AUTONOMOUS ROBOTIC WHEELCHAIR

Authors

ALI AMMAR NAQVI	10-NUST-BE-ME-19
KOMAL ZULFIQAR	10-NUST-BE-ME-101
MUHAMMAD AQEEL ASHRAF	10-NUST-BE-ME-50

Supervisor

Dr.YASAR AYAZ

HoD, Robotics & Artificial Intelligence

DEPARTMENT OF MECHANICAL ENGINEERING

SCHOOL OF MECHANICAL& MANUFACTURING ENGINEERING

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY

ISLAMABAD

JUNE, 2014

Autonomous Robotic Wheelchair

Authors

ALI AMMAR NAQVI	10-NUST-BE-ME-19
KOMAL ZULFIQAR	10-NUST-BE-ME-101
MUHAMMAD AQEEL ASHRAF	10-NUST-BE-ME-50

A thesis submitted in partial fulfillment of the requirements for the degree of

BS Mechanical Engineering

Thesis Supervisor:

Dr. YASAR AYAZ

Thesis Supervisor's Signature:____

DEPARTMENT OF MECHANICAL ENGINEERING

SCHOOL OF MECHANICAL & MANUFACTURING ENGINEERING,

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY, ISLAMABAD

JUNE, 2014

Declaration

We certify that this research work titled "*Autonomous Robotic Wheelchair*" is our own work. The work has not been presented elsewhere for assessment. The material that has been used from other sources has been properly acknowledged / referred.

Signature of Students

ALI AMMAR NAQVI
10-NUST-BE-ME-19

KOMAL ZULFIQAR
10-NUST-BE-ME-101

MUHAMMAD AQEEL ASHRAF
10-NUST-BE-ME-50

Language Correctness Certificate

This thesis has been read by an English expert and is free of typing, syntax, semantic, grammatical and spelling mistakes. Thesis is also according to the format given by the university.

Signature of Students

ALI AMMAR NAQVI
10-NUST-BE-ME-19

KOMAL ZULFIQAR
10-NUST-BE-ME-101

MUHAMMAD AQEEL ASHRAF
10-NUST-BE-ME-50

Signature of Supervisor
Dr.YASAR AYAZ
HoD, Robotics & Artificial Intelligence

Acknowledgements

We are thankful to our Creator Allah Subhana-Watala for having guided us throughout this project at every step and for every new thought that He setup in our minds to improve it. Indeed we could have done nothing without His priceless help and guidance. Whosoever helped us through the course of our thesis, whether our parents or any other individuals was His will, so indeed none be worthy of praise but Him.

We are profusely thankful to our beloved parents who raised us when we were not capable of walking and continued to support us throughout in every department of our lives.

We would also like to express special thanks to our supervisor Dr. YASAR AYAZ for his help throughout our thesis and also for Intro to Mechatronics & Robotics course which he has taught us. We can safely say that we haven't learned any other engineering subject in such depth than the one which he has taught.

We would also like to pay special thanks to Lab Engineer Khawaja Fahad Iqbal for his tremendous support and cooperation. Each time we got stuck in something, he came up with the solution. Without his help we wouldn’t have been able to complete our thesis. We appreciate his patience and guidance throughout the whole thesis.

We would also like to thank Dr. Omer Gillani and Dr. Mohsin Jamil, for being on our thesis guidance and evaluation committee and express our special Thanks to Lab Engineer Ahmed Hussain Qureshi for his help. We are also thankful to Ali Bin Wahid, Usama Siraj and Muhammad Affan for their support and cooperation.

Finally, we would like to express our gratitude to all the individuals who have rendered valuable assistance to our study.

Dedicated to our exceptional parents and adored siblings whose tremendous support and cooperation led us to this wonderful accomplishment

Abstract

The Autonomous Robotic Wheelchair is being designed to enhance the capabilities of the joystick-controlled SAKURA Wheelchair made available through SAKURA's collaboration with RISE Lab, SMME. While an electric wheelchair can be successfully navigated by most persons, people with Parkinson's, Multiple Sclerosis or other severe mental difficulties remain unable to navigate a standard automated wheelchair in the same way. We have endeavored to design the Autonomous Robotic Wheelchair so that these people can rely on our smart wheelchair to do the navigation for them. Our Final Year Project is a small step towards this ambitious and important goal being pursued by a variety of student teams working under the umbrella of the Mobile Robotics Group at RISE Lab, SMME. Our project goals required that the wheelchair should be able to perceive the layout of its surroundings, localize itself within that layout, and given particular start and endpoints, it should be able to steer itself from one to the other following a reasonable collision-free path. To this end, we analyzed the hardware configurations of various other autonomous wheelchairs and adopted the hardware configurations which appealed to us the most. We modified the existing wheelchair hardware to incorporate an LRF sensor and a table-unit for an on-board laptop and electronics housing. Following that, we developed our obstacle avoidance and navigation codes based on C++ and Player\Stage. Our obstacle avoidance codes are making use of real-time laser sensing by the LRF and our navigation codes are being controlled either by the potential field navigation algorithm or by the initial position and headings provided by the Player\Stage 'cfg' files. Our obstacle avoidance algorithms are being monitored by Player\Stage running on the Ubuntu open-source platform which conveys LRF inputs to our C++ code and our C++ code, in turn, sends the forward and turning velocities to our windows-based LABVIEW program over a secure TP-Link Wireless Network. Once our Wheelchair becomes fully functional, probably after a series of coding modifications, integration of further types of sensors, rigorous lab-testing and optimization for environments bigger than the RISE Lab test space, we will then endeavor to prototype and market this wheelchair as a healthcare product meant to provide mobility and enhanced independence to the disabled and mentally challenged people we hope to assist.

Key Words: *Autonomous Wheelchair, Obstacle Avoidance, Navigation, Player\Stage*

Table of Contents

List of Figures

List of Tables

CHAPTER 1: INTRODUCTION

1.1 Background, Scope and Motivation

The Autonomous Robotic Wheelchair is one of the flagship research projects of the Modular Robotics Group at RISE Lab, SMME. We are working on the SAKURA automated wheelchair MC-2000 and incorporating sensors onto the wheelchair in order to enable intelligent motion.

For the purpose of our Final Year Project, we have worked with an LRF sensor and used it to enable the wheelchair to perceive the obstacles in its environment and to navigate its path around those obstacles. Other undergraduate and graduate groups working on the Autonomous Robotic Wheelchair are working with a large variety of sensors such as the webcams and headsets etc.

For us, the motivation for working on the Autonomous Wheelchair project was the opportunity to work on something that is at the cutting edge of robotics research and simultaneously a comprehensive engineering challenge in itself. Not only that the autonomous wheelchair is also a deeply humanitarian project as well for it seeks to empower and mobilize disabled people suffering from Parkinson's, Multiple Sclerosis and Tetraplegia etc.

The scope of our project is to enhance the capabilities of the SAKURA wheelchair in order to stimulate high maneuverability and navigational intelligence for the wheelchair. It is desired that the wheelchair should be able to perceive the layout of its surrounds, localize itself within that layout and given particular start and end points, the wheelchair should be able to steer a reasonable collision-free pathway.

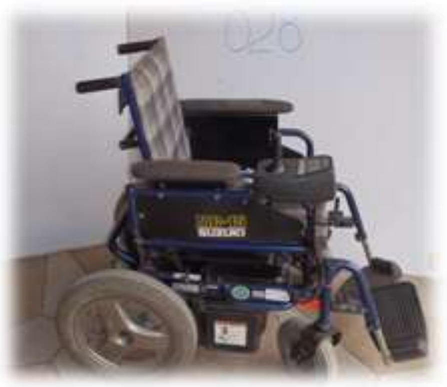


Fig 1.1: Sakura Electric Wheelchair MC 2000

1.2 Hardware

1.2.1 SAKURA WHEELCHAIR MC-2000

We have used the SAKURA Wheelchair MS-2000. The wheelchair is currently joystick controlled with two on-board batteries connected in series providing net 24V to the wheelchair. The front two wheels of the wheelchair are passive wheels and the back two wheels are powered with a differential drive. The joystick provides velocity values to the wheelchair in the form of forward or angular velocities. The wheelchair also has an efficient clutch system and a safety breaking mechanism which comes into action if the wheelchair is not started properly.

1.2.2 LRF SENSOR

We have used the SICK LMS-200 LIDAR Laser scanner and interfaced it with UBUNTU and Player\Stage. The cone angle for LMS-200 is 180˚ and it has a measurement range of 80m with 10mm resolution and +/-15mm systematic error. The LRF weighs approximately 4.5kg and is 210mm × 156mm × 155mm in size. In order to work, the LRF sensor requires 24V supply voltage and around 0.67Amps current. The LRF we have used was provided to us by the RISE Lab. We have tested and installed onto the wheelchair the LRF sensor which showed the most consistent LRF scan results.

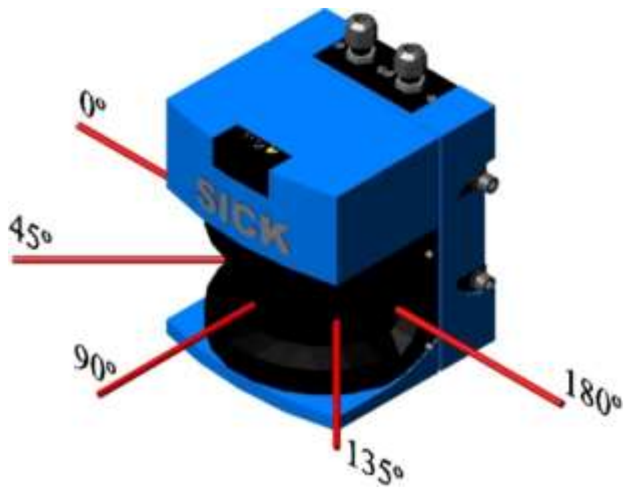


Fig 1.2: LRF Sensor

1.2.3 NI-DAQ

The NI USB-6009 provides basic DAQ functionality for applications such as simple data acquisition, portable measurements and academic experiments. The NI USB 6009 provides connections to eight single-ended analog input (AI) channels, two analog output (AO) channels, 12 digital input/output (IO) channels, and a 32 bit counter with a full speed USB interface. In order to use the NI USB-6009 we can use the DAQ Assistant using LabVIEW in order to configure the virtual and measurement channels which will drive our wheelchair.

Fig 1.3: NI 6009 USB

1.3 Software

1.3.1 UBUNTU

Ubuntu is an Open-Source Linux operating system. We have opted to use this for the Autonomous Robotic Wheelchair project since it is anticipated that the project will lead to ongoing high level research and development in the field of autonomous motion planning and navigation at the RISE Lab. The Version of Ubuntu we have worked on for the purpose of our project is: Ubuntu 12.04. It offered regular upgrades, software patches and downloadable applications which we availed at various times during our project.

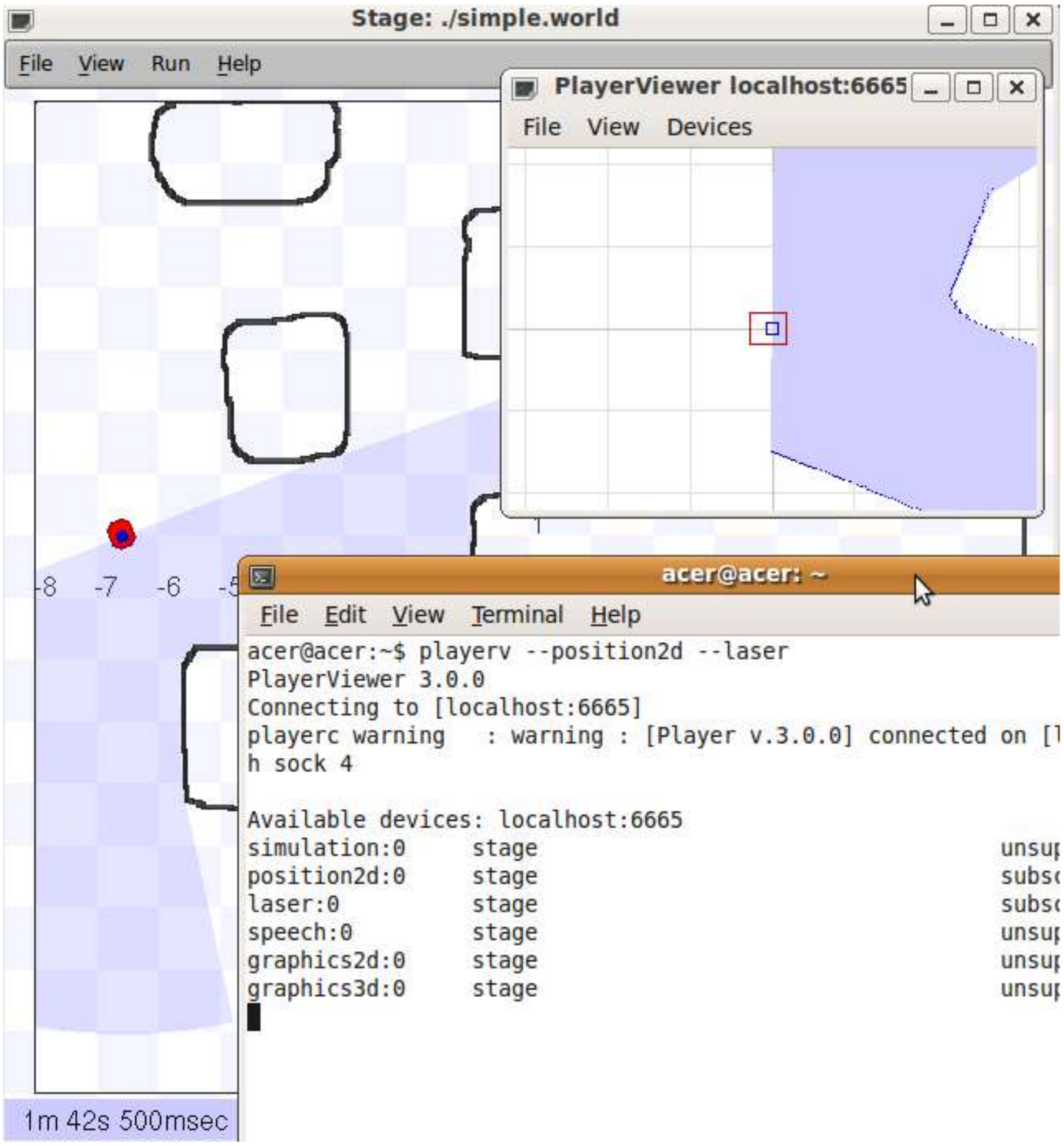
1.3.2 PLAYER\STAGE

The Player Project creates Open-Source software under the GNU Public License that enables advanced research in robotics. It is the most widely used robot control interface in the world.

The Player robot device interface provides a network server for robot control. Player has a client-server architecture which allows robot control programs to be written in any programming language and to run on any computer that has a network connection to the robot. Your Player clients can talk to Player over a TCP socket, reading data from sensors, writing commands to actuators, and configuring devices on the fly.

The Stage is Player's backend simulator. It simulates multiple robots moving around concurrently and sensing a two dimensional bitmap environment. The sensor models provided include sonar, LRF, webcam and odometery. But Player's modular architecture makes it easy to add support for new hardware and commonly the control programs written for Stage's simulated robots work just as well on real hardware as well.

Before implementing our algorithms on the wheelchair for trail-runs, we tested them in Player\Stage simulations on a P3AT robot model. Player has a variety of robot models available for simulations, a decent amount of online learning materials and programming examples, and an active user/developer community which continually contributes new drivers. The following is a look at the Player User Interface.



```
acer@acer:~$ playerv --position2d --laser
PlayerViewer 3.0.0
Connecting to [localhost:6665]
playerc warning   : warning : [Player v.3.0.0] connected on [l
h sock 4

Available devices: localhost:6665
simulation:0     stage                                      unsup
position2d:0     stage                                      subsc
laser:0          stage                                      subsc
speech:0         stage                                      unsup
graphics2d:0     stage                                      unsup
graphics3d:0     stage                                      unsup
```

Fig 1.4: Player\Stage User Interface

1.3.3 NETBEANS

NetBeans is an Open-Source integrated development environment for developing programs in Java, PHP, C/C++, and HTML5. The NetBeans platform allows applications to be developed from a set of modular software components called modules. NetBeans also offers cross-platform support for both Windows and Linux-based OS and has numerous plugins and APIs available.

1.3.4 LABVIEW

LabVIEW is a visual design and development environment from National Instruments. Its programming language ‘G’ is a dataflow programming language using which we can develop programs or subroutines called ‘virtual instruments’ or VIs. LabVIEW has three main components: the front panel, the block diagram, and the icon connector pane.

In LabVIEW, we can build the front panel user-interface using controls and indicators. Controls are knobs or dials etc. while indicators are graphs, LEDs or other displays. After we have built the user interface, we can insert our code into the block diagram using VIs and structures to control our front panel objects. Furthermore, LabVIEW communicates with hardware for data acquisition, vision and motion control using devices such as the NI 6009 USB which we have used, and LabVIEW has inbuilt features for connecting our programs to the web using the LabVIEW web server and software standards such as TCP/IP networking.

For our project, we have used LabVIEW in order to receive our velocity values from across the TP-Link wireless network. We feed in our laptop's IP address to enable a Windows-based computer and Ubuntu-based computer to network. We use NetBeans and TCP broadcasting to repeatedly update our forward and angular velocity values for the wheelchair. LabVIEW translates these into power values for the wheelchair's differential drive and these variable power signals are then sent to the wheelchair control using the NI DAQ USB. The following is a snapshot of LabVIEW's interface.

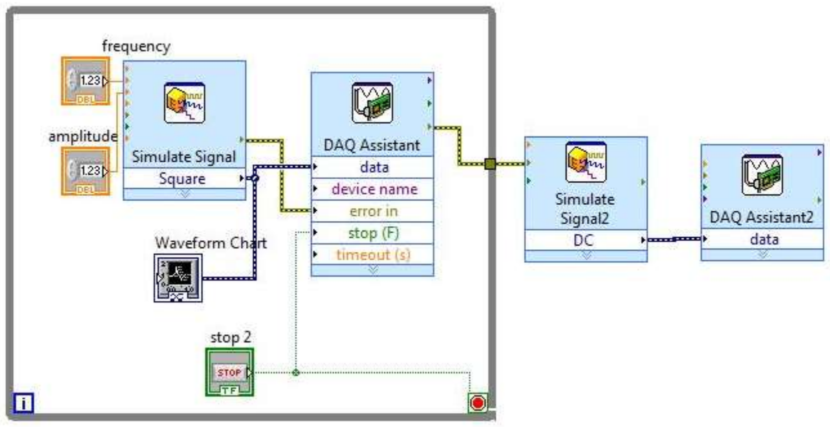


Fig 1.5: LabVIEW Graphical Interface

CHAPTER 2: DESIGN & FABRICATION

The wheelchair given to us was provided by SKURA (Japan) and it was joystick controlled. In order to make this wheelchair autonomous/intelligent, certain alteration in its design was needed. So our final year project work distribution includes a major task which we named *Design & Fabrication.*

2.1 Design

We were required to make a design which was totally detachable and could accommodate LRF (Laser Range Finder) on it. Furthermore we were also required to design a laptop table in front of the wheelchair user which could simultaneously serve as a mounting for sonar sensors or webcams as well. This table also was also meant to have a shelf where different electronic circuitry could be placed.

So we designed the following new features in our wheelchair:

i. LRF sensor mount;

ii. Laptop table (*including sonar mount & electronic housing*)

Our design has following three characteristics:

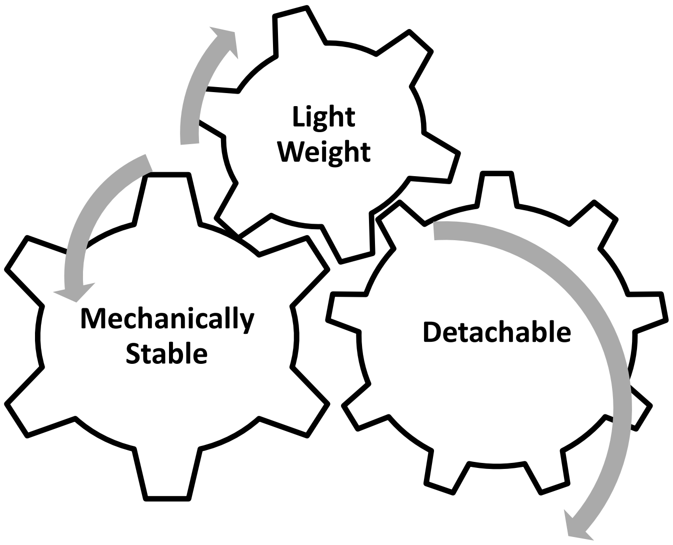
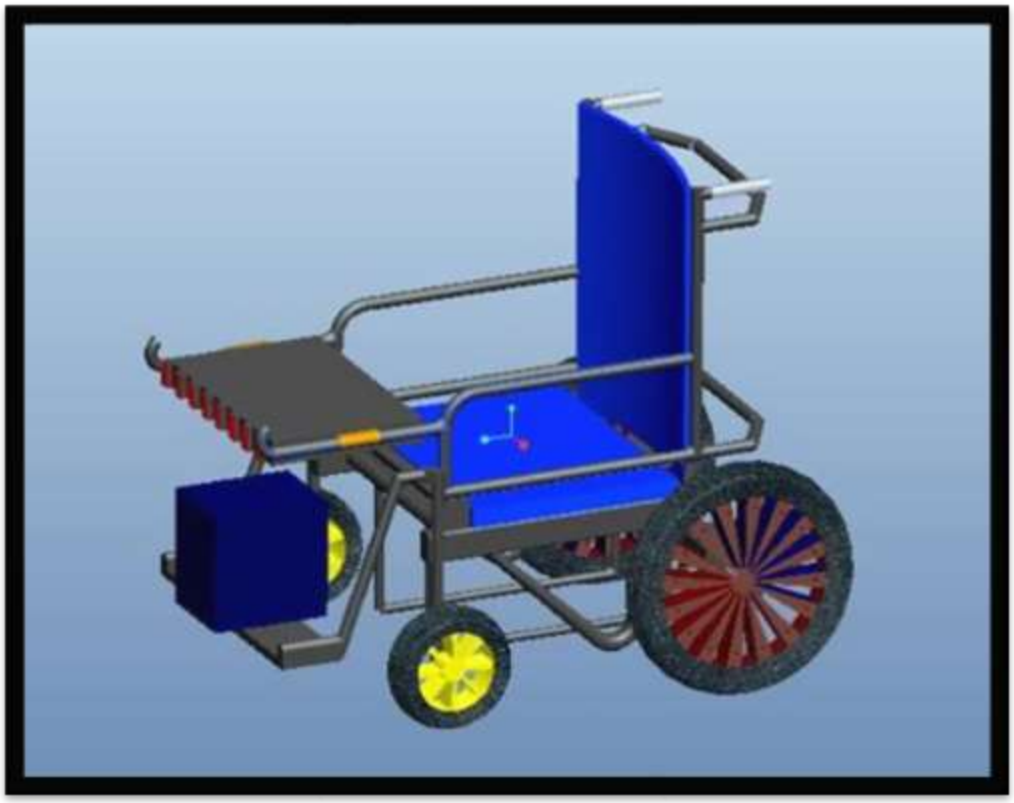


Fig 2.1: Wheelchair Mechanical Design Features

2.1.1 Motivation

We studied various designs of the autonomous wheelchairs that are being developed around the world in different universities like:



- The Intelligent Wheelchair Project @ MIT CSAIL
- The SmartWheeler Robotic Wheelchair Project @ McGill
- The Wheelchair Project @ Carnegie Mellon
- ‘Rolland’ Bremen Autonomous Wheelchair - Universität Bremen

Fig 2.2: PRO\E Model for the Wheelchair Hardware

2.1.2 Proposed Model

The above diagram is the assembly of the wheelchair consisting of different separately made components. We designed our proposed wheelchair model in Pro-E software. In the picture above, the dotted region shows the newly designed portion of the wheelchair. The above drawn model has exactly the same dimensions as of the original wheelchair. During modeling different complex joints were simplified. This modeling gave us a rough idea about how it will look after fabrication and this also helped us in reviewing our design again and again to make it more accurate. After this all our detailed diagrams, we used for the next stage i.e. *Fabrication.*

2.2 Fabrication

For fabrication purposes, we used the workspace of *Manufacturing Resource Center- SMME.* Three things were fabricated namely:

i. LRF Mount
ii. Laptop Table
iii. Extended arm rests (to support table)

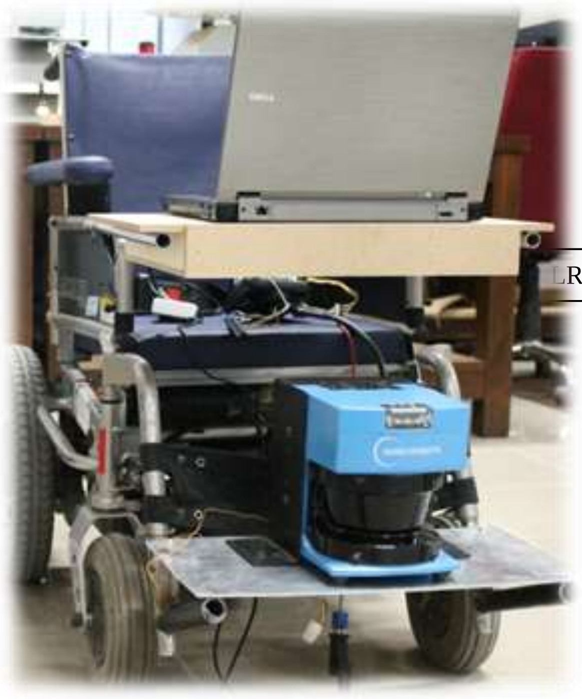


Fig 2.3: Final Hardware Representation of the Wheelchair

2.2.1 Materials & Processes

Following materials were used during this fabrication:

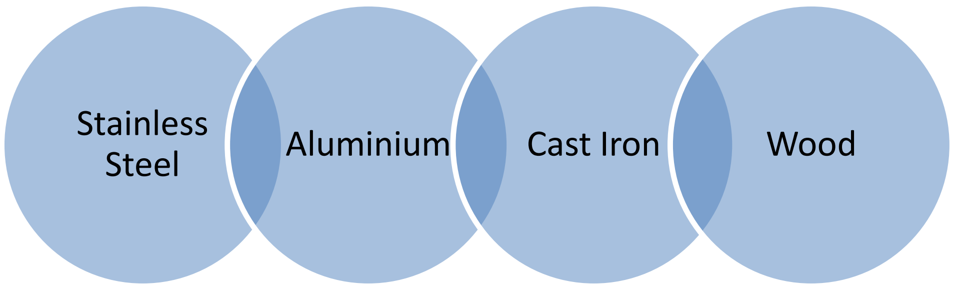


Fig 2.4: Wheelchair Fabrication Materials

Following processes were used in fabrication:

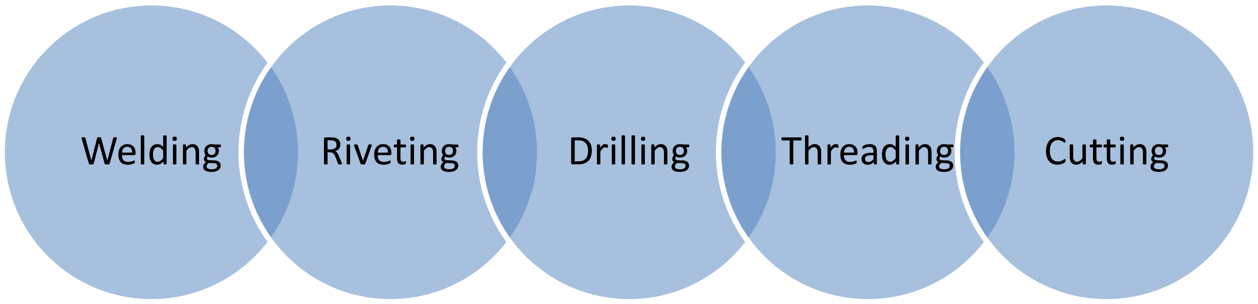


Fig 2.5: Wheelchair Fabrication Processes

2.2.2 LRF Mount

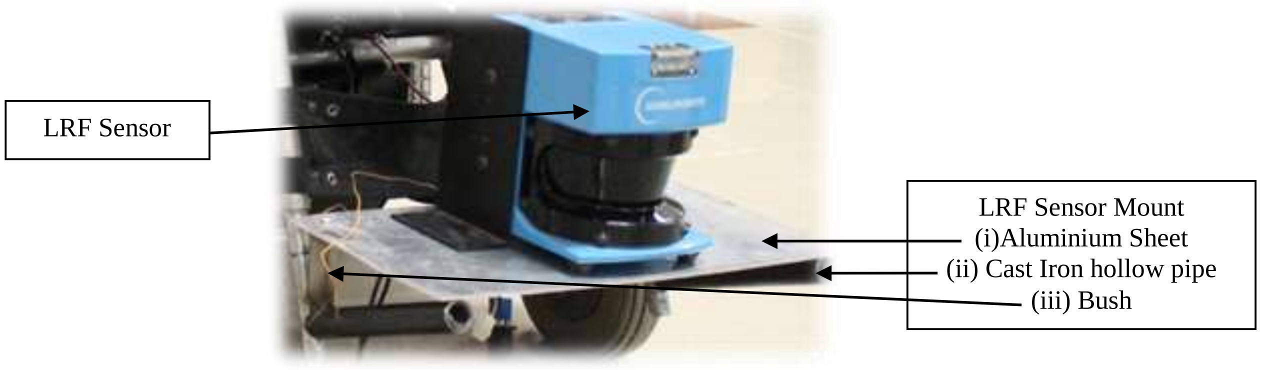


Fig 2.6: LRF Sensor Mount

2.2.3 Laptop Table

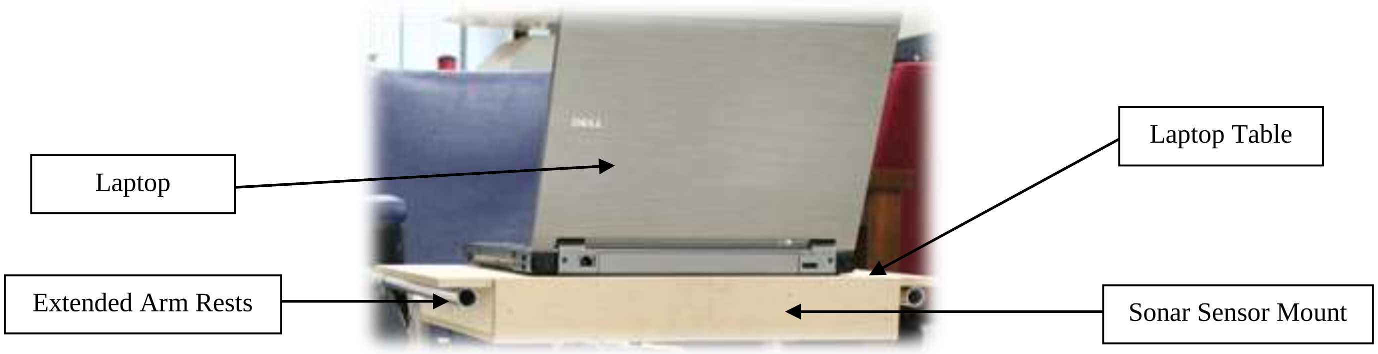


Fig 2.7: Wheelchair Laptop Table

CHAPTER 3: WHEELCHAIR CONTROL

3.1 Joystick Control

Initially the wheelchair provided by SAKURA (Japan) is joystick controlled. The person sitting on the wheelchair has a joy stick on his right side. The person has to move joy stick in the direction in which he/she is intended to go. For example if person pushes joy stick forward then wheelchair will move in the forward direction. Similarly if person pushes joy stick backward then wheelchair will move in the backward direction.

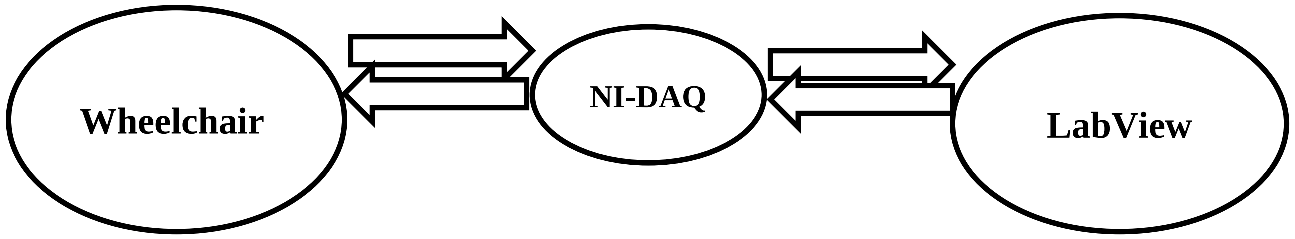
Joy stick can be move at any angle. The amount by which the joy stick is moved is proportional to the amount of voltage signal send to the internal circuitry and then that specified voltage signal is translated into a specific speed. This wheelchair is rare wheel driven and it has differential drive.

Fig 13: Joystick

3.2 Shifting Control : Joystick $\longrightarrow$ LabVIEW

In order to make wheelchair autonomous, we shiftedits control from joystick to LabView. For this purpose the wires were detached from the joystick and they were inserted into the NI-DAQ. NI-DAQ is the real time USB developed by National Instruments and it connects hardware with software. In this case hardware is our wheelchair and software is LabView.

Fig 14: Wheelchair Control Representation



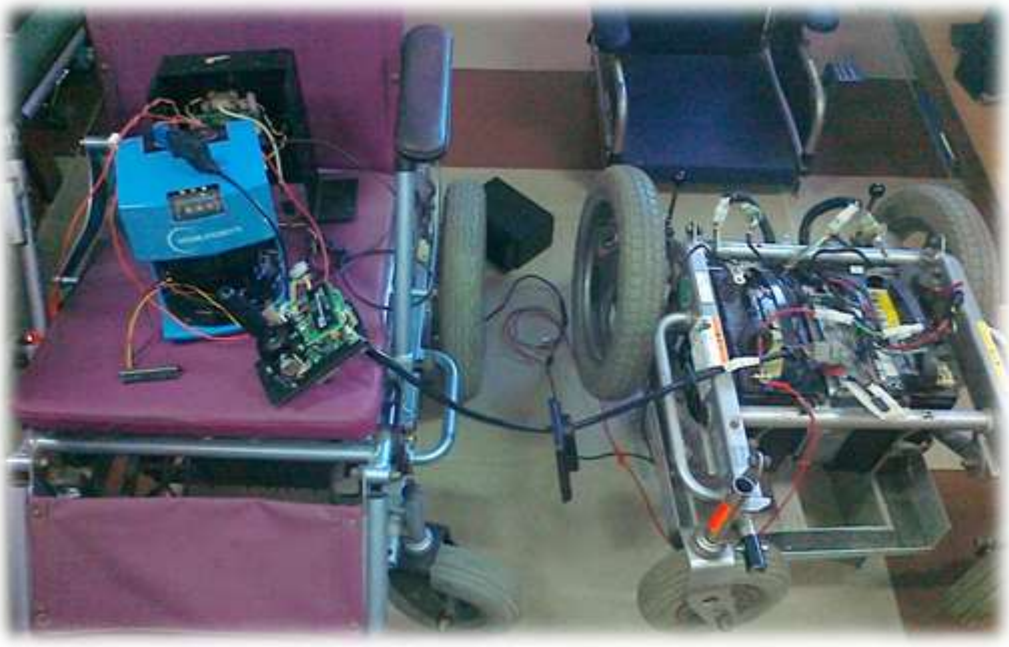


Fig 15: Shifting control form joystick to LabView

3.3 LabVIEW Control

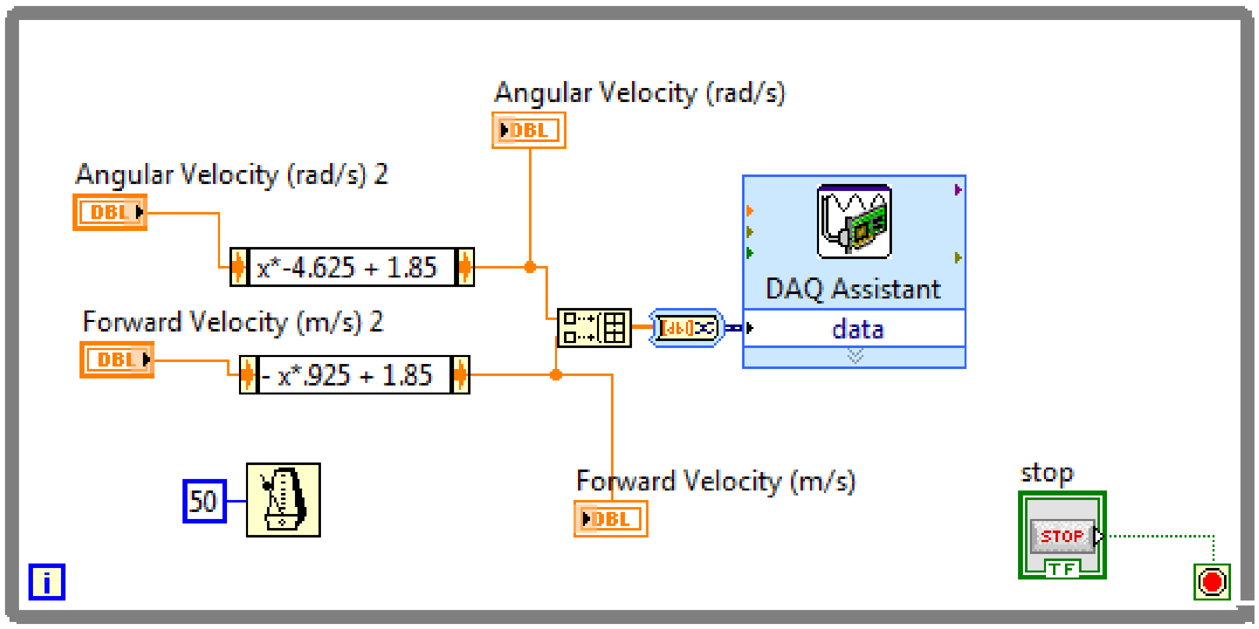


Fig 16: Graphical LabVIEW Control

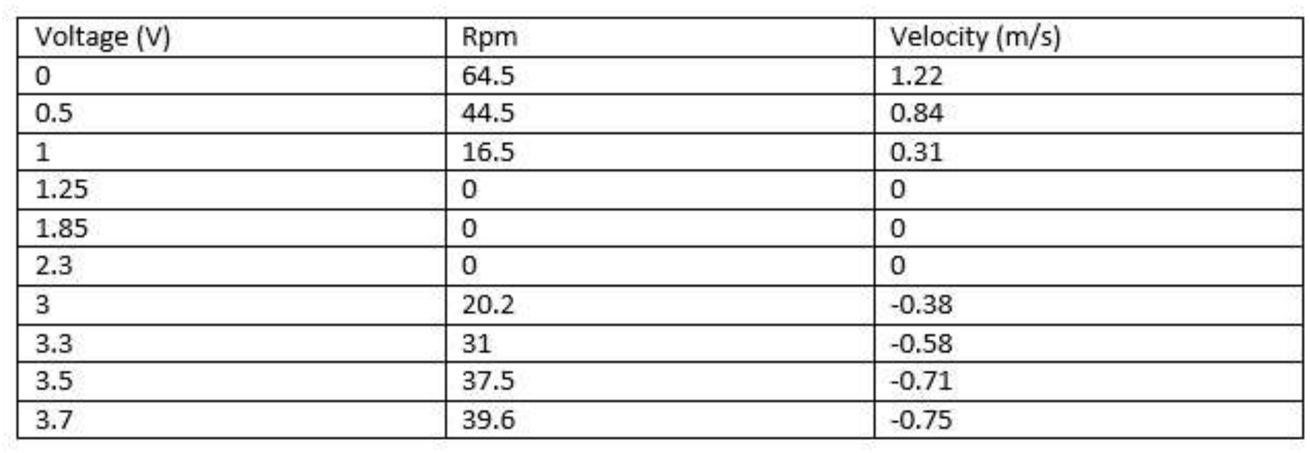
Description:

This LabView program was made to control wheelchair motion. Previously this was controlled by joystick. A continuous while loop is implemented so that a required value is being sent to wheelchair at all times. There is a stop button in the program by which code can be terminated at any time. Actually an angular velocity and a forward velocity values are given to the program then these values are converted to a specific voltage value. This conversion is done by two functions; one made for forward speed and another for angular speed. This function is made by using straight line equation i.e. y= mx + c on the velocity-voltage graphs *(graph is explained in the next section).* These voltage values for forward and angular speed the then send to wheelchair via NI-DAQ. Constant voltages of 1.85 volts are being provided to wheelchair when wheelchair is stationary.

3.4 Control Graph Analysis

Following table shows our acquired observations during control shifting:

Voltage vs. Rpm vs. Velocity



Voltage (V)	Rpm	Velocity (m/s)
0	64.5	1.22
0.5	44.5	0.84
1	16.5	0.31
1.25	0	0
1.85	0	0
2.3	0	0
3	20.2	-0.38
3.3	31	-0.58
3.5	37.5	-0.71
3.7	39.6	-0.75

Table 1: Voltage Vs. RPM Vs. Velocity for the Wheelchair

Voltages were provided by LabView software via NI-DAQ. RPM of the rare wheels were measured by the tachometer. Then velocity was calculated on the basis of rpm.

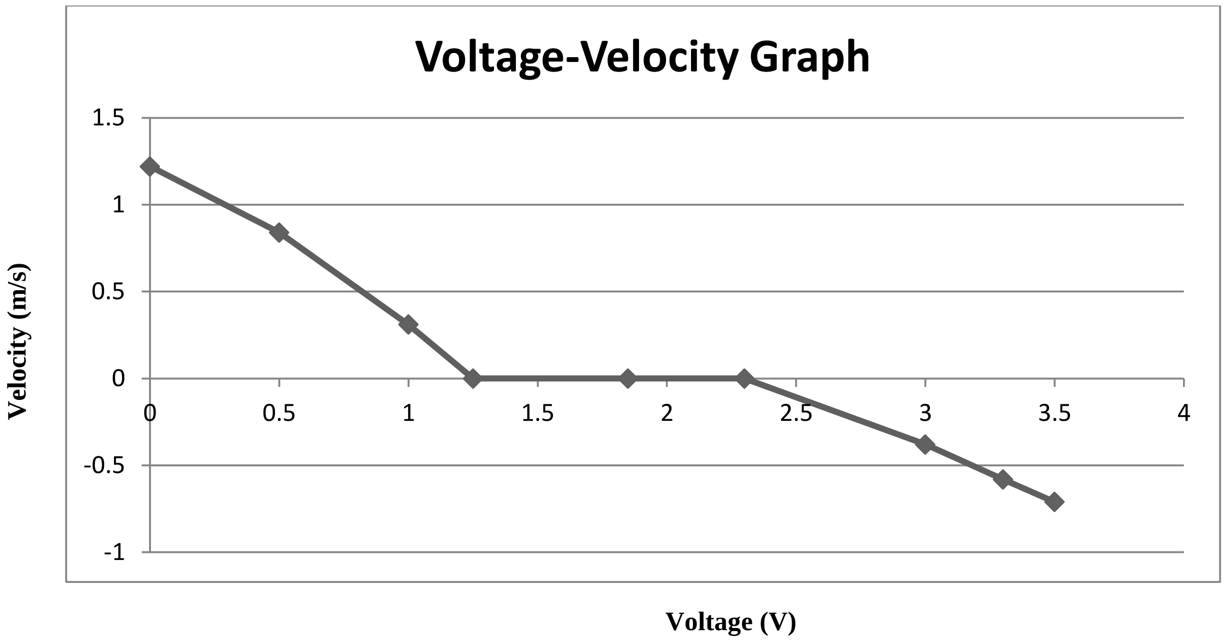


Fig 17: Voltage-Velocity Graph

The graph above shows the trend of voltage vs. velocity. We gave wheelchair different sets of voltages through LabView and then measured the resultant speed of wheelchair by measuring rpm of its rare wheels.

Following conclusions can be drawn from the above graph:

- Maximum forward speed of wheelchair is 1.22 m/s @ 0 volts
- Maximum backward speed of wheelchair is 0.75m/s @ 3.7 volts
- Wheelchair is stationary i.e. zero velocity @ 1.85 volts
- There is a voltage window where wheelchair does not move due to inertia and that is 1.25 volts – 2.3 volts
- Wheelchair maximum forward speed is greater than maximum backward speed
- Graph between voltage and velocity is discontinued and approximately linear

The main purpose of drawing this graph was to know exactly about the speed output when a specified voltage is provided. This knowledge helped during the developing of obstacle avoidance and navigation codes and it also helped us during real-time wheelchair testing phase.

CHAPTER 4: OBSTACLE AVOIDANCE

4.1 Literature Review

Obstacle Avoidance is a means of a robot being able to move around in an unknown environment without unknown obstacles coming in its path. It is one of the most important aspects of mobile robotics.

According to Seigwart et al, local obstacle avoidance focuses on changing the robot's trajectory as informed by its sensors during real-time robot motion. The resulting function is both a function of robot's current or recent sensor readings and its goal position and relative location to the goal position.

In developing our obstacle avoidance algorithms for the Autonomous Robotic Wheelchair, we were able to benefit significantly from the work completed in lieu of our MRDS semester project for our Intro to Mechatronics and Robotics class taught last semester by Dr. Yasar Ayaz and Dr. Umar Gillani.

We studied and practiced coding for primarily the obstacle avoidance algorithms presented in the Introduction to Autonomous Mobile Robotics by Seigwart et al. For example:

1) The Bug Algorithm:
The Bug Algorithm is the simplest most intuitive obstacle avoidance algorithm. The basic idea is to follow the contour of the obstacle in order to circumnavigate around it. It has two different forms Bug1 and Bug2.

In Bug1 algorithm, the robot fully circles around the obstacle and then moves along the line of shortest path towards the endpoint. This is the obstacle avoidance technique we have been able to implement in our code. With Bug2 algorithm, the robot starts to follow the object's contour but departs immediately when it able to move directly towards the object. This would have been more efficient and future coding on the Autonomous Wheelchair could attempt to develop this.

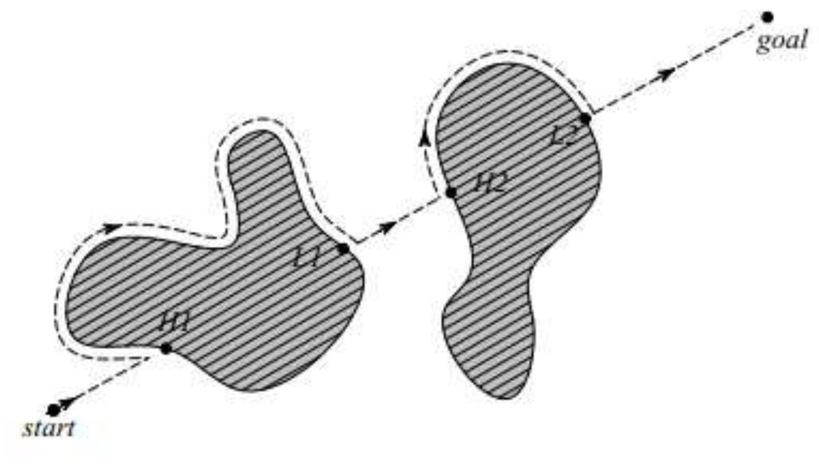


Figure 6.15
Bug2 algorithm with H1, H2, hit points, and L1, L2, leave points [199].

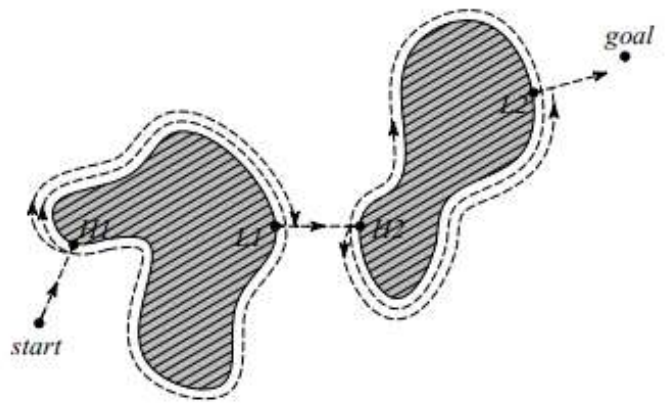


Figure 6.14
Bug1 algorithm with H1, H2, hit points, and L1, L2, leave points [199].

Fig 18: Bug Algorithms

2) The VFH Algorithm:

Developed by Bronstein & Koren, the VFH algorithm creates a local map of the environment around the robot which translates onto an occupancy grid populated by sensor readings. The VFH algorithm generates a polar histogram, in which the X-axis represents the angle alpha at which the obstacle was found, and the Y-axis represents the probability grid that there is an obstacle in that direction based on the occupancy grid's cell values.

We were able to obtain and study a basic implementation of the VFH algorithm in the example files for Player\Stage. However, we did not pursue it further due to its redundancy with the potential field path-planning algorithm which we were working on implementing at the time. Also the there are now more efficient VFH+ and the VFH* algorithms which have also been developed. There are a handful of other promising obstacle avoidance algorithms presented in Seigwart et al, however we were able to investigate and understand only the ones discussed.

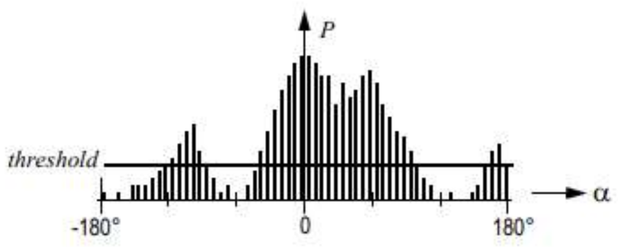


Figure 6.16
Polar histogram [177].

Fig 19: VFH polar histogram

4.2 Pseudo-Code

From Jennifer Owen's 'How to Use Player\Stage' manual we were able to obtain a basic obstacle avoidance function which we were able to expand upon in different ways. Our obstacle avoidance functions attempt to update motor speeds if there is an obstacle to be avoided. We update the function just before sending data to the motors. Thus, obstacle avoidance overwrites onto our navigation behavior for the robot.

Once the obstacle is no longer in its way, the robot moves along as it was. This allows us to transition from any navigation behavior into obstacle avoidance and then back again, as per the requirements of our control structure.

We wrote three distinct obstacle avoidance codes for the Autonomous Robotic Wheelchair. These are discussed as follows:

1) Left-Right Obstacle Avoidance:
This was the first obstacle avoidance code that we developed. It was based on the obstacle avoidance example code obtained from the Player\Stage tutorial files. It created and initialized two variables 'newspeed' and 'newturnrate' within a continuous for loop. Next, it used the inbuilt Player\Stage LRF commands 'lp.GetMinRight()' and

'lp.GetMinLeft()' in order to see to the left and right of the robot for obstacles. The readings obtained are processed and we check to see if either the left or the right distances to the obstacle are greater than 100. Based on that, we calculate robot's 'newspeed' and 'newturnrate' which is limited to values of -40° to +40°. And use the SetSpeed command to set these motor speeds.

Pseudocode for Left-Right Obstacle Avoidance:

```
        Enable Motors
        Initialize Newspeed & NewTurnRate
        Obtain minR & minL
Check to see if obstacle is close enough to avoid?
         If either distance is smaller; Turn the Other Way
Calculate NewSpeed using minR & minL
Calculate NewTurnRate using minR & minL
SetSpeeds
```

This code was implemented with the robot wandering in the 'Cave' cfg environment. It was not tested in real-time on the wheelchair because we were providing the wheelchair with the path beforehand from our navigation algorithm and we wanted primarily that the wheelchair should be able to detect and avoid the obstacles in its path rather than the obstacles coming to the left and right close to its path. A snapshot of the Left-Right Obstacle avoidance simulation has been provided in the Section 4.4.

2) GoTo-based Obstacle Avoidance:

This was the second Obstacle Avoidance code we developed. Our GoTo-based Obstacle Avoidance Code countered the problems of the Left-Right Obstacle Avoidance. However, we could not implement this one in the actual lab trials as well.

In GoTo-based Obstacle Avoidance, we declared our Data variables ‘robotSpeed.forwardSpeed’ and ‘robotSpeed.turningSpeed’, we initialized them to zero and set the motor speeds to their values. Then inside a continuous for loop, we gave forward speed an initial value of 0.25 and zero turning speed to get the robot moving along its provided path. The Robot.Read() command was used to update LRF values obtained from the LRF simulation proxy and we read in the LRF readings of minimum distances at angles 88° to 91° using the lp.GetRange() command. Doing so checks to see if there is an obstacle directly in front of the robot which if the robot was to continue moving along its given path it would collide with. If the obstacle is in front of the robot and the lp.GetRange() value of distance in front of the obstacle comes out as less than 1.5 we make the robot turn around an obstacle whose dimensions we assume to be known. So, if there is an obstacle closer than 1.5m, the robot would immediately stop. It would detect its X and Y coordinates at that point using the robot.GetXPos() and robot.GetYPos() commands. Thereafter, it would use those x and y position values to navigate around the obstacle using a sequence of GoTo commands that take the robot out of its path, enable it to cross the obstacle from its left and makes it come back directly in the position it would have been moved along to had there been no obstacle in its path. Lastly it checks to see if the endpoint has been reached, at which point it would stop immediately.

Pseudocode for GoTo-based Obstacle Avoidance:

```
Enable Motors
Initialize robotSpeed.ForwardSpeed to some initial value
Initialize robotSpeed.TurningSpeed to zero
Use Robot.Read()
```

```
        Use GetRange() to look in front of the robot
Check to see if obstacle in front close than set distance (1.5)?
        If Yes, read robot coordinates at current position
Use GoTo commands with current position coordinates to move
around obstacle
        Else Continue moving along original path
Check to see if EndPoint has been reached?
        If Yes, Stop.
```

3) SetSpeeds-Based Obstacle Avoidance:

This was the Obstacle Avoidance code which we eventually used for trial runs in the lab. Logically it follows the same sequence of instructions as out GoTo-based Obstacle Avoidance code except that it uses a set of SetSpeed commands in conjunction with various sleep() calls in order to navigate around the obstacle instead of the GoTo commands. It was important to develop this code because ultimately we needed that the forward speed and turning speed values for the robot should change against different time calls and we should be able to pass these changing forward and turning speed values to drive our wheelchair.

For this code, we again declare our data variables 'robotSpeed.forwardSpeed' and 'robotSpeed.turningSpeed', we initialized them to zero and set the motor speeds to their values. This is an important first step because not setting these speed values to zero initially causes the wheelchair's automatic safety system to activate causing the wheelchair to lock itself and give a beep sound.

Once we've passed zero values to the motors for some time, we give the wheelchair an initial forward speed of 0.8. This is the minimum

initial speed for which the wheelchair would move. For power values below this velocity level, the wheelchair does not move. Thereafter, inside a continuous for loop we use Robot.Read() command to update LRF values obtained from the LRF attached to the wheelchair. Details of how we can set up Player\Stage in order to read in these LRF values will be covered in the next section. As before, we read in LRF readings of minimum distances at angles 88˚ to 91˚ using the lp.GetRange() command. And we check to see if at some point all of these GetRange() values would be less than 1.3 or some other specified distance directly in front of our robot? If the obstacle is present, the robot halts and reads its X and Y coordinates using the robot.GetXPos() and robot.GetYPos() commands. Thereafter, it uses those x and y positions to navigate around the obstacle using a sequence of SetSpeed commands that take the robot out of its path, make it cross the obstacle and makes it come back to its path after having crossed the obstacle. Lastly, it checks to see if the endpoint has been reached at which point the robot stops.

Pseudocode for SetSpeeds-based Obstacle Avoidance:

```
Enable Motors
        Initialize ForwardSpeed & TurningSpeed to zero
        Initialize ForwardSpeed to 0.8
        Use Robot.Read()
        Use GetRange() to look in front of the robot
Check to see if obstacle in front close than set distance (1.3)?
         If Yes, read robot coordinates at current position
Use SetSpeed commands with current position coordinates to move
around obstacle
         Else Continue moving along original path
```

Check to see if EndPoint has been reached?

If Yes, Stop.

In this Obstacle Avoidance code, the wheelchair takes 90˚ turns around the obstacle. We have also written and tested a modified form of the SetSpeeds based obstacle avoidance code. In this version, instead of using 90˚ turns, the wheelchair moves around the obstacle at some angle. We are using different values for sleep() to give different values for the angle at which the wheelchair moves out of its path. The advantage of this variation is that the wheelchair has to cover less overall distance while it is avoiding the obstacle. But the problem with this technique is that the wheelchair is frequently unable to properly swerve around the obstacle. Depending on where the LRF will actually detect the obstacle, the angle required for obstacle avoidance will change from one trail run to the next. Our code does not have the capability for calculating a changing obstacle avoidance angle in this way every single time and this is one area which calls for further coding in order to perfect the performance of the Autonomous Robotic Wheelchair.

4.3 Integrating LRF and Player\Stage

In order to integrate the SICK LMS 200 with the Player\Stage software for real time obstacle avoidance, one has to first execute a series of terminal window instructions so that the LRF can be hooked up with our Player code. The commands are as follows:

```
        cd\dev                             // create a device folder
ls –l tty*                        // displays pointers for all the root
commands
sudo chmod +r ttyUSB0               // enables read at port zero
sudo chmod +w ttyUSB0             // enables write at port zero
ls –l ttyUSB0                     // displays the output at USB Port zero
ls –l ttyU*                        // shows the USB outputs
sudo chmod a+w ttyUSB0    //access and write for USB0
ls –l ttyU*
```

Along with following this sequence of instructions, we also modify our 'cfg' file so that instead of simulating an LRF sensor player recognizes the actual LRF SICK LMS 200 and tries to take the reads from it. We point the laser proxy to dev/ttyUSB0 in the cfg file and set baud rate for sampling and resolution parameters.

Thereafter, we open a second terminal window and use the following command:

```
playerv --laser --position2d
```

This command causes a stage window to open in which we can visually see the output of our LRF sensor. This stage window can be kept open and we can use a third terminal window to open a second Stage window in which we can run our robot while we're doing obstacle avoidance in real time.

4.4 Simulations

In this section, we will discuss the simulations we generated using our Lef-Right Obstacle Avoidance and GoTo-based Obstacle Avoidance codes.

In Left-Right Obstacle Avoidance, the robot wanders around in the cave environment and turns right or left based on the position of obstacles. The following is a snapshot of the Left-Right Obstacle Avoidance code.

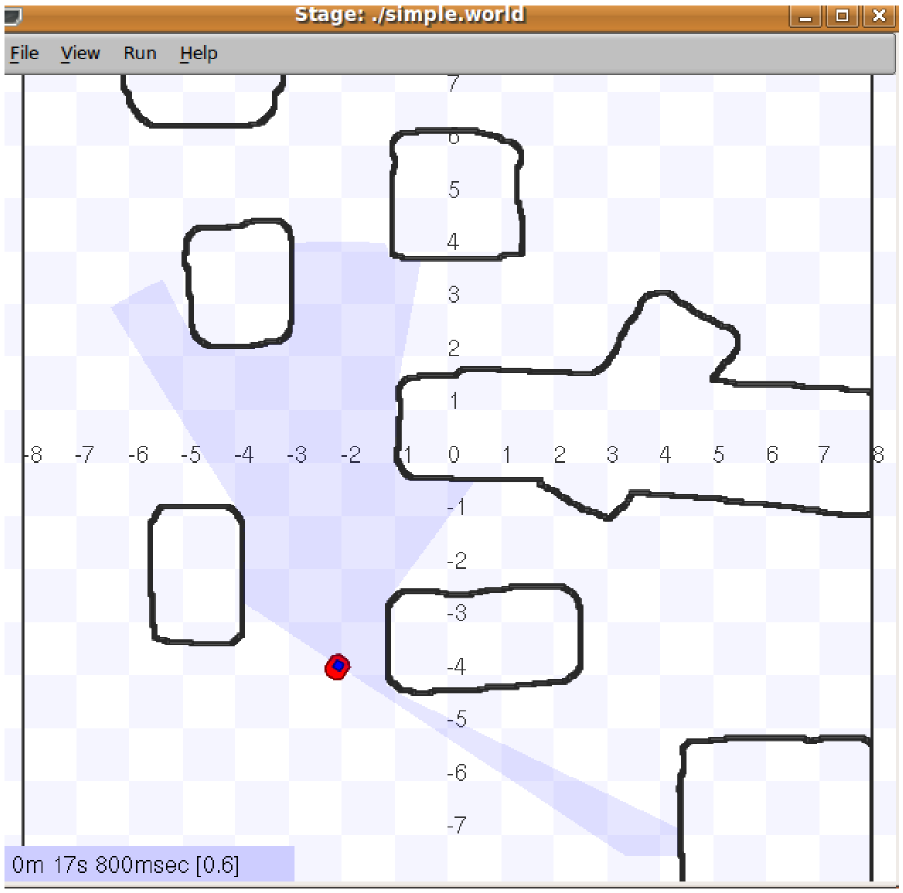
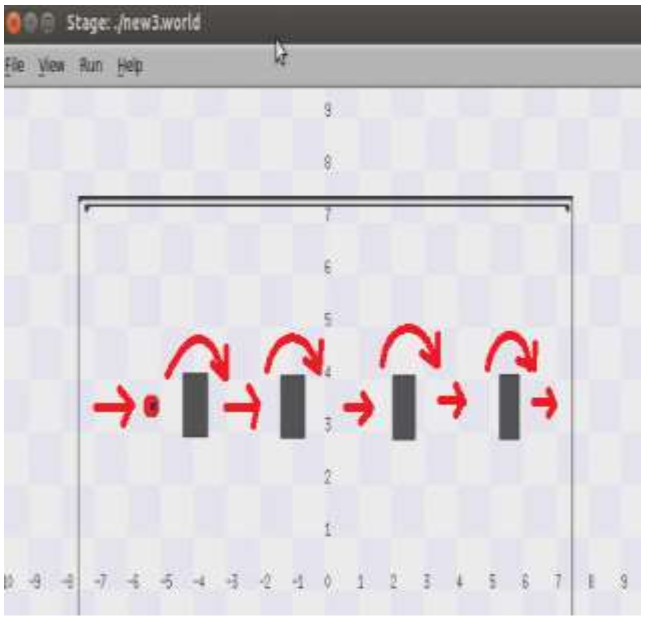


Fig 20: Left-Right Obstacle Avoidance



In the GoTo-based Obstacle Avoidance, the robot moves along its path and if it finds obstacles it navigates around it and returns to keep moving along its original path until endpoint is reached. The following snapshots show our GoTo-based obstacle avoidance simulations for a straight axis and for an inclined axis.

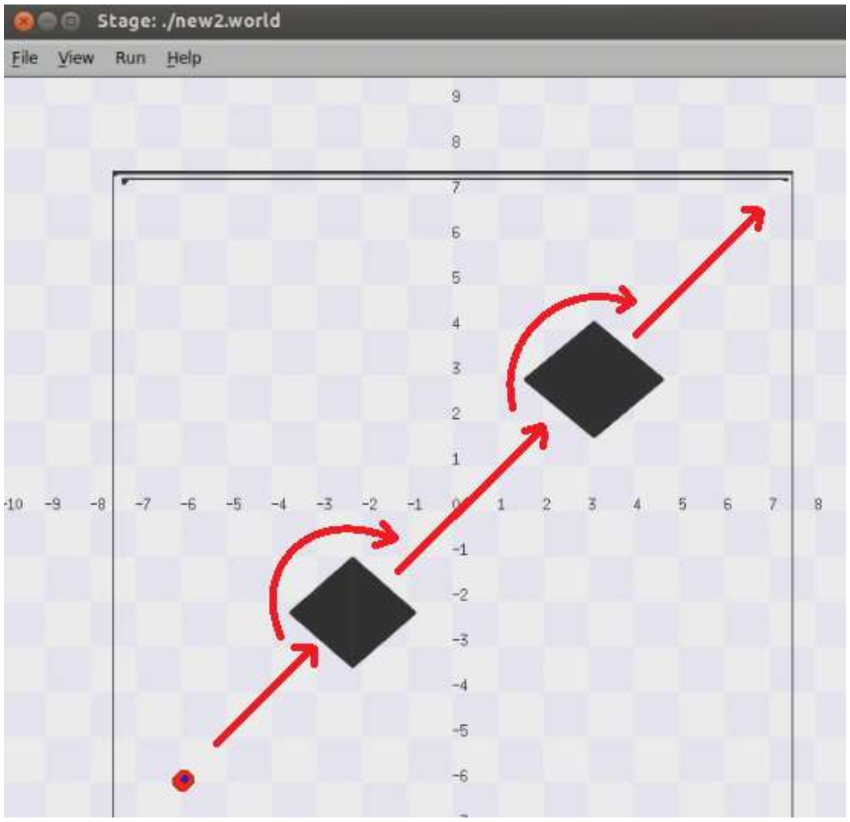


Fig 21: GoTo Based Obstacle Avoidance in X and Y Axis

CHAPTER 5: NAVIGATION

5.1 Literature Review

According to Introduction to Autonomous Mobile Robots by Seigwart et al, for a mobile robot, the specific aspect of cognition directly linked to robust mobility is navigation competence. Given partial knowledge about its environment and a goal position or a series of positions, navigation encompasses the ability of a robot to act based on its knowledge and sensor values so as to reach its goal positions as efficiently and as reliably as possible.

Therefore, navigation refers to the robot's ability to determine its own position in its frame of reference and then to plan a path towards its goal position. Thus, for navigation, the robot requires representation i.e. a map of the environment and the ability to interpret that representation. Navigation can be defined as a combination of three fundamental competencies: (i) Self-Localization (ii) Path-Planning (iii) Map-building and map interpretation.

Also there are two different types of Navigation:
(i) Global Path Planning: which deals with finding a suitable path from a starting point to a goal point using a given representation of the environment. We attempt to find the path with the lowest cost for the journey.
(ii) Local Path Planning: defines path points taking into account the vehicle dimensions and kinematic constraints. Thus, in local path planning we are looking for a planned path with small deviations which bring the robot back to its planned path.

In developing our navigation code we studied the various algorithms presented in the Introduction to Autonomous Mobile Robots by

Seigwart et al. The following is a brief discussion of the algorithms that we reviewed:

1) Breadth First Search:

The Breadth First Search algorithm begins at the start node and explores all of its neighboring nodes. For each of these nodes it explores all of its unexplored neighbors marking its nodes as either 'visited' or 'open'. In this way the algorithm iterates until it reaches it goal where it terminates. The breadth first search always returns the path with the fewest number of edges between the start and end nodes. If we assume that the cost of all individual edges in the graph is constant than the breadth first search returns the optimal-cost path.

2) Depth First Search:

In contrast to the Breadth First Search, the Depth First Search Navigation algorithm expands every single node to its deepest level, its branches are removed and the search backtracks by expanding the next neighboring node of the start node until its deepest level. Depth First Search offers the advantage of space complexity, in that it stores only the single path from the start node to the goal node.

5.2 Potential Fields Algorithm

For Potential Field navigation we start out by creating a two dimensional discretized configuration space of X and Y coordinates. We can then attach a potential value to each of these grid configurations and denote as a potential function or gradient spread over the entire 2D field within which the robot moves.

The potential field method treats the robot as a point under the influence of an artificial potential field U(q). The robot moves by following the field just as a ball were rolling downhill under the action of gravity. The basic idea is that the endpoint is pulling an attractive potential onto the robot and the obstacles within the robot's environment are exerting a repulsive potential onto the robot. Thereafter the potential field of the robot is computed as a sum of the attractive field of the goal and the repulsive fields of the obstacles.

$U(q) = U_{attr}(q) + U_{rep}(q)$

At every point in time, the robot will look at the potential field vector at that point and start going in that direction. The attractive and repulsive potentials can be found using any of the linear or quadratic formulae with various scaling factors for the endpoint attraction and the repulsion by obstacles. Some of the problems associated with the potential field algorithm are the Local Minima Problems in which robot is unable to navigate its way to the endpoint of the repulsive potentials due to the obstacles chocking its pathway.

5.3 BrushFire PseudoCode

In order to implement the Potential Field Navigation method for our code, we implemented the BrushFire Navigation scheme. The BrushFire algorithm is a clearance-based path finding algorithm which overlays a grid onto the entire navigational space of the robot.

In the BrushFire algorithm we've represented the 2D navigational space of our robot as a grid of pixels. For example, if we've used a 400 X 400 jpg image for our robot environment, we will first break it down into a grid of blocks. Each of the block is then checked using nested iterative for loop of pointers, if the pixels are occupied by obstacles we mark them with 1s and if they're free and the robot can travel to them we mark them as zero.

In order to compute the attractive potential of the endpoint we compute it as a function of the distance of each pixel from the goal. On the other hand, the repulsive potential of the obstacles is generated by traversing the map such that at each pixel visited the 8 neighbors connected to it are updated to a value one greater than the value of the current pixel. Doing so we have a grid where the occupied pixels are blocked out and the empty pixels close to the obstacles have higher grid values than those adjacent to them and the pixels as they go towards the endpoint have a decreasing grid value.

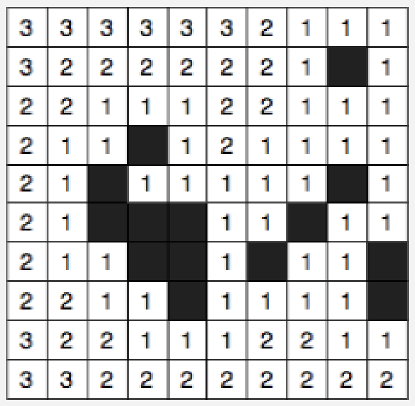


Figure 1. A small toy map annotated with values computed by the Brushfire algorithm. Black tiles are not traversable.

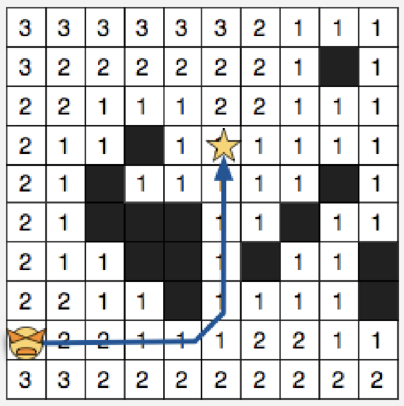


Figure 2. A 1x1 agent finds a path to the goal. All traversable tiles are included in the search space.

Fig 22: BrushFire Algorithm Grid

The architecture of our BrushFire Navigation program was made in the following way:

(i) CreateMap:

the CreateMap program takes a jpg or a png image as its input and it overlays a grid onto the image marking the occupied grid blocks as (1) and the unoccupied free space grid blocks as (0)

(ii) BrushFireMap:

the BrushFireMap program generates the BrushFire grid. It uses nested for loop to access every grid block and for each occupied grid block marked with (1) provided by the CreateMap grid it updates the cells around those obstacles to show repulsion. Then it goes around to the cells around those updated blocks and keeps updating values on the grid to show to the rippling effect of the repulsion produced by the obstacles in the navigation space.

(iii) DistanceMap:

the DistanceMap program is used to generate grid values to denote the attractive potential of the endpoint for the robot. Once again we make use of nested for loops to iterate to all the grid blocks and the blocks closer to the endpoint are marked with lower and lower values with the endpoint having the value of zero and the starting point having the maximum value on the grid

(iv) PotentialMap:

the PotentialMap program relies on the outputs of the BrushFireMap denoting the repulsion potential and the DistanceMap denoting the attractive potential. The PotentialMap program uses the grid output from both to compute a grid showing the net potential function at each point. The potential function is computed as follows:

Potential Function = P * DistanceMap[x][y] + N*pow(1/BrushFireMap[x][y], 3)

The values of P and N are varied by looking at the results of different simulation trials. Usually P is around 20 and N is on the order of 10^5

(v) RunBrushFireMap

the RunBrushFireMap program is the ultimate client program of CreateMap, BrushFireMap, DistanceMap, and PotentialMap. Based on the output of the PotentialMap program, the runBrushFireMap causes the robot to move from its start point to its endpoint. For our simulation we have used a for loop to get the robot moving along its PotentialMap path of shortest cost from the start to end. At each grid point, it checks to see if endpoint has been reached, if not it keeps moving along its PotentialMap shortest cost path.

5.4 BrushFire Simulations

We have tested the BrushFire in Player\Stage simulations only. We were able to obtain successful test trails in some of the BrushFire simularions but not all. Some of the problems arose to the programming complexity for the algorithm and due to the local minima problems of the BrushFire algorithm.

The following are screen shorts of our BrushFire Simulations which enables the robot to move from start point (provided by the cfg file) to endpoint (specified in the code). We started out by using simpler jpg images for testing as shown:

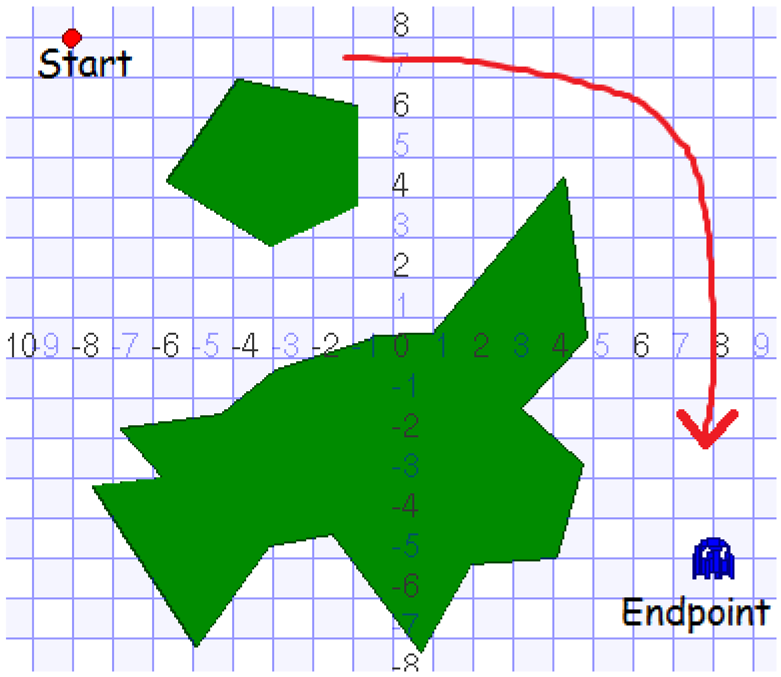


Fig 23: BrushFire Navigation

Eventually we progressed to more complicated scale model representations of our RISE Lab workspace and more realistic model-type obstacles placed within that workspace. In almost all of these simulations the robot was able to determine the shortest cost path to its end point specified. However, perhaps making the obstacles as models within the cfg file or for some other reason the BrushFire algorithm almost always crashed during more challenging test runs.

5.5 Run-Time Navigation

We were not able to use our BrushFire programs for real run-time navigation. There were not only logical problems in implementation of the algorithm but also the test constraints which became a significant hurdle in the way. For example, we need to provide forward and angular velocities in order to physically drive our wheelchair using the NI 6009 USB. However, the BrushFire code does not navigate based on velocity variables, it navigates based the relative grid values from start point to end point. Logically we could not translate the output of the latter into giving velocity values to the power drive of our wheelchair. Another significant problem was the small test space and the limited scope of obstacle avoidance and navigation we could afford within that test space as well as the front two wheels of the wheelchair being idle and causing significant deviations betweens the expected and actual wheelchair motion.

In view of these and other problems, we simplified our navigation scheme for the wheelchair based on a solution proposed by Lab Engineer Ahmed Hussain Qureshi. We used our Player\Stage 'cfg' file as our means for simple navigation. Using our cfg file, the robot was given an initial start position and an initial angular heading in order to move the robot along that heading. The robot was thus able to move along the line we wanted it move along, meanwhile our code was used to provide an end point along the length of the line the robot travels along. Once the robot has reached that specified point along the line, the robot motion terminates.
Since most of our testing on the wheelchair required for the chair to move along a straight line while avoiding obstacles in its path, we were able to successfully use this navigation technique in our simplified simulations and trail runs in the lab.

CHAPTER 6: NETWORKING

6.1 Introduction

networking between computers allows communication between different users and programs, thus enabling sharing of information between computers located at different locations. The information shared on networks is accessible to all the computers connected on the network. There are two main types of Computer networks,

(I) Wireless Network

(II) Wired Network

We used Wireless Networking for our project as it enabled us to communicate with the wheelchair from a distance without any wired connection, therefore enabling free motion of the Wheelchair without any hindrance.

6.1.1 Wireless Networking

Wireless networking is a flexible data communications system, which uses wireless media such as radio frequency technology to transmit and receive data over the air, minimizing the need for wired connections. Such networks are used to replace wired networks and are most commonly used to provide last few stages of connectivity between a mobile user and a wired network.

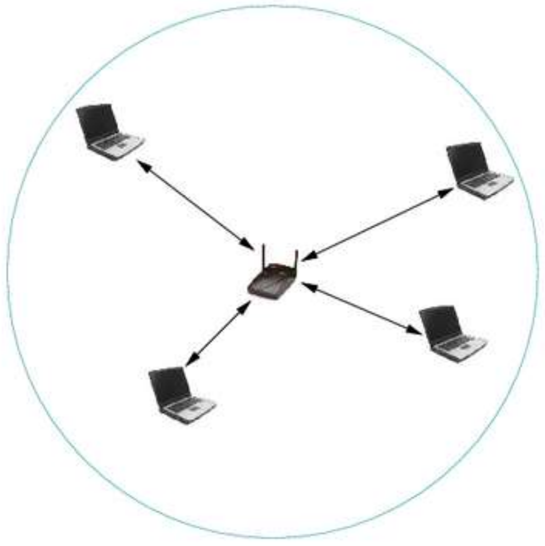


Fig. 24: Wireless Networking

Wireless networks use electromagnetic waves to communicate information from one point to another without relying on any physical connection. Radio waves are often referred to as radio carriers because they simply perform the function of delivering energy to a remote receiver. The data being transmitted is superimposed on the radio carrier so that it can be accurately extracted at the receiving end. Once data is superimposed (modulated) onto the radio carrier, the radio signal occupies more than a single frequency, since the frequency or bit rate of the modulating information adds to the carrier. Multiple radio carriers can exist in the same space at the same time without interfering with each other if the radio waves are transmitted on different radio frequencies. To extract data, a radio receiver tunes in one radio frequency while rejecting all other frequencies. The modulated signal thus received is then demodulated and the data is extracted from the signal.

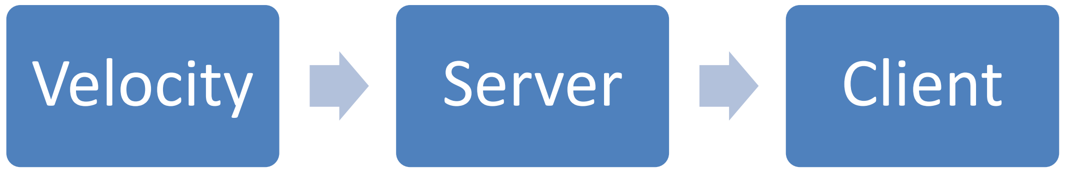
6.1.2 Advantages of Wireless Networking

Wireless networks offer the following advantages over traditional wired Networks:

- **Mobility:** They provide mobile users with access to real-time information so that they can roam around in the network without getting disconnected from the network. This mobility supports productivity and service opportunities not possible with wired networks.
- **Installation speed and simplicity:** Installing a wireless system can be fast and easy and can eliminate the need to pull cable through walls and ceilings.
- **Reach of the network:** The network can be extended to places which cannot be wired
- **More Flexibility:** Wireless networks offer more flexibility and adapt easily to changes in the configuration of the network.
- **Reduced cost of ownership:** While the initial investment required for wireless network hardware can be higher than the cost of wired network hardware, overall installation expenses and life-cycle costs can be significantly lower in dynamic environments.
- **Scalability:** Wireless systems can be configured in a variety of topologies to meet the needs of specific applications and installations. Configurations can be easily changed and range from peer-to-peer networks suitable for a small number of users to large infrastructure networks that enable roaming over a broad area.

6.2 Scope

In this project we used two computers operating in parallel. The first computer had Ubuntu as an Operating System with Player/Stage and NetBeans. The working algorithm gave the velocity output stored in variable form that was to be transmitted. We used wireless communication to transmit these velocity values from the navigation algorithm to the LabView operated computer. This LabView code further translated these values into voltage outputs, thereby enabling the accurate movement of the wheelchair



6.3 Literature Review

The Networking task required a lot of literature reading as the coding for creating a network and data transfer was a fairly new for us. This involved the study of standard networking tutorials and manuals. Following are some of the Tutorials that were used for Networking.

(I) Beej Networking tutorial

(II) Lantronix Networking tutorial

6.4 Server/Client

The Network communication is based on a server/client system. We used the server to transmit float type data to the client by casting it into a binary value. The Client interpreted this data type in binary and then re-calculated the values in float type data.

6.4.1 Sockets

The first step for creating a server was to create a parent socket. There are two types of sockets Stream sockets and Datagram sockets. We used Stream Sockets for our server because Stream sockets are reliable two-way connected communication streams. If we two output items into the socket in the order “1, 2”, they will arrive in the order “1, 2” at the opposite end and they will be error free.

6.4.2 IP addresses

The IP address of the parent socket connection to the server was known. It was in the form of IPv4. This IP address was used by the client program to connect with the server on the network and pick up the data.

6.4.3 Byte order

The computer stores data bytes in two different orders.

- **(I)** **Little-Endian:** When representing a two byte digit like b34f, this type of byte order will store the sequential bytes 4f followed by b3 in the memory. This storage method is called Little-Endian.
- **(II)** **Big-Endian:** When we want to represent a two byte like b34f, then we'll need to store it in two sequential bytes b3 followed by 4f. This number, stored with the big end first, and the small end last is called the Big-Endian order

Thus we understand that for Intel microprocessors that use little-Endian, The computer may have been storing bytes in reverse order. To counter this problem we cast the element in binary order and use the function network byte order that makes sure that the byte order remains the same in both computers. We make use of the long type variables that enable four byte storage.

6.5 Application

The whole networking process can be summed up in the following points:

- (I) Setting up a Server
- (II) Making a Responsive Client
- (III) Establishing a wireless Network for communication
- (IV) Sending and Receiving Data

CHAPTER 7: TESTING AND OPTIMIZATION

7.1 Testing Limitations

7.2 Obstacle Avoidance Testing

We performed the obstacle avoidance testing on the wheelchair within the RISE Lab test space. The wheelchair was set up by connecting the LRF to its power cords hooked up to the wheelchair's battery. The USB Port from the LRF was connected to the Port0 of Laptop 1 running Ubuntu and Player\Stage which was setup in order to enable it to read LRF readings in real time. Meanwhile Laptop 2 running LabVIEW was set up onto the wheelchair table and this laptop was connected wireless with the other one over the TP-Link. Laptop 2 was also hooked up to the joystick control of the differential drive of the wheelchair via the NI 6009 USB. We check the wheelchair emergency brake system once and run our code on Player\Stage and run the LabVIEW code once a connection has been established between the two computers. The follow are the descriptions of the main types of Obstacle Avoidance trails we performed:

7.2.1 Single Obstacle Avoidance

For our trail run for Single Obstacle Avoidance we used our SetSpeeds based Obstacle Avoidance algorithm. The algorithm made the wheelchair halt for a few seconds initially in order allow a network to setup and also so as not to activate the emergency brake mechanism of the wheelchair. After a while, the wheelchair is provided forward velocity of 0.8 which is the minimum velocity required for driving the wheelchair. Once the wheelchair detects obstacle at 1.3m or closer directly in front of it, the wheelchair notes its coordinates and uses them to move around the obstacle in front whose dimensions we have assumed to be known.

7.2.2 90˚ Avoidance

Our Single Obstacle Avoidance is of two types both based on SetSpeeds Obstacle avoidance. One is 90˚ avoidance which makes use of 90˚ angles in order to move around the

obstacle. After detecting the obstacle the wheelchair moves out of its path horizontally, it crosses the obstacles at a specified distance from it and comes back to its original path line horizontally after making yet another 90˚ turn. Problems arise during 90˚ Obstacle Avoidance when the front wheels behave differently and trace a different angle (not equivalent to 90˚) from one trial run to the other. A possible solution for this is to keep the testing conditions as similar as we can so that test results can be repeated based on our time calibrations. Video 1 provided shows Single Obstacle Avoidance with 90˚ turns.

7.2.3 Variable Angle Avoidance

Variable Angle Avoidance is essentially the same thing as 90˚ avoidance except that it makes the wheelchair avoid the obstacle in front at an angle. Consequently, the wheelchair travels less distance than it does in case of 90˚ avoidance. As before, we again use SetSpeeds based avoidance but this time, instead of using time calls for long enough to allow the wheelchair to make 90˚ angles we only allow it to make smaller 45-60˚ angles and to go out of the path of the obstacle and to come back to the original path in the same way as before.

There is a major problem with implement angular obstacle avoidance using our code, based on the point at from which the wheelchair detects the obstacle in front of it. Its turning angle constantly varies from one trial run to the next. This point keeps changing because of the LRF scan variations and the inaccuracies caused by the front wheels, thus one trail run may not lead to a second successful trial run and we need to rely on hit-and-trial to get this to work for any specific test condition. Variable Angle Avoidance is shown in Video 3.

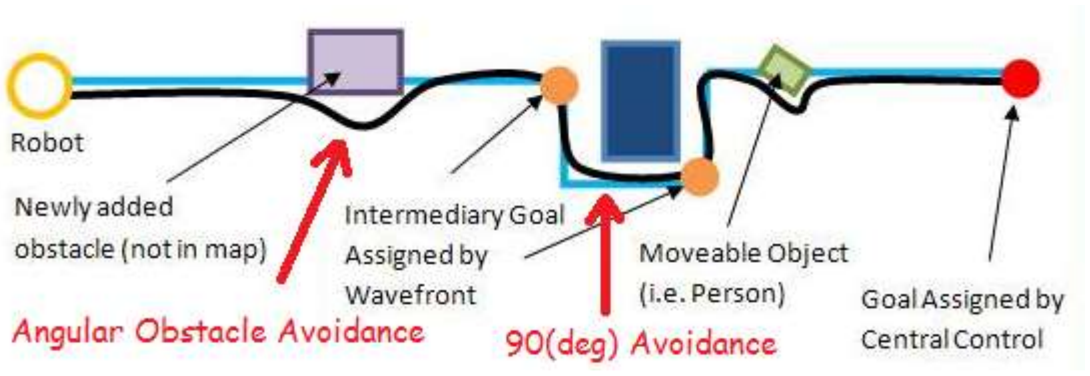

Robot
Newly added obstacle (not in map)
Angular Obstacle Avoidance
Intermediary Goal Assigned by Wavefront
90(deg) Avoidance
Moveable Object (i.e. Person)
Goal Assigned by Central Control

Fig 25: 90 degrees and Angular Obstacle Avoidance

7.3 Multiple Obstacle Avoidance

Our Multiple Obstacle Avoidance code makes use of the Single Obstacle Avoidance with 90˚ turns consecutively. Like before, wheelchair is prepped for motion by giving it zero velocity values. After a while, it is given a forward velocity of 0.8m to get it started. When the first obstacle is detected, the wheelchair halts and makes a 90˚ turns and moves horizontally to avoid the first obstacle. It continues on its new path this time and as it encounters a second obstacle in front of it. It again makes a 90˚ turn and avoids the obstacle by moving horizontally.

Another slight variation we could have programmed for this code could have been multiple obstacle avoidance such that the obstacle returned to its original path along which it started. It is easily possible by making slight modifications on our current code.

7.4 Dynamic Obstacle Avoidance

This is another small task we could accomplish using our current code however, we have not yet perfected it during trails runs. Simplistic Dynamic Obstacle Avoidance is also possible using our current Single Obstacle Avoidance Code. However, we have not had the time to implement it in lab as such. We need to use the same code as that we used before for Single Obstacle Avoidance. However, this time once the obstacle has been detected we need to give the wheelchair a velocity somewhat greater than 0.8 say 1.8 or something which would cause the wheelchair to move faster than the other dynamic object moving directly in front of it and to maintain that speed during and after obstacle avoidance. Further testing is encouraged in this direction since it would quickly yield results.

7.5 Navigation Testing

Our Run-Time based Navigation techniques making use of the cfg file to do simple navigation for the wheelchair using its start point and its angular heading at the start point worked perfectly in every trail run. The wheelchair was able to end up at exactly where we wanted it to go in almost every single trial run.

CHAPTER 8: CONCLUSION & RECOMMENDATIONS

8.1 Conclusions

The Autonomous Robotic Wheelchair is an important technical solution to the problems posed by the traditional powered wheelchairs. It can enable the end-user to use the wheelchair by himself without the need of any other person, irrespective of their disability or physical condition. For people with Parkinson's, Multiple Sclerosis or Tetraplegia, an autonomous wheelchair can be a means to an independent, fulfilling life.

In the course of our Final Year Project entitled 'Autonomous Robotic Wheelchair' we have been able to make decent progress towards our dream of developing such a wheelchair here in Pakistan. We could not have gotten anywhere had it not been for the SAKURA automated wheelchair provided to us at RISE Lab. The automated wheelchair had the electronic control and the hardware which was a necessary prerequisite for our project.

We were able to modify the existing hardware of the wheelchair to mount our LRF and the laptop table with electronics housing. Our hardware design is modular, simplistic and completely detachable. In the course of future work on the wheelchair hardware, subsequent sensor mounts can the designed either on top of our own mounts or in place of the holes and fasteners that we have already used.

As for the project tasks of Obstacle Avoidance and Navigation, we were able to make respectable progress on both the fronts. Not only could we successfully implement Obstacle Avoidance and Navigation in our simulations, we were able to implement both during our trails runs in the lab. However, only on a more restricted scale and approximating the behavior of our original algorithm commands by using sleep calls with different time values.

Finally, we were able to successfully upgrade the wheelchair from its joystick control to laptop control while making use of sensor readings all in real time. A robust LabVIEW program was used to drive the wheelchair by giving it power/ velocity values and Player\Stage was used to collect data from the LRF for successful obstacle avoidance in almost every single trial run.

8.1.1 Learning Outcomes

When we started out with the Autonomous Robotic Wheelchair at the start of the year we did not have a lot of background knowledge to go on. In fact, the only thing we had was the idea of the final shape of the Autonomous Robotic Wheelchair that we wanted to get to. And as we started out, it was unclear how exactly we would have to proceed in order to get to that final stage.

We took the first steps under the guidance of Dr. Yasar Ayaz and Sir Khawaja Fahad Iqbal and eventually the road ahead became clear step by step. We started by getting acquainted with Ubuntu and the Player\Stage interface. We studied from the Player\Stage manual and built and tested several smaller Player\Stage programs. Eventually we started integrating our C++ coding into our Player\Stage Simulations and programming for more complicated problem scenarios.

We learning coding in LabVIEW with Ali Bin Wahid who had already developed a small LabVIEW program to convert the wheelchair's joystick control to laptop control. We complicated upon his original program as well incorporate the Wireless settings and TCP broadcasting for LabVIEW so that LabVIEW could communicate in real time with our Player\Stage code.

Another challenging task was setting up the Player\Stage codes to work on the hardware rather than in simulations. The LRF had to be upgraded from the simulations proxy to the USB Port 0 proxy and next, we had to network the computers. The crux of our networking was based on the wireless TP-Link and in our C++ code that used sendData commands repetitively each time a SetSpeed call was made in our code.

Another learning outcome of the project was getting to understand the actual behavior of the wheelchair as it differed from the faultless simulations we could create in Player\Stage. Several test variables modified the behavior of our wheelchair in the real time e.g. the surface friction of the test space, the uncontrollable variations in the front wheels motion and the amount of battery power available to the wheelchair.

8.2 Future Recommendations

Future work on the Autonomous Robotic Wheelchair could involve improvements on the current Obstacle Avoidance and Navigation theorems. We have only made use of the most simplistic algorithms at this point which are good enough for room level movements but in order to make the wheelchair more enhanced we need to move towards building level navigation and for that complication navigation algorithms like the BrushFire or the DFS or BFS algorithms need to be implemented for wheelchair motion.

The wheelchair hardware also needs slight improvements. In its current state, the wheelchair table is not positioned correctly to accommodate fully the wheelchair user's legs. The table platform needs to be raised and also contraptions need to be installed for other sensors.

Lastly there is also a lot of scope for improvement in the design of our current LabVIEW program for driving the wheelchair. As it is our LabVIEW program is fairly simplistic and is merely translating the velocity values being received from the code. We need to develop a LabVIEW program that could interact with multiple Player\Stage C++ codes reading and processing data from multiple sensors. Once we have developed techniques to incorporate such processing within LabVIEW itself the wheelchair will be become vastly more intelligent than it is right now.

There is a long road ahead calling for extensive efforts in programming especially and students with a strong programming or technical background would be best suited to take on this project in the future.

APPENDIX A

REFERENCES

http://playerstage.sourceforge.net/

http://playerstage.sourceforge.net/player/player.html

http://www.ni.com/pdf/manuals/371303m.pdf

http://beej.us/guide/bgnet/

http://beej.us/guide/bgnet/output/print/bgnet_USLetter.pdf

http://www.lantronix.com/resources/networking.html

[1] Ruffles, P.C.; 2001, "Expanding the Horizons of Gas Turbine in Global Markets"; *ISABE 2001-1010*

[2] Dring, R. P., Joslyn, H. D., Hardin, L. W., and Wagner, J. H., 1982, "Turbine Rotor-Stator Interaction," ASME Journal of Engineering for Power, vol.104, pp.729-742

[3] Arndt, N., 1993, " Blade Row Interaction in a Multistage Low Pressure Turbine", ASME Journal of turbomachinery, Vol. 115, pp. 370-376

[4] Denton J.D., 1993, " Loss Mechanisms in Turbomachines:, ASME Journal of Turbomachinery, Vol.115

[5] Denton J.D., 1993, " Loss Mechanisms in Turbomachines:, ASME Journal of Turbomachinery, Vol.115

[6] R.E. Walraevens, A.E. Gnllus, "Stator-Rotor-Stator Interaction inAn Axial Flow Turbine And its Influence on Loss Mechanisms", AGARD CP 571, 1995, UK, pp 39(1-13).

[7] Dawes, W.N., 1994, " A Numerical Study of the Interaction of Transonic Compressor Rotor Over Tip Leakage Vortex with the Following Stator Blade Row", ASME Paper No. 94-GT-156

[8] Sharma, O,P., Renaud, E., Butler, T.L., Milasps, K., Dring, R.P., and Joslyn, H.D., 1988, " Rotor- Stator Interaction in Multi- Stage- Axial Flow Turbines", AIAA Paper No. 88-3013

[9] Busby, J.A., Davis, R.L., Dorney, D.J., Dunn, M.G., Haldeman , C.W., Abhari, R.S., Venable, B.L., and Delany, R.A., 1999, " Influence of Vane-Blade Spacing on Transonic Turbine Stage Aerodynamics: Part II- Time- Resolved Data and Analysis:, ASME Journal of Turbomachinery , Vol. 121, pp. 673-682

[10] Collar (1947), "The Expanding Domain of Aeroelasticity," *Journal of the Royal Aeronautical Society* 51-1.

[11] Bell Loyed, "Three dimensional Unsteady flow analysis in vibrating Turbine Cascades", Durham University, 1999.

[12] Jeff Green, PHD Thesis, "Controlling Forced Response of a High Pressure Turbine Blade", Royal Institute of Technology, Stockholm, 2006.