# DESIGN AND DEVELOPMENT OF FUZZY LOGIC OPERATED MICROCONTROLLER BASED SMART MOTORIZED WHEELCHAIR

By

Hamid Reza Moslehi

Submitted in partial fulfilment of the
requirements for the degree of

Master of Applied Science

at

Dalhousie University
Halifax, Nova Scotia
April 2011

DALHOUSIE UNIVERSITY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

The undersigned hereby certify that they have read and recommend to the Faculty of Graduate Studies for acceptance a thesis entitled "DESIGN AND DEVELOPMENT OF FUZZY LOGIC OPERATED MICROCONTROLLER BASED SMART MOTORIZED WHEELCHAIR" by Hamid Reza Moslehi in partial fulfillment of the requirements for the degree of Master of Applied Science.

Dated: April 15th 2011

Supervisor: _____

Readers: _____

_____

DALHOUSIE UNIVERSITY

DATE: April 15th 2011

AUTHOR: Hamid Reza Moslehi

TITLE: DESIGN AND DEVELOPMENT OF FUZZY LOGIC OPERATED MICROCONTROLLER BASED SMART MOTORIZED WHEELCHAIR

DEPARTMENT OR SCHOOL: Department of Electrical and Computer Engineering

DEGREE: M.A.Sc. CONVOCATION: October YEAR: 2011



_____

Signature of Author

# Table of Contents

# List of Tables

# List of Figures

# ABSTRACT

Independent mobility is critical to quality of life for people of all ages, and impaired mobility leaves one with both physical and mental disadvantages. Unfortunately, there are some individuals unable to operate an electric wheelchair due to physical, perceptual, or cognitive deficits. The prime objective of this research was to develop a prototype system which can provide mobility assistant to individuals who would otherwise find it difficult or impossible to operate a power wheelchair.

To accomplish this goal, a prototype system consisting of several components including an embedded microcontroller and multiple sensors has been designed which can be added to a standard power wheelchair and make it smart. The control system algorithm designed for this prototype model is based on the fuzzy logic control theory and its main purpose is to augment the user ability to navigate the wheelchair and will provide a safe and comfortable journey to the user.

The proposed system has been tested in simulation under different obstacle configurations and taking different routes and the results are presented to demonstrate the ability and validity of the designed system and algorithm in avoiding any possible collision. In addition to the Simulation tests, the prototype system has been built and implemented on an actual power wheelchair and the results were promising and a positive step toward a commercial smart wheelchair.

# ACKNOWLEDGEMENTS

First and foremost, I thank my parents, Majid Moslehi and Touba Ghafouri Nobahari, for the sacrifices made, and their unconditional support during this challenging period of my education. I am honoured by the investment they have made in my future.

The guidance and critical support I received from my Supervisor, Dr. Jason Gu, has been immeasurable. From the beginning, Dr. Gu has played a crucial role in the development of my thesis. I have been inspired and motivated by his encouragement.

It has been my distinct pleasure to have had the opportunity to benefit from the overall expertise of committee members Dr. M. El-Hawary, and Dr. Ya-Jun Pan.

I would also like to extend a sincere thank you to the faculty and staff of the Electrical and Computing Engineering Department. Special appreciation goes to the ECED secretaries, Selina Cajolais, and Nicole Smith, along with ECED technologists Chris Hill, Mark LeBlanc, and Ian McKenzie for the integral role they have played in the successful completion of my thesis.

My brothers Amir and Mojtaba, my colleagues in Control Systems and Robotics research lab, and my dear friends Mohsin Khan, Joey Fitzpatrick, and James Wilson, have made a lasting impression. You have been there during the difficult times, and I have benefited enormously from your friendship.

Hamid Reza Moslehi

# Chapter 1

# Introduction

In this chapter; we will talk about what a smart wheelchair is, what is the motivation behind it, previous research projects and the thesis project contributions.

## 1.1 What is a smart wheelchair?

A smart wheelchair is a motorized platform which typically consists of either a standard power wheelchair to which a computer and a collection of sensors have been added or a mobile robot base to which a seat has been attached and its purpose is to assist a user with a disability or anyone who is not able to operate a regular power wheelchair and to reduce or eliminate the user role of driving. It provides navigation assistance to the user in a number of different ways, such as assuring collision-free travel, aiding the performance of specific tasks, and autonomously transporting the user between locations [1].

Different types of sensors can be implemented on the smart wheelchairs such as sonars, infrared sensors or laser rangefinders. Sensors are being used to detect obstacles and modify the user intended drive path to avoid collision. Control system techniques such as path-planning, artificial reasoning, and behaviour based control are being used to augment or replace user control of the wheelchair [2].

# 1.2 Why a smart wheelchair?

Independent mobility is critical to individuals of any age. Children without safe and independent mobility are denied critical learning opportunities, which place them at a developmental disadvantage relative to their self-ambulating peers [3]. This will often produces a cycle of deprivation and reduced motivation that leads to learned helplessness [4].

Adults who lack an independent means of locomotion are less self-sufficient, which can manifest itself in a negative self-image and self-esteem [5]. Mobility limitations are the leading cause of functional limitations among adults, with an estimated prevalence of 40 per 1,000 persons age 18 to 44 and 188 per 1,000 at age 85 and older [6].

A lack of independent mobility at any age places additional obstacles in the pursuit of vocational and educational goals [7], and while the needs of many individuals with disabilities can be satisfied with power wheelchairs, some members of the disabled community find operating a standard power wheelchair difficult or impossible.

A clinical survey of 200 practicing clinicians indicated [8] that a significant percentage of people with disabilities have difficulty operating a power wheelchair. Significant survey results:

- Clinicians indicated that 9 to 10 percent of patients who receive power wheelchair training find it extremely difficult or impossible to use the wheelchair for activities of daily living.
- When asked specifically about steering and maneuvering tasks, the percentage of patients jumped to 40.
- Eighty-five percent of responding clinicians reported seeing some number of patients each year that cannot use a power wheelchair because they lack the requisite motor skill strength, or visual acuity. Of these clinicians, 32 percent (27 percent of all respondents) reported seeing at least as many patients who cannot use a power wheelchair as who can.
- Nearly half of patients unable to control a power wheelchair by conventional methods would benefit from an automated navigation system, according to the clinicians who treat them.

## 1.3 Related Research

Several researchers have used different methods and technologies to provide a collision-free journey for the users. As shown in Table 1, prototypes of several smart wheelchairs have been developed till today, but few have made the transition to a commercial product. A Canadian company located in Ottawa, Applied AI [9], sells smart wheelchair prototypes for use by researchers, but the system is not intended for use outside of a research lab. The CALL Center of the University of Edinburgh, Scotland, has developed the use of a wheelchair with bump sensors and the ability to follow tape tracks on the floor as part of a wheeled-mobility training program [10]. Their chair is sold in the United Kingdom, Australia, and USA by Smile Rehab Ltd.

If we classify the smart wheelchairs by the form factor [1]: Some of the designed systems are simply mobile robots with a chair on top (e.g., Mister Ed [11], VAHM [12]). And some are based on modified commercial wheelchairs (e.g., OMNI [13], SENARIO [14]) or the third group which are collections of components with an add-on unit that can be attached or removed from the wheelchairs (e.g., SWCS [7], Hephaestus [15]).

From control point of view, we can divide smart wheelchairs to three groups; 1) Autonomous, 2) Semi-Autonomous, 3) Autonomous and Semi-Autonomous mode. The first group [16], [17] operate very similar to autonomous robots; the user gives the destination and the smart wheelchair plans and executes a path to the target location. Autonomous systems typically require either a complete map of the area or some sort of modification to their environment. The disadvantage is that they can't avoid unplanned obstacles or navigate in unmapped environments. The second group of smart wheelchairs limit their assistance to collision avoidance and leave the planning and navigation duties to the user (e.g., NavChair [18], TinMan [19]). The advantage is that they aren't limited to modeled and planned environments and can operate in unmodified environments. However, they can be used by user who is able to effectively plan and navigate the wheelchair to a destination. A final group of smart wheelchairs offers both autonomous and semiautonomous navigation (e.g., VAHM [20], SENARIO [14]).

Different input methods have been used for smart wheelchairs ranging from the traditional input methods such as joystick and switches to the more advanced techniques such as touch screen interfaces [21] and voice recognition [22].

Table 1.1: Smart wheelchair system projects [1, 7]

| Smart Wheelchair | Sensors | Description |
|---|---|---|
| VAHM [12] | Sonar, Infrared, Dead Reckoning | Offers autonomous navigation based on an internal map and semiautonomous navigation in which the VAHM provides obstacle avoidance in the form of wall following and obstacle avoidance. |
| Mister Ed [11] | Sonar, Infrared, Bump | Robot base with chair on top. Subsumption architecture for control. |
| NavChair [18] | Sonar | Uses minimum vector field histogram (MVFH) and vector force field (VFF) as a control system. Prevents wheelchair from colliding with obstacles. |
| OMNI [13] | Sonar, Infrared, Bump, Dead Reckoning | Provides hierarchy of functionality: simple obstacle avoidance, task-specific operating mode (wall following, door passage), and autonomous navigation. |
| Hephaestus [15] | Sonar, Bump | Provides obstacle avoidance. Compatible with multiple brands of wheelchairs and does not require any modifications to underlying power wheelchair. Based on NavChair navigation system. |
| SWCS [7] | Sonar, Infrared, Bump | Prevents wheelchair from colliding with obstacles. Is compatible with multiple brands of wheelchairs and does not require any modifications to underlying power wheelchair. Rule-Base control system. |
| SENARIO [14] | Dead Reckoning, Sonar | Provides shared-control navigation obstacle avoidance and autonomous navigation based on internal map. Uses neural networks for localization, and distributed control architecture. |
| CALL [10] | Bump | Used as mobility training aid. Follows lines and backs up when it collides with an obstacle. |

| | | |
|---|---|---|
| TAO Applied AI Systems [9] | Infrared, Computer Vision | Uses subsumption architecture, from which several behaviors emerge. Including collision avoidance, door passage, wall following, and autonomous navigation. |
| Wheelesly [23] | Vision, Infrared, Sonar | Has exploring vision-based navigation assistance. I based on TinMan [19]. TinMan Original prototype used mechanical interface to wheelchair joystick, but subsequent prototypes integrated into control electronics of wheelchairs. Provides collision avoidance and autonomous navigation. |
| INRO [24] | GPS, Sonar, Drop-Off Detector | Provides autonomous navigation and wheelchair convoying between any two points. |
| CPWNS [25] | Vision, Dead Reckoning | User can automatically reproduce routes taught to system by manually driving wheelchair from starting point to goal point. Uses machine vision to identify landmarks in environment. No obstacle avoidance mode. |

# 1.4 Research Contribution

The proposed prototype system consists of several components that can be attached to the electrical motorized wheelchair and convert it into smart wheelchair, providing navigation assistant to the user and ensuring a collision-free journey. This system differs from previous systems in a number of respects, from the hardware (1.The type and number of sensors which have been used and 2.how they have been implemented on the wheelchair, 3.the computational hardware which is an embedded microcontroller, and last but not least 4.the interface between the components and the underlying wheelchair) to the software (The navigational software which is based on fuzzy logic control theory to avoid modeled and not-modeled obstacles).

The reason fuzzy logic control has been used for this system is because of the features [26] that make it an adequate tool for autonomous navigation problems where there is a need to cope with large amount of uncertainty that is inherent in natural environments. Fuzzy logic is not limited to a few feedback inputs and one or two control outputs, nor is it necessary to measure or compute rate-of-change parameters in order for it to be implemented. Any sensor data that provides some indication of a system's actions and reactions is sufficient. This allows the sensors to be inexpensive and imprecise, thus keeping the overall system cost and complexity low [27]. Using fuzzy logic control makes the implementation of the system much more practical which is the number one goal - to design a non-expensive and non-complex system which anybody can use, and will be able to augment the commercial wheelchairs and make them smart.

## 1.5 Thesis Outline

The second chapter talks about the mechanical design of the system. It gives a deep description of how the proposed system was designed, what components it is made of, components details and how the designed system has been integrated to the base power wheelchair.

Chapter 3 is about the control system architecture of the designed system. It talks about the control system components, criteria and how it was designed. Furthermore it will give a deep step by step overview about the fuzzy logic control algorithm used for this thesis project.

Chapter 4 talks about the designed Simulink system. It gives a deep description of how the simulation process was designed, why was it designed and the simulation results.

Chapter 5 gives a deeper view of the most challenging part of the project, the embedded microcontroller programming and implementation. It is divided by six sections, the USART connection which has been used for debugging proposes, the joystick interface which talks about how to interface a joystick with a microcontroller and how to use the ADC feature of the AVR Atmega644P, the sensory circuit and how the ultrasonic sensors have been programmed, the pulse width modulation which serves as the output of the system and last but not least the system implementation which talks about how the embedded microcontroller has been interfaced with the other components.

Finally, in chapter 6, the effectiveness of the prototype system in providing a collision free journey to the user is discussed along with future scope of work. Appendix contains MATLAB codes and the embedded microcontroller codes.

# Chapter 2

# Mechanical Design

The proposed system consists of sensors, computational hardware, and the control system software. The main purpose is to increase the user ability to navigate in an unmodified environment safely and collision-free. The proposed system is a semi-autonomous platform which will get the direction from the user, but be able to alter it, change the direction and speed of the wheelchair, and avoid any planned and unplanned object in its way. In this chapter the electric motorized wheelchair, proposed system hardware parts and how to interface it with the wheelchair will be discussed.

## 2.1 Proposed System Hardware Interface

In an unmodified commercial electric power wheelchair a joystick or any user interface system (switches, touchscreen displays, etc.) is linked to the wheelchair main controller (which acts as a motor controller) and the controller is connected to the two motors. Batteries are also connected to the controller, providing the necessary power for the system. The user selects the desired speed and direction using the joystick and the controller drives the motors based on the signal received by the joystick.

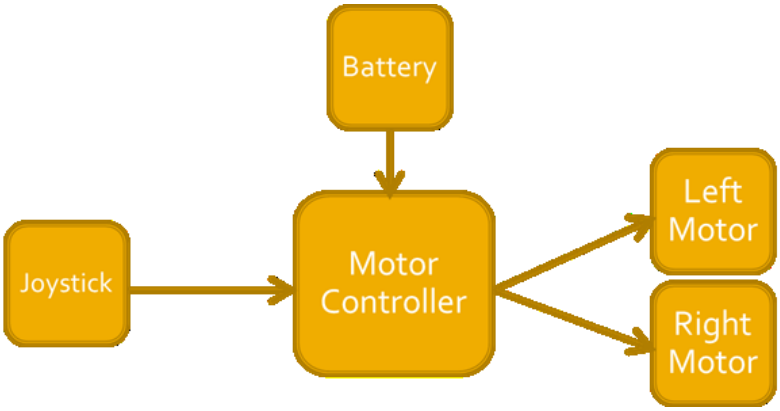


Figure 2.1: Electric power wheelchair block diagram

In the proposed system however, the connection between the joystick and the controller was interrupted by inserting an embedded microcontroller in between. There are also multiple sensors added and interfaced to the embedded microcontroller. The user chooses the direction and speed with the joystick, the joystick sends the information to the embedded microcontroller and the microcontroller checks it's surrounding with the help of the sensors and then corrects and alternates the joystick signals if necessary based on the control system algorithm before sending it to the motor controller. So if the user selects a direction and speed but there is an obstacle in the way the microcontroller will change the initial direction and speed so the wheelchair can avoid possible collisions.

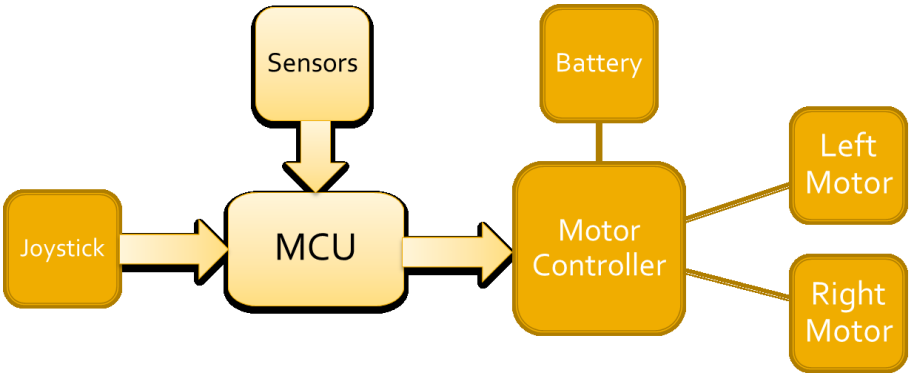


Figure 2.2: Proposed prototype system block diagram

# 2.2 Electric Wheelchair

The prototype system is mounted on an Everest and Jennings electric motorized wheelchair which was available in the Mobile Robotics laboratory. The wheelchair is controlled by an analog joystick that connects directly to the wheelchair controller. It also comes equipped with two 12V gel cell batteries connected in series and two 24V permanent magnet DC motors. The wheelchair was not in a good condition and was not working properly. After series of testing I find out that couple of repairs was necessary to make it work again. The main issues were with the wheelchair controller which needed to be replaced. More details will be discussed in the Motor Controller part. The other issue was the Lead-acid batteries which lost the ability to hold a charge because they were discharged for too long due to sulfation, the crystallization of lead sulfate. They were replaced by two Gel-Cell batteries each rating 12V and making up to 24V when connected in series, which is enough power to run the two 24V DC motors. The wheelchair wiring were also changed and modified. The wiring details will be discussed in the wiring section.

Figure 2.3: The not-modified electric wheelchair

## *2.2.1 Joystick Mechanism*

The Everest and Jennings wheelchair uses a standard analog joystick as the user interface. The analog joystick needs 5V to operate and will output two voltages which go to the wheelchair controller and represent the speed and direction of the chair. The handle moves a narrow rod that sits in two rotatable, slotted shafts. The shafts are connected each to a potentiometer. Tilting the stick forward and backward pivots the Y-axis shaft from side to side. Tilting it left to right pivots the X-axis shaft. When you move the stick diagonally, it pivots both shafts. Several springs center the stick when you let go of it. By moving the contact arm along the track, the resistance acting on the current flowing through the circuit will be increased or decreased [28].

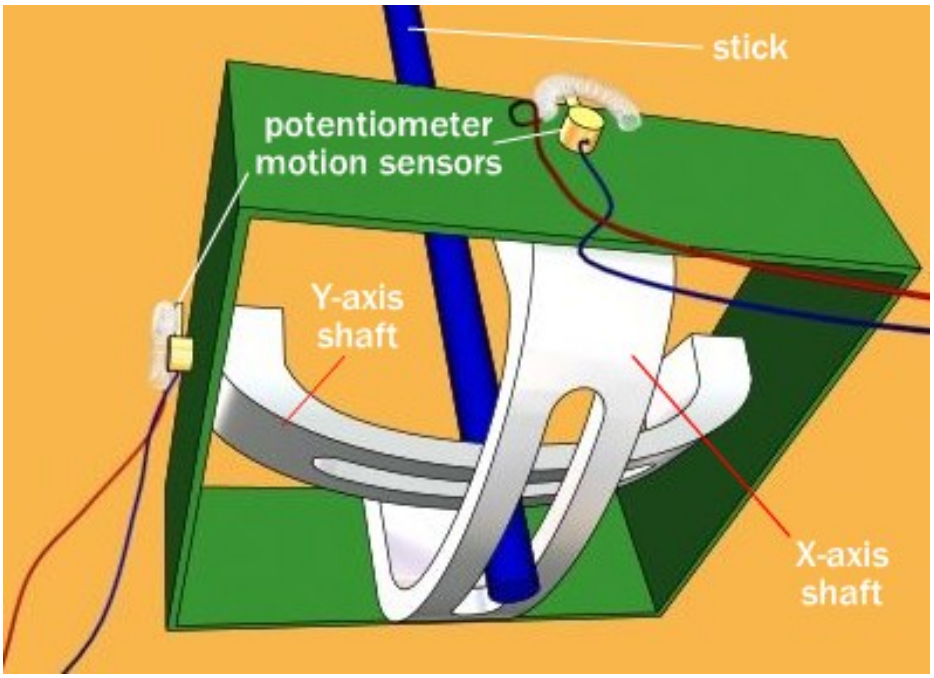


Figure 2.4: The analog joystick [28]

The joystick will output two voltages ranging from 0V to 5V. The analog voltage variations from "0V to 2V" indicate speed variation from maximum to minimum in clockwise direction and variations from "3V to 5V" gives the speed variation from minimum to maximum in counter clockwise direction. The slot of variations "2.02 to 2.98" represents the no operation state.

Table 2.1: Variations of motor speed and direction with reference to joystick voltage

| Analog Voltage | Speed | Motor Direction |
|---|---|---|
| **0.00 – 2.00** | Max - Min | Clock Wise |
| **2.02 – 2.98** | Zero | OFF |
| **3.00 – 5.00** | Min - Max | Counter Clock Wise |

The connection between the joystick and motor controller was interrupted by cutting the wires that send the X-Axis and Y-Axis states of the joystick to the motor controller by two. Then one end of the wires which comes from the joystick were fed to the two ADC (Analog to Digital Convertor) pins of the microcontroller and the other end of the wire which goes to the motor controller were fed to the two other pins of the microcontroller that outputs PWM (Pulse Width Modulation) signals.

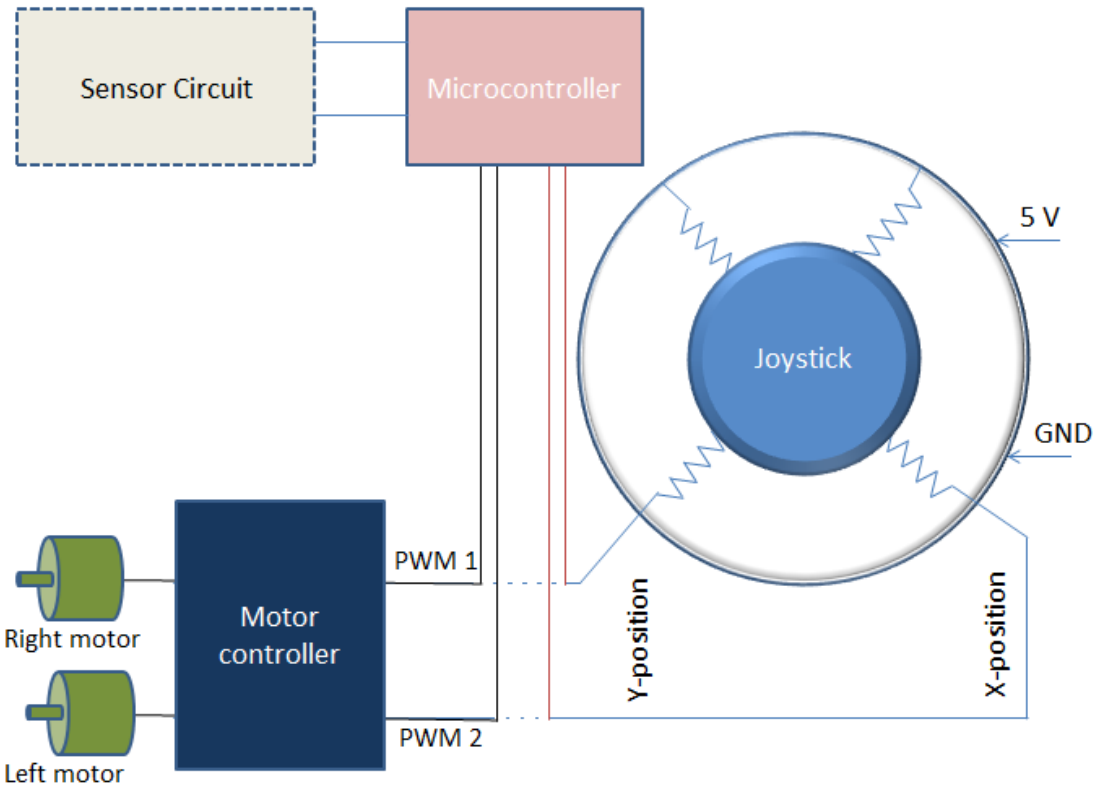


Figure 2.5: Joystick interfacing with the system

## *2.2.2 Motor Controller*

In an electric wheelchair the controller acts as the command center for the wheelchair and is responsible for amperage, speed control and the maintenance of straight line propelling and turning control when the chair is in use. The controller derives its energy from two rechargeable batteries, and the controller gives the individual the ability to move the chair forward, backward, left and right and to make any variation of turns up to 360 degrees.

The original controller however was not working and it had to be replaced. For replacing the controller with a new one several considerations had to be made:

1. The first consideration is the motor's nominal voltage. The wheelchair has two DC motors which each operate to the maximum 24V capacity so the motor controller must be powerful enough to provide 24V to each motor [29].
2. The next consideration is the continuous current the controller needs to supply. I had to find a controller that was able to provide current equal or above the motor's continuous current consumption under load.

   The wheelchair DC motors need 250W power when used on a flat field but the power consumption jumps to 450W when climbing over curbs and up steeps.

(2.1)

$$P = VI$$

$P$ – Power (W)

$V$ – Voltage (V)

$I$ – Current (A)

Based on the above power formula the current each motor needs to operate is 10.41A in the first case and 19A when climbing.

The stall current also has been measured by holding the armature and supplying a small amount of voltage to the motor and since the motors are each a powerful 24V, it's not possible to hold the armature when more voltage is applied so maximum voltages of 1V and 1.6V has been tried (Table ). Maximum current draw is at zero RPM (stall) where the motor is unable to turn [30]. Maximum current draw is referred to as 'stall current' or

'stall amps'. Knowing the stall current of a motor is valuable when planning for worst-case design parameters, although a power wheelchair or any well designed robotic platform will rarely encounter a stall condition.

Table 2.2: DC motors measured stall current

| Applied Voltage | Measured Stall Current | Static Resistance($R = \frac{V}{I}$) |
|---|---|---|
| **1V** | **2A** | **0.5Ω** |
| **1.6V** | **4A** | **0.4Ω** |

One way to estimate the stall current at full operation voltage is to multiply the reading obtained with smaller voltages by (Full Voltage/ Tested Voltage ).

(2.2)

$$2\text{A (Measured Current)} \times \frac{24V(Max\ Voltage)}{1V(Tested\ Voltage)} = 48\ \text{A}$$

The motors have been further tested in the laboratory (Dalhousie research lab. C102) to find out the characteristics and to have a better understanding of their power requirements. The DC motor has been supplied with the 6V, 12V, and 24V and the current and speed has been measured using a multimeter and a tachometer (Table 2.3).

Table 2.3: DC motors measured speed

| Applied Voltage | Current | Revolutions Per Minute |
|---|---|---|
| **24V** | **1.5A** | **350rpm** |
| **12V** | **1.3A** | **200rpm** |
| **6V** | **1A** | **100rpm** |

As it can be seen from the above table, the maximum rotational speed of the motor is 350 rpm which is proportional to 36.65 rad/s. knowing the maximum angular speed is

especially useful when designing the control system algorithm. The frequency of rotation can be converted to the angular speed using the following formula:

(2.3)

$$W\left(\frac{rad}{s}\right) = \frac{(RPM)\times 2 \times Pi}{60}$$


Figure 2.6: DC motors in testing

3. The control method is another important consideration. Control methods include analogue voltage, $I^2C$, PWM, R/C, UART (a.k.a. serial). The new controller will be interfaced with the microcontroller and will use PWM to communicate, one PWM channel per motor.
4. The final consideration is a choice between single and dual motor controllers. A dual DC motor controller is preferred since it can control the speed and direction of two DC motors independently which will save money and time.

After researching the motors behaviours and future system expectations, the Dimension Engineering Sabertooth dual 25A regenerative motor driver has been chosen to replace the original controller. Some of the key features of this controller are as following [31]:

- Dual motor controller, 25A (6-24V nominal, 30V absolute max)
  Can supply two brushed DC motors with up to 25A continuously. Peak currents of 50A per channel are achievable for a few seconds. Sabertooth has independent speed + direction operating modes, making it the ideal driver for differential drive.
- Analog, R/C, simplified serial and packetized serial interfaces (TTL)
  Sabertooth is able to control two motors with: analog voltage, radio control, serial and packetized serial.
- Synchronous regenerative drive
  The regenerative topology means that the batteries get recharged whenever the robot or the wheelchair slows down or goes reverse.
- Ultrasonic switching frequency (32KHz)
  Sabertooth's transistors are switched at ultrasonic speeds (32 KHz) for silent operation.
- Thermal and overcurrent protection
  Overcurrent and thermal protection means we will never have to worry about killing the driver with accidental stalls or by hooking up a motor.
- Lithium protection mode
  The lithium cut-off mode allows the controller to operate safely with lithium ion and lithium polymer battery packs - the highest energy density batteries available.
- suitable for high powered robots - up to 100lbs in combat or 300lbs for general purpose robotics.
- It’s able to make very fast stops and reverses
- It has a built in 5V BEC that can provide power to a microcontroller or R/C receiver.

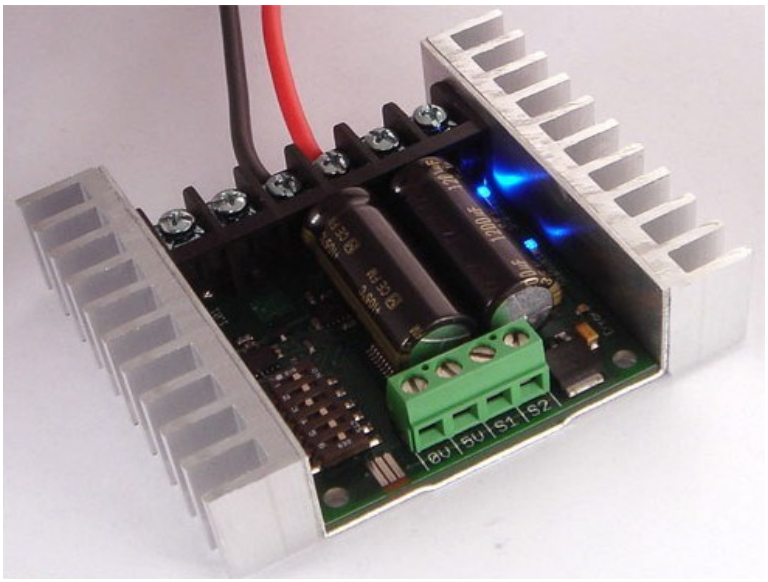


Figure 2.7: The Dimension Engineering Sabertooth dual 25A regenerative motor driver

Figure 2.8: Wheelchair original controller

# 2.3 Wheelchair Mechanism

Figure 2.9: 3D model of the electric wheelchair

The electric wheelchair used in this research is a rear-wheel drive system powered by two permanent magnet dc motors. Rear-wheel drive is the traditional and most popular power wheelchair style. They are generally faster than front-wheel models, but provide poor turning capabilities in comparison to front-wheel drive wheelchairs and mid-wheel drive wheelchairs models [32]. It also uses belt to transform power from the motors to the wheels. Electric/power wheelchairs make use of either gears or belts, or sometimes both. Power wheelchairs with belt drives are usually quiet, but tend to be high-maintenance. Gear drives are fairly low-maintenance, but tend to wear out quickly and getting noisy in the process. The wheelchair used in this research is a low-end power model which has a light frame that is suitable for indoor use, but tends to crack, the front forks bend and motors die when they are abused outside. In the following pages the mechanism and kinematics of the wheelchair will be discusses more deeply.

The wheelchair will have some torque loss from the friction of the wheels, bearing and rolling [33] which have to be considered when designing the system.

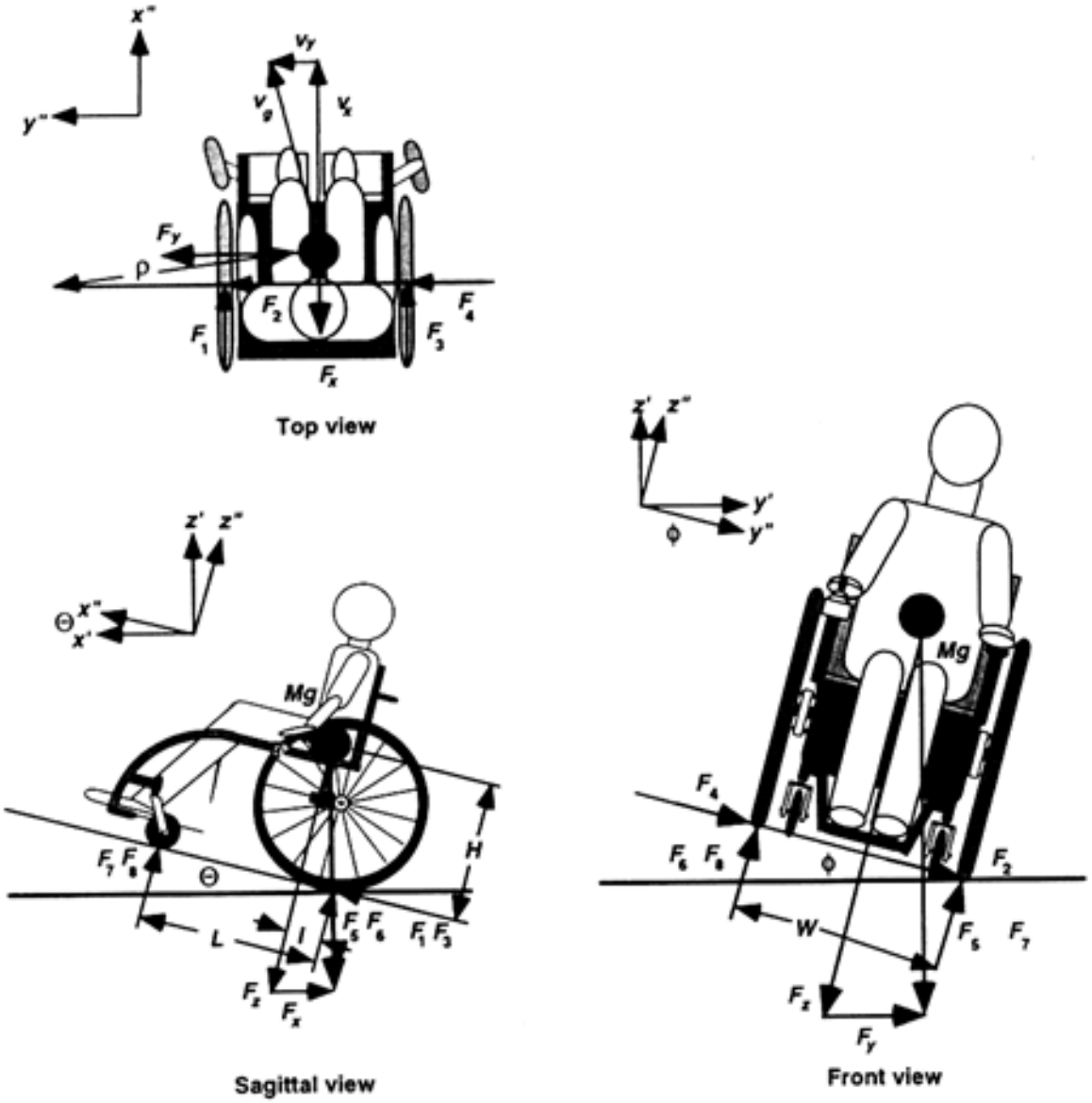


Figure 2.10: Free-body diagram of a powered wheelchair and rider on an inclined surface [33]

$\theta$ – Pitch (slope) angle;

$\phi$ – Incline angle

*M*- Mass of the wheelchair/rider system

Acceleration of the wheelchair/rider system along the *x* and *y* axes:

(2.4)

$$\frac{dv_x}{dt} = \frac{(\frac{dv_r}{dt} + \frac{dv_l}{dt})}{2} - \frac{l(v_r - v_l)^2}{W^2}$$

$$\frac{dv_y}{dt} = \frac{l(\frac{dv_r}{dt} - \frac{dv_l}{dt})}{W} - \frac{W(v_r - v_l)^2}{2}$$

$v_r$ – Linear velocity of the right wheel

$v_l$ - Linear velocity of the left wheel

$l$ – Distance between the center of mass and the rear axles

$W$ – Width of wheelchair between the rear wheels

Forces, acting at the center of the mass (M) of the wheelchair/rider system:

(2.5)

$$F_x = \frac{M\ g\ tan\theta}{(1 + tan^2\theta + tan^2\phi)^{\frac{1}{2}}}$$

$$F_y = \frac{M\ g\ tan\phi}{(1 + tan^2\theta + tan^2\phi)^{\frac{1}{2}}}$$

$$F_z = \frac{M\ g}{(1 + tan^2\theta + tan^2\phi)^{\frac{1}{2}}}$$

g - Acceleration due to gravity.

Linear acceleration of the left and the right wheels:

(2.6)

$$\frac{dv_L}{dt} = \frac{(A-B)}{2}$$

$$\frac{dv_R}{dt} = \frac{(A+B)}{2}$$

Wheelchair angular acceleration about the "z" axis:

(2.7)

$$w_z = 180\,(v_R - v_L)\,(\frac{W}{\Pi})$$

Where:

(2.8)

$$A = \frac{1}{\left[\frac{J_w + J_m K_g^2}{r} + \frac{M_r}{2}\,(\frac{Hf}{L} + 1)\right]\left[K_t K_g (i_{mR} + i_{mL}) - \beta_c\left(\left(\frac{V_R}{r}\right)^{\frac{2}{3}} + \left(\frac{V_L}{r}\right)^{\frac{2}{3}}\right) - \frac{rfF_z(L-l)}{L} - r\left(\frac{fH}{L}+1\right)\left(F_x - \frac{Ml}{W^2}(v_R - v_L)^2\right)\right]}$$

(2.9)

$$B = \frac{1}{\left[\frac{J_w + J_m K_g^2}{r} + \frac{2r}{W^2}\,(HfMl + I_z + Ml^2)\right]\left[K_t K_g (i_{mR} - i_{mL}) - \beta_c\left(\left(\frac{V_R}{r}\right)^{\frac{2}{3}} - \left(\frac{V_L}{r}\right)^{\frac{2}{3}}\right) - \frac{2r}{W}\,(Hf+1)\left(\frac{MW}{2}\,(v_R^2 - v_L^2) - F_y\right)\right]}$$

### *2.3.1 Permanent Magnets DC Motors Models*

A permanent magnet dc motor is a mechanism which converts electrical power to mechanical power via magnetic coupling. The electrical power is provided by a voltage source, while the mechanical power is provided by a spinning rotor [34].

(2.10)

$$V_a = i_a R_a + K_v w_m + L_a \frac{di_a}{dt}$$

$$J_m \frac{dw_m}{dt} = i_a K_t - T_m$$

$V_a$ – armature voltage;

$i_a$ – armature current;

$R_a$ – armature resistance;

$K_v$ – motor voltage constant;

$w_m$ – motor angular velocity;

$L_a$ – armature inductance;

$J_m$ – motor inertia;

$K_t$ – motor torque constant;

$T_m$ – motor torque.

## 2.3.2 Kinematic Equations of Wheelchair

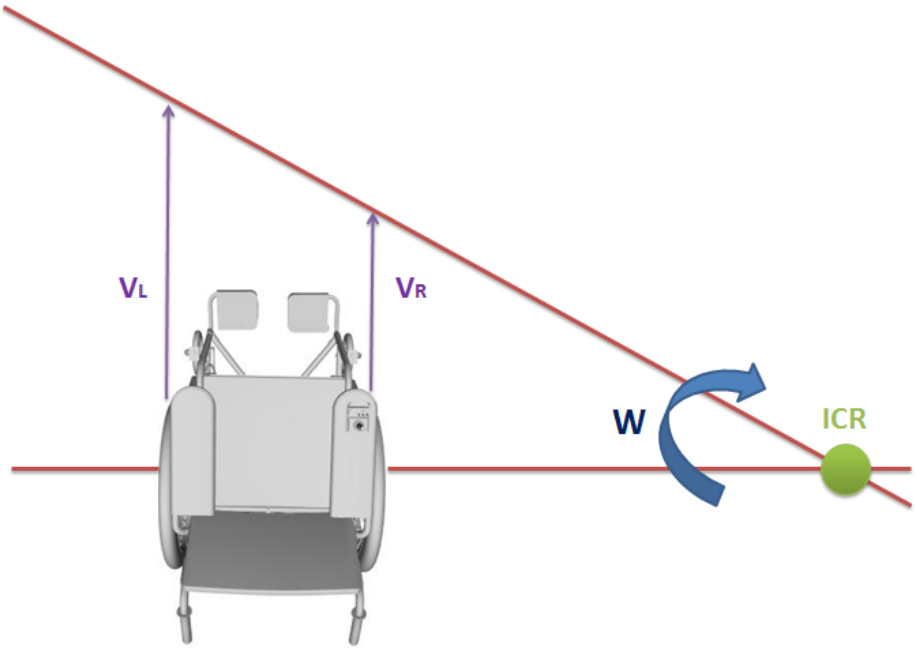


Figure 2.11: Instantaneous center of rotation (ICR)

The wheelchair can be driven to any position by the rear wheels velocities. It can be coordinated by the (x, y) the position of the wheelchair center in the Cartesian space and φ the heading angle of the wheelchair from the x-axis as shown in Figures 2.11 and 2.12.

If $W_{rw}$ and $W_{lw}$ are the actuated angular velocities on the right and left wheel respectively, then the right and left wheel linear speeds without slipping are:

(2.11)

$$V_i = r \times W_{iw}$$

And the linear and angular velocities of the wheelchair are:

(2.12)

$$V_t = \frac{(V_r + V_l)}{2}$$

$$W = \frac{(V_r - V_l)}{2b}$$

Where r is the wheel radius, b is the distance from the point C to each wheel, $V_t$ is the advance speed of the wheelchair without slipping and W is the angular velocity of the wheelchair.

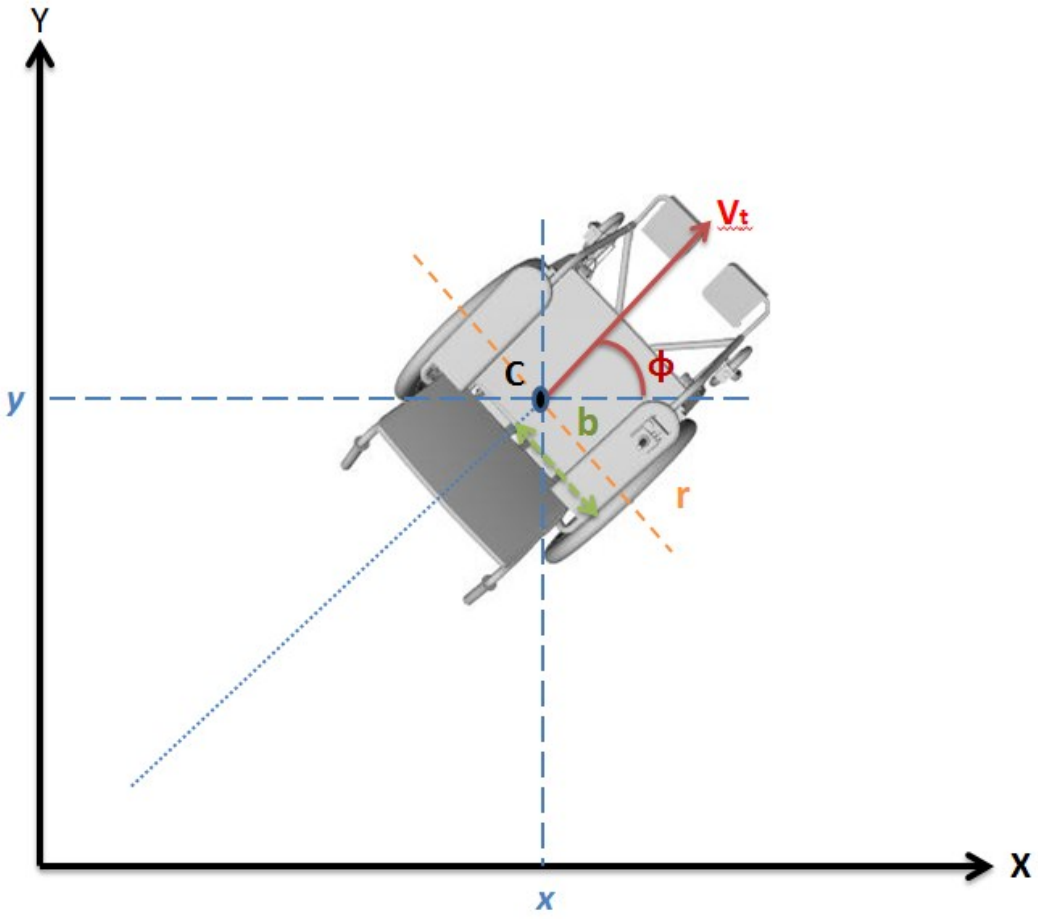


Figure 2.12: The geometry of the wheelchair

Using the above relationships the absolute velocity of point C is related to $V_t$ and W by the following kinematics equations [35]:

(2.13)

$$x' = v_t \cos \varphi$$

$$y' = v_t \sin \varphi$$

$$\varphi' = w$$

It should be noted that the center of rotation is the intersection point of the axis between the two rear wheels and the axis linking the two endpoints of wheel speed vectors as shown in Figure 2.12.

# 2.4 Embedded Microcontroller

A microcontroller is a computing device capable of executing a program and is often referred to as the "brain" or "control center" in a robot since it is usually responsible for all computations, decision making, and communications [36].

In order to interact with the outside world, a microcontroller possesses a series of pins (electrical signal connections) that can be turned HIGH (1/ON) or LOW (0/OFF) through programming instructions. These pins can also be used to read electrical signals (coming from sensors or other devices) and tell whether they are HIGH or LOW. Most modern microcontrollers can also measure analogue voltage signals (i.e. signals that can have a full range of values instead of just two well defined states) through the use of an Analogue to Digital Converter (ADC). By using the ADC, a microcontroller can assign a numerical value to an analogue voltage that is neither HIGH nor LOW.

Although microcontrollers can seem rather limited at first glance, many complex actions can be achieved by setting the pins HIGH and LOW in a clever way like creating very complex algorithms such as advanced vision processing and intelligent behaviours.

Microcontrollers can be used to control other electrical devices such as actuators (when connected to motor controllers), storage devices (such as SD cards), Wi-Fi or Bluetooth interfaces, etc. As a consequence of this incredible versatility, microcontrollers can be found in everyday products. Practically every home appliance or electronic device uses at least one (often many) microcontroller. For instance TV sets, washing machines, remote controls, telephones, watches, microwave ovens, and now robots require these little devices to operate.

Unlike microprocessors (e.g. the CPU in personal computers), a microcontroller does not require peripherals such as external RAM or external storage devices to operate. This means that although microcontrollers can be less powerful than their PC counterpart, developing circuits and products based on microcontrollers is much simpler and less expensive since very few additional hardware components are required.

Many microcontrollers readily support the most popular communication protocols such as UART (a.k.a. serial or RS232), SPI and $I^2C$.This feature is incredibly useful when communicating with other devices such as computers, advanced sensors, or other microcontrollers.

Analogue-to-digital converters (ADC) are used to translate analogue voltage signals to a digital number proportional to the magnitude of the voltage, this number can then be used in the microcontroller program. In order to output an intermediate amount of power different from HIGH and LOW, some microcontrollers are able to use pulse-width modulation (PWM). For example this method makes it possible to smoothly dim an LED.

Finally, some microcontrollers integrate a voltage regulator in their development boards. This is rather convenient since it allows the microcontroller to be powered by a wide range of voltages that do not require you to provide the exact operating voltage required. This also allows it to readily power sensors and other accessories without requiring an external regulated power source.

So an embedded microcontroller was chosen as the computational hardware since it provides much better real-time operation than most of the operation systems. It is also relatively cheap compared to a laptop or a PC, it consumes less power, it is much lighter in weight, and doesn't block the user's view if used on a wheelchair. Although a computer is much easier for debugging or optimization purposes, and is easier to interface with the sensors and joystick, ultimately the embedded microcontroller is a better practical choice.

The embedded microcontroller used in this project is an AVR Atmega644P based controller supplied by the university (Figure 2.13). The Atmega644P is an 8-bit microcontroller with 64 Kbyte flash memory, 2 Kbyte EEPROM and 4 Kbyte RAM [37]. C programming language has been used to code the microcontroller along with the AVR Studio 4 and WINAVR compiler. The microcontroller was programed using bootloader software. The programming details are discussed in chapter 5.

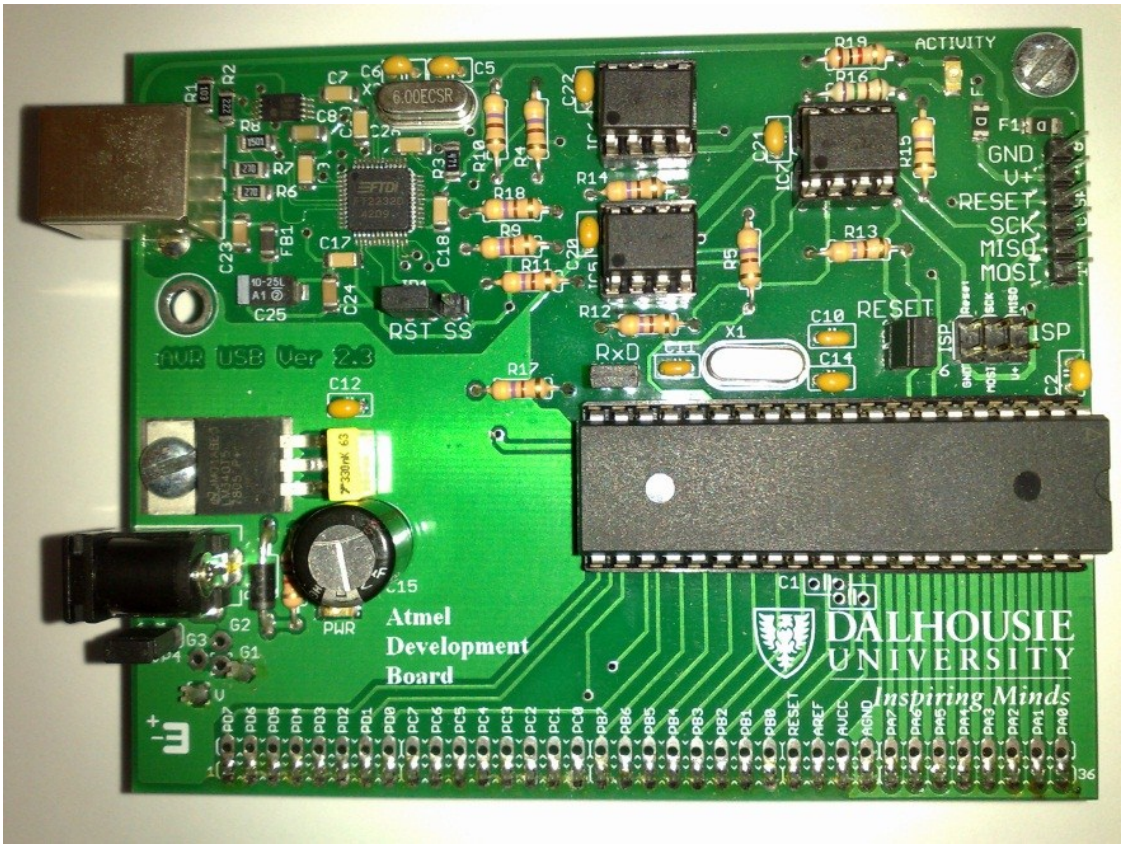


Figure 2.13: The Atmel embedded microcontroller board

# 2.5 Sensory Circuit

Sensors act as the mobile robot eyes and sensing is the key requirement for any but the simplest mobile behaviour [38]. Sensors and sensing algorithms are required for a robot in order to know where it is, how it got there and where it should go.

Figure 2.14: Devantech SRF05 sensor

Multiple ultrasonic (sonar) sensors have been used for the system sensory circuit. Ultrasonic sensor works by emitting a high frequency sound wave (Ping) and evaluating the received echo. They determine the distance from the object by calculating the time interval between sending the signal and receiving the echo.

By accurately measuring the time from the start of the ping until the echo returns back to the sensor, the distance to the nearest object can be easily calculated. Sound travels at 1116.4 feet/second (340.29 meters/second) at sea level. The distance to the nearest object can be calculated by dividing the elapsed time (time between issuing the sound and hearing the echo) by twice the speed of sound, as follows [39]:

(2.14)

$$Time\ Passes\ /\ (Speed\ of\ Sound\ x\ 2)\ =\ Distance\ from\ Object$$

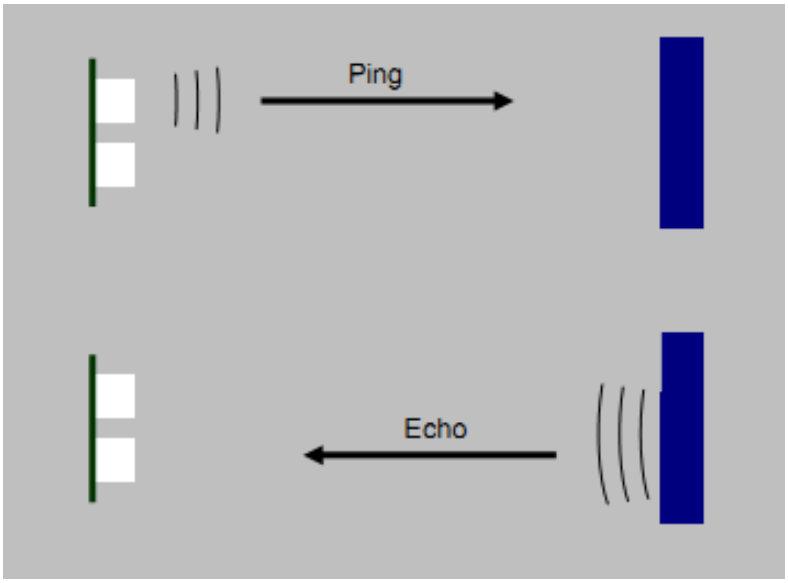


Figure 2.15: Sonar Ping and Echo [39]

The reason for dividing by twice the speed of sound is that the distance to the object is only half the distance the sound wave actually travels. The sound wave must travel to the object and back to the sensor in order for the sensor to hear the echo.

The sonar sensor used in this project is the Devantech SRF05 ultrasonic range finder. It has a 3-4cm resolution and it can detect objrcts from min 1cm to max 4m. The dimension is 43mm x 20mm x 17mm. More details about the sonar sensor operation will be discussed in the chapter 5, MCU Programming.

The reasons ultrasonic sensors have been chosen for the system are because they are lightweight, do not occupy a lot of space, easy to interface and relatively cheap. They are also very accurate when the sound wave hits the target at the right angle. Although it will be inaccurate if the target is made of sound-absorbent materials or has a smooth surface. Another problem is that the sound wave can get lost or bounce between walls multiple times before returning to the sensor [40].

Eight sonar sensors have been placed on the wheelchair (Figure 2.16). The number of sonars sensors used for the prototype was limited by the cost of adding new sensors and the performance and capacity of the embedded microcontroller but have been tried the best to decrease blind spots as much as possible with the help of the control system algorithm and

placement of the sensors. The system has been simulated multiple times to find the perfect number and position of the sensors.

The biggest disadvantage with using multiple sonars is the “crosstalk” effect in which one sensor sends the signal but another sensor receives the echo. For solving this problem the sonars were mounted in a way so they will be pointed at different angles and outside of viewing angle of each other. In addition to that the sensors will be fired up two by two and at different time slots to reduce erroneous readings from sensor crosstalk.

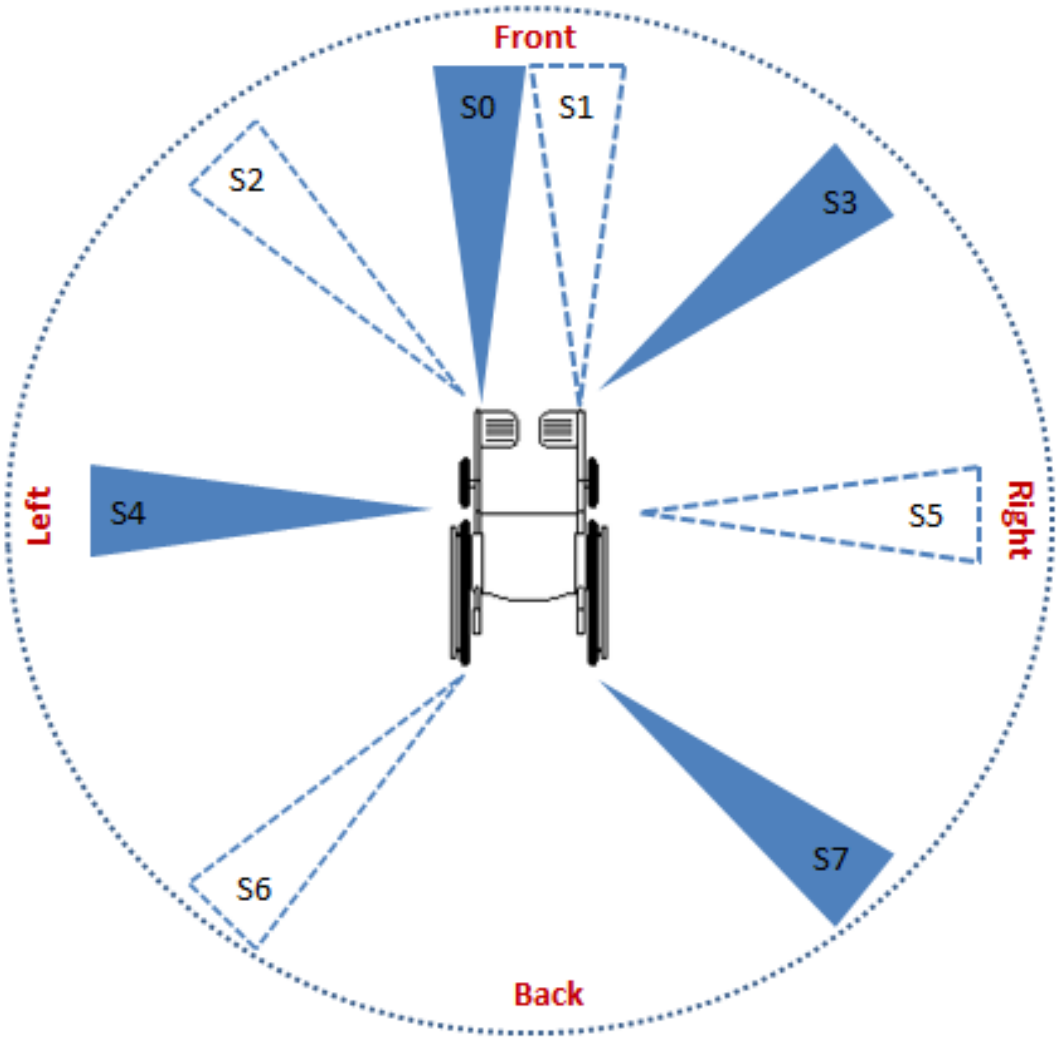


Figure 2.16: Sonar sensors placement on the wheelchair. The triangle’s represent the cones emitted by each sonar.

# Chapter 3

# Control System Architecture

A control system is a device or set of devices to manage, command, direct or regulate the behaviour of other devices or systems [41].There are two common classes of control systems, with many variations and combinations: logic or sequential controls, and feedback or linear controls. There is also fuzzy logic, which attempts to combine some of the design simplicity of logic with the utility of linear control.

The main purpose of the control system in this project is to augment the user ability to drive the wheelchair and help the user to navigate through the indoor space with maximum comfort and minimum risk of accident.

## 3.1 Fuzzy Logic Control

Fuzzy logic control has been used to design the controller for this smart motorized wheelchair. Fuzzy logic control is derived from the fuzzy logic and fuzzy set theory that were introduced in 1965 by Lotfi A. Zadeh [42]. Since then the theory has been applied in wide range of fields such as economics, data analysis, engineering and other arias that involve a high level of uncertainty,

complexity, or nonlinearity [43] and in wide range of applications from wash-machines to digital cameras to automated space docking.

Fuzzy logic is a form of many valued logic that deals with type of reasoning that is robust and approximate rather than brittle and exact. In contrast with "crisp logic", where binary sets have two valued logic (0 or 1), fuzzy logic variables may have a truth value that ranges in degree between 0 and 1 [44]. A form of reasoning, derived from fuzzy set theory, doesn't need to be exactly zero (false) or one (true), but rather can be zero, one, or any value in between. Fuzzy logic is designed for situations where information is inexact and traditional digital on/off decisions are not possible. It divides data into vague categories such as "hot", "medium" and "cold" called linguistic variables [45].

The fuzzy logic control has features [26] that make it an adequate tool for autonomous navigation problems where there is a need to cope with large amount of uncertainty that is inherent in natural environments. Fuzzy logic is not limited to a few feedback inputs and one or two control outputs, nor is it necessary to measure or compute rate-of-change parameters in order for it to be implemented. Any sensor data that provides some indication of a system's actions and reactions is sufficient. This allows the sensors to be inexpensive and imprecise, thus keeping the overall system cost and complexity low [27] and as a result, using fuzzy logic control makes the implementation of the semi-autonomous wheelchair system much more practical.

The fuzzy control is known as a robust controller since it doesn't require precise, noise-free inputs, and for that matter has been one of the favourite high level behaviour control choices for complex and nonlinear systems where lot of uncertainty is involved [46]. And new inputs and outputs can easily be interfaced to a fuzzy system by generating the appropriate membership functions and equations.

Fuzzy logic differs from conventional control methods because incorporates a simple, rule-based IF X AND Y THEN Z approach to a solving control problem rather than attempting to model a system mathematically. The fuzzy logic model is empirically-based, relying on an operator's experience rather than their technical understanding of the system [27].

The fuzzy control for this project has been designed using MATLAB fuzzy logic toolbox. And since designing a fuzzy controller relies heavily on the designer experience and knowledge of the system the designed fuzzy logic controller system has been tested in the simulation environment

(MATLAB Simulink) along with the rest of the system for debugging and optimization purposes before moving to the hardware mode. Simulation results are discussed in chapter 4.

## 3.2 How to design a Fuzzy Logic Control?

1) First step is to define the control objectives and criteria: What do I want to control? What do I have to do to control the system? What kind of response do I need? What are the possible system failure modes?
2) Second step is to determine the input and output relationships and choose a minimum number of variables for input to the fuzzy logic engine.
3) Using the rule-based structure of fuzzy logic, break the control problem down into a series of IF X AND Y THEN Z rules that define the desired system output response for given system input conditions. The number and complexity of rules depends on the number of input parameters that are to be processed and the number fuzzy variables associated with each parameter.
4) Creating fuzzy logic membership functions that define the meaning and values of Input/output terms used in the rules.
5) Creating the necessary pre- and post-processing fuzzy logic routines or program the rules into the fuzzy logic engine.
6) Test the system, evaluate the results, tune the rules and membership functions, and retest until satisfactory results are obtained.

# 3.3 Input/output Fuzzification

The first step toward designing the fuzzy control algorithm is to fuzzify input and output variables. In fuzzification process we need to transform the real/crisp valued variables into the fuzzy sets. Fuzzy sets are sets whose elements have degrees of membership (DOM) [47].There are different ways to map the data into fuzzy sets such as Gaussian membership function, singleton membership function, triangular membership function, etc.

The membership function is a graphical representation of the magnitude of participation of each input. It associates a weighting with each of the inputs that are processed, define functional overlap between inputs, and ultimately determines an output response. The rules use the input membership values as weighting factors to determine their influence on the fuzzy output sets of the final output conclusion. Once the functions are inferred, scaled, and combined, they are defuzzified into a crisp output which drives the system [27].

The characteristics of the membership function are defined by three properties [48]. They are:

1. Core: If the region of universe is characterized by full membership (1) in the set A then this gives the core of the membership function of fuzzy at A. The elements, which have the membership function as 1, are the elements of the core ($\mu A(x) = 1$). Note that the membership can take value between 0 and 1.
2. Support: If the region of universe is characterized by nonzero membership in the set A, this defines the support of a membership function for fuzzy set A. The support has the elements whose membership is greater than 0 ($\mu A(x) > 0$).
3. Boundary: If the region of universe has a nonzero membership but not full membership, this defines the boundary of a membership; this defines the boundary of a membership function for fuzzy set A. The boundary has the elements whose membership is between 0 and 1, ($0 < \mu A(x) < 1$).

There is a unique membership function associated with each input parameter. The membership functions associate a weighting factor with values of each input and the effective rules. These weighting factors determine the degree of influence or degree of membership (DOM) each active

rule has. By computing the logical product of the membership weights for each active rule, a set of fuzzy output response magnitudes are produced. All that remains is to combine and defuzzify these output responses.

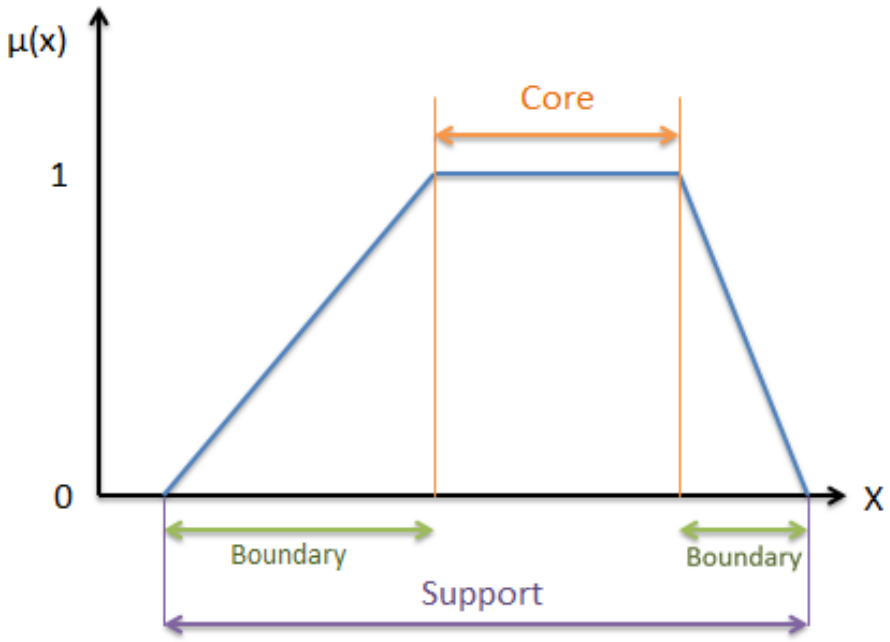


Figure 3.1: The fuzzy membership function

### *3.3.1 Input Membership Functions*

The proposed system has nine inputs consisting of eight sonar sensors distance inputs and one joystick direction input. The fuzzifier converts each crisp input values to linguistic variables which are described by fuzzy sets. In our system each input is fuzzified using triangular membership function method. The main reason for using the triangular membership function is that it will be easier when transferring them to the microcontroller and embedded language. Triangular functions are defined using the following equation:

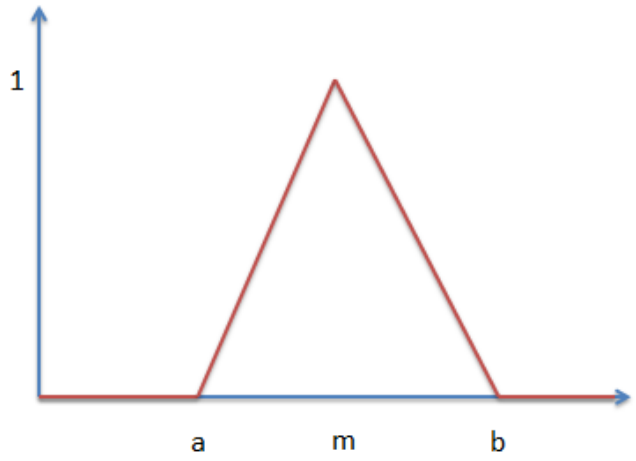


Figure 3.2: Triangular membership function

(3.1)

$$\mu_A(x) = \begin{cases} 0, & x \leq 0 \\ \dfrac{x-a}{m-a}, & a < x \leq m \\ \dfrac{b-x}{b-m}, & m < x < b \\ 0, & x \geq b \end{cases}$$

Triangular function are defined by a lower limit a, an upper limit b, and a value m, where a < m < b.

The joystick is fuzzified to five membership functions each representing a direction selected by the user. The linguistic terms used in this fuzification are rearL (Rear left direction), left, front, right, and rearR (Rear right direction). The input output membership functions have been designed from the experience, other similar projects, and mostly the simulation results. Different membership functions have been tested in the Simulink environment and the best has been chosen based on the performance of the system under different simulation testing's.

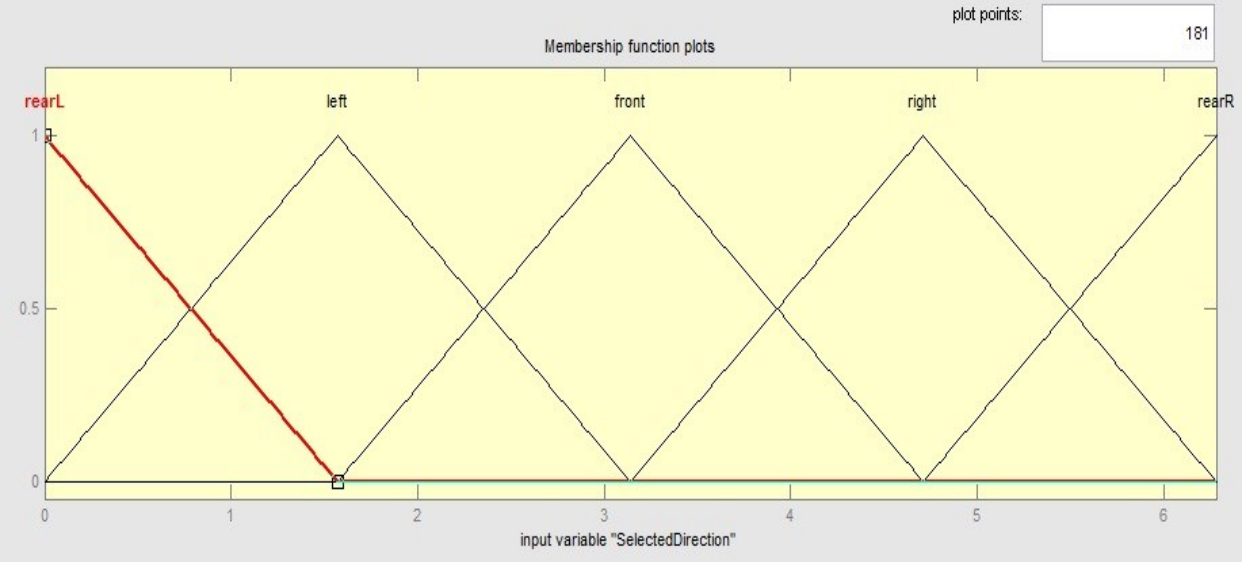


Figure 3.3: The joystick membership functions

Each sonar sensor is fuzzified to three membership functions. The membership functions are designed based on the distance between the wheelchair and obstacles detected by the sonars. *For example* if the sonars detect an obstacle in the three meters distance then based on the membership function ( Figure 3.4), the obstacle is in the far category and will have less urgency than an obstacle that is in half meter distance. The linguistic terms used in this fuzzification are near, middle, and far. The X-axis of the graph represents the distance in meters.

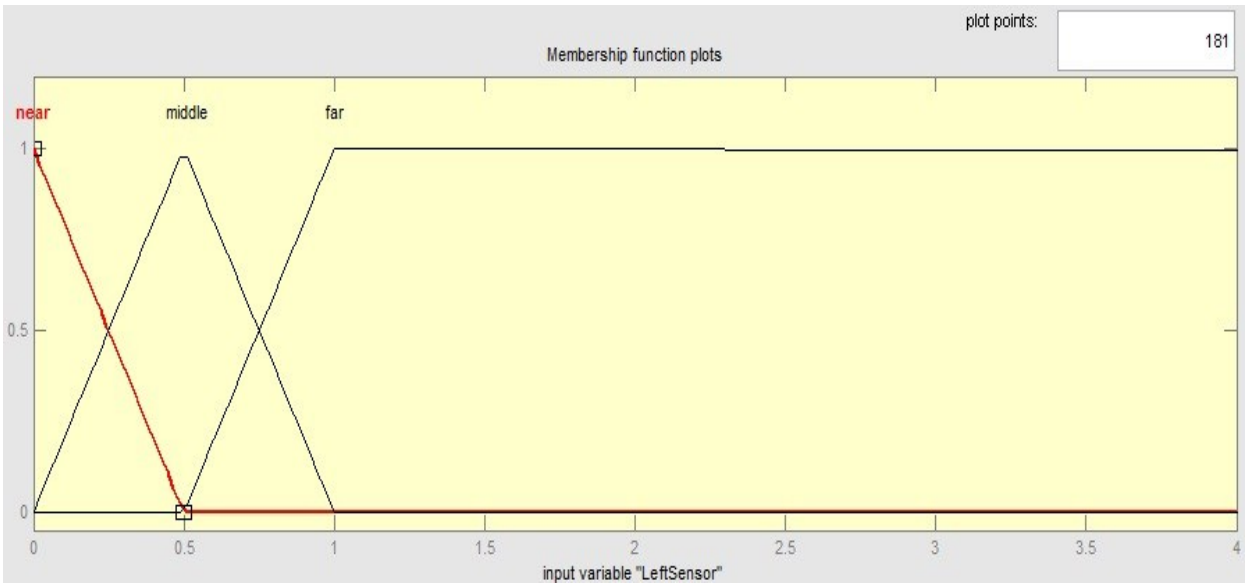


Figure 3.4: Sonar sensors membership functions

## *3.3.2 Output Membership Functions*

The embedded microcontroller outputs are two PWM channels which will be fed into the wheelchair motor driver. The two PWM channels will indicate the direction and speed of the wheelchair. That's why the fuzzy controller output has been designed based on the speed and direction. The output direction membership function is the same as the input direction membership function.

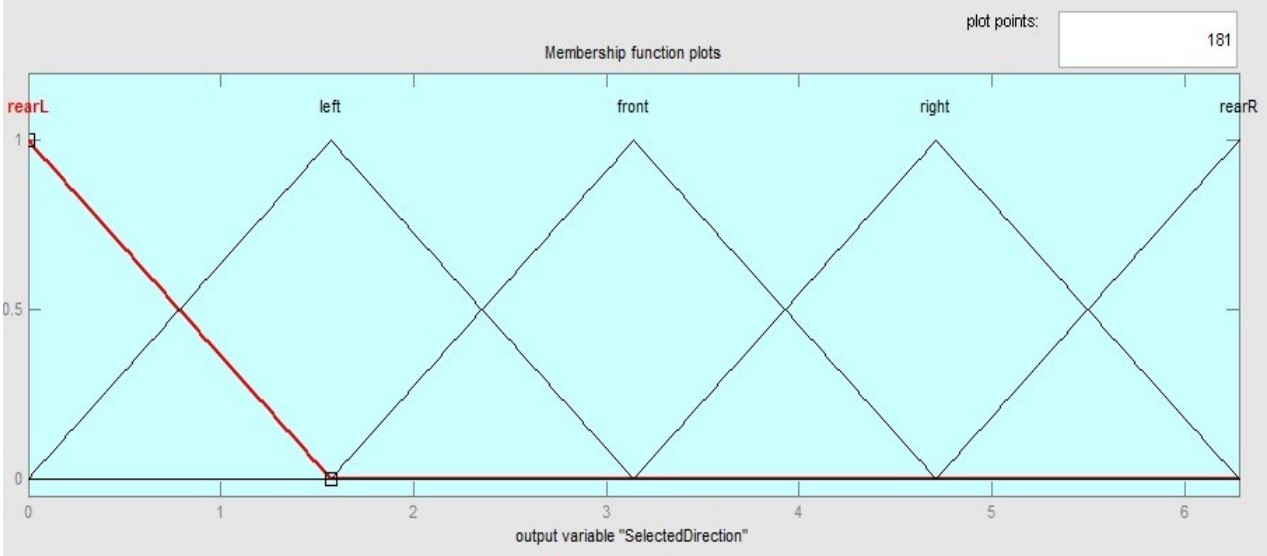


Figure 3.5: The output direction membership functions

The speed membership function has three members (slow, medium, and fast) and is based on the initial velocity and speed which has been chosen by the user. The output speed membership function is however based on percentage. *For example* if the user chooses (with the help of the joystick) the maximum speed of 4m/s but there is an object in the close distance to the wheelchair, the controller will automatically decrease the velocity to 10 percent or 0.4 m/s so it won't hit the object and will be able to manoeuvre safely.

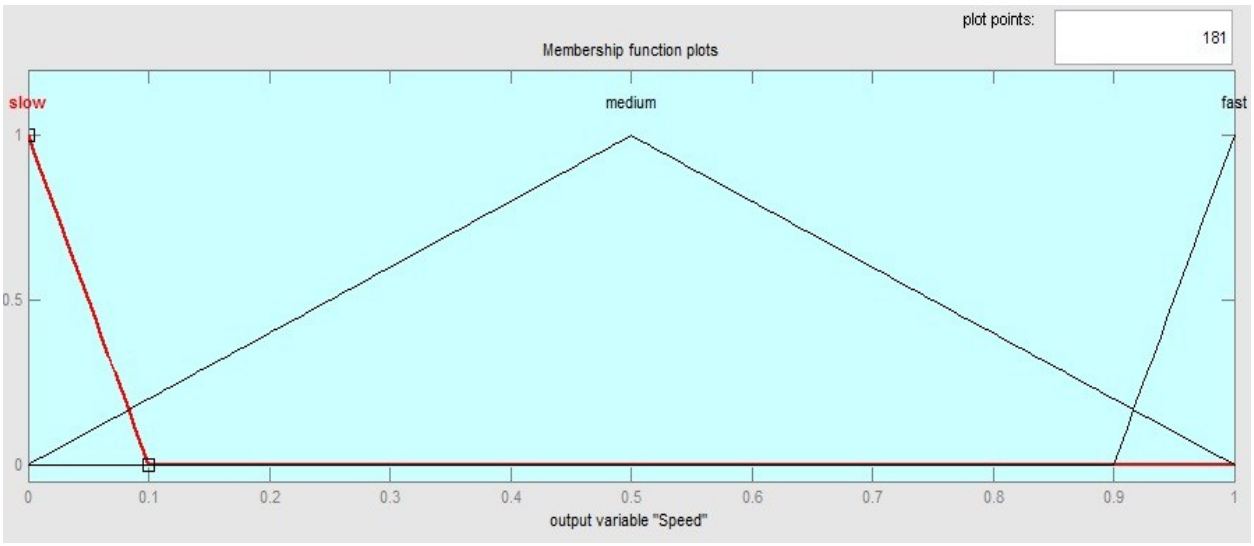


Figure 3.6: The output speed membership functions

# 3.4 Fuzzy Operators

As in classical logic, in fuzzy logic there are three basic operations on fuzzy sets: union, intersection and complement [49].

- Union: Let $\mu_A$ and $\mu_B$ be membership functions that define the fuzzy sets A and B, respectively, on the universe X. The union of fuzzy sets A and B is a fuzzy set defined by the membership function:

$$\mu_{AUB}(x) = Max(\mu_A(x), \mu_B(x)) \quad (3.2)$$

- Intersection: Let $\mu_A$ and $\mu_B$ be membership functions that define the fuzzy sets A and B, respectively, on the universe X. The intersection of fuzzy sets A and B is a fuzzy set defined by the membership function:

$$\mu_{A\cap B}(x) = Min(\mu_A(x), \mu_B(x)) \quad (3.3)$$

- Complement: Let $\mu_A$ be a membership function that defines the fuzzy set A, on the universe X. The complement of A is a fuzzy set defined by the membership function:

$$\mu'_A(x) = 1 - \mu_A(x) \quad (3.4)$$

## *3.4.1 T-norms and T-conorms*

T-norms and t-conorms are binary operators that generalize intersection and union operations, respectively.

- t-norm: it is a binary operation T: [0,1] x [0,1] → [0,1] which satisfies the following properties:

- Commutativity: $T(a,b) = T(b,a)$
- Associativity: $T(a, T(b,c)) = T(T(a,b), c)$
- Identity element: $T(a,1) = T(1,a) = a$
- Monotonicity: if $a \leq c$ and $b \leq d$ then $T(a,b) \leq T(c,d)$

These operators represent the intersection of two fuzzy sets. Some examples of t-norms are the minimum min (a,b), the product prod (a,b) = a•b and Lukasiewicz W (a,b) = max (0,a+b-1).

- t-conorm: it is a binary operation $S: [0,1] \times [0,1] \rightarrow [0,1]$ which satisfies the following properties:
  - Commutativity: $S(a,b) = S(b,a)$
  - Associativity: $S(a, S(b,c)) = S(S(a,b), c)$
  - Identity element: $S(a,0) = S(0,a) = a$
  - Monotonicity: if $a \leq c$ and $b \leq d$ then $S(a,b) \leq S(c,d)$

These operators represent the union of two fuzzy sets. Some examples of t-conorms are the maximum max (a,b), the probabilístic sum or sum-product sum-prod (a,b) = a+b - a•b and Lukasiewicz W*(a,b)= min (1,a+b).

# 3.5 Inference Engine

A Fuzzy Inference System (FIS) is a way of mapping an input space to an output space using fuzzy logic. A FIS tries to formalize the reasoning process of human language by means of fuzzy logic (that is, by building fuzzy IF-THEN rules). Rules form the basis for the fuzzy logic to obtain the fuzzy output [50]. The inference engine combines If-Then type fuzzy rules and converts the fuzzy inputs to the fuzzy outputs.

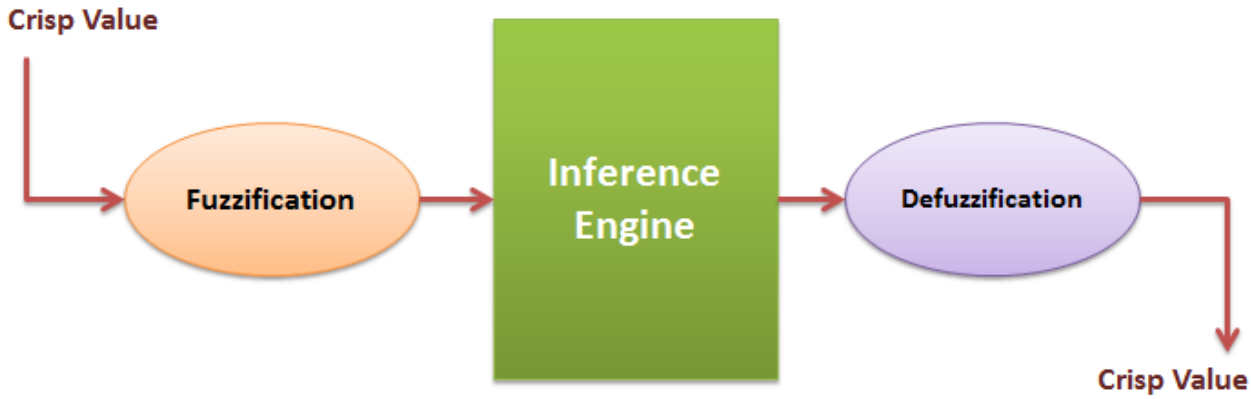


Figure 3.7: Fuzzy logic controller system

The rule-based uses linguistic variables as its antecedents and consequents. The antecedents express an inference or the inequality, which should be satisfied [51]. The consequents are those, which we can infer, and is the output if the antecedent inequality is satisfied. The fuzzy rule-based system uses IF–THEN rule-based system, given by, IF antecedent, THEN consequent. There are three types of inference engine: Mamdani's model, Takagi-Sugeno-Kang model (TSK), and Standard Additive model (SAM).

Mamdani's method is been used to design the system which is the most commonly used in applications, due to its simple structure of 'min-max' operations. Twenty four (24) rules have been designed for the system (Table 1.1). Min operation has been used for the fuzzy AND method and the fuzzy implication. Max operation has been used for the fuzzy OR method and the fuzzy aggregation.

Table 3.1: The smart wheelchair control system rules

| Rules No | Direction | S 0 | S 1 | S 2 | S 3 | S 4 | S 5 | S 6 | S7 | Speed (V) | Direction |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | *Front* | *F* | *F* | | | | | | | **F** | **Front** |
| **2** | *Left* | | | | | *F* | | | | **F** | **Left** |
| **3** | *Right* | | | | | | *F* | | | **F** | **Right** |
| **4** | *Left* | | | | | *N* | | | | **S** | **Right** |
| **5** | *Right* | | | | | | *N* | | | **S** | **Left** |
| **6** | *Front* | | *N* | | | | | | | **S** | **Left** |
| **7** | *Front* | *N* | | | | | | | | **S** | **Right** |
| **8** | *Left* | *M* | | *N* | | | | | | **M** | **Right** |
| **9** | *Right* | | *M* | | *N* | | | | | **M** | **Left** |
| **10** | *Left* | ` | | *N* | | | | | | **S** | **Right** |
| **11** | *Right* | | | | *N* | | | | | **S** | **Left** |
| **12** | *Front* | *N* | *N* | *N* | *N* | | | | | **S** | **Rear right** |
| **13** | *Front* | | | *N* | | | | | | **S** | **Right** |
| **14** | *Front* | | | | *N* | | | | | **S** | **Left** |
| **15** | *Rear left* | | | | | | | *N* | | **S** | **Right** |
| **16** | *Rear right* | | | | | | | | *N* | **S** | **Left** |
| **17** | *Rear left* | | | | | | | *F* | | **F** | **Rear left** |
| **18** | *Rear right* | | | | | | | | *F* | **F** | **Rear right** |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **19** | *Rear left* | | | | | | | *N* | *N* | **S** | **Right** |
| **20** | *Rear right* | | | | | | | *N* | *N* | **S** | **Left** |
| **21** | *Front* | | | | *N* | | | | | **S** | **Left** |
| **22** | *Front* | | | *N* | | | | | | **S** | **Right** |
| **23** | *Front* | | *M* | | *N* | | *M* | | | **S** | **Rear left** |
| **24** | *Front* | *M* | | *N* | | *M* | | | | **S** | **Rear right** |

Sensors abbreviations:

*N* – Near

*M* – Middle

*F* – Far

Output speed abbreviations:

*S* – Slow

*M* – Medium

*F* – Fast

# 3.6 Defuzzifications

Defuzzification process is the opposite of fuzzification which means converting the fuzzy output to crisp values. Once the rules have been composed the solution we get a fuzzy set, however, for most applications there is a need for a `crisp' solution to emanate from the inferencing process. This will involve the `defuzzification' of the solution set.

Centroide method is the most widely used method which has been also used in this system. Centroid defuzzification returns the center of area under the curve. It also been called as center of gravity or center of area method. It can be defined by the algebraic expression:

(3.5)

$$Z^* = \frac{\int \mu_A(z) z dz}{\int \mu_A(z) dz}$$

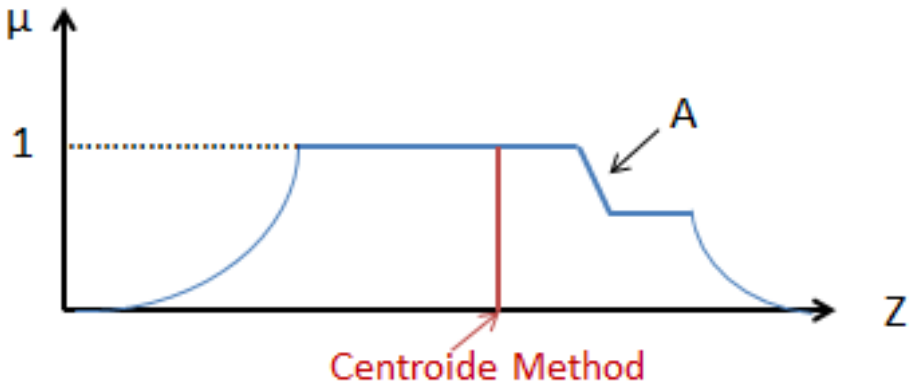


Figure 3.8: The centroide defuzzification method

Some other commonly used defuzzification methods are: Weighted average method, Centre of sums, Mean of maximum.

The defuzzification process will output two numbers indicating the corrected angular velocity and linear velocity. The microcontroller will then build two PWM channels based on this data and sends it to the wheelchair motor driver.

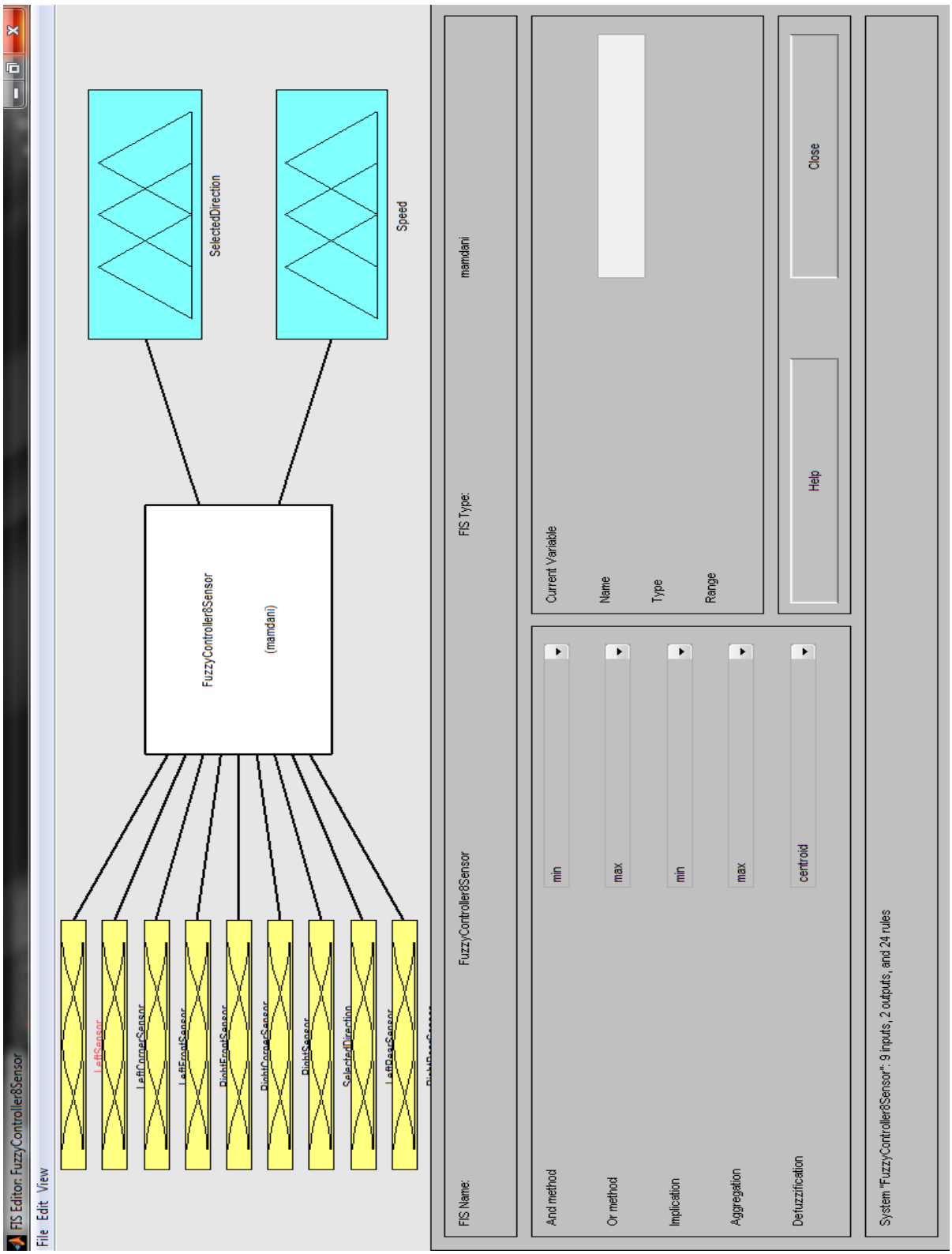


Figure 3.9: The smart wheelchair FIS editor

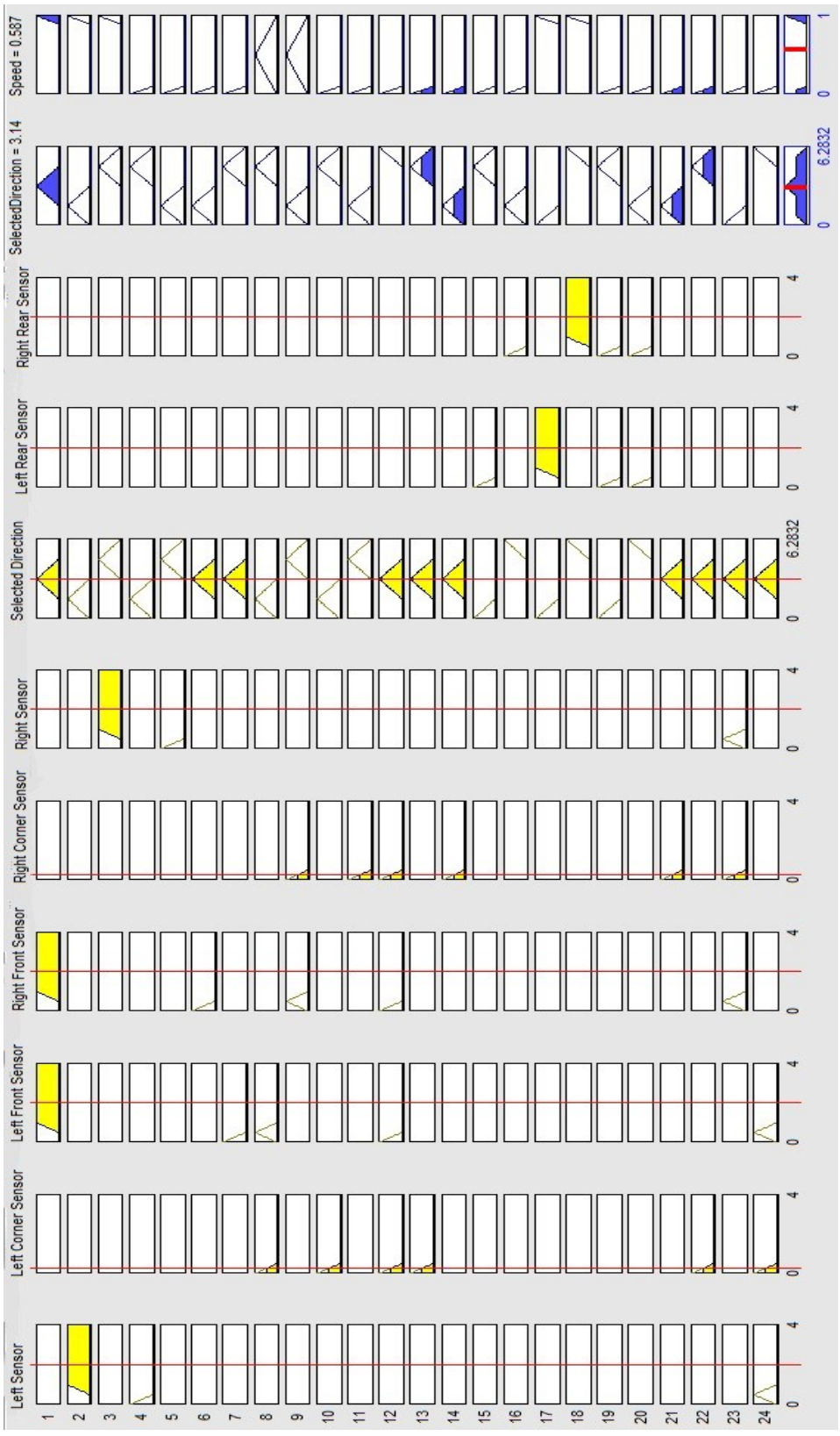


Figure 3.10: The smart wheelchair fuzzy rule viewer

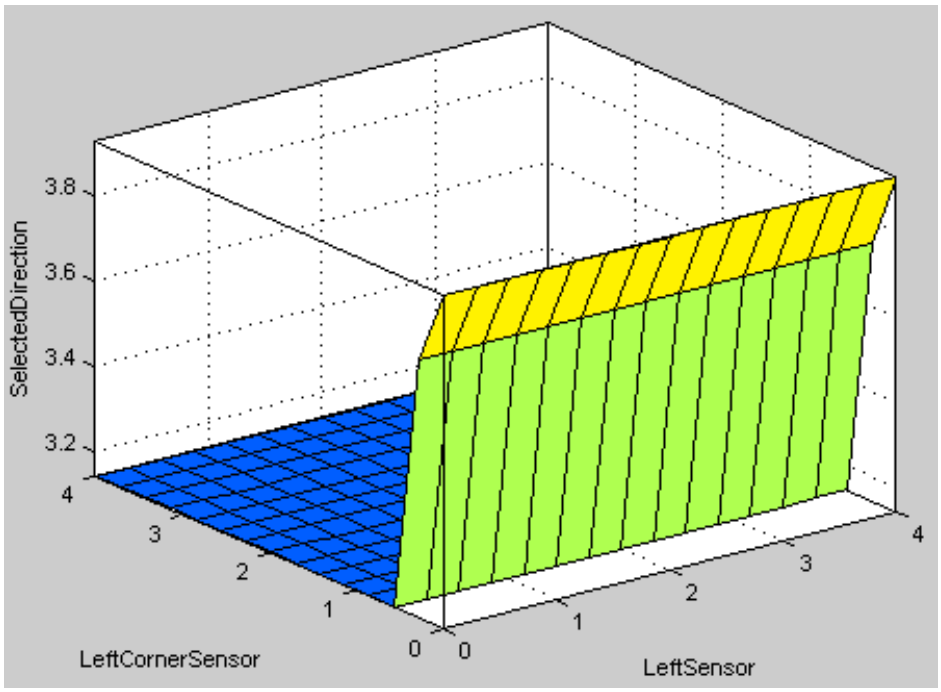


Figure 3.11: The fuzzy logic control surface viewer (sensor1 and 3 vs. output direction)

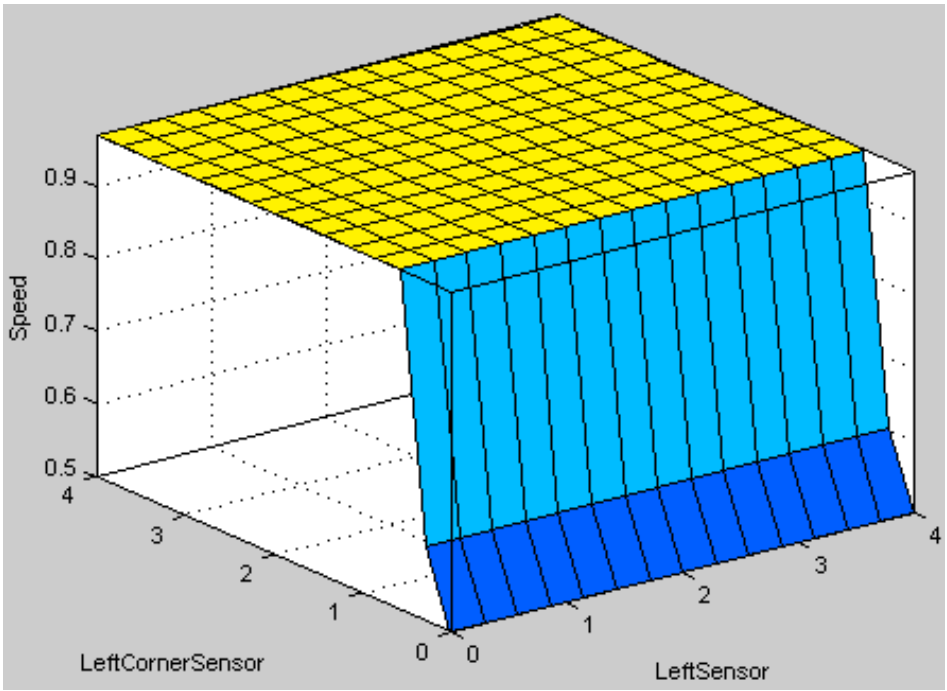


Figure 3.12: The fuzzy logic control surface viewer (sensor1 and 3 vs. output speed)

# Chapter 4

# Simulink and Simulation Results

On the following chapter a brief description of the Simulink model, its functions and the simulation results are provided. The section 4.2 explains the Simulink blocks developed to model the modified smart wheelchair including the room/sensory model, motors model, the microcontroller, the joystick, and the wheelchair itself. The section number 4.3 shows the MATLAB M-files that are used to develop the graphics of the environment. To join the Simulink blocks and the MATLAB M-files, some inference functions have been added, they are explained in section 4.4. The GUI (Graphical User Interface) is described in section number 4.5. Finally the simulation results are been discussed in the section 4.6.

## 4.1 MATLAB Simulink

MATLAB is an interactive program for numerical computation and data visualization; it is used extensively by control engineers for analysis and design. There are many different toolboxes available which extend the basic functions of MATLAB into different application areas [52].

Simulink is an environment for multi-domain simulation and Model-Based Design for dynamic and embedded systems. It provides an interactive graphical environment and a customizable set

of block libraries which makes it possible to design, simulate, implement, and test a variety of time-varying systems, including communications, controls, signal processing, video processing, and image processing [53].

MATLAB fuzzy logic toolbox was used to design the control system for the system. The fuzzy controller then was integrated to the Simulink environment along with the wheelchair model to test the validity of the designed system.

## 4.2 Main Functions

The main function of the system has been divided into four main blocks, they represent different group of elements:

1. The embedded microcontroller model function has the algorithms to implement the embedded microcontroller that will be inserted between the wheelchair controller and the joystick. It contains the control system and the fuzzy logic algorithm and the interfaces between the input signals and output signals.

2. The motor-driver model function implements the motors and the wheelchair controller. It has both the motor controller algorithm and DC motors models, so the inputs are the X-axis and Y-axis signals of the joystick and the outputs are the linear and angular speed of the wheels.

3. The electric wheelchair model is the third block and it takes the speed of the wheels and calculates the position of the chair as an integration of them.

4. The room and sensors models function is the way to incorporate the M-files to the Simulink system, so it will call the M-functions with the input of the wheelchair model and puts the measurement of the sensors inside the Simulink blocks.

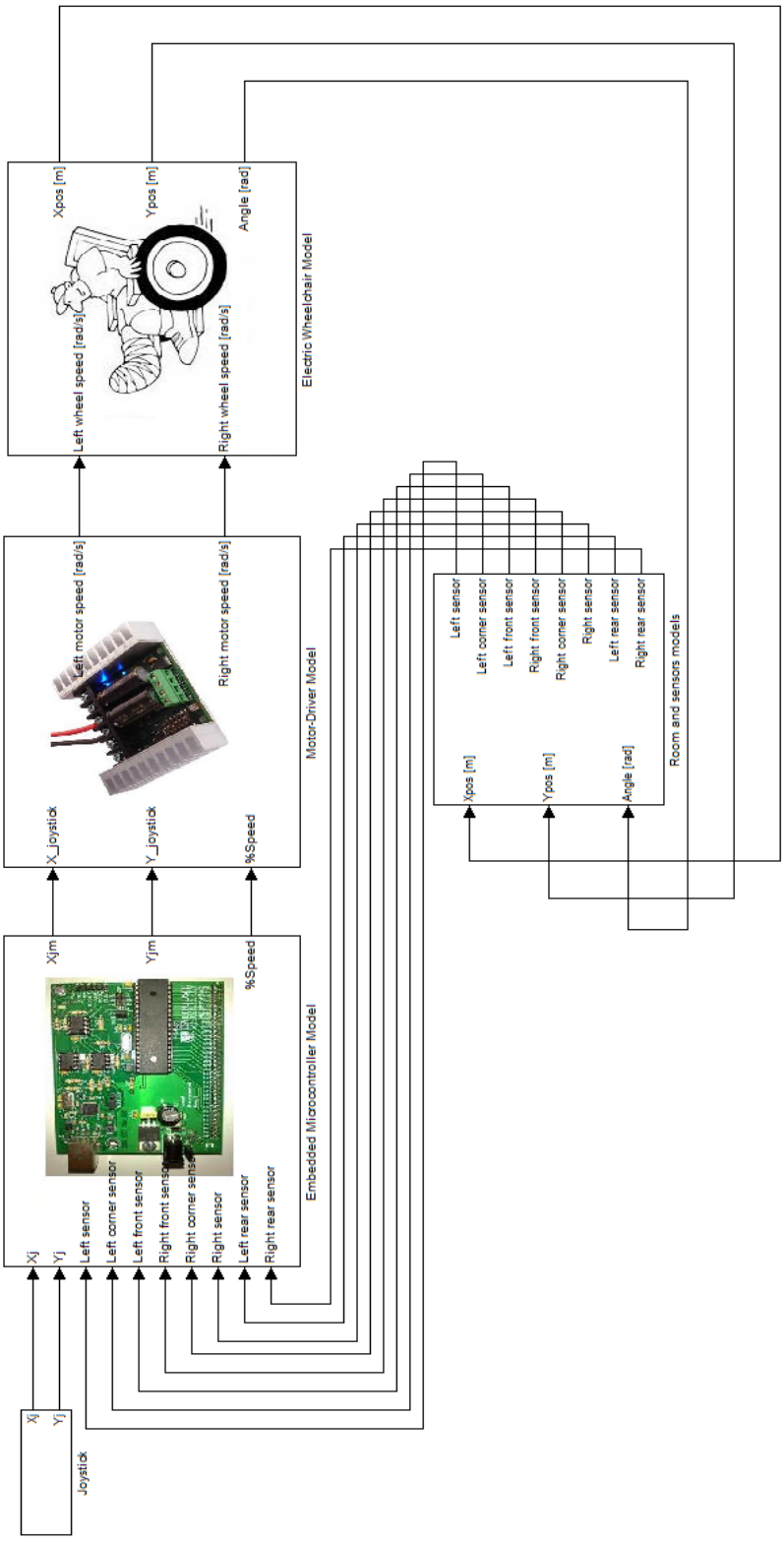


Figure 4.1: The smart wheelchair Simulink model

### 4.2.1 Embedded Microcontroller Model

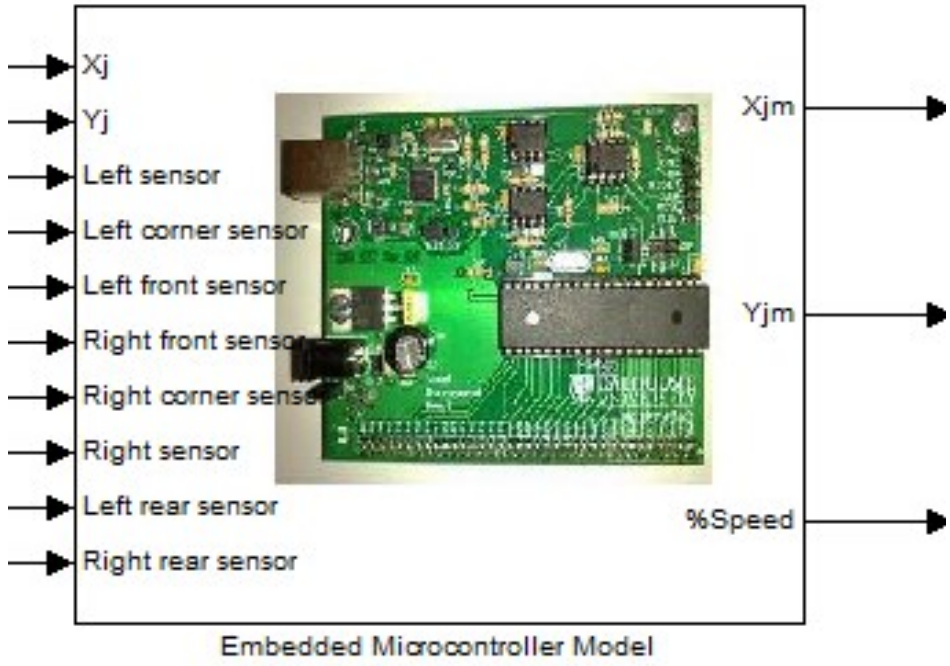


Figure 4.2: The embedded microcontroller Simulink model

This model converts the X and Y joystick positions and calculates the joystick angle for the fuzzy algorithm. It also has the sensors as inputs which are fed directly to the fuzzy logic controller. Once the fuzzy algorithm is computed depending on the distance to the obstacles, the modified joystick angle is calculated. Finally this angle is transformed to the X and Y joystick signals that represent the angular speed of the motors. There is also another output from the fuzzy algorithm representing the linear speed of the motors. The linear speed is based on the percentage of the maximum speed chosen by the user. The linear speed signal along with the angular speed, will transform to the right and left motors speed and thus the direction of the wheelchair.

The "Embedded Microcontroller" block model will be implemented on the actual microcontroller hardware. The most important sub-function inside the block is the fuzzy algorithm that will be translated to embedded C code and will be installed on the microcontroller along with the other sub-functions which convert the joystick signals to the joystick-direction

used inside the fuzzy and then convert the output joystick-direction into joystick signals (or into speed on the wheels). The combination of X, Y and speed percentage (%) signals provide a one, and only one, combination of right and left motor speeds. It means that if we know these three signals we can calculate the speed of the wheels and thus the wheelchair movement and vice versa.

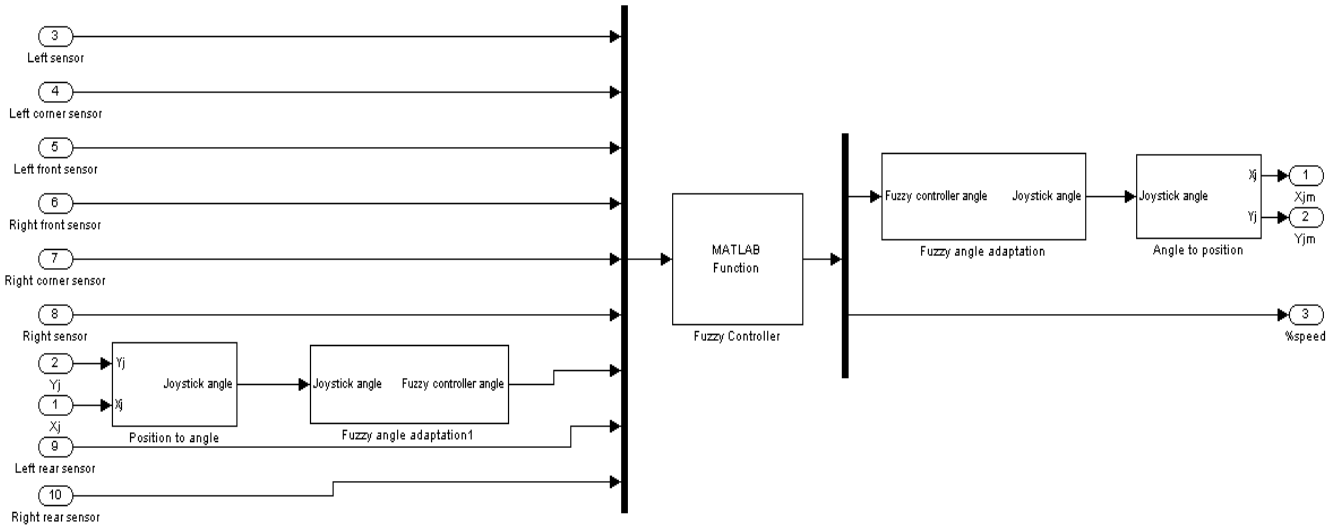


Figure 4.3: The embedded microcontroller model inside blocks

## *4.2.2 Motor Driver Model*

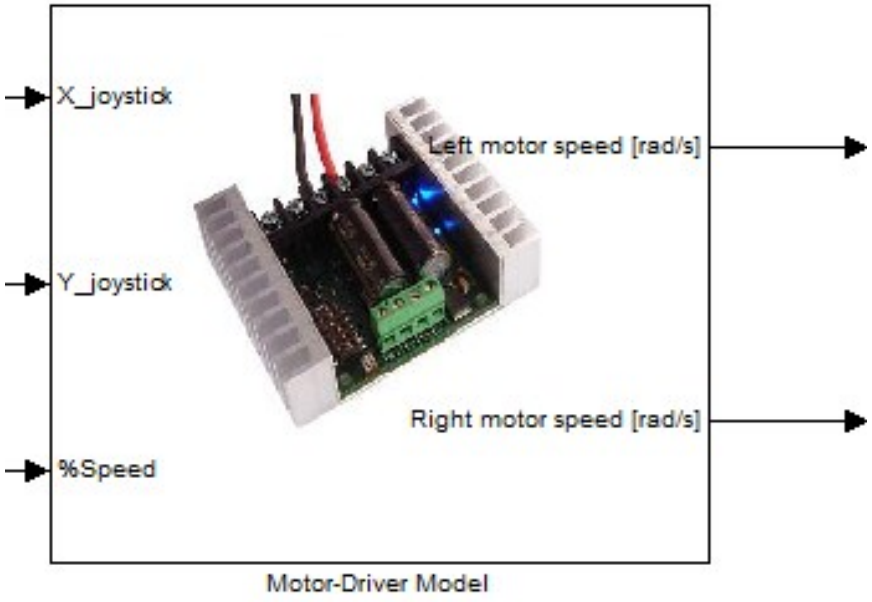


Figure 4.4: The motor controller/driver Simulink model

This block model simulates the wheelchair controller/driver in the Simulink environment. As can be seen on the figure 4.5, the Simulink model has a sub-function in which the motor driver is implemented. In this sub-function the transformation between joystick angle and motor speed is done based on the table 4.1 and figure 4.6. As it can be seen from the table 4.1 the analog voltage variations from 0V-2V indicates speed variations from maximum to minimum in clockwise direction and variations from 3V-5V gives the speed variations from minimum to maximum in counter clockwise direction. The slot of variations between 2.02V-2.98V has been chosen for a no operation state [54].

The sub-function modified output is a signal similar to the PWM signal produced by the microcontroller, with a digital value from 0 Hex to 255 Hex. When multiplying by the speed percentage and reconverted by the PWM, the speeds of the motors are calculated.

The figure 4.6 gives a detailed outline of how the motor movement takes place by varying joystick position in a 360 degree plane. The arrows drawn on each circle representing left and right motors shows the speed and direction of the motors at different joystick angles. When the

line is pointing upward it means that the motor moves in clockwise direction and when the speed line is pointed downward then the motor is moving in counter clockwise direction. The yellow arrows mean that the motor or wheel corresponding to it has less power and speed than the motor with the red arrow. The cross signs are an indication for a turned off (not moving) motor.

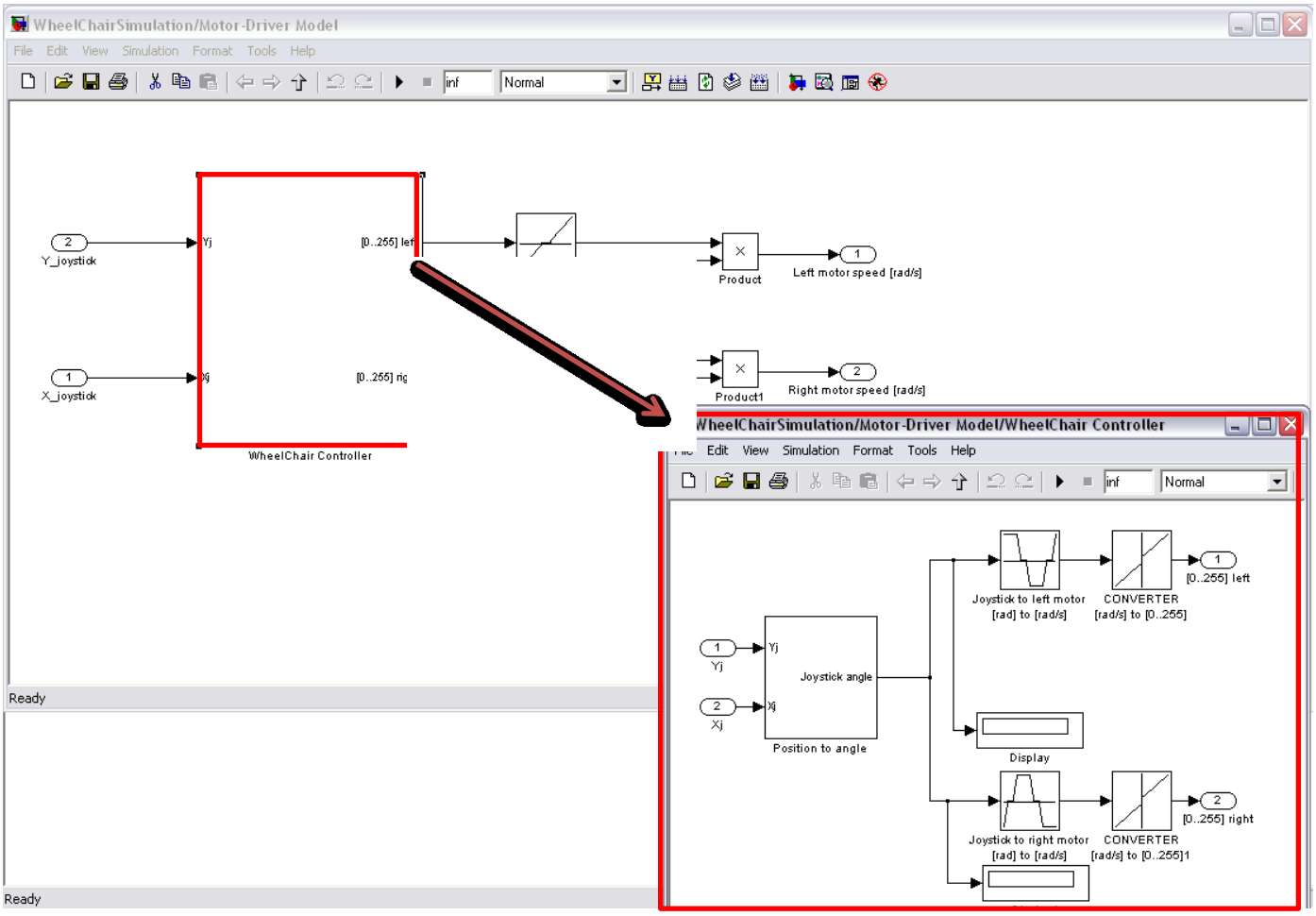


Figure 4.5: The motor controller model inside blocks

Table 4.1: Variation of motor speed and direction with regarding to the controller output voltage

| Analog Voltage (V) | Digital (Hex) | Speed | Motor Direction |
|---|---|---|---|
| **0.00 – 2.00** | 00 - 66 | Max - Min | Clockwise |
| **2.02 – 2.98** | 67 - 98 | Zero | OFF |
| **3.00 – 5.00** | 99 - FF | Min - Max | Counter Clockwise |

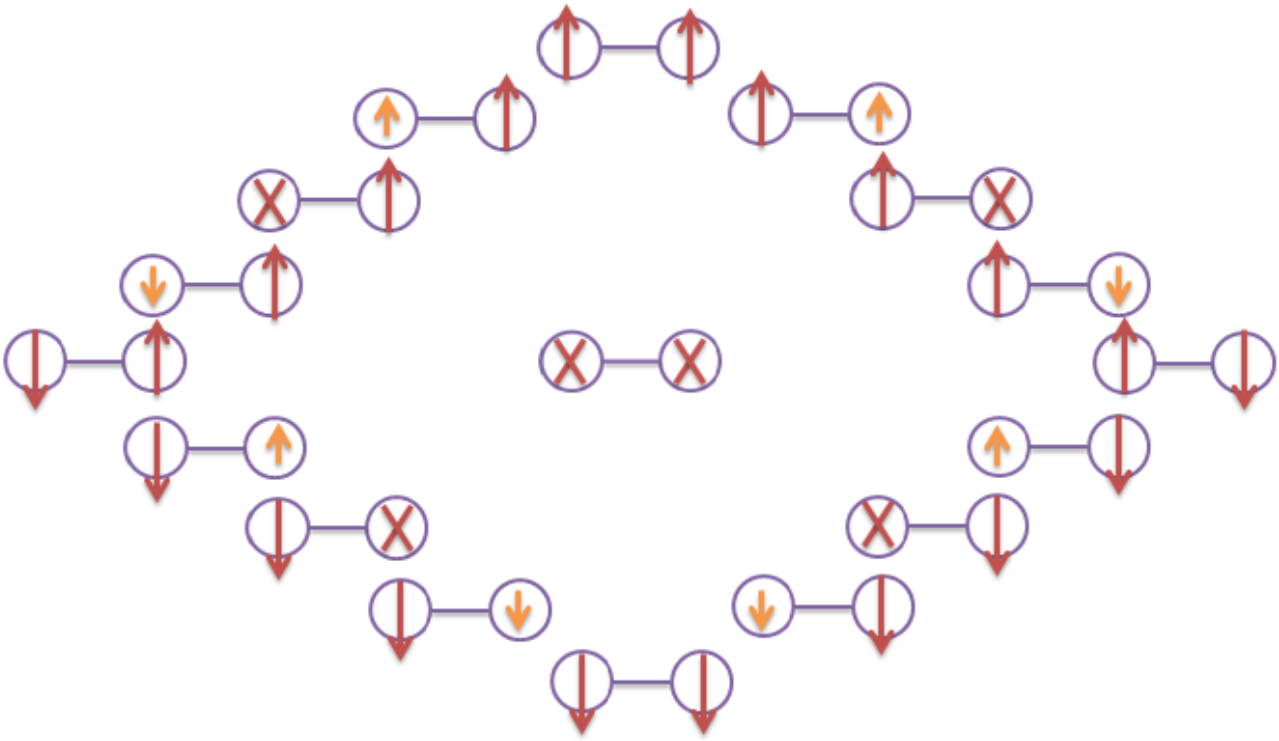

Figure 4.6: Direction and speed of left and right motors corresponding to the different position of the joystick

## *4.2.3 Electric Wheelchair Model*

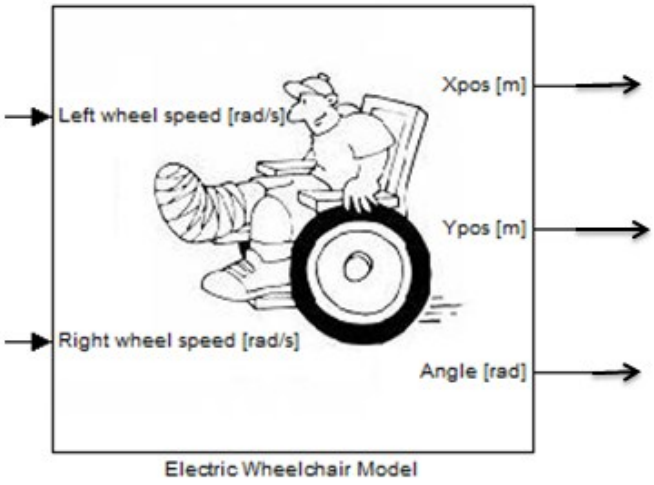


Figure 4.7: The electric wheelchair Simulink model

The electric wheelchair model simulates the wheelchair behaviors according to the received inputs. It takes the speed of the motors (calculated by the fuzzy algorithm) and computes the X and Y positions along with the direction of the wheelchair (Angle [rad]) over the room. Note that the block model output signals are based on the global coordinates.

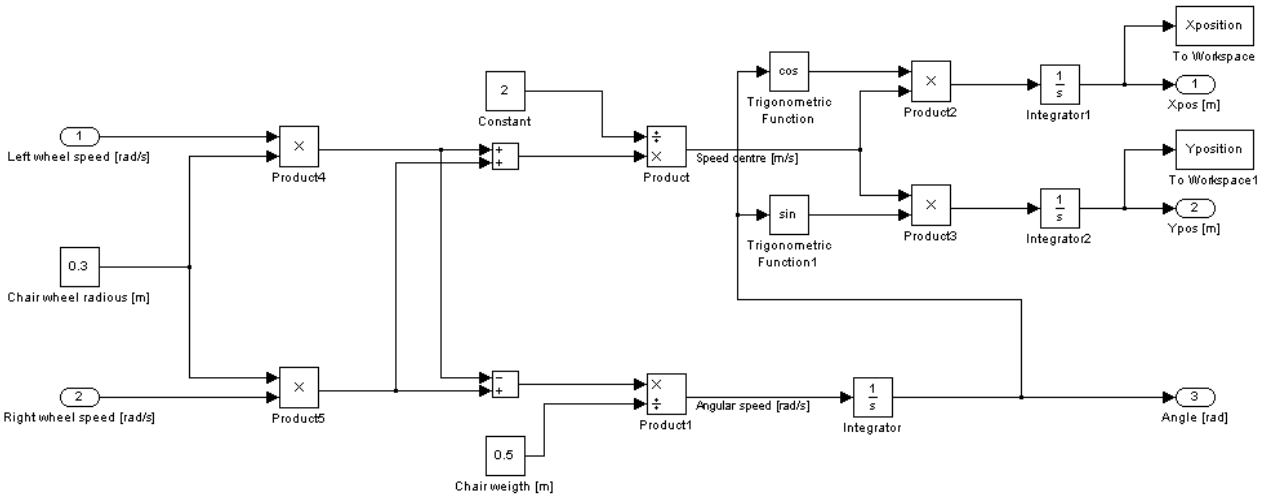


Figure 4.8: The electric wheelchair model inside blocks

The model of the chair is based on the calculation of the "Instantaneous Rotation Center" of a solid axle and the kinematic equations of the wheelchair.

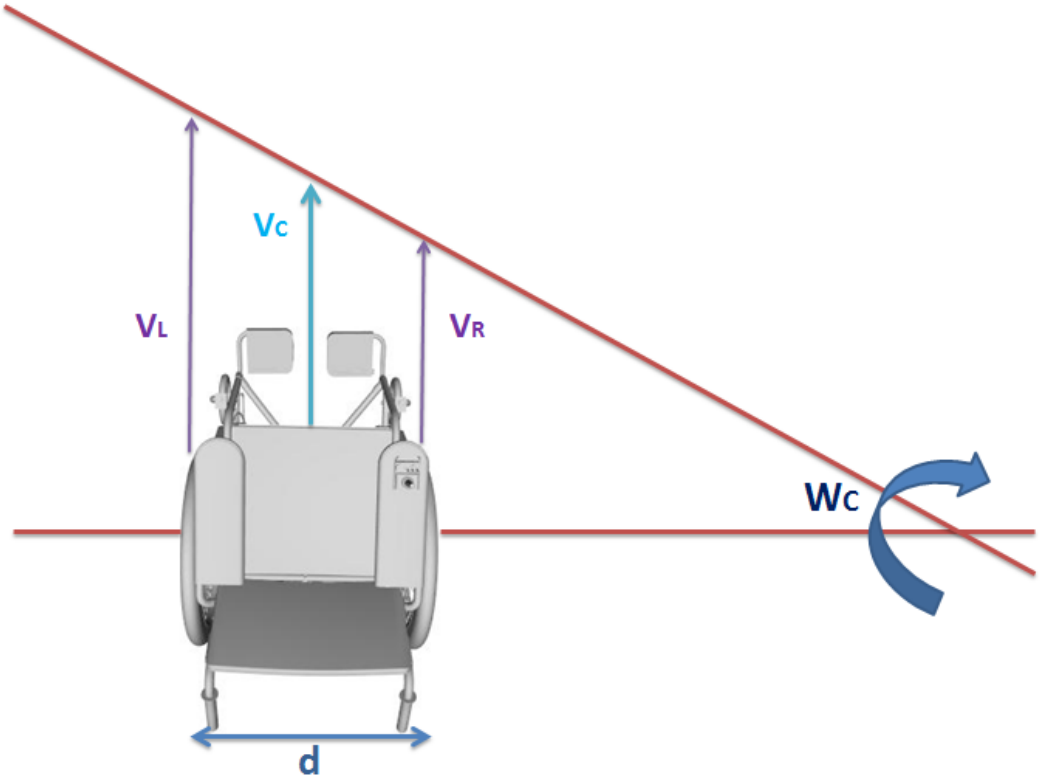


Figure 4.9: IRC of the wheelchair

(4.1)

$$V_C = (V_R + V_L)/2$$

$$W_C = (V_R - V_L)/d$$

$V_R$ - Right side speed of the axle.

$V_L$ - Left side speed of the axle.

$V_C$ - Center speed of the axle.

$W_C$ - Angular speed of the axle.

$d$ - Length of the axle (Distance between the wheels on wheelchair).

## *4.2.4 Room and Sensors Model*

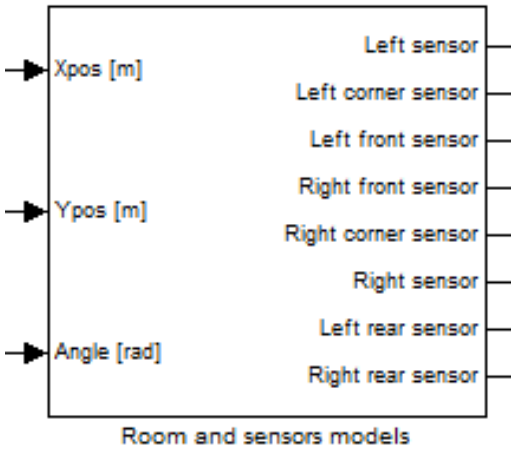


Figure 4.10: The room and sensory circuit Simulink model

In this block model, the calculation of the sensor measurements and the command to plot the environment takes place. The inner blocks are "Plot function" and "Sensors measurement", and they call the M-files "GetSensors" and "PlotChair" which are coded outside the Simulink environment.

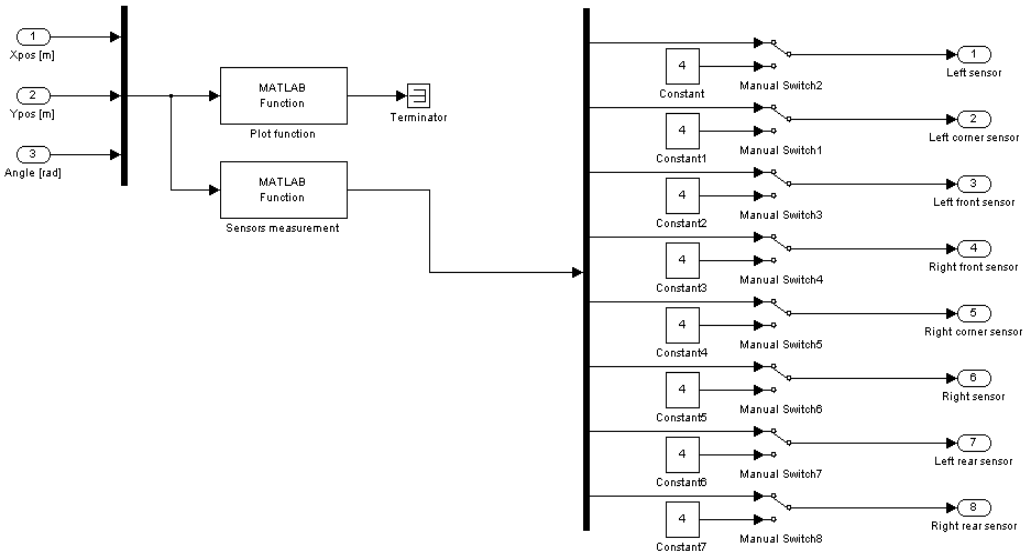


Figure 4.11: The room and sensory model inside blocks

# 4.3 MATLAB M-files

The expressions that are not allowed to use in Simulink, are coded in MATLAB M-language, they are the functions that show the environment and compute the distances between the chair and the obstacles along with the sensor's directions.

Users are able to tailor MATLAB by creating their own functions and scripts of MATLAB commands and functions. Both scripts and functions are ordinary ASCII text files external to MATLAB. The name of each file ends in "`.m`" and can be found on MATLAB's search path.

A script may contain any sequence of MATLAB statements, including references to other M-files. It is invoked like any other command without arguments and acts on the variables of the workspace globally. Each command in the file is executed as though you had typed it into MATLAB [55].

MATLAB functions can be run by just calling the function name. This is useful because functions can be called in MATLAB scripts or other functions without having to write out the whole function again. The Source codes for PlotChair.m, PlotSensorare.m, GetSensors.m, and Measurement.m are attached as Appendix A.

## *4.3.1 PlotChair.m*

Here, the most important goal is to plot the environment and the wheelchair with the sensors, the code transforms local coordinates into global coordinates depending on the X position, Y position and Angle provided by the Simulink block model. So, first the objects are designed in local coordinates and then transformed to global coordinates or coordinates of the room.

### 4.3.2 PlotSensor.m

This M-file is called by the PlotChair.m function to plot the 8 sensors in global coordinates. First the sensor coordinates are transformed to the local-chair coordinates and finally to the global coordinates (Room coordinates).

### 4.3.3 GetSensors.m

Once the plotting is done, the next point is to compute the distance between the wheelchair and the obstacles in global coordinates, so this M-file calculates the point and direction in which each sensor is placed and then calls the Measurement.m function for measuring the distance.

### 4.3.4 Measurement.m

When the GetSensors.m function calls Measurement.m, the program calculates the distance between the sensor and the nearest obstacle in front of it. The measurement has been implemented with an error below 1 cm to reproduce the imperfections of the real obstacles and the real sensors.

# 4.4 Interface Functions

Interface Functions have been designed for communication between MATLAB workspace and the Simulink.

## *4.4.1 Init.m*

Upon the staring of the Simulink model this function is computed. It initializes the fuzzy algorithm, sets the joystick input to "up" (forward movement) and calls the GUI.

```
%Init file -> it runs upon opening the Simulink

FuzzyController8Sensor=readfis('FuzzyController8Sensor');
GUI;
global Jcontrol
Jcontrol = 1;
```

## *4.4.2 Joystick.m*

This function looks over the joystick global variables on the workspace and implements its value inside the Simulink model.

```
function out=Joystick(u)

global Jcontrol

out = Jcontrol;
```

# 4.5 The GUI (Graphical User Interface)

A graphical user interface (GUI) is a graphical display in one or more windows containing controls, called components, which enable a user to perform interactive tasks. The user of the GUI does not have to create a script or type commands at the command line to accomplish the tasks. Unlike coding programs to accomplish tasks, the user of a GUI doesn't need to understand the details of how the tasks are performed.

GUI components can include menus, toolbars, push buttons, radio buttons, list boxes, and sliders. GUIs created using MATLAB tools can also perform any type of computation, read and write data files, communicate with other GUIs, and display data as tables or as plots [56].

In the smart wheelchair Simulink model the GUI is the way to navigate the wheelchair along the room and choose the desired direction just as the real joystick. It has several buttons each representing a direction which will modify the joystick global variables when pressed. The GUI does not save the values of the joystick on the workspace. The GUI source code is attached as Appendix B.

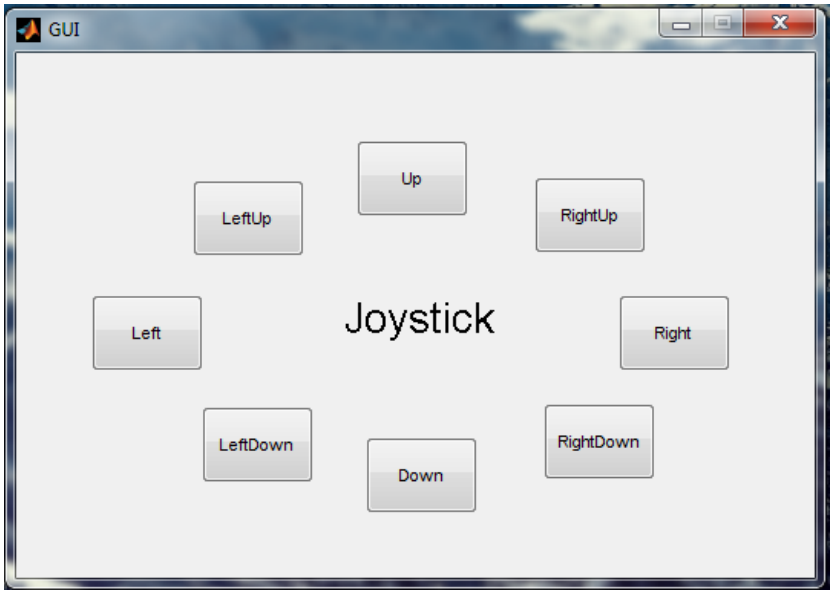


Figure 4.12: The joystick graphical user interface

# 4.6 Simulation results

Designing a fuzzy control algorithm depends heavily on the designer knowledge and experience, and so it's always better to test the control system in the simulation environment before applying it on the hardware. That's why the whole system has been simulated in order to verify the validity of the proposed system and to optimize the control system algorithm.

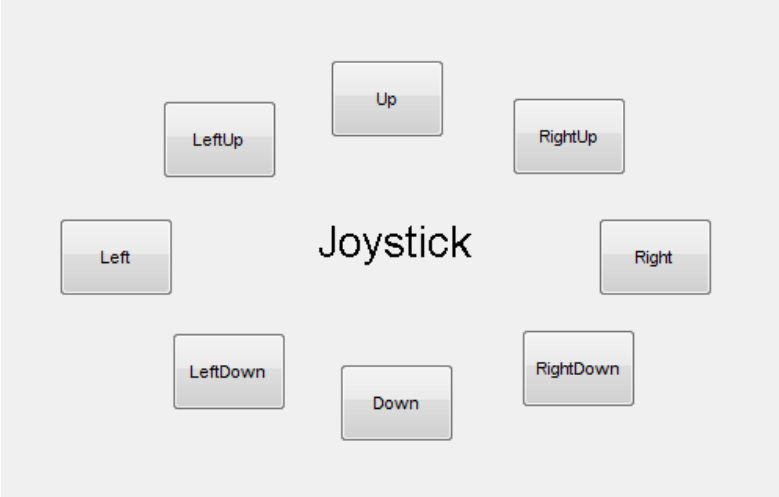


Figure 4.13: The virtual joystick (GUI)

In the simulation steps shown in Figures 4.14, 4.15... 4.31, the wheelchair navigates in the given environment, corrects the user direction if needed and avoids collision by manoeuvring around the upcoming obstacles and objects. The user can change the wheelchair direction with the virtual joystick GUI (Figure 4.13) but the wheelchair does the rest and avoids any object in its way using the designed fuzzy logic control system and the virtual sonar sensors.

For testing the system the wheelchair has been moved multiple times in a crowded environment taking different paths and in all cases the wheelchair was able to navigate while avoiding obstacles successfully.

In the following figures, green lines show the cones emitted by the sensors, the blue lines represent the obstacles, and the black line is the wheelchair path.

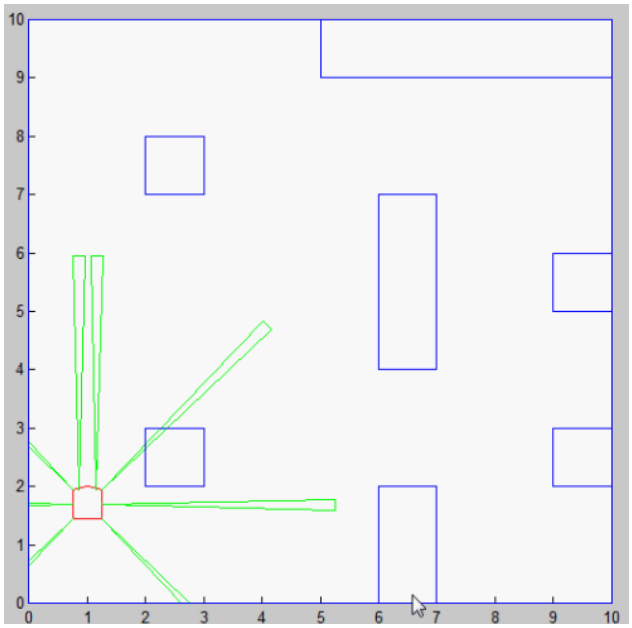

Figure 4.14: This is the start point. The user starts the wheelchair with pressing one of the direction buttons on the virtual joystick.

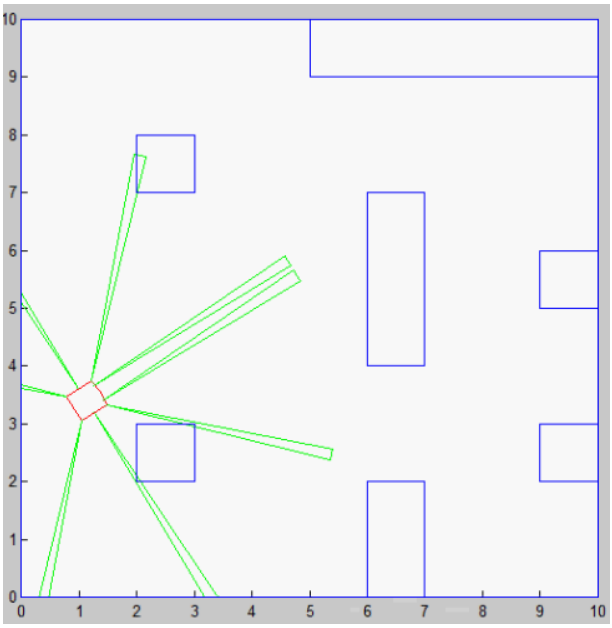

Figure 4.15: The user chooses to go the right direction by pressing the right-up button on the joystick GUI.

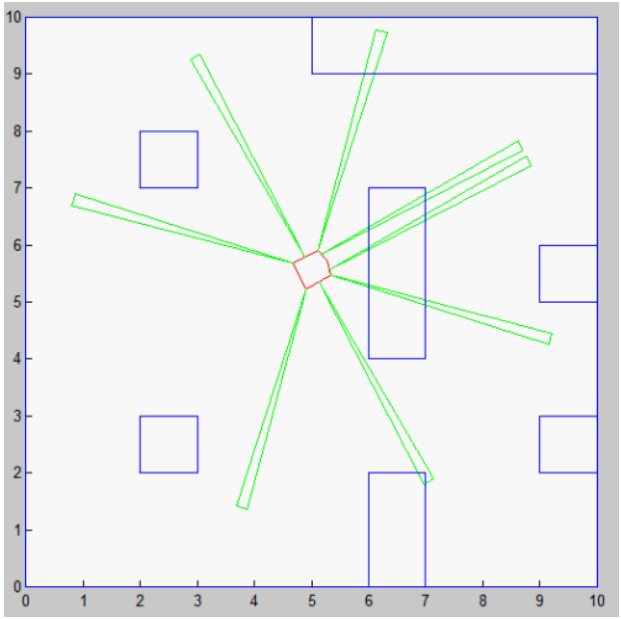

Figure 4.16: The wheelchair gets too close to the wall.

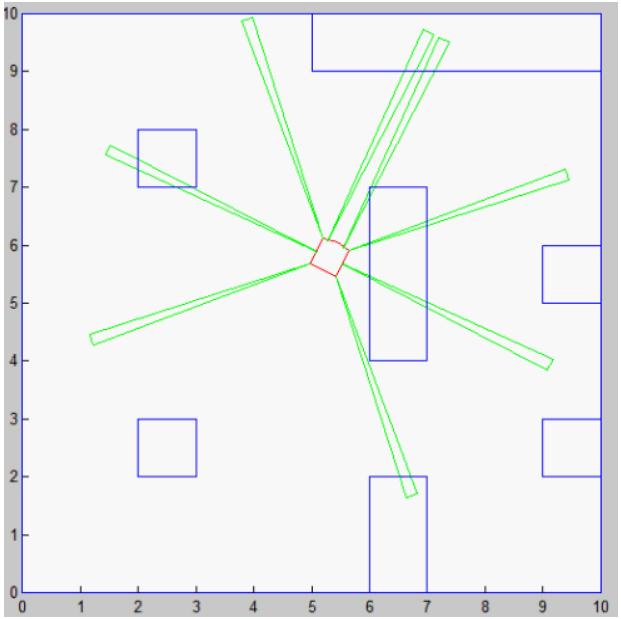

Figure 4.17: The wheelchair automatically notices the wall in front of it and avoids it by turning to its left.

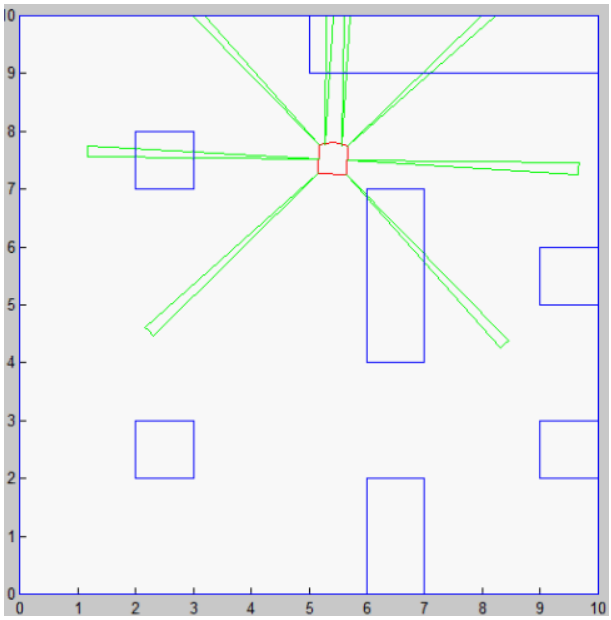

Figure 4.18: The wheelchair is getting close to the second wall.

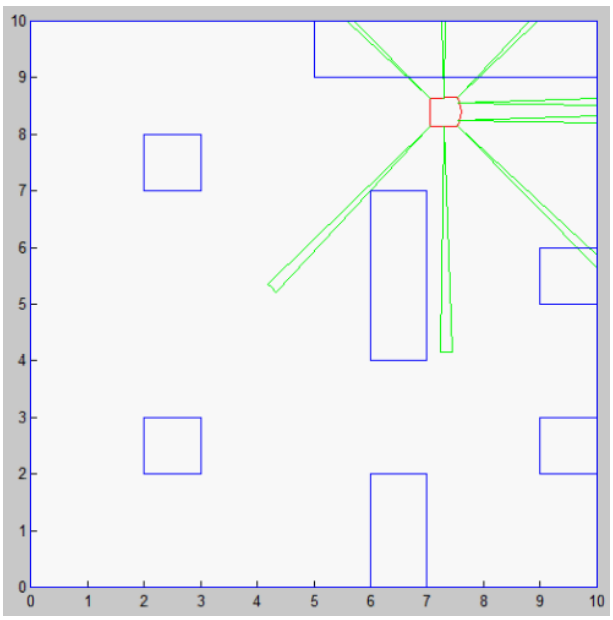

Figure 4.19: The wheelchair senses the wall, cuts the joystick connection and avoid collision by turning to its right.

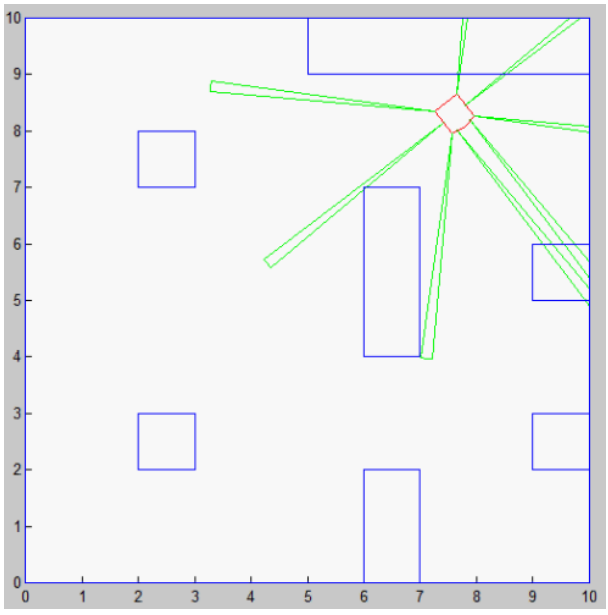

Figure 4.20: The user decides to turn to its right and pass through the hallway by pressing the right-up button on the joystick.

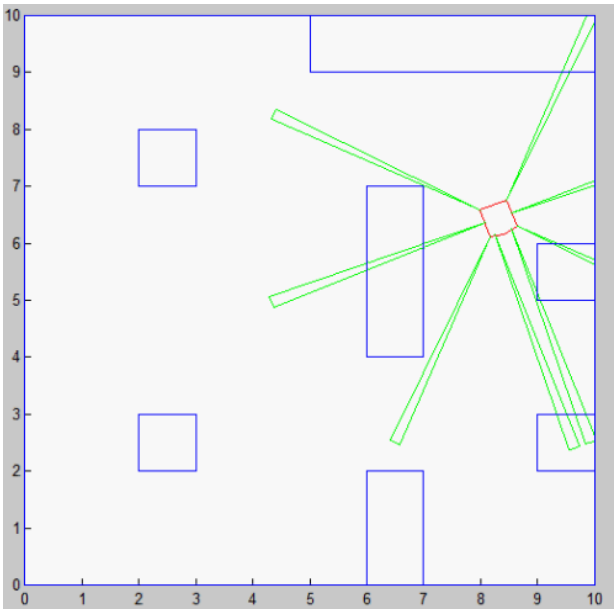

Figure 4.21: The wheelchair is getting close to an obstacle and there is a chance for a collision.

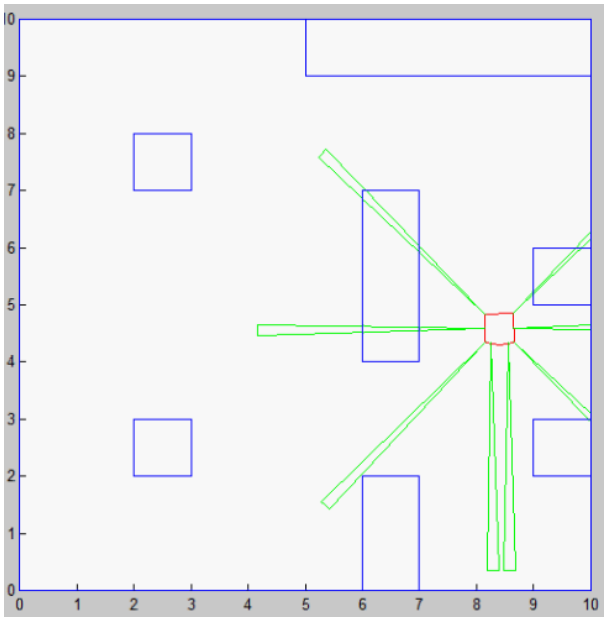

Figure 4.22: The wheelchair control system kicks in again and changes the wheelchair direction.

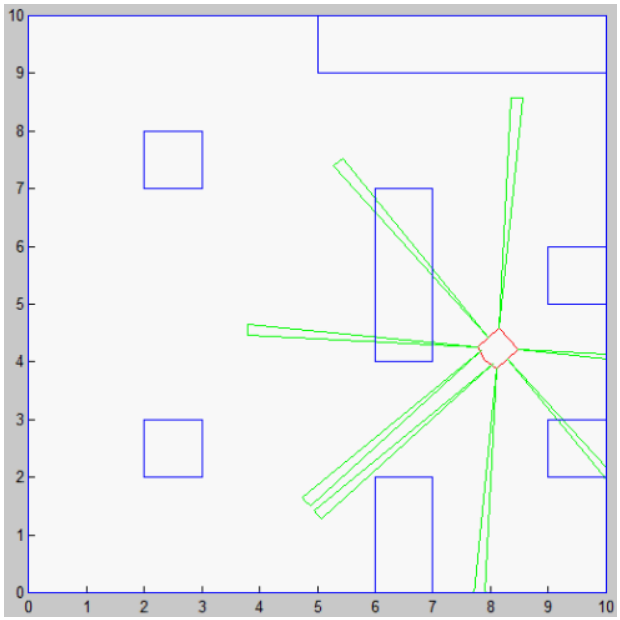

Figure 4.23: The user decides to go from between the walls (doorway) and changes the wheelchair direction.

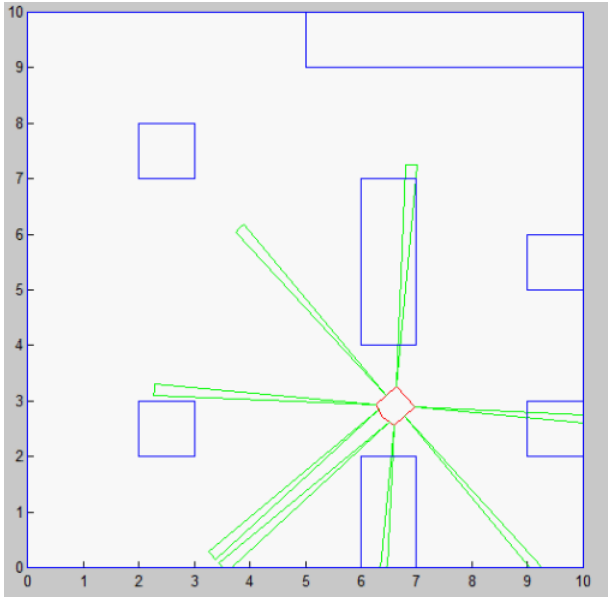

Figure 4.24: There is a chance for collision if the wheelchair keeps going in this direction.

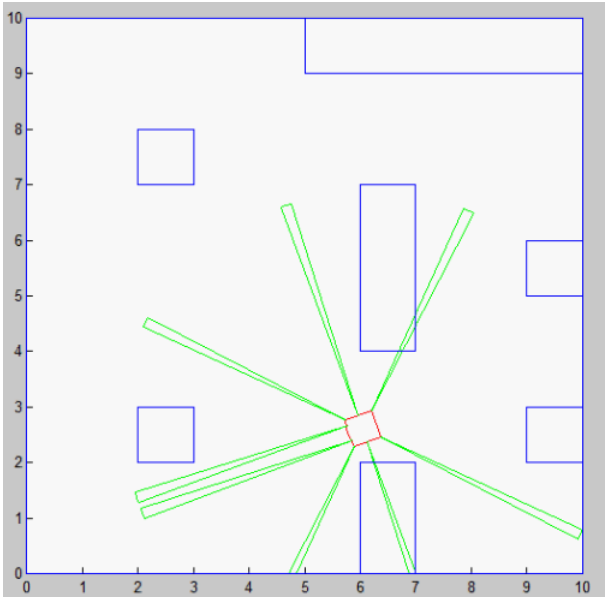

Figure 4.25: The wheelchair senses the possible collision and corrects it direction.

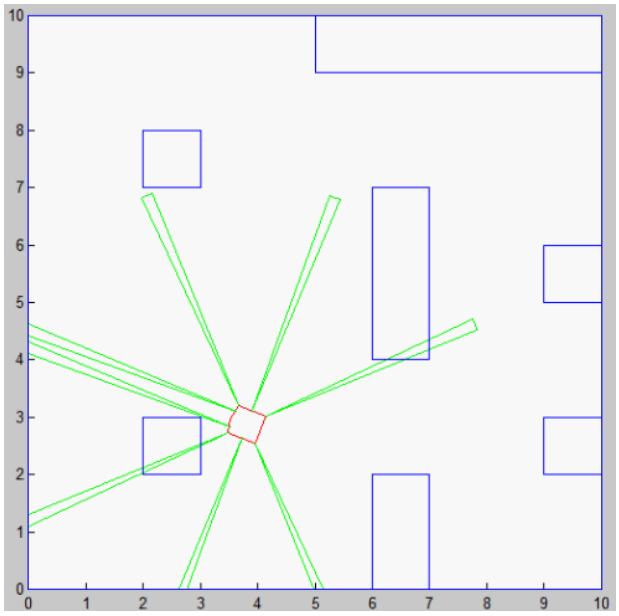

Figure 4.26: The user changes the wheelchair direction to right.

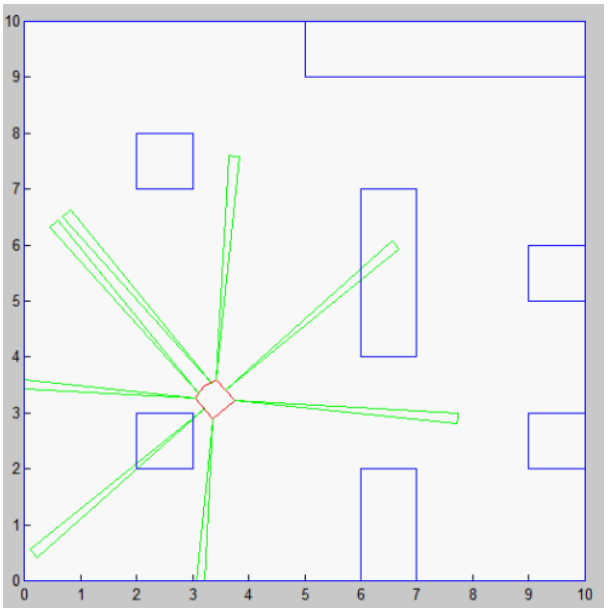

Figure 4.27: When the wheelchair gets too close to the object, the control system cuts the user joystick connection and manoeuvres the wheelchair around the object.

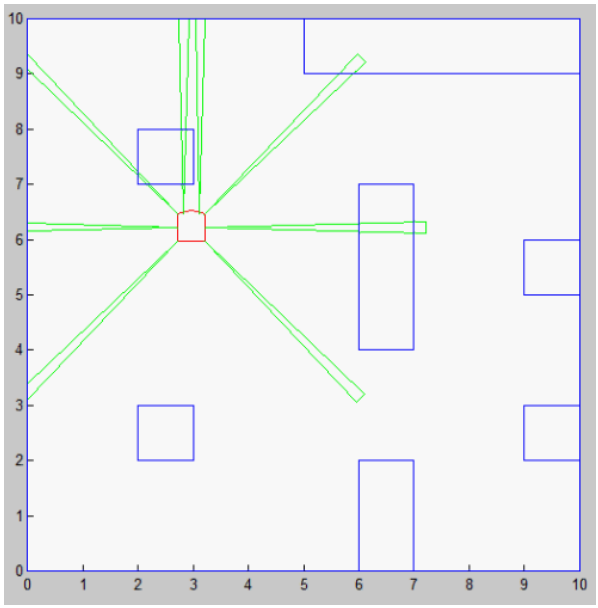

Figure 4.28: The user navigates the wheelchair.

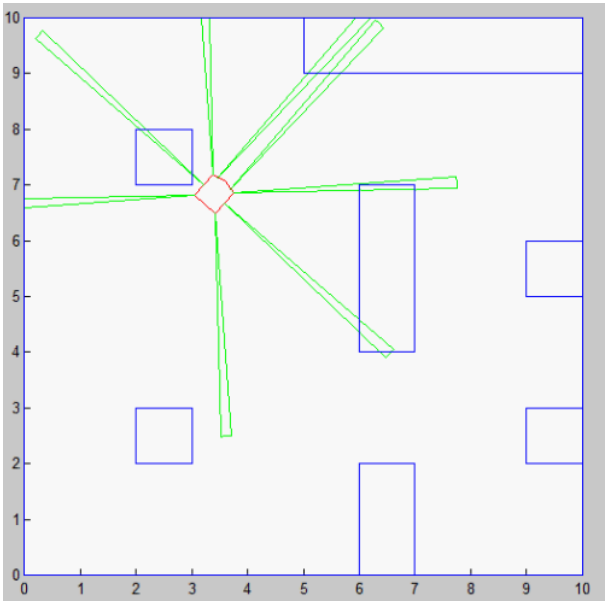

Figure 4.29: The wheelchair designed control system alters the direction when senses any kind of danger. In this case it turns right so the upper left side of the wheelchair will not collide.

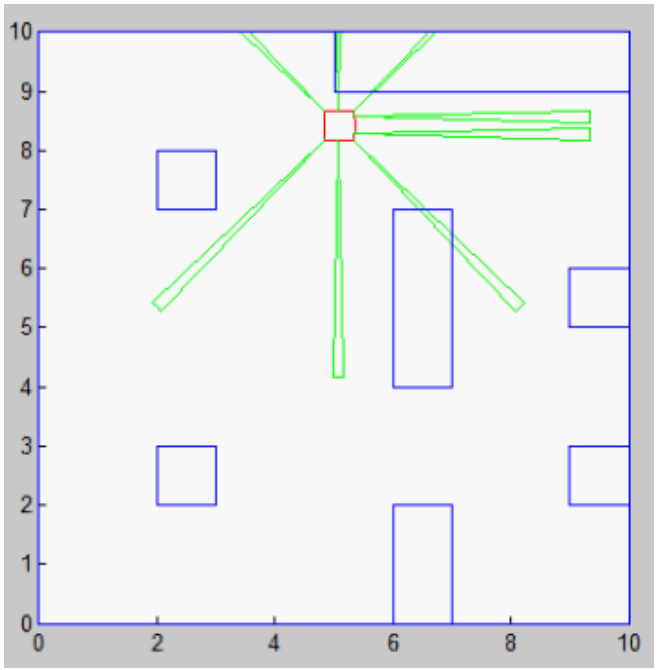


Figure 4.30: The wheelchair autonomously follows the wall by setting up a safe distance with it.

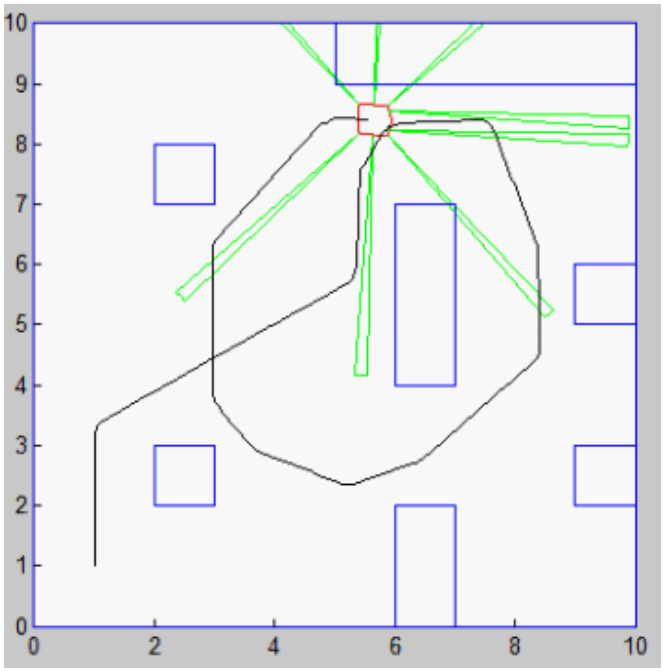


Figure 4.31: The wheelchair successfully travels through the crowded environment using its designed fuzzy logic based control system. It alters the user navigation inputs when necessary and changes the direction when senses any kind of collision.

The black line shows the traveled path in this simulation test.

# Chapter 5

# MCU Programming and Implementation

The microcontroller is the brain and control center of the system. It is an entire computer system contained within a single integrated circuit or chip and consists internally of a relatively simple CPU, clock, timers, I/O ports, and memory. Microcontroller operation is controlled by a user-written program interacting with the fixed hardware architecture resident within the microcontroller [57].

To program a microcontroller the following steps must be taken:

1. Write C programs in AVR Studio. AVR Studio is an integrated development environment that includes an editor, the assembler, HEX file downloader and a microcontroller emulator.
2. Compile them into a .hex file using the AVR-GCC compiler (which integrates into AVR Studio). GCC-based compiler is used which appears in AVR Studio as a plug-in tool.
3. Simulate the target AVR chip and debug the code within AVR Studio.
4. Program the actual chip using the Bootloader programmer software. Bootloader is software that is located in flash memory and makes it possible to connect the

microcontroller to the PC application directly. If not using a bootloader then a programmer device (STK500, ATMEL AVRISP mkII) must be used.

5. Once programmed, the chip runs the program in the circuit.

The wheelchair system software consists of five parts: USART, Joystick interface, Sensory circuit interface, Fuzzy algorithm control system, and the output signals (PWM). Each part will be discussed deeply in this chapter and the source codes are attached as Appendix B.

# 5.1 USART Connection

The Universal Synchronous and Asynchronous serial Receiver and Transmitter (USAR) is a type of serial communication and it allows the AVR to transmit and receive data serially to and from other devices - such as a computer or another AVR [58].

Microcontrollers must often exchange data with other microcontrollers or peripheral devices using parallel or serial techniques. In parallel techniques, an entire byte of data is typically sent simultaneously from the transmitting device to the receiver device. It is efficient from a time point of view but it requires eight separate lines for the data transfer. In serial transmission, a byte of data is sent a single bit at a time. Once 8 bits have been received at the receiver, the data byte is reconstructed. Although this is inefficient from a time point of view, it only requires a line (or two) to transmit the data. The Atmega644P microcontroller used in this project comes equipped with different types of serial communication subsystems such as USART, SPI (Serial Peripheral Interface), and TWI (Two-Wire Serial Interface).

The USART or UART is used to debug the code and test the sensors in this project. The USB connection has been used to transmit/receive data between the embedded microcontroller board and the laptop. To allow the two interfaced devices (laptop and the embedded microcontroller) to communicate together, we need to decide first on a baud rate for the communication. Baud is a measurement of transmission speed in asynchronous communication. The computer, any adaptors, and the UART must all agree on a single speed of information - 'bits per second' [59].

The baud rate used for this project is 115200. Terminal software is being used for data logging and measurement purposes. Note that the USART coding can be found in the Appendix B.1.

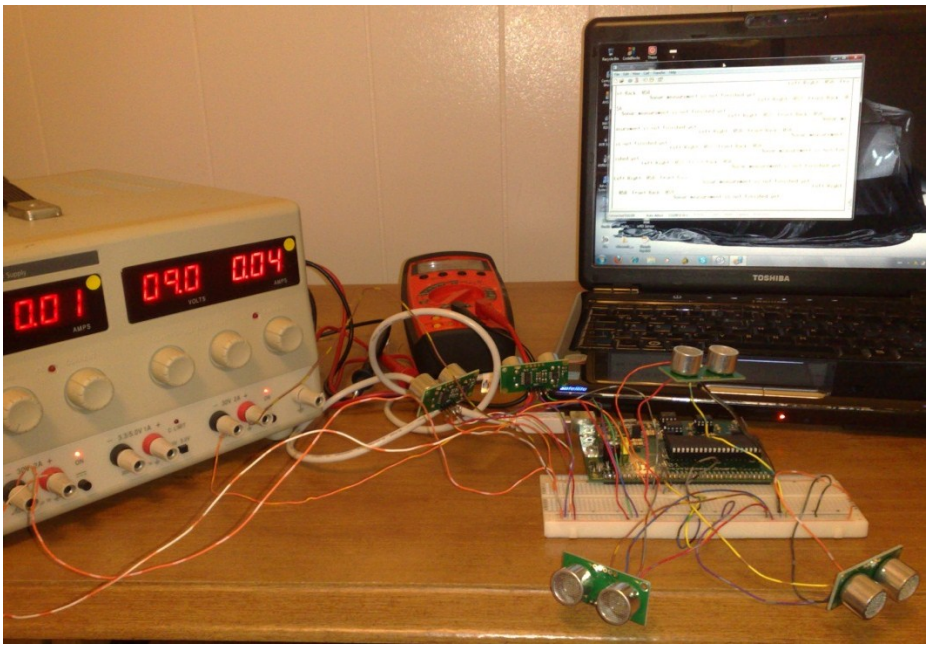

Figure 5.1: Sonar debugging with the help of the UASRT connection

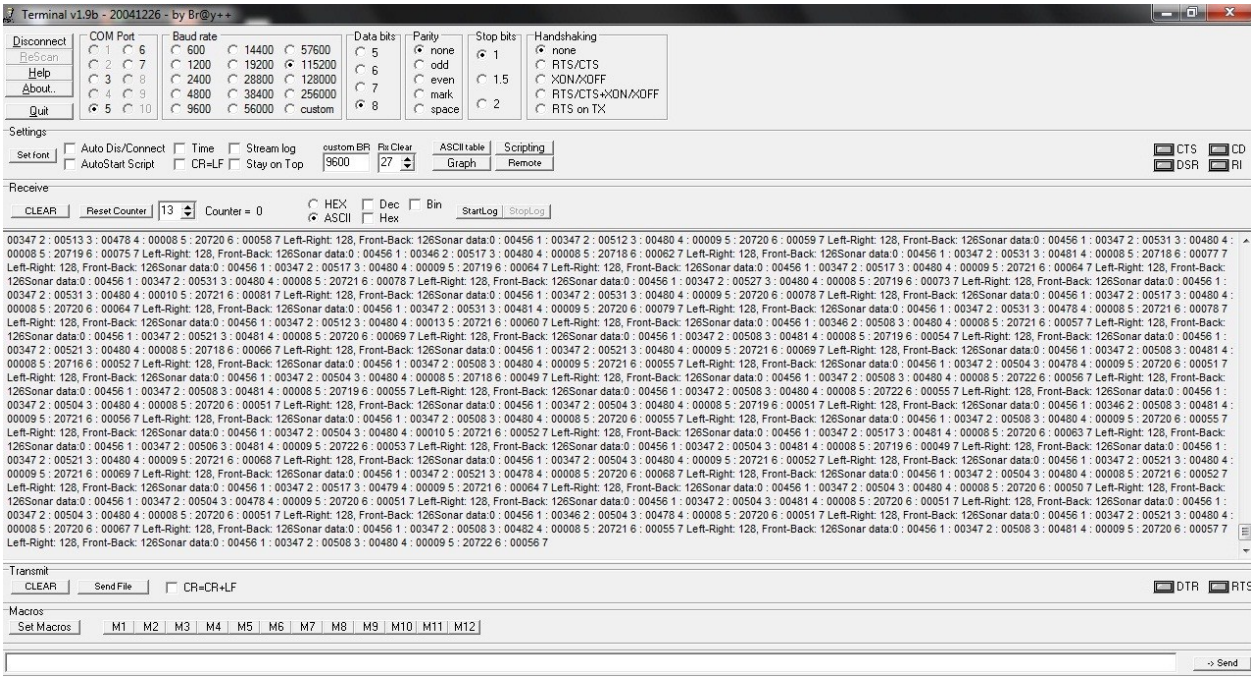


Figure 5.2: Terminal USART User Interface

# 5.2 Joystick Interface

The joystick used in the system is an analog joystick which consists of two independent 10K potentiometers with common ground. More details about the joystick and how it operates can be found on Chapter 2, 2.2.1, page 10 and Chapter 4, 4.2.2, page 56.

The joystick outputs two analog voltages ranging from 0V to 5V representing the knob position in the X and Y axis. The analog values can't be fed directly to the microcontroller since the microcontroller doesn't have an analog input and so the ADC (Analog to Digital Convertor) is being used.

The ADC converts an analog signal from the outside world into a binary representation suitable for use by the microcontroller [60].Atmega 644P has 8 ADC channels, allowing up to 8 analog sources to be attached to the microcontroller.

TheATmega644P ADC has 10 bit resolution, analog voltage between 0 and 5V will be encoded into one of 1024 binary representations between (000) 16 and (3FF) 16. This is a voltage resolution of approximately 4.88mV.

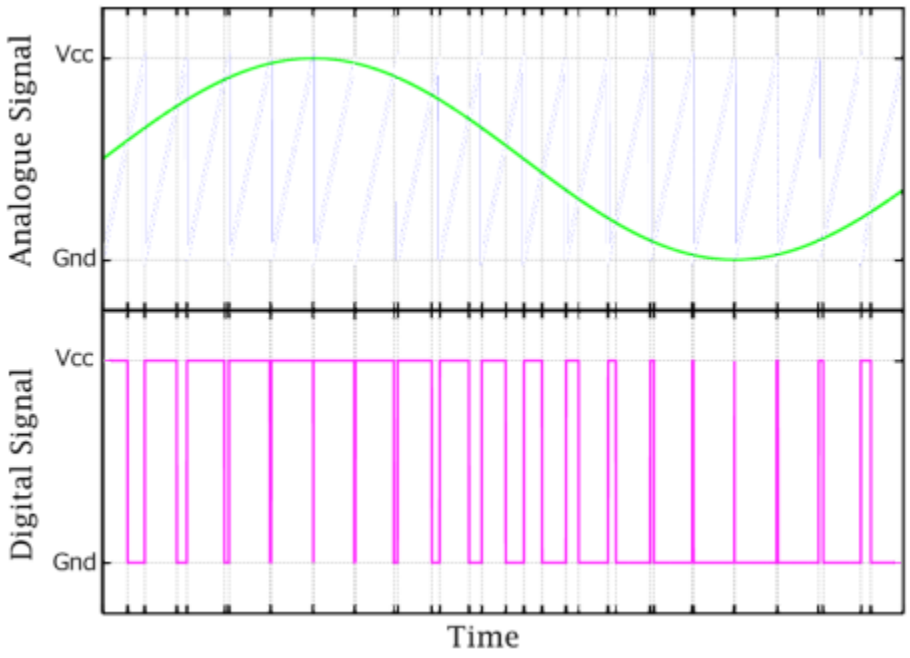


Figure 5.3: Converting the Analog signal to Digital

Table 5.1: Analog values of the joystick is converted to digital values using ADC

| Analog Voltage (V) | Digital (Hex) | Speed | Motor Direction |
|---|---|---|---|
| **0.00 – 2.00** | 00 - 66 | Max - Min | Clockwise |
| **2.02 – 2.98** | 67 - 98 | Zero | OFF |
| **3.00 – 5.00** | 99 - FF | Min - Max | Counter Clockwise |

ADC unit is powered with separate power supply pins AVCC with AGND, but AVCC must not differ ±0.3V of VCC [61]. Free running conversion has been used for this system which means that the conversion in continuous. Once initialized it takes 13 ADC cycles for single conversion. In this mode ADC data register has to be read before new value is written.

AVR ADC has a nice feature ADC noise reduction technique which allows performing conversion with minimal noise induced from AVR core and I/O peripherals. When noise cancelling is enabled microcontroller is put to sleep (CPU clock stops). After conversion completes, interrupt wakes processor to read and process converted data.

There are three steps needed to in order to make ADC work. First of all, ADC needs to be initialized. For this adc_init() function is written. Next step is to convert data itself. As we need to read values from two channels, there also multiplexing is needed. If the conversion mode invokes an interrupt after conversion is complete, third step is writing interrupt service routine. For the actual code please refer to the Appendix B.2.

# 5.3 Sensory Circuit Interface

Ultrasonic rangefinders are being used as the sensors for the system. The sensory circuit consist of eight ultrasonic sensors and has been implemented on the wheelchair in a way to give the system the maximum view. More details on the sensory circuit is given in chapter two section 2.5, page 27.

The ultrasonic sensor used in the system is a SRF05 made by Devantech. It has two operation modes; 1. Separate Trigger and Echo, 2. Single pin for both Trigger and Echo [62].

To save pins on the embedded microcontroller, mode 2 has been used to interface the sensors. To use this mode, the mode pin has been connected to the 0v Ground pin. In this mode the echo signal will appear on the same pin as the trigger signal. The SRF05 will not raise the echo line until 700uS after the end of the trigger signal so we have that long to turn the trigger pin around and make it an input and to have our pulse measuring code ready.

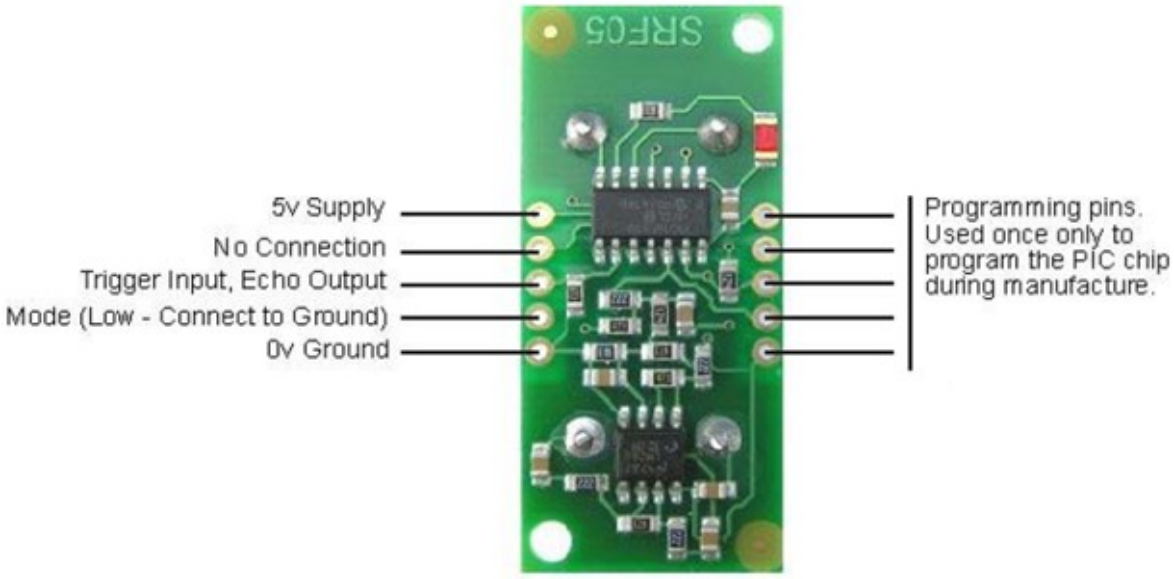


Figure 5.4: SRF05 Ultrasonic Sensor Connection Scheme [62]

The SRF05 Timing diagram for mode 2 is shown in Figure 5.5. A short 10uS pulse needs to be supplied to the trigger input to start the ranging. The SRF05 will send out an 8 cycle burst of

ultrasound at 40 kHz and raise its echo line high. It then listens for an echo, and as soon as it detects one it lowers the echo line again. The echo line is therefore a pulse whose width is proportional to the distance to the object. By timing the pulse the range is calculated in inches or centimeters. If nothing is detected then the SRF05 will lower its echo line anyway after about 30mS.

The SRF05 provides an echo pulse proportional to distance. The width of the pulse is measured in µS, then dividing it by 58 gives the distance in cm. To find it in inches the measured pulse must be divided by 148. The SRF05 can be triggered as fast as every 50mS, or 20 times each second.

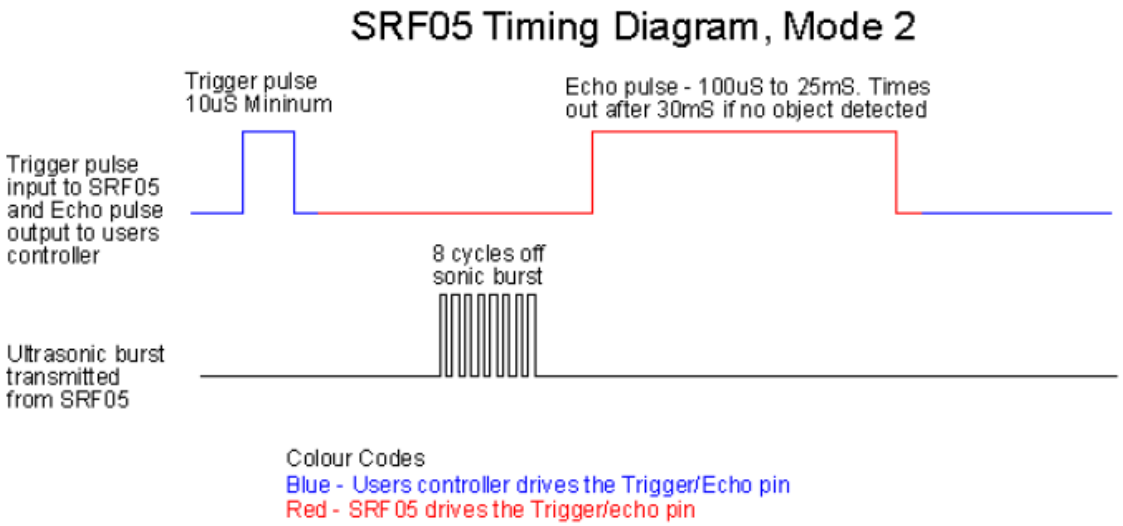


Figure 5.5: SRF05 Ultrasonic Timing Diagram when used in the Mode 2 [62]

The sensor code consists of the following steps (The source code is attached as Appendix B.3):

Send a pulse out

Start timer with count

Wait for the echo

Echo gotten

Stop timer

Take the timers count and calculate a distance

# 5.4 Pulse Width Modulation

Pulse Width Modulation or PWM means that we can generate a pulse whose width can be altered. Since microcontrollers are digital then their output pins can be either low (0v) or high (5v). However everything else is analog rather than just being on or off for example motors tend to need speed control, lighting may need to be dimmed, servos need to move to a particular position, buzzers need a sound frequency and etc. [63].

AVR microcontrollers have Analogue to Digitals Convertors (ADC) to convert a voltage from the analogue world to a number but do not have Digital to Analogue Convertors (DAC) to convert digital numbers back into variable voltages. And to solve that problem the PWM is the closest solution.

By turning an output pin repeatedly high and low very quickly then the result is an average of the amount of time the output is high. If it is always low the result is 0v, always high then the result is 5v, if half-and-half then the result is 2.5v.

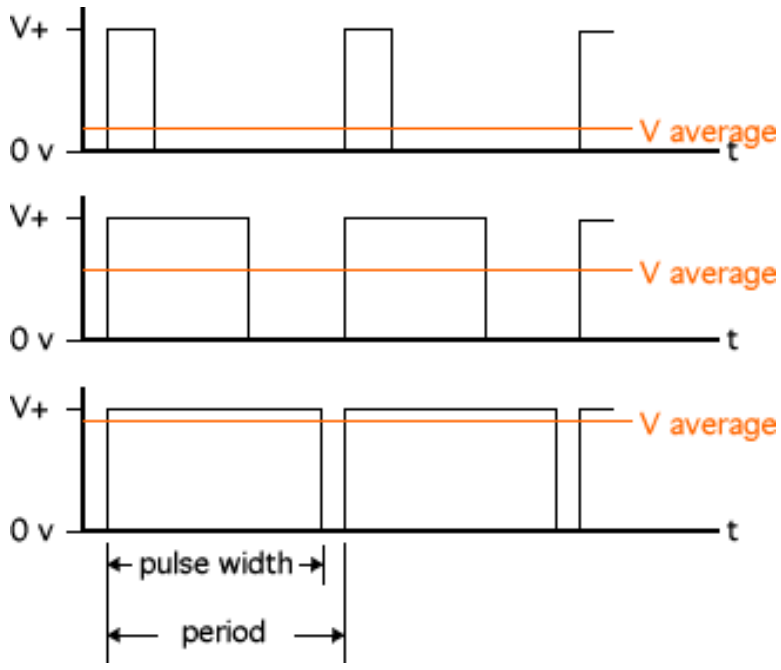


Figure 5.6: The speed of the motor increases by increasing the duty cycle of the PWM signal. In this figure the speed of the motor increases from top to bottom [64].

PWM signals are used in the prototype system to control the speed of the DC motors. The DC motors can't be connected directly to the output pins of the microcontroller or the microcontroller will blow up. That's why a motor controller should be used. More information about the motor controller can be found in chapter 2, section 2.2.2, and page 12.

To drive a DC motor a PWM signal will be used and the duty cycle will be varied to act as a throttle: 100% duty cycle = full speed, 0% duty cycle = stop, 50% duty cycle = half speed etc.

An R/C filter with component values of 10k ohms resistors and at 10uf capacitors are used before feeding the PWM channels to the motor controller. This will result in smoother motor operation. A PWM frequency of 5000Hz is used.

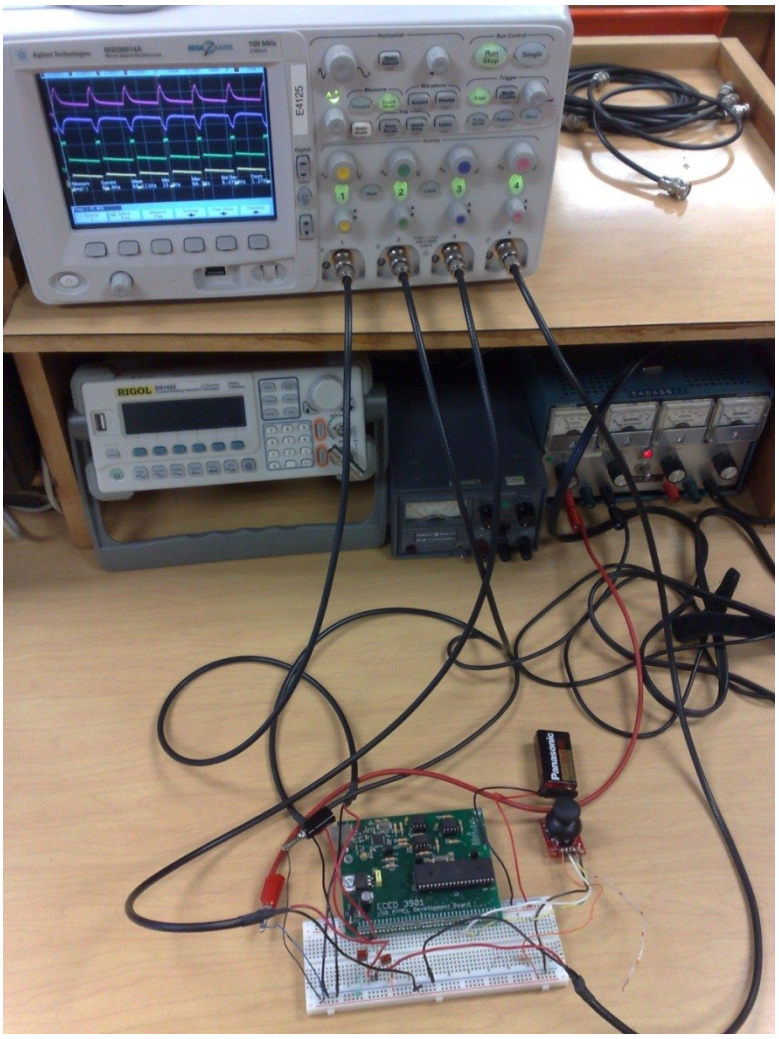


Figure 5.7: Testing the output PWM signals by changing the joystick position. The top two signals in the oscilloscope screen represent the PWM 1 and 2 with the use of RC filter and the two signals in the bottom are without the RC filter

# 5.5 Fuzzy Logic Implementation

The fuzzy logic control algorithm has been developed with MATLAB fuzzy toolbox and has been proven to work – at least – in the simulation environment. The next step is to transfer the algorithm to the microcontroller. The easiest way to transfer the designed control system algorithm from fuzzy logic toolbox to C or C++ language is to use especial software's such as Byte Craft or fuzyTech which will automatically transform the linguistic variables and membership functions to the desired high level language. fuzzyTech even goes one step further and claims that it can transform the fuzzy or neural-fuzzy algorithms directly to the desired microcontroller (AVR, Microchip, …) embedded language. These are all good products but none of them has been used in this project.

MATLAB has a stand-alone C code fuzzy interface engine which makes two C files "fismain.c" and "fis.c" from each project built with fuzzy logic toolbox and stores them in the directory of the toolbox and which are provided as the source codes for the stand-alone fuzzy interface engine or can be embedded in other external applications. These FIS files are ANSI C compatible but need to be changed a lot to fit into an 8-bit microcontroller with limited on-chip memory. In fact, the whole algorithm has been coded and redesigned in order to make it Atmega compatible and the FIS files have been used as a reference. First, the fuzzy control algorithm has been written in C language and then it has been re-coded in AVR embedded C language. The fuzzy logic control algorithm source code can be found in the Appendix B section.

Of course the work with MATLAB has not been wasted. First, with using MATLAB, the fuzzy algorithm has been modeled, the output values have been checked for different inputs, graphical models has been used to debug the system, and the last but not least the SIMULINK has been used to prove the validity of the designed control algorithm.

# 5.6 System Implementation

To interface the Embedded Microcontroller board with the sonar's, joystick, and PWM outputs a PCB (Printed Circuit Board) called control interface board has been designed. EAGLE 5.11.0 software has been used to draw the schematic of the circuit and design the board layout. The manufacturing files such as the Gerber and drill files have been sent to the university to build the board. The schematic and board layout can be seen in the figure 5.8 and figure 5.9.

Some of the key features of the designed PCB are as following:

- For PWM connection on the board as it can be seen in the figure 5.8, an RC low pass filter has been used for each PWM output. The RC filter will keep the noise to minimum; will cut the variations in the signal and leaving only the continuous component of the signal or its medium value to pass through. The components value used for the filter are R= 10 kΩ and C= 10 μF.
- Additional pull up resistors has been used for each sonar echo/trigger line. A pull up is a resistor that 'pulls up' the voltage at a certain pin. More precisely, it is a resistor placed through the supply rail (VCC, 'up'), and the desired pin (The ECHO line in the case of sonar sensor). That forces than when the transistor is not in conduction, the voltage on the collector is close to VCC. Not using pull up resistors may cause false voltage readings and cause a power supply grounding problem.
- The ADC has a separate analog supply voltage pin, AVCC. AVCC must not differ more than ±0.3V from VCC. VCC is the standard power in for the digital circuitry and AVCC is for the analog. The separation is for when there is a mixed mode circuit or board so the power lines can be run separately to avoid crosstalk noise. Digital switching noise may affect the analog signals and vice versa. So it's better to separates them on the board. Low pass filter has been used when connecting the AVCC to VCC which allows the DC voltage through while blocking the high frequency digital surges as demanded from the switching circuits. The low pass filter used for noise cancelling consists of a 10μH inductor and a 100nF capacitor as the Atmega644P datasheet suggests.

- In an AVR Atmega microcontroller internal reference voltages of nominally 1.1V, 2.56V or AVCC are provided On-chip. The voltage reference may be externally decoupled at the AREF pin by a capacitor for better noise performance. VREF tied to 5V (VCC for example) is the voltage that the analog system compares the analog input to. Since in the current program the ADC has been set to use the external VCC as the reference voltage, The AREF pin has been connected to the 5V VCC through a decoupling and noise cancelling capacitor of 0.1 μF.
- The analog joystick used for this project has five pins, one connected to ground, one to 5V VCC and two which are representing the vertical and horizontal state of the joystick have been connected to PA0 and PA1 (ADC pins of the microcontroller) via 4.7K resistors. The resistors along with the ADC capacitors act as filter against the short time noise pulses. There is also another pin (SELECT) which is not connected anywhere since the press button is not being used in the program.
- The 5V VCC used to power the sonars and joystick comes from the motor driver. The Demension 2x25V motor driver used in this system is capable of outputting 5V and GND which has been connected to the PCB through the power port.

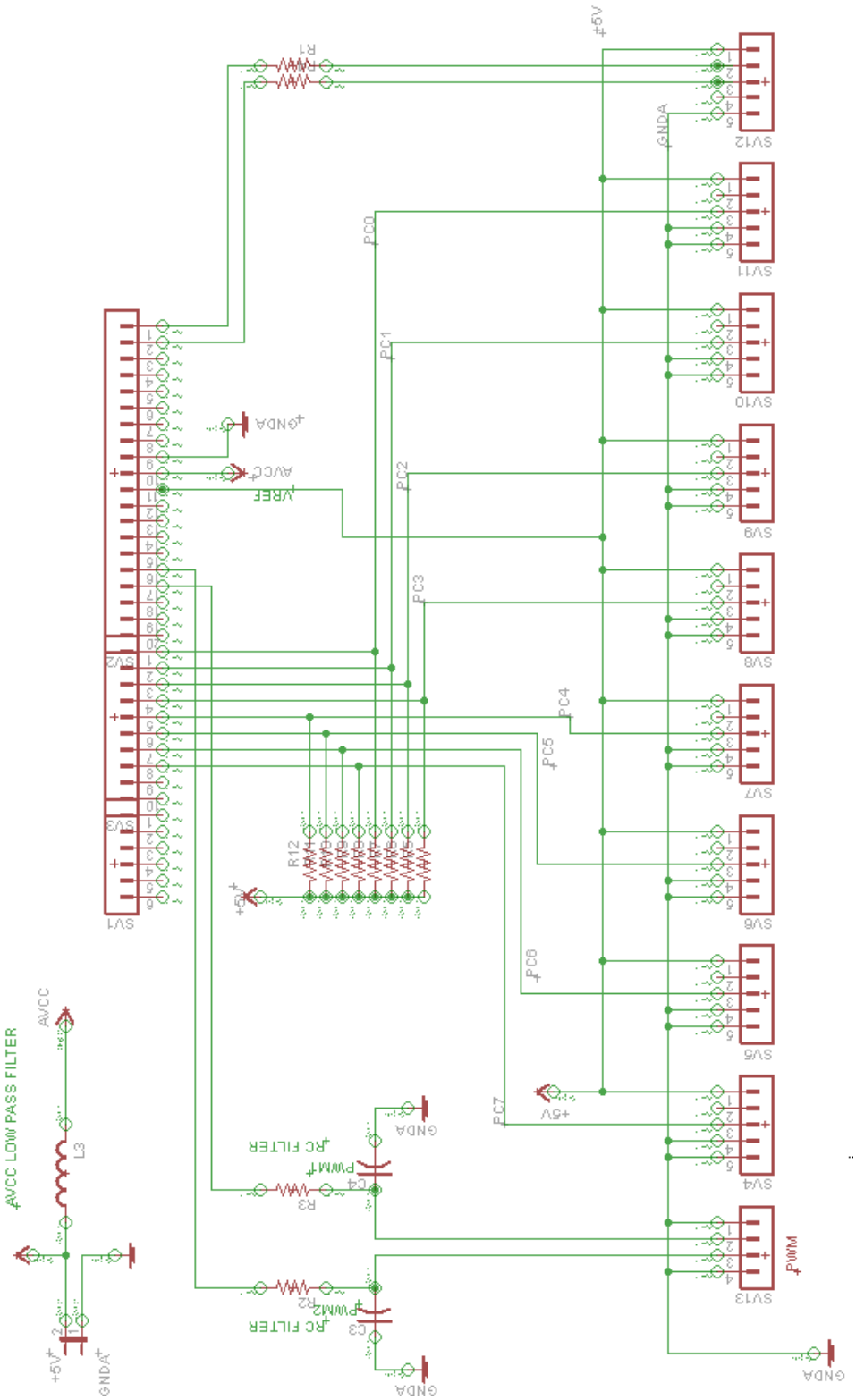


Figure 5.8: The control interface PCB schematic

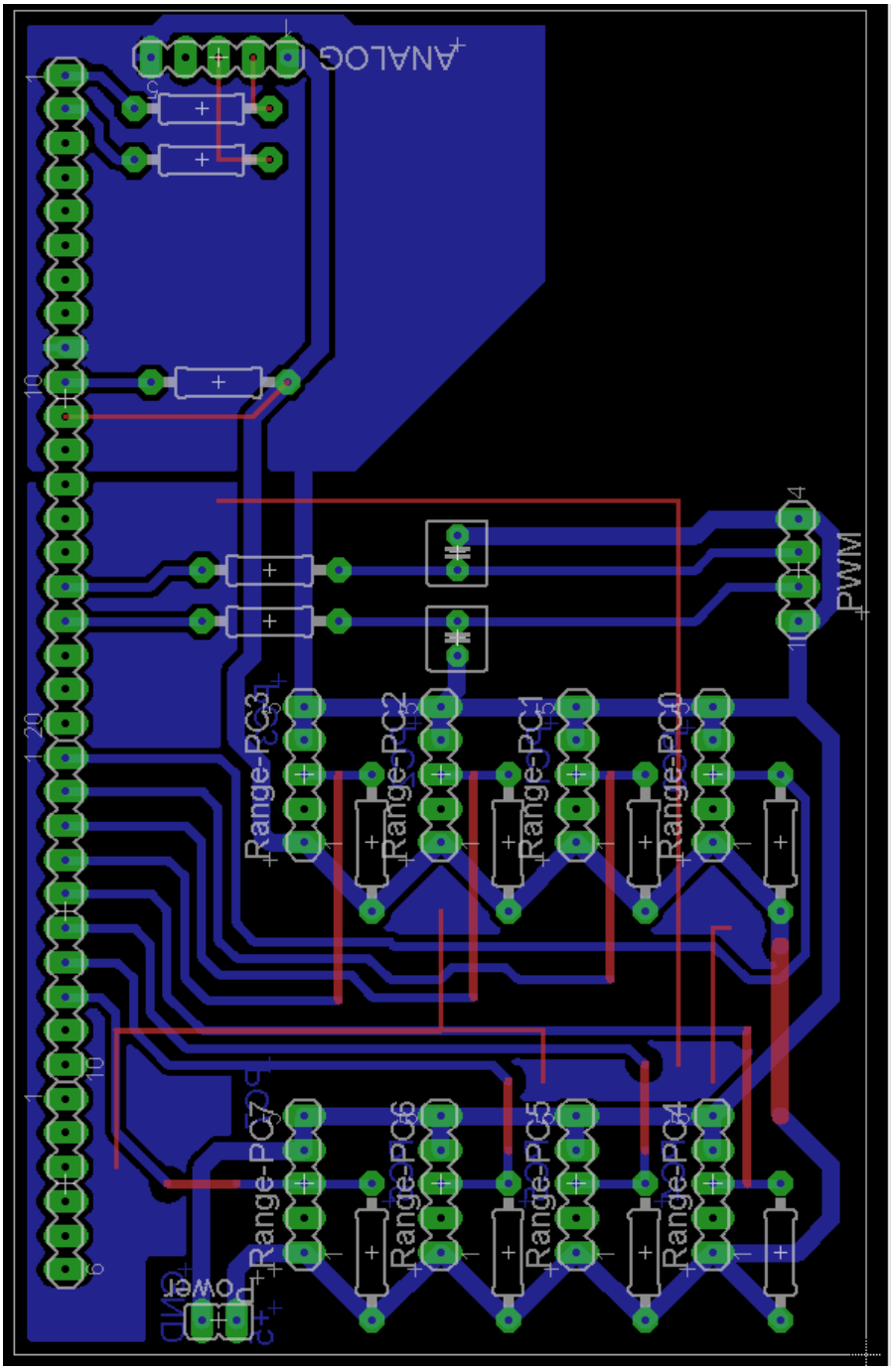


Figure 5.9: The control interface PCB lay-out

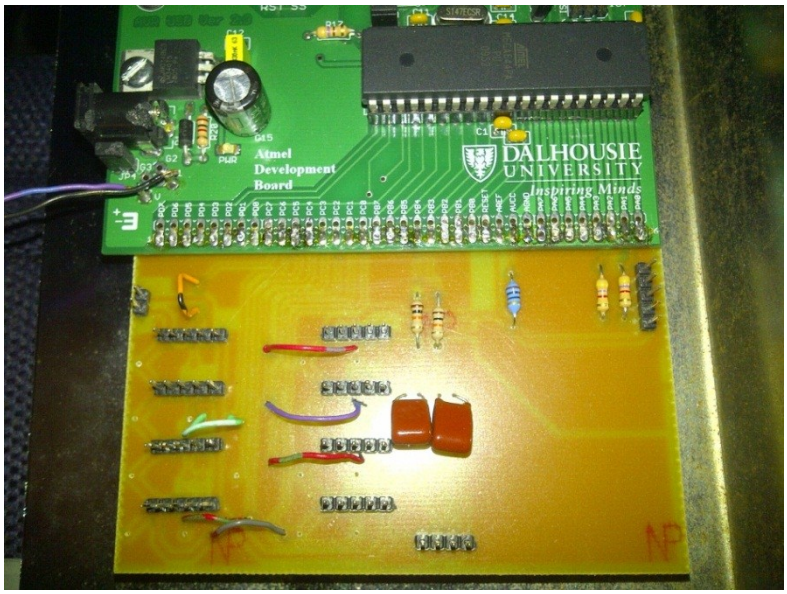


Figure 5.11: The control interface board

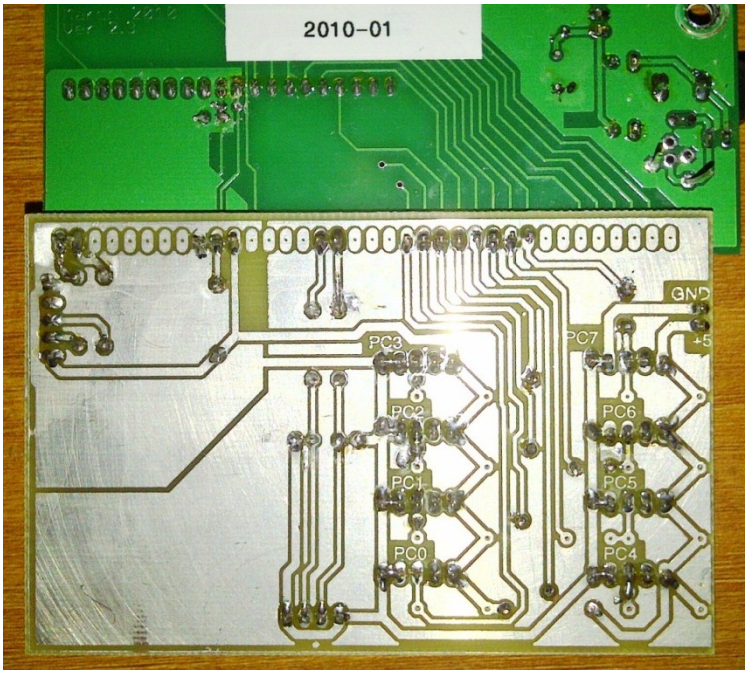


Figure 5.10: The control interface board bottom layer

# Chapter 6

# Conclusion and Future Works

In this chapter, the effectiveness of the prototype system in providing a collision free journey to the user is discussed along with the future scope of work.

## 6.1 Conclusion

In this thesis, the smart wheelchair prototype system was designed and tested to demonstrate the first steps toward a commercially smart wheelchair. The main criteria in designing such system was to build a non-complex, non-expensive system that can be added to a normal power wheelchair to augment the navigation ability of the user and ensuring a safe and collision free journey.

One of the factors that differentiate this system with the previous attempts is the use of Fuzzy Logic as the primary high level control system. Fuzzy logic membership functions give the advantage of designing the system based on the level of uncertainty the system is facing and because it doesn't necessary need precise inputs, inexpensive sensors can be used and therefor resulting a decrease in the overall cost of the system.

The other important difference between this system and other designs is the use of microcontroller instead of a computer or laptop. This will allow the system to be lighter, energy-efficient, more portable, inexpensive and easier to interface with different wheelchairs control systems. Currently the microcontroller is programmed so it can interface with any wheelchair controller using PWM (pulse width modulation) as input.

Using MATLAB Simulink and testing the system in simulation environment under different obstacle configurations and taking different routes proved the ability and validity of the designed system and algorithm in avoiding any possible collision. The wheelchair was able to avoid any objects in its way with maximum comfort which is very important when dealing with wheelchairs where the comfort and safety of the user can't be overlooked. The wheelchair moves smoothly because of the fuzzy logic control and the addition of the membership functions comparing to the systems using model-based control (IF X… THEN Z), which are usually jerky.

In addition to the simulation, the actual hardware has been also built. The designed prototype system which consists of multiple ultrasonic sensors, embedded microcontroller and control system algorithm has been implemented on the Everest and Jennings wheelchair which was provided by the control systems and robotic lab. The wheelchair was turned into a power wheelchair first and then a smart wheelchair. Multiple tests have been undertaken and although the results were promising, further improvements on the hardware and especially the sensors are necessary.

## 6.2 Future Works

Although the simulation works flawlessly, further tweaking of the hardware would lead to safer and more accurate results. For improved reliability, the sensitivity of the sensors needs to be revaluated. The process of revaluating the sensors sensitivity will lead to some reprogramming and debugging. Ultimately the debugging will result in safe and smart option of transportation for people of disable community.

In addition, the following recommendation can improve the functionality of the smart wheelchair system:

- Using multiple sensors combination: Each type of sensor has different advantages and disadvantages and the system reliability will be increased by using different kind of sensors. For example when using just sonars, reflection varies based on the surface material and this problem especially shown itself in case of wall-following.
- Drop offs detector: There is a need for a new sensory circuit to identify the sudden drop-offs likes curbs, stairs, and potholes.
- Software customization: The control system algorithm, navigation software and thresholds have been designed to meet the researcher's criteria's and to balance the safety and functionality of the wheelchair. However, it's better to allow the user to manually change the software thresholds or allowing the chair to automatically adapt these software thresholds based on the user's behaviour and observations of the environment.
- Testing the system with members of the disable community for better adjustments of the system.

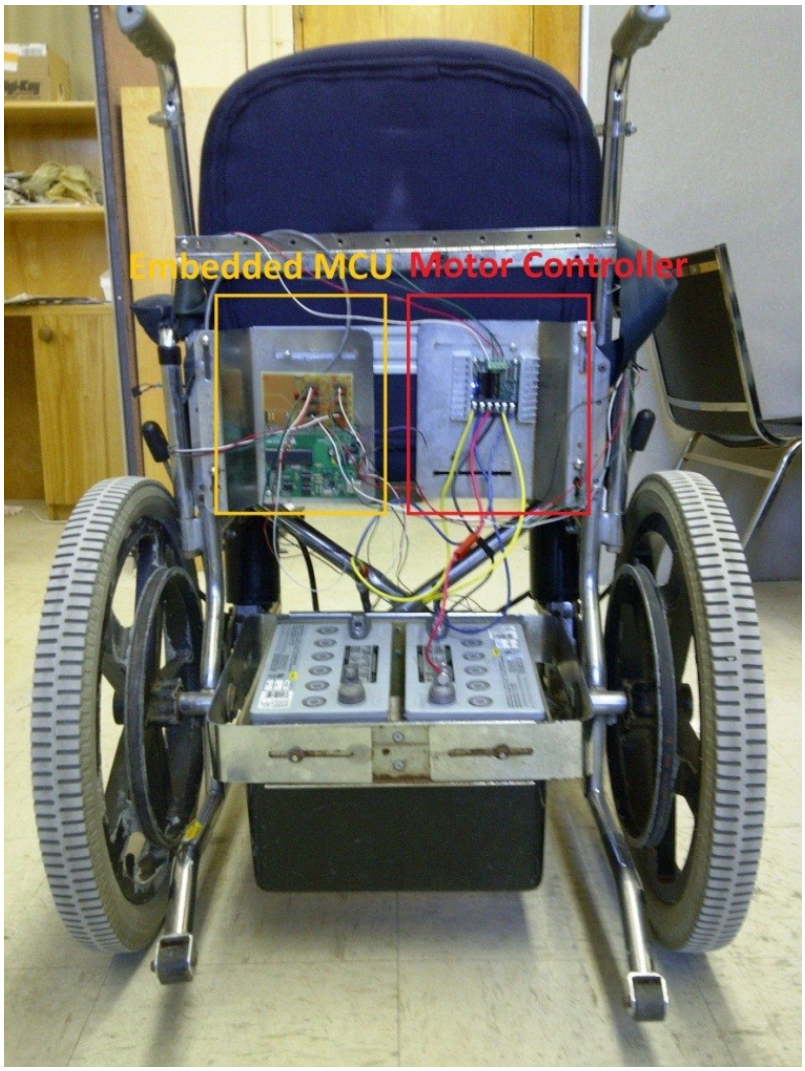


Figure 6.1: The smart wheelchair motor controller and microcontroller integration


Figure 6.2: The smart wheelchair sonar sensors and joystick implementation

# Bibliography

[1] Richard C. Simpson, “Smart wheelchairs: A literature review”, Journal of Rehabilitation Research & Development. Volume 42, Number 4, Pages 423–436 July/August 2005.

[2] Smart wheelchair, Wikipedia, “http://en.wikipedia.org/wiki/Smart_wheelchair”. [Accessed: 23 Sep. 2009].

[3] Rosenbloom L. Consequences of impaired movement: a hypothesis and review. In: Movement and child development. Holt KS, editor. London, England: HarperCollins; 1975.

[4] Gignac MA, Cotta C, Badley EM. Adaptation to chronic illness and disability and its relationship to perceptions of independence and dependence. J Gerontol B Psychol Sci Soc Sci. 2000.

[5] Wright BAP. Physical disability—A psychosocial approach. New York: Addison-Wesley; 1983.

[6] Pope A, Tarlov A, editors. Disability in America: Toward a national agenda for prevention. Washington (DC): National Academies Press; 1991.

[7] Richard Simpson, Edmund LoPresti, Steve Hayashi, Illah Nourbakhsh, David Miller, “The Smart Wheelchair Component System”, Journal of rehabilitation research & develop. May/June 2004.

[8] L. Fehr, W. Langbein, and S. Skaar, “Adequacy of power wheelchair control interfaces for persons with severe disabilities: A clinical survey”, Journal of Rehabilitation Research and Development Vol.37,No.3, June 2000.

[9] Gomi T, Ide K. The development of an intelligent wheelchair. Conference on Intelligent Vehicles; Sep 19–20, 1996.

[10] Nisbet PD, Craig J, Odor JP, Aitken S. “Smart” wheelchairs for mobility training. Technol Disabil. 1995.

[11] Connell J, Viola P. Cooperative control of a semi-autonomousmobile robot. Robotics and Automation: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA); 1990 May 13–18; Cincinnati, OH. Piscataway (NJ): IEEE; 1990.

[12] Bourhis G, Moumen K, Pino P, Rohmer S, Pruski A. Assisted navigation for a powered wheelchair. Systems Engineering in the Service of Humans: Proceedings of the IEEE International Conference on Systems, Man and Cybernetics; 1993 Oct 17–20; Le Touquet, France. Piscataway (NJ): IEEE; 1993.

[13] Borgolte U, Hoyer H, Buehler C, Heck H, Hoelper R. Architectural concepts of a semi-autonomous wheelchair. J Intell Robotic Syst. 1998.

[14] Katevas NI, Sgouros NM, Tzafestas SG, Papakonstantinou G, Beattie P, Bishop JM, Tsanakas P, Koutsouris D. The autonomous mobile robot SENARIO: A sensor-aided intelligent navigation system for powered wheelchairs. IEEE Robot Autom Mag. 1997.

[15] Simpson RC, Poirot D, Baxter MF. The Hephaestus smart wheelchair system. IEEE Trans Neural Syst Rehabil Eng. 2002.

[16] J. Crisman and M. Cleary, "Progress on the deictic controlled wheelchair," Assistive Technology and Artificial Intelligence. Springer Verlag, 1998.

[17] P. Nisbet, J. Craig, P. Odor, and S. Aitken, " Smart wheelchairs for mobility training", Technology Disability, Vol. 5, 1995.

[18] Levine SP, Bell DA, Jaros LA, Simpson RC, Koren Y, Borenstein J. "The NavChair assistive wheelchair navigation system", IEEE Trans Rehabil Eng. Dec 1999.

[19] Miller DP, Slack MG. Design and testing of a low-cost robotic wheelchair prototype. Auton Robots. 1995.

[20] Bourhis G, Moumen K, Pino P, Rohmer S, Pruski A. "Assisted navigation for a powered wheelchair". p. 553–58. IEEE; 1993.

[21] Abigail Drury, Rittika Shamsuddin, Melissa Frechette, Audrey Lee St. John, Dan Barry, William Kennedy, "Autonomous Wheelchair", "wiki.cs.mtholyoke.edu/mediawiki/rmc_images/6/64/Spaulding10.ppt". [Accessed: 15 July 2009].

[22] Gabriel Pires and Urbano Nunes, "A Wheelchair Steered through Voice Commands and Assisted by a Reactive Fuzzy-Logic Controller", Journal of Intelligent and Robotic Systems 34: 301–314, 2002.

[23] Yanco HA. Wheelesley: a robotic wheelchair system: Indoor navigation and user interface. In: Assistive technolgy and artificial intelligence. Mittal VO, Yanco HA, Aronis J, Simpson RC, editors. New York: Springer-Verlag; 1998.

[24] Schilling K, Roth H, Lieb R, Stutzle H. Sensors to improve the safety for wheelchair users. 3rd Annual TIDE Congress; 1998 July; Helsinki, Finland. Helsinki: TIDE; 1998.

[25] Yoder JD, Baumgartner ET, Skaar SB. Initial results in the development of a guidance system for a powered wheelchair. IEEE Trans Rehabil Eng. 1996.

[26] A. Saffiotti, "The uses of fuzzy logic in autonomous robot navigation", Soft Computing 1 (1997) pp. 180-197, Springer-Verlag 1997.

[27] Steven D. Kaehler, Seattle Robotics, "Fuzzy Logic - An Introduction", http://www.seattlerobotics.org/encoder/dec97/fuzzy.html. [Accessed: 17 Aug. 2009].

[28] Tom harris, "How joysticks works", "http://electronics.howstuffworks.com/joystick.htm/printable". [Accessed: 12 Jan. 2009].

[29] Lesson 5, choosing a motor controller, how to make a robot, GoRobotics, "http://www.robotshop.com/gorobotics/how-to-make-a-robot/how-to-make-a-robot-lesson-5-motor-controller". [Accessed: 15 Jan. 2010].

[30] Stall current, Combat Robot Wiki, "http://combots.net/wiki/index.php/Stall_current". [Accessed: 10 Feb. 2010].

[31] Dimension engineering, Sabertooth dual 25A regenerative motor driver manual, http://www.dimensionengineering.com/datasheets/Sabertooth2x25.pdf. [Accessed: 15 Feb. 2010].

[32] Power engineering, The wheelchair site,
"http://www.thewheelchairsite.com/power-wheelchairs.aspx". [Accessed: 10 March 2010].

[33] Marek Andrezej Perkowski, Electrical and Computer Engineering, Portland State University, Rehabilitation robots course mterials, http://web.cecs.pdx.edu/~mperkows/Rehabilitation_Robots/. [Accessed: 21 Sep. 2010]

[34] Ismail H. Altas, Electrical and Electronics Engineering, Karadeniz Technical University, ELK 5320 Nero Fuzzy Systems, Project 37 Dynamic model of a permanent magnet DC motor. 1992.

[35] A. Fattouh, Y. Dadam, D. T. Pham, “MATLAB-Based 3D Model of a Powered Wheelchair”, Laboratory of Automatic Control and Automation Faculty of Electrical and Electronic Engineering, University of Aleppo, Aleppo, Syria. July 2008.

[36] Lesson 4, Understanding Microcontrollers, How to Make a Robot, GoRobotics, “http://www.robotshop.com/gorobotics/articles/microcontrollers/how-to-make-a-robot-lesson-4-understanding-microcontrollers”. [Accessed: 13 Jan. 2011].

[37] Atmel AVR Atmega 644P 8-bit microcontroller manual, http://www.atmel.com/dyn/resources/prod_documents/8011S.pdf. [Accessed: 27 Aug. 2009].

[38] Gregory Dudek, Michael Jenkin, “Computational Principles of Mobile Robotics”, Cambridge University Press, 1 edition, Sensors, pp. 51-62, Feb 28th 2000.

[39] Sonar Made Simple, RidgeSoft, LLC, “http://www.ridgesoft.com/articles/sonar/SonarMadeSimple.pdf”. [Accesses: 10 Sep. 2009].

[40] Robot Sonars, Sensors, Society of Robots, “http://www.societyofrobots.com/sensors_sonar.shtml”. [Accesses: 15 Sep. 2009].

[41] Control system, Wikipedia, http://en.wikipedia.org/wiki/Control_system. [Accesses: 05 Feb. 2011].

[42] L. A. Zadeh, "Fuzzy sets", Information and Control Vol. 8 (3): 338–353, 1965.

[43] Siripun Thongchai and Kazuhiko Kawamura, “Application of Fuzzy Control to a Sonar-Based Obstacle Avoidance Mobile Robot”, International Conference on Control Applications, Anchorage, Alaska, USA. September 25-27, 2000.

[44] Fuzzy logic, Wikipedia, http://en.wikipedia.org/wiki/Fuzzy_logic. [Accesses: 17 Oct. 2009].

[45] Fuzzy logic, TLAs Glossary, DER Engineering, http://dereng.com/tlas_glossary.htm. [Accesses: 24 Oct. 2009].

[46] Tijana T. Ivancevic, Bojan Jovanovic and Sasa Markovi, “Fuzzy control strategies in human operator and sport modeling “, Fuzzy Information and Engineering, Volume 2, Number 2, pp 157-186, SpringerLink. July 2009.

[47] Didier Dubois, Henri Prade, “Fuzzy elements in a fuzzy set”, RIT, Universite Paul Sabatier, Toulouse, France.

[48] Timothy J. Ross, “Fuzzy logic with engineering applications“, page 91, second edition, WILEY Publication, 2004.

[49] Sanjay Krishnankutty Alonso, “Mamdani’s fuzzy inference method”, eMathTeacher, Polytechnic University of Madrid. http://www.dma.fi.upm.es/java/fuzzy/fuzzyinf/. [Accessed: 07 Sep. 2010].

[50] S. N. Sivanandam, S. Sumathi and S. N. Deepa, “Introduction to fuzzy logic using MATLAB”, Springer Publication, 2006.

[51] Prasan Pitiranggon, Nunthika Benjathepanun, Somsri Banditvilai, and Veera Boonjing, “Fuzzy Rules Generation and Extraction from Support Vector Machine Based on Kernel Function Firing Signals”, International journal of engineering and applied sciences, 2010.

[52] MATLAB Basic Tutorial, Control tutorials for MATLA and Simulink, “http://www.library.cmu.edu/ctms/ctms/basic/basic.htm”. [Accessed: 16 Nov. 2010].

[53] Simulink – simulation and model-based design, MathWorks, “http://www.mathworks.com/products/simulink/”. [Accessed: 22 Dec. 2010].

[54] H.R. Singh, Abdul Mobin, Sanjeev Kumar, Sundeep Chauhan and S.S. Agrawal, “Design and development of voice/joystick operated microcontroller based intelligent motorised wheelchair”, IEEE TENCON, pp. 1573 – 1576, 1999.

[55] Ian Cavers, “A Brief Introductory Guide to MATLAB”, UBC computer science, CPSC 303. http://www.cs.ubc.ca/~ascher/542-403/MatlabGuide.pdf. [Accessed: 13 Dec. 2010].

[56] What is a GUI?, MathWorks, “http://www.mathworks.com/help/techdoc/creating_guis/f2-998436.html”. [Accessed: 17 Jan. 2011].

[57] Steven F. Barrett, Daniel J. Pack, Chap 1, Atmel AVR Architecture Overview, pp. 1-24, Atmel AVR microcontroller primer: programming and interfacing, MORGAN & CLAYPUL, 2008.

[58] Dean Camera, “Using the USART - Serial communications”, AVRFreaks, 2006.
http://www.avrfreaks.net/index.php?name=PNphpBB2&file=viewtopic&t=45341. [Accessed: 11 Sep. 2010].

[59] Microcontroller UART Tutorial, Society of Robots, http://www.societyofrobots.com/microcontroller_uart.shtml. [Accessed: 13 Oct. 2010].

[60] Steven F. Barrett, Daniel J. Pack, Chap 3, Analog-to-Digital Conversion, pp. 41-64, Atmel AVR microcontroller primer: programming and interfacing, MORGAN & CLAYPUL, 2008.

[61] Running TX433 and RX433 RF modules with AVR microcontrollers, WINAVR-GCC Tutorials, http://winavr.scienceprog.com/. [Accessed: 22 Aug. 2009].

[62] Devantech SRF05 Ultrasonic Range Finders datasheet manual, http://www.robotstorehk.com/sensors/doc/srf05tech.pdf. [Accessed: 08 Nov. 2009].

[63] Webbot, "PWM - an overview", Society of Robots, http://www.societyofrobots.com/member_tutorials/node/228. [Accessed: 03 Sep. 2010].

[64] Digital I/O Ports, Society of Robots, http://www.societyofrobots.com/microcontroller_tutorial.shtml. [Accessed: 05 Sep. 2010].

# Appendix A: MATLAB M-files and the GUI

## A.1 PlotChair.m

```
function h=PlotChair(Xpos,Ypos,Angle)

LocalChair=[-0.25 0.25;
     0.25 0.25;
     0.3 0;
     0.25 -0.25;
     -0.25 -0.25;
     -0.25 0.25];

 Sxlocal=[0 0.25 0.25 0.25 0.25 0 -0.25 -0.25];
 Sylocal=[0.25 0.25 0.15 -0.15 -0.25 -0.25 0.25 -0.25];
 Sanglelocal=[pi/2 pi/4 0 0 -pi/4 -pi/2 3*pi/4 -3*pi/4];

GlobalChair=LocalChair;

for i=1:length(LocalChair)
    GlobalChair(i,1)=Xpos+LocalChair(i,1)*cos(Angle)-
LocalChair(i,2)*sin(Angle);
GlobalChair(i,2)=Ypos+LocalChair(i,1)*sin(Angle)+LocalChair(i,2)*cos(Angle);
end;

cla;
hold on;
plot(GlobalChair(:,1),GlobalChair(:,2),'r');
for i=1:length(Sxlocal)
    PlotSensor(Xpos,Ypos,Angle,Sxlocal(i),Sylocal(i),Sanglelocal(i));
end;
axis square;

Room = [0 0;
    0 10;
    10 10;
    10 0;
    0 0];
Table1 = [2 2;
    2 3;
    3 3;
    3 2;
    2 2];
Table2 = [2 7;
    2 8;
    3 8;
```

```
    3 7;
    2 7];
Table3 = [6 4;
    6 7;
    7 7;
    7 4;
    6 4];
Table4 = [6 0;
    6 2;
    7 2;
    7 0;
    6 0];
Table5 = [5 9;
    5 10;
    10 10;
    10 9;
    5 9];
Table6 = [9 5;
    9 6;
    10 6;
    10 5;
    9 5];
Table7 = [9 2;
    9 3;
    10 3;
    10 2;
    9 2];

plot(Room(:,1),Room(:,2),'b');
plot(Table1(:,1),Table1(:,2),'b');
plot(Table2(:,1),Table2(:,2),'b');
plot(Table3(:,1),Table3(:,2),'b');
plot(Table4(:,1),Table4(:,2),'b');
plot(Table5(:,1),Table5(:,2),'b');
plot(Table6(:,1),Table6(:,2),'b');
plot(Table7(:,1),Table7(:,2),'b');
xlim([0 10]);
ylim([0 10]);
hold off;

global Xj Yj
Xj = 0;
Yj = 0;
pause(0.001);
h = 1;
```

# A.2 PlotSensor.m

```
function PlotSensor(Xpos,Ypos,Angle,Sxlocal,Sylocal,Sanglelocal)

Sensor=[0 0;
    4 0.1;
    4 0;
    4 -0.1;
    0 0];

SensorChair=Sensor;

for i=1:length(Sensor)
    SensorChair(i,1)=Sxlocal+Sensor(i,1)*cos(Sanglelocal)-
Sensor(i,2)*sin(Sanglelocal);

SensorChair(i,2)=Sylocal+Sensor(i,1)*sin(Sanglelocal)+Sensor(i,2)*cos(Sanglel
ocal);
end;

SensorGlobal=SensorChair;

for i=1:length(SensorChair)
    SensorGlobal(i,1)=Xpos+SensorChair(i,1)*cos(Angle)-
SensorChair(i,2)*sin(Angle);

SensorGlobal(i,2)=Ypos+SensorChair(i,1)*sin(Angle)+SensorChair(i,2)*cos(Angle
);
end;

plot(SensorGlobal(:,1),SensorGlobal(:,2),'g');
```

# A.3 GetSensor.m

```
function sensors=GetSensors(Xpos,Ypos,Angle)

Sensor=[0 0;
    4 0];
SensorChair=Sensor;

 Sxlocal=[0 0.25 0.25 0.25 0.25 0 -0.25 -0.25];
 Sylocal=[0.25 0.25 0.15 -0.15 -0.25 -0.25 0.25 -0.25];
 Sanglelocal=[pi/2 pi/4 0 0 -pi/4 -pi/2 3*pi/4 -3*pi/4];
```

```
sensors =[];

for j=1:length(Sxlocal)

    for i=1:length(Sensor)
        SensorChair(i,1)=Sxlocal(j)+Sensor(i,1)*cos(Sanglelocal(j))-
Sensor(i,2)*sin(Sanglelocal(j));

SensorChair(i,2)=Sylocal(j)+Sensor(i,1)*sin(Sanglelocal(j))+Sensor(i,2)*cos(S
anglelocal(j));
    end;

    SensorGlobal=SensorChair;

    for i=1:length(SensorChair)
        SensorGlobal(i,1)=Xpos+SensorChair(i,1)*cos(Angle)-
SensorChair(i,2)*sin(Angle);

SensorGlobal(i,2)=Ypos+SensorChair(i,1)*sin(Angle)+SensorChair(i,2)*cos(Angle
);
    end;

    s=Meassurement(SensorGlobal(1,1),SensorGlobal(1,2),Angle+Sanglelocal(j));

    sensors = [sensors,s];

end;
```

# A.4 Measurement.m

```
function s=Meassurement(Xini,Yini,Angle)

 Room = [0 10;
    0 10];
Table1 = [2 3;
    2 3];
Table2 = [2 3;
    7 8];
Table3 = [6 7;
    4 7];
Table4 = [6 7;
    0 2];
Table5 = [5 10;
    9 10];
Table6 = [9 10;
    5 6];
Table7 = [9 10;
```

```
    2 3];
s=4;

 for i=0:0.01:4
    xpos = Xini + i*cos(Angle);
    ypos = Yini + i*sin(Angle);

    if ((xpos < Room(1,1)) | (xpos > Room(1,2)) | (ypos < Room(2,1)) | (ypos
> Room(2,2)) | ((((xpos > Table1(1,1)) & ...
            (xpos < Table1(1,2))) & ((ypos > Table1(2,1)) & (ypos <
Table1(2,2)))) | ...
            (((xpos > Table2(1,1)) & (xpos < Table2(1,2))) & ((ypos >
Table2(2,1)) & ...
            (ypos < Table2(2,2)))) | (((xpos > Table3(1,1)) & (xpos <
Table3(1,2))) & ...
            ((ypos > Table3(2,1)) & (ypos < Table3(2,2)))) | (((xpos >
Table4(1,1)) & (xpos < Table4(1,2))) & ...
            ((ypos > Table4(2,1)) & (ypos < Table4(2,2)))) | (((xpos >
Table5(1,1)) & (xpos < Table5(1,2))) & ...
            ((ypos > Table5(2,1)) & (ypos < Table5(2,2)))) | (((xpos >
Table6(1,1)) & (xpos < Table6(1,2))) & ...
            ((ypos > Table6(2,1)) & (ypos < Table6(2,2)))) | (((xpos >
Table7(1,1)) & (xpos < Table7(1,2))) & ...
            ((ypos > Table7(2,1)) & (ypos < Table7(2,2)))))) & (i < s)
        s = i;
        break;
    end;
end.
```

# A.5 Joystick Graphical User Interface Source Code

```
function varargout = GUI(varargin)

% GUI M-file for GUI.fig
%      GUI, by itself, creates a new GUI or raises the existing
%      singleton*.
%
%      H = GUI returns the handle to a new GUI or the handle to
%      the existing singleton*.
%
%      GUI('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in GUI.M with the given input arguments.
%
%      GUI('Property','Value',...) creates a new GUI or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before GUI_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
```

```
%      stop.  All inputs are passed to GUI_OpeningFcn via varargin.
%

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @GUI_OpeningFcn, ...
                   'gui_OutputFcn',  @GUI_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);

if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End of the initialization process


% --- Executes upon GUI initialization.
function GUI_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to GUI (see VARARGIN)

% Choose default command line output for GUI
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes GUI wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = GUI_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes when left button is pressed.
function left_Callback(hObject, eventdata, handles)
% hObject    handle to left (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Jcontrol
Jcontrol = 3;

% --- Executes when left-up button is pressed.
function leftup_Callback(hObject, eventdata, handles)
global Jcontrol
Jcontrol = 2;

% --- Executes when up button is pressed.
function up_Callback(hObject, eventdata, handles)
global Jcontrol
Jcontrol = 1;

% --- Executes when right-up button is pressed.
function rightup_Callback(hObject, eventdata, handles)
global Jcontrol
Jcontrol = 8;

% --- Executes when right button is pressed.
function right_Callback(hObject, eventdata, handles)
global Jcontrol
Jcontrol = 7;

% --- Executes when left-down button is pressed.
function leftdown_Callback(hObject, eventdata, handles)
global Jcontrol
Jcontrol = 4;

% --- Executes when down button is pressed.
function down_Callback(hObject, eventdata, handles)
global Jcontrol
Jcontrol = 5;

% --- Executes when right-down button is pressed.
function rightdown_Callback(hObject, eventdata, handles)
global Jcontrol
Jcontrol = 6;
```

# Appendix B: The AVR Atmega644P Embedded MCU Source Codes

## B.1 USART

```
//*****************************************************************************
//  HEADER avr_UART0.h
//*****************************************************************************
#ifndef _AVR_UART0_H_
        #define  _AVR_UART0_H_

//*****************************************************************************
#include "common.h"

//*****************************************************************************
// constants

// USART buffers dimensions: MUST be 2, 4, 8, 16, 32, etc.
#define  UART0_TX_BUFF_SIZE 128
#define  UART0_RX_BUFF_SIZE 32

// uart speeds index - low order register byte only, high order byte is 0
enum
{
        SPEED0_38400,
        SPEED0_57600,
        SPEED0_115200,
        //
        SPEED0_MAX
};

#define  UCSR0A_VAL   (1<<U2X0)                                  // double speed
#define  UCSR0B_VAL   ((1<<RXCIE0)|(1<<RXEN0)|(1<<TXEN0))
#define  UCSR0C_VAL   ((1<<UCSZ01)|(1<<UCSZ00))                  // 8 bits, asynch.,no parity, one stop

// macros
#define  UART0_TX_INTERRUPT_ENABLE              sbi (UCSR0B, UDRIE0)
#define  UART0_TX_INTERRUPT_DISABLE             cbi (UCSR0B, UDRIE0)
```

```
#define  UART0_RX_INTERRUPT_ENABLE              sbi (UCSR0B, RXCIE0)
#define  UART0_RX_INTERRUPT_DISABLE             cbi (UCSR0B, RXCIE0)

//****************************************************************************
// exported functions
UCHAR UART0_init (UCHAR speed);
UCHAR UART0_get_RX_status (void);
UCHAR UART0_get_TX_status (void);
UCHAR UART0_send_data (UCHAR* data, UCHAR len);
UCHAR UART0_send_long_data (UCHAR* data, UINT len);
UCHAR UART0_get_data (UCHAR* data);

unsigned char receiveByte(void);
void transmitByte(unsigned char);
void transmitString(unsigned char*);

#endif    //_AVR_UART0_H_
//****************************************************************************
// end of file
//****************************************************************************


//****************************************************************************
// SOURCE avr_UART0.c
//****************************************************************************
#include "avr_UART0.h"

//****************************************************************************
// local functions

// local constants
#define  UART0_TX_MASK        (UART0_TX_BUFF_SIZE - 1)
#define  UART0_RX_MASK        (UART0_RX_BUFF_SIZE - 1)

// USART speed definition with double speed and 14.7 MHz
#define  UBRR0H_VAL                0
#define  UBRR0L_38400             47
#define  UBRR0L_57600             31

#ifdef CLOCK_10MHZ
        #define  UBRR0L_115200     10
```

```
#else
        #define  UBRR0L_115200          15
#endif
static const UCHAR UART0_speed[] =
{
        UBRR0L_38400,
        UBRR0L_57600,
        UBRR0L_115200
};

// exported variables


// local variables
SVUCHAR TxData0[UART0_TX_BUFF_SIZE];
SVUCHAR      TxStartPointer0;
SVUCHAR      TxEndPointer0;
SVUCHAR TxStatus0;

SVUCHAR RxData0[UART0_RX_BUFF_SIZE];
SVUCHAR      RxStartPointer0;
SVUCHAR      RxEndPointer0;
SVUCHAR RxStatus0;
/*****************************************************************************
| NAME:                          UART0_init
| ABSTRACT:                      uart0 initialization
| PARAMETER:  none
| RETURN:                        none
*****************************************************************************/
UCHAR UART0_init (UCHAR speed)
{       // UART registers initialization
        if (speed >= SPEED0_MAX)
        {
                return S_FAIL;                  // invalid speed requested
        }

        UCSR0A = 0;
        UCSR0A = UCSR0A_VAL;            //0
        UBRR0H = UBRR0H_VAL;
        UBRR0L = UART0_speed[speed];
        UCSR0C = UCSR0C_VAL;
        UCSR0B = UCSR0B_VAL;
```

```
        // variables initialization
        TxStartPointer0 = 0;
        TxEndPointer0 = 0;
        TxStatus0 = 0;

        RxStartPointer0 = 0;
        RxEndPointer0 = 0;
        RxStatus0 = 0;

        return S_OK;
}

/*****************************************************************************
| NAME:                 UART0_get_RX_status
| ABSTRACT:             clears RxStatus0 but returns its real value
| PARAMETER:  none
| RETURN:               RxStatus0
*****************************************************************************/
UCHAR UART0_get_RX_status (void)
{
UCHAR i;
        UART0_RX_INTERRUPT_DISABLE;
        i = RxStatus0;
        UART0_RX_INTERRUPT_ENABLE;
        return i;
}

/*****************************************************************************
| NAME:                 UART0_get_TX_status
| ABSTRACT:
| PARAMETER:  none
| RETURN:               TxStatus0
*****************************************************************************/
UCHAR UART0_get_TX_status (void)
{
UCHAR i;
        UART0_TX_INTERRUPT_DISABLE;
        i = TxStatus0;
        UART0_TX_INTERRUPT_ENABLE;
        return i;
}
```

```
/*****************************************************************************
| NAME:                         UART0_get_data
| ABSTRACT:
| PARAMETER:                    pointer where the received data have to be saved
| RETURN:                       number of received bytes
*****************************************************************************/
UCHAR UART0_get_data (UCHAR* data)
{
UCHAR i = 0;

        UART0_RX_INTERRUPT_DISABLE;
        RxEndPointer0 &= UART0_TX_MASK;
        RxStartPointer0 &= UART0_TX_MASK;

        if (RxStartPointer0 == RxEndPointer0)
        {
                UART0_RX_INTERRUPT_ENABLE;
                return 0;           // there are not any received bytes or overflow appears
        }

        while (RxEndPointer0 != RxStartPointer0)
        {
                *(data + i) = RxData0[RxEndPointer0];
                RxEndPointer0 ++;
                RxEndPointer0 &= UART0_TX_MASK;
                i++;
        }

        UART0_RX_INTERRUPT_ENABLE;
        return i;
}

/*****************************************************************************
| NAME:                         UART0_send_data
| ABSTRACT:
| PARAMETER:                    pointer to data to be sent, number of bytes
| RETURN:                       number of sent bytes
*****************************************************************************/
UCHAR UART0_send_data (UCHAR* data, UCHAR len)
{
UCHAR i = 0;
```

```
UCHAR FreeBytes;

        UART0_TX_INTERRUPT_DISABLE;
        TxEndPointer0 &= UART0_TX_MASK;
        TxStartPointer0 &= UART0_TX_MASK;

        // looks for the free space in the local buffer
        if (TxEndPointer0 == TxStartPointer0)
        {       // the local buffer is whole empty
                FreeBytes = UART0_TX_BUFF_SIZE;
        }
        else
        if (TxEndPointer0 < TxStartPointer0)
        {
                FreeBytes = TxStartPointer0 - TxEndPointer0;
        }
        else
        {
                FreeBytes = TxEndPointer0 - TxStartPointer0;
        }

        // copy the data to the local buffer
        while ((i < len) && (i < FreeBytes))
        {
                TxData0[TxStartPointer0 & UART0_TX_MASK] = *(data + i);
                TxStartPointer0 ++;
                i ++;
        };

        UART0_TX_INTERRUPT_ENABLE;
        return i;
}

/*****************************************************************************
| NAME:                         UART0_send_long_data
| ABSTRACT:
| PARAMETER:                    pointer to data to be sent, number of bytes (up to 64 kB)
| RETURN:                       number of sent bytes
*****************************************************************************/
UCHAR UART0_send_long_data (UCHAR* data, UINT len)
{
UINT uLength = len;
```

```
UINT uSentNum = 0;
UINT temp;

        while (uSentNum < uLength)
        {
                while (TxStatus0 != S_OK);                      // wait for empty buffer

                if ((uLength - uSentNum) > UART0_TX_BUFF_SIZE)
                {
            temp = (UINT)UART0_send_data ((UCHAR*)(data + uSentNum), UART0_TX_BUFF_SIZE);
                }
                else
                {
            temp = (UINT)UART0_send_data ((UCHAR*)(data + uSentNum), (UCHAR)(uLength - uSentNum));
                }

                uSentNum += temp;
        }

        return S_OK;
}

//*****************************************************************************
// interrupt routines
/*****************************************************************************
| NAME:                         SIGNAL (SIG_UART0_RECV)
| ABSTRACT:                     Uart0 RX interrupt handler
| PARAMETER:  none
| RETURN:                       none
*****************************************************************************/
//SIGNAL (SIG_UART0_RECV)
ISR (USART0_RX_vect)
{
UCHAR temp = UDR0;              // read data register

        RxData0[RxStartPointer0] = temp;
        RxStartPointer0 ++;
        RxStartPointer0 &= UART0_RX_MASK;

        if (RxStartPointer0 == (RxEndPointer0 & UART0_RX_MASK))
        {       // error: buffer overflow
                RxStatus0 = S_OVERFLOW;
```

```
        }
}

/*****************************************************************************
| NAME:                         SIGNAL (SIG_UART0_DATA)
| ABSTRACT:                     Uart0 TX Data Register Empty interrupt handler
| PARAMETER:  none
| RETURN:                       none
*****************************************************************************/
//SIGNAL (SIG_UART0_DATA)
ISR (USART0_UDRE_vect)
{
        UDR0 = TxData0[TxEndPointer0];
        TxEndPointer0 ++;
        TxEndPointer0 &= UART0_TX_MASK;

        if (TxEndPointer0 == (TxStartPointer0 & UART0_TX_MASK))
        {       // end of the data to be transmitted
                UART0_TX_INTERRUPT_DISABLE;
                TxStatus0 = S_OK;
        }
        else
        {
                TxStatus0 = S_BUSY;
        }
}

/*****************************************************************************
| NAME:                         SIGNAL (SIG_UART0_TRANS)
| ABSTRACT:                     Uart0 TX complete interrupt handler, practically not used
| PARAMETER:  none
| RETURN:                       none
*****************************************************************************/
//SIGNAL (SIG_UART0_TRANS)
ISR (USART0_TX_vect)
{
        cbi (UCSR0B, TXCIE0);   // to be sure, disable TX interrupt
}


//*****************************************************************************
// direct access functions: first time only:
```

```
//***************************************************
//Function to receive a single byte
//***************************************************
unsigned char receiveByte( void )
{
	unsigned char data, status;

	while(!(UCSR0A & (1<<RXC0)));		// Wait for incoming data

	status = UCSR0A;
	data = UDR0;

	return(data);
}

//***************************************************
//Function to transmit a single byte
//***************************************************
void transmitByte( unsigned char data )
{
	while ( !(UCSR0A & (1<<UDRE0)) )
		;					/* Wait for empty transmit buffer */
	UDR0 = data;					/* Start transmition */
}

//***************************************************
//Function to transmit a string in RAM
//***************************************************
void transmitString(unsigned char* string)
{
	while (*string)
	{
		transmitByte(*string++);
	}
}
//*****************************************************************************
// end of file
//*****************************************************************************
```

# B.2 Joystick

```
//****************************************************************************
// joystick.h
//****************************************************************************
#include "common.h"

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

void Joystick_init (void);
void JoystickMeasurementTask (void);
UCHAR JoystickGetData (UCHAR Direction);

//****************************************************************************
// end of file
//****************************************************************************


//****************************************************************************
// joystick.c
//****************************************************************************
#include "common.h"
#include "joystick.h"

//****************************************************************************
#define  MEASUREMENT_NUMBER              4                  // to use average value from these
                                                            // measurement results
#define  INDEX_MASK                      (MEASUREMENT_NUMBER - 1)

//****************************************************************************
// local
static UCHAR FrontBackDirection[MEASUREMENT_NUMBER];
static UCHAR LeftRightDirection[MEASUREMENT_NUMBER];
static UCHAR MeasurementCounter;

/*****************************************************************************
| NAME:                 Joystick_init
| ABSTRACT:             component initialization after reset
| PARAMETER:  none
| RETURN:               none
```

```
*****************************************************************************/
void Joystick_init (void)
{
	MeasurementCounter = 0;
	JoystickTimeOut = JOYSTICK_MEASUREMENT_TIME;

	ADMUX = (1<<ADLAR);			// external Vref = Vcc, right adjustment

	// ADC input clock 14MHz/8, approximately 10 microseconds conversion time
	ADCSRA = (1<<ADEN) | (1<<ADPS1) | (1<<ADPS0);
	ADCSRB = 0;				// ADC free running mode

	// set analogue inputs
	DIDR0 = (1<<ADC_FRONT_BACK) | (1<<ADC_LEFT_RIGHT);
}
/*****************************************************************************
| NAME:			JoystickMeasurement
| ABSTRACT:		do measurement of both directions
| PARAMETER:  none
| RETURN:		none
*****************************************************************************/
void JoystickMeasurementTask (void)
{
	if (JoystickTimeOut != 0)
	{
		return;
	}

	JoystickTimeOut = JOYSTICK_MEASUREMENT_TIME;

	// doing front-back joystick direction measurement
	ADMUX = (1<<ADLAR) + ADC_FRONT_BACK;
	sbi (ADCSRA, ADSC);

	while ((ADCSRA & (1<<ADSC)) != 0)
	{	// wait conversion end
	}
	FrontBackDirection[MeasurementCounter & INDEX_MASK] = ADCH;

	// doing left-right joystick direction measurement
	ADMUX = (1<<ADLAR) + ADC_LEFT_RIGHT;
	sbi (ADCSRA, ADSC);
```

```
        while ((ADCSRA & (1<<ADSC)) != 0)
        {       // wait conversion end
        }
        LeftRightDirection [MeasurementCounter & INDEX_MASK] = ADCH;

        MeasurementCounter ++;
}

/*****************************************************************************
| NAME:                         JoystickGetData
| ABSTRACT:                     gives the average measurement value
| PARAMETER:                    direction
| RETURN:                       average measurement value
*****************************************************************************/
UCHAR JoystickGetData (UCHAR Direction)
{
        UINT summ = 0;
        UCHAR idx;

        for (idx = 0; idx < MEASUREMENT_NUMBER; idx ++)
        {
                if (Direction == ADC_FRONT_BACK)
                {
                        summ += (UINT)(FrontBackDirection[idx]);
                }
                else
                {
                        summ += (UINT)(LeftRightDirection[idx]);
                }
        }

        return (UCHAR)(summ / MEASUREMENT_NUMBER);
}

//*****************************************************************************
// end of file
//*****************************************************************************
```

# B.3 Sensors

```
//*****************************************************************************
// sonar.h
//*****************************************************************************
#ifndef _SONAR_H
#define  _SONAR_H

//*****************************************************************************
// exported functions
void InitSonar (void);
void SonarMeasurementTask (void);
UCHAR IsSonarMeasurementDone (void);
void SonarStartMeasurement (void);
void SonarDataConversion (void);

//*****************************************************************************
// The sensors are placed in clockwise direction with 45 degrees angle distance
// between them.
// sonar lines definitions:
#define  SONAR_FRONT                    0       // 0 degrees
#define  SONAR_FRONT_RIGHT              1       // 45
#define  SONAR_RIGHT                    2       // 90
#define  SONAR_BACK_RIGHT               3       // 135
#define  SONAR_BACK                     4       // 180
#define  SONAR_BACK_LEFT                5       // 225
#define  SONAR_LEFT                     6       // 270
#define  SONAR_FRONT_LEFT               7       // 315
//*****************************************************************************
// exported variables
extern UINT SonarDataInCentimeters [];
extern volatile UCHAR SonarMeasurementTimeOut;

#endif    //_SONAR_H
//*****************************************************************************
// end of file
//*****************************************************************************
```

```
/*****************************************************************************
// joystick.c
//*****************************************************************************
#include "common.h"
#include "sonar.h"

//*****************************************************************************
#define  MEASUREMENT_NUMBER              4              // to use average value from these
                                                        // measurement results
#define  INDEX_MASK                      (MEASUREMENT_NUMBER - 1)

//*****************************************************************************
// local
static UCHAR FrontBackDirection[MEASUREMENT_NUMBER];
static UCHAR LeftRightDirection[MEASUREMENT_NUMBER];
static UCHAR MeasurementCounter;

/*****************************************************************************
| NAME:                 Joystick_init
| ABSTRACT:             component initialization after reset
| PARAMETER:  none
| RETURN:               none
*****************************************************************************/
void Joystick_init (void)
{
        MeasurementCounter = 0;
        JoystickTimeOut = JOYSTICK_MEASUREMENT_TIME;

        ADMUX = (1<<ADLAR);             // external Vref = Vcc, right adjustment

        // ADC input clock 14MHz/8, approximately 10 microseconds conversion time
        ADCSRA = (1<<ADEN) | (1<<ADPS1) | (1<<ADPS0);
        ADCSRB = 0;                     // ADC free running mode

        // set analogue inputs
        DIDR0 = (1<<ADC_FRONT_BACK) | (1<<ADC_LEFT_RIGHT);
}
/*****************************************************************************
| NAME:             JoystickMeasurement
| ABSTRACT:         do measurement of both directions
```

```
| PARAMETER:            none
| RETURN:               none
*****************************************************************************/
void JoystickMeasurementTask (void)
{
        if (JoystickTimeOut != 0)
        {
                return;
        }

        JoystickTimeOut = JOYSTICK_MEASUREMENT_TIME;

        // doing front-back joystick direction measurement
        ADMUX = (1<<ADLAR) + ADC_FRONT_BACK;
        sbi (ADCSRA, ADSC);

        while ((ADCSRA & (1<<ADSC)) != 0)
        {       // wait conversion end
        }
        FrontBackDirection[MeasurementCounter & INDEX_MASK] = ADCH;

        // doing left-right joystick direction measurement
        ADMUX = (1<<ADLAR) + ADC_LEFT_RIGHT;
        sbi (ADCSRA, ADSC);

        while ((ADCSRA & (1<<ADSC)) != 0)
        {       // wait conversion end
        }
        LeftRightDirection [MeasurementCounter & INDEX_MASK] = ADCH;

        MeasurementCounter ++;
}

/*****************************************************************************
| NAME:                 JoystickGetData
| ABSTRACT:             gives the average measurement value
| PARAMETER:            direction
| RETURN:               average measurement value
*****************************************************************************/
UCHAR JoystickGetData (UCHAR Direction)
{
        UINT summ = 0;
```

```
	UCHAR idx;

	for (idx = 0; idx < MEASUREMENT_NUMBER; idx ++)
	{
		if (Direction == ADC_FRONT_BACK)
		{
			summ += (UINT)(FrontBackDirection[idx]);
		}
		else
		{
			summ += (UINT)(LeftRightDirection[idx]);
		}
	}

	return (UCHAR)(summ / MEASUREMENT_NUMBER);
}

//***************************************************************************
// end of file
//***************************************************************************
```

# B.4 Common

```
/*****************************************************************************
// common.h        common definitions for all project
//*****************************************************************************
#ifndef _COMMON_H_
#define   _COMMON_H_

//*****************************************************************************
#define  CLOCK_10MHZ
//
// if defined the whole PORTC will be initialized as inputs with internal pull ups
// and by connecting to ground it is possible to check one or more pins. PORTC status
// sends via UART on every ~ 800 milliseconds
//#definePORTC_INPUT_TEST

// if defined below the whole PORTC is initialized as output and
// its value increments on every milliseconds, so it's possible to see each pin by oscilloscope.
//#definePORTC_OUTPUT_TEST
//
//*****************************************************************************
#define define __AVR_ATmega644__

#ifdef CLOCK_10MHZ
        #define F_CPU 10000000UL                    // 10.00 MHz
#else
        #define F_CPU 14745600UL                    // 14.74 MHz
#endif
//*****************************************************************************
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <avr/eeprom.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <inttypes.h>
#include <avr/wdt.h>
//#include <avr/iom128.h>
#include <avr/iom644.h>
```

```
#include <math.h>
//*****************************************************************************
// preprocessor predefinitions
#define  UCHAR          unsigned char
#define  UINT           unsigned int
#define  ULONG          unsigned long
#define  LONG           long

#define  VUCHAR         volatile unsigned char
#define  VUINT          volatile unsigned int
#define  VULONG         volatile unsigned long
#define  VLONG          volatile long

#define  SVCHAR         static volatile char
#define  SVUCHAR        static volatile unsigned char
#define  SVUINT         static volatile unsigned int
#define  SVINT          static volatile int

// for compatibility with old versions
#define inp(port)                (port)
#define outp(val, port)          (port) = (val)

#define sbi(port, bit)           (port) |= (1 << (bit))
#define cbi(port, bit)           (port) &= ~(1 << (bit))

typedef  float tMember;

//*****************************************************************************
// conditional compilation
//#define_DEBUG_SONAR_TASK

//*****************************************************************************
// global returned results
#define  S_OK                    0
#define  S_WAIT                  1
#define  S_OVERFLOW              2
#define  S_BUSY                  3
#define  S_FAIL                  0xFF

//*****************************************************************************
#define  JOYSTICK_MEASUREMENT_TIME          10          // on every 10
milliseconds
```

```
#define  ADC_FRONT_BACK                    0          // ADC channels
#define  ADC_LEFT_RIGHT                    1
#define  SONAR_SENSORS                     8

//*************************************************************************
extern volatile UCHAR JoystickTimeOut;
extern void delay (UCHAR cycles);

#endif    //_COMMON_H_
//*************************************************************************
// end of file
//*************************************************************************
```

# B.5 Fuzzy Algorithm

```
//*****************************************************************************
// fuzzy.h
//*****************************************************************************
#include "common.h"

//*****************************************************************************
void FuzzyProcess (void);

extern float distance[8];
extern tMember FuzzyOutputSpeed;
extern tMember FuzzyOutputDirection;
extern UCHAR JoystickXValue;
extern UCHAR JoystickYValue;

//*****************************************************************************
// end of file
//*****************************************************************************

//*****************************************************************************
// joystick.c
//*****************************************************************************
#include "common.h"
#include "sonar.h"

//*****************************************************************************
#define  MEASUREMENT_NUMBER              4               // to use average value from these
                                                         // measurement results
#define  INDEX_MASK                      (MEASUREMENT_NUMBER - 1)

//*****************************************************************************
// local
static UCHAR FrontBackDirection[MEASUREMENT_NUMBER];
static UCHAR LeftRightDirection[MEASUREMENT_NUMBER];
static UCHAR MeasurementCounter;

/*****************************************************************************
| NAME:                 Joystick_init
| ABSTRACT:             component initialization after reset
```

```
| PARAMETER:                    none
| RETURN:                       none
*****************************************************************************/
void Joystick_init (void)
{
        MeasurementCounter = 0;
        JoystickTimeOut = JOYSTICK_MEASUREMENT_TIME;

        ADMUX = (1<<ADLAR);                     // external Vref = Vcc, right adjustment

        // ADC input clock 14MHz/8, approximately 10 microseconds conversion time
        ADCSRA = (1<<ADEN) | (1<<ADPS1) | (1<<ADPS0);
        ADCSRB = 0;                             // ADC free running mode

        // set analogue inputs
        DIDR0 = (1<<ADC_FRONT_BACK) | (1<<ADC_LEFT_RIGHT);
}
/*****************************************************************************
| NAME:                         JoystickMeasurement
| ABSTRACT:                     do measurement of both directions
| PARAMETER:  none
| RETURN:                       none
*****************************************************************************/
void JoystickMeasurementTask (void)
{
        if (JoystickTimeOut != 0)
        {
                return;
        }

        JoystickTimeOut = JOYSTICK_MEASUREMENT_TIME;

        // doing front-back joystick direction measurement
        ADMUX = (1<<ADLAR) + ADC_FRONT_BACK;
        sbi (ADCSRA, ADSC);

        while ((ADCSRA & (1<<ADSC)) != 0)
        {       // wait conversion end
        }
        FrontBackDirection[MeasurementCounter & INDEX_MASK] = ADCH;

        // doing left-right joystick direction measurement
```

```
        ADMUX = (1<<ADLAR) + ADC_LEFT_RIGHT;
        sbi (ADCSRA, ADSC);

        while ((ADCSRA & (1<<ADSC)) != 0)
        {       // wait conversion end
        }
        LeftRightDirection [MeasurementCounter & INDEX_MASK] = ADCH;

        MeasurementCounter ++;
}

/*****************************************************************************
| NAME:                         JoystickGetData
| ABSTRACT:                     gives the average measurement value
| PARAMETER:                    direction
| RETURN:                       average measurement value
*****************************************************************************/
UCHAR JoystickGetData (UCHAR Direction)
{
        UINT summ = 0;
        UCHAR idx;

        for (idx = 0; idx < MEASUREMENT_NUMBER; idx ++)
        {
                if (Direction == ADC_FRONT_BACK)
                {
                        summ += (UINT)(FrontBackDirection[idx]);
                }
                else
                {
                        summ += (UINT)(LeftRightDirection[idx]);
                }
        }

        return (UCHAR)(summ / MEASUREMENT_NUMBER);
}

//*****************************************************************************
// end of file
//*****************************************************************************
```

```
//*****************************************************************************
// fuzzy.c
//*****************************************************************************
#include "common.h"
#include "sonar.h"
#include "joystick.h"

//*****************************************************************************
//typedeffloat tMember;

#define  RULES_NUMBER                24
#define INPUTS_NUMBER                 9        // 1 direction and 8 distances
#define  DISTANCE_NUM                 8        // 8 sonars
#define  OUTPUTS_NUMBER               2

// sonar sensors indexes
enum
{
        S_LEFT_FRONT,           // S0
        S_RIGHT_FRONT,          // S1
        S_LEFT_CORNER,          // S2
        S_RIGHT_CORNER,         // S3
        S_LEFT,                 // S4
        S_RIGHT,                // S5
        S_LEFT_BACK,            // S6
        S_RIGHT_BACK            // S7
};

// sonar distances input members
enum
{                               // index    value
        Near,                   // 0    0.00 - 0.49
        Middle,                 // 1    0.50 - 1.00
        Far,                    // 2    > 1.00
        NONE                    // not used for calculations
};

// direction input/output members
enum
{                           // index
        REAR_L,             // 0       0.0PI   -   0.5PI
```

```
        LEFT,                   // 1       0.5PI   -   1.0PI
        FRONT,                  // 2       1.0PI   -   1.5PI
        RIGHT,                  // 3       1.5PI   -   2.0PI
        REAR_R,                 // 4       > 2.0PI
        DIR_FREE                // not used for calculations
};

// speed output members
enum
{                               // index
        SLOW,                   // 0
        MEDIUM,                 // 1
        FAST                    // 2
};

#define  OUTPUT_SPEED_INDEX                  9
#define  OUTPUT_DIR_INDEX                   10

// The Rules
UCHAR Rules[RULES_NUMBER][INPUTS_NUMBER + OUTPUTS_NUMBER] =
{       //  DIR_IN,    S0     S1      S2      S3      S4      S5      S6      S7      SPEED   DIR_OUT
//              0       1       2       3       4       5       6       7       8       9       10
        {  FRONT,  Far,    Far,    NONE,   NONE,   NONE,   NONE,   NONE,   NONE,   FAST,   FRONT   },
// 01
        {  LEFT,   NONE,   NONE,   NONE,   NONE,   Far,    NONE,   NONE,   NONE,   FAST,   LEFT    },
 // 02
        {  RIGHT,  NONE,   NONE,   NONE,   NONE,   NONE,   Far,    NONE,   NONE,   FAST,   RIGHT   },
// 03
        {  LEFT,   NONE,   NONE,   NONE,   NONE,   Near,   NONE,   NONE,   NONE,   SLOW,   RIGHT   },
// 04
        {  RIGHT,  NONE,   NONE,   NONE,   NONE,   NONE,   Near,   NONE,   NONE,   SLOW,   LEFT    },
// 05
        {  FRONT,  NONE,   Near,   NONE,   NONE,   NONE,   NONE,   NONE,   NONE,   SLOW,   LEFT    },
// 06
        {  FRONT,  Near,   NONE,   NONE,   NONE,   NONE,   NONE,   NONE,   NONE,   SLOW,   RIGHT   },
// 07
        {  LEFT,   Middle, NONE,   Near,   NONE,   NONE,   NONE,   NONE,   NONE,   MEDIUM, RIGHT   },
// 08
        {  RIGHT,  NONE,   Middle, NONE,   Near,   NONE,   NONE,   NONE,   NONE,   MEDIUM, LEFT    },
// 09
        {  LEFT,   NONE,   NONE,   Near,   NONE,   NONE,   NONE,   NONE,   NONE,   SLOW,   RIGHT   },
// 10
```

```
        { RIGHT, NONE, NONE, NONE, Near, NONE, NONE, NONE, NONE, SLOW, LEFT  },
// 11
        { FRONT, Near, Near, Near, Near, NONE, NONE, NONE, NONE, SLOW, REAR_R },
// 12
        { FRONT, NONE, NONE, Near, NONE, NONE, NONE, NONE, NONE, SLOW, RIGHT },
// 13
        { FRONT, NONE, NONE, NONE, Near, NONE, NONE, NONE, NONE, SLOW, LEFT  },
// 14
       { REAR_L, NONE, NONE, NONE, NONE, NONE, NONE, Near, NONE, SLOW, RIGHT },
 // 15
        { REAR_R, NONE, NONE, NONE, NONE, NONE, NONE, NONE, Near, SLOW, LEFT  },
// 16
        { REAR_L, NONE, NONE, NONE, NONE, NONE, NONE, Far,  NONE, FAST, REAR_L },
// 17
        { REAR_R, NONE, NONE, NONE, NONE, NONE, NONE, NONE, Far,  FAST, REAR_R },
// 18
        { REAR_L, NONE, NONE, NONE, NONE, NONE, NONE, Near, Near, SLOW, RIGHT },
// 19
        { REAR_R, NONE, NONE, NONE, NONE, NONE, NONE, Near, Near, SLOW, LEFT  },
// 20

        { FRONT, NONE, NONE, NONE, Near, NONE, NONE, NONE, NONE, SLOW, LEFT  },
// 21
        { FRONT, NONE, NONE, Near, NONE, NONE, NONE, NONE, NONE, SLOW, RIGHT },
// 22
        { FRONT, NONE, Middle, NONE, Near, NONE, Middle, NONE, NONE, SLOW, REAR_L },
// 23
        { FRONT, Middle, NONE, Near, NONE, Middle, NONE, NONE, NONE, SLOW, REAR_R }
// 24
       // DIR_IN,     S0      S1      S2      S3        S4      S5      S6      S7      SPEED   DIR_OUT
};

//*****************************************************************************
static float ajm;

static tMember InputDistanceMembers[DISTANCE_NUM][3];
static tMember InputDirectionMembers[5];

static tMember OutputSpeedMembers[RULES_NUMBER][3];
static tMember OutputDirectionMembers[RULES_NUMBER][5];

static tMember FinalOutputSpeedMembers[3];
static tMember FinalOutputDirectionMembers[5];
```

```
static tMember AuxOutputSpeedMembers[3];
static tMember AuxOutputDirectionMembers[5];
// exported variables
float distance[8];

tMember FuzzyOutputSpeed;
tMember FuzzyOutputDirection;

UCHAR JoystickXValue;
UCHAR JoystickYValue;

//*****************************************************************************
static void CalcInputDirectionMembers(void);
static void CalcJoystickAngle (UCHAR x, UCHAR y);
static void CalcInputDistanceMembers (float dist, tMember* member);
static void ClearArray (UCHAR *adr, UINT bytes);
static void CalcInputFuzzyMembers (void);
static void CalcOutputFuzzyMembers (void);
static tMember LookForMinValue (tMember *arr, UCHAR number);

/*****************************************************************************
| NAME:                         CalcJoystickAngle
| ABSTRACT:
| PARAMETER:                    ADC joystick values: x, y - range 0 - 255
| RETURN:                       none
*****************************************************************************/
static void CalcJoystickAngle (UCHAR x, UCHAR y)
{
        //atan2 returns a value in the range –PI to PI radians, using the signs of
        //both parameters to determine the quadrant of the return value.
        //returns x/y
        double X = (double)(x - 128);
        double Y = (double)(y - 128);
        double res;

        if (Y == 0)
        {       // to avoid division by zero
                Y = 1;
        }

        res = atan2 (X, Y);
```

```
	if (res >= 0)
	{
		ajm = (float)res;
	}
	else
	{
		ajm = (float)(res + 2*3.1415);
	}
}

/*****************************************************************************
| NAME:				CalcInputDirectionMembers
| ABSTRACT:
| PARAMETER:			input direction 0...2*PI
| RETURN:			none
*****************************************************************************/
static void CalcInputDirectionMembers(void)
{//mue7[1:5] = calculateInput7(ajm);

	/*Calculation membership function 1*/
	if (ajm > 1.571)
	{
		InputDirectionMembers[0] = 0;
	}
	else
	{
		InputDirectionMembers[0] = (tMember)(1 - ajm/1.571);
	}

	/*Calculation membership function 2*/
	if (ajm > 3.142)
	{
		InputDirectionMembers[1] = 0;
	}
	else
	{
		if (ajm < 1.571)
		{
			InputDirectionMembers[1] = (tMember)(ajm/1.571);
		}
		else
		{
```

```
                InputDirectionMembers[1] = (tMember)(2 - ajm/1.571);
        }
}

/*Calculation membership function 3*/
if (ajm > 4.712)
{
        InputDirectionMembers[2] = 0;
}
else
{
        if (ajm < 1.571)
        {
                InputDirectionMembers[2] = 0;
        }
        else
        {
                if (ajm < 3.142)
                {
                        InputDirectionMembers[2] = (tMember)((1.571 + ajm)/1.571 - 2);
                }
                else
                {
                        InputDirectionMembers[2] = (tMember)(3 - ajm/1.571);
                }
        }
}

/*Calculation membership function 4*/
if (ajm > 6.283)
{
        InputDirectionMembers[3] = 0;
}
else
{
        if (ajm < 3.142)
        {
                InputDirectionMembers[3] = 0;
        }
        else
        {
                if (ajm < 4.712)
```

```
                {
                        InputDirectionMembers[3] = (tMember)((3.142 + ajm)/1.571-4);
                }
                else
                {
                        InputDirectionMembers[3] = (tMember)(4 - ajm/1.571);
                }
        }
    }

    /*Calculation membership function 5*/
    if (ajm > 6.283)
    {
        InputDirectionMembers[4] = 0;
    }
    else
    {
        if (ajm < 4.712)
        {
            InputDirectionMembers[4] = 0;
        }
        else
        {
            InputDirectionMembers[4] = (tMember)((4.712 + ajm)/1.571 - 6);
        }
    }
}

/*****************************************************************************
| NAME:                 CalcInputDistanceMembers
| ABSTRACT:
| PARAMETER:            distance, address
| RETURN:               none
*****************************************************************************/
static void CalcInputDistanceMembers (float dist, tMember* member)
{
    /*Calculation membership function 1*/
    if (dist > 0.5)
    {
        *member = 0;
    }
    else
```

```
	{
		*member = 1 - (2*dist);
	}

	/*Calculation membership function 2*/
	if (dist > 1)
	{
		*(member + 1) = 0;
	}
	else
	{
		if (dist < 0.5)
		{
			*(member + 1) = 2 * dist;
		}
		else
		{
			*(member + 1) = 2 - (2 * dist);
		}
	}

	/*Calculation membership function 3*/
	if (dist > 1)
	{
		*(member + 2) = 1;
	}
	else
	{
		if (dist < 0.5)
		{
			*(member + 2) = 0;
		}
		else
		{
			*(member + 2) = -1 + (2 * dist);
		}
	}
}

/*************************************************************************
| NAME:                         ClearArray(...)
| ABSTRACT:                     clears all data before rules calculations
```

```
| PARAMETER:                    UCHAR *adr, UINT bytes
| RETURN:                       none
*****************************************************************************/
static void ClearArray (UCHAR *adr, UINT bytes)
{
        UINT idx;

        for (idx = 0; idx < bytes; idx ++)
        {
                *(adr + idx) = 0;
        }
}

/*****************************************************************************
| NAME:                         LookForMinValue
| ABSTRACT:
| PARAMETER:                    tMember *arr, UCHAR number
| RETURN:                       minimal value
*****************************************************************************/
static tMember LookForMinValue (tMember *arr, UCHAR number)
{
        UCHAR idx;
        UCHAR ZeroFlag = 0;
        tMember minVal = 1.0;

        for (idx = 0; idx < number; idx ++)
        {
                if (*(arr + idx) != 0)
                {
                        ZeroFlag = 1;
                }
        }

        if (ZeroFlag == 0)
        {   // all values are zeros
                return 0;
        }

        for (idx = 0; idx < number; idx ++)
        {
                if ((*(arr + idx) != 0) && (*(arr + idx) < minVal))
                {
```

```
                        minVal = *(arr + idx);
                }
        }

        return minVal;
}

/*****************************************************************************
| NAME:                         CalcInputFuzzyMembers
| ABSTRACT:
| PARAMETER:                    none
| RETURN:                       none
*****************************************************************************/
static void CalcInputFuzzyMembers (void)
{
        UCHAR idx;

        CalcJoystickAngle (JoystickXValue, JoystickYValue);
        CalcInputDirectionMembers ();


        for (idx = 0; idx < DISTANCE_NUM; idx ++)
        {
                CalcInputDistanceMembers (distance[idx], &InputDistanceMembers[idx][0]);
        }
}

/*****************************************************************************
| NAME:                         CalcOutputFuzzyMembers
| ABSTRACT:
| PARAMETER:                    none
| RETURN:                       none
*****************************************************************************/
static void CalcOutputFuzzyMembers (void)
{
        UCHAR rules_idx;
        tMember used_members[INPUTS_NUMBER];
        tMember MinValue;
        UCHAR used_index;
        UCHAR current_index;

        ClearArray ((UCHAR*)&OutputSpeedMembers[0][0], sizeof (OutputSpeedMembers));
```

```
ClearArray ((UCHAR*)&OutputDirectionMembers[0][0], sizeof (OutputDirectionMembers));

for (rules_idx = 0; rules_idx < RULES_NUMBER; rules_idx ++)
{
        ClearArray ((UCHAR*)&used_members[0], sizeof (used_members));
        used_index = 0;

        // Input direction member index
        current_index = Rules[rules_idx][0];
        // relevant input direction value
        used_members[used_index] = InputDirectionMembers[current_index];

        // collect data from input distance members
        for (used_index = 1; used_index < INPUTS_NUMBER; used_index ++)
        {
                // Input distance member index
                current_index = Rules[rules_idx][used_index];
                if (current_index < NONE)
                {
                used_members[used_index] = InputDistanceMembers[used_index - 1][current_index];
                }
        }

        MinValue = LookForMinValue (&used_members[0], INPUTS_NUMBER);

        // output speed member index
        current_index = Rules[rules_idx][OUTPUT_SPEED_INDEX];
        OutputSpeedMembers[rules_idx][current_index] = MinValue;

        // output direction member index
        current_index = Rules[rules_idx][OUTPUT_DIR_INDEX];
        OutputDirectionMembers[rules_idx][current_index] = MinValue;
}

ClearArray ((UCHAR*)&FinalOutputSpeedMembers[0], sizeof (FinalOutputSpeedMembers));
ClearArray ((UCHAR*)&FinalOutputDirectionMembers[0], sizeof (FinalOutputDirectionMembers));

// output speed
//mus2[1:3] = calculateOutputs2(mur1s2[1:3] to mur24s2[1:3])
for (used_index = 0; used_index < 3; used_index ++)
{
        MinValue = 1.0;
```

```
	for (rules_idx = 0; rules_idx < RULES_NUMBER; rules_idx ++)
	{
		if (OutputSpeedMembers[rules_idx][used_index] != 0)
		{
			if (OutputSpeedMembers[rules_idx][used_index] < MinValue)
			{
				MinValue = OutputSpeedMembers[rules_idx][used_index];
			}
		}
	}

	FinalOutputSpeedMembers[used_index] = MinValue;
}

// output direction
// mus1[1:5] = calculateOutputs1(mur1s1[1:5] to mur24s1[1:5])
for (used_index = 0; used_index < 5; used_index ++)
{
	MinValue = 1.0;
	for (rules_idx = 0; rules_idx < RULES_NUMBER; rules_idx ++)
	{
		if (OutputDirectionMembers[rules_idx][used_index] != 0)
		{
			if (OutputDirectionMembers[rules_idx][used_index] < MinValue)
			{
				MinValue = OutputDirectionMembers[rules_idx][used_index];
			}
		}
	}

	FinalOutputDirectionMembers[used_index] = MinValue;
}

for (used_index = 0; used_index < 3; used_index ++)
{
	AuxOutputSpeedMembers[used_index] = FinalOutputSpeedMembers[used_index] -
		(FinalOutputSpeedMembers[used_index] * FinalOutputSpeedMembers[used_index])/2;
}

//aux1 = 0.2*(mus2[1]-mus2[1]^2/2);
//aux2 = 1*(mus2[2]-mus2[2]^2/2);
//aux3 = 0.2*(mus2[3]-mus2[3]^2/2);
```

```
	AuxOutputSpeedMembers[0] = 0.2 * AuxOutputSpeedMembers[0];
	AuxOutputSpeedMembers[2] = 0.2 * AuxOutputSpeedMembers[0];

	for (used_index = 0; used_index < 5; used_index ++)
	{
		AuxOutputDirectionMembers[used_index] = FinalOutputDirectionMembers[used_index] -
			(FinalOutputDirectionMembers[used_index] *
FinalOutputDirectionMembers[used_index])/2;
	}

	//u2 = (0*aux1/2+0.5*aux2+1*aux3/2)/(aux1/2+aux2+aux3/2);
	FuzzyOutputSpeed = (/*0.0 * AuxOutputSpeedMembers[0]/2 +*/
		0.5 * AuxOutputSpeedMembers[1]/2 +
		1.0 * AuxOutputSpeedMembers[2]/2)/(
		AuxOutputSpeedMembers[0] +
		AuxOutputSpeedMembers[1] +
		AuxOutputSpeedMembers[2]);

	//u1 =
(0*aux1/2+3.1416/2*aux2+3.1416*aux3+3*3.1416/2*aux4+2*3.1416*aux5/2)/(aux1/2+aux2+aux3+aux4+aux5/2);
	FuzzyOutputDirection = (/*0.0 * AuxOutputDirectionMembers[0]/2 +*/
		1.571 * AuxOutputDirectionMembers[1] +
		3.141 * AuxOutputDirectionMembers[2] +
		1.5 * 3.141 * AuxOutputDirectionMembers[3] +
		3.141 * AuxOutputDirectionMembers[4])/(
		AuxOutputDirectionMembers[0]/2 +
		AuxOutputDirectionMembers[1] +
		AuxOutputDirectionMembers[2] +
		AuxOutputDirectionMembers[3] +
		AuxOutputDirectionMembers[4]/2);
}

/*************************************************************************
| NAME:				FuzzyProcess
| ABSTRACT:
| PARAMETER:			none
| RETURN:			none
*************************************************************************/
void FuzzyProcess (void)
{
	CalcInputFuzzyMembers ();
	CalcOutputFuzzyMembers ();
```

```
}
//*****************************************************************************
// end of file
//*****************************************************************************
```

# B.6 PWM

```
//*****************************************************************************
// pwm_output.h

//*****************************************************************************
#include "common.h"
#include "sonar.h"

void PWM_init (void);
void PWM_set_duty (UCHAR ch, UCHAR val);

//*****************************************************************************
// end of file
//*****************************************************************************

//*****************************************************************************
// pwm_output.c
//*****************************************************************************
#include "common.h"
#include "sonar.h"

/*****************************************************************************
| NAME:                          PWM_init
| ABSTRACT:
| PARAMETER:                     none
| RETURN:                        none
*****************************************************************************/
void PWM_init (void)
{
        TCCR0A = 0xA3;          // fast PWM mode
```

```
	OCR0A = 0x80;		// PWM duty 50 %
	OCR0B = 0x80;		// PWM duty 50 %
	TCCR0B = 0x02;		// 1/8 prescaller, ~ 5KHz output
	sbi (DDRB, 3);		// output A
	sbi (DDRB, 4);		// output B
}

/*****************************************************************************
| NAME:				PWM_set_duty
| ABSTRACT:
| PARAMETER:			channel 0 - 1, value 0 - 255
| RETURN:			none
*****************************************************************************/
void PWM_set_duty (UCHAR ch, UCHAR val)
{
	if (ch == 0)
	{
		OCR0A = val;
	}
	else
	{
		OCR0B = val;
	}
}
//*****************************************************************************
// end of file
//*****************************************************************************
```

# B.7 Main

```
//*****************************************************************************
// main.c test sonar functions

//*****************************************************************************
#include "common.h"
#include "sonar.h"
#include "avr_UART0.h"
#include "joystick.h"
#include "fuzzy.h"
#include "pwm_output.h"

//*****************************************************************************
// at 14745600 Hz - pulses per second (main quartz oscillator)
// => 14745 pulses per millisecond / 64 (prescaller) = 230
// at 10000000 Hz - pulses per second (main quartz oscillator)
// => 10000 pulses per millisecond / 64 (prescaller) = 156
// max timer value 255 - 230 = 25 - constant to be reload to have 1 millisecond
// the system interrupt for all timeouts

#ifdef CLOCK_10MHZ
        #define  TMR0_CONST             156
#else   //14745600 Hz
        #define  TMR0_CONST             230
#endif

#define  TMR0_RELOAD_VALUE      (0xFF - TMR0_CONST)
#define  UART_TIME              800

//*****************************************************************************
static void Interrupts_init (void);
static void Hardware_init (void);

//*****************************************************************************
volatile UCHAR JoystickTimeOut;

//*****************************************************************************
static volatile UINT MeasurementTimeOut;
```

```
static volatile UCHAR MainIntFlag;
static UCHAR str[128];

static UCHAR Output_0, Output_1;

#ifdef PORTC_OUTPUT_TEST
	static UCHAR PORTC_value = 0;
#endif

#ifdef PORTC_INPUT_TEST
	static UCHAR PORTC_value = 0;
	static sData[30];
#endif
//***************************************************************************
// interrupt routines
/***************************************************************************
| NAME:                    SIGNAL (SIG_OVERFLOW0 )
| ABSTRACT:                interrupt handler - provides all timeouts with 1 millisecond
|                          resolution
| PARAMETER:  none
| RETURN:                  none
***************************************************************************/
//SIGNAL (SIG_OVERFLOW0 )
SIGNAL (SIG_OVERFLOW2 )
{
	cbi (PORTB,0);
	//TCNT0 +=TMR0_RELOAD_VALUE;
	TCNT2 +=TMR0_RELOAD_VALUE;

	if (MeasurementTimeOut != 0)
	{
		MeasurementTimeOut --;
	}

	if (JoystickTimeOut != 0)
	{
		JoystickTimeOut --;
	}

	if (SonarMeasurementTimeOut != 0)
	{
		SonarMeasurementTimeOut --;
```

```
        }

        MainIntFlag = 1;
        sbi (PORTB, 0);
}

/*****************************************************************************
| NAME:                         void Interrupts_init (void)
| ABSTRACT:                     interrupts initialization and enabling
| PARAMETER:                    none
| RETURN:                       none
*****************************************************************************/
static void Interrupts_init (void)
{
        //sbi (TIMSK0, TOIE0);              // TIMER0 overflow interrupt enabled
        sbi (TIMSK2, TOIE2);                // TIMER2 overflow interrupt enabled
        sei ();                             // enable all interrupts
}

/*****************************************************************************
| NAME:                         void Hardware_init (void)
| ABSTRACT:                     interrupts initialization and enabling
| PARAMETER:                    none
| RETURN:                       none
*****************************************************************************/
static void Hardware_init (void)
{
        MeasurementTimeOut = UART_TIME;

        //TCNT0 = TMR0_RELOAD_VALUE;
        //TCCR0A = 0;                       // Timer0 normal operation
        //TCCR0B = 0x03;                    // prescaller = clk/64, start timer
        TCNT2 = TMR0_RELOAD_VALUE;
        TCCR2A = 0;                         // Timer2 normal operation
        TCCR2B = 0x04;                      // prescaller = clk/64, start timer

        Output_0 = 0;
        Output_1 = 0;

        MainIntFlag = 0;

        sbi (DDRB, 0);                      // PORTB0 as output
```

```
        sbi (DDRB, 1);                  // PORTB1 as output
}

/*****************************************************************************
| NAME:                         void SonarTask (void)
| ABSTRACT:                     start and check sensor on every SONAR_TIME milliseconds
| PARAMETER:                    none
| RETURN:                       none
*****************************************************************************/
static void UART0_Task (void)
{
UCHAR temp1, temp2;
UCHAR s[60];

        if (MeasurementTimeOut != 0)
        {
                return;
        }

        MeasurementTimeOut = UART_TIME;
/*
        str[0] = 0;             // zero broken
        sprintf (str,"FuzzyOutputSpeed:      %04d\n", (UINT)(FuzzyOutputSpeed*100));
        sprintf (s,"FuzzyOutputDirection:   %04d\n", (UINT)(FuzzyOutputDirection*100));
        strcat (str, s);
*/
        temp1 = JoystickGetData (ADC_LEFT_RIGHT);
        temp2 = JoystickGetData (ADC_FRONT_BACK);

        str[0] = 0;             // zero broken
        sprintf(str, "L-R:%03d, F-B:%03d", temp1, temp2);
        strcat (str, "\n");

        // print the calculation time:
        //sprintf (s, "Fuzzy time:%2d\n", FuzzyTimeCounter);
        //strcat (str, s);

        if (S_OK == IsSonarMeasurementDone ())
        {

                strcat (str, "Sonars:\n");
                SonarDataConversion ();
```

```
                for (temp1 = 0; temp1 < 8; temp1 ++)
                {
                        sprintf (s, "%1d: %04d cm\n", temp1, SonarDataInCentimeters[temp1]);
                        strcat (str, s);
                }

                SonarStartMeasurement ();
        }
        else
        {
                strcat (str, "Sonar measurement is not finished yet.\n");
        }


        UART0_send_data (str, strlen (str));

        Output_0 += 16;
        Output_1 += 64;

        PWM_set_duty (0, Output_0);
        PWM_set_duty (1, Output_1);

}

/*****************************************************************************
| NAME:                         int main (void)
| ABSTRACT:                     infinite main software loop
| PARAMETER:                    none
| RETURN:                       none
*****************************************************************************/
int main (void)
{
        Hardware_init ();
#ifdef PORTC_OUTPUT_TEST
        DDRC = 0xFF;    // all as outputs
        PORTC = PORTC_value;
        Interrupts_init ();

        while (1)
        {
                if (MainIntFlag != 0)
```

```
                {
                        MainIntFlag = 0;
                        PORTC_value ++;
                        PORTC = PORTC_value;
                }
        }
#endif  //PORTC_OUTPUT_TEST

#ifdef PORTC_INPUT_TEST
        UART0_init (SPEED0_115200);
        cbi (MCUCR, PUD);               // all PULLUP's are enabled
        DDRC = 0x00;                    // all as inputs
        PORTC = 0xFF;                   // all pull ups ON
        MeasurementTimeOut = 800;

        Interrupts_init ();

        while (1)
        {
                if (MeasurementTimeOut == 0)
                {
                        MeasurementTimeOut = 800;

                        PORTC_value = PINC;
                        sData[0] = 0;

                        sprintf (sData, "PORTC value: %02X\n", PORTC_value);
                        UART0_send_data (sData, strlen (sData));
                }
        }

#endif  //PORTC_INPUT_TEST

        InitSonar ();
        UART0_init (SPEED0_115200);
        Joystick_init ();
        PWM_init ();
        Interrupts_init ();
        SonarStartMeasurement ();
        PWM_set_duty (0, 196);
        PWM_set_duty (1, 64);
```

```
// input distances:
//JoystickXValue = 200;
//JoystickYValue = 20;

//distance[0] = 0.35;
//distance[1] = 0.65;
//distance[2] = 0.80;
//distance[3] = 0.98;
//distance[4] = 1.20;
//distance[5] = 1.40;
//distance[6] = 1.90;
//distance[7] = 2.10;

//FuzzyOutputSpeed:       0.22
//FuzzyOutputDirection:   2.83

//JoystickXValue = 200;
//JoystickYValue = 20;

//distance[0] = 0.65;
//distance[1] = 0.35;
//distance[2] = 0.80;
//distance[3] = 0.98;
//distance[4] = 1.20;
//distance[5] = 1.40;
//distance[6] = 1.90;
//distance[7] = 2.10;

//FuzzyOutputSpeed:       0.22
//FuzzyOutputDirection:   2.97

JoystickXValue = 200;
JoystickYValue = 20;

distance[0] = 0.95;
distance[1] = 0.85;
distance[2] = 0.80;
distance[3] = 0.98;
distance[4] = 0.20;
distance[5] = 0.40;
distance[6] = 1.90;
distance[7] = 2.10;
```

```
        //FuzzyOutputSpeed:       0.21
        //FuzzyOutputDirection:   2.70


        while (1)
        {
                //sbi (PORTB, 1);
                ////delay( 200);
                //_delay_us (50);
                //cbi (PORTB, 1);
                ////delay (100);
                //_delay_us (50);
                JoystickMeasurementTask ();
                SonarMeasurementTask ();
                FuzzyProcess ();
                UART0_Task ();
        }
}
//***************************************************************************
// end of file
//***************************************************************************
```