSim*. Active objects are distinguished from passive objects by adding an additional XML tag "Submarine" inside a simulation description file.

2.4.4 Physics Engine

To enable realistic simulation behaviour an appropriate physics engine is required. The influence of forces and the liquid effects as they occur in water environments also needed to be considered. In particular sensors have to act like their real counterparts. Like other simulations developed at the robotics lab of CIIPS, it came to the decision to implement the physical abstraction layer (PAL) [25]. PAL provides a unique interface for physical bodies, sensors, actuators and fluids. Its most important property is the unified interface to a number of different physic engines and thus allows the integration of multiple physics engines with ease. This makes it possible to choose an engine that gives the best performance for a certain application and makes it easy to compare them. For *SubSim* two different, free commercial physic engines are applied, the *Newton Games Dynamics* [2] and *PhysX* [3]. However, the way both the physic engines were implemented does not allow switching between them easily as yet. PAL and *SubSim* are interconnected without a clear interface. For each of them the application has to be compiled independently. *SubSim* version 2.3 was published using Newton engine. The interconnection makes it also difficult to update PAL to newer versions and provides additional complexity.

Newton is an integrated solution for real time simulations of physical environments and can be integrated with ease. It assures producing a realistic behaviour with basic physic principles.

PhysX, also known as Novodex, is a real time physics engine developed by Nvidia. It uses the hardware acceleration for compatible graphic- or physic cards and runs in software mode for the others. PhysX offers a greater API and allows real fluid simulation. This feature is currently not implemented in *SubSim*, but it looks very promising in terms of realistic liquid effects and might be regarded for future projects.

2.4.5 Graphics Engine

As in EyeSim, the Milkshape shareware was used to provide 3D polygon models. It allows easy 3D modelling of complex graphics. With the decision to use Milkshape it was ensured that users can add their own object models to use them in simulations. Again OpenGL was used for graphics drawings and allows to render the scene from any viewpoint. With the help of the 3D polygon models the environment and all its contained objects can be visualized. Furthermore, the prerequisite to visualize PhysXs real fluid is given.

2.4.6 Other 3rd Party Libraries

The GUI of *SubSim* was created with wxWidgets [4], an open source C++ frame for writing cross platform GUIs. Besides the common standard widgets, it offers support libraries for multi-threading and a special canvas, wxGLCanvas, for OpenGL drawings. Multi-threading support was used to emulate the *EyeBot* LCD controller so that it supports multi-threading as its real counterpart. The current version of wxWidgets is 2.8.9 but version 2.4.2 is still used for *SubSim* and might be upgraded in the future.

All description files used by the application and simulations are written in the XML syntax. These files are processed through tinyXML, a simple XML parser [5]. It creates a Document Object Model (DOM) that can be easily read within the data layer of *SubSim*.

2.4.7 Sensor Suite

PAL provides the available sensor class suite of *SubSim* version 2.3 and was extended through this project. This section describes the sensor suite of version 2.3, a depth and sonar sensor were implemented within this thesis and are described in chapter 3.

The PAL sensor suite was designed to enable them to couple with an error model. An error model is necessary to allow the user to simulate their equipment to return data as the physical equipment does. Position and orientation of certain sensors can be modelled by a low level physics library. Each sensor instance has to be attached to the body of an active object. Position and orientation vectors inside an active object description file determine accordingly the position and orientation relative to the body of its owning active object. Physical and graphical orientation apply the right-handed coordinating system as shown in figure 2.11.

These settings are read independently from other description files since PAL itself is designed as an independent component. This enables PAL to define a simpler interface, but on the other hand it is more complex since data from the same description file are read in different components inside *SubSim*. In *SubSim* version 2.3 the following sensor classes are available:

- PSD sensor
- Inclinometer
- Gyroscope
- Compass
- Velocity sensor
- Camera

2.5 Feedback Control Theory

Feedback control is one of the most used control mechanisms for mobile robot control applications. Measurements from sensors are compared with set values. The difference between both is given as input to appropriate controllers. Depending on the implementation, a controller produces an output which is applied to the robot to change its status and thus to reduce the discrepancy. A feedback loop is an iterative process. After each iteration, consisting of the calculation of the difference, producing the output value and applying it on the robot, the controller awaits the next measurement.

Two different, simple input simple output (SISO) and very common controller were deployed in this project and implemented in software, one PID- and one FUZZY Logic controller.

2.5.1 PID Controller

A PID controller contains three independent control terms: a proportional, an integral and a derivative term. Process variables (PV) refer to the measurements of sensors, set values are referred as set points (SP) and the difference of both is referred as an error (e). It uses current and recent error values and error changing rates as input parameters [36]. Depending on these input values and on controller gain factors it produces different outputs. This output is applied to *Mako* to change its status. The PID controller is used as a part of a software control program and is processed by the *EyeBot* microcontroller. The controller gains K_p K_i K_d are constant factors set independent for each term and determines the influence of the respective term.

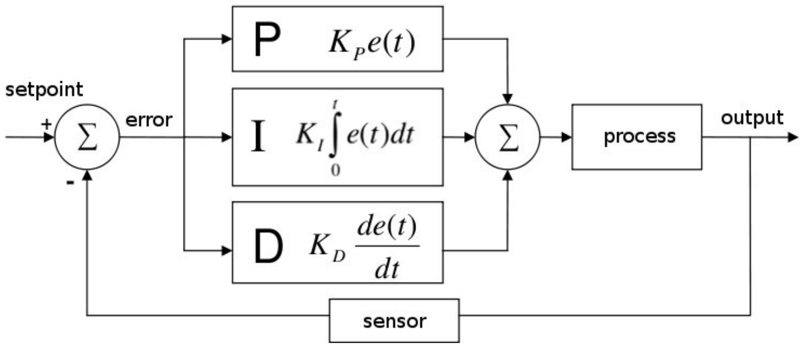


FIGURE 2.12: Block diagram of a PID Controller

Figure 2.12 shows the 3 terms of an PID controller. The proportional part is calculated with

$$P_{out} = K_p e(t) \tag{2.1}$$

Where P_{out} is the proportional term output and behaves proportional to the error e at the time t. The integral output I_{out} is calculated with:

$$I_{out} = K_i \int_0^{\tau} e(\tau)d\tau \tag{2.2}$$

Where τ is the relative time used to accumulate error values from the past. This term is used to eliminate a residual steady state, which can occur by using a controller with a proportional term. Finally the derivative term is calculated with:

$$D_{out} = K_d \frac{de}{dt} \tag{2.3}$$

The derivative term uses the error changing rate by determining the slope of the error over time. This term can be considered as a damping term to prevent the controller from "overshooting".

All terms can be put together into one formula:

$$u_{out} = K_p e(t) + K_i \int_0^\tau e(\tau) d\tau + K_d \frac{de}{dt} \tag{2.4}$$

where u_{out} is defined as controller output.

2.5.2 FUZZY Logic Controller

The human reasoning is imprecise and uncertain. Machines however often reason in binary. FUZZY logic, proposed by Lofy Zade in 1965 [57], enables machines to reason in a fuzzy manner. This is done by setting up heuristic rules in the form of "IF <condition >THEN <consequence>" [53], which associates conclusions with conditions. Strategies are developed from experience rather than from complex mathematical models. This represents the human understanding of describing fuzzy things. A linguistic control implementation is faster accomplished and easier to setup then a PID controller. A disadvantage has to be taken in account due to a computationally intensive processing.

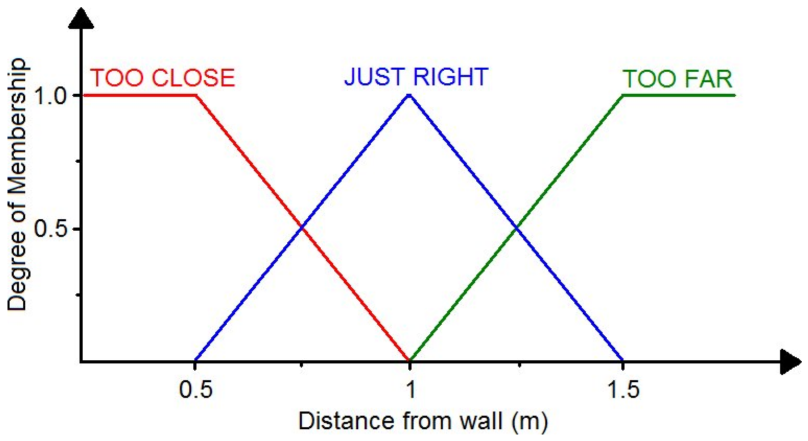


FIGURE 2.13: A fuzzy set for describing the distance to a wall [33]

Fuzzyfication is the process to define input and output fuzzy sets. One set consists of at least one membership function. Figure 2.13 shows a typical input fuzzy set to describe the distance from a wall. It contains three membership functions. The functions are setup by defining their spaces and belonging values. The higher the degree of membership for a function is, the more it belongs to this space. Membership functions should overlap to accomplish smooth transitions of the system. Multiple input and output sets can be defined. Rules describe how input membership functions are mapped to output membership functions. Defuzzification is the process used to map output membership functions back to a single value. For this process different methods are available. Some of them take into account just the element corresponding to the maximum point of the resulting membership function. Andrew de Gruchy implemented in 2008 a fuzzy controller with two input sets for the *Mako* [34] and uses the centroid defuzzification method. This method computes the centroid of the composite area in the fuzzy output set. This controller was readjusted and used within this thesis for several experiments that will be described in chapters 4 and 5.

To follow a wall on the left hand side de Gruchy developed nine rules for the controller:

IF “TOO CLOSE” AND “GETTING CLOSER” THEN “TURN RIGHT”
IF “TOO CLOSE” AND “JUST RIGHT” THEN “TURN RIGHT”
IF “TOO CLOSE” AND “GETTING FURTHER AWAY” THEN “TURN RIGHT”
IF “JUST RIGHT” AND “GETTING CLOSER” THEN “TURN RIGHT”
IF “JUST RIGHT” AND “JUST RIGHT” THEN “CONTINUE STRAIGHT”
IF “JUST RIGHT” AND “GETTING FURTHER AWAY” THEN “TURN LEFT”
IF “TOO FAR” AND “GETTING CLOSER” THEN “TURN LEFT”
IF “TOO FAR” AND “JUST RIGHT” THEN “TURN LEFT”
IF “TOO FAR” AND “GETTING FURTHER AWAY” THEN “TURN LEFT”

Chapter 3

Sensors

The word sensor is derived from the Latin word “Sentiere” and means to sense and to feel. A sensor in technical terms is a device for qualitative or quantitative surveys of chemical or physical properties like temperature, moisture, pressure or sound. The information gained from a sensor is used for further processing, mostly converted to electrical signals [23].

3.1 Sensors for Robotics

At present a vast number of sensors are used for robotics. To choose the right sensor for a given task can therefore be a difficult task [27]. Various sensor systems offer their own unique strengths and weaknesses. Currently there is no system available which suits all needs [42]. The success or failure of a mobile robots operation depends highly on the interactions between the robot and its environment. The quality of the sensors used for a robot also determines the reliability of the robot. Most sensors used in robotics are derived from a biological role model. Their role models are e.g. the human five senses.

There are three important and notable properties to sensors which are listed at [15] and were influences in this thesis.

- the sensor should be sensitive to the measured property
- the sensor should be insensitive to any other property
- the sensor should not influence the measured property

3.2 Sensor Errors and Limits

This section describes typical sensor errors and behaviours. In the following chapter, each sensor of *Mako’s* sensor suite is analysed in detail. National Instruments provides on its homepage a substantial and detailed sensor terminology, which provides the basis for this section. It can be found at [16].

Sensitivity : sensitivity is defined as the input change which produces a noticeable change in the sensors output. The sensitivity can been visualized as the slope of a characteristics curve dy/dx where y is the sensor output and x the sensor input, thus a sensitivity error is the deviation from the actual slope.

Range : a maximum and minimum output value, due to change of input parameters, define a sensors range. The positive or negative ranges are often unequal. Ranges play a major role for distance sensors, where the minimum range is greater than zero.

Precision : ideal sensors produce always the same output value for the same input parameter. Real sensors however produce an output which is distributed relative to its ideal output. Deviations have therefore to be considered for all sensors.

Resolution : the resolution is defined as the incremental change of the sensors output, due to a change of its input parameter. This is somehow similar to the sensitivity, however resolution is mostly a matter for digital sensors or A/D (analog-digital) converted values.

Accuracy : the accuracy of a sensor indicates the maximum possible difference between a sensors actual value and measured output.

Offset : An offset of a sensor indicates the linear difference between real value and the sensors output under certain conditions of their environment. E.g. sensors are often affected by temperature changes, but most effects of environmental conditions can be easily compensated.

Linearity : The linearity of the sensor is often specified in terms of the percentage of non-linearity. The static non-linearity is defined by

$$Nonlinearity(\%) = \frac{D_{in(max)}}{IN_{f.s.}} * 100 \tag{3.1}$$

where $Nonlinearity(\%)$ is the percentage of the non-linearity, $D_{in(max)}$ is the maximum input deviation (accuracy) and $IN_{f.s.}$ is the maximum, full-scale input. It simply describes the non-linear sensitivity of a sensor [15].

Response time : sensors do not respond immediately. The change over a period of time is called response time, and is defined as the time the sensors needs to change from a previous state to a new correct value within a given tolerance band.

Hysteresis : ideal sensors produce an output regardless of the direction of the input change. Some real sensors however show a different offset in different directions due to a delay in the response time. Since *Mako* operates in controlled environments [39], rapid input changes for all sensors are unlikely.

Dynamic linearity : the dynamic linearity of a sensor shows its ability to adapt to follow rapid changes. Amplitude distortion characteristics, phase distortion characteristics, and response time are important to determine dynamic linearity. It is defined by

$$f(x) = ax + bx^2 + cx^3 + ... + k \tag{3.2}$$

where $f(x)$ is the sensors output, the x-terms are its harmonics and k an optional offset. As mentioned before, *Mako* is built to operate in controlled environments and thus dynamic linearity can be disregarded.

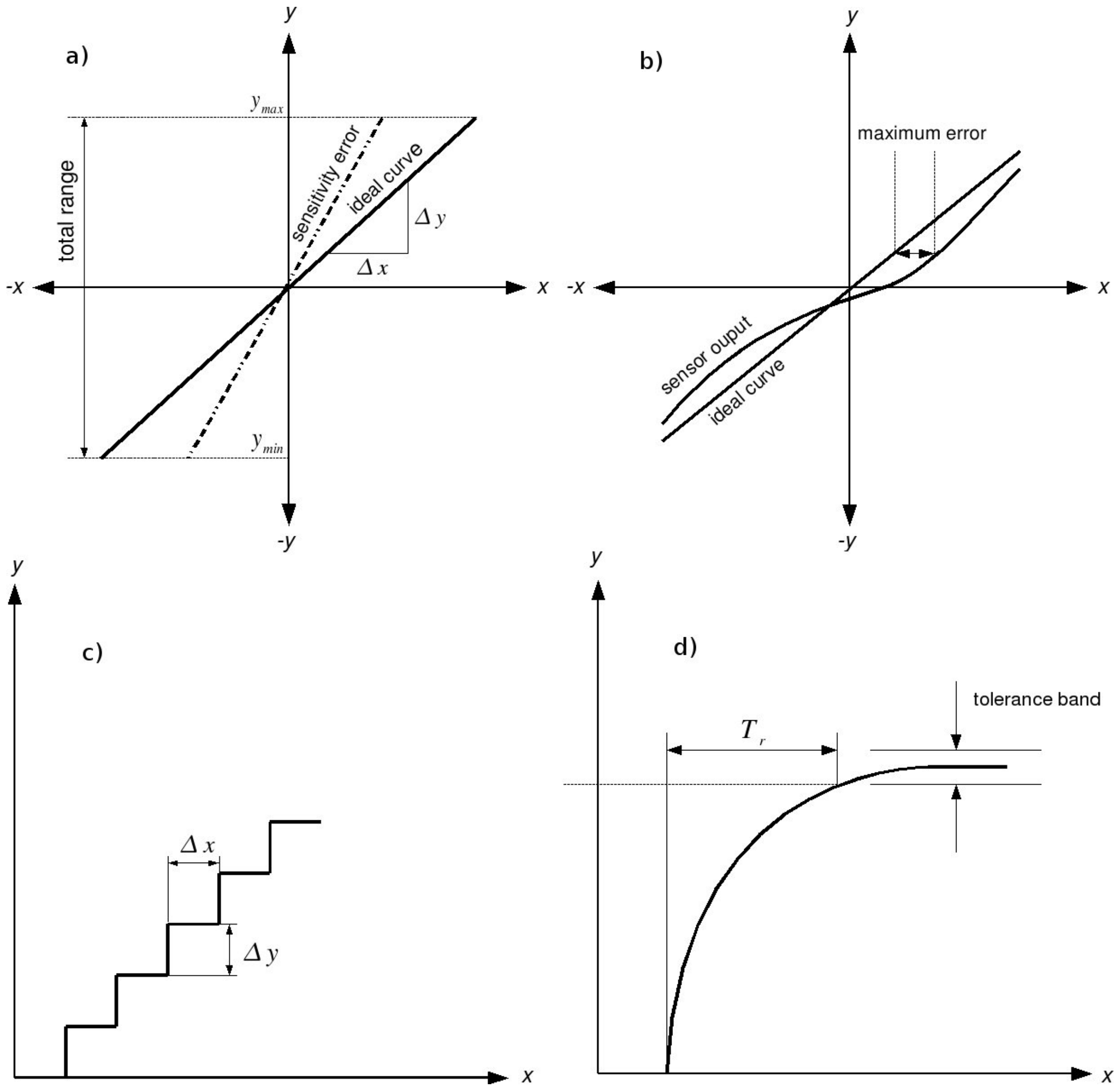


FIGURE 3.1: Example characteristics of sensors where y is the sensor output and x the input parameter : a) shows an ideal curve of a sensor, a respective sensitivity error and the maximum and minimum range y_{max} and y_{min}, b) demonstrates a non-linear behaviour of a sensor, c) visualizes the meaning of resolution, d) T_r shows the response time of a sensor caused by a sudden change in the input parameter.

Some sensors produce an analogue output instead of digital signals. To process them on microcontrollers their values have to be digitized with an A/D converter. Their most important characteristics can be described with their accuracy, depending on the number of bits used to express a value, the speed in terms of the number of conversion per time interval and their input range. These characteristics lead to another source of possible errors sources like quantization or offset errors [17].

3.3 Sensors Mako

After identifying typical characteristics of sensors, each sensor of *Mako's* sensor suite was analysed in detail.

3.3.1 Accelerometer

The low-cost three axis low-g micro machined accelerometer from Free Scale Semiconductor was implemented to measure the acceleration in x, y and z direction [18]. It integrates a low pass filter and a temperature compensation to reduce noises and influences from the environment. The lowest selectable sensitivity with $1.5g$ is set but it is still a high value. *Mako* is never expected to reach that value in operational conditions. The sensitivity is given with $800mV/g$, the maximum output range from $0.45V$ to $2.85V$. Since the *EyeBot's* A/D-converter have an input range of 0 to $5V$ and an output resolution of 10 bit, only 491 different digital values are detectable. The typical effective noise influence is given as $4.7mVrms$.

3.3.2 Sonar System

The sonar system is one of the most important and at the same time the most complex sensor system mounted on *Mako*. Additionally the characteristics of ultrasonic sensors offer their own extensive complexity. An active sonar is necessary to detect passive objects like walls. Transmitters convert electrical signals to acoustical vibrational energy. This energy produces ultra sonic waves which are geometrically spread out and reflected on the surfaces of objects. The returned signal then is then converted back by a receiver into an electrical signal.

There are a couple of reasons which are responsible for energy transmission losses. A finite amount of time is needed to transmit the energy wave through a medium. The intensity decreases over the distance due to the continuous dispersion of the wave. Another reason is scattering and absorption caused by the transmission through a medium [54].

Acoustic signals, especially in enclosed or confined environments, are returned scattered off the surrounding surfaces. This results in the return signal containing multiple components from many different ray paths and thus masking the direct path signal to receiver. These components are called *reverberations*. *Crosstalk* is another noise source which and occurs when different sensors receiving signals from one sender at the same time (figure3.2a).

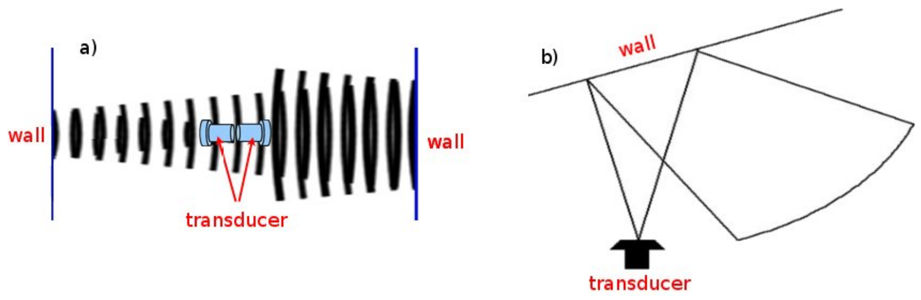


FIGURE 3.2: Figure a) shows a typical of crosstalk situation, b) a sonar sensor out of its beam width can not receive return signals

3.3.2.1 Transducer and Controller

Mako posses four low-cost Navman 2100 Depth sonar transducers and two sonar controllers as a part of its navigation system [19]. The transducers act as transmitters and receivers which are connected via relays to the sonar controllers. One controller is therefore connected with two transducers. These controllers produce the electrical signal which is converted by the transducers to acoustical vibrational energy. The reflected wave again is received by these transducers, and their electrical return signal is sent back to the controllers.

This system was originally developed for depth measurements. Their performance is not suitable to act as a navigation system due to several shortcomings [50]:

- very slow data rate of 1 Hz
- serial data line
- lack of programmability
- poor resolution of 10 cm

Transmitters send signals at their resonance frequency and receivers receive signals at their anti-resonance frequency. Since transducers are used as a dual system, both frequencies have to be as close as possible, but at the same time they have to be different. This causes the transducers to transmit and receive signals on a frequency that is not optimal [19].

Ultra sonic receivers have only a finite beam width in which they can detect an echo return signal (Figure3.2a). The Navman 2100 Depth sonar controller returns only a distance value if the return value of the transducer was valid.

3.3.2.2 Multiplexer

Due to the lack of serial ports of the *EyeBot* controller, the sonar system was implemented with a multiplexer [22]. The multiplexer uses three PICAXE microcontrollers, one micro controller represents the master, the other two are slaves.

The master is responsible for receiving instructions from the *EyeBot* controller and sending sonar data back to it. The communication between the master and the *EyeBot* controller happens via a serial RS232 connection. Therefore a simple protocol was developed and implemented.

Each slave is responsible for one controller and determines, with the activation of the relays, which transducer is currently connected to a controller. At the same time only one controller can communicate with one transducer. The sonar controllers digitise the returned signal from the transducers and then send it to their connected slaves. The data is then available for requests from the master. For the communications between the slaves and the master a kind of the *NMEA 0183*[1] protocol was implemented. Is specifies the communication between marine electronic devices such as echo sounder and enables unidirectional conversation with multiple listeners.

A couple of issues on the multiplexer could be solved within this thesis. The input pins for reprogramming the PICAXE controller where not grounded, thus floating and forcing the controllers to reset regularly. It had a major effect on the response time which could be reduced from an average value of 2.8 seconds to 1.5 seconds [33]. Further the protocol to change between two transducers was inadequately implemented. A simple commitment, sent by the master to the *EyeBot*, ensures now correct switching on the multiplexer. The time to switch the relays could be also reduced by an average of 75%.

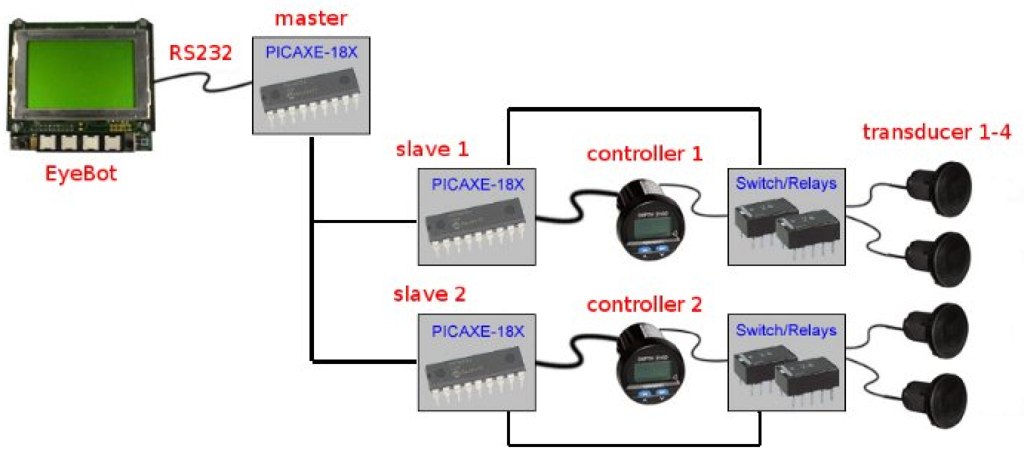


FIGURE 3.3: The sonar system of *Mako*

[1]http://www.nmea.org/pub/0183/

3.3.3 Velocity Sensor

The Navman Speed 2100 velocimeter shall provide the surge velocity of *Mako*. It determines the velocity of the vehicle by simply counting the number of half rotations of its paddle. The sensors poor resolution of $5cm/s$ is unsuitable for navigating purposes. With this sensor no experiments with sufficient results could be accomplished in the past.

The output of the velocity sensor is connected to a $5V$ A/D-converter of the *EyeBot* controller. At the beginning of this thesis, the velocimeter was connected to a $12V$ power supply which respectively was the output voltage. This caused a significant distortion on all A/D-converters of the *EyeBot*. The connected A/D converter is polled every 10ms and is used to count the number of encoder ticks from velocity sensor, every second then the velocity is calculated.

3.3.4 Digital Compass

A low-cost Vector 2X digital magnetic compass provides the yaw for heading information. This sensor has an accuracy of $2\,^\circ$ and a resolution of $1\,^\circ$ [20]. It has proven as the most reliable among all sensors used within this thesis, however, since it measures the magnetic field in a single plane, it is strongly influenced by tilts as can be seen in chapter 4.

3.3.5 Depth Senor

A depth sensor system, using the SenSym SX15GD2 pressure sensor, was introduced by Elliot Alfirevich in 2005 for on-board usage [22] to measure the absolute depth of *Mako*. It was reported with an operating range of 0 to $5m$. It outputs an analogue value from 0 to $5V$ which is digitised by one of *Mako's* A/D-converter, thus using the full accuracy of 10 bits. The application note assures sufficient characteristics for its arranged operations [21]. Typical values are:

- 0.1% hysteresis of its full scale range
- sensitivity of 0.1mV
- response time of 0.1ms

At the beginning of this thesis, the system was completely destroyed due to a faulty seal and had to be replaced and recalibrated.

3.4 Sensors in SubSim

The physical model of *Mako* in *SubSim* is realized by a simple box with the dimensions $x * y * z$, a *PAL Body Box* [25]. PAL itself supports connected bodies to allow different shapes presented by multiple bodies. This was not implemented in *SubSim* and has an influence to the physics as it can be seen in chapter 5.

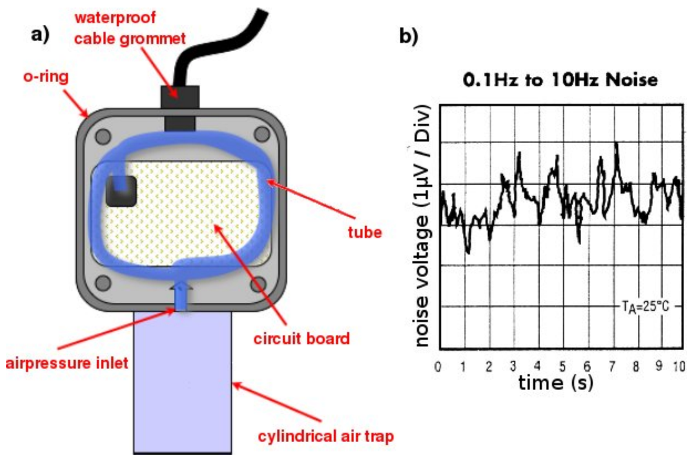


FIGURE 3.4: a) depth sensor system schematics [22], b) typical noise of the applied SenSym SX15GD2 pressure sensor [21]

PAL sensors have to be always attached to a parent body like a PAL Body Box. Their initial position and orientation are set in a relative term to their parent's body if appropriate. This is done with 3D vectors, one vector indicates the orientation, the other indicates the position.

Physic engines like *Newton Game Dynamics* allows to choose custom defined units. For *SubSim* the basic units are “meter” for length and distances, “seconds” for the time, “radians” for angles, “kg” for weight and “Newton” for forces are used. Respectively is the output of the PAL sensors defined with these units, which are accessible through the low-level API. However in *SubSim* access to the sensors is also possible on a *high level* through the emulated *EyeBot* controller plugin, and thus different output units are possible. Adding new sensor classes to the application and attaching them on a submarine model makes it necessary to update the HDT respectively.

PSD sensor : a positional sensitive device measures the distance from its body to the nearest surface which blocks the line of sight. In *SubSim* a single ray is projected from the sensors position into the sensors direction until it reaches an object, thus determining the intersection point. The return value is the length of this ray. The high-level distance is returned in millimetres.

Inclinometer : an inclinometer returns the angle which depends on the orientation of its body. The return value is the angular difference of its current and the initial orientation. The body of inclinometers are always attached to the dead centre of its parent's body, thus an orientation vector is not present. The orientation vector indicates the initial orientation which indicates implicitly the reference plane.

Gyroscope : this class of sensor is very similar to the inclinometer class. The return value is the angular velocity, calculated with the comparison of angular orientations in constant time intervals. A sensor instance needs also one vector to indicate its reference plane and is attached to the dead centre of its parent's body.

Compass : a *SubSim* compass sensor is identical to an inclinometer with its reference plane along the x- and z-axis. It is located at the dead centre of its parent's body and the high-level return value is between 0 and 359 degrees.

Velocity sensor : the return value of this sensor in version 2.3 was the absolute velocity of it's attached body. Within this thesis the possibility to influence objects with a steady flow was implemented inside the simulation. Thus the sensor was adjusted to return the velocity relative to the water.

Sonar : in version 2.3 of *SubSim* a sonar was not available. Since sonar sensors are a very common sensor class for submarines and one of *Mako's* most important sensor system, the implementation of a sonar sensor class was an important part of this thesis. To simulate sonar systems is a big challenge in itself. Due to the fact that *Mako's* operation environments are of simple shapes like rectangles or ovals (appendix A), a simplified sensor was implemented. This sensor class inherits the functionality of the PAL PSD sensor class and extends it with additional attributes.

Depth sensor : a depth sensor class was implemented within this thesis and gives the opportunity to measure the absolute distance to the virtual water surface.

Camera : camera instances simulate a robots camera to receive images from the robots environment. The camera class extends the functionality of the synthetic camera, which is used to visualize the environment on the applications canvas. Two vectors indicate their position and orientation relative to the attached body.

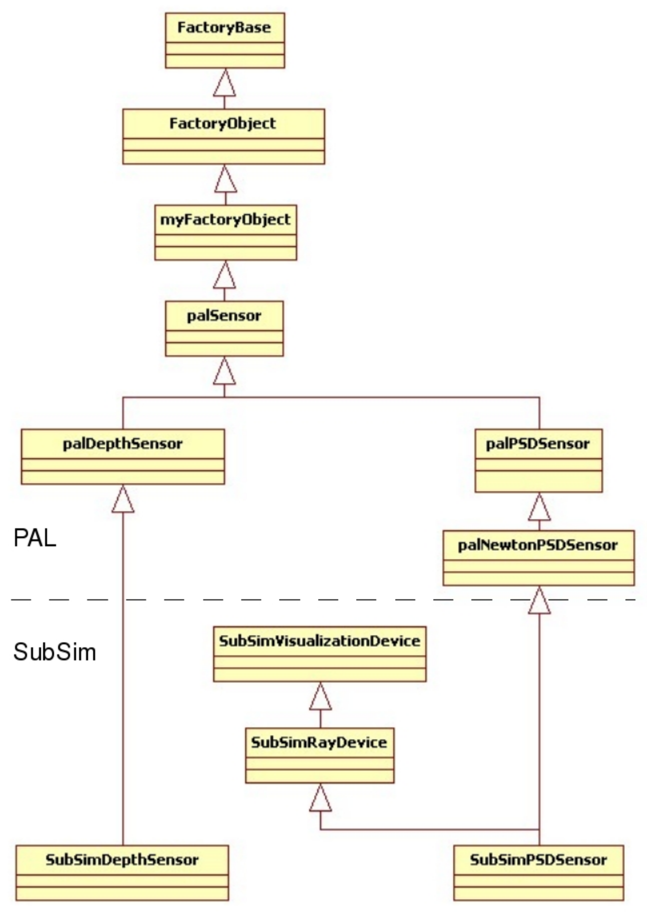


FIGURE 3.5: Inheritance class diagram of the *SubSim* sensors. One device class is implemented with up to 6 level of inheritance

Chapter 4

System Identification Mako

After studying the characteristics of sensors, several static and dynamics experiments with *Mako* and it's sensor suite were processed. These experiments have been necessary to identify *Mako* under operational conditions. The outcomes of the experiments were not only beneficial for the model adaption of *SubSim*, weaknesses and short comings could be highlighted, test procedures will be discussed and the results can be used for future projects.

4.1 Depth Sensor

Elliot Alfirevich conducted experiments with the depth sensor as part of his final year project [22]. His methods and results have been found sufficient for the purposes of this thesis.

The first experiment was processed to indicate the influence of electromagnetic interference (EMI) and supply voltage noise produced by the pulse width modulation from the motor drives. Fast on and off switching of the motors leads to a sudden increase of the electric current on the power supply. Since depth sensor and motors are connected to the same power supply, this has a strong effect on the accuracy of the depth sensor. Figure 4.1 shows the effect of different PWM settings of the stern motor. The stern motor is the closest to the depth sensor, thus the strongest EMI effects have been observed using this motor. A simple low-pass filter however could reduce the effect of both noise sources significantly.

The results of the second experiment (figure 4.1) demonstrates the output linearity of the sensor. As shown in chapter chapter 3 the sensor shows a good linearity. At each measurement depth four sensor readings were taken.

4.2 Compass

The digital compass has been found to be the most reliable sensor analyzed within this thesis. The accuracy of 2° given in the data sheet could not be verified. It indicates

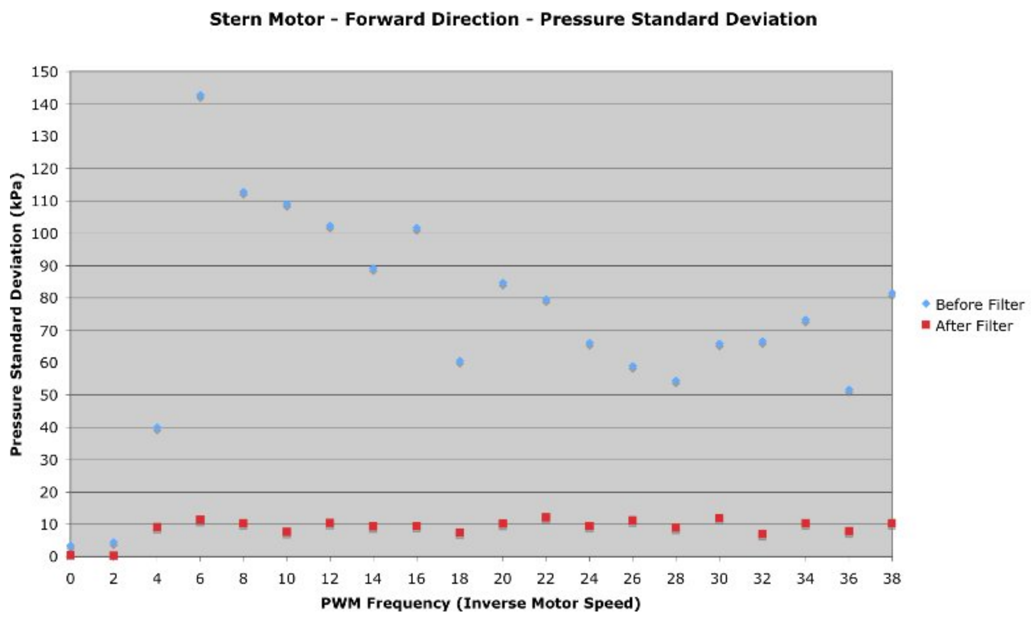


FIGURE 4.1: Influence of the PWM of the stern motor on the depth sensor [22]

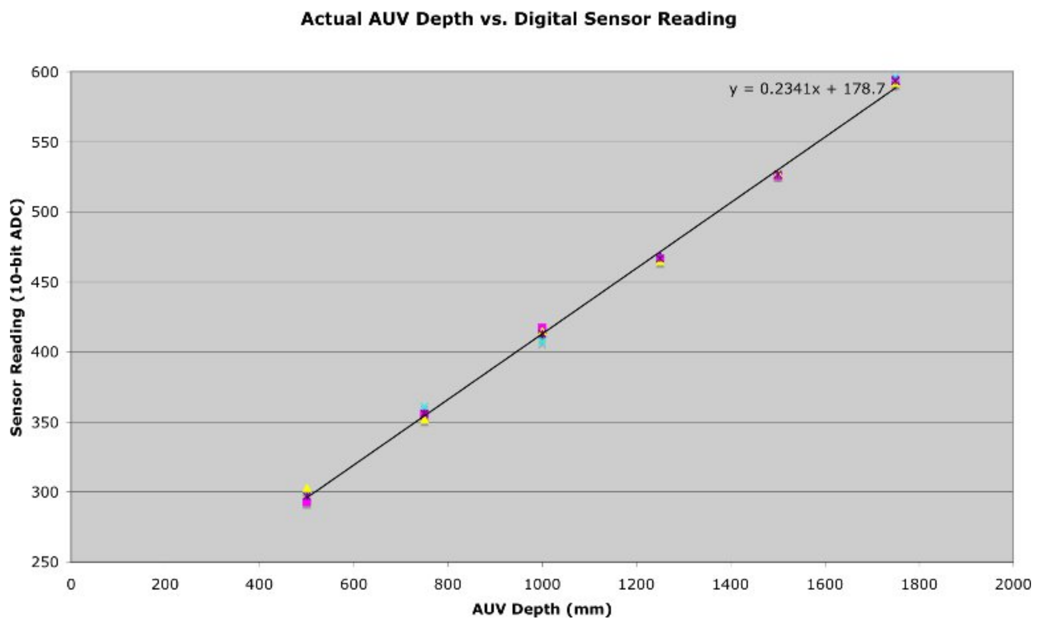


FIGURE 4.2: Actual depth against depth sensor reading [22]

the precision of this sensor rather than its accuracy. A reason might be that the sensor is very close attached on the aluminium tray what influences the sensors environment. Furthermore, this sensor has shown a strong reaction to tilts. Compass errors due tilt measurements can be quite significant [41] and depending on the geographical location and inclination of the compass measurements can lead to different results.

To examine the effect of a tilt of 5 ° an accurate experiment was designed and executed. A tilt of 5 ° represents an usual operational condition of *Mako*, since it is influenced by wind and waves when it is not submerged. When *Mako* accelerates it experiences a jiggling behaviour as will be shown within this chapter. Under submerged conditions a slope might occur due to non-uniform thrusts of the stern and bow motor.

Before the experiment the compass had to be recalibrated due to a high output non-linearity. The RoBIOS provides the function *COMPASSCalibrate(int mode)* to calibrate the sensor. This function has to be called twice. The second call has be done after the sensor is turned around 180 °.

The hole tray of *Mako's* upper hull was clamped in an adjustable bench vise to set up a tilt of 5 °. The bench vise itself was mounted on a turn table, thus it was possible to turn the tray about 360 ° precise around the perpendicular axis.

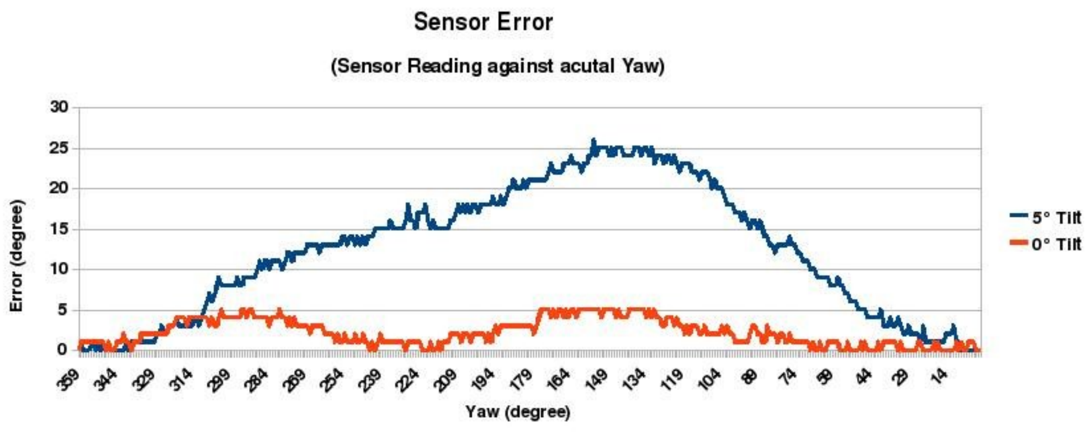


FIGURE 4.3: Absolute heading error of the digital compass with a tilt of 5 ° and without

4.3 Motors

During experimentation it could be observed that *Mako* tends to turn right when setting the same thrust on port and starboard actuator. Gonzales [22] processed thrust measurements with a load cell on all actuators. All four motors produced different thrusts when setting them to the same PWM. Gonzaless solution was to implement a look-up table in the motor controller to balance the different forces on the actuators. Thus it was possible to set each motor to a similar thrust with ease.

Since the motors are inexpensive trolling motors and especially with the horizontal motors being used by many experiments from 2004, a considerable amount of wear was assumed. Due to the fact the motors are fixed to *Mako's* body thrust tests with a load cell could not be repeated. Instead an optical tachometer was used to measure the revolutions of the propellers. Each propeller consists of three propeller blades, so that the results of the tachometer had to be divided by three.

The tachometer provided its output on a LCD display. The output was updated 1.5 times per second. To capture the readings contiguous, movies of the display were recorded which could be played in slow motion afterwards. The readings were processed with a step size of 3% of the maximum pulse width beginning with 0% (full speed due to the inverted logic) until the motor stopped. For each motor and each pulse width modulation, 30 contiguous readings were captured and the arithmetic average was calculated. The results of the bow motor are shown in figure 4.4 and 4.5 in relation to the thrust measurements from 2004. A linear dependency between revolution per minutes (RPM) and produced thrust can be seen. However, the experiment shows that the motors still rotate with higher pulse width modulation than in the experiments done in 2004. One reason is the missing water resistance since the revolution experiment was processed with the actuators in air. Another reason because of wear the motors are run in and experience a lower resistance.

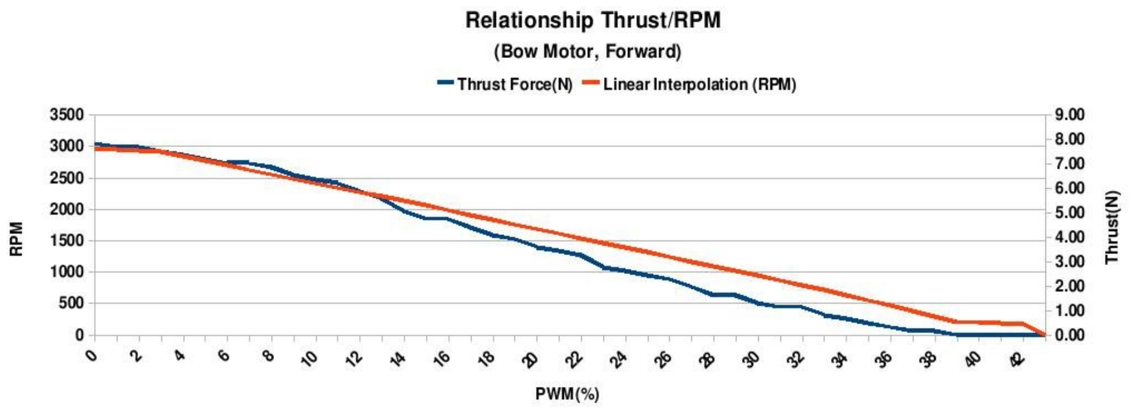


FIGURE 4.4: Motor revolution versus actuator thrust force, bow motor forward direction

As a result the look-up table was respectively updated. The results of the other motors can be found in appendix C.

Figure 4.6 also indicates how the speed of the motors fluctuates. The motors are only controlled by the PWM without any feedback to control the actual speed. Higher speeds have a higher fluctuation.

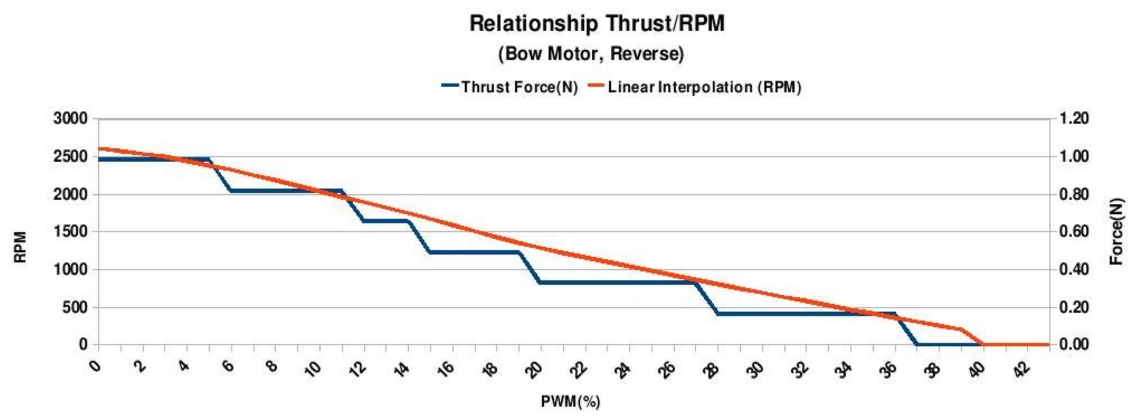


FIGURE 4.5: Motor revolution versus actuator thrust force, bow motor reverse direction. A much lover thrust can be observed due to motor and propeller optimizations to operate in forward direction

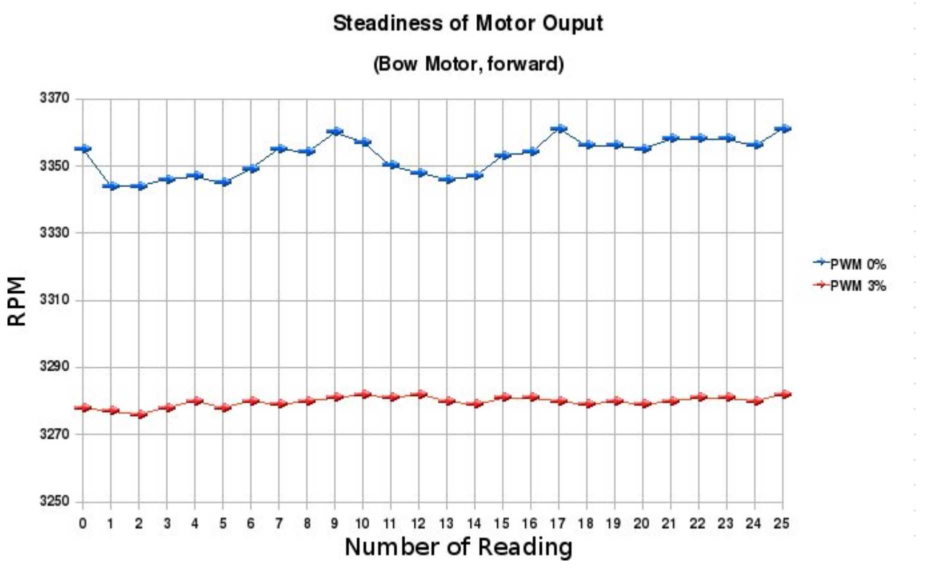
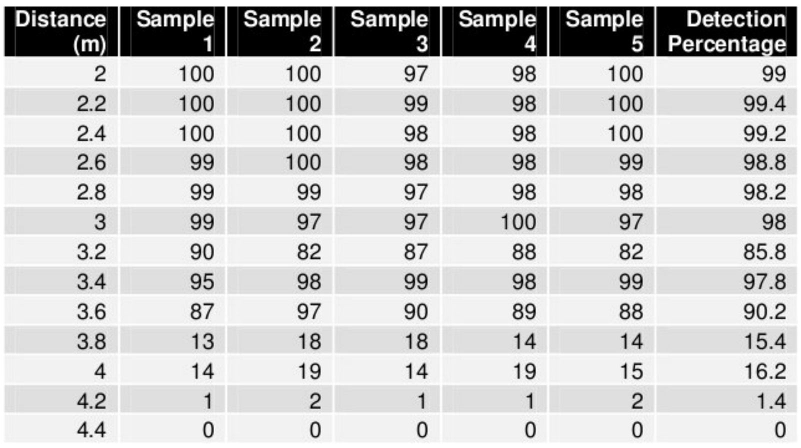


FIGURE 4.6: Fluctuation of the bow motor in forward direction

4.4 Sonar

The sonar system works well only in a water environment, and large space is necessary to avoid disturbing reverberations. This makes it necessary to process experiments with the sonar in a environment such as a pool. The shortcomings mentioned by Nguyen in his final thesis were replicated within this thesis [50]. Many experiments by Nguyen and Alfirevich were processed to analyze the sonar system and especially the sonar transducer without the sonar controller. Since the the transducers are fixed mounted on *Mako*, and no seperate transducer has been available, these experiments could not repeated within this thesis. However, more roughly static and dynamic experiments were processed with the whole *Mako* involved, and the behaviour could be verified which was observed by previous experiments. For transducer experiments both Alfirevich and Nguyen, used a digitally output from the *EyeBot* microcontroller to create the input signal for the transducer. The returning signal was again received by the microcontroller, thus replacing the function of the sonar controller to retrieve transparent results from the transducer. This was quite helpful since the sonar controller provides data with a rate of only 1Hz. One measured distance value from the controller is obtained from multiple samples which the controller generates.

The results shown in figure 4.7 show the detection rate of samples in a static situation. Individual signals were sent out orthogonal to a wall and reflected on its surface. The required detection rate was defined with a minimum of 50% to be sufficient for navigating purposes.



Distance (m)	Sample 1	Sample 2	Sample 3	Sample 4	Sample 5	Detection Percentage
2	100	100	97	98	100	99
2.2	100	100	99	98	100	99.4
2.4	100	100	98	98	100	99.2
2.6	99	100	98	98	99	98.8
2.8	99	99	97	98	98	98.2
3	99	97	97	100	97	98
3.2	90	82	87	88	82	85.8
3.4	95	98	99	98	99	97.8
3.6	87	97	90	89	88	90.2
3.8	13	18	18	14	14	15.4
4	14	19	14	19	15	16.2
4.2	1	2	1	1	2	1.4
4.4	0	0	0	0	0	0

FIGURE 4.7: Static detection rate for different distances [50]

As experienced within this thesis, the detection rate decreases with a deviation from the surface normal until the transducer leaves the beam width of the signal (Figure 3.2b).

Figure 4.8 shows the result of a dynamic experiment. The transducer was directed to a wooden rod. This rod was moved then several times perpendicular to the transducer in a distance from 0.5 meters to 2 meters. The effect shows in a lower detection rate. One reason for this is the produced distortion of the water which comes along with environmental noise. Another reason might be the occurring Doppler Effect and it was suggested by Nguyen to operate *Mako* with very slow speed.

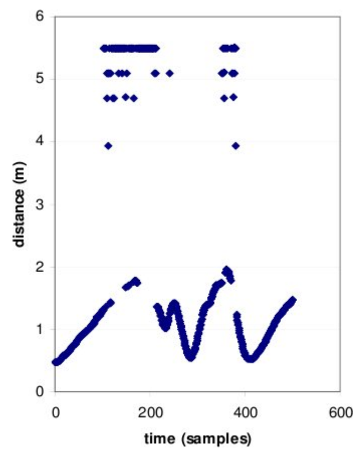


FIGURE 4.8: Measuring distances from a moving wooden rod [50]

An experiment processed by Alfirevich gave the possibility to estimate the beam width of the transducers (figure 4.9). The transducer was turned around a fixed point at one side of a pool to send and receive signals from a greater area. In the middle of that pool an obstacle was placed. The result indicates that the transducer had to pass the obstacle of around 5 ° before it measures the pool wall rather than the obstacle. Thus a beam width of 10 ° was estimated. Experiments done within this thesis commit to this estimation.

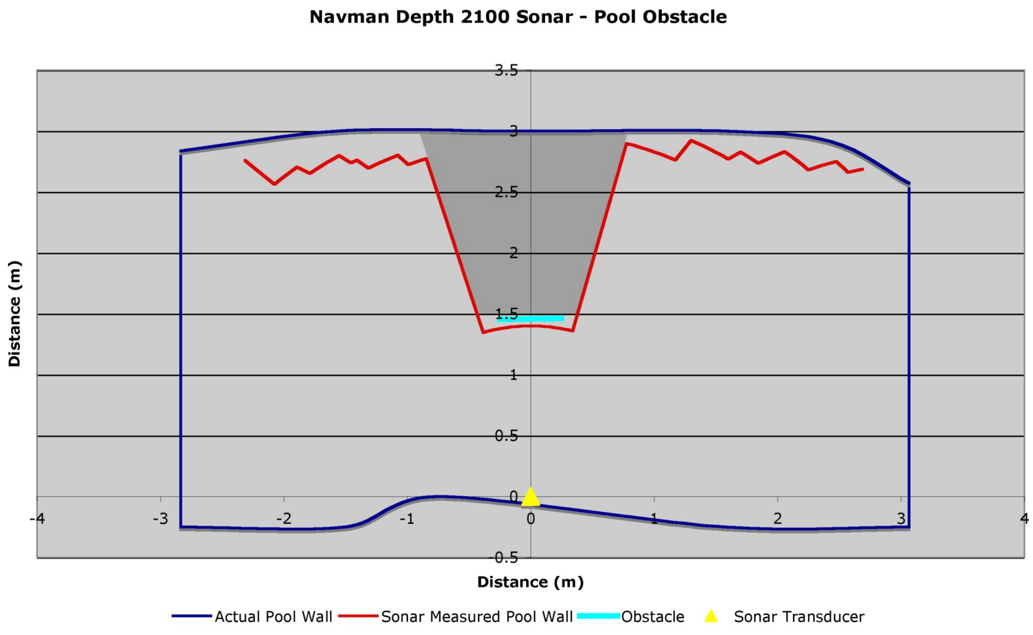


FIGURE 4.9: Distance measurement at different angles [22]

4.5 Velocity Measurement

To compare the surge behaviour of *Mako* with the model in *SubSim* a couple of experiments were processed to obtain *Mako's* maximum velocity and acceleration. In these experiments a heading control using the digital compass was applied to have *Mako* travelling straight as possible. A control was necessary due to the fact that setting the same thrust on port and starboard motor, even with the updated look-up table, does not mean that *Mako* will travel in a straight line. With the heading control *Mako* is able to drive almost on an ideal straight line.

All experiments were processed without submerging thus *Mako* travels like a ship. Because of time constraints and a destroyed depth sensor at the beginning of this thesis, not all targeted experiments could be completed. A picture in appendix E shows that, even when *Mako* is in a non submerged state, it is almost completely underwater and thus only small differences to submerged experiments are expected.

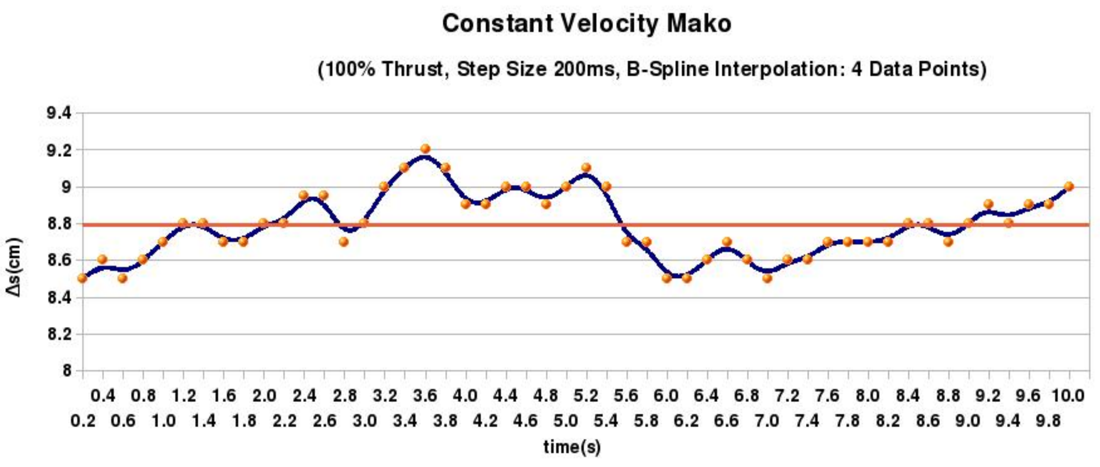


Figure 4.10: Maximum velocity of *Mako* at 100% thrust

Since the accelerometer does not provide a sufficient sensitivity, and the velocity sensor has a low resolution, a different method had to be found to obtain the wanted data. A water tight camera was mounted on top of the camera facing to the port side of the submarine. At the inside of a pool wall a 10 meter long measuring tape with large scale markings was attached. Movie records with a resolution of 30 frames per second were taken from this measure tape. This gave the possibility to step through movie afterwards frame by frame to measure the progress of *Mako* in forward direction in respect to the time.

The maximum velocity of *Mako* was determined by setting both horizontal motors to 100% thrust. For the acceleration experiment *Mako* was accelerated from a stationary position and suddenly accelerated by setting port and starboard motor to 100% thrust. The deceleration data were gained by switching off both motors while running *Mako* at full speed.

Through the recorded movies the progress in respect to the measure tape was determined each 200 milliseconds. The result is shown in the figures 4.10, 4.11 and 4.12. An average maximum velocity of $5 * \frac{8.8cm}{200ms} = 44cm/s$ has been determined. The fluctuation of the values shown in figure 4.10 can be explained by the jiggling *Mako* experiences in port/starboard direction while driving straight. *Mako* reaches its maximum velocity after 6 seconds and returns after 32 seconds of drifting from the full velocity to a stationary position. Figure 4.11 shows also the result of the velocity sensor. The sensors output was divided by 5 to adjust it to the determined velocity with the camera. The outputs gradual rise is an indication of the low resolution of the sensor. With higher velocities the sensors accuracy decreases.

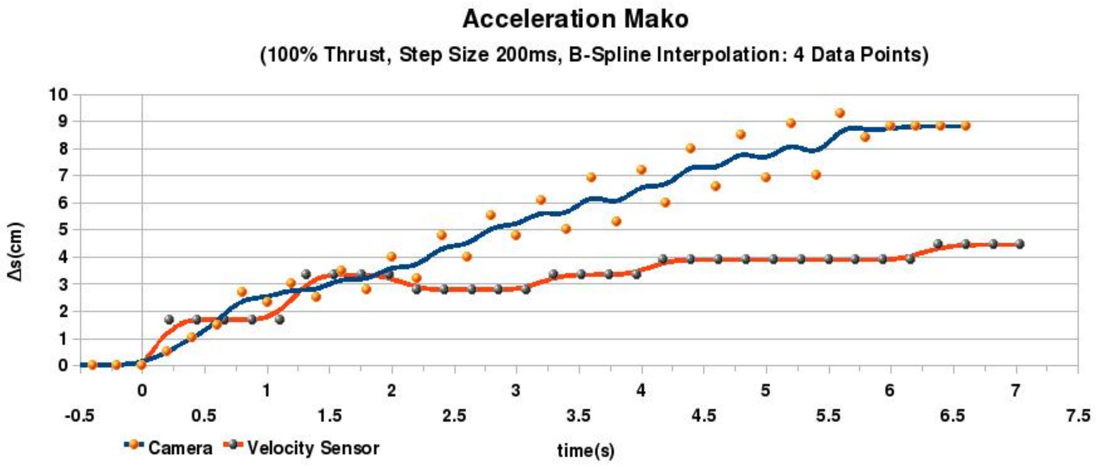


FIGURE 4.11: Determined velocity at full acceleration from a stationary position

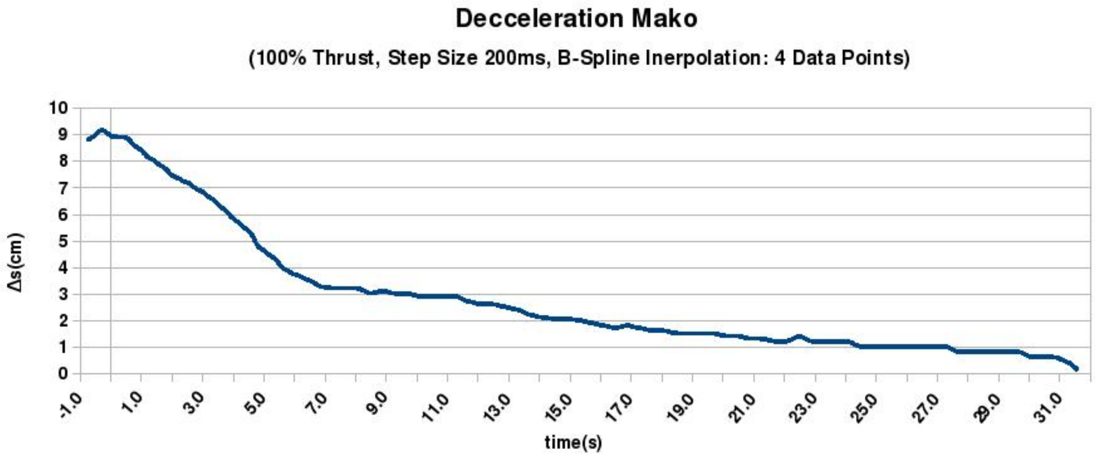


FIGURE 4.12: Drifting from maximum velocity to a stationary position

4.6 Acceleration

The velocity measurements also allowed the determination of *Mako's* maximum and minimum acceleration. Reading the maximum progress from figure 4.11 implicates the highest acceleration is $9.5cm/s^2 \equiv 9.7mg$. Due to the jiggling behaviour of *Mako* this value can be regarded as an upper bound. The data obtained during this experiment from the accelerometer did not indicate any detectable acceleration. This is because of a too low sensitivity of the sensor. The sensitivity in the data sheet [18] is given with $800mV/g$. The maximum detectable difference in the output voltage is thus

$$\begin{aligned}
\Delta V_{max} &= resolution * acceleration && (4.1)\\
\Delta V_{max} &= 800\frac{mV}{g} * 0.0097g && (4.2)\\
\Delta V_{max} &= 7.76mV && (4.3)
\end{aligned}$$

The A/D converter of the *EyeBot* microcontroller has a resolution of $5V/10bit = 4.883mV$. Assumed the A/D converter has an offset of $0.5LSB$ (least significant bit) the maximal detectable digital difference will be $1.58LSB$, since

$$7.76mV - 1.5 * 4.883mV = 0.44mV < 4.883mV/2 \tag{4.4}$$

Considering that the sensors typical noise influence is given as $4.7mVrms$ [18] it becomes obvious that almost no acceleration of *Mako* is detectable by this sensor. This means the sensor unusable for navigation purposes.

The method to gain the maximum acceleration values for the sensor applied by de Gauchy [34], by simply shaking the sensor with the hands in different directions, does not lead to reasonable results. The results were not repeatable since the applied force is unknown.

To calibrate the sensor it's axis of sensitivity has to be turned into the direction of gravity to gain the acceleration of $1g$ and reversed to gain the acceleration of $-1g$ where g is the acceleration of gravity. The output function of the sensor follows the sinusoid, thus having the highest sensitivity at 0g and the lowest at $\pm 1g$.

Figure 4.13 shows the calculated output line of the sensor at $Vdd = 2.6V$. To verify the sensors sensitivity it was turned $\pm 90\,^\circ$, $\pm 45\,^\circ$ and $0\,^\circ$ to the surface normal for each axis and 50 samples from the output of the A/D converter in each position were recorded. The results of the y- and z-axis can be found in appendix D.

4.7 Spot Turn

An experiment was processed to determine how fast *Mako* can turn $360\,^\circ$ around the yaw direction. Therefore either port motor was set to 100% forward thrust and the starboard

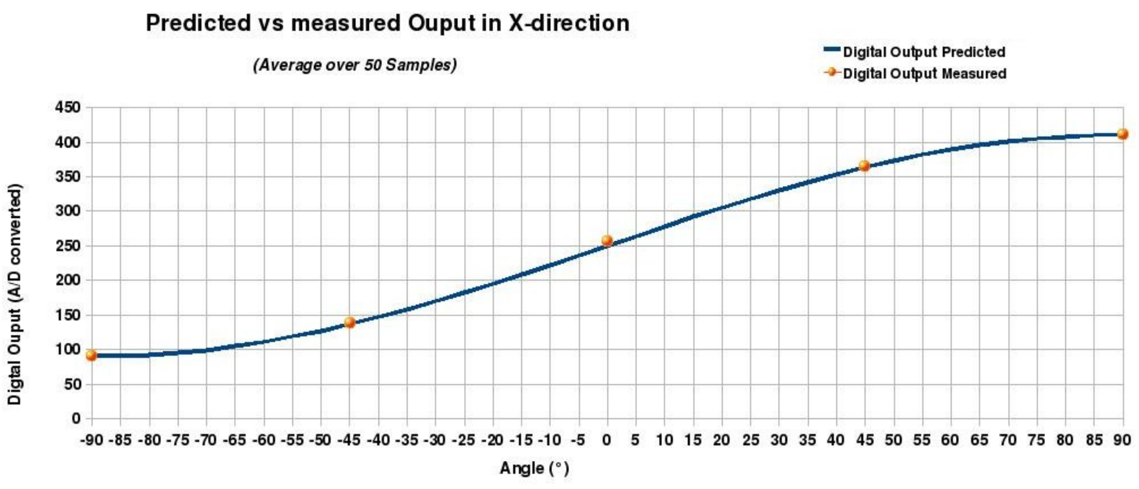


FIGURE 4.13: Angular response of the accelerometer

Axis	-90 °(LSB)	-45 °(LSB)	+0 °(LSB)	+45 °(LSB)	+90 °(LSB)
x	1.31	1.5	1.34	1.24	1.2
y	1.2	1.32	1.31	1.26	1.14
z	1.17	1.19	1.25	1.07	1.08

TABLE 4.1: Measured standard deviation of the accelerometer over 50 samples

motor to 100% backward thrust or vice versa. This did not lead to a spot turn due to the fact *Mako* traveled in a circle as shown in figure 4.14. Forward and backward thrust of the motors are quite different as it can be seen in figure 4.4 and 4.5. To determine the way *Mako* traveled a movie of the scene was recorded. Therefore a camera was mounted on a 4.5m high stand directly over *Mako*. For an anti-clockwise turn *Mako* needed 23.17 seconds, for a clockwise turn 23.5 seconds were measured. A picture of the experiment setup can be found in appendix E.

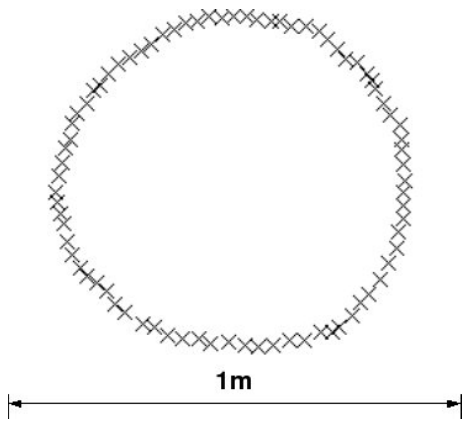


FIGURE 4.14: Travel way of *Mako* in anti-clockwise direction with 100%thrust forward direction on the starboard motor and backward direction on the port motor

4.8 Wall Following

The wall following problem is the first competition task from the project *Mako* initiative. It has been used to evaluate the sonar system with FUZZY and PID controllers by Alfirevich, Daws and de Gauchy during their final year projects. However the results of their experiments were not always repeatable and therefore not reliable as already mentioned by Daws [33]. The problem is the shortcoming of the sonar system. The data rate is too slow, the output data has a too high resolution, and with a moving *Mako* a certain degree of rejection rate is unavoidable. An additional problem occurs when *Mako* tries to correct the distance and thus changing the angle of the sensor in respect to the wall. The greater the angle between the transducer normal and the the normal from the wall the greater is the measured distance from the sonar as seen in figure 4.15a). The feedback of the sonar used as input value for the FUZZY or PID controller lead to a even stronger correction.

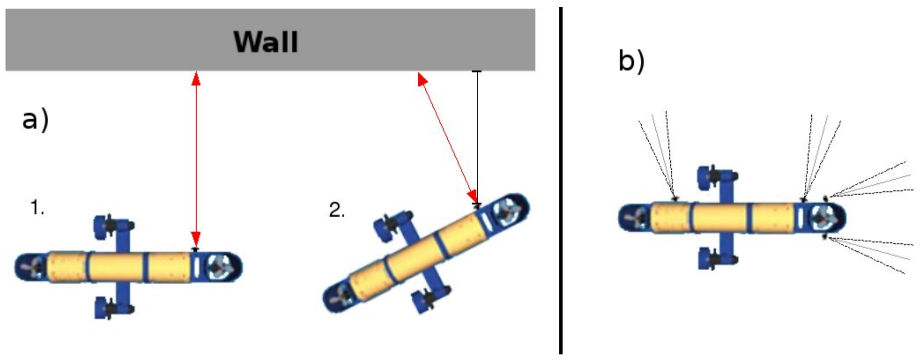


FIGURE 4.15: a) The turning transducer problem, b) the new transducer setup

A new transducer setup was chosen to reduce this effect and to increase the effective beam width of the transducer. Two transducer on the port side and two on the bow were mounted, each with an offset of 15° of their respective axis. The setup can be seen in figure 4.15b). Additionally to avoid *Mako* turning out of the beam width, the control program applied for this task used the information from the digital compass to narrow the possible turn angle. Crosstalking is avoided with the help of the multiplexer. Only one transducer from a pair of two transducer at each side is active at the same time.

Since it is unknown from which transducer the returned value from the multiplexer is three readings were taken to ensure readings from two different transducers. The first reading from the transducer 1, the second from transducer 2 and the third again from transducer 1 for each pair. The shortest distance of all three received values was taken as the controller input. This however lead to a significant time overhead. The total time for three readings was determined with 4.8 seconds in average and forced a very slow operation speed for *Mako*.

Regardless the new setup has proven more robust and with the help of the digital compass *Mako* passed for the first time a rectangular corner autonomously thus accomplishing the first mission task of the project *Mako* initiative.

The result of a FUZZY wall following experiment is shown in figure 4.16. *Mako* tried to keep a distance of $10dm$ of its port side. The average thrust on port and starboard motor was set to 10%. The effective distance is the smallest distance determined by the two transducers on the port side. Velocity measurements with the velocity sensor did not lead to reasonable results due to the low resolution of the sensor and the low operating velocity applied at this experiment. A successful run with a PID controller could not be accomplished due to time constraints.

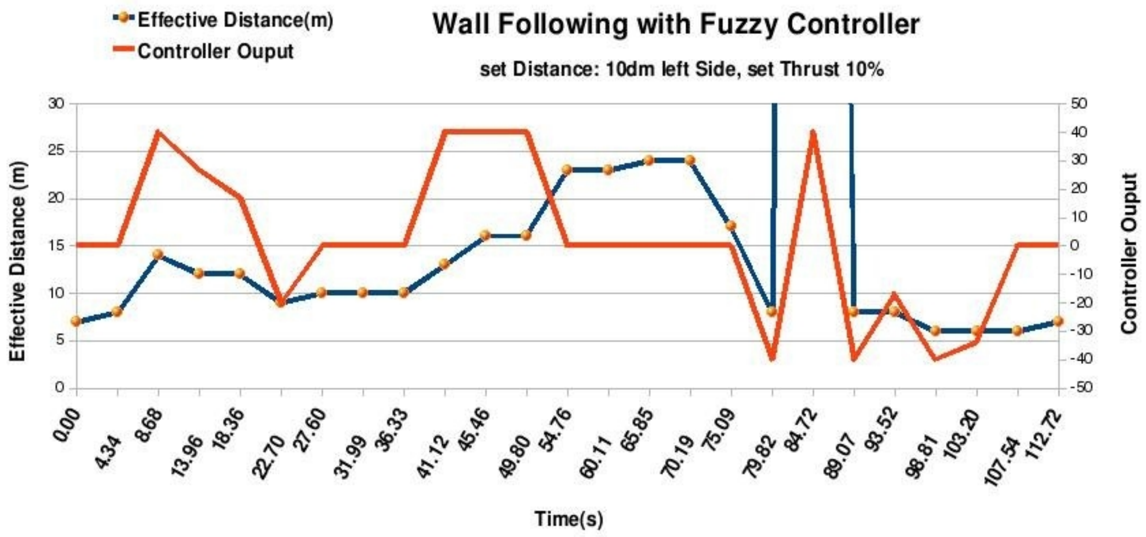


FIGURE 4.16: The result of a FUZZY wall following experiment

Chapter 5

Simulation Model Adaption

The experiments done in chapter 4 were necessary for the model adaption to fill the reality gap in *SubSim*. With the observations and results of the experiments a detailed error model could be implemented within this thesis. *SubSim* reached its version status 2.4. To evaluate the changes on *SubSim* the same experiments designed and processed for *Mako* were repeated in *SubSim*, using the same control programs. However, the RoBIOS API does not provide access to all sensors attached on *Mako*. E.g the RoBIOS does not support velocity sensors and deals not with a multiplexer system used for the sonar. The *EyeBot* plugin in *SubSim* was emulated to represent its original counter part, including the RoBIOS API. The decision was made to write a *wrapper* API, which allows to run control programs written for *Mako*, without any changes in *SubSim*. This enables applications to be written for *Mako* without considering the implementation of the API in *SubSim*. Not changing the *EyeBot* plugin specifically for *Mako* also ensures a consistent and independent interface for different robots using the simulation.

5.1 Drag, Buoyancy and Propulsion in SubSim

In *SubSim* the physics engine is responsible for calculations of the physical model. A liquid model provided by a physics engine to simulate water has not been implemented. Therefore a *fake* buoyancy and drag was implemented to simulate the liquid effects of water.

In *SubSim* version 2.3 the body of *Mako* was represented by a PAL box with the dimensions $1.8m * 0.5m * 0.5m$ which results in a volume of $0.45m^3$. The weight of the body was given with $500g$ and the density of the liquid with $\rho = 0.99829kg/m^3$. These settings coupled with an error in the buoyancy calculation lead to a physical model, which allows the *Mako* model to travel in its environment under ideal conditions with no noticeable effects of draft and drift. Tasks like "wall following" can be accomplished with ease, thus making sophisticated controls unnecessary.

The applied formula in *SubSim* for the liquid drag is:

$$D = \frac{1}{2} * \rho * C_d * A * v^2 \tag{5.1}$$

Where D is the applied drag force in Newton (N) on the object body like *Mako*, ρ is the fluid density in kg/m^3, A is the frontal surface of the object in m^2(PAL Box) and v the relative velocity in m/s^2 [25]. The fake buoyancy is calculated by dividing the models volume in four equal spheres, calculating their buoyancy and applying impulses on four different symmetrical points on the body. This enables an even appliance of impulses to stabilize the body (figure 5.1).

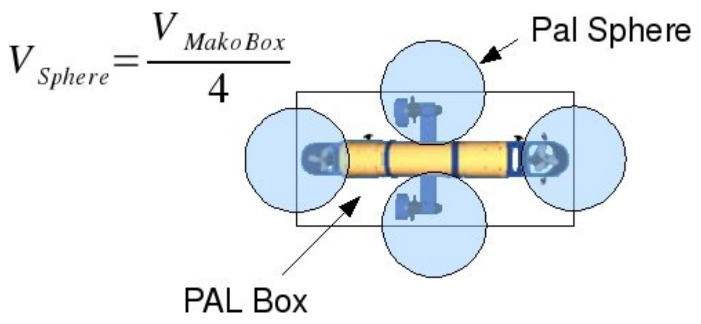


FIGURE 5.1: The *fake* buoyancy Model of *SubSim*

Changing the density of the fluid to $\rho = 998.29 kg/m^3$ and the weight of the *Mako* that it produces a buoyancy similar to the real *Mako* lead to a series of reactions, since it has also a major influence on the propulsion and drift model. Due to the fact that no units were specified anywhere the impact of this outcome could not be foreseen at the beginning of this thesis.

The propulsion model is implemented by propeller by class, which simulates a propeller actuator and a corresponding DC motor. The simplified formula to calculate the applied thrust on an attached body is

$$T = \alpha_{lumped} * V(t) \tag{5.2}$$

where α_{lumped} is an experimentally determined lumped parameter and V the voltage which is set by a control program. No units for this formula are specified.

Another outcome was found via running experiments on *SubSim* with different simulation speeds. Different speeds lead to different results. The problem is a lack in synchronisation with the simulation time and real time, but it does not effect the physics itself as seen in figure 5.2. Despite the fact that with different simulation speeds the model travels the same path, the model travels different speeds. For a simulation speed of “5”, the *Mako* model needs 38.5 seconds of simulation time to complete one circle. With a speed of “15” it needs 40 seconds and with a speed of “30” it needs around 70 seconds. The model travels an ideal circle with a diameter of around 1.3 meter. The real *Mako* needs only 0.8 meter.

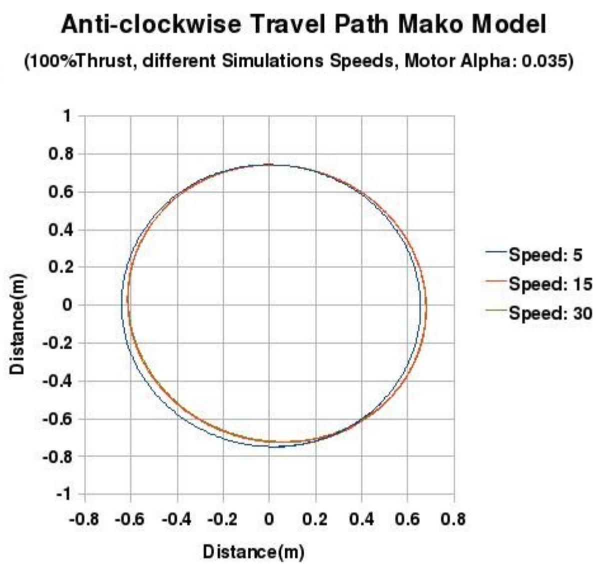


FIGURE 5.2: Anti-clockwise Spot Turn Experiment, with three different simulation speeds and changed physical model

Figure 5.3 and 5.4 show the results of acceleration experiments in *SubSim* using the same heading control as being used for *Mako*. Three different simulation speeds show the effect of the lack of synchronisation. Figure 5.4 shows that the *model* travels different distances within the same time. The maximum velocity of this model was determined to be $9cm/s$ in comparison to $44cm/s$ for its real counterpart. This is explainable due to the inexistence of an appropriate physical model.

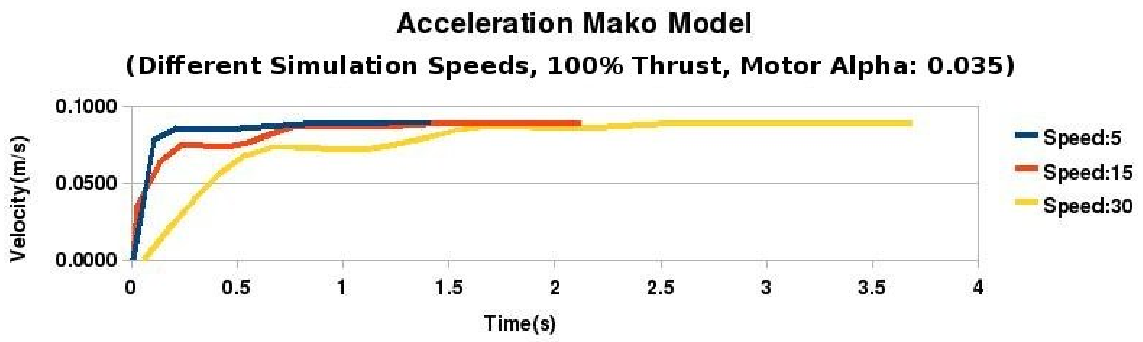


FIGURE 5.3: Velocity of the *Mako* model during the acceleration in *SubSim* with different simulation speeds and changed physical model

Due to the fact that the physical object of *Mako* model does not contain information about applied forces or accelerations on its body, no accelerometer could be implemented into *Makos* sensor suite in *SubSim*.

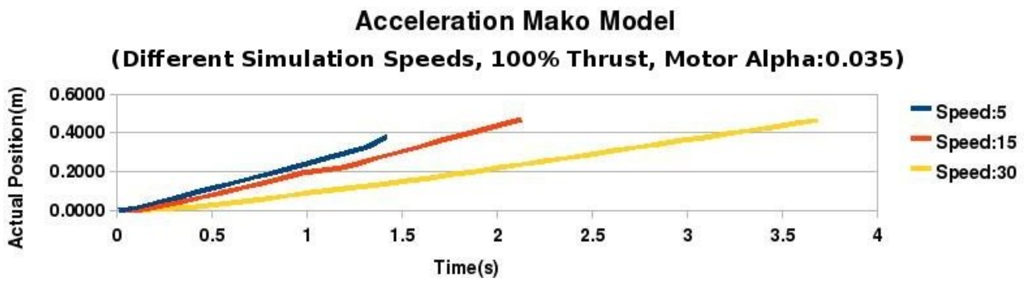


FIGURE 5.4: Surge progress of the *Mako* model in *SubSim* during the acceleration with 100% thrust, different simulation speeds and changed physical model. Significant different travel distances can be seen at different simulation speeds

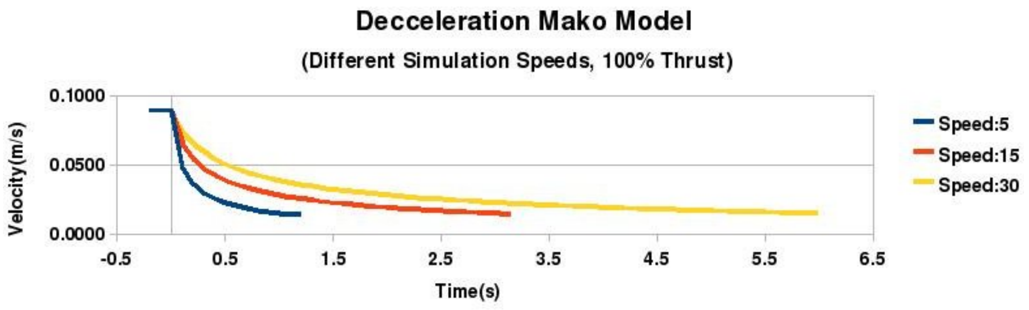


FIGURE 5.5: Surge progress of the *Mako* model in *SubSim* during the deceleration from 100% thrust, different simulation speeds and changed physical model

In theory it is possible to determine acceleration of a body with:

$$a = \frac{\Delta v}{\Delta t} \tag{5.3}$$

Where a is the acceleration in m/s^2, Δv is the difference of velocity in m/s and Δt is the difference in time in s. Due to a non consistent simulation time in *SubSim*, applying this formula does not lead to reasonable results.

5.2 Error Model

As the experiments in chapter 4 have shown, introducing a noise model to *SubSim* is crucial to test, improve and evaluate the robustness and reliability of control algorithms written for *Mako*. EyeSim played a role model for developing and implementing a noise model for *SubSim*. However the noise model in EyeSim is based on a statistical noise model. The results and the experience gained within this thesis resulted in an implementation of an error model not only based on a statistical model.

Before noise classes were defined and implemented, it was necessary to determine the output units for each single sensor class. As already described in chapter 2, the basic units meter for distances, rad for angles and seconds for time are used for the return values of the sensors.

As in EyeSim all sensors and the actuators can be influenced with a standard normal distribution with zero mean. Depending on the output value for a certain sensor class the variance of the distribution can be changed. The central limit theorem method was chosen to produce the standard distribution [55]. An appropriate algorithm was found at [35]. The model implemented in *SubSim* can be described by

$$f[i] = s[i] + n[i] \tag{5.4}$$

where $f[i]$ is the resulting signal at the time i, s the unchanged source signal and n is the standard normal distribution [45]. Further a uniform discrete distribution model was implemented as it was applied in [48]. It can be described by

$$f[i] = s[i] + u[i] \tag{5.5}$$

where u is the uniform noise.

A number of different noise sources were added to each sensor class and different settings can be chosen. E.g the the maximum range for PSD or the sonar sensors can be adjusted. Additionally for sonar sensors, settings for the maximum angle of incidence, the resolution and the response time are available. Compass sensors can be influenced by tilt noise and for velocity sensors the resolution can be changed. Despite the fact that the vision system is not part of this thesis, the error model for the *EyeBot* camera of EyeSim was adopted. This error model consists of *Salt and Pepper noise*, *Hundreds and Thousands noise* and a standard normal distribution noise (Gaussian distribution) (figure 5.6). Salt and Pepper noise adds random black and white pixels to the camera image. Hundreds and Thousands noise produces colored random pixel. The standard normal deviation is applied on all RGB (red, green, blue) channels for each individual pixel.

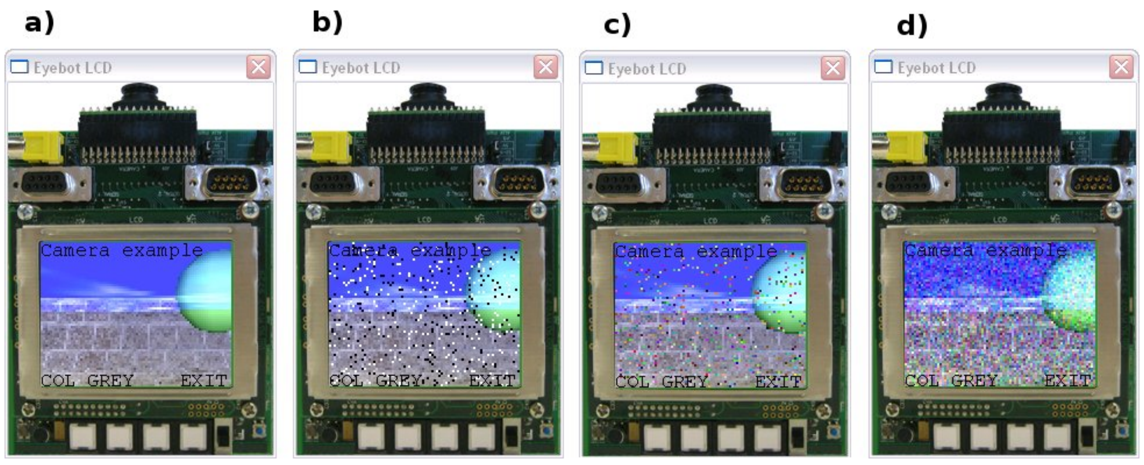


FIGURE 5.6: *EyeBot* plugin with activated camera: a) original image, b) Salt and Pepper noise, c) Hundreds and Thousands noise, d) Gaussian noise

Salt and Pepper and Hundreds and Thousands noise can be described by [45]

$$\begin{aligned} f(i) &= (1-e) * s(i) + en(i) \qquad \text{where} \\ e &= \begin{cases} 1, & \text{with probability } P_e \\ 0, & \text{otherwise} \end{cases} \end{aligned} \tag{5.6}$$

Figure 5.7 shows the designed and implemented software package “noise source‘” with available noise sources. A software package is a group of related classes. All noise sources extend the class *Noise* which provides a pure virtual function to apply “noise” on an given input value. This allows the application to *register* noise sources for each sensor class. Available noise sources for a sensor are kept in a list of the class *SubSimNoiseDevice*, which itsself extends all sensor classes and the simulated *EyeBot* camera.

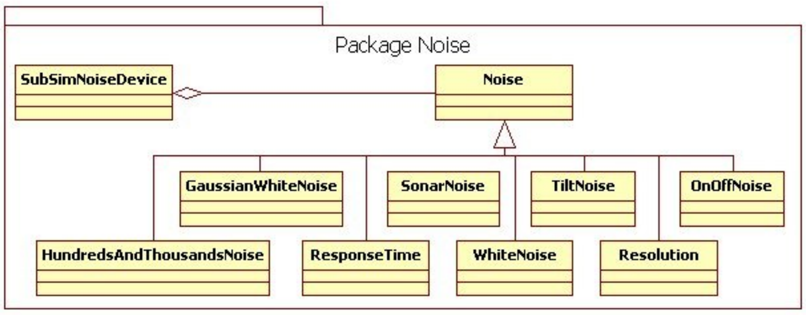


FIGURE 5.7: Noise package: the noise source family

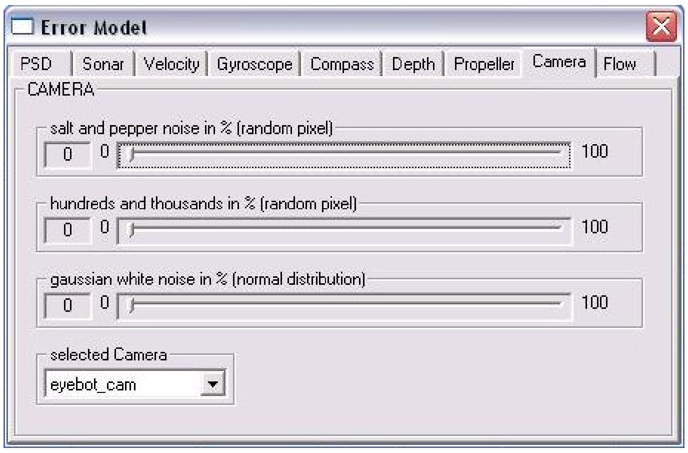
5.3 Error Model Control

The error model was implemented that it is accessible via the robot settings file or the GUI. By adding parameters into the robot settings file it is possible to preset a robots error model for each single sensor. It allows for different robot description files of the same robot, with different noise settings, to be defined. This comes quite handy if settings have to be compared and used for multiple experiments.

An error GUI control was implemented and enables access to all sensors of a robot during runtime. Allowing changes on the error model even when the control program of a simulation has been started. The GUI control is implemented in a new frame which is accessible trough the main frame menu bar (figure 5.8). It extends the *wxFrame* class and contains a *wxNotebook* object [4]. The notebook acts like a register where new *wxPanels* can be added as pages. Therefore each sensor class has its own panel hence enables an easy overview over all available sensor classes.

For each single error source, for a specific sensor class the strength of the noise can be changed with *wxSliders*. These are sliders which can be defined with a minimum and maximum range. Unfortunately floating point values are not supported.

On each sensor panel a pull down menu was implemented to allow a sensor to be chosen and thus adjusting the noise settings for each sensor individually. The names of the sensors in the pull down menu are equal to those in the robots settings file. Hence it is possible to identify each sensor with ease. If no sensor of a given sensor class is attached to a robot, the pull down menu for this sensor class indicates this simply with “none” in its text field.



salt and pepper noise in % (random pixel): 0 (0–100)

hundreds and thousands in % (random pixel): 0 (0–100)

gaussian white noise in % (normal distribution): 0 (0–100)

selected Camera: eyebot_cam

FIGURE 5.8: Error model GUI control

It has been shown in EyeSim that visualizing sensor devices and sensor readings is of great help for users in terms of debugging and understanding their applications. Within this thesis the visualization of PSD sensors and sonar sensors has been implemented. A PSD sensor is represented as a small black cube and indicates its localisation at the robots body. For the sonar sensors a frustum can be drawn, which indicates besides its localisation the beam width. This frustum changes its angle when the beam width is changed in the error model respectively. A line, projected from the sonar sensor itsself, shows the perpendicular beam of the acoustic signal. Further, to have a instantaneous visual feedback if the sonar sensor reads in its beam width, a line is drawn on intersecting surfaces of surrounding objects. This line represents a normal vector, which is calculated of the position where the beam of the sensor hits the surface. A blue line shows that the sensor reads within its beam width and switches over to red, when the sensor leaves the beam width (figure 5.9).

To simulate steady flows in water, the error model provides the possibility to apply equal impulses on all objects within a simulation. Two sliders allow the setting of longitude and latitude direction of a applied impulse, one slider determines the strength. To give the user a visual feedback of the impulse an arrow is drawn above the scene. Its direction shows respectively the direction of the applied impulse. The tail is implemented bi-coloured. Besides the arrows colour a blue line in the tail indicates the strength of the applied impulse.

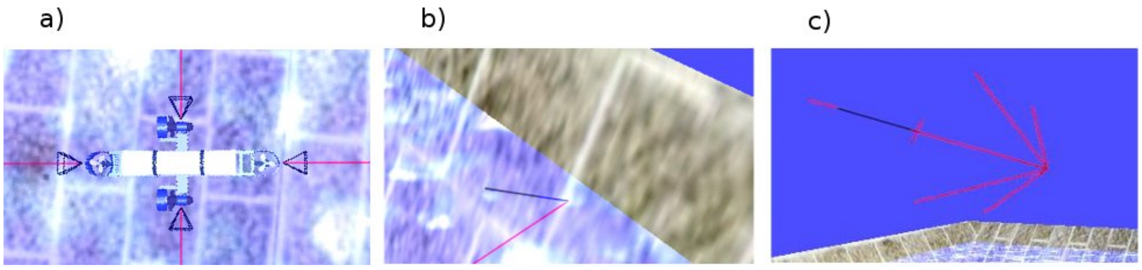


FIGURE 5.9: Visualization in *SubSim*: a) frustum of the sonar beam width, b) the normal vector of a intersecting wall from a sonar beam, c) arrow indicator for steady flow

5.4 Wall Following - Comparison

The wall following experiment presented in chapter 4 was repeated in the simulation to evaluate the changes on the physics model and the newly implemented sonar sensor. Multiple runs with different parameters for the physics model were tried. Owing to a oversimplified physical model of *Mako* and a missing liquid model provided by the physics engine it was impossible to find parameters, which lead to a behaviour of the model close to its real counterpart.

In [26] a comparison between different physic engines has been processed. The experiments show significant different physical behaviours of different engines. To find an appropriate physic engine is therefore difficult for developers. Further it indicates how important it is to find physical models as close to reality as possible.

To evaluate the experiments it was attempted to repeat the method applied in [49]. A quantitative measure of discrepancy of behaviours could be calculated with the Euclidean distance between the coordinates of a simulated and real travel path. Therefore, coordinates in commensurate time steps from the travel paths were taken. To record the travel path in *SubSim* was accomplished with ease, this was not the case for the real *Mako*. The stand used for the sport turn experiment was not high enough. A visible distance of only 5.5 meter could be recorded with the applied camera, to less for a comparison. The recorded movies however can be used for documentation purposes.

Because of this drawback the following two charts were designed to have a visible feedback of the control data. Both diagrams show the results of a wall following experiments. The same control program as it was used by experiments on the real *Mako* was applied, and the same sonar setup was attached to the model as used for the experiment described in chapter 4.

The first chart 5.10 shows five different runs of the wall following experiment with different settings on the error model. Different settings in terms of the sensor resolution and the response time were applied to visualize the effective difference. The response time is applied for each sensor individual. Thus multiplying this time by four gives the absolute response time. The higher the resolution and the higher the response time is set up, the more time it takes until the model reacts to correct its distance. A setting with the resolution of 10 centimeters and the response time of 1.2 seconds matches the observations from the real *Mako*. However, with these settings the model reacted too slowly and hit the wall.

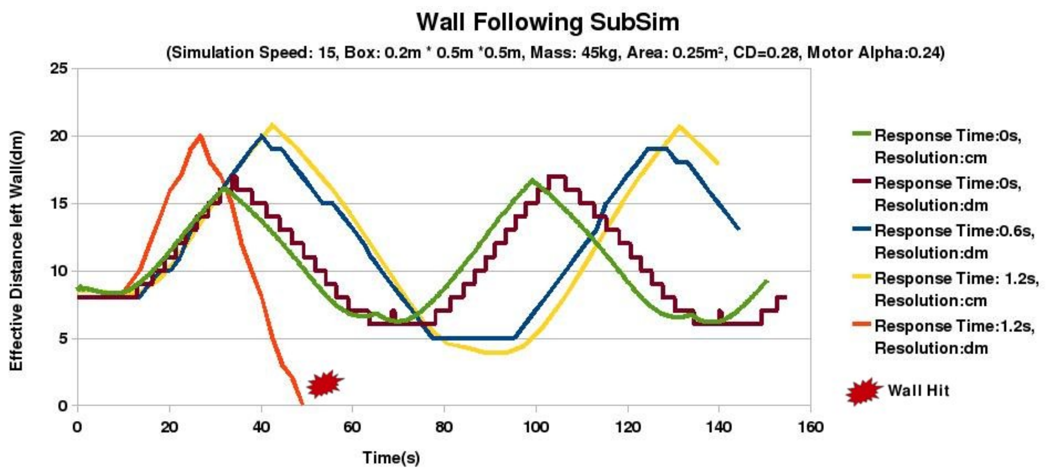


FIGURE 5.10: Comparison of different error model settings using the wall following control program

The chart seen at Figure 5.11 shows the result of the wall following experiment from chapter 4 of the real *Mako* and experiments processed with *SubSim* version 2.3 and 2.4. An example wall following control program, available for *SubSim* version 2.3 using PSD sensors, shows that no sophisticated control is necessary to accomplish the task. The *Mako* model travels on an ideal line without any deviation. However, due to the new sensor set up and due to the fact, that the FUZZY control program was not designed for this physical model, the travel path indicates a variation as seen in figure 5.11. In version 2.3, the *alpha* parameter for the motors was given with 0.5. This caused a too high thrust and thus a too fast traveling model. Another run with an *alpha* factor of 0.005 was processed to slow down the model appropriately, both runs lead to a wall hit.

The results gained from *SubSim* version 2.4 indicates a path way with a similar sinusoidal amplitude as the real *Mako* did.

5.5 Software Framework of Mako

Within this thesis, available control software for *Mako* has been adjusted, logically divided, and extended. For some sensors new classes have been designed. Many sensor classes are implemented with a singleton design pattern, since the respective physical sensors are only one time existent. This avoids multiply instances of the same sensor and hence reduces possible software failure. A FUZZY and a PID controller are now integral part of a controller system. This controller can be extended with additionally control algorithm.

For each single experiment a new main application was written. This reduced the size and the complexity of applications developed for a certain task.

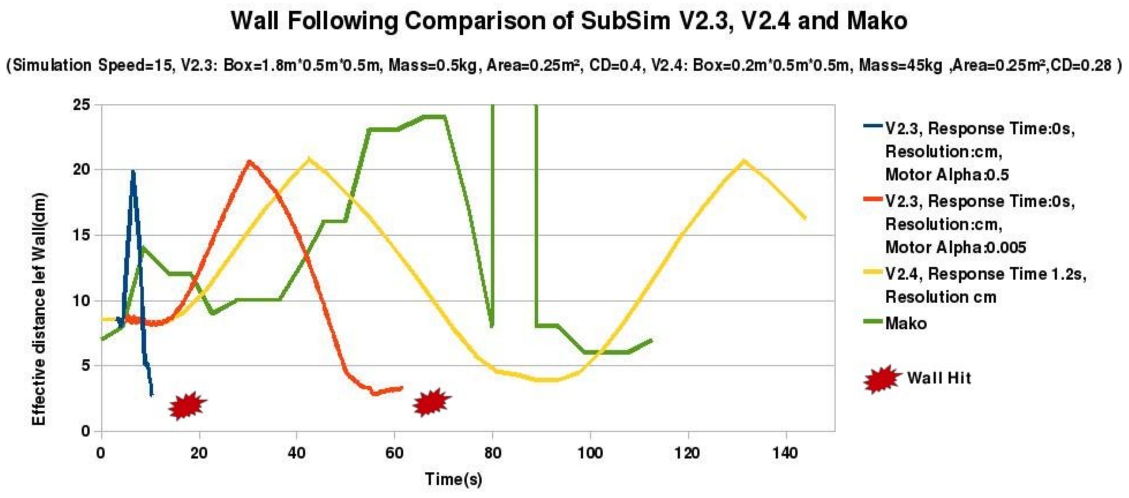


Figure 5.11: Comparison of *SubSim* version 2.3, 2.4 and *Mako* using the wall following control program

Experiences from experiments have shown that it is crucial to have fully customizable applications during runtime. This enables the setup of a control program with ease and to react to unforeseeable conditions. Therefore, much effort was spent for the user interface. As a result a full software framework *Mako* could be implemented and is available for future projects. A class diagram is shown in figure 5.12.

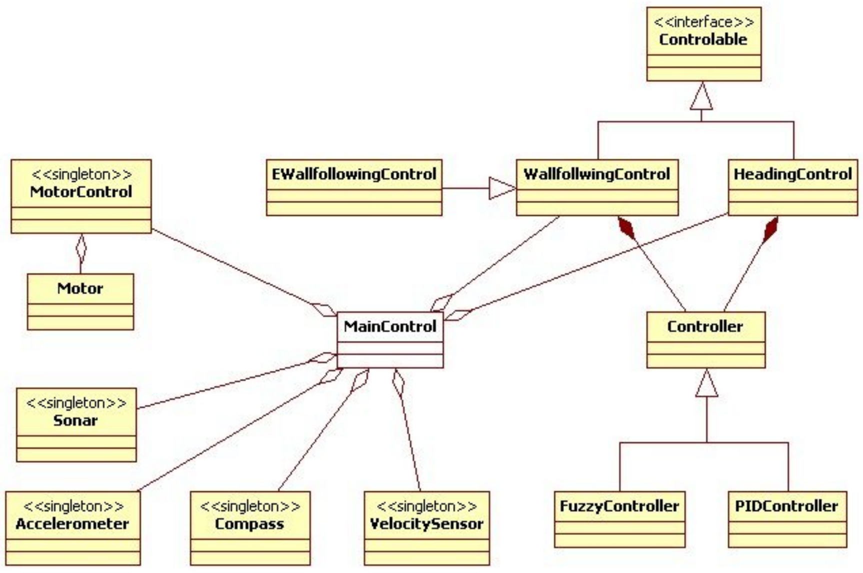


Figure 5.12: Software Framework of *Mako*

Chapter 6

Conclusions and Future Work

The comparison between *SubSim* version 2.3 in chapter 5 has indicated how big the reality gap has been at the begin of this thesis. Due to the implementation of the error model and the changes of the physical model a much closer behaviour to the real counterpart was accomplished. For the first time a controller program written for *Mako* was applied successfully in *SubSim*. The API wrapper and the new added sensor classes for *SubSim* allows now testing control programs in a simulated environment before testing it on the real *Mako*. This means a major improvement for software developing and debugging. The implementation of a steady force allows now more realistic simulations for offshore environments. *SubSim* reached version status 2.4 successfully.

A new sonar setup for *Mako* was applied and has proven more robust for the wall following task. The depth sensor system was replaced, calibrated and its water tightness was assured. It will be available for future projects. For the first time the first mission task of the project *Mako* initiative was accomplished by passing successfully a rectangular corner.

6.1 Mako

Makos body has proven robust, stable and well balanced. The solution of mounting all inertial devices on a tray inside the upper hull enabled easy access. Thus smaller defects could be repaired while conducting experiments. The maximum velocity speed has been found more than sufficient, since speed is not a matter of the missions tasks.

The velocity sensors resolution is too low to gain reasonable output values. The accelerometer is not applicable for navigational purposes due to its too low sensitivity. A stable communication with the sonar multiplexer could be accomplished after fixing a problem with floating input gates. With only 4 transducers it is impossible to navigate *Mako* through an unknown environment, even with very low velocity. Due to the fact that the vision system is not in service no image processing can be applied.

The platform *Mako* posses the prerequisites to accomplish its mission task but suffers under its insufficient sensor suite.

6.2 SubSim

An additional plugin might be introduced which simulates the needs of *Mako* and replacing the developed wrapper API used within this thesis, thus having multiple micro controller emulators for *SubSim*.

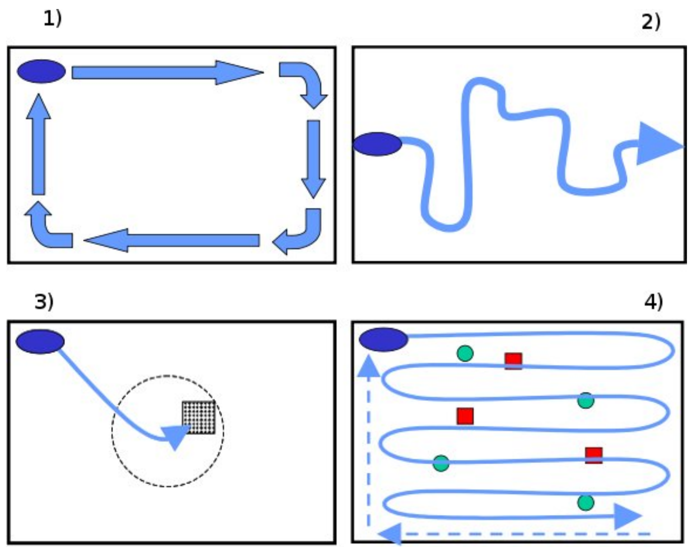
The simulation posses an easy to use GUI interface. The possibility to preset the environment via many settings files makes the application highly customizable. The XML file format ensures readability for users. Many available examples are assistant for developing new environments, robots and control programs.

The implementation of a multiple layer architecture keeps the software development clear. However the lack of documentation will be drawback for new developers. The way the physics engine is implemented might confuse developers. The possibility to update PAL with ease is not given. Specific changes for *SubSim* were applied. These changes are necessary to provide the functionality needed by *SubSim*.

For the future it is recommended to improve the physical model and to provide a developer documentation. Introducing new features to *SubSim*, without re-factoring its current state, will increase complexity, thus becoming a bottle neck.

Appendix A

Mission Tasks Mako 2005 [29]



Task1

Wall Following: The AUV is placed close to a corner of the pool. The task is to follow the pool wall without touching it. The AUV should perform one lap around the pool, return to the starting position, then stop.

Task2

Pipeline Following: A plastic pipe (diameter and color to be specified) is placed along the bottom of the pool, starting on one side of the pool and terminating on the opposite side. The pipe is made out of straight pieces and angle pieces. The AUV is placed over the beginning of the pipe close to a side wall of the pool. The task is to follow the pipe on the ground until the opposite wall has been reached.

Task3

Target Finding: The AUV is placed close to a corner of the pool. A target plate with a distinctive checkerboard texture (size and color to be specified) will be placed at a random position within a 3m diameter from the center of the pool. The task is to find the target plate in the pool, drive the AUV directly over it and drop an object on it from the AUVs payload container. (AUVs without the ability to drop a payload should just hover over the target plate).

Task4

Object Mapping. A number of simple objects (balls and boxes, color and sizes to be specified) will be placed near the bottom of the pool, distributed over the whole pool area. The AUV starts in a corner of the pool. The task is to survey the whole pool area, e.g. using a sweeping pattern, and record all objects found at the bottom of the pool. The AUV shall return to its start corner and stop there. In a subsequent data upload (AUV may be taken out of the pool and a cable attached to its on-board system), a map of some format should be uploaded that shows location and type (ball or box) of all objects found by the AUV

Appendix B

Class Diagrams

B.1 Full class diagram *SubSim* version 2.4

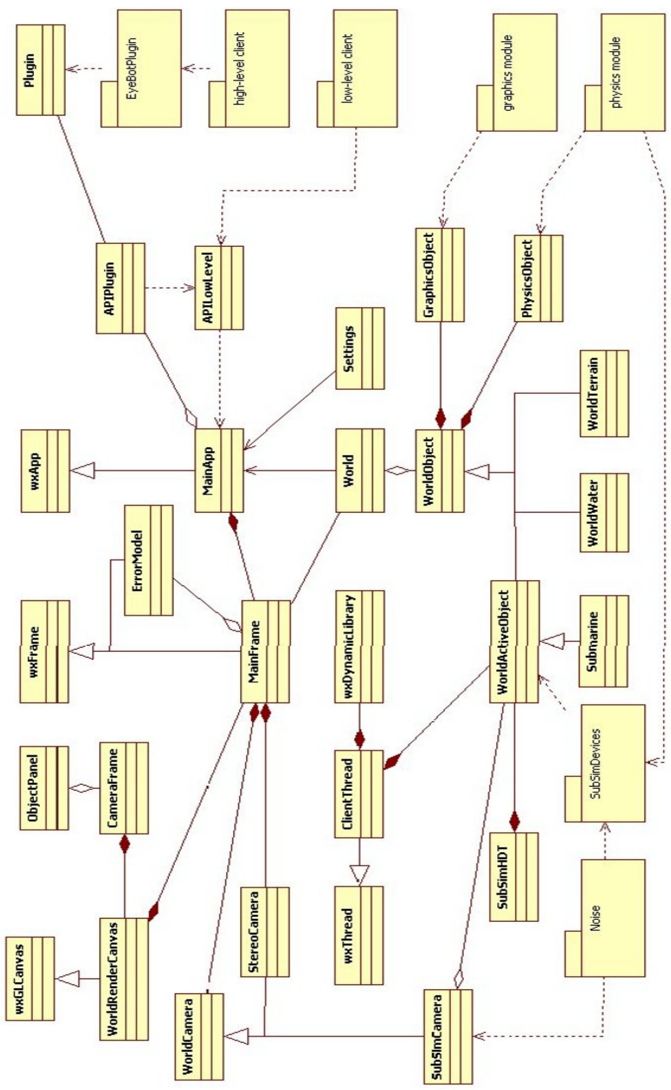


FIGURE B.1: Full class diagram of *SubSim* version 2.4

B.2 Class Diagram SubSimDevices

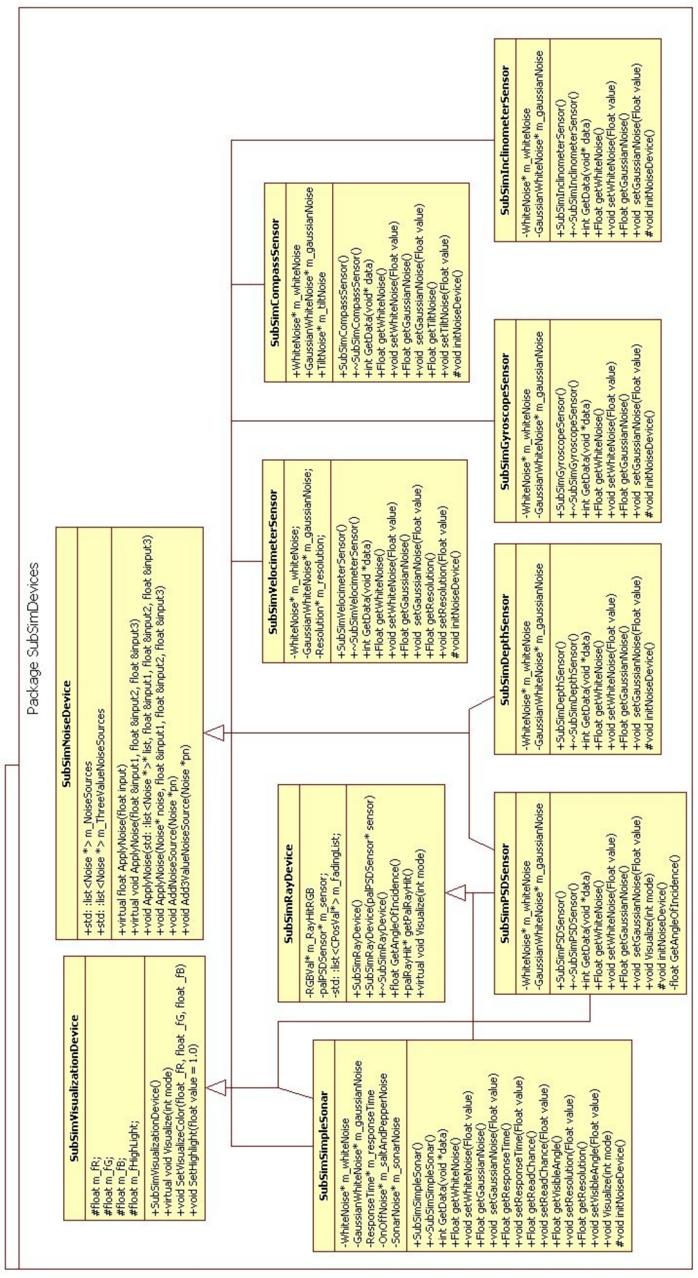

Package SubSimDevices
SubSimVisualizationDevice
#float m_fR;
#float m_fG;
#float m_fB;
#float m_fHighLight;
+SubSimVisualizationDevice()
+virtual void Visualize(int mode)
+void SetVisualizeColor(float _fR, float _fG, float _fB)
+void SetHighlight(float value = 1.0)
SubSimNoiseDevice
+std: :list<Noise *> m_NoiseSources
+std: :list<Noise *> m_ThreeValueNoiseSources
+virtual float ApplyNoise(float input)
+virtual void ApplyNoise(float &input1, float &input2, float &input3)
+void ApplyNoise(std: :list<Noise *>* list, float &input1, float &input2, float &input3)
+void ApplyNoise(Noise* noise, float &input1, float &input2, float &input3)
+void AddNoiseSource(Noise *pn)
+void Add3ValueNoiseSource(Noise *pn)
SubSimRayDevice
-RGBVal* m_RayHitRGB
-palPSDSensor* m_sensor;
-std: :list<CPosVal*> m_fadingList;
+SubSimRayDevice()
+SubSimRayDevice(palPSDSensor* sensor)
+~SubSimRayDevice()
+float GetAngleOfIncidence()
+palRayHit* getPalRayHit()
+virtual void Visualize(int mode)
SubSimVelocimeterSensor
-WhiteNoise* m_whiteNoise;
-GaussianWhiteNoise* m_gaussianNoise;
-Resolution* m_resolution;
+SubSimVelocimeterSensor()
+~SubSimVelocimeterSensor()
+int GetData(void *data)
+Float getWhiteNoise()
+void setWhiteNoise(Float value)
+Float getGaussianNoise()
+void setGaussianNoise(Float value)
+Float getResolution()
+void setResolution(Float value)
#void initNoiseDevice()
SubSimCompassSensor
+WhiteNoise* m_whiteNoise
+GaussianWhiteNoise* m_gaussianNoise
+TiltNoise* m_tiltNoise
+SubSimCompassSensor()
+~SubSimCompassSensor()
+int GetData(void* data)
+Float getWhiteNoise()
+void setWhiteNoise(Float value)
+Float getGaussianNoise()
+void setGaussianNoise(Float value)
+Float getTiltNoise()
+void setTiltNoise(Float value)
#void initNoiseDevice()
SubSimSimpleSonar
-WhiteNoise* m_whiteNoise
-GaussianWhiteNoise* m_gaussianNoise
-ResponseTime* m_responseTime
-OnOffNoise* m_saltAndPepperNoise
-SonarNoise* m_sonarNoise
+SubSimSimpleSonar()
+~SubSimSimpleSonar()
+int GetData(void *data)
+Float getWhiteNoise()
+void setWhiteNoise(Float value)
+Float getGaussianNoise()
+void setGaussianNoise(Float value)
+Float getResponseTime()
+void setResponseTime(Float value)
+Float getReadChance()
+void setReadChance(Float value)
+Float getVisibleAngle()
+void setResolution(Float value)
+Float getResolution()
+void setVisibleAngle(Float value)
+void Visualize(int mode)
#void initNoiseDevice()
SubSimPSDSensor
-WhiteNoise* m_whiteNoise
-GaussianWhiteNoise* m_gaussianNoise
+SubSimPSDSensor()
+~SubSimPSDSensor()
+int GetData(void *data)
+Float getWhiteNoise()
+void setWhiteNoise(Float value)
+Float getGaussianNoise()
+void setGaussianNoise(Float value)
+void Visualize(int mode)
#void initNoiseDevice()
-float GetAngleOfIncidence()
SubSimDepthSensor
-WhiteNoise* m_whiteNoise
-GaussianWhiteNoise* m_gaussianNoise
+SubSimDepthSensor()
+~SubSimDepthSensor()
+int GetData(void *data)
+Float getWhiteNoise()
+void setWhiteNoise(Float value)
+Float getGaussianNoise()
+void setGaussianNoise(Float value)
#void initNoiseDevice()
SubSimGyroscopeSensor
-WhiteNoise* m_whiteNoise
-GaussianWhiteNoise* m_gaussianNoise
+SubSimGyroscopeSensor()
+~SubSimGyroscopeSensor()
+int GetData(void *data)
+Float getWhiteNoise()
+void setWhiteNoise(Float value)
+Float getGaussianNoise()
+void setGaussianNoise(Float value)
#void initNoiseDevice()
SubSimInclinometerSensor
-WhiteNoise* m_whiteNoise
-GaussianWhiteNoise* m_gaussianNoise
+SubSimInclinometerSensor()
+~SubSimInclinometerSensor()
+int GetData(void* data)
+Float getWhiteNoise()
+void setWhiteNoise(Float value)
+Float getGaussianNoise()
+void setGaussianNoise(Float value)
#void initNoiseDevice()

FIGURE B.2: Class diagram software package SubSimDevices of *SubSim* version 2.4 with members

Appendix C

Motor Characteristics

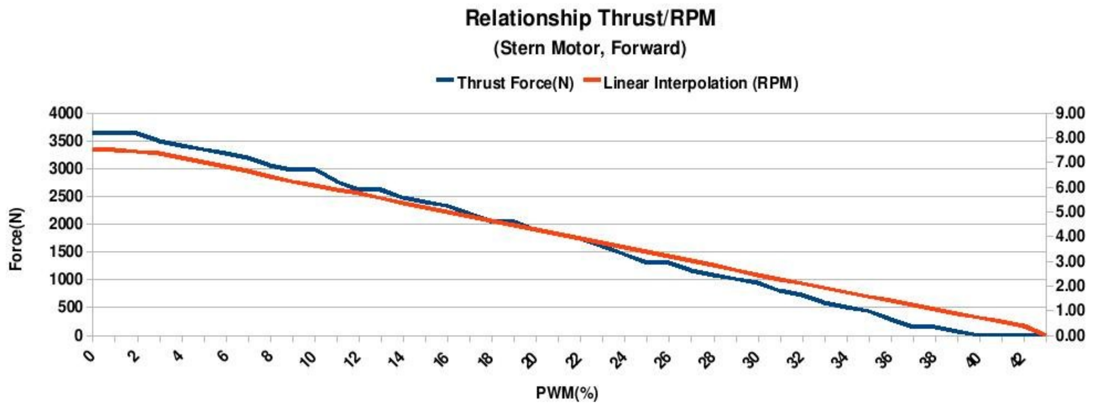


FIGURE C.1: Motor revolution versus actuator thrust force, stern motor forward direction

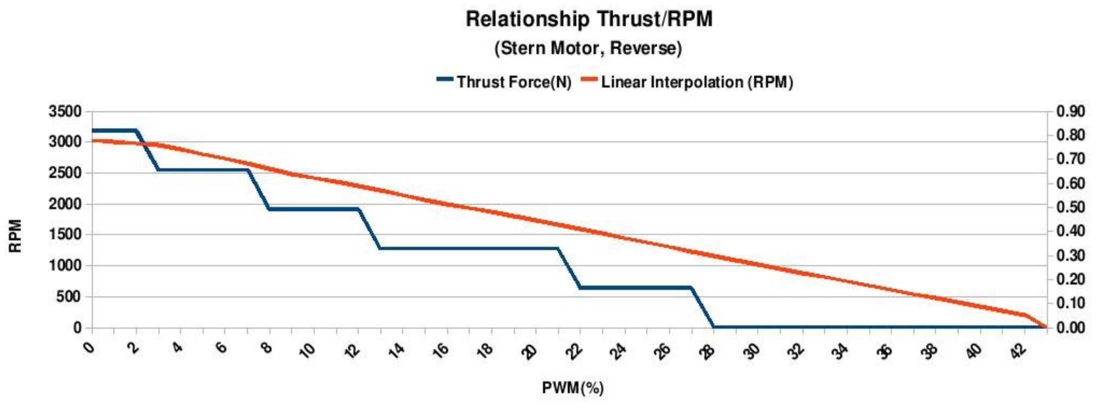


FIGURE C.2: Motor revolution versus actuator thrust force, stern motor reverse direction

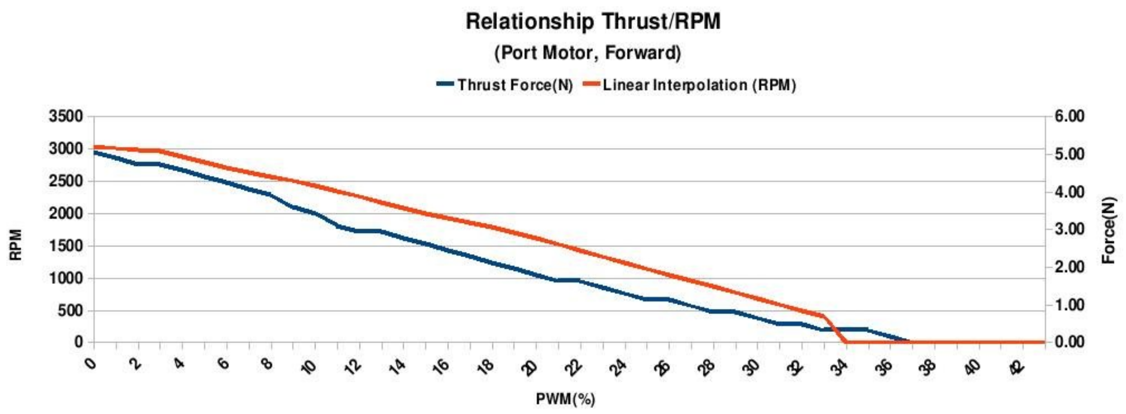


Figure C.3: Motor revolution versus actuator thrust force, port motor forward direction

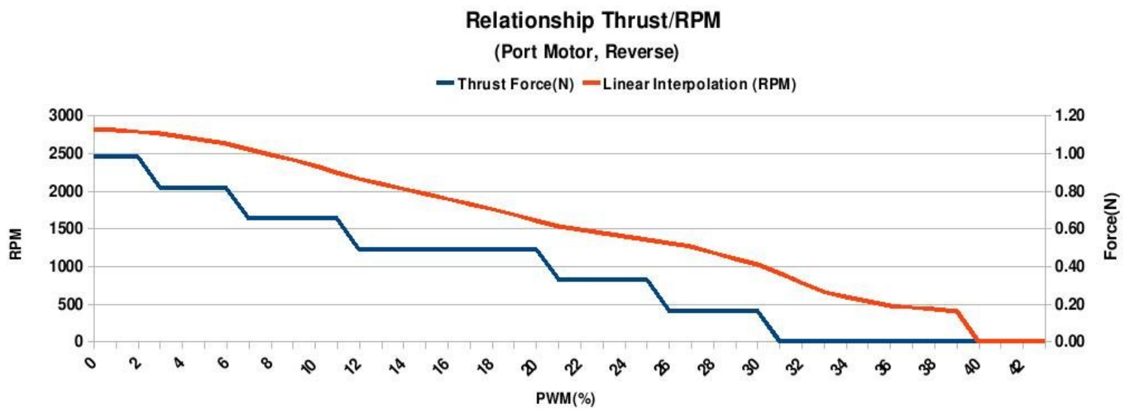


Figure C.4: Motor revolution versus actuator thrust force, port motor reverse direction

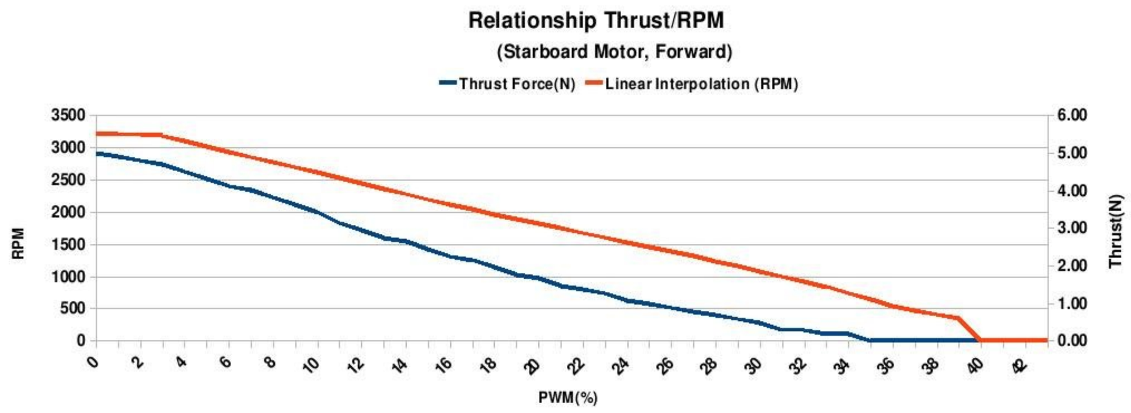


FIGURE C.5: Motor revolution versus actuator thrust force, starboard motor forward direction

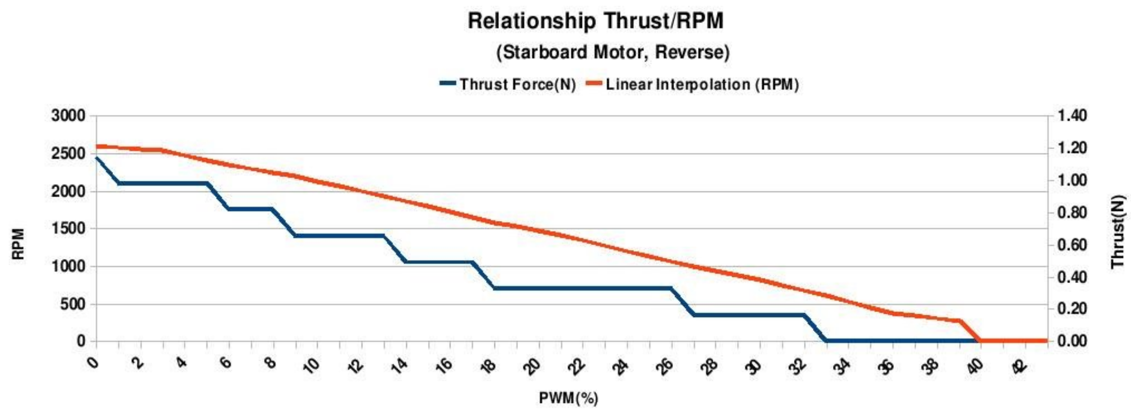


FIGURE C.6: Motor revolution versus actuator thrust force, starboard motor reverse direction

Appendix D

Accelerometer Characteristics

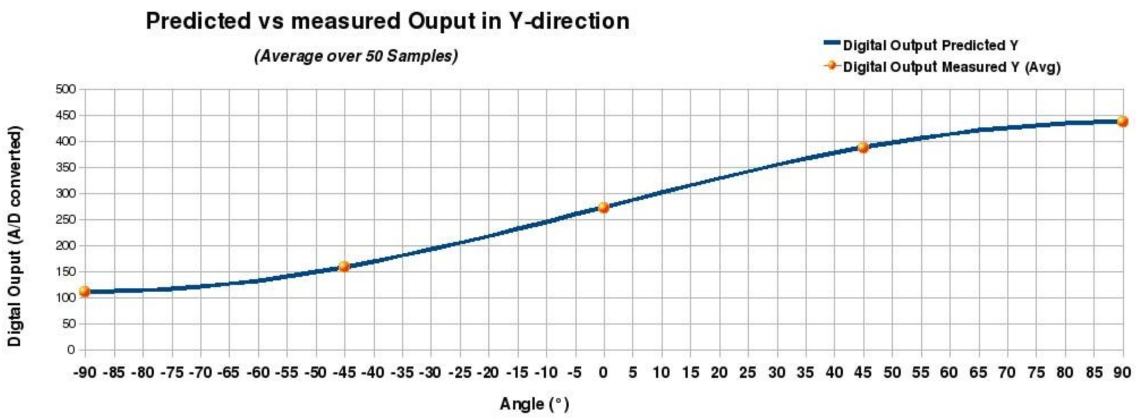


FIGURE D.1: Angular response of the accelerometer y-axis

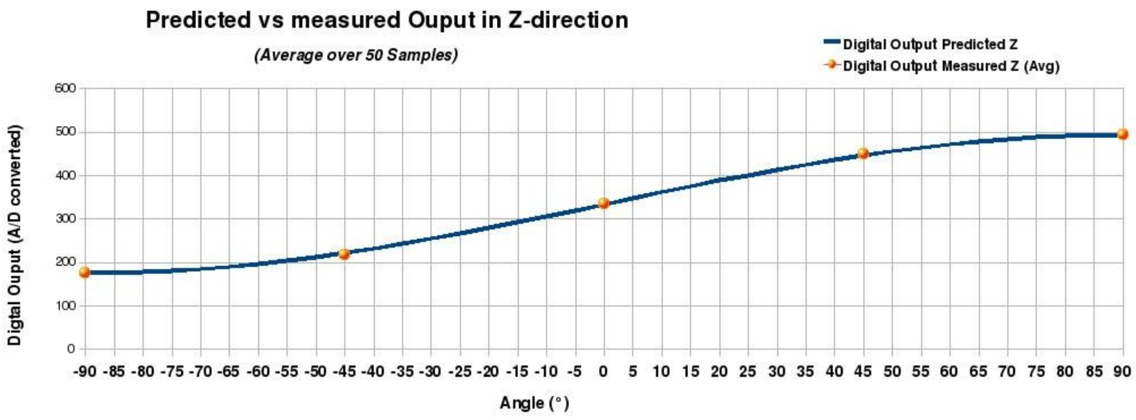


FIGURE D.2: Angular response of the accelerometer z-axis

Appendix E

Photos of Experiment Setups

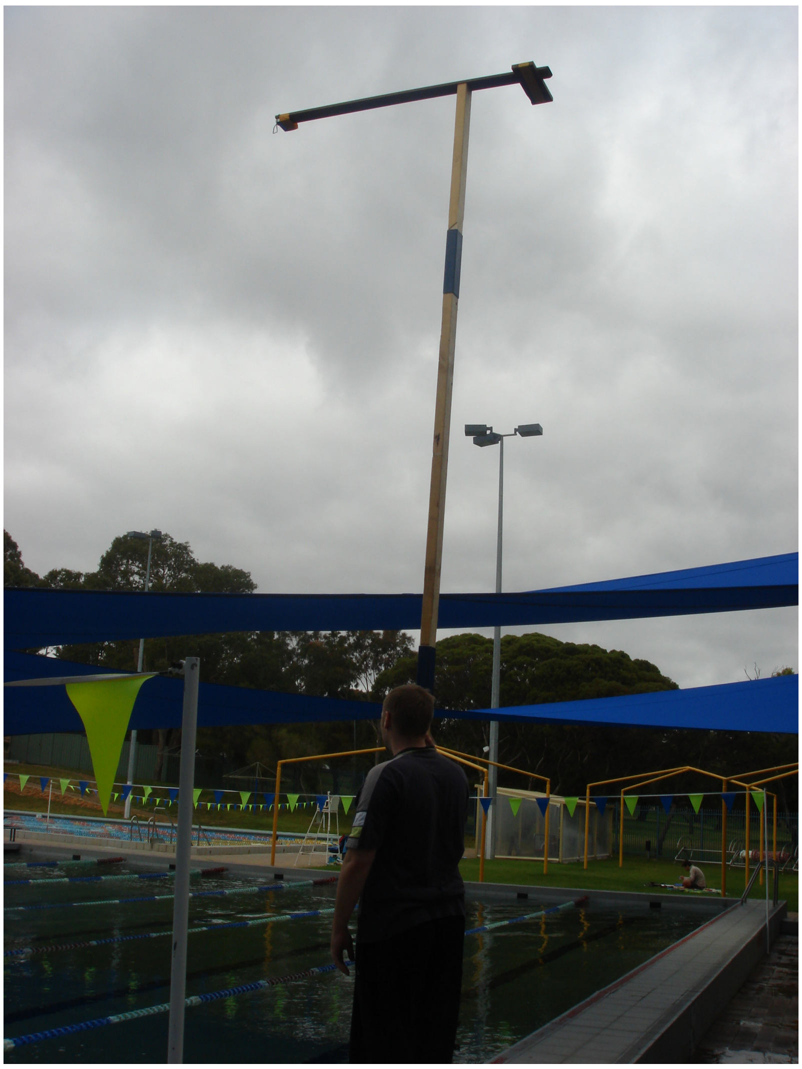


FIGURE E.1: With a 4.5m high stand the scene could be recorded and observed from the bird's eye view

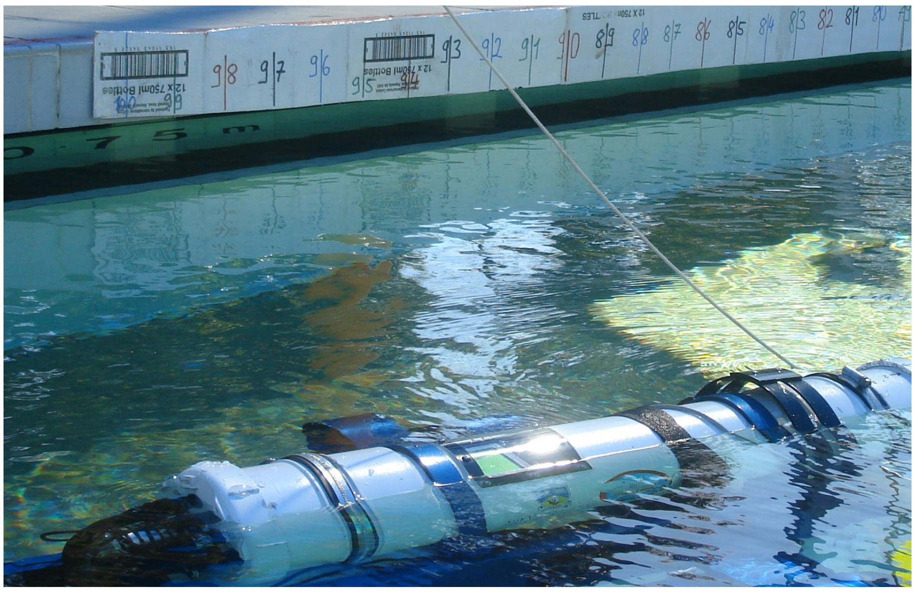


FIGURE E.2: A 10m long measure tape was attached on the wall to determine the velocity of *Mako*

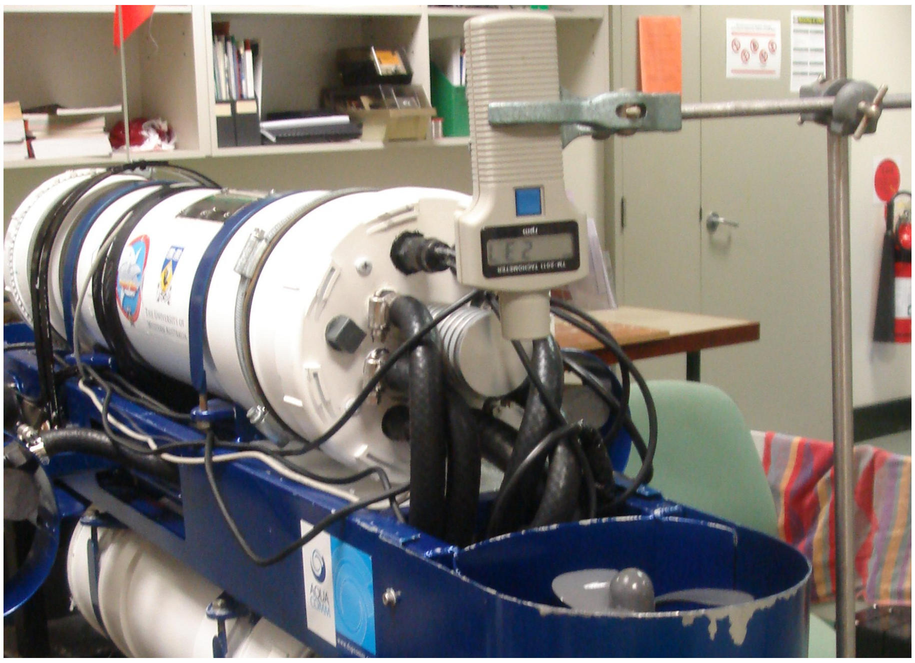


FIGURE E.3: An optical tachometer was used to measure the revolutions of the propellers

Appendix F

Upper and lower PVC Hull

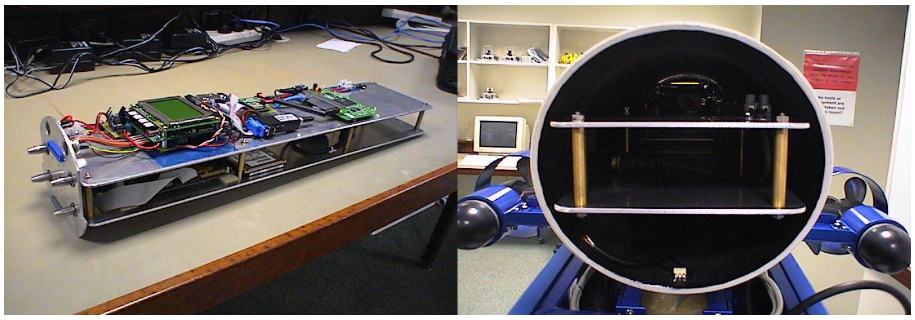


FIGURE F.1: The upper PVC Hull and the Tray with internal sensors, the sonar multiplexer and *EyeBot* microcontroller

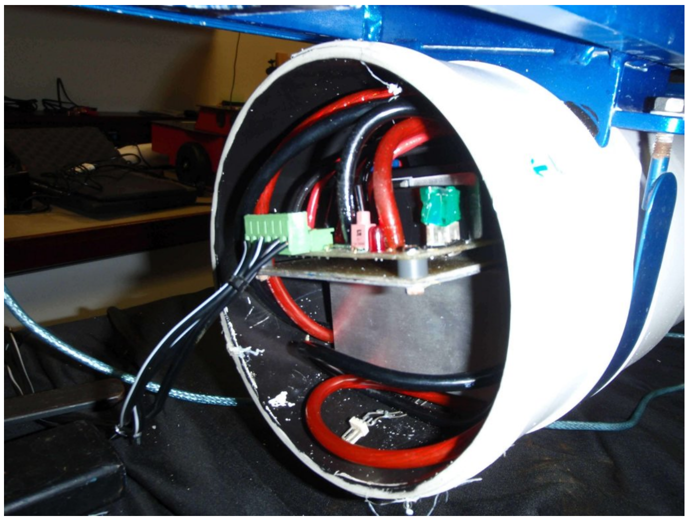


FIGURE F.2: The lower PVC Hull with its containing three batteries for the power supply

Appendix G

Multiplexer Circuit Diagram

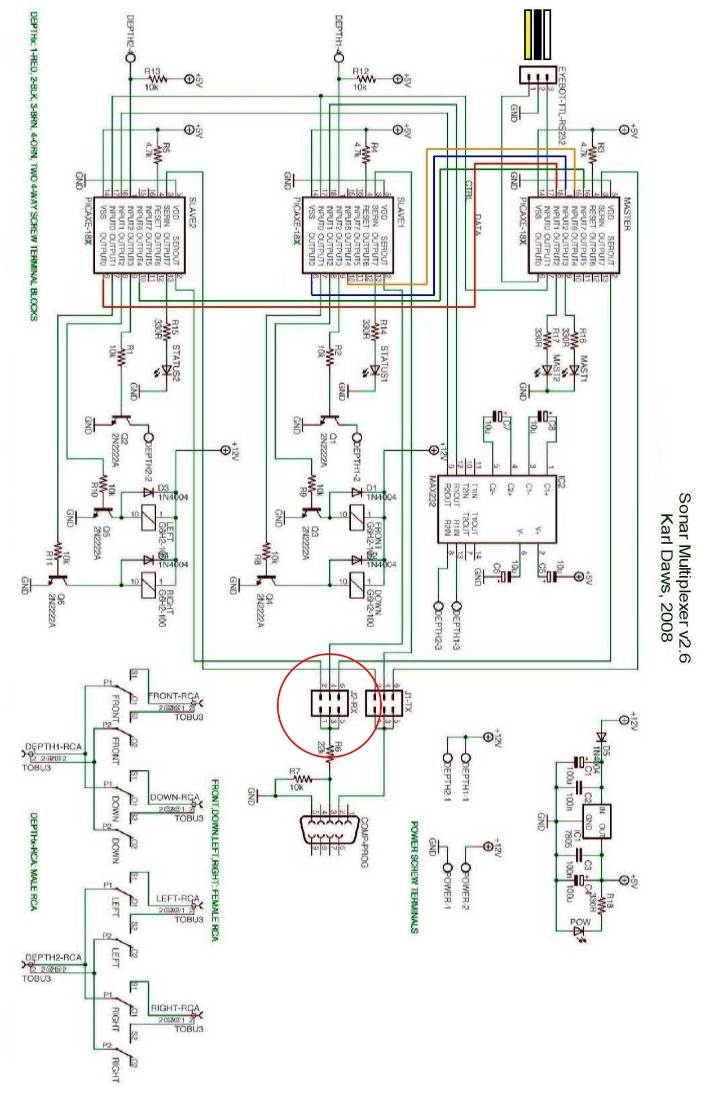


Figure G.1: The multiplexer of the sonar system. The red circle highlights the open gates [33]

References

[1] Underwater robots. [online], 2008. Available: http://ezinearticles.com/?Underwater-Robots&id=408074.

[2] Newton game dynamics. [online], 2008. Available: http://www.newtondynamics.com/.

[3] Physx by nvidia. [online], 2008. Available: http://www.nvidia.com/object/nvidia_physx.html.

[4] wxwidgets - cross platfrom library. [online], 2008. Available: www.wxwidgets.org.

[5] Tinyxml. [online], 2008. Available: http://www.grinninglizard.com/tinyxml/.

[6] Hydroid, inc. [online], 2008. Available: http://www.hydroidinc.com/contact.html.

[7] Kongsberg maritime. [online], 2008. Available: http://www.km.kongsberg.com/.

[8] Bluefin robotics. [online], 2008. Available: http://www.bluefinrobotics.com/.

[9] Forsvarets forskningsinstitutt. [online], 2008. Available: http://www.mil.no/felles/ffi/hugin/start/program/.

[10] Tortuga ii autonomous underwater vehicle. [online], 2008. Available: http://ram.umd.edu/trac/attachment/wiki/Public/UMD_AUVSI_2008_paper.pdf?format=raw.

[11] Auvsi and onr's 11th international autonomous underwater vehicle competition. [online], 2008. Available: http://www.auvsi.org/competitions/water.cfm.

[12] Autonomous underwater vehicles, a collection of groups and projects. [online], 2008. Available: http://www.transit-port.net/Lists/AUVs.Org.html.

[13] Opengl. [online], 2008. Available: http://www.opengl.org/.

[14] Cyberbotics. [online], 2008. Available: http://www.cyberbotics.com/.

[15] Sensor. [online], 2008. Available: http://en.wikipedia.org/wiki/Sensor.

[16] Sensor terminology. [online], 2008. Available: http://zone.ni.com/devzone/cda/ph/p/id/227#toc0.

[17] Understanding analog to digital converter specifications. [online], 2008. Available: http://www.embedded.com/columns/technicalinsights/60403334?_requestid=157002.

[18] Fact sheet of mma7260q. Data sheet, 2004. Available: http://www.freescale.com/files/sensors/doc/fact_sheet/MMA7260QFS.pdf.

[19] Installation and operation manual. Data sheet, 2004. Available: http://www.navmanmarine.net/upload/Marine/Internet_Manuals/2000/2100_fuel_MN000207A_eng_web.pdf.

[20] Data sheet. Data sheet, 2004. Available: http://www.robotstorehk.com/vector-2x.pdf.

[21] General description. Data sheet, 2004. Available: http://www.sensortechnics.com/download/sx-052.pdf.

[22] Elliot Alfirevich. Depth and position sensing for an autonomous underwater vehicle. Master's thesis, The University of Western Australia, 2005.

[23] Georg A. Bekey. *Autonomous Robots: From Biological Inspiration to Implementation and Control*. The MIT Press, 2005.

[24] Tobias Bielohlawek. Subsim - an autonomous underwater vehicle simulation system. Master's thesis, The University of Western Australia, 2006.

[25] Adrian Boeing. Pal - physical abstraction layer. [online], 2008. Available: http://www.adrianboeing.com/pal/.

[26] Adrian Boeing Thomas Bräunl. Evaluation of real-time physics simulation systems. In *Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia*, pages 281–288, 2007.

[27] Thomas Bräunl. *Embedded robotics : mobile robot design and applications with embedded systems*. Springer, 2003.

[28] Thomas Bräunl. E y e b o t. [online], 2008. Available: http://robotics.ee.uwa.edu.au/eyebot/.

[29] Thomas Bräunl, Adrian Boeing, Louis Gonzales, Andreas Koestler, Minh Nguyen, and Joshua Petitt. The autonomous underwater vehicle initiative project mako. In *Robotics, Automation and Mechatronics, 2004 IEEE Conference*, volume 1, pages 446– 451, 2004.

[30] Rodney A. Brooks. Artificial life and real robots. In *Proceedings of the First European Conference on Artificial Life*, pages 3–10, 1992.

[31] S. K. Choi and J. Yuh. A virtual collaborative world simulator for underwater robotos using multi-dimensional synthetic enviroment. In *Proceedings of the 2001 IEEE International Conference on Robotics & Automation*, 2001.

[32] Jack Coleman. Undersea drones pull duty in iraq hunting mines. [online], 2003. Available: `www.hydroidinc.com/pdfs/Cape_Cod_Times_Archives.pdf`.

[33] Karl Daws. Autonomous control of an underwater vehicle. Master's thesis, The University of Western Australia, 2008.

[34] Andrew de Gruchy. Autonomous control of an underwater vehicle. Master's thesis, The University of Western Australia, 2008.

[35] Matt Donadio. How to generate white gaussian noise. [online], 2008. Available: `http://www.dspguru.com/howto/tech/wgn.html`.

[36] Control Engineering. Loop tuning fundamentals. [online], 2008. Available: `http://www.controleng.com/article/CA307745.html`.

[37] International Submarine Engineering. Ise web based auv designinfo explorer. [online], 2002. Available: `http://www.ise.bc.ca/AUV_Design/ISEs%20-%20Design%20Your%20Own%20AUV%20_%20ISE-E172-TCN-001-00%20.pdf`.

[38] H. R. Everett. *Sensors for Mobile Robots, Theorie and Application.* A K Peters, Ltd., 1995.

[39] Louis Andrew Gonzalez. Design, modelling and control of an auv. Master's thesis, The University of Western Australia, 2004.

[40] Cédric Hartland and Nicolas Bredèche. Evolutionary robotics, anticipation and the reality gap. In *Proceedings of the 2006 IEEE International Conference on Robotics and Biomimetics*, 2006.

[41] Lindsey Hines. Digital compass accuracy. Technical report, University of California, 2007. Available: `http://www.mbari.org/education/internship/07interns/papers/InternPapers/LHines.pdf`.

[42] John M. Holland. *Designing Autonomous Mobile Robots.* Elsevier, 2004.

[43] Nick Jakobi, Phil Husbands, and Inman Harvey. Evolution versus design: Controlling autonomous robots. In *Proceedings of 3rd Annual Conference on Arti cial Intelligence, Simulation and Planning, IEEE*, pages 139–146, 1992.

[44] Nick Jakobi, Phil Husbands, and Inman Harvey. Noise and the reality gap: The use of simulation in evolutionary robotics. In *Design and Development of Autonomous Agents, IEE Colloquium*, pages 1/1–1/3, 1995.

[45] Andreas Koestler and Thomas Bräunl. Mobile robot simulation with realistic error models. In *International Conference on Autonomous Robots and Agents, ICARA*, pages 46–51, 2004.

[46] Daniel Loung Huat Lim. Design of a vision system for an autonomous underwater vehicle. Master's thesis, The University of Western Australia, 2004.

[47] Justin E. Manley. The role of risk in auv development and deployment. In *OCEANS 2007 - Europe*, pages 1–6, 2007.

[48] Roger Meier, Terrence Fong, Charles Thorpe, and Charles Baur. A sensor fusion based user interface for vehicle teleoperation. In *International Conference on Field and Service Robotics*, 1999.

[49] Orazio Miglino, Henrik Hautop Lund, and Stefano Nolfi. Evolving mobile robots in simulated and real environments. *Artificial Life*, 2:417–434, 1995.

[50] Minh Tu Nguyen. Design of an active acoustic sensor system for an autonomous underwater vehicle. Master's thesis, The University of Western Australia, 2004.

[51] Ewald von Puttkamer Rainer Trieb. The 3d7- simulation environment: A tool for autonomous mobile robot development. In *Proceedings of the Second International Workshop on Modeling, Analysis, and Simulation On Computer and Telecommunication Systems*, pages 358–361. IEEE Computer Society Washington, DC, USA, 1994.

[52] Robert E. Shannon. *Systems Simulation the art and science.* Prentice-Hall, 1975.

[53] Marcelo Godoy Simoes. Introduction to fuzzy control. [online], 2008. Available: `http://egweb.mines.edu/faculty/msimoes/tutorials/Introduction_fuzzy_logic/Intro_Fuzzy_Logic.pdf`.

[54] D. G. Tucker and B. K. Gazey. Applied underwater acoustics. In *Journal of Sound and Vibration*, volume 6, pages 472–473, 1965.

[55] Eric W. Weisstein. Central limit theorem. [online], 2008. Available: `http://mathworld.wolfram.com/CentralLimitTheorem.html`.

[56] Liu Yongkuan. Auvs trends over the world in the future decade. In *Autonomous Underwater Vehicle Technology, 1992. AUV '92., Proceedings of the 1992 Symposium*, pages 116–127, 1992.

[57] L. A. Zadeh. Fuzzy sets. 1965.