Implementation and Evaluation of an Adaptive Neural Fuzzy Inference Control Algorithm for a Mobile Robot

---

BACHELOR THESIS 2

In Partial Fulfillment

of the Requirements for the Degree

“Bachelor of Science in Engineering”

Course of Studies:

“Mechatronic - Mechanical Engineering”

Management Center Innsbruck

Thesis Advisor:

Dr. Gordon Lee

(San Diego State University)

Thesis Reviewer:

Dr. Andreas Mehrle
(Management Center Innsbruck)

Author:
Gerold Huber
0810602018

Declaration in Lieu of Oath

I hereby declare, under oath, that this bachelor thesis has been my independent work and has not been aided with any prohibited means. I declare, to the best of my knowledge and belief, that all passages taken from published and unpublished sources or documents have been reproduced whether as original, slightly changed or in thought, have been mentioned as such at the corresponding places of the thesis, by citation, where the extent of the original quotes is indicated.

The paper has not been submitted for evaluation to another examination authority or has been published in this form or another.

Place, Day / Month / Year	Signature

Acknowledgements

I wish to thank all my friends and family for their inspiration and believe in my dreams, the Austrian Marshall Plan Foundation for the financial support, their representative at the MCI for helping me with the application, Dr. Lee, Dr. Paolini and all the other people from the San Diego State University for the good collaboration and answers to any questions and especially my parents, who make all my adventures even possible. I could not have done this thesis without you, Thank You!

Abstract of the Thesis

This project deals with the implementation of an adaptive neuro fuzzy inference system (ANFIS) controller on an Arduino ATmega microcontroller board used in a mobile robotic system at the San Diego State University (SDSU). An ANFIS is a fusion of two traditional intelligent control algorithms: a Fuzzy Inference System and an Adaptive Neural Network. This makes the ANFIS architecture an appealing and widely used control strategy, as it features advantages of both approaches.

The current robotic system was developed through several projects and is designed as a tele-robotic system. This makes it possible to control the robot via a web interface, using an arbitrary Internet browser. The robot could for example navigate in San Diego, while the user is controlling it using a smart phone in Austria. The interface shows a live stream of the webcam on the robot, as well as the most relevant data about the robot itself and a certainty grid, showing the robot's environment detected by an ultra sonic sensor array.

To operate in an autonomous mode, the ANFIS code provides the additional opportunity of tracking a path, planned by e.g. a virtual field force strategy. The implemented ANFIS code is designed to be very flexible and adaptable, through processing serial requests from the single board computer, using the developed 'pcb2arduino'-protocol. Besides the ANFIS code, an inertial measurement unit serves as an orientation sensor and a simple path tracking strategy was implemented. It is possible to change the architecture and load/store different parameter sets from/on the EEPROM as well as switching between tele-operated and autonomous mode. Hence, the robot with the ANFIS-controller provides a mobile robotic testbed, that can be used for future algorithm research studies.

Kurzfassung

Dieses Projekt beschäftigt sich mit der Implementierung eines 'Adaptive Neuro Fuzzy Inference Systems' (ANFIS) auf ein Aruino ATmega Microcontroller-Board, das auf einem mobilen Roboter der 'San Diego State University' zum Einsatz kommt. Ein ANFIS, ist dabei eine Mischung der beiden traditionellen Ansätze für intelligente Steuerung: 'Fuzzylogic' und 'Neuronale Netzwerke'. Die neue Technik des ANFIS vereint die Vorteile beider Ansätze und kommt heute in verschiedensten Steuereinrichtungen häufig zum Einsatz.

Der erwähnte Roboter basiert auf dem bekannten Roomba Staubsaugerroboter und wurde im Zuge mehrere Projekt- und Abschlussarbeiten, als ein über das öffentliche Internet fernsteuerbares Tele-Robotic System enwickelt. Dies macht es möglich den Roboter, der in San Diego stationiert ist, zum Beispiel über einen beliebigen Internet Browser auf einem Smartphone von Österreich aus zu steuern. Der/die Benutzter/Benuzterin sieht den Livestream einer auf dem Roboter angebrachten Kamera, sowie ein Koordinatennetz, das von Ultrasonic-Sensoren detektierte Objekte in der unmittelbaren Umgebung des Roboters anzeigt.

Um den Roboter auch in einem autonomen Modus zu betreiben, wurde der erwähnte ANFIS-Controller implementiert. Dieser erlaubt es dem Roboter einem mittels 'Virtual Field Force' geplanten Pfad zu folgen und dabei aus seinen Fehlern zu lernen. Das Microcontroller-Programm wurde als möglichst flexibel entwickelt und erlaubt es Befehle laut dem eigenen 'pcb2arduino'-Protokoll entgegen zu nehmen und zu bearbeiten. Neben dem ANFIS Code, wurde auch eine 'Inertial Measurement Unit' (IMU) als essentieller Orientierungs-Sensor und eine einfache Pfadverfolgungs-Strategie implementiert. Um nur die wichtigsten Optionen zu nennen, können zum Beispiel verschiedene Parametersets geladen/gespeichert werden und sogar die komplette Architektur, kann mit unterschiedlicher Anzahl von Eingängen und 'Membershipfunctions' des Systems, während des Betriebes des Roboters verändert werden. Damit bildet das neue Roboter-Controller System eine ideale Grundlage für weitere Algorithmusstudien und Entwicklungen und eröffnet neue Möglichkeiten zur Forschung der Verbindung Mensch-Maschine.

Contents

1 Introduction **1**
- 1.1 Background . . . 1
- 1.2 Thesis Organization . . . 1

2 The Mobile Robotic System **3**
- 2.1 iRobot . . . 4
- 2.2 Sensors . . . 5
  - 2.2.1 Webcam . . . 5
  - 2.2.2 Ultrasonic Sensor Array . . . 5
  - 2.2.3 Thermal Sensor . . . 7
  - 2.2.4 Inertial Measurement Unit . . . 8
- 2.3 Arduino Platform . . . 8
- 2.4 Single Board Computer . . . 9
  - 2.4.1 Telerobotic System . . . 10
  - 2.4.2 Path Planning . . . 12

3 Inertial Measurement Unit **14**
- 3.1 IMU Sensors . . . 14
- 3.2 AHRS Code . . . 15

4 Path Tracking **17**
- 4.1 Definitions of the Coordinate Systems . . . 17
- 4.2 Position Dead Reckoning . . . 18
  - 4.2.1 Using IMU Data . . . 18
  - 4.2.2 Using Desired Data . . . 18
- 4.3 Tracking Strategy . . . 20

5 Communication Protocol between SBC and Arduino **22**
- 5.1 Single Board Computer to Microcontroller . . . 25
  - 5.1.1 Print Data Commands . . . 25
  - 5.1.2 ANFIS Commands . . . 26

5.1.3 Path Tracking Commands 29
5.1.4 Miscellaneous 29
5.2 Microcontroller to Single Board Computer 29

6 ANFIS Controller **31**
6.1 How an ANFIS Controller Works 31
6.1.1 Fuzzy Inference System 31
6.1.2 Adaptive Neural Network 32
6.1.3 Adaptive Neural Fuzzy Inference System 33
6.2 Learning Algorithms 36
6.2.1 Gradient Descent Method 36
6.2.2 Least Squares Estimation 38
6.2.3 Hybrid Learning Procedure 40
6.3 Implementation of the ANFIS-Code 40
6.3.1 Issues with the Micro-Controller 40
6.3.2 Modified MIMO ANFIS Controller 42
6.3.3 Collecting Training Data 46
6.3.4 Off-Line Learning 48
6.3.5 On-Line Learning 49

7 Results **51**
7.1 ANFIS Training 51
7.2 ANFIS Responding Time 53
7.3 Tracking in Telerobotic Mode 54
7.4 Tracking in Autonomous Mode 57

8 Conclusions and Future Work **60**

Figures **VII**

Tables **IX**

References **X**

Abbreviations **XIII**

Attachment **XIV**

Chapter 1

Introduction

1.1 Background

The field of robotics has grown from simple industrial robots that paint cars and place components on an assembly line to robots that have human-like qualities, can work in colonies and can learn from their mistakes. Today, there are many applications in military, commercial and civilian scenarios where one or more robots must perform complex tasks in an uncertain environment. For example, autonomous intelligent robot colonies may be used in reconnaissance missions or seek-and-capture scenarios involving a complex set of interactions between machines as well as between machines and humans and may cover long distances to remote sites. Because of the nature of the tasks, new classes of robotic systems will be required that have a high level of specification for efficiency and reliability. This can only be accomplished through sophisticated intelligent control and efficient sensor integration as an integral part of the design of the robot and the robot's supporting systems.

In this thesis, we present some results of ongoing research in developing intelligent controller for mobile robots. The work presented here focuses on a non-parametric control strategy with an adapt to varying environments, avoid obstacles while moving towards a goal.

1.2 Thesis Organization

This Thesis is organized in 6 different chapters. Following the introducing chapter, Chapter 2 discusses the mobile robotic system which is the result of several past projects at the San Diego State University and in particular some hardware development, a video streaming program and telerobotic user interface.

Chapters 3 to 7 are the result of the authors work on the project. In chapter 3, the IMU (Inertial Measurement Unit) is discussed which is based upon an open source AHRS (Attitude Heading Reference System) code. Chapter 4 discusses a first simple path tracking strategy and Chapter 5 explains the defined protocol for the communication between the SBC (Single Board Computer), controlling the robot and the connection to the web interface, and the Arduino microcontroller board, running the ANFIS code and organizing the sensors. The main contribution to the overall robot project, is the implementation of the ANFIS code on the microcontroller described in Chapter 6. Chapter 7 and 8 show experimental results of the mobile robot system, with the use of the implemented ANFIS controller, and discusses conclusions and future work.

Chapter 2

The Mobile Robotic System

The robotic system combines a robot base with several of the SDSU-developed modules as shown in Figure 2.1. The center of the architecture is a single board computer. This board provides an interface between the web server with its graphical user interface, the microcontroller board with its connected sensors, the webcam and the robot base with the wheels.

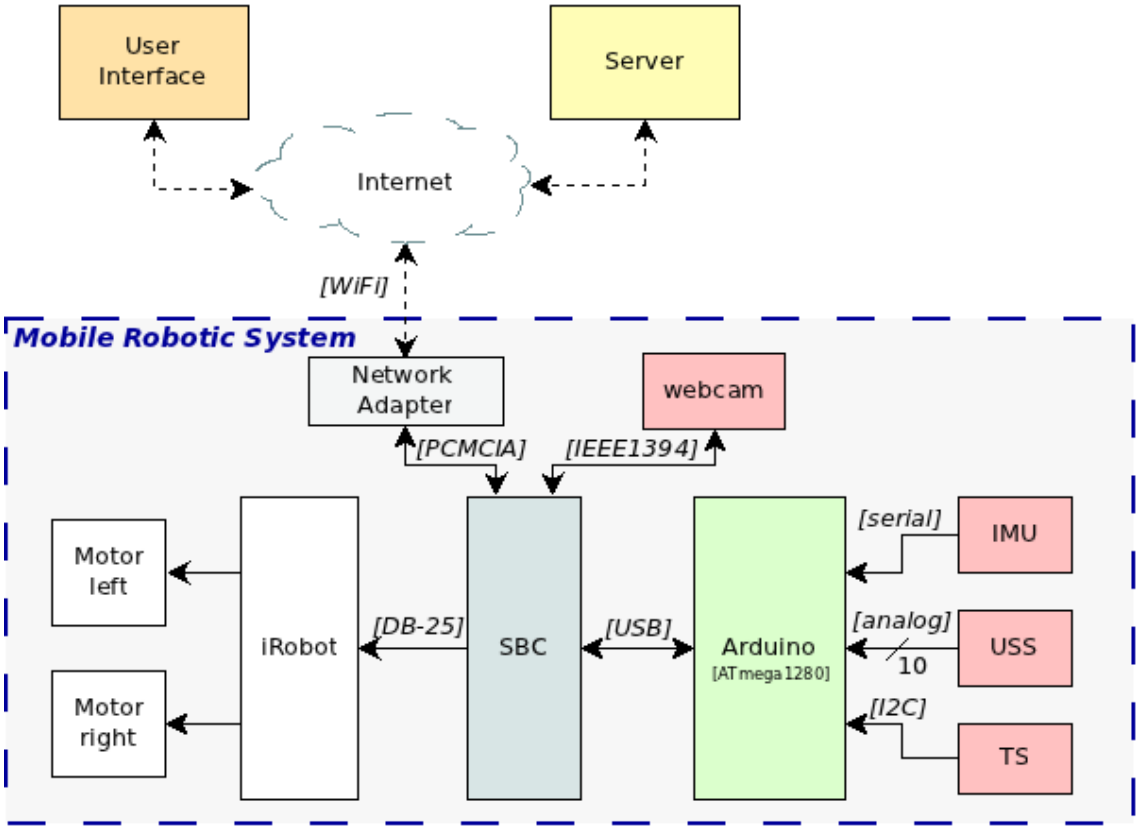

Figure 2.1: System Architecture of the Mobile Robotic System

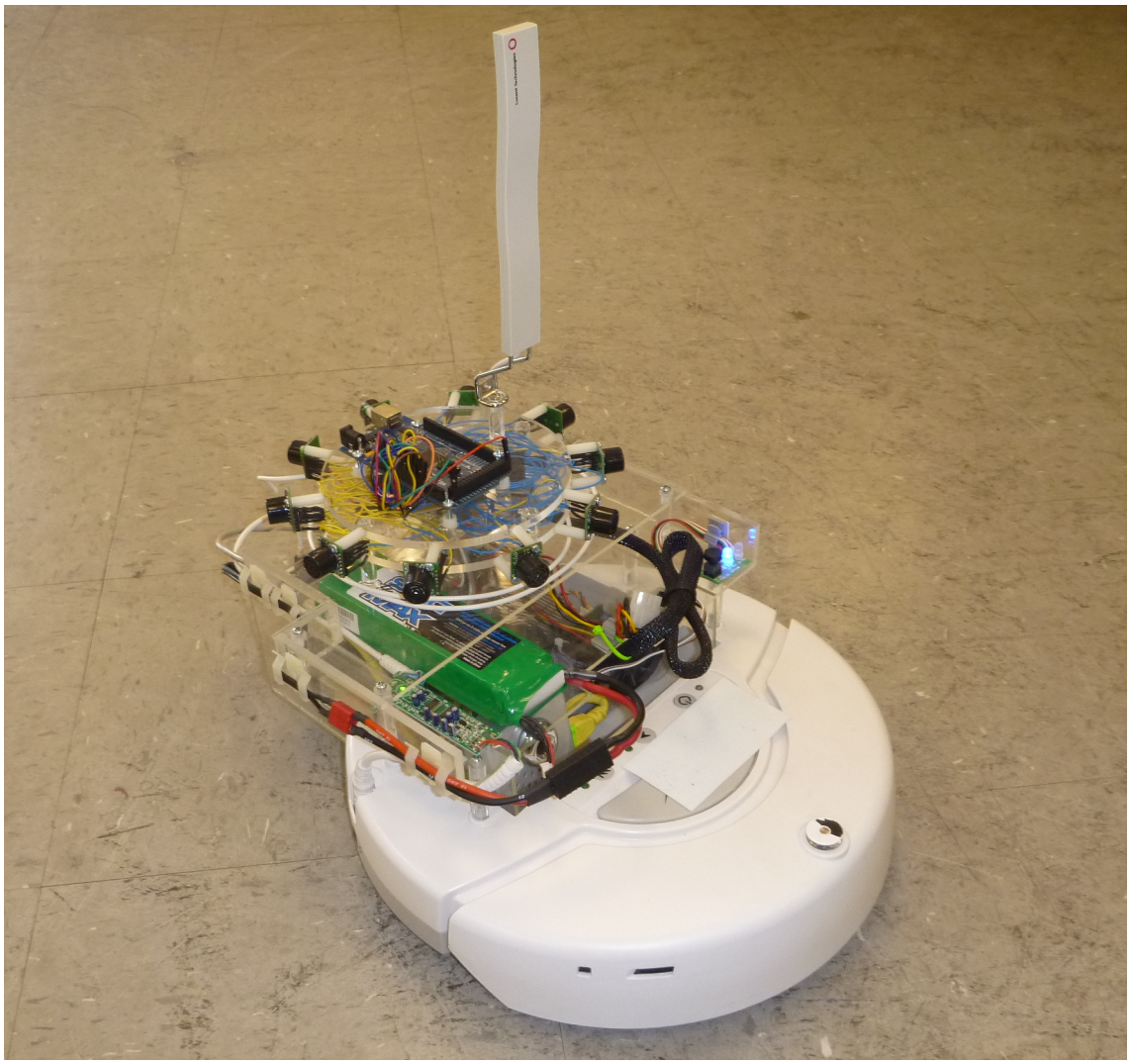

Figure 2.2: The Mobile Robotic System

2.1 iRobot

The robot is based upon customizing an iRobot Create® system as shown in Figure 2.3 [1]. The low cost platform, which is commercially used by the robot company *iRobot Corporation* as an automated vacuum cleaning robot, is designed in a way that it can be easily adopted for hobby use but also research projects. It contains 32 built-in sensors, two powered wheels, a castor, a cargo bay with mounting points and an expandable DB25 input/output port as well as serial interfaces, for the integration of custom sensors and actuators. Besides the SBC, the acrylic case also holds in place an external $12\mathrm{V}4200\mathrm{mAh}\ \mathrm{NiMH}$ rechargeable battery. In a future project, a custom circuit will allow the iRobot to dock and recharge not only the robot's battery, but also the external $12\mathrm{V}\ \mathrm{NiMH}$ battery

through a charging base.

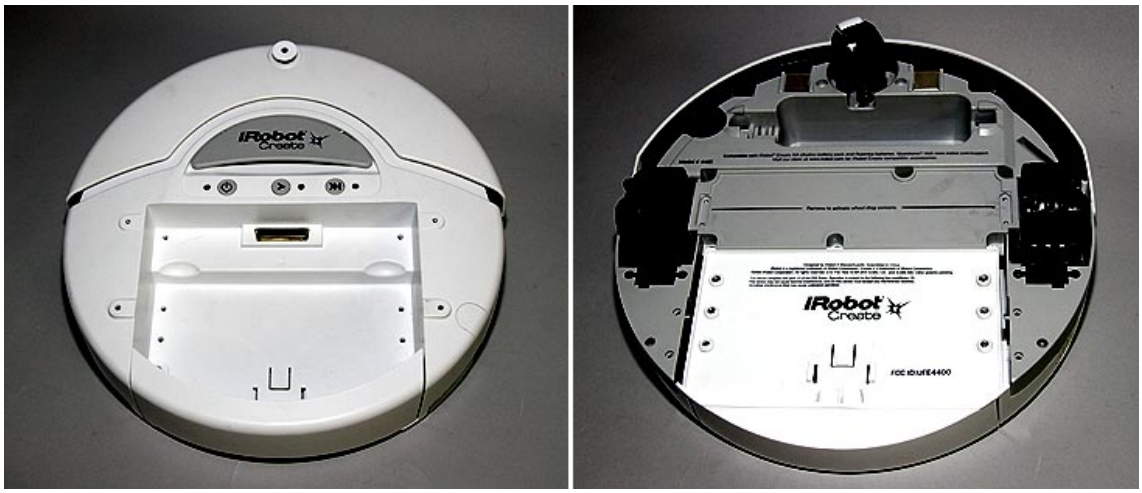

Figure 2.3: Top and Bottom View of the iRobot Create Platform

2.2 Sensors

To let the robot interact with its surrounding, in addition to the iRobot integrated sensors a ultrasonic sensor array has been developed to detect obstacles around the robot and a thermal sensor has been implemented to distinguish between warm and cold objects. Due to the thermal picture, one can distinguish between static, cold objects such as like a chair, and dynamic, mostly warm objects such as a human or an animal. As it is not possible to navigate the robot without knowing its actual position and orientation, we also use an inertial measurement unit (IMU) for this operation.

2.2.1 Webcam

The camera is an Unibarin Fire-i^{TM} digital system which is connected to the SBC via a $400\mathrm{Mbps}$ IEEE1394 (Firewire) interface and supports video streaming at a resolution of $640\mathrm{x}480$ pixels with up to $30\mathrm{fps}$ [2].

2.2.2 Ultrasonic Sensor Array

In order of recognizing obstacles in the robot's environment, an USS (Ultrasonic Sensor Array) was developed [3]. To cover a $360\mathrm{degree}$ field of view, 10 of the MaxBotix LV-EZ1 ultrasonic range finder sensors are used, as shown in Figures 2.4 and 2.5. With this sensor array, we can recognize obstacles in a range of

150mm to 6500mm at a resolution of 25.4mm. To do so, the sensor works as a transceiver. It sends out a high frequency sound wave (42kHz), which is reflected by an obstacle and received again by the sensor. From the elapsed time interval, the sensor can determine the distance to the obstacle. If all the sensors send the same sound wave and wait for an answer, and the signal is reflected in a diffuse way, the sensors would interfere if they all measure at the same time. To solve that problem, the USS works as a so called daisy chain. One sensor after the other takes its measurement at different time intervals. With this technique we achieve a measurement update of the whole array at a frequency of 2Hz.

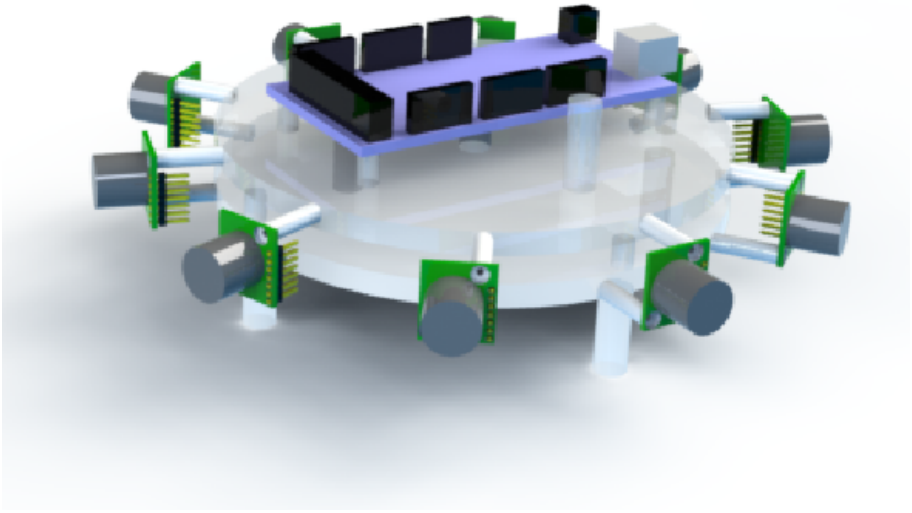

Figure 2.4: Ultra Sonic Sensor Array

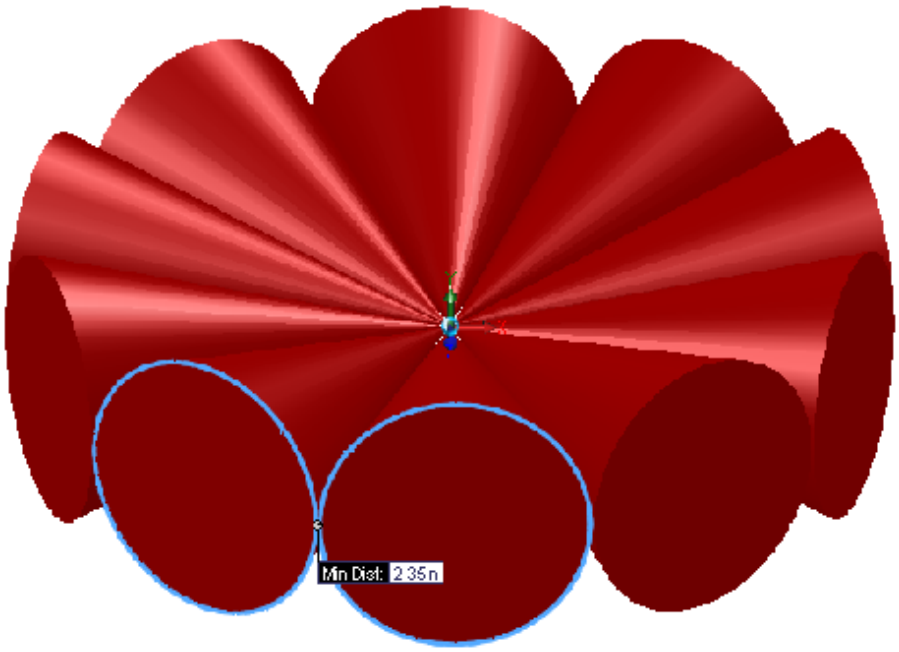

Figure 2.5: Acoustic Cone Pattern of the Ultra Sonic Sensor Array

2.2.3 Thermal Sensor

As dynamic objects are more dangerous to collision, because of their range of movement compared to static objects, it is useful to identify a obstacle as either static or dynamic. As we know most of static objects are 'cold' e.g., a wall, a chair, or a desk whereas many dynamic obstacles such as a human or an animal are warm bodies. Thus we can use a thermal measurement to distinguish between static and dynamic objects. We use a thermal array sensor TPA81, with a spectral response that is typically between $2\mu m$ and $22\mu m$, to categorize the detected objects. This sensor array is mounted on a servomotor to achieve a sensor detection field of $180degrees$ (this array will be replaced with a $360degree$ servomotor as part of the robot's future extensions). A example measurement is shown in Figure 2.6 [3]. We see a $32x8px$ bitmap, according to the 8 adjacent points that the sensor can measure simultaneously, and the 32 different positions of the servomotor. One finds a more detailed description in the technical report of the project [3].

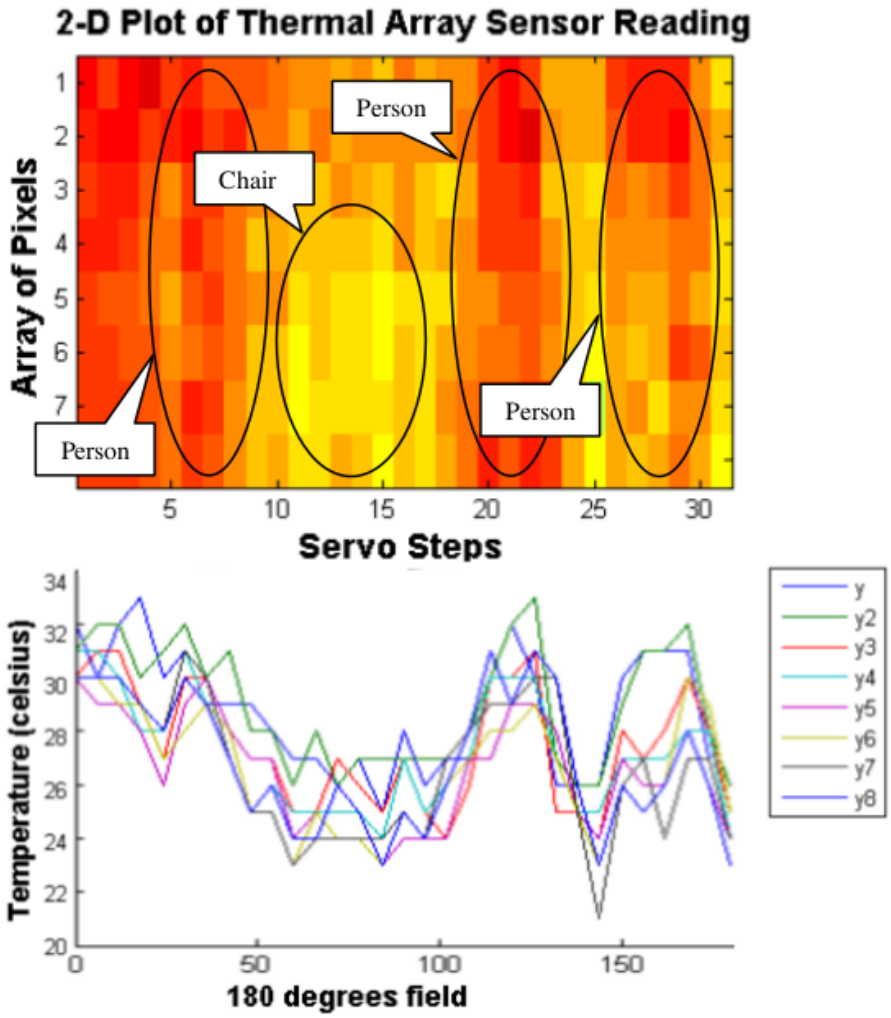

Figure 2.6: 2-D Plot of Thermal Array Sensor Reading

2.2.4 Inertial Measurement Unit

The IMU is used to gather information about the robot's orientation and movement, which is the essential information for path tracking. Chapter 3 provides a detailed description.

2.3 Arduino Platform

An ATmega1280 based microcontroller board by Arduino is used. This board has 54 digital i/o pins, 16 analog inputs, 4 serial ports and a $16\mathrm{MHz}$ crystal oscillator. This board processes all measurements of the inertial measurement unit (IMU), the ultra sonic sensors (USS) and the thermal sensor (TS). Furthermore, it runs the ANFIS-controller, whenever the command line interpreter on the controller

registers a 'pcb2arduino-protocol' request (described in Chapter 5) from the single board computer (SBC).

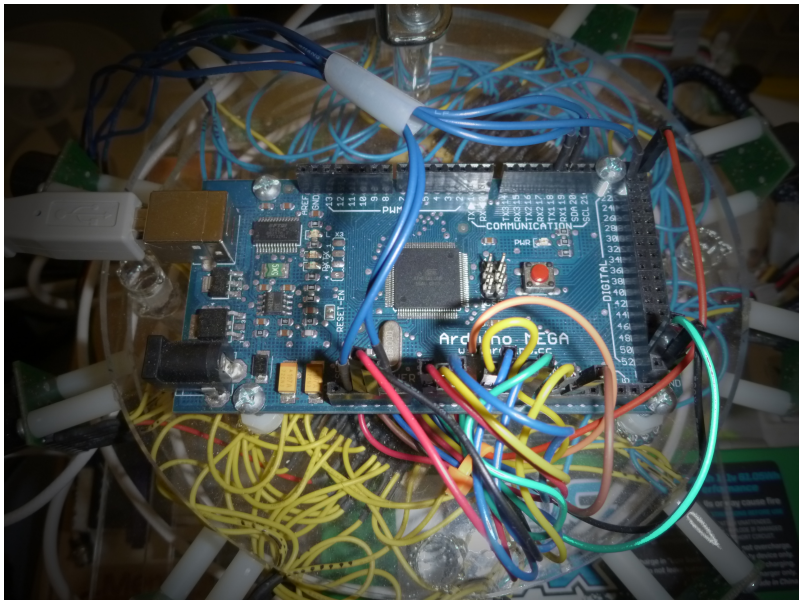

Figure 2.7: Used ATmega1280 Based Microcontroller Board by Arduino

2.4 Single Board Computer

The iRobot platform houses a Migrus C787 DCF-P single board computer with a $1.2\mathrm{GHz}$ Eden ULV Processor. The Migrus C787 has a 16-bit card socket and a flash socket. A Debian derived distribution of the Linux operating system called Voyage Linux, is used as the operating system which is designed to run x86 –platforms. Further a FB-4652 compact flash adapter board is used for memory. The connection to the web server is achieved with a Proxima 802.11g PCMCIA adapter card with external 5db gain antenna.

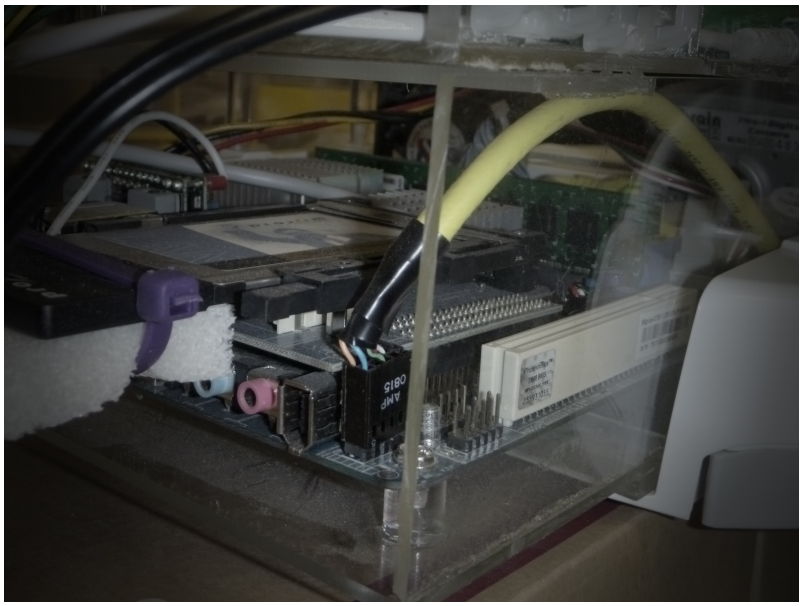

Figure 2.8: Used Migrus C787 DCF-P Single Board Computer

2.4.1 Telerobotic System

The robot can be controlled in telerobotic mode via a graphical user interface on a web browser. The interface, shown in Figure 2.9, contains a target area in which the user can drive the robot by positioning the cursor. The user also sees a live flash video stream from the webcam on the mobile robot system. This stream is adaptive and dynamically modifies either the frame rate, or the quality scale, such that the data rate required for streaming, depends on the data rate available to the robot's wireless transceiver, to guarantee a satisfying live stream. This infrastructure was developed in the course of a Master Thesis at the San Diego State University by Sudha Narajan [2, 4]. Furthermore, all relevant sensor values from the robot and a certainty grid (Figure 2.10), which shows the probability of recognized obstacles in the robot's environment measured by the USS array and the thermal sensor are displayed.

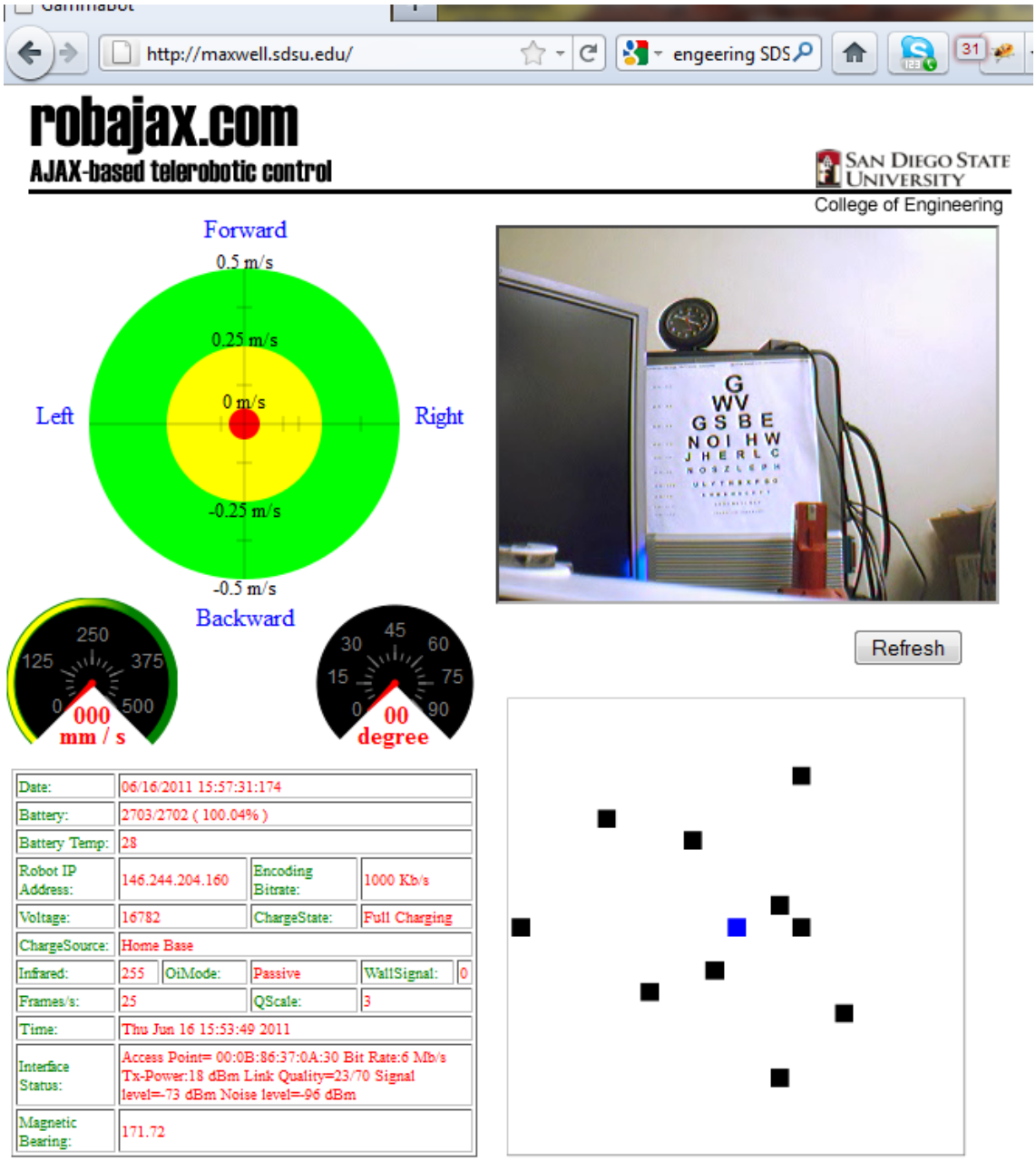

Figure 2.9: Simple AJAX Based Web Interface for Telerobotic Control of the Mobile Robot System

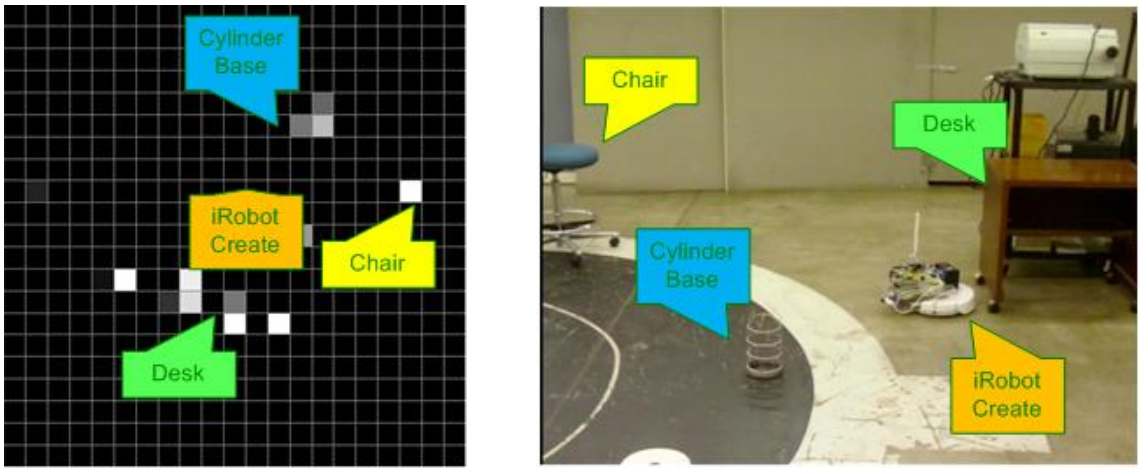

Figure 2.10: Certainty Grid Compared With Real Environment

2.4.2 Path Planning

To achieve real-time obstacle avoidance using the data from the ultrasonic sensor array, the Vector Field Histogram and Virtual Force Field methods were implemented in one of the projects for the mobile robotic system [5]. Simulations, as in Figure 2.11, show the success of the project.

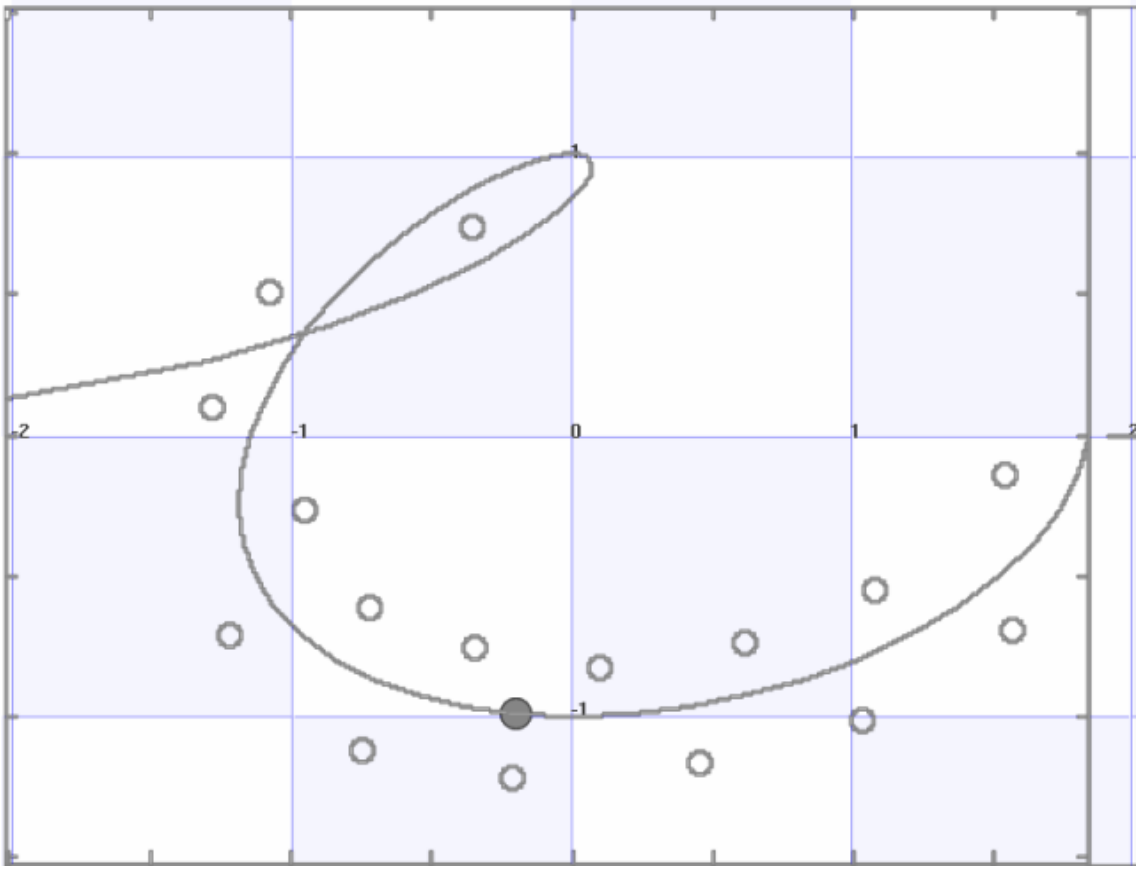

Figure 2.11: Path Planning to Avoid Obstacles

Chapter 3

Inertial Measurement Unit

We use a 9 degree of freedom IMU from sparkfun [6]. We measure acceleration, rotation and magnetic orientation of the sensors in all translational and rotational axises. All measurements are gathered by a on-board ATmega328 microcontroller that turns the sensor board into a AHRS and sends a data string with the sensor information via a serial port to the Arduino board on the robot.

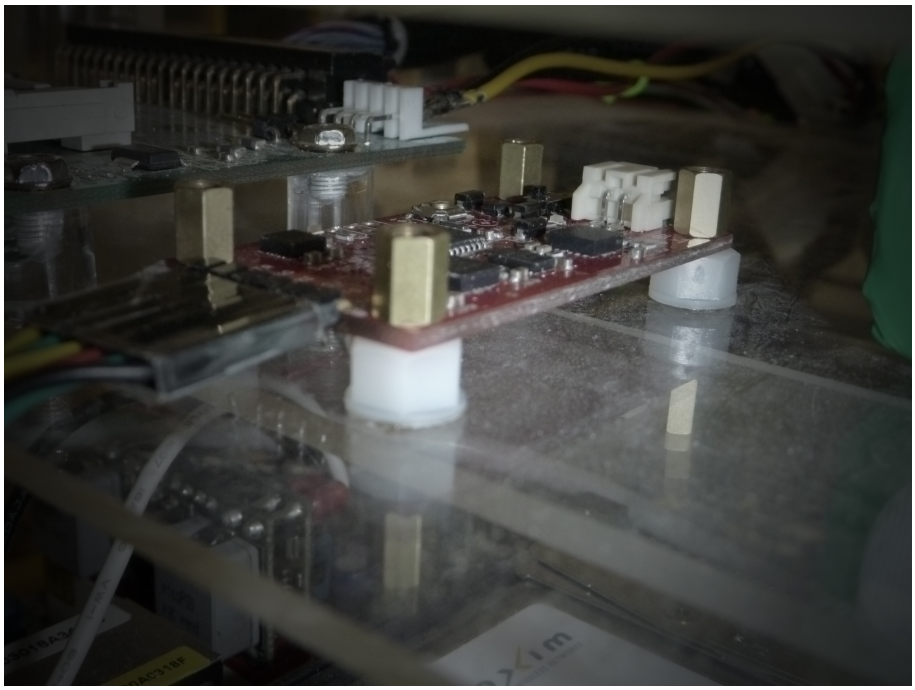

Figure 3.1: 9 Degree of Freedom IMU from Sparkfun

3.1 IMU Sensors

The sensor board contains a triple-axis accelerometer ADXL345, a single-axis gyro LY530ALH (yaw), a dual-axis gyro LPR530ALH (pitch and roll) and a triple-

axis magnetic compass HMC5843. With the used settings, the most significant characteristics of the sensors, are shown in the Tables 3.1, 3.2 and 3.3.

Communication	I^2C	
Measurement Range	± 2	g
Resolution	3.9	$^{mg}/_{\mathrm{LSB}}$
Measurement Rate	200	Hz

Table 3.1: Characteristics of the Acceleration Sensor ADXL345

Communication	analog	
Measurement Range	± 300	$^{\circ}/_{\mathrm{s}}$
Resolution	3	$\frac{\mathrm{mV}}{^{\circ}/_{\mathrm{s}}}$
Measurement Rate	100	Hz

Table 3.2: Characteristics of the Gyro Sensors LY530ALH and LPR530ALH

Communication	I^2C	
Measurement Range	± 4	gauss
Resolution	7	mgauss
Measurement Rate	50	Hz

Table 3.3: Characteristics of the Magnetic Digital Compass HMC5843

3.2 AHRS Code

The Open Source code by Jordi Munoz [7], turns the sensor board into a Direction Cosine Matrix (DCM) based Attitude Heading Reference System (AHRS) with gyro drift correction based on accelerometer (gravity) vector and magnetometer (compass) vector.
This code was optimized by setting the measurement rate of the sensors to a maximal frequency according to Tables 3.1, 3.2 and 3.3. The on-board microcontroller then calculates the average values, over the elapsing time between the requests of the ANFIS microcontroller board, and sends them to the ANFIS board on the robot. With these average values over e.g. 5/10/20 measurements (at a request frequency of $10\mathrm{Hz}$ of the ANFIS board), measurement noise can be filtered in the first step. An advanced noise filter could be reached by using an Kalman

filter as described by Tom Pycke on [8]. However, this advanced version is not implemented yet, as the IMU was not the focus of this thesis.

```
"[angle_x] [_y] [_z] [gyro_x] [_y] [_z] [accel_x] [_y] [_z] [magn_heading] [s] [v]"
```

Figure 3.2: Answer String of the IMU on a ANFIS-Board Request

The IMU board communicates with the ANFIS board via a serial port. The data transfer is unidirectional and all values are sent in a character string of 12 values as described in Figure 3.2, where the magnetic heading is calculated from the sensor values in *latitude* and *longitude* as in Equation (3.1). For the definitions of directions and angles, see Chapter 4.1.

$$\texttt{magn\_heading} = \pi - \arctan -\frac{\texttt{MAG}_\texttt{Y}}{\texttt{MAG}_\texttt{X}} \tag{3.1}$$

where $\texttt{MAG}_\texttt{x}$ and $\texttt{MAG}_\texttt{y}$ are the tilt compensated values, $magn$ is the magnetic measurements and $pitch$ and $roll$ are the calculated tilt bearings:

$$\begin{aligned}\texttt{MAG}_\texttt{X} &= magn_x \cdot \cos(pitch) + magn_y \cdot \sin(roll) \cdot \sin(pitch) + magn_z \cdot \cos(roll) \cdot \sin(pitch) \\ \texttt{MAG}_\texttt{Y} &= magn_y \cdot \cos(roll) - magn_z \cdot \sin(roll)\end{aligned} \tag{3.2}$$

The s and v values of the IMU string are simply the first and second derivatives of the acceleration in the x direction, which is a straight movement of the iRobot. As there is no tilt and Coriolis correction of the calculation of position and velocity currently implemented, these values are not very useful, because each small tilt in the calibration surface causes a variation in gravity measured by the acceleration sensor in the x direction. The gravity acceleration is fairly big compared to the robot's movement. So any influence on the sensor in x direction brings an unacceptable error in the calculations of position and velocity.

Chapter 4

Path Tracking

4.1 Definitions of the Coordinate Systems

To avoid misinterpretation and confusion about values and directions used in this thesis, a global coordinate system for the mobile robotic system was defined as shown in Figure 4.1. A forward movement of the robot means a movement in the x direction. The relations to the other translational and rotational vectors are defined in the common way of the right-hand rule, except the z vector *facing* the earth, to create a positive value for the gravity.
The bearing of the robot relative to the magnetic north pole is defined with the mathematical convention as a counter clockwise rotation meaning a positive value.
The last definition concerns the position of the robot. To work with a global coordinate path leading to a target, it is necessary to describe this path in a defined coordinate system. The coordinate system is always north oriented, where the x value describes the direction `west` to `east` and the y value describes the `south` to `north` direction. A point $[0/0]$ refers to the origin of the path.
Unless mentioned differently, all values are given in millimeter mm and radians rad.

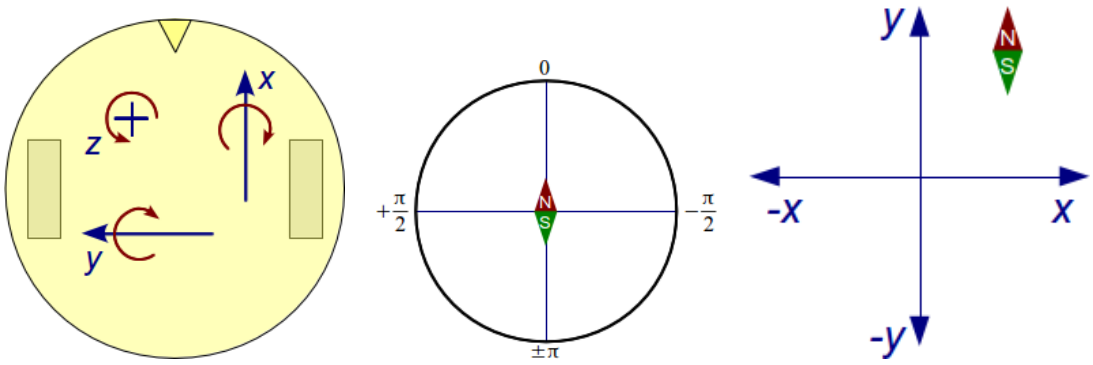

Figure 4.1: Definition of (left) Translational and Rotational Directions in the System, (middle) the Robot's Bearing and (right) the Coordinates for Global Position of the Robot

4.2 Position Dead Reckoning

To have the robot track a path, it is essential to know where the robot's actual position is. Given the current position, one may then calculate the desired next path coordinates. To do so, we can use different strategies as described in [9, 10]. According to the fact that our system is used in buildings, we confine our system to inertial position dead reckoning, as GPS signals may not be reliable enough.

4.2.1 Using IMU Data

We can use the values sent from the IMU to calculate the robot's position. By integration of the acceleration measurement in the x direction we can calculate velocity and the covered distance of the robot as in (4.1). However, as mentioned in Chapter 3.1, a simple integration of the acceleration measurement in x direction is not useful, due to the missing error compensation of the AHRS.

$$\begin{aligned} a_t &= a_x \\ v_t &= \int a_t = a_x \cdot t + v_{t-1} \\ s_t &= \int v_t = a_x \cdot t^2 + v_{t-1} \cdot t + s_{t-1} \end{aligned} \tag{4.1}$$

4.2.2 Using Desired Data

Currently the position dead reckoning is done by using the input (desired change in bearing over time) and the given velocity from the last ANFIS computation, and

the elapsed time since then:

$$r = \frac{v_{M\texttt{old}}}{\omega_{\texttt{old}}} \tag{4.2}$$

Where ω_{old} is the change in bearing over time and $v_{M\texttt{old}}$ is the given or 'middle' velocity of the robot, which are the values of the last ANFIS request. We can then use the radius r, the robot's velocity v_M and the time t to estimate the robot's new position as in Figure 4.2:

$$\alpha = \omega_{old} \cdot \Delta t \tag{4.3}$$

$$\begin{aligned} x_r &= r \cdot \sin(\alpha) \\ y_r &= r \cdot (1 - \cos(\alpha)) \end{aligned} \tag{4.4}$$

$$\begin{aligned} v &= \sqrt{(x_r^2 + y_r^2)} \\ \beta &= \arctan\left(\frac{y_r}{x_r}\right) \end{aligned} \tag{4.5}$$

$$\begin{aligned} x \mathrel{+}= v \cdot \cos\left(\frac{\pi}{2} + \gamma + \beta\right) \\ y \mathrel{+}= v \cdot \sin\left(\frac{\pi}{2} + \gamma + \beta\right) \end{aligned} \tag{4.6}$$

where γ is the actual bearing of the robot, reported by the IMU as the yaw value. Also note, that due to our definition, counter clockwise angles have a negative value.

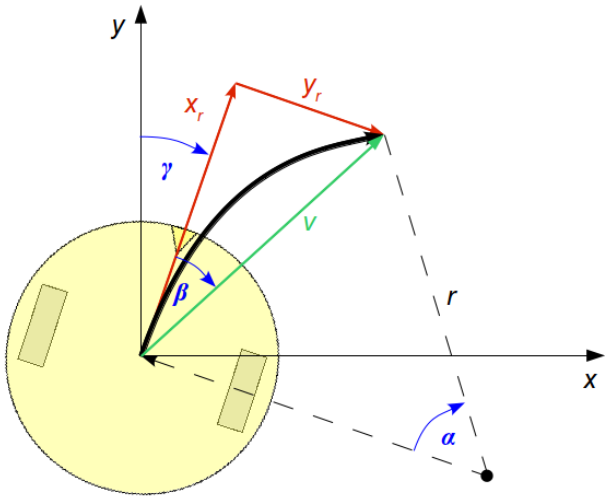

Figure 4.2: Position Dead Reckoning

4.3 Tracking Strategy

As we are very limited with the memory on the microcontroller and the ANFIS system is designed to be as flexible as possible, the path is not stored on the microcontroller. Instead the SBC, which also runs the path planning algorithm, just the path coordinates are reported to the ANFIS controller, whenever the controller requests a new coordinate.
For now, a very simple tracking strategy with 3 coordinates is implemented, as shown in Figure 4.3. However this method can be improved in a future project to achieve a smoother path tracking.

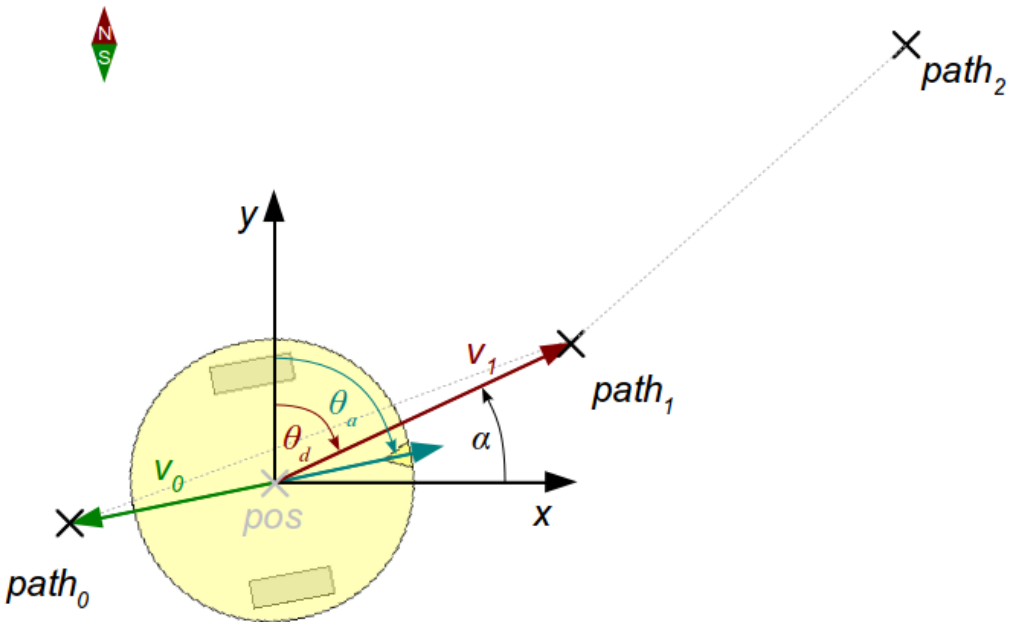

Figure 4.3: Path Tracking Strategy

The ANFIS microcontroller stores 3 coordinates: $path_0$ is the last coordinate, $path_1$ is the current target and $path_2$ is the next coordinate. Further v_0 and v_1 are the vectors from the current estimated position pos to the coordinates $path_0$ and $path_1$, respectively Θ_a is the actual bearing of the robot according to magnetic north, as reported by the IMU. The bearing of v_1 is the desired bearing for the next step $\Theta_d = -\frac{\pi}{2} + \alpha$ and is used as the ANFIS input, if autonomous mode is chosen.
As soon as the robot reaches a threshold distance from the $path_1$ coordinate and v_1 is smaller than v_0, the ANFIS microcontroller requests a new coordinate from the SBC. When the new path coordinate is received, the coordinates are shifted as:

$$path_0 \leftarrow path_1$$
$$path_1 \leftarrow path_2$$
$$path_2 \leftarrow path_{new}$$

Chapter 5

Communication Protocol between SBC and Arduino

The developed protocol is flexible designed so that it can be easily adopted for other projects. One command from the SBC consists of the command mnemonic, the length of the data block and the data block itself.

[mnemonic]	[length *n*]	[data]
1 Byte	1 Byte	*n* Byte

Figure 5.1: Structure for a Request from the SBC to the Arduino

Packages are processed if the length-Byte (amount of Bytes in data block) is equal to amount of received data bytes. Furthermore, a # indicates messages from ANFIS board to the SBC, whereas a $ indicates requests from the ANFIS board to the SBC. Commands from the SBC to the ANFIS board do not have a special start character and have a structure like shown in Figure 5.1. Note that a request from the *SBC to the microcontroller* does not have any determiner, whereas the determiner between values of the answer string of the *microcontroller to the SBC* is a "␣" (whitespace). The current program version uses the protocol of Table 5.2 and the definition of values as in Table 5.1.

variable	value	description
		SBC to Arduino
[vm]	$0..+500$	$\mathrm{mm/s}$ as 2 Byte Integer big-endian;
[Ot]	$-179..+179$	kompass bearing in $\mathrm{degrees}$ as 2 Byte Integer big-endian;
[x]/[y]		mm as 2 Byte Integer big-endian; xy-coordinate relativ to point of origin (reset with 'set path origin' command);
[para_set]	$1, 2$	set number as 1 Byte Integer;
[mode]	$1, 2, 3$	mode number as 1 Byte Integer;
		Arduino to SBC (all sent as char strings with space seperator)
[vr]	$-500..+500$	int value in $\mathrm{mm/s}$
[vl]	$-500..+500$	int value in $\mathrm{mm/s}$
raw data		sensor value as integer
inches data		converted sensor value as float
cm data		converted sensor value as float
[angle_x]	$-\pi..+\pi$	roll; float value in $\mathrm{radiant}$
[angle_y]	$-\pi/2..+\pi/2$	pitch; float value in $\mathrm{radiant}$
[angle_z]	$-\pi..+\pi$	yaw; float value in $\mathrm{radiant}$
[gyro_x/y/z]	$-300..+300$	float value in $\mathrm{rad/s}$
[accel_x/y/z]	$-2\mathrm{g}..+2\mathrm{g}$	float value in $\mathrm{m/s^2}$
[magn heading]	$0..2\pi$	float value in $\mathrm{radiant}$
[s]		float in mm
[v]		float in $\mathrm{mm/s}$
[pos_x]/[pos_y]		int value in mm

Table 5.1: Description of Protocol Values

command	**mnemonic**	**data**	**answer**
Print Data			
print raw data	'1'		"[uss_0]...[uss_9]"
print inches data	'2'		"[uss_0 · INCH_PER_VOLT] ...[uss_9 · INCH_PER_VOLT]"
print cm data	'3'		"[uss_0 · CM_PER_VOLT] ...[uss_9 · CM_PER_VOLT]"
print IMU data	'4'		"[angle_x][_y][_z] [gyro_x][_y][_z] [accel_x][_y][_z] [magn_heading][s][v]"
print cm & IMU data	'5'		"[output_like_'3'] [output_like_'4']"
ANFIS commands			
tele-robotics mode	't'		"#ACK"
autonomous mode	'a'		"#ACK"
anfis update	'u'	[vm] [Ot]	"[vl][vr][pos_x][pos_y][RMSE]"
extended anfis update	'U'	[vm] [Ot]	"[output_like_'u'] [output_like_'3'] [output_like_'4']"
learning mode	'm'	[mode]	"#ACK"
set up new anfis	'n'	[para_set_index]	"#ACK"
save para to EEPROM	's'		"#ACK"
print parameter	'0'		list of all parameters as string
set step size	'z'	[stepsize] [inc_rate] [dec_rate]	"#ACK"
Path Tracking			
set path origin	'o'		"#ACK"
get path coordinate	'p'	[x] [y]	"#ACK"
Miscellaneous			
reset IMU	'r'		"#ACK"
debug on/off	'd'		"#ACK"

Table 5.2: List of Commands from SBC to Arduino

5.1 Single Board Computer to Microcontroller

Examples of requests follow and show the sent byte values (they do not have any determiner in between). Note that the first byte is the mnemonic and is sent as the *ASCII-value* of the command.

5.1.1 Print Data Commands

print raw data

Mnemonic:	`'1' or ASCII 49`
Description:	Prints the raw values of the USS array, as they are read from the sensors.
Example Request:	`49 0`
Example Respond:	`"38 30 105 32 29 118 53 59 52 43"`

print inches data

Mnemonic:	`'2' or ASCII 50`
Description:	Prints the distance measurements of the USS array in *[inch]*.
Example Request:	`50 0`
Example Respond:	`"18.93 14.94 52.31 15.94 14.44 58.79 26.40 28.89 25.90 21.42"`

print cm data

Mnemonic:	`'3' or ASCII 51`
Description:	Prints the distance measurements of the USS array in *[cm]*.
Example Request:	`51 0`
Example Respond:	`"48.09 37.97 132.88 40.50 36.70 149.33 67.07 73.40 65.81 54.42"`

print IMU data

Mnemonic:	`'4' or ASCII 52`
Description:	Prints the IMU measurements as shown in table 5.2 with the 3 calculated AHRS-angles, the raw 3 gyro and 3 acceleration measurements and the magn_heading, which already includes tilt compensation in roll and pitch
Example Request:	`52 0`
Example Respond:	`"0.000 0.002 3.006 0.000 0.000 0.016 0.000 -0.040 9.810 3.017 2.041 0.059"`

print cm & IMU raw data

Mnemonic:	`'5' or ASCII 53`
Description:	Prints the measurements of the USS array in *[cm]* and also the IMU data. So the SBC just needs 1 request to gather all the information.
Example Request:	`53 0`
Example Respond:	`48.09 37.97 132.88 40.50 36.70 149.33 67.07 73.40 65.81 54.42 0.000 0.002 3.006 0.000 0.000 0.016 0.000 -0.040 9.810 3.017 2.041 0.059`

5.1.2 ANFIS Commands

telerobotics mode

Mnemonic:	`'t' or ASCII 116`
Description:	Turns the ANFIS program to *telerobotic mode*. So the ANFIS uses the reported bearing Θ and velocity v from the graphical user interface for it is velocity computations.
Example Request:	`116 0`
Example Respond:	`"#ACK"`

autonomous mode

Mnemonic:	`'a' or ASCII 97`
Description:	Turns the ANFIS program to *autonomous mode*. The desired change of bearing is then calculated via path tracking strategy and solely the velocity v from the GUI is used as the robot's middle velocity. If no velocity is sent with the 'anfis update' command, the ANFIS controller uses the most recent given velocity.
Example Request:	`97 0`
Example Respond:	`"#ACK"`

anfis update

Mnemonic:	`'u' or ASCII 117`
Description:	The ANFIS uses the reported bearing Θ and/or velocity v to calculate the new velocities, and reports them with the new calculated robot's position and the last RMSE. It is optional to send a bearing and/or velocity with the request.
Example Request:	`117 4 0 45 1 44`
Example Respond:	`"200 -352 120 1115 0.002"`

extended anfis update

Mnemonic:	`'U' or ASCII 85`
Description:	Same function as 'anfis update', but reports additionally the USS array and IMU values, to use just 1 request instead of 'u','3' and '4'
Example Request:	`85 4 0 45 1 44`
Example Respond:	`200 -352 120 1115 0.002 48.09 37.97 132.88 40.50 36.70 149.33 67.07 73.40 65.81 54.42 0.000 0.002 3.006 0.000 0.000 0.016 0.000 -0.040 9.810 3.017 2.041 0.059`

learning mode

Mnemonic:	`'m' or ASCII 109`
Description:	Set the learning mode of the ANFIS. Possible modes are: *(0)* $\rightarrow$ no learning, *(1)* $\rightarrow$ update premise parameters in layer 1, *(2)* update consequent parameters in layer 4, *(3)* update all parameters in layer 1 and 4.
Example Request:	`109 1 3`
Example Respond:	`"#ACK"`

set up new anfis

Mnemonic:	`'n' or ASCII 110`
Description:	Rebuild ANFIS architecture and load the requested parameter set from the EEPROM.
Example Request:	`110 1 6`
Example Respond:	`"#ACK"`

save para to EEPROM

Mnemonic:	`'s' or ASCII 115`
Description:	Save actual parameter set to EEPROM at given set position.
Example Request:	`115 1 10`
Example Respond:	`"#ACK"`

print parameter

Mnemonic:	`'0' or ASCII 48`
Description:	Print the actual parameter set.
Example Request:	`48 0`
Example Respond:	`"3.810000 2.000000 -3.810000 3.810000 2.000000 3.810000 0 0 0 0 "`

set step size

Mnemonic:	`'z' or ASCII 122`
Description:	Set step size and update factors.
Example Request:	`122 3 10 90 110`
Example Respond:	`"#ACK"`

5.1.3 Path Tracking Commands

set path origin

Mnemonic:	`'o' or ASCII 111`
Description:	Reset robot's position to *[0/0]*
Example Request:	`111 0`
Example Respond:	`"#ACK"`

get path coordinate

Mnemonic:	`'p' or ASCII 112`
Description:	Sent new path coordinate for the path tracking strategy. Should be sent after each *"#NP"* request of the microcontroller board.
Example Request:	`112 4 0 100 1 50`
Example Respond:	`"#ACK"`

5.1.4 Miscellaneous

reset IMU

Mnemonic:	`'r' or ASCII 114`
Description:	Send a reset signal to the IMU board. IMU request takes about 6 seconds, until the Arduino board responds with acknowledgement.
Example Request:	`114 0`
Example Respond:	`"#ACK"`

debug on/off

Mnemonic:	`'d' or ASCII 100`
Description:	Switch debug mode on/off. In debug mode, the ANFIS board reports more information about the ANFIS update etc.
Example Request:	`100 0`
Example Respond:	`"#ACK"`

5.2 Microcontroller to Single Board Computer

As the microcontroller side is very passive in the communication, there is just one real request "`$NP`" from the ANFIS board to the SBC. All the other messages

are either responding answers to requests from the SBC, or just a message if the request was acknowledged or an error occurred (Table 5.3). The request for the next path point is sent whenever the path tracking strategy needs a new coordinate.

request	**comment**
`"$NP"`	request next path point
`"#ERR"`	error with request
`"#ACK"`	request acknowledged

Table 5.3: Messages from Arduino to SBC

Chapter 6

ANFIS Controller

6.1 How an ANFIS Controller Works

An ANFIS is a combination of the two traditional intelligent control algorithms, *Fuzzy Inference Systems* and *Adaptive Neural Networks*. The Fuzzy System's strength is the possibility to implement human expertise without describing it with a mathematical model. Instead the expert's knowledge is formulated in simple linguistic expressions as control rules. What still has to be done, though, is to shape the membership functions, which weight the rules. As the form of the membership functions has an drastic effect on the controllers characteristic, it is usually hard to set them in an optimal way; trial and error is typically done.
The Neural Network has its strength in the autonomous fine tuning of the system itself. However, it is a very complex network and is more or less just a black box for the user. The ANFIS combines the advantages of both systems and is a powerful and widely used tool in the field of control engineering. As a result, we have a system with the transparency of a Fuzzy System, but an autonomous tuning capability of the membership functions through a Neural Network [11].

6.1.1 Fuzzy Inference System

Basically a Fuzzy Inference System functions as shown in Figure 6.1. The steps of a fuzzy inference are:

1. Compare the input variables with the membership functions to obtain membership values of each linguistic label (fuzzification)

2. Combine the membership values (usually multiplication or min.) to get a *'firing strength'* of each rule and generate accordingly the output of each rule (rule evaluation)

3. Combine the outputs of all rules to achieve a crisp output (defuzzification)

For a more detailed description and the evaluation of several types of fuzzy reasoning have a look at [12] and [13].

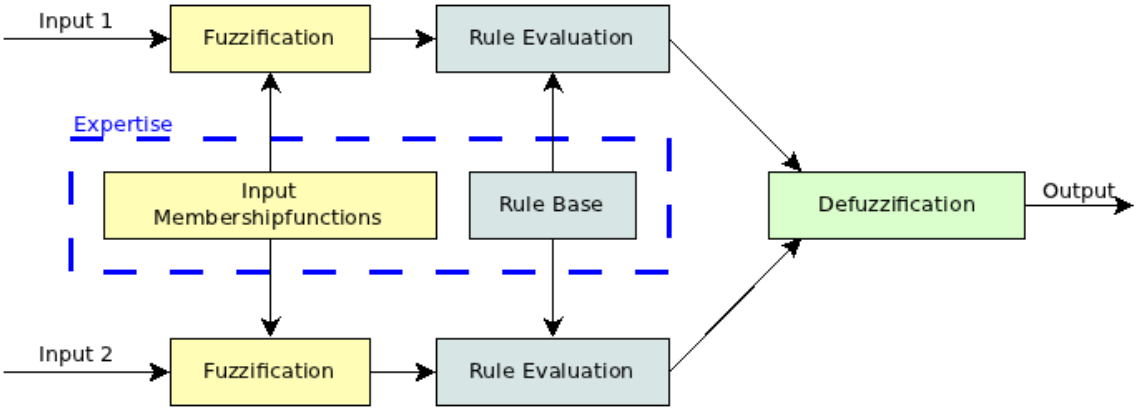

Figure 6.1: Fuzzy Inference System

6.1.2 Adaptive Neural Network

An adaptive neural network is a multilayer feed-forward network as shown in Figure 6.2. It consists of an input layer, one or more hidden layers and an output layer. It functions principally as the human brain. Each circle symbolizes a neuron with adaptive parameters and each arrow a connection between neurons. Each of this nodes has a node function to combine the inputs and handles the outputs with different weights. If the system learns, it simply changes the weights of the connections in a way to minimize an overall error between desired and actual signals. This learning algorithms are discussed in Chapter 6.2 and can be adopted for the ANFIS. The number of nodes in the hidden layers is arbitrary. However, more nodes make the system more complex; hence computation is more, and not necessarily better. A detailed description is given in [14].

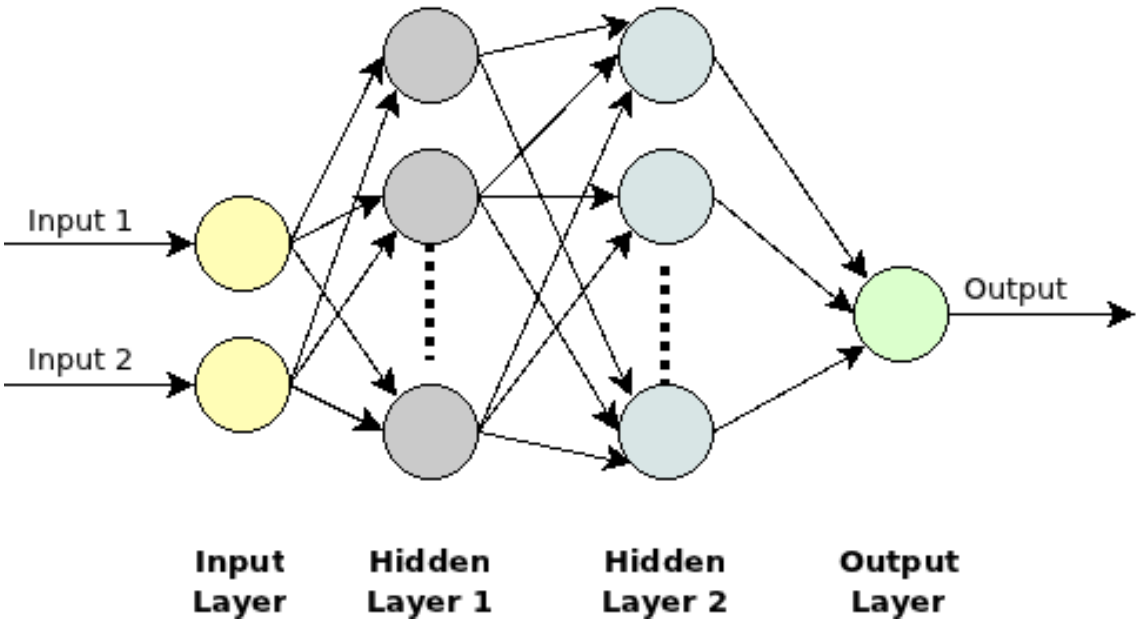

Figure 6.2: Adaptive Neural Network

6.1.3 Adaptive Neural Fuzzy Inference System

As mentioned before, an ANFIS is a combination of the two common intelligent control algorithms: a *Fuzzy Inference System* and an *Adaptive Neural Network*. They are combined in the way that we use the semantic of an Fuzzy Inference System, but describe it as a Neural Network with its node functions, weights and connections (Figure 6.4). A square node (Layer 1 and 4) means it has adaptive parameters, whereas a circle node just consists of a node function. Table 6.1 shows the description of the 2-input-ANFIS layers by Jang [11].

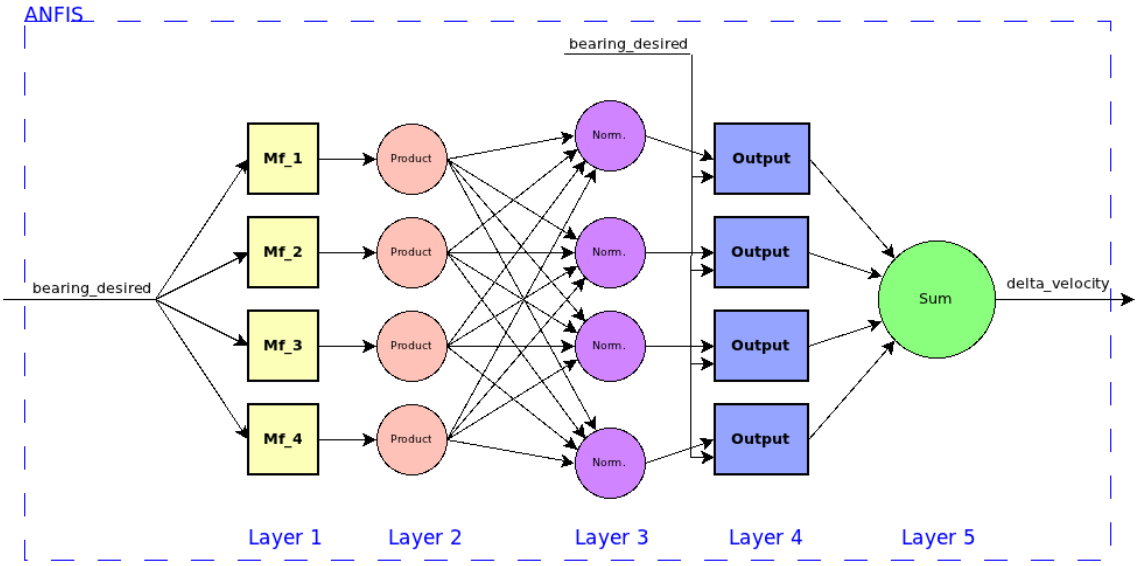

Figure 6.3: Single Input Single Output ANFIS

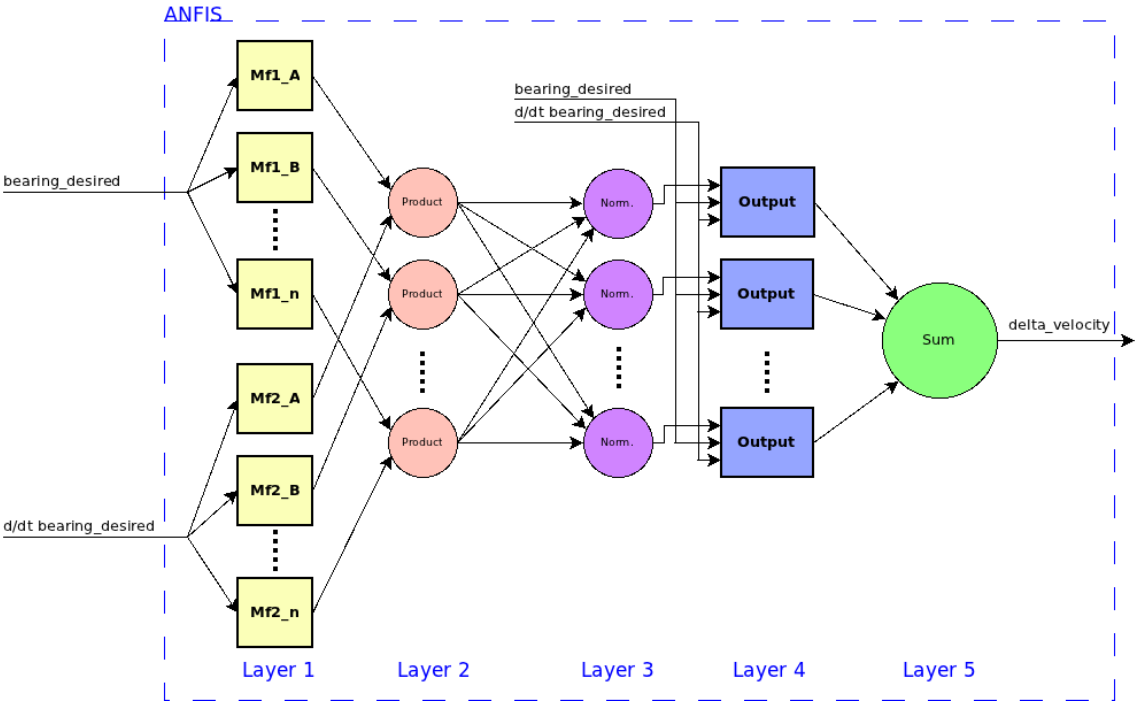

Figure 6.4: Multi Input Single Output ANFIS

Layer	Node Function	Description
1	$O_i = \mu_{A_i}(x)$	**Membership Functions:** A_i is the linguistic label A for node i. In other words it is the *membership function* μ and specifies the degree to which the given input x satisfies the quantifier A. Usually we choose μ_A to be a bell-shaped function with maximum equal to 1 and minimum equal to 0, such as $$\mu_{A_i}(x) = \frac{1}{1 + \left(\frac{x-c_i}{a_i}\right)^2 \cdot b_i}$$ or $$\mu_{A_i}(x) = exp\left(-\left(\frac{x-c_i}{a_i}\right)^2\right)$$ where $\{a_i, b_i, c_i\}$ are *adaptive parameters*. Accordingly the shape of the membership functions change throughout learning. Parameters in this layer are referred to as *premise* parameters.
2	$\omega_i = \Pi\ \mu_{j_i}\ (j = A, B..n)$	**Rule Fulfillment:** Every node in this layer multiplies the incoming signals from the membership function nodes in layer 1 and sens the *product* out. Each node output represents the firing strength of a rule.
3	$\bar{\omega_i} = \frac{\omega_i}{\omega_1 + \omega_2}$, $(i = 1, 2)$	**Sum of Rules:** The *i* th node in the layer calculates the ratio of the *i* th rule's firing strengths. The firing strength gets *normalized*.
4	$O_i = \bar{\omega}_i f_i = \bar{\omega}_i\ (p_i x + q_i y + r_i)$	**Rule Output:** The nodes in this layer contain among the node function the *adaptive parameters* $\{p_i, q_i, r_i\}$. Parameters in this layer will be referred to as *consequent parameters*.
5	$O_i = \Sigma\bar{\omega}_i f_i = \frac{\Sigma\omega_i f_i}{\Sigma\omega_i}$	**Over All Output:** The single node in this layer computes the *overall output* as the summation of all incoming signals.

Table 6.1: Layer Description of an ANFIS

The complexity of the an ANFIS structure is shown in Table 6.2, where In_n is the number of inputs, $Rule_n$ is the number of rules and $Node_n$ is the number of nodes. $Rule_n$ and $Node_n$ can be calculated from the following formulas:

$$\begin{aligned} Rule_n &= Mf_n^{\,In_n} \\ Node_n &= In_n + In_n \cdot Mf_n + 3 \cdot Rule_n + 1 \end{aligned} \tag{6.1}$$

Further the amount of adoptable parameters in layers 1 and 4 is:

$$parameter_n = 3 \cdot In_n \cdot Mf_n + (In_n + 1) \cdot Rule_n + 1 \tag{6.2}$$

Layer	Name	#Nodes	#Parameters
0	Input	In_n	-
1	Membership Functions	$In_n \cdot Mf_n$	3
2	Rule Fulfillment	$Rule_n$	-
3	Sum of Rules	$Rule_n$	-
4	Rule Output	$Rule_n$	$In_n + 1$
5	Overall Output	1	-

Table 6.2: Amount of Nodes and Parameters in an Single-Output ANFIS

6.2 Learning Algorithms

The ANFIS can be tuned with different learning methods. However, the most used ones are the Gradient Descent Method and the Least Squares Estimation. Moreover, it is also possible to combine different algorithms to make the method even more efficient, as proposed by Jang in [11].

6.2.1 Gradient Descent Method

The GDM [15] is a back-propagation learning method and works with a calculated squared error. This error is calculated as the following:

$$E = (O_d - O_a)^2 \tag{6.3}$$

where O_d is the desired Output and O_a is the actual Output. We can then calculate the *error rate* from (6.3):

$$\frac{\partial E}{\partial O} = -2 \cdot (O_d - O_a) \tag{6.4}$$

The error rate of the internal nodes can be derived with use of the chain rule:

$$\frac{\partial E}{\partial O_k} = \sum_{m=1}^{\#(k+1)} \frac{\partial E}{\partial O_{k+1}} \cdot \frac{\partial O_{k+1}}{\partial O_k} \tag{6.5}$$

where k stands for the layer and i for the position in the kth layer. As a result, the update formula for the generic parameter α is

$$\Delta\alpha = -\eta \cdot \frac{\partial E}{\partial \alpha} \tag{6.6}$$

η is a learning rate and can be expressed as

$$\eta = \frac{S}{\sqrt{\sum_\alpha \left(\frac{\partial E}{\partial \alpha}\right)^2}} \tag{6.7}$$

where S is the *step size*. S can vary the speed of convergence to the optimal function. However, a step size that is too large causes an *overshoot* in the learning curve.

In the adopted code by Jang, an adaptive step size with 2 update rules (Figure 6.5) is used. The step size gets *increased*, when RMSE is decreasing 4 times in a row and the step size gets *decreased*, if 2 bounces in a row occur (Figure 6.5. With these simple rules, a faster and more efficient learning curve is achieved. The difference in convergence speed is shown in Figure 6.6. However, the GDM is known to be generally slow and likely to become trapped in a local minima.

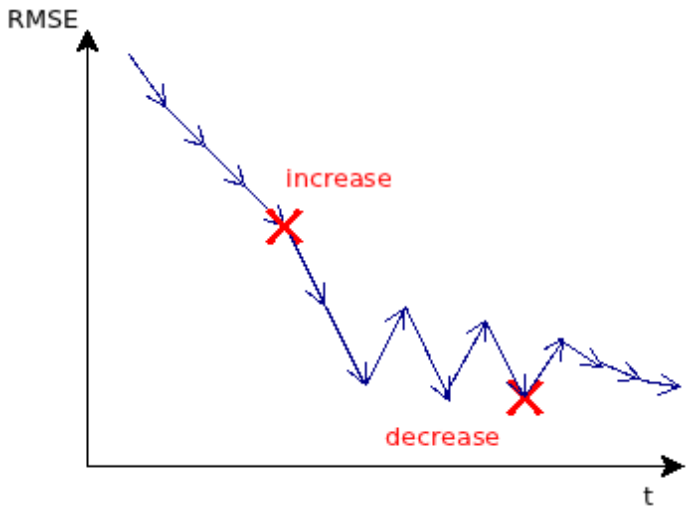

Figure 6.5: Step Size Update Rules

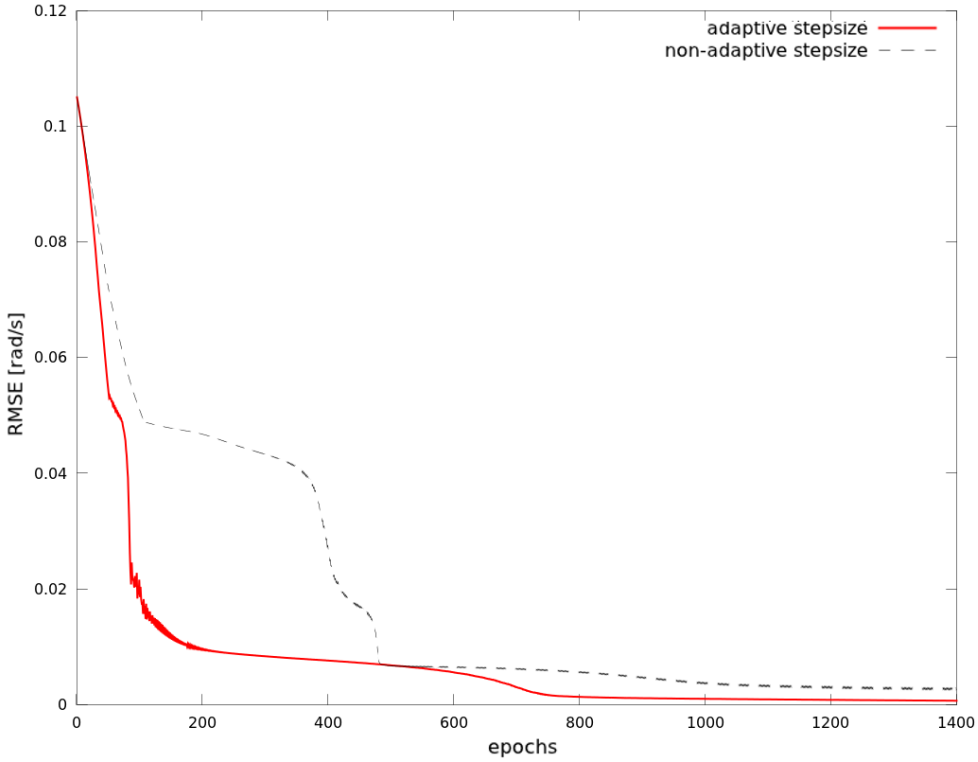

Figure 6.6: Comparison of Off-Line Training With and Without Adaptive Step Sizes

6.2.2 Least Squares Estimation

This algorithm is a common method to approach the best solution in an overdetermined system and was first described by Carl Friedrich Gauss around 1794 [16]. Jang describes its use for an adaptive network in [11]. We assume, for simplicity, that an adaptive network has only one output

$$output - F\left(\vec{I}, S\right) \tag{6.8}$$

where $\vec{I}$ is the set of input variables and S is the set of parameters. The elements of S can be identified by the LSE, if there is a function H, such that the composite function $H \circ F$ is linear in some of the elements of S. If the set S can be decomposed into two sets

$$S = S_1 \oplus S2 \tag{6.9}$$

(where $\oplus$ represents a direct sum) such that $H \circ F$ is linear in the elements $S2$, then upon applying H to (6.8), we have

$$H\left(output\right) = H \circ F\left(\vec{I}, S\right) \tag{6.10}$$

which is linear in the elements of S_2. We can use our training data as the elements of S_1 and obtain a equation

$$A \cdot X = B \tag{6.11}$$

where A are the input of our training patterns, X is an unknown vector whose elements are the *consequent parameters* in ANFIS-Layer 4 (Figure 6.4), and B is the output value of the training patterns. As we have more than 1 pattern, this is an overdetermined problem and therefore there is no exact solution to (6.11). So we want to calculate the least squares estimate (LSE) of X, X^*, which is sought to minimize the squared error $[A \cdot X - B]^2$ [11]. The most well-known formula for X^* uses the pseudo-inverse of X:

$$X^* = \left(A^T \cdot A\right)^{-1} \cdot A^T \cdot B \tag{6.12}$$

where A^T is the transpose of A, and $\left(A^T \cdot A\right)^{-1} \cdot A^T$ is the pseudo-inverse of A if $A^T \cdot A$ is non-singular. As (6.12) is very computation heavy, we use the sequential Kalman Filter formulation for the LSE:

$$\begin{aligned} X_{i+1} &= X_i + S_{i+1} \cdot ga_{i+1}\left(b_{i+1}^T - a_{i+1}^T \cdot X_i\right) \\ S_{i+1} &= S_i - \frac{S_i \cdot a_{i+1} \cdot a_{i+1}^T \cdot S_i}{1 + a_{i+1}^T \cdot S_i \cdot a_{i+1}}, \quad i = 0, 1, \ldots, P-1 \end{aligned} \tag{6.13}$$

If we add a forgetting factor λ to (6.13), to weight the patterns and consider the most recent computations as more important, the LSE-equations become the following:

$$\begin{aligned} X_{i+1} &= X_i + S_{i+1} \cdot a_{i+1}\left(b_{i+1}^T - a_{i+1}^T \cdot X_i\right) \\ S_{i+1} &= \frac{1}{\lambda} \cdot \left[S_i - \frac{S_i \cdot a_{i+1} \cdot a_{i+1}^T \cdot S_i}{\lambda + a_{i+1}^T \cdot S_i \cdot a_{i+1}}\right], \quad i = 0, 1, \ldots, P-1 \end{aligned} \tag{6.14}$$

The value of λ is between 0 and 1, where a smaller number means a faster decay of old values. However, a small λ can also cause numerical instability and should be avoided.

6.2.3 Hybrid Learning Procedure

Jang proposed a Hybrid Learning Procedure in [11], which combines the Gradient Descent Method and the Kalman Filter formulation of the Least Squares Estimation. In a first pass the consequent parameters from ANFIS-Layer 4 are pre-set via LSE, and after calculating the error of the system, in the second pass the error is back-propagated via the GDM, and the premise parameters of Layer 1 are updated. See Table 6.3 for an overview.

	Forward Pass	**Backward Pass**
Premise Parameters	Fixed	Gradient Descent
Consequent Parameters	Least Squares Estimate	Fixed
Signals	Node Outputs	Error Rates

Table 6.3: Two Passes in the Hybrid Learning Procedure by Jang

6.3 Implementation of the ANFIS-Code

6.3.1 Issues with the Micro-Controller

The main issue in implementing the ANFIS was the limitation of memory available, especially the availability of the SRAM, which is $8\mathrm{kByte}$ on the Arduino Mega microcontroller board, as the ANFIS architecture and its calculations require big arrays for computation. The on-line version of the ANFIS is based on the open source code version, written in C by Jang [11]. In a first step, data-types for the ANFIS structure were changed, such as using *byte* instead of *integer* if possible. A huge amount of working memory could be saved, with taking the Kalman Filter out of the code, as there is no use for the LSE algorithm in the on-line version of the ANFIS code. The reason for this is described in Chapter 6.3.5. Another way to save memory is to replace the configuration file with all the node connections as Jang does it in his code, with the algorithm that creates the file, and calculate the connections just while building the ANFIS. Moreover, to keep the ANFIS program as flexible as possible, the parameter set is stored in the EEPROM. With a special reading algorithm, it is even possible to read and store different parameter sets from the EEPROM which are stored as in Figure 6.7. To test the ANFIS code, a off-line trained parameter set was loaded into the ANFIS architecture and trained on-line, by randomly moving the IMU to simulate an error of the output and thus forcing high update values of the gradient descent method. Figure 6.8 shows the learning capability of the on-line ANFIS code on the Arduino microcontroller

board.

set 1						set 2			. . .
In_n	Mf_n	Stepsize	parameter 1	parameter 2	. . .	In_n	Mf_n	. . .	. . .
1 Byte	1 Byte	4 Byte float	4 Byte float						

Figure 6.7: Datastructur of Parametersets in EEPROM

The final version of the code is able to run a single input ANFIS from 2 to 7 membership functions and 2 input ANFIS with 2 to 4 membership functions. The used space of the total $8219\mathrm{Bytes}$ is shown in Table 6.4.

#Inputs	**#Membershipfunctions**	**Used SRAM [byte]**
1	2	1680
	3	2000
	4	2300
	5	2619
	6	2980
	7	3365
2	2	2456
	3	4335
	4	7680

Table 6.4: Used Amount of SRAM on the Microcontroller Board

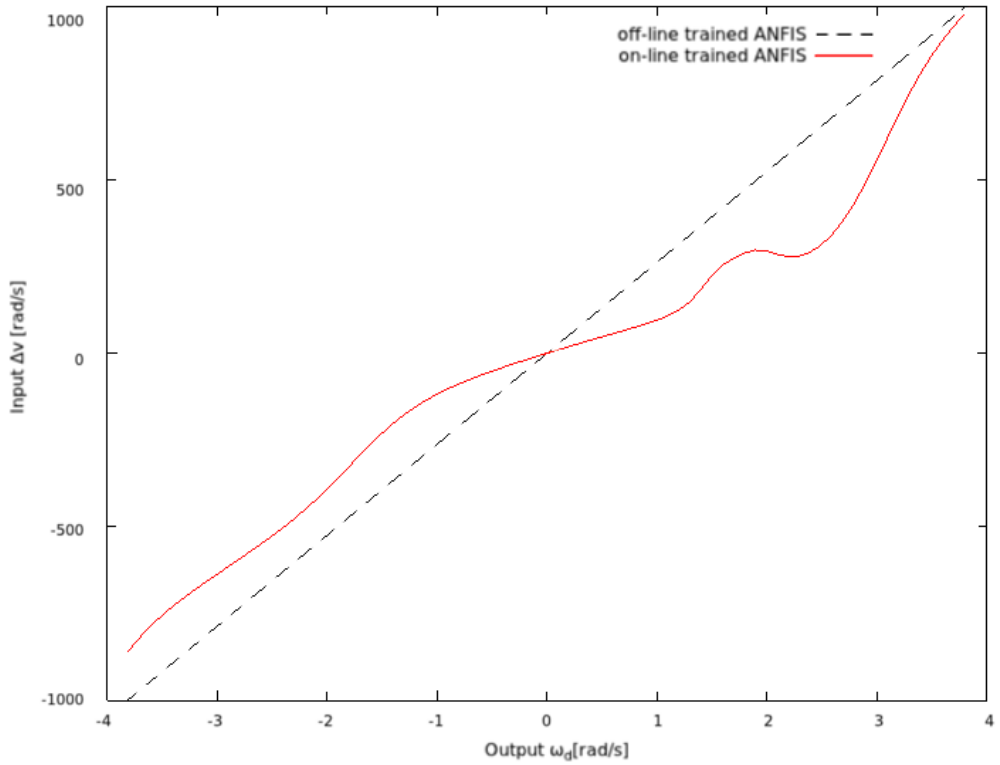

Figure 6.8: Test Learning Capability of the On-Line ANFIS Code

6.3.2 Modified MIMO ANFIS Controller

Previously, we discussed the implementation of a *multi-input-single-output ANFIS*. However, we have a *multi-input-multi-output ANFIS Controller* as it reports 2 velocities for the robot's wheels.

The user reports a desired change of bearing and a velocity of the robot. Depending on the chosen mode, the ANFIS uses the users bearing or the bearing reported by the tracking strategy. As the modified MIMO-ANFIS Controller works as an inverse model of the robot itself, it corrects the desired bearing in a way that the robot's movement corresponds with the desired movement.

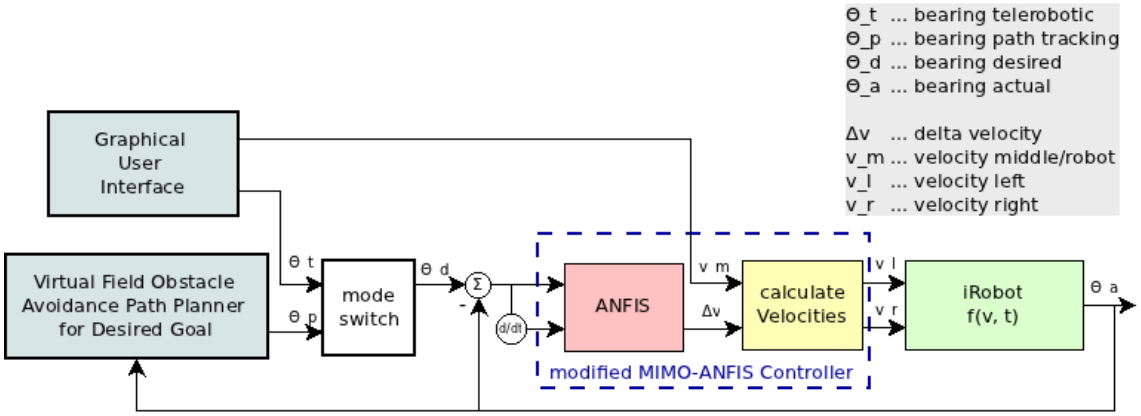

Figure 6.9: Integration of a MIMO Anfis Controller into the iRobot

As the SBC has to request every new ANFIS computation but should not lose connection to the server, the ANFIS microcontroller board is implemented in the existing code with a separate running thread as in Figure 6.10. To avoid a buffering of requests on the ANFIS microcontroller, which is very slow in the GUI, we use mutex protected global variables between the ANFIS Thread and the Main Thread. In this way we can guarantee that a new ANFIS request is always made with the newest GUI commands of the user. The Main Thread updates the variables velocity v and change in bearing ω whenever the user interacts with the user interface. Accordingly, the ANFIS Thread runs in a timed action (determined through the responding time of the microcontroller board and optional with a timed action) and always requests the ANFIS board with the most recent values. The ANFIS on the microcontroller then requests the IMU due to update information about the robot, as shown in Figure 6.11, to also work with the most recent measurements of the AHRS.

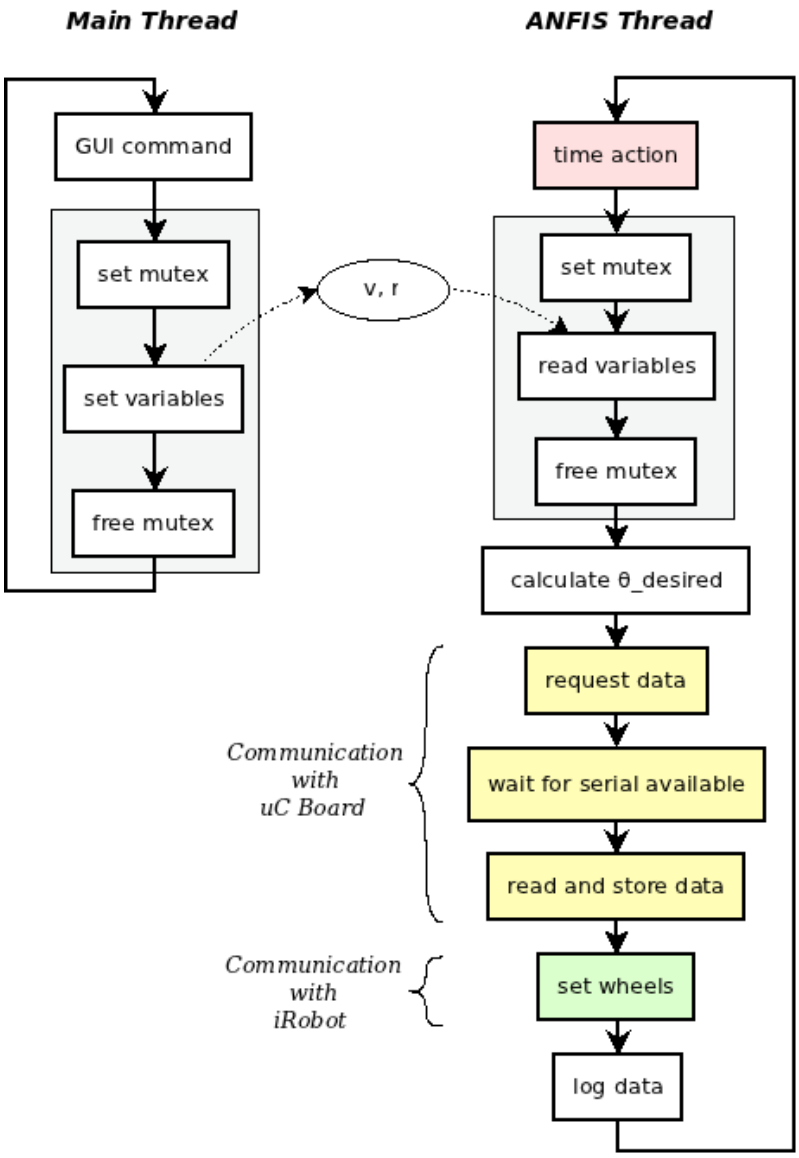

Figure 6.10: Flow Diagram of the ANFIS Thread Running on the SBC

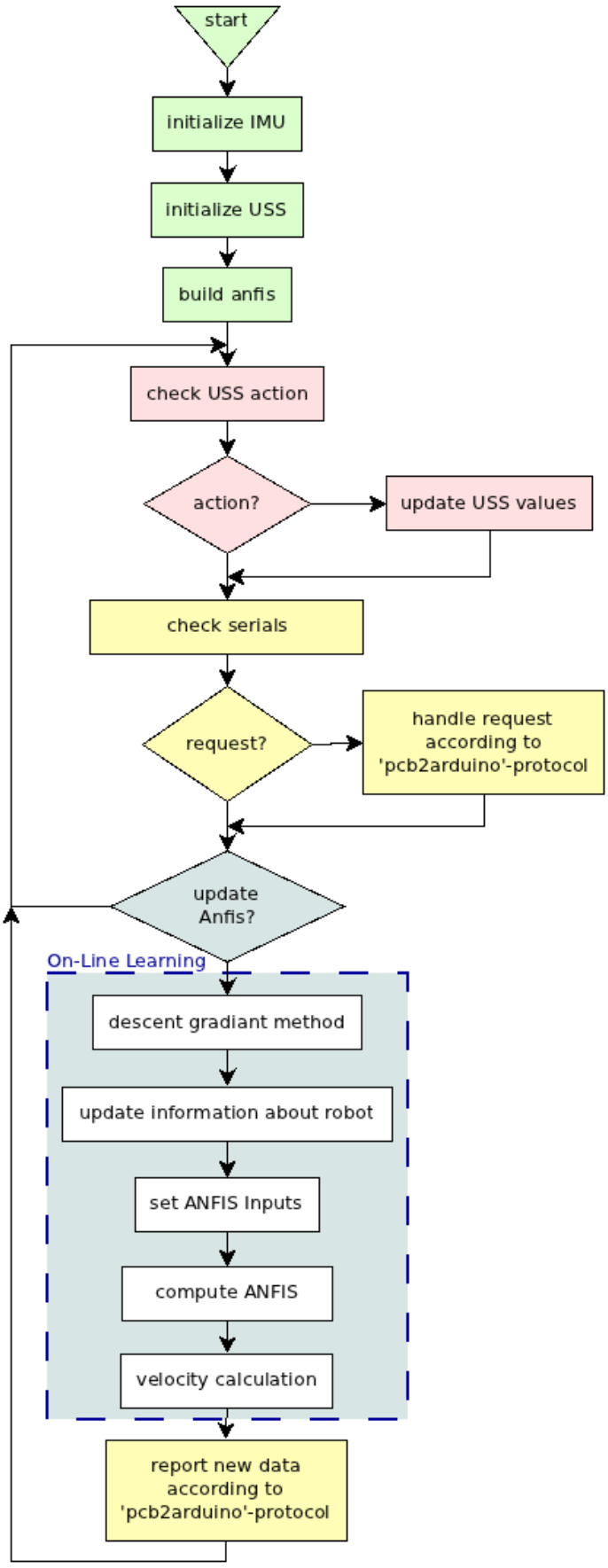

Figure 6.11: Flow Diagram of the iRobot Controller Running on the Arduino Microcontroller Board

6.3.3 Collecting Training Data

Theoretical Function

For an ideal case, perfect symmetric motor characteristics are assumed, and no regard on acceleration ramps of the motors, nor wheel slippage is taken. If the robot rotates around a point R, the geometrical relations between the velocities of the left wheel, right wheel and the middle velocity of the center of the robot, are then as shown in Figure 6.12.

As the angular velocities between the wheels are equivalent $\omega_L = \omega_M = \omega_R$, it is possible to derive a formula for the change in orientation $\omega = f(\Delta v)$, using the geometric congruence of velocities and radii:

$$\begin{aligned} \frac{r}{v_M} &= \frac{r + \frac{k}{2}}{v_R} \\ r &= \frac{k \cdot v_M}{2 \cdot (v_R - v_M)} \end{aligned} \tag{6.15}$$

With $\omega = \frac{v_M}{r}$ and $v_M = \frac{v_L + v_R}{2}$ we can calculate ω as:

$$\omega = \frac{(v_R - v_L)}{k} = \frac{\Delta v}{k} \tag{6.16}$$

where k is the distance between the wheels. With a maximum velocity of $\pm 500 \frac{\text{mm}}{\text{s}}$ [17] and measured wheel distance $k \approx 262\text{mm}$, we get maximum $\omega \approx \pm 3.81 \frac{\text{rad}}{\text{sec}}$. Accordingly, the formula to create training data with ω as the input and Δv as the outputs:

$$\Delta v = \Delta\omega \cdot k, \quad \Delta\omega = -3.81, -3.80, \ldots, +3.81 \tag{6.17}$$

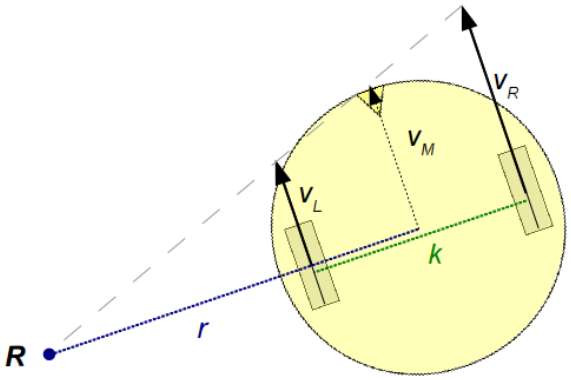

Figure 6.12: Theory of the Robot's Movement

Collecting Data with iRobot

In comparison to the theoretical training data, another set of training data is collected by using the robot itself. Thus, the training data for the off-line training already contains the dynamic characteristics of the iRobot.
The training data is created out of a log file (e.g. Fig.6.5 with the full log in the appendix) from a test drive in telerobotic mode. From such a log, the desired bearing $thetaD$, which is the user input from the GUI 2.9, is used as ω_c, and the actual change in bearing over time ω_b is calculated with $thetaA$, which is the logged magnetic compass value from the IMU. Additionally ω_y is calculated with yaw, which is the bearing value reported from the AHRS on the IMU with.

$$\omega_{b_t} = \frac{thetaA_{t+1} - thetaA_t}{time_{t+1} - time_t} \tag{6.18}$$

and alternatively

$$\omega_{y_t} = \frac{yaw_{t+1} - yaw_t}{time_{t+1} - time_t} \tag{6.19}$$

Training patterns are created with the desired ω_d, and respectively $\frac{d\omega_d}{dt}$, as inputs, and Δv as output

$$\omega_d = \omega_b \tag{6.20}$$

or

$$\omega_d = \omega_y \tag{6.21}$$

and

$$\Delta v = \omega_c \cdot k \tag{6.22}$$

where k is again the wheel distance.

time[s]	[ms]	v	r	thetaD	thetaA	yaw	rmse
⋮		⋮	⋮	⋮	⋮	⋮	⋮
25	740	170	215	279	38.45	28.02	0.02
25	881	170	215	279	42.23	33.86	0.058
26	34	170	215	279	47.38	39.71	0.035
26	136	175	787	303	54.6	45.15	0.003
26	264	175	787	303	58.38	51.34	0.058
26	403	130	1191	320	63.08	56.09	0.076
26	535	130	1191	320	64.34	58.38	0.134
26	659	120	-1929	8	64.34	60.5	0.064
26	792	120	-1929	8	65.83	61.48	0.038
26	936	120	-1929	8	66.86	62.4	0.025
27	66	145	-1739	16	65.6	63.03	0.065
27	195	145	-1739	16	65.49	63.43	0.002
27	328	145	-1739	16	64.46	63.71	0.106
27	451	145	-1739	16	63.6	63.71	0.101
⋮		⋮	⋮	⋮	⋮	⋮	⋮

Table 6.5: Log of a Test Drive in Telerobotic Mode

6.3.4 Off-Line Learning

For the supervised learning procedure, the open source ANFIS code (written in C) by Jang is used, as proposed in [11]. Trained with our ideal function (6.17) the membership functions for a single input system with 2 to 7 membership functions are shown in Figure 6.13. Due to the training data being derived from an first-degree polynomial function, the membership functions are perfectly symmetric.

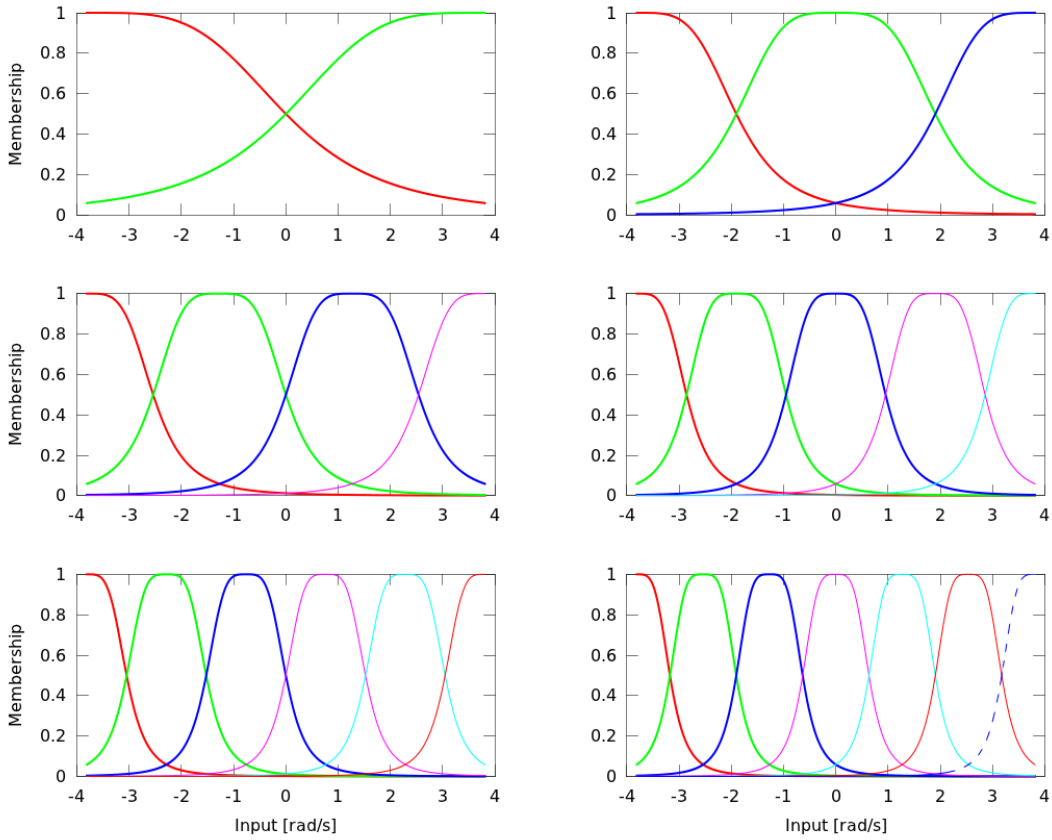

Figure 6.13: Membership Functions After Off-Line Training on Theoretical Function Set

6.3.5 On-Line Learning

The second part of the ANFIS training takes place right on the robot during operation. Therefore, after every ANFIS update request from the SBC, the microcontroller operates using the following 5 steps, shown in Figure 6.11:

First: At the beginning of each ANFIS computation, a *Gradiant Descent Method* (Chapter 6.2.1) is applied. The error of the ANFIS is calculated by comparison of the desired change in bearing ω_d, which was the input of the last ANFIS pass, and the actual change in bearing ω_a reported by the IMU as in (6.3). The parameters in layer 1 and/or layer 4 (depending on chosen learning mode) then get updated as described in equations (6.4) to (6.6).

Second: After the learning algorithm, the information about the robot's orientation is *updated*, through requesting new values from the IMU. The new position of the robot is calculated as discussed in Chapter 4.2.2.

Third: To work with the latest measurement of the IMU, the microcontroller requests the IMU just before computing the ANFIS. Depending on telerobotic- or autonomous mode, the desired change in bearing ω_d as input, is either taken from the SBC request as GUI input, or calculated with the tracking strategy of Chapter 4.3. After the correction of possible bearing changes from 3^{rd} to 4^{th} quadrant the *inputs* get set.

Fourth: Compute the new inputs through all ANFIS layer and set the ANFIS ouput Δv.

Fifth: The last step of the ANFIS-update-request-cycle is the computation of the new velocities for the wheels, using the ANFIS output Δv and the given velocity for the robot v_g. These take effect until the next update request of the SBC, which takes place in a periodic manner.

We can not adopt the Kalman formulation of the LSE from Jang's Code to compute the consequent parameters of Layer 4 because the essential function (6.11) can not be solved, as B is unknown during operation since the actual change in bearing of the robot during the next time period can not be predicted. Consequently, this on-line version of the Code is not a hybrid learning ANFIS, as it solely uses the GDM for learning. However, as the ANFIS is always trained and thus the ANFIS function is formed, the disadvantages of the GDM i.e., being slow and susceptibility to getting trapped in minima, are not a problem as the ANFIS is solely fine tuned while operating on-line.

Chapter 7

Results

7.1 ANFIS Training

The comparison of the ideal function (6.17) and the collected data from a log, as discussed in Chapter 6.3.3, is shown in Figure 7.1. We see, that the measurements of actual change in bearing ω_y, calculated with the yaw and $time$ values of the log in (6.5), are following the ideal function.

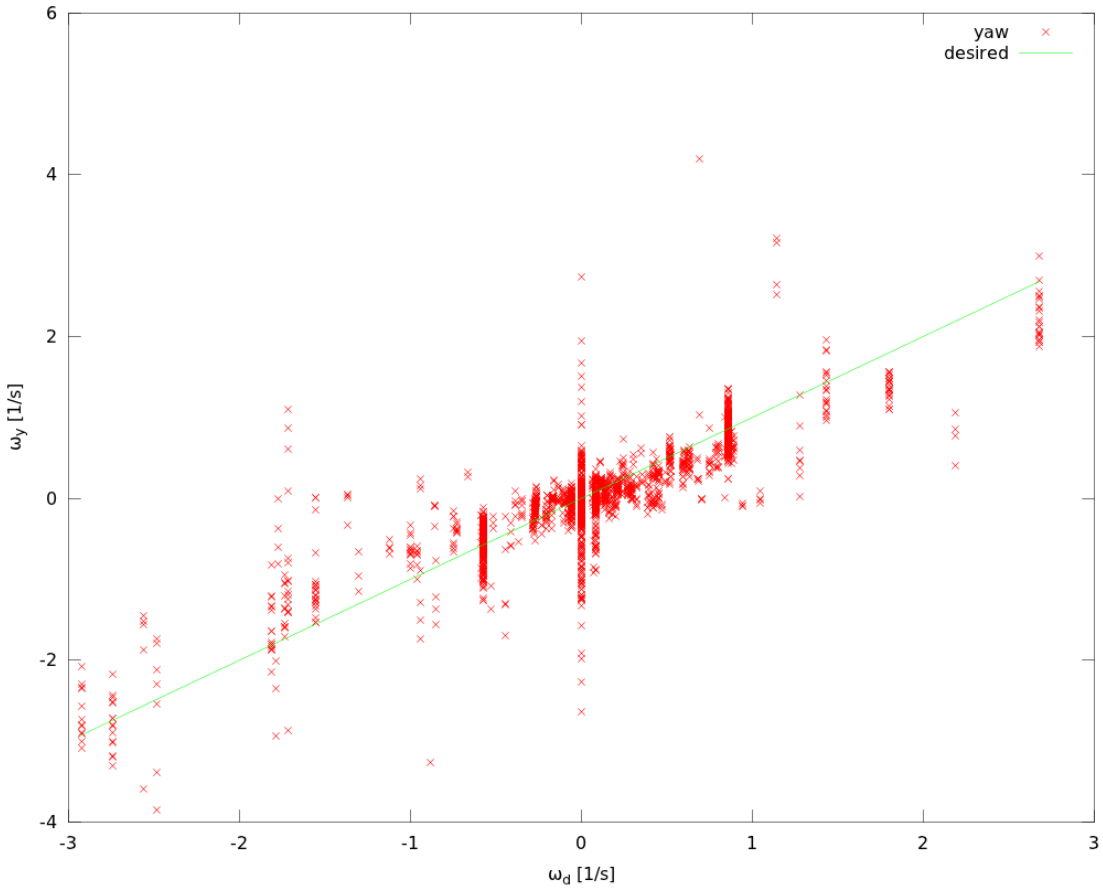

Figure 7.1: Comparison of Ideal Function and Experimental Data Collection

However, they are very dispersed, due to the noisy sensor measurements of the IMU and taking a measurement approximately every 120ms, which causes an amplification of the error by about a factor 8, when projecting the value to bearing per second. For our experiments, we used different ANFIS architectures that are trained with the theoretical linear function of Chapter 6.3.3 during off-line training, as the collected training data is too dispersed. The ANFIS representing function, will then be fine tuned in on-line mode, to compensate, for the nonlinearities in the robot's characteristics, Figure 7.2 shows the ANFIS characteristics for a 2 input system after a short test drive.

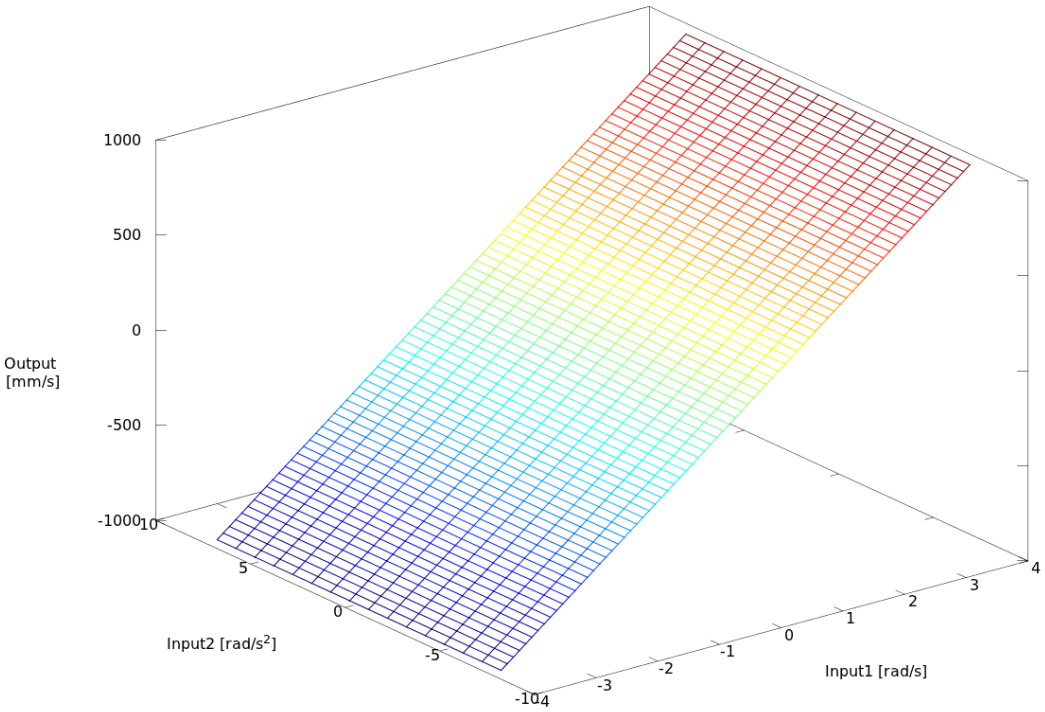

Figure 7.2: Representing Surface of the 2 Input ANFIS System

7.2 ANFIS Responding Time

The microcontroller program was tested with a special programmed serial monitor program that lets a laptop communicate with the arduino board through the 'pcb2arduino'-protocol, and measures the elapsed time, from sending the request to receiving the last byte of the answer. It turns out that an increase in complexity of the ANFIS structure results in an exponential increase of computation time (Table 7.1). This can be explained due to (6.1) and (6.2), which show the strong increase of used nodes. The table also shows that the serial communication itself, without the ANFIS computation, takes about $14\mathrm{ms}$. Further, experiments may show that if there is a significant delay between the request and the actual setting of the wheels, a less complex ANFIS architecture can be used instead.

Inputs	Membership-functions	ANFIS Computation Time	Response Time
1	2	3-4	16-18
	3	7-8	20-23
	4	9-11	23-26
	5	13-15	26-29
	6	17-19	30-32
	7	22-23	36-38
2	2	12-14	26-28
	3	30-31	44-46
	4	80-82	94-96

Table 7.1: Responding Time of the ANFIS Microcontroller System

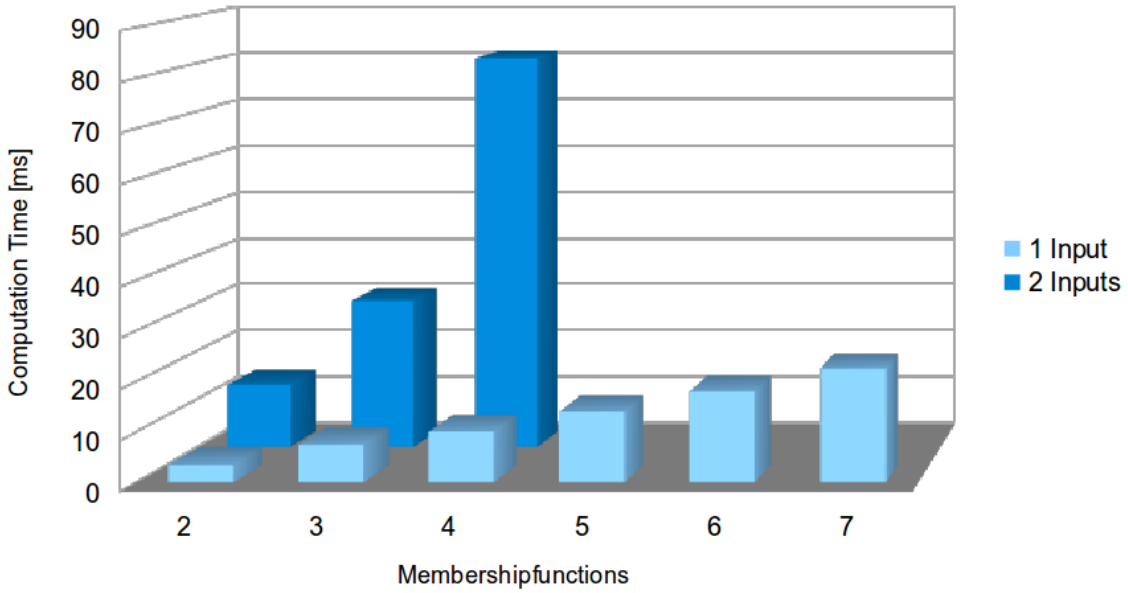

Figure 7.3: Responding Time of the ANFIS Microcontroller System as in table 7.1

7.3 Tracking in Telerobotic Mode

To test the ANFIS in telerobotic mode, the same log file from the test drive in Chapter 6.3.3 was used to plot some graphs as shown in Figures 7.4 and 7.5. For this first test drives, the learning mode of the ANFIS was set to *'no learning'*. The first plot shows the unfiltered desired change in bearing (ω_d), with the measured compass (ω_c) and yaw values (ω_y) of bearing changes over time. The second plot shows the same changes with an average window filter over 10 measurements. The middle plot shows the root mean square error as described in

Chapter 6.3.5 and the lower 2 plots show the bearing (Θ) as cos and as *absolute* value of the bearing. This last 2 plots just show the yaw values, as they are less noisy than the simple magnetic compass sensor measurements reported by the IMU.

Figure 7.4 shows how the ANFIS tracks the desired change in bearing ω_d. The bearings Θ, shown in the lower 2 plots, moves further apart, as $\Theta = \sum \omega \cdot \Delta t$, and the errors in ω are summed in Θ. Furthermore, in Figure 7.5, we see how the robot follows the arbitrary desired commands.

In Figure 7.6 the learning mode was set to *'update all parameters'* and we can see how the ANFIS learns; hence, the RMSE is minimized .

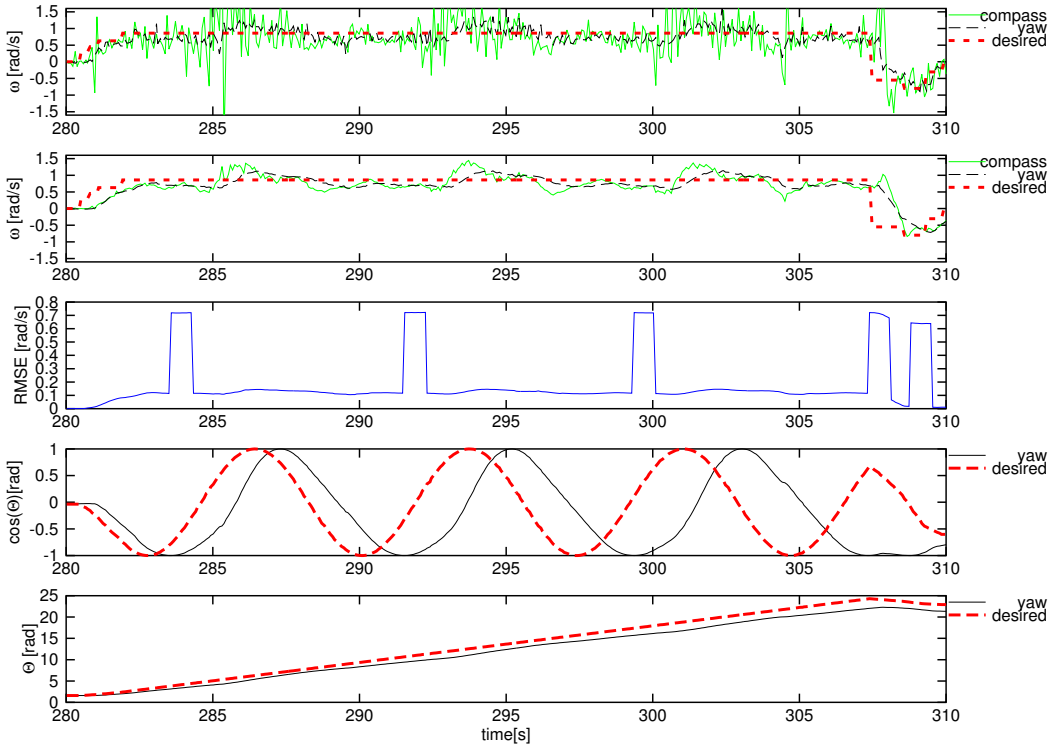

Figure 7.4: Telerobotic Test Drive: ANFIS Reaction on a Continuous Change of Bearing

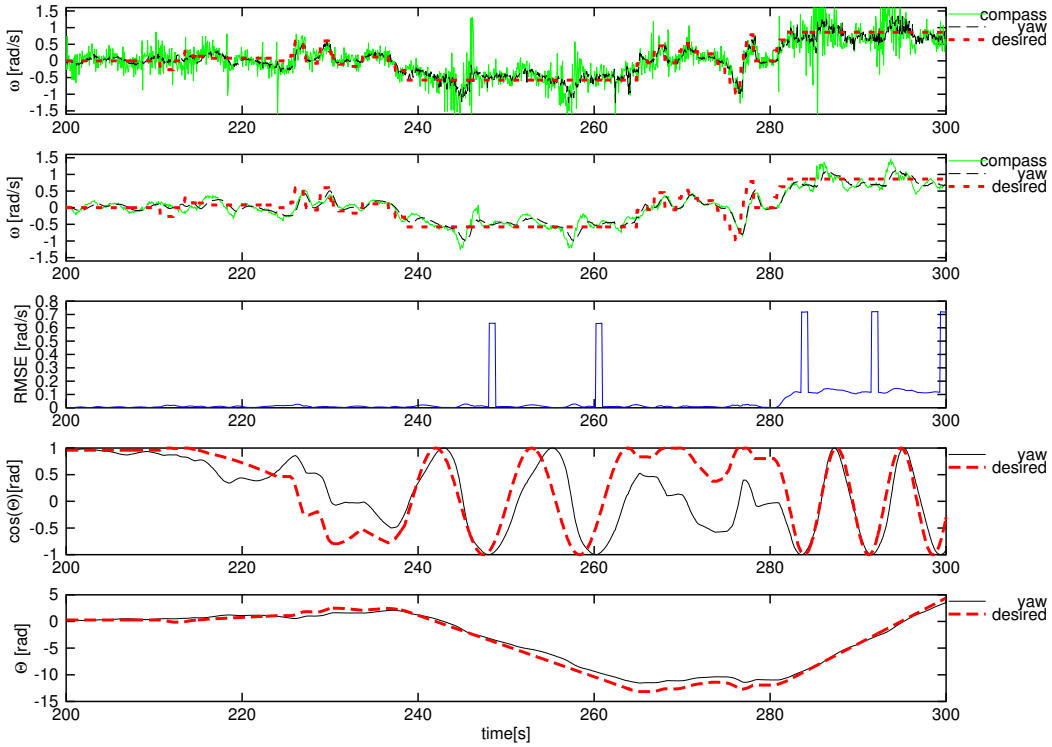

Figure 7.5: Telerobotic Test Drive: ANFIS Reaction on an Arbitrary Change of Bearing

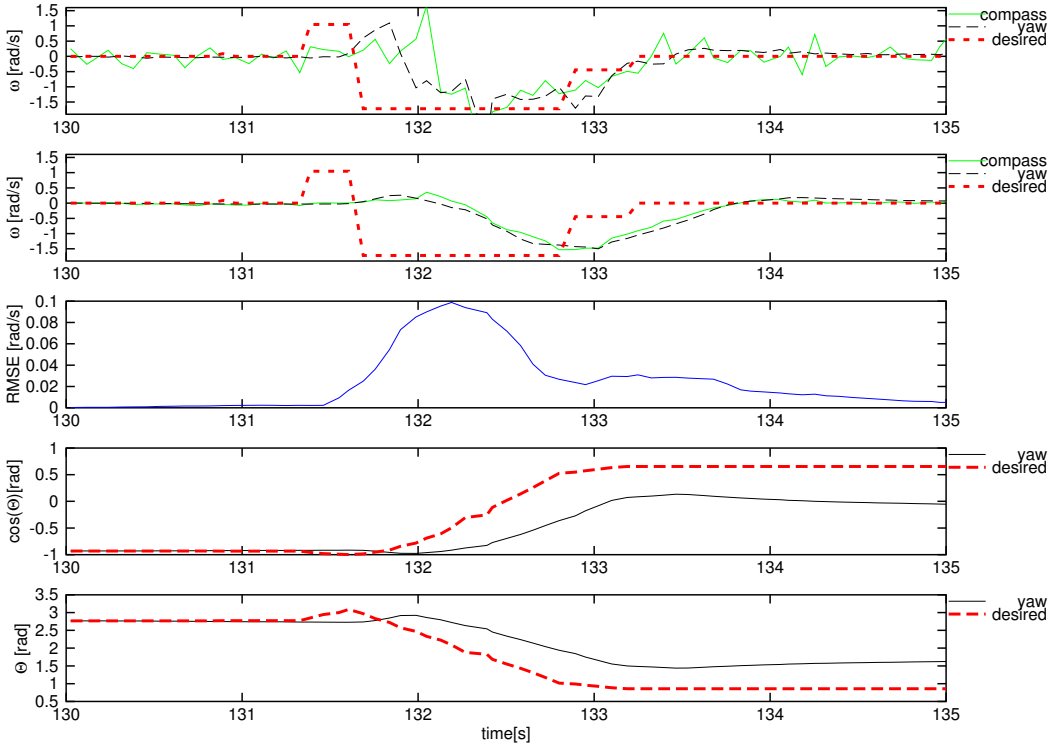

Figure 7.6: Telerobotic Test Drive: ANFIS Tracking and Learning

7.4 Tracking in Autonomous Mode

For an autonomous test, we disabled the request for a new path coordinate from the Arduino board and instead, the coordinate path was automatized to follow an eternal eight with 52 path points (Figure 7.7) with

$$\begin{aligned} x(i) &= 500 * sin(i) \\ y(i) &= 1000 * sin(i) .* cos(i) \\ i &= 0, \frac{2 \cdot \pi}{21} \ldots 2 \cdot \pi \end{aligned} \tag{7.1}$$

Figure 7.7 shows the logged path coordinates of the robot and how it tries to follow the planned path, using a 1 input ANFIS with 5 membership functions and the simple path tracking strategy discussed in Chapter 4.3. This proofs, that with the assumption of the robot knowing its real position, the system is able to follow a path described in coordinates. The `yaw` vectors show the reported orientation from the IMU at each update of the ANFIS. The thick `threshold` line shows the threshold of the coordinates with $50\mathrm{mm}$.

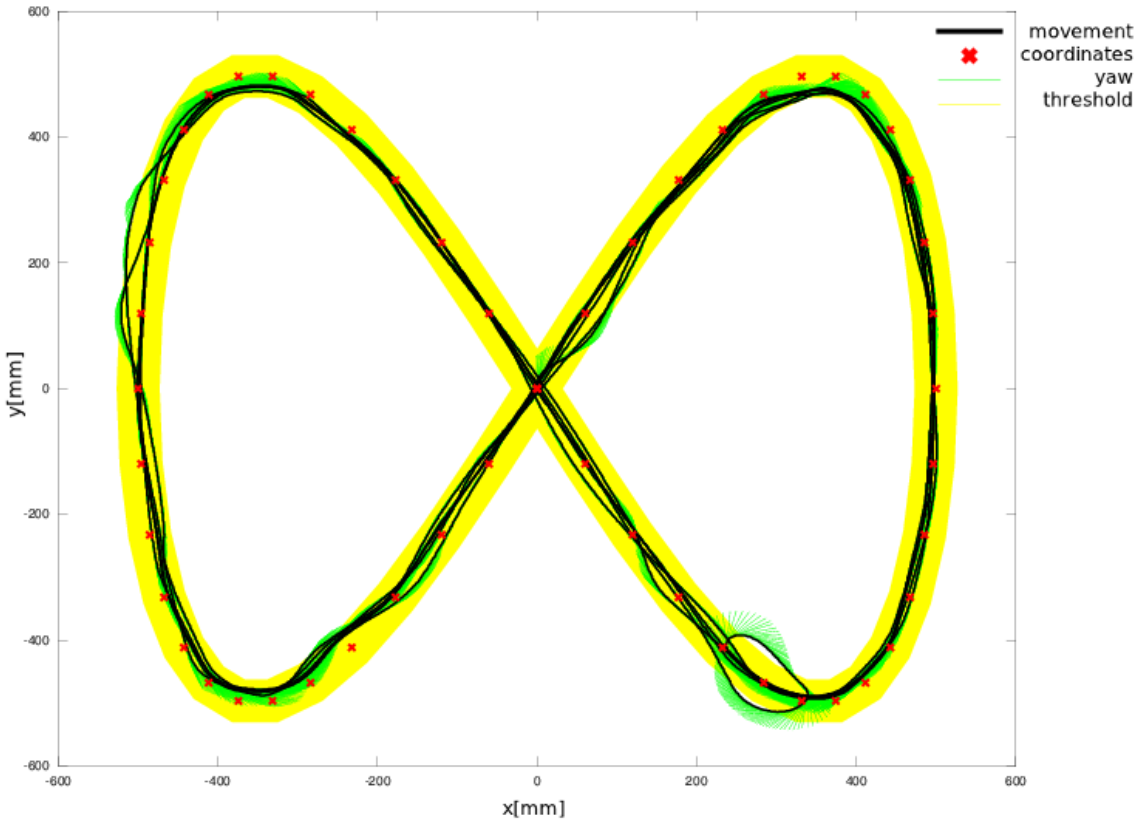

Figure 7.7: Autonomous Test Drive: Eternal Eight

However, experiments about the comparison of desired movement, real movement and assumed movement show that the position calculation out of the commands is not accurate enough. The average results (out of 3 performances) of the commands, measured distances and calculated positions are shown in Figure 7.8, where `calc w/` is the calculated distance with the reported `yaw` of the IMU and `calc wo/` is the calculated distance without regard of the yaw value. That means `calc wo/` used a simulated static `yaw` $=$ `0` instead. The error of `calc w/` can be traced to the robot thinking it drives in little left/right turns, due to the noisy `yaw` measurement. As a result, the distance in the figure represents the vector calculated to the end position of a curved movement. According to this error, the robot's movement is not the same as the logged positions of Figure 7.7. With additional sensor information on the actual position of the robot, it is anticipated that better results can be generated, which is an area of future research.

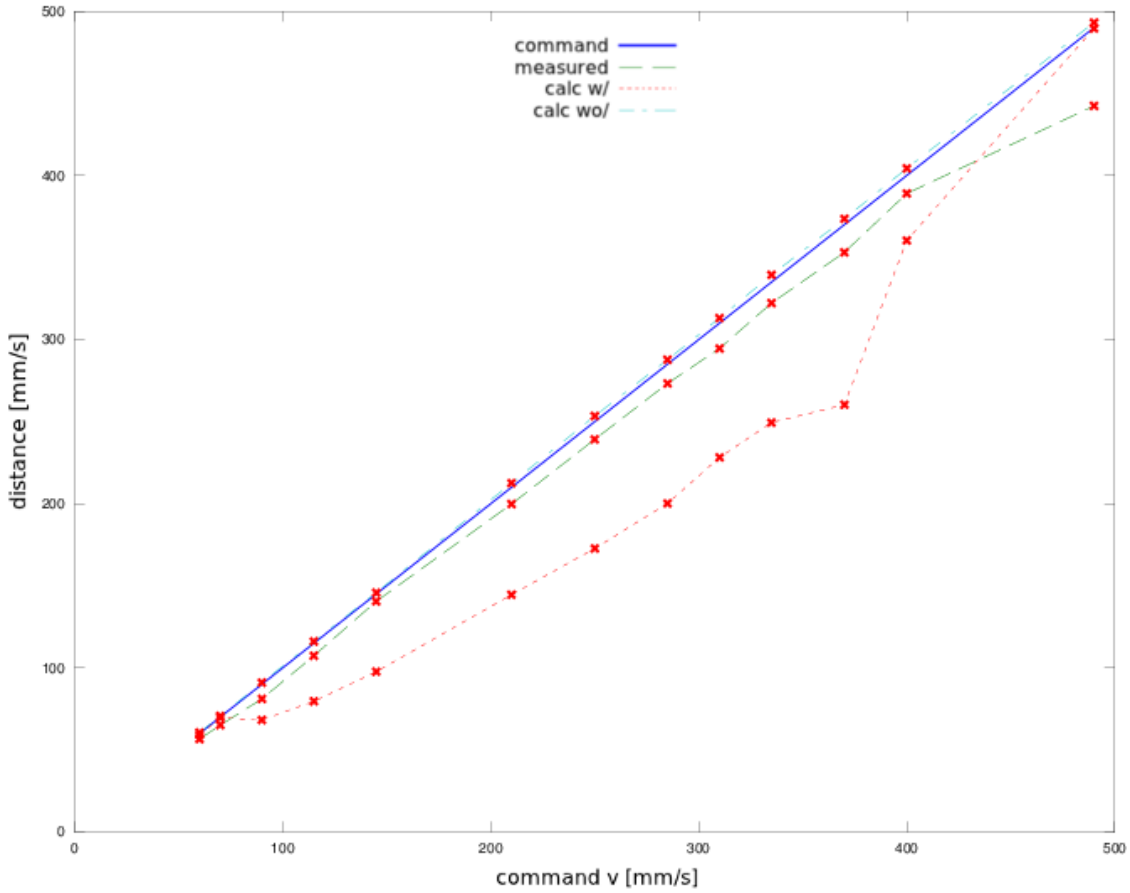

Figure 7.8: Comparison of the Robot's Movement

Chapter 8

Conclusions and Future Work

The ANFIS controller together with protocol was designed as a very flexible system. With the developed adaptive ANFIS controller, a mobile robot testbed was created that can be used for future control engineering research projects, such as studies with different ANFIS architectures and the development of learning algorithms but also more advanced path tracking and path planning strategies. To improve the system, above all it is necessary to improve the IMU algorithm as discussed in [18] or with a Kalman filter as in [8]. Hence, implementation of a compensation algorithm that makes it possible to estimate the robot's position from the sensor values in addition to the dead reckoning using the wheel velocities; so slippage of the wheels does not cause an error in position calculation anymore. Another research topic would be the tracking strategy algorithm which calculates the desired orientation from the path coordinates which is then sent to the ANFIS controller to achieve a sufficient tracking in autonomous robot operation. It is expected that this new mobile robotic testbed will provide additional opportunities for future algorithm development and testing as well as human-machine interface advances.

List of Figures

2.1 System Architecture of the Mobile Robotic System 3
2.2 The Mobile Robotic System 4
2.3 Top and Bottom View of the iRobot Create Platform 5
2.4 Ultra Sonic Sensor Array 6
2.5 Acoustic Cone Pattern of the Ultra Sonic Sensor Array 7
2.6 2-D Plot of Thermal Array Sensor Reading 8
2.7 Used ATmega1280 Based Microcontroller Board by Arduino 9
2.8 Used Migrus C787 DCF-P Single Board Computer 10
2.9 Simple AJAX Based Web Interface for Telerobotic Control of the Mobile Robot System 11
2.10 Certainty Grid Compared With Real Environment 12
2.11 Path Planning to Avoid Obstacles 13

3.1 9 Degree of Freedom IMU from Sparkfun 14
3.2 Answer String of the IMU on a ANFIS-Board Request 16

4.1 Definition of Directions, Bearing and Coordinates 18
4.2 Position Dead Reckoning 20
4.3 Path Tracking Strategy 21

5.1 Structure for a Request from the SBC to the Arduino 22

6.1 Fuzzy Inference System 32
6.2 Adaptive Neural Network 33
6.3 Single Input Single Output ANFIS 34
6.4 Multi Input Single Output ANFIS 34
6.5 Step Size Update Rules 37
6.6 Comparison of Off-Line Training With and Without Adaptive Step Sizes 38
6.7 Datastructur of Parametersets in EEPROM 41
6.8 Test Learning Capability of the On-Line ANFIS Code 42
6.9 Integration of a MIMO Anfis Controller into the iRobot 43

6.10 Flow Diagram of the ANFIS Thread Running on the SBC 44
6.11 Flow Diagram of the iRobot Controller Running on the Arduino Microcontroller Board 45
6.12 Theory of the Robot's Movement 46
6.13 Membership Functions After Off-Line Training on Theoretical Function Set 49

7.1 Comparison of Ideal Function and Experimental Data Collection . 52
7.2 Representing Surface of the 2 Input ANFIS System 53
7.3 Responding Time of the ANFIS Microcontroller System 54
7.4 Telerobotic Test Drive: ANFIS Reaction on a Continuous Change of Bearing 55
7.5 Telerobotic Test Drive: ANFIS Reaction on an Arbitrary Change of Bearing 56
7.6 Telerobotic Test Drive: ANFIS Tracking and Learning 57
7.7 Autonomous Test Drive: Eternal Eight 58
7.8 Comparison of the Robot's Movement 59

List of Tables

3.1 Characteristics of the Acceleration Sensor ADXL345 15
3.2 Characteristics of the Gyro Sensors LY530ALH and LPR530ALH . 15
3.3 Characteristics of the Magnetic Digital Compass HMC5843 15

5.1 Description of Protocol Values 23
5.2 List of Commands from SBC to Arduino 24
5.3 Messages from Arduino to SBC 30

6.1 Layer Description of an ANFIS 35
6.2 Amount of Nodes and Parameters in an Single-Output ANFIS . . . 36
6.3 Two Passes in the Hybrid Learning Procedure by Jang 40
6.4 Used Amount of SRAM on the Microcontroller Board 41
6.5 Log of a Test Drive in Telerobotic Mode 48

7.1 Responding Time of the ANFIS Microcontroller System 54

Bibliography

[1] irobot create. http://www.botmag.com/articles/irobot_create.shtml, June 2011.

[2] C. Paolini and S. Natarajan. Adaptive ajax-based streaming video for the irobot create platform for use in buildings with infrastructure mode 802.11 networks. In *World Automation Congress (WAC), 2010*, pages 1–6. IEEE.

[3] A. Gallardo, J. Taylor, C. Paolini, H.K. Lee, and G. Lee. An anfis-based multi-sensor structure for a mobile robotic system. In *Proc. of the IEEE Symposium on Computational Intelligence*, April 2011.

[4] An adaptive streaming video infrastructure for a wireless irobot create platform. Master's thesis, San Diego State University, 2009.

[5] A. Gallardo. Implementation of the vector vield histogram and virtual force field methods for mobile robots. Technical report, MESA/SDSU, 2010.

[6] 9 degrees of freedom - razor imu - ahrs compatible. http://www.sparkfun.com/products/9623, June 2011.

[7] Ahrs for sparkfun's "9dof razor imu". http://code.google.com/p/sf9domahrs/, June 2011.

[8] Kalman filtering of imu data. http://tom.pycke.be/mav/71/kalman-filtering-of-imu-data, June 2011.

[9] J. Borenstein, HR Everett, L. Feng, D. Wehe, NAVAL COMMAND CONTROL, OCEAN SURVEILLANCE CENTER RDT, and E DIV SAN DIEGO CA. Mobile robot positioning: Sensors and techniques. *Journal of robotic systems*, 14(4):231–249, 1997.

[10] AD King. Inertial navigation-forty years of evolution. *GEC review*, 13(3):140–149, 1998.

[11] J.S.R. Jang. ANFIS: Adaptive-network-based fuzzy inference system. *Systems, Man and Cybernetics, IEEE Transactions on*, 23(3):665–685, 1993.

[12] C.C. Lee. Fuzzy logic in control systems: fuzzy logic controller. i. *Systems, Man and Cybernetics, IEEE Transactions on*, 20(2):404–418, 1990.

[13] C.C. Lee. Fuzzy logic in control systems: fuzzy logic controller. ii. *Systems, Man and Cybernetics, IEEE Transactions on*, 20(2):419–435, 1990.

[14] W.T. Miller, R.S. Sutton, and P.J. Werbos. *Neural networks for control*. MIT Press (MA), 1995.

[15] J.S.R. Jang. Self-learning fuzzy controllers based on temporal backpropagation. *Neural Networks, IEEE Transactions on*, 3(5):714–723, 1992.

[16] H. W. Sorenson. Least-squares estimation: from gauss to kalman. *IEEE Spectrum*, 7:63–68, July 1970.

[17] iRobot Corporation. *iRobot Create Open Interface (OI) Specification*, 2006.

[18] S.O.H. Madgwick. An efficient orientation filter for inertial and inertial/magnetic sensor arrays. *Report x-io and University of Bristol (UK)*, 2010.

Abbreviations

ANFIS	Adaptive Neural Network Fuzzy Inference Structure
AHRS	Attitude Heading Reference System
DB25	D-sub miniature electrical connector with shell size B and 25 pins
DoF	Degrees of Freedom
DCM	Direction Cosine Matrix
fps	Frames Per Second
GHz	Giga Hertz
GPS	Global Positioning System
GDM	Gradient Descent Method
g	Gravity $\approx 9.81^{m}/_{s^2}$
Hz	Hertz
I2C	Inter-Integrated Circuit; a serial communication bus
IMU	Inertial Measurement Unit
LSE	Least Squares Method
Mbps	Mega Bits Per Second
m	Meter
mAh	Milli Ampere Hours
mg	Milli Gravity $\approx 9.81^{mm}/_{s^2}$
mm	Milli Meter
Ni-MH	Nickel-Metal Hybrid Cell
PCB	Personal Computer Board
px	Pixel
rad	Radiant
RMSE	Root Mean Square Error
s	Seconds
SBC	Single Board Computer
SRAM	Static random-access memory
TS	Thermal Sensor
ULV	Ultra Low Voltage
USS	Ultra Sonic Sensors
USB	Universal Serial Bus
V	Volt
WLAN	Wireless Local Area Network
°	Degree

Attachment

1. Code
   (a) iRobotController v1 (for Arduino ATmega1280)
   (b) pcb2eeprom Protocol v1
   (c) write2eeprom v1_1 (for Arduino ATmega1280)
   (d) readeeprom v1_1 (for Arduino ATmega1280)
   (e) off-line ANFIS code by Jang (C program)
   (f) AHRS code by Doug Weibel and Jose Julio (for 9dof IMU)
2. Log Files
   (a) Log Telerobotic Test Drive
   (b) Log Autonomous Eight Test Drive
3. Data-sheets (Arduino/IMU/iRobot)
4. Used Papers (ANFIS/Navigation)