Human Robot Interaction for an Assistive Mobile Robot Interacting with Blind Adults

THESIS

in partial fulfillment of the requirements of
BITS F421T

by

Aditi Kulkarni
(2012A3PS161G)

Under the supervision of

M. Bernardine Dias, Ph. D.,
Associate Research Professor, Robotics Institute

BITS, Pilani, K.K. Birla Goa Campus

Date: 28.11.2015

CERTIFICATE

This is to certify that the thesis entitled Human Robot Interaction for an Assistive Mobile Robot interacting with blind adults is submitted by Aditi Kulkarni ID No. 2012A3PS161G in partial fulfillment of the requirements of **BITS F 423T/BITS F421T.** Thesis embodies the work done by her under my supervision.

Signature of the supervisor	Name: M. Bernardine Dias
Date: 28.11.2015	Designation: Associate Research Professor, Robotics Institute

INDEX

Abstract

The basic idea of this project is to develop an indoor navigation system to assist persons who are blind. The aim of the project comprises of three parts namely, an Android app for navigation, a Baxter robot for help with directions and a Pioneer 3 DX (P3DX) mobile robot for guiding the user. I worked with the P3DX robot for my thesis. It was used in a manner similar to a dog-guide for navigational assistance.

Our aim was to make the P3DX capable of leading the user to a destination of their choice. The P3DX is being designed to accurately locate its position, find an appropriate path to the destination and avoid obstacles on its way. We worked on getting the odometry of the robot using two different approaches. Two approaches were used and the results in both cases were compared.

The focus of my thesis was on human-robot interaction. The aim of research in human-robot interaction is to ensure the comfort of the user and enhance the usefulness of the robot. A number of possible devices and features were taken into consideration. After an evaluation based on the feasibility of these features and their pertinence to the final goal, an order of implementation was decided.

For persons who are visually impaired, speech generation assumes a lot of importance. To incorporate audio features, we first addressed the issue of positioning the speakers. This was important because it conveyed the size and distance of the robot to the user. The deck of the robot qualified as the ideal place. We used the Festival text-to-speech (TTS) software to give a voice to the robot. Festival allows the user to choose from a variety of voices that differ in gender, age and nationality. Thus users get to choose a voice that they are familiar with.

The users respond to the robot using a touchscreen Android phone. I have designed a basic keypad that will allow them to type in text using the Braille alphabet. To enable them to use touchscreen, an app called TalkBack was used. It speaks out the name of a button that a user places their finger on.The keypad's main usage is to get personal data from the user. This data is stored in a database that acts like the memory of a robot. So the next time the person visits, the robot can greet him with a more personal greeting than a generic hello.

I have also worked on the robot's ability to monitor its battery. I have explained the method of making the robot aware of its own charge. This saves the human effort of constantly monitoring the robot's charge. Also, the robot can check to see if it has enough charge for a journey before it begins. This ensures the safety of the user.

Apart from interactive features, remote control of the robot is a necessity to tackle any unprecedented event. It allows manual control when the robot fails to make a

decision or faces an unexpected dilemma. I was able to interface a joystick with the robot to control it remotely. While we try to minimize the possibility of error, we need to be prepared for an unforeseen emergency.

All these features have enhanced the interactive nature of the robot. We are also considering some more additions in the future. Work on human robot interaction is important primarily because it models the user's perception of the robot. It plays a major role in influencing user acceptance and approval.

1. Introduction

People with disabilities should have the means to live life like any ordinary person. Technology has made this possible to some extent over the last decade. Hearing aids, remote controlled wheelchairs and other devices have made it possible for differently abled persons to overcome their disabilities. For people who are blind, it is the white cane and dog-guides that help them navigate independently. Robot dog-guides have also begun to appear as alternatives to real dog-guides.

Our project aims at developing an indoor navigation system to assist persons who are blind. It comprises of three parts namely, an Android app for navigation, a Baxter robot for help with directions and a Pioneer 3 DX (P3DX) mobile robot for guiding the user. My thesis focuses on the third part involving the P3DX robot. Its function is that of a robot dog-guide with some additional capabilities.

The Pioneer 3 DX is a suitable robot for this use because of its small size and compact shape. With a harness attached to its deck, it can be moved around freely as it is not very heavy. It is not too light either which prevents it from toppling over. So a user can hold on to the harness and follow the robot to their destination of choice. The robot is being developed to find its path and avoid obstacles that may come in its way. The Player/Stage interface is being used for backend development. Algorithms and codes are tested using the Gazebo simulation tool before implementing them on the real robot.

An important aspect of utilizing robots for the purpose of assistance is the interaction that occurs between human and robot. The robot must make the users comfortable and carry out necessary tasks that will benefit them. This was the major area of research for my thesis. I began by trying to figure out all the possible features that could be useful. Then based on the feasibility, appropriateness and importance of each feature to the ultimate goal, a basic idea was formulated. At every stage of development the algorithm and codes were tested on the simulation tool. Only after ensuring the propriety of the application it was tested on the real robot. Thus we were able to avoid any accident with the robot.

In this report I have documented my work in the order of implementation. The entire report is divided into two halves - literature review and experimental developments. The literature review begins with a chapter on the robot. It helps the reader to understand robot hardware, its capabilities and limitations. The next part is to emphasize on certain characteristics of the robot that affect its motion and manoeuvrability. The last chapter in the literature review section is about the Player/Stage and Gazebo interface. After these chapters on literature review, the next chapters explain the actual work done in the thesis. The first chapter in this section is about localization which focuses on obtaining accurate odometry readings from the robot. The next chapter covers the most important part of my thesis. It explains the important features that enable the communication between human and robot. It also covers the work on charge-awareness. It also enlists the devices and features that are being considered for future work.

The thesis was carried out over a duration of around five months. Weekly meetings were held throughout this period. The discussions held during these meetings and the feedback received from mentors and team members was fundamental to attaining the proposed goals.

2. The Pioneer 3 DX Mobile Robot

Introduction:

Pioneer 3 DX is a compact, differential-drive robot.
Properties of the P3DX:

1. Embedded controller(32-bit RISC microprocessor)-Hitachi based H8S microprocessor(128K flash and 32K RAM;18MHz)
2. Motors with 500-tick encoders
3. 19cm wheels
4. 8 forward facing ultrasonic (sonar) sensors
5. 8 optional rear facing sonar sensors
6. 1, 2 or 3 hot-swappable batteries
7. Tough aluminum body

It can reach speeds of up to 1.6m/s and can carry a payload of 17kgs.[1]
It communicates using wi-fi.

Hardware - Major Components:

1. The deck- try centering weight on drive wheels and balance heavy weights placed on the sides.
2. Motor Stop Button-Stops power to the wheels and a beeping sound starts. At the rear of the deck
3. User Control Panel:
i. Red LED- ON when main power to robot
ii. Green LED- moderate toggle when in maintenance mode and waiting for client and high toggle speed when connected to client or in joydrive mode.
-can be programmed
iii. Serial connector-Tx and RX LEDs are also present. It is used to connect to an off-board PC via an RS232.This port is shared by the HOST serial port to which we connect the on-board computer or radio/Ethernet.
Precaution: Disconnect the wire if you plan to connect to the controller through the on-board computer or PC.
iv. Reset button - to unconditionally reset the microcontroller
v. Motor(white button)- When connected to a client press once for activating motors
When not connected press once for joydrive and again for self-test mode
4. SONAR Panel: The SONAR sensitivity adjustment screw is on the underside of the SONAR panel. Turn it clockwise for increasing sensitivity(in low-noise and

smooth-terrain environments) and counter-clockwise for reducing the sensitivity(in noise and uneven terrain environments)
5. Motors: High-speed, high-torque, reversible-DC motors.
Pneumatic tires inflated to 23 psi.
Precaution: Make sure to fill tires evenly

Software:

The microprocessor has ARCOS (Advanced Robot Control and Operations Software client-server interface). It is used if we need to build our own controls algorithm.[2]
Another software that comes equipped with all P3 robots is the ARIA or Advanced Robotics Interface for Applications. It is a C++ based development environment that provides TCP/IP communication with the robot.

ARIA:
ARIA is an object-oriented, application programming interface (API) for the
Adept MobileRobots (and ActivMedia) line of intelligent mobile robots.[3]
The ARIA package includes both source code and pre-built libraries.
These libraries and programs were build with
GCC 3.4 if on Linux, and MS Visual C++ 2010 (10.0), Visual C++ 2012
(11.0) and Visual C++ 2013 (12.0) for Windows Desktop if on Windows.
Using the above compilers for development is recommended.

Modes of Operation:

1. Client Mode- Controlled using a remote PC
2. Maintenance and standalone mode- programming
3. Joydrive and Self-test mode- using a joystick (when there is no controlling client) [2]

3. More about Differential Drives

Differential drive is a method of controlling a robot with only two motorized wheels [4]. The term 'differential' means that robot turning speed is determined by the speed difference between both wheels, each on either side of your robot. For example: keep the left wheel still, and rotate the right wheel forward, and the robot will turn left.
As long as both wheels go at the same speed, the robot does not turn - only moves linearly (forward or reverse).

PID Control (Proportion Differential Integral Control):
The aim of any control system is to move the actuators (motors, servos) using the microcontrollers based on data got from the sensors.

Terminology:

Error (E)- the difference between the present situation and what is desired. If the robot is supposed to be at A and is at B right now then the error is the absolute value of (A-B).

Proportion(P)-The proportion term is the error. So if the temperature required is t2 and we are at t1 then the proportion term will be t1-t2.

Derivative(D)- It is the error difference per unit time. If the error at time t1 was E1 and at time t2 was E2 then the derivative will be E1-E2/(t1-t2).
(It is not required if the device works slowly enough to stabilize itself or precision is not required)

Integral(I) - It is the accumulated error over time. If the errors at times t1, t2 and t3 are E1, E2 and E3, then the integral will be (E1/t1)+(E2/t2)+(E3/t3).
(It is required only if the robot is very heavy or its working on very steep surfaces(against gravity))

Tweak Constant (gain) -
If Kp, Ki and Kd are the constants, the complete PID equation becomes = KpP+KiI+KdD

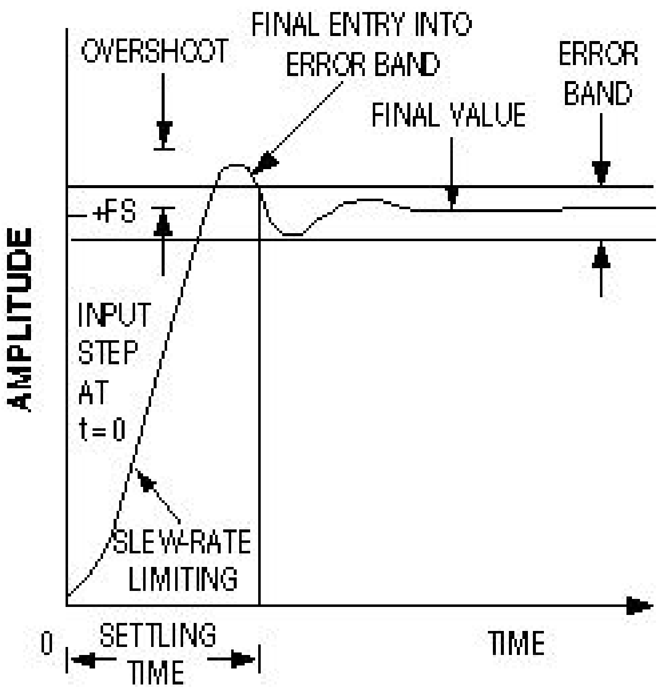

Fig. 1: Graph of amplitude versus time to depict settling time
(Source: https://www.societyofrobots.com/programming_PID.shtml)

Sampling Rate: It is the speed at which the algorithm can update itself. Slower sampling rates lead to overshoots as the algorithm takes longer to record the next change. The ideal sampling rate is between one-tenth and one-hundredth of the desired settling time.

H-Bridge:
It is a low cost way to control DC motors. It is the link between the digital output from a controller and the hardware. [6]

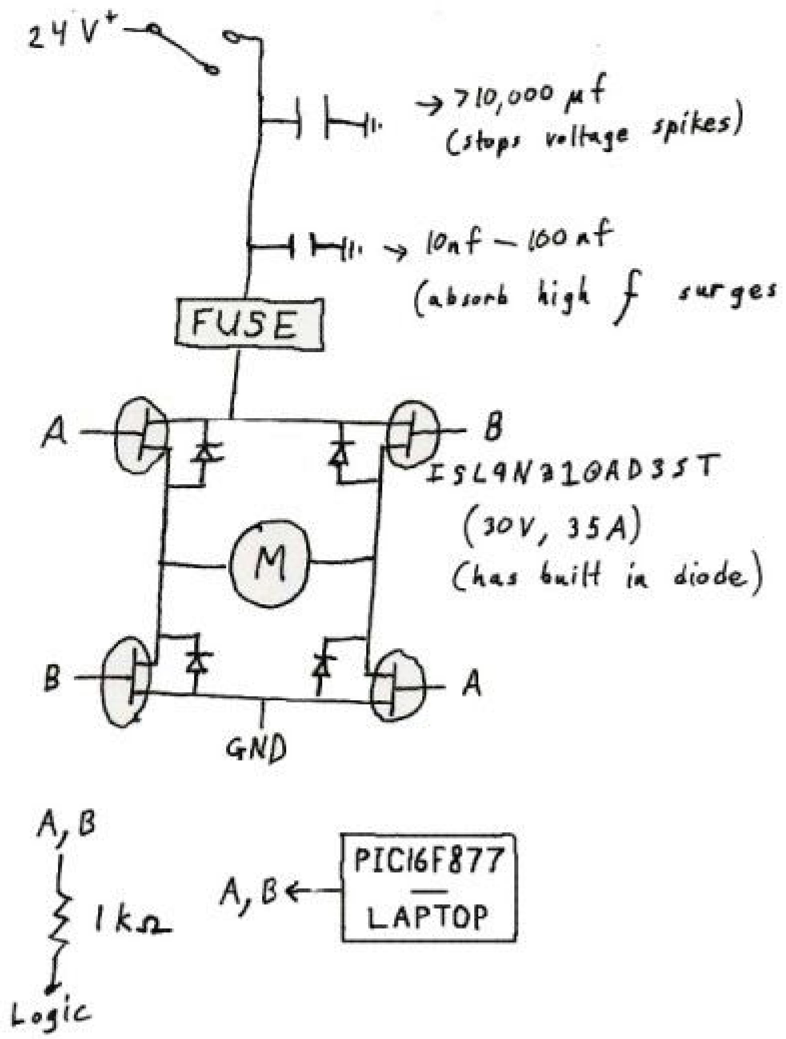

Fig 2: H-Bridge explained
Source: https://www.societyofrobots.com/schematics_h-bridgedes.shtml

PNP transistors for the top two MOSFETs and NPN for the bottom two.
A=0 B=0 Then the motor is off
A=0 B=1 The motor rotates clockwise
A=1 B=0 The motor rotates anti-clockwise
A=1 B=1 Will cause a short-circuit

For controlling the speed, PWM is used. Through inductance it controls the speed of the motors.

3. The Player/Stage Gazebo Interface

(A) Demo Simulation using ARIA

The robot was first tested using the Mobile Robots' Advanced Robot Interface for Applications (ARIA). ARIA is a C++ library that helps control I/O devices connected to the robot. Before starting with running the robot directly, they were tested on MobileSim which is a Simulator.
The steps for installing and running MobileSim and ARIA have been given in the following paragraphs.

Getting started with MobileSim:
Download MobileSim from the CD that comes with the robot or from [17]

To run MobileSim on the Command Prompt type:
MobileSim
or
MobileSim -m columbia.map -r p3dx

Probable Error and the fix-
Error message : Error creating temporary file.
Stage world file could not be created
Changing the TMP and TMP variables to accessible locations and then restarting Windows helped resolve the issue

Getting started with ARIA:
Download ARIA from the CD that comes with the robot or from [18]

ARIA is for the coding part (It is a C++ library):
Requirements - MS Visual C++ 2010/2013(Free Express editions are sufficient)
1. Start by trying to run the Demo files:
Probable Error and the fix-
Error message : The application was unable to start correctly-0xc000007b
Resolved after downloading MS Visual- a file *Ariavc-2010 or Ariavc-2013* must appear in your
C:\Program Files\MobileRobots\Aria folder.

Steps to use both together:
1. First run the MobileSim from the Command Prompt
2. Then start the demo files
3. A connection must be formed and data is streamed on the command prompt

1.

2.) Creating your own programs:lib*64* and x64

In ARIA there could be the following errors:

1.
2. Message: The AriaVC11.dll could not be opened

Check that the path to this file is correct

1.
2. Message: X86 module in conflict with x64 target

Go to properties/configuration manager and in the 'Active Solution Platform' drop down select x64.

1.
2. Then an executable file (.exe) will be created in some location specified in the output window. Copy that executable and paste it in the bin folder. Then run it. The command prompt shows that it has connected to the robot and the robot moves accordingly.

<u>Built-in interfaces in ARIA and MobileSIm:</u>

The Devices available for use on ARIA and MobileSim are:

SONAR Sensors

LASER Sensors

Bumper pads

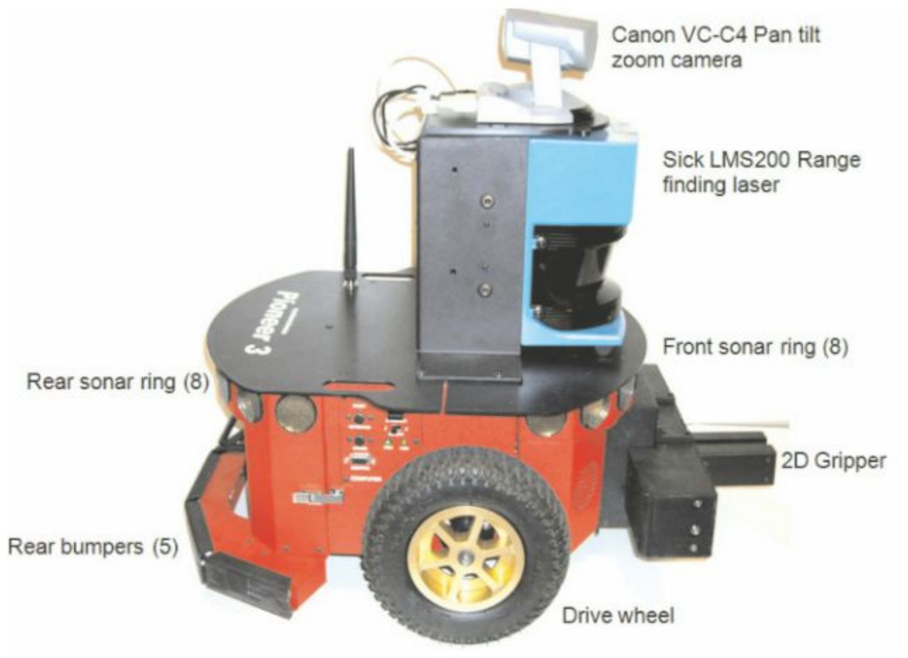

Fig3: The parts of the Pioneer 3 DX robot

(B) Player-Stage-Gazebo:

The Player Project creates free software that enables research in robot and sensor systems. The Player robot server is probably the most widely used robot control interface in the world, and supports a wide variety of hardware. Client libraries in C, C++, Python and Ruby are officially supported while Java, Ada, Octave and others are supported by third parties. Its simulation backends Stage and Gazebo are also widely used. [7]

-
- **Player robot device interface**

Player provides a network interface to a variety of robot and sensor hardware. Player's client/server model allows robot control programs to be written in any programming language and to run on any computer with a network connection to the robot. Player supports multiple concurrent client connections to devices, creating new possibilities for distributed and collaborative sensing and control.
Player supports a wide variety of mobile robots and accessories. [8]

-
- **Stage multiple robot simulator**

Stage simulates a population of mobile robots moving in and sensing a two-dimensional bitmapped environment. Various sensor models are provided, including sonar, scanning laser rangefinder, pan-tilt-zoom camera with color blob detection and odometry.
Stage devices present a standard Player interface so few or no changes are required to move between simulation and hardware. Many controllers designed in Stage have been demonstrated to work on real robots. [8]

-
- **Gazebo 3D multiple robot simulator**

Gazebo is a multi-robot simulator for outdoor environments. Like Stage, it is capable of simulating a population of robots, sensors and objects, but does so in a three-dimensional world. It generates both realistic sensor feedback and physically plausible interactions between objects (it includes an accurate simulation of rigid-body physics).
Gazebo presents a standard Player interface in addition to its own native interface. Controllers written for the Stage simulator can generally be used with Gazebo without modification (and vise-versa). [8]

Downloading and Installing Player[10]:
The official download page for player is on SourceForge:
http://sourceforge.net/projects/playerstage/files/
The steps for installation for Linux are as follows:
To install Player in the default location (/usr/local), follow these steps:

1. Download the latest Player source tarball (player-<version>.tgz) from SourceForge.
2. Uncompress and expand the tarball:
3. $ tar xzvf player-<version>.tgz
4. Change directory to Player's source directory:
5. $ cd player-<version>
6. Create a subdirectory called `build' and enter it:
7. $ mkdir build
8. $ cd build
9. To configure Player with default settings:
   $ cmake ../
10. Compile Player
    :$ make
11. Install Player. By default, Player will be installed in /usr/local so you need to become root for this step. Remember to return to your normal user ID afterwards.
12. $ make install
13. Executables(e.g., the Player utilities)are in /usr/local/bin, libraries (e.g.,libplayercore,libplayerdrivers) are in /usr/local/lib, and so on.

- Errors:

Many errors did come up while installing player. Quite a lot of documentation is available to debug them if you are installing it on Linux. I have listed the errors that I got along with the solutions that fixed them.

Error Message (After Step 6)-
client_libs/libplayerc/bindings/python/CMakeFiles/_playerc.dir/playercPYTHON_wrap.o: '$self" undeclared (first use in this function)

Since I was not using any Python dependencies, I disabled them using CMake. So first delete the CMakeCache.txt created in build and then run the following instead of cmake ../:
cmake –DBUILD_PYTHONC_BINDINGS –DBUILD_PYTHONCPP_BINDINGS ../
This should fix the above error.
If you want to use the python dependencies, then try the following:
sudo apt-get install python2.7-dev

After the above error, I got a similar error for Ruby dependencies. It was solved by installing Ruby dependencies. The command for that is:
sudo apt-get install Ruby2.0.0
If you do not need these dependencies:
Run cmake as : cmake –DBUILD_RUBY_BINDINGS ../

If you get any ‘permission denied’ errors during the process, it means that you need administer rights. Just add ‘sudo’ before the command. It should work.
After the installation, test that you have everything in place to get started with player:
On the terminal type: player pioneer.cfg
If it works, then you have successfully installed player. If not, the errors need to be resolved. The errors at this stage are usually one of the following.

Error Message : player command not found
This means that player was not added to your system path. You can do two things here.
Modify the system variables as follows-

```
Export PATH=$PATH:”/home/admin/player-…(version)”
Export
LIBRARY_LD_PATH=$LIBRARY_LD_PATH:”/home/admin/player-…(version)/build”
Export PLAYERPATH=”/home/admin/player-…(version)”
```

Type the player pioneer.cfg command after changing your directory (using ‘cd’) to the config directory in the player folder.

Downloading and Installing Gazebo [12]:

a. Setup your computer to accept software from packages.osrfoundation.org:
   sudo sh -c 'echo "deb http://packages.osrfoundation.org/gazebo/ubuntu `lsb_release -cs` main" > /etc/apt/sources.list.d/gazebo-latest.list'
b. Setup keys
   wget http://packages.osrfoundation.org/gazebo.key -O - | sudo apt-key add -
c. Install Gazebo.
   sudo apt-get update
   sudo apt-get install gazebo6
d. For developers that work on top of Gazebo, one extra package
   sudo apt-get install libgazebo6-dev
e. Check your installation
   i. * The first time gazebo is executed requires the download of some models and it could take some time, please be patient.
      gazebo

Running Player and Gazebo:

1. On a terminal type: cd /src/gazebo/worlds
2. gazebo pioneer2dx.world
3. On a new terminal: cd /src/player-3.0.2/config
4. player position2d.cfg
5. On a new terminal: playerv
6. In the Player Viewer GUI: Go to Devices, select subscribe, then select command and then select enable.

(C) Coding in Player for Gazebo:

Player/Stage is a robot simulation tool, it comprises of one program, Player, which is a Hardware Abstraction Layer. That means that it talks to the bits of hardware on the robot (like a claw or a camera) and lets you control them with your code, meaning you don’t need to worry about how the various parts of the robot work. Gazebo is a plugin to Player which listens to what Player is telling it to do and turns these instructions into a simulation of your robot. It also simulates sensor data and sends this to Player which in turn makes the sensor data available to your code.
A simulation then, is composed of three parts:
• C++ code: This talks to Player.
• Player: This takes your code and sends instructions to a robot. From the robot it gets sensor data and sends it to your code.

• Gazebo: Gazebo interfaces with Player in the same way as a robot's hardware would. It receives instructions from Player and moves a simulated robot in a simulated world, it gets sensor data from the robot in the simulation and sends this to Player.

- Steps to creating a simulation:
    - Write Gazebo worldfile
    - Start Gazebo
    - Write corresponding Player configuration file
    - Start Player
    - Start client program

- Note:
  Gazebo must be started *before* Player
  Player must be *re-started* whenever Gazebo is restarted

<u>(D) Interfaces, Drivers and Devices :</u>
• <u>Drivers</u> are pieces of code that talk directly to hardware. These are built in to Player so it is not important to know how to write these as you begin to learn Player/Stage. The drivers are specific to a piece of hardware so, say, a laser driver will be different to a camera driver, and also different to a driver for a different brand of laser. This is the same as the way that drivers for graphics cards differ for each make and model of card. Drivers produce and read information which conforms to an "interface"[11].
• <u>Interfaces</u> are a set way for a driver to send and receive information from Player. Like drivers, interfaces are also built in to Player and there is a big list of them in the Player manual . They specify the syntax and semantics of how drivers and Player interact [11].
• A <u>device</u> is a driver that is bound to an interface so that Player can talk to it directly. This means that if you are working on a real robot that you can interact with a real device (laser, gripper, camera etc) on the real robot, in a simulated robot you can interact with their simulations.

Player is a server, and a hardware device on the robot is subscribed as a client to the server via a thing called a proxy. The .cfg file associated with your robot (or your simulation) takes care of telling the Player server which devices are attached to it, so when we run a configuration file (.cfg) this starts up the Player server and connects all the necessary hardware devices to the server.

In Player/Stage the same command will start the Player server and load up the worldfile in a simulation window, this runs on your computer and allows your code to interact with the simulation rather than hardware.

Finally, in order to **compile** your program you use the following commands (in Linux):
g++ -o example0 `pkg-config --cflags playerc++` example0.cc `pkg-config --libs playerc++`

The first thing to do within your code is to include the Player header file. Assuming Player/Stage is installed correctly on your machine. There may be several proxies connected to the server at any time. This can be done with the line #include (if you're using C then type #include instead).
Next we need to establish a Player Client, which will interact with the Player server for you. To do this we use the line: PlayerClient client_name(hostname, port); What this line does is declare a new object which is a PlayerClient called client_name which connects to the Player server at the given address. If your code is running on the same computer (or robot) as the Player server you wish to connect to then the hostname is "localhost" otherwise it will be the IP address of the computer or robot. The port is an optional parameter usually only needed for simulations, it will be the same as the port you gave in the .cfg file. This is only useful if your simulation has more than one robot in and you need your code to connect to both robots. So if you gave your first robot port 6665 and the second one 6666 (like in the example of section 4.2) then you would need two PlayerClients, one connected to each robot, and you would do this with the following code: PlayerClient robot1("localhost", 6665); PlayerClient robot2("localhost", 6666); If you are only using one robot and in your .cfg file you said that it would operate on port 6665 then the port parameter to the PlayerClient class is not needed. Once we have established a PlayerClient we should connect our code to the device proxies so that we can exchange information with them. Which proxies you can connect your code to is dependent on what you have put in your configuration file. For instance if your configuration file says your robot is connected to a laser but not a camera you can connect to the laser device but not the camera, even if the robot (or robot simulation) has a camera on It . [11]

Proxies take the name of the interface which the drivers use to talk to Player. For example:
driver (name "stage" provides ["6665:position2d:0" "6665:sonar:0" "6665:blobfinder:0" "6665:laser:0"])

Here we've told the Player server that our “robot” has devices which use the position2d, sonar, blobfinder and laser interfaces. In our code then, we should connect to the position2d, sonar, blobfinder and laser proxies like so:

```
Position2dProxy positionProxy_name(&client_name,index);
SonarProxy sonarProxy_name(&client_name,index);
BlobfinderProxy blobProxy_name(&client_name,index);
LaserProxy laserProxy_name(&client_name,index);
```

A full list of which proxies Player supports can be found in the Player manual , they all follow the convention of being named after the interface they use. In the above case xProxy_name is the name you want to give to the proxy object, client_name is the name you gave the PlayerClient object earlier and index is the index that the device was given in your configuration file (usually 0).

(E) Setting Up Connections: an Example.

For an example of how to connect to the Player sever and device proxies we will use the following example configuration file. For convenience this is reproduced below:

```
driver ( name "stage" plugin "libstageplugin" provides ["simulation:0" ] # load the named
file into the simulator worldfile "worldfile_name.world" )
driver ( name "stage" provides ["6665:position2d:0" "6665:sonar:0"
"6665:blobfinder:0""6665:laser:0"] model "bob1" )                          [11]
```

To set up a PlayerClient and then connect to proxies on that server we can use principles discussed in this section to develop the following code:

```
#include
#include
int main(int argc, char *argv[])
{ /*need to do this line in c++ only*/
using namespace PlayerCc;
PlayerClient robot("localhost");
Position2dProxy p2dProxy(&robot,0);
SonarProxy sonarProxy(&robot,0);
BlobfinderProxy blobProxy(&robot,0);
LaserProxy laserProxy(&robot,0);
//some control code
return 0;
}                                                                          [11]
```

Writing Configuration Files :

A Player configuration file is a text file that, by convention, has the extension .cfg. The file is composed of one or more driver sections, each of which instantiates and configures a single driver. Consider an example:

```
driver
(
name "sicklms200"
provides ["laser:0"]
)
```

This block instantiates the sickLMS200 driver. The provides keyword is used to specify the device address to which the driver will be bound. Using this configuration, an instance of the sickLMS200 driver will be available as laser:0.

There are four driver-independent options:

-
- name(string): The name of the driver to instantiate, as it was provided to DriverTable::AddDriver(). This option is mandatory.
-
- plugin(string): The name of a shared library (i.e., a "plugin") that contains the driver. This shared library will be loaded before attempting to instantiate the driver. If you are using a built-in driver, you do not need to use this option.
-
- provides(tuple of strings): The device address(es) through which the driver can be accessed. This option is mandatory.
-
- requires(tuple of strings): The device address(es) to which the driver will subscribe.

Example for provides and requires:
Some drivers require other devices. For example, the vfh driver, a local navigation / obstacle-avoidance algorithm, needs two devices: a position2d device to command and a laser device from which to read laser scans. This driver in turn provides a position2d device to which target poses can be sent. So the vfh driver might be configured like so:

```
driver
(
name "vfh"
provides ["position2d:1"]
requires ["position2d:0" "laser:0"]
)
```

Drivers providing "position2d:0" and "laser:0" must have already been declared. In general, a driver's required devices must be instantiated prior to the driver itself being instantiated. Circular dependencies are not allowed.
Device addresses for the provides and requires options are given as strings in a 5-part colon-separated format:

```
key:host:robot:interface:index
```

Only the interface and index fields are required. Default values will be taken for any fields left blank. The default values for host and robot can vary depending on the context, but are usually "localhost" and 6665, respectively. The default value for the key field is NULL (i.e., no key). Leading fields can be omitted, but intermediate fields must be explicitly left blank. For example, to specify a key but leave host and robot blank:

```
odometry:::position2d:0
```

The purpose of the key field is to allow a driver that supports multiple interfaces of the same type to map those interfaces onto different devices. For example, consider the p2os driver, which supports three position2d interfaces:

```
driver
(
name "p2os"
provides ["odometry:::position2d:0"
"compass:::position2d:1"
"gyro:::position2d:2"]
)
```

This declaration says, provide the robot's odometry as position2d:0, the compass as position2d:1, and the gyro as position2d:2. The mapping of a driver's interfaces to devices is up to you.
The same goes for requires.

4. Localization

Localization is a technique used by a robot to estimate its position. The goal of localization is to estimate its state given a history of observations and actions [15]. Some important terms in localization.

State-Information sufficient to predict observations
Observation-Information derived from state
Action-Inputs that affect the state

A state is any parameter/s used to describe the system. For instance the x co-ordinate, the y co-ordinate, temperature, etc.
Observations in robotics are largely based on odometry or the study of data obtained from sensors. It is extremely important to have good if not precise sensor readings to make any deductions based on the readings. But we cannot solely rely on the sensor readings like GPS data for position estimation because of the following reasons-

1.
2. There are environments where signals cannot received
3.
4. The accuracy and speed are insufficient. Most robots do not have a large enough receiver to store the data obtained.

Localization is of two types:

1.
2. Absolute-

It defines the state of the robot relative to a common, fixed reference frame. It is useful for co-ordination, navigation, etc.

1.
2. Relative- It defines the state relative to a local, non-fixed reference frame. It is useful for exploration, displacement estimation, etc.

(A) Some equations for the differential drive robot:

vl – Left wheel velocity
vr – Right wheel velocity
R - turn radius
t - time between two readings
v- net velocity
w- rotational velocity
theta1 – current angle of rotation with respect to the starting angle
theta0 – previous angle of rotation with respect to the starting angle
xpos1- current x co-ordinate with respect to the starting position
ypos1- current y co-ordinate with respect to the starting position

xpos0- previous x co-ordinate with respect to the starting position
ypos0- previous y co-ordinate with respect to the starting position

vl= $\dfrac{Current\ encoder\ ticks\,(\,) - previously\ saved\ encoder\ ticks}{t} \times \dfrac{\pi}{180} \times R$

vr= $\dfrac{Current\ encoder\ ticks\,|motor| - previously\ saved\ encoder\ ticks}{t} \times \dfrac{\pi}{180} \times R$

v= $\dfrac{vl+vr}{2}$

w= $\dfrac{vl-vr}{R}$

theta1=theta0+t*w

xpos1=xpos0+v*cos(theta1)

ypos1=ypos0+v*sin(theta1)

(B) Comparison between direct readings and calculations from velocity and angular velocity

There are two ways to get the position of the robot.
The two methods have been explained below and a comparison has been drawn between the two methods.

[I] From Velocity and Angular readings:

The Player library has functions that return the velocity and angular velocity readings. The functions are part of the *positionProxy* interface. The velocity values are calculated in metres per second and angular velocity values are calculated in radians per second. The functions are:

1. GetXSpeed()-It gives the velocity of the robot in the direction of linear motion. This is done using the formula for differential drive robots.
velocity=(velocity of right wheel)+(velocity of left wheel)/2
Usage – double x_velocity=<position_proxy_name>.GetXSpeed();
2. GetYSpeed()-It gives the side velocity of a robot. We do not need this function as the robot travels in a straight line unless it has to take a turn. It is used for robots that move like a ball in all directions.
Usage – double y_velocity=<position_proxy_name>.GetYSpeed();
3. GetYawSpeed()-It calculates the angular speed of the robot.
angular velocity=difference between the velocities of the two wheels
Usage – double angular_velocity=<position_proxy_name>.GetYawSpeed();

The odometry equations require the velocity and the angular velocity for calculating the position of the robot. So using these functions, we can get the position of the robot.
Let v be the speed, w be the angular velocity, t be the time between two readings. Using these we can calculate x(the X position), y(the Y position) and theta(the angular position). The equations used are as follows:
theta1=theta0+w*t
y1 = y0 + v*sin(theta1)*t
x1 = x0 + v*cos(theta1)*t
where theta1 is the new value and theta0 is the previous value, x1 is the new position and x0 is the previous position, y1 is the new position and y0 is the previous position.

[II] Direct readings:

The Player library has some functions that give the desired position readings directly. These are also part of the *positionProxy* interface. They are:

1. GetXPos()-It gives the x-position with respect to the starting position.
2. GetYPos()-It gives the y-position with respect to the starting position.
3. GetYaw() -It gives the angular deviation with respect to the starting angle.

These functions calculate the positions using odometry equations. This can be understood by reading the positionProxy.h file.

[III] Comparison between the two methods:

On comparison, the second method gives better results. A summary of the results has been tabulated below.

Reading	Actual(measured)	Method 1	Error	Method 2	Error
Linear 1	0.2413 m	0.1579	0.3456	0.252	-0.0404
Linear 2	0.5334 m	0.3169	0.4058	0.584	-0.0948
Linear 3	0.8382 m	0.5624	0.3290	0.722	0.1386
Theta 1	0 degrees	0.0008	-	0	0
Theta 2	90 degrees	38.5	0.5722	82.99	0.0778

These readings are not accurate. However, they serve the purpose of reaching the goal while using the A-star path-planning algorithm.
In the future, work needs to be done on the following aspects.
1. Getting more accurate readings using encoder ticks for obtaining the velocities of the two wheels.
2. Getting the initial position of the robot.

5. Human Robot interaction with the P3DX

The P3DX comes equipped with a few *devices* like sonars and USB ports. A few accessories can be added to the robot to make it more effective in assisting visually impaired people. Some of them have been listed below.

(A) **Speaker Positioning and Speech Generation:**

The human ear can detect the source of sound based on direction and distance. The direction is determined by two different methods – one for the horizontal plane and one for the vertical plane along the midline of a person's head [22]. The difference in frequency of sound helps in determining the source of the sound. For the user to get an idea about the robot's size and distance, we selected the deck of the robot as the ideal place for conveying robot position. In the figure below, the black top is the deck of the robot.

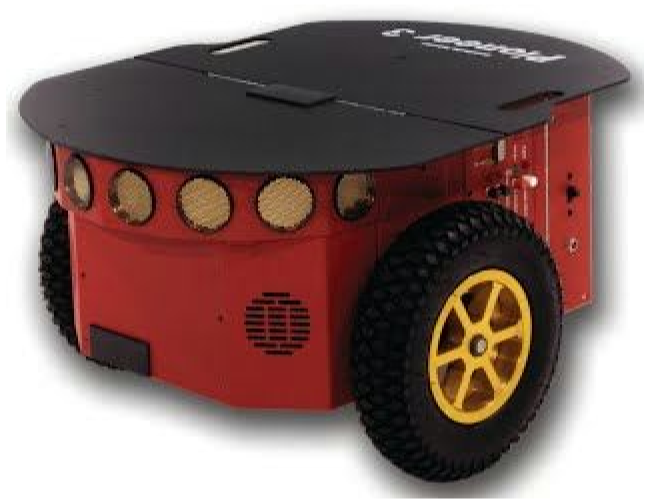

Fig. 4: The deck of the P3DX (ideal for speaker positioning)

When compared to a dog-guide, the positioning of the laptop closely resembles that of the dog's mouth.

Some users may prefer earphones which we plan to support later but this will occur after meeting the robot. This delay will allow users to hear the robot size and direction.

An audio feature is probably the most important means of interaction of a machine with a visually impaired person. For audio interaction we have opted to use simple text -to-speech (TTS) software installed on the robot's laptop. Modern TTS softwares are more natural in their tone and modulation than older versions. Most systems also provide voices with different regional accents. We examined a number of TTS options and selected based on the variety of voices available, quality of sound, etc.. Festival TTS software was chosen [4].

<u>Getting Started with Festival:</u>

1. Install preliminary packages-

```
 sudo apt-get install festival festlex-cmu festlex-poslex festlex-oald
libestools1.2 unzip
```

2. Install the different voices-

a. Standard diphone voices provided by the CMU speech group

<u>American males-</u>

```
sudo apt-get install festvox-kallpc16k festvox-kdlpc16k
```

<u>British males-</u>

```
sudo apt-get install festvox-don festvox-rablpc16k
```

b. Enhanced CMU Arctic voices developed by the CMU Language Technologies Institute-

<u>American Female-</u>

```
wget -c
http://www.speech.cs.cmu.edu/cmu_arctic/packed/cmu_us_clb_arctic-0.95-release.tar.b
z2
tar xf cmu_us_clb_arctic-0.95-release.tar.bz2
sudo mkdir -p /usr/share/festival/voices/english/
sudo mv cmu_us_clb_arctic /usr/share/festval/voices/english/
sudo mv /usr/share/festval/voices/english/cmu_us_clb_arctic_clunits
```

<u>Indian Male-</u>

```
wget -c
http://www.speech.cs.cmu.edu/cmu_arctic/packed/cmu_us_ksp_arctic-0.95-release.tar.
bz2
tar xf cmu_us_ksp_arctic-0.95-release.tar.bz2
sudo mkdir -p /usr/share/festival/voices/english/
sudo mv cmu_us_clb_arctic /usr/share/festval/voices/english/
sudo mv /usr/share/festval/voices/english/cmu_us_ksp_arctic_clunits
```

3. Testing the voices-

   a. Run festival without any options
   b. Then type

      ```
      festival>(voice.list)
      ```

   to check if all the installed voices have been placed properly.

The output should be something like this:

```
(kal_diphone
rab_diphone
don_diphone
cmu_us_clb_arctic_clunits
cmu_us_ksp_arctic_clunits
cmu_us_jmk_arctic_clunits
ked_diphone
cmu_us_awb_arctic_clunits)
```

c. Then to hear the voices say something:

```
festival>(SayText “Hello World.”)
```

4. To include festival as part of a code-
Create a script file with greetings and the main c++ codes [19]

(B) **Retaining User Information for Evolving HRI:**

Good assistants remember the people they help, so we have equipped our robot with a database for user information. When a user gets the robot's help for the first time an account will be created with basic personal information obtained during the interaction. (eg. name, age, gender, destination, etc.)
We plan to get this information by asking the users to enter data through an accessible Android app [5]. This information can also be obtained with speech-to-text software, however manual methods are usually found more reliable [26].

CELL 1	CELL 2	Speakers
CELL 3	CELL 4	Speed
CELL 5	CELL 5	Volume
Send	Space	STOP/GO

Fig 5: Interface for supporting Android app

Touchscreen interaction for people who are blind are slightly different. One button press reads the label, while second press activates the button. This basic keypad will allow users to type in text using the Braille alphabet. To enable them to use touchscreen, an app called TalkBack [25] was used. It speaks out the name of a button that a user places their finger on. The users then have to double click it to press the button. The keypad’s main usage is to get personal data from the user. This data is stored in a database that acts like the memory of a robot. So the next time the person visits, the robot can greet him with a more personal greeting than a generic hello. The keypad also has a button so that the users can stop the robot if they need it to. Besides, it allows the users to select a particular voice and adjust the volume.

When the robot is first called upon to assist the user, it will be informed about the destination If it is the user's first interaction with the robot, details will be requested from the user and saved. The user will also be asked to select a preferred voice from the available choices. If the user's information is already stored in the system, personal preferences and destination will be loaded into the robot. Therefore, the interaction at the beginning of a journey will either be a generic greeting for the new user or a more customized greeting for a returning user.

(C) **Periodic User Checking:**

After the user has set out on their journey with the robot, they might wish to rest for a moment before proceeding. While this could be a rare occurrence in an indoor environment, it is important to be taken into consideration since some users need a break for stamina or a secondary task (eg. phone call). The robot asks the user every five minutes if they need anything. The user also has the option to press the STOP/GO button to inform the robot of their choice. In the future, we are considering incorporating a pulse sensor to the robot's handle to infer tiredness from the user's pulse rate and interact accordingly.

During the journey, the robot will inform the user about obstacles in the path, turning points, elevator areas, water coolers and other surrounding features that will help them be useful. Details are important information when guiding this population. As mentioned before, periodic checking is important for a longer journey.

(D) **Charge Awareness:**

The robot has the ability to check it's level of charge before it undertakes any task. Thus for every user, the robot can safely complete the entire trip.

Approach:

Identify low battery level as a high priority condition (charge awareness):
Home base-mobile charging systems will prove less costly than a fixed station
State estimation-Determine work that can be done with the available battery charge. It is similar to estimating the number of miles a car can travel based on the amount of fuel in its tank.
Power estimation based on Operation State-Roughly we can classify the robot as being in either a moving state or an idle state.
CurrentTimeConsumed=LastTimeConsumed-idleTime(start,end)-movingTime(start,end)
Power estimation based on Battery Voltage-The battery discharges in a non-linear manner. So the function for discharging needs to found from experimentation.

Main Areas:

1. Recharging Hardware-
   Up to 3 hot-swappable batteries
   7 Ah, 12 VDC, 252 watt-hours
   Charge life - 6 hours or more / 4 hours with on-board computer
   Charger - 800 mA
   Above 12.5 V - LED glows bright green
   Below 11.5 V - LED turns orange then red
   Buzzer alarm sounds repetitively if the battery voltage goes below 11.5 V.

2. Recharging Location-
   For starters we can just take the P3DX to the charger.

3. Conditions for recharging-
   The p2os driver provides the following device interface (in the configuration file).
   power - returns the current battery voltage (12 V when fully charged)

Cost Functions:

Cost functions translate work of the robot to the cost to complete the task.

Pairwise cost functions A and B can be used to determine the cost of completion of B given the completion of A.

Schedulers keep track of a robot's schedule that keeps the track of a robot's tasks and the order in which they need to be performed.

Balanced schedule comprises of a schedule comprising of tasks that need to be completed given the battery charge of the robot and some scheduled recharging tasks.

Cost of a robot is the distance that it needs to travel for the completion of a task. It is calculated by considering the type of a task.

Our cost function must have the following properties.

1. Conservative in our creation of balanced schedules and pessimistic in the availability of mobile chargers. It is assumed that the robot needs to go to the home station for recharging.
2. An accurate estimate of the current runtime is necessary when a cost function evaluates a robot's cost.
3. All cost functions must leave the schedule in a stable state []. The robot must not deplete its batteries during the completion of a task. It must have enough battery left to recharge. This is done by guaranteeing that the robot will have enough battery to reach the home recharge station after the completion of tasks used by the cost function.

Algorithm for Recharging Program:

At docking station the robot will receive the destination
When called by Baxter, check for total cost
After reaching Baxter, check battery (NullToPoint)
When task is assigned calculate cost and ensure that the battery will be enough for the j ourney and back (PointToPoint) to the docking station or Rathu Baxter and then to the recharging point(PointToRecharge).
If the battery is insufficient go to charging station
After a complete journey check battery level and if charging is required. If yes then to charging station or to docking station.

Calculations for the Algorithm:

The power required for the motors is provided by using PWM(DC batteries). The PWM cycle is a 50 microseconds or 20KHz cycle. The ratio is 0-500 for 0-100% of the duty cycle.
So for running the motors for a duration of t minutes, the power needed can be calculated as

P= (½) x t x 60x 500 watts

However, this is not what actually happens. As time progresses, the discharging function of the robot varies non-linearly.
Also for the idle state (without the motors running), the robot loses charge non-linearly. The graph for the same has been given below.

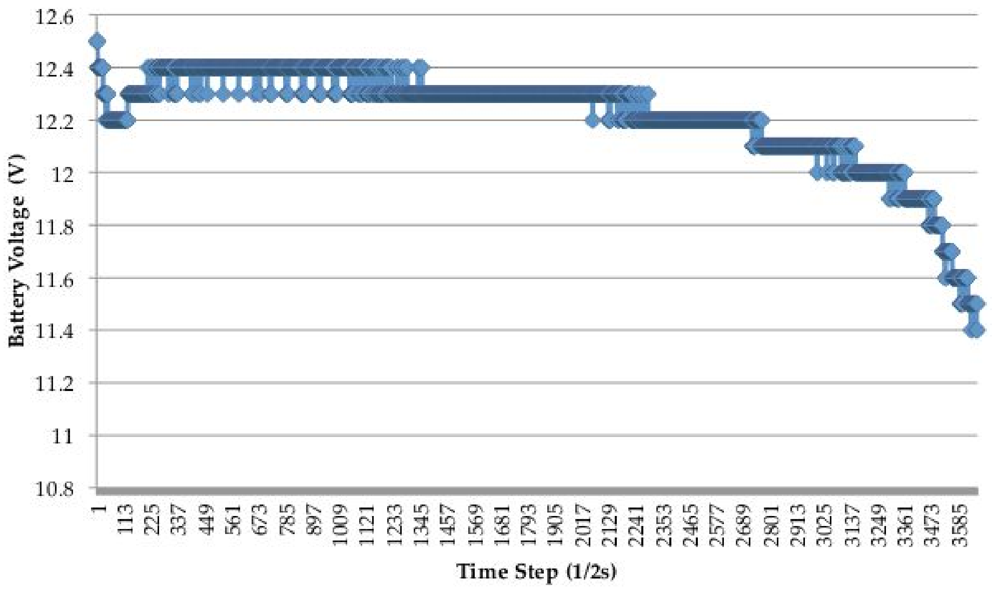

Fig. 4: Graph of voltage versus time depicting volatility between voltage thresholds
Source:http://www.ri.cmu.edu/publication_view.html?pub_id=7291&menu_code=0307

Future Work:

In this paper, we have made an effort to enhance a user's experience while interacting with the robot. However, numerous features still remain to be improved or implemented. We hope to add many new features for a more enhanced interaction. They are mentioned here.

(E) **Manipulator Arm:**
A manipulator is a device that helps the robot access and grip objects around it if it has a gripper attached to its end. If we connect it to the P3DX, it can be used to pick up objects dropped accidentally by the user. To pick the object we first need to detect that an object has fallen. For the detection of a falling object we have the following options.
i. Image processing – The laptop camera can be used to capture the image of a falling image. Once this has been detected, MATLAB or OpenCV can be used to identify the falling object and then we can control the gripper. There is also a software called MobileEyes specifically for the P3DX.
ii. Mechanical – Piezo sensors can be used to detect the object by direct contact. The object either bends the film or taps the surface. The change in resistance detected can be used to identify a following object.
After an object has been detected we need to control the gripper as accurately as possible. It needs to move in the direction of the object and then pick up the object. The grippers that are compatible with the P3DX are as follows.

Gripper	DOF	Payload	Range	Arm Speed	Interface
Gamma 1500	7	1500g(full extension) 2000g (partial)	68 cm	20 cm/sec	Serial or USB
Gamma 1500	4	300g	68 cm	20 cm/sec	Serial or USB
Epsilon 300	7	300g	53.4 cm	20 cm/sec	Serial or USB

(F) **Pedometer/Calorie counters:**
For most people exercise is all about making your body attractive and appealing. From this point of view exercise would be almost meaningless for them. However if viewed as

a source of refreshment and a guarantee of good health, exercise is essential for everyone. A pedometer that can tell a person about the steps walked, distance covered and the calories burnt helps some people to keep themselves motivated to exercise. There are blind-friendly pedometers in the market that are available for around $20. We can either attach a normal pedometer to our robot and give it audio features. Or we could use an inexpensive pulse sensor connected to the handle to count the number of calories that a person burns.
A pulse sensor could also act as a health monitoring device for the elderly. A sudden fall or rise in heart rate could be observed and other people could be notified in time.
An example is the pulse sensor from Sparkfun.

(G) **Camera:**
A camera can be added to the robot for detecting approaching people. This will alert them that someone is approaching them or is in their way. The same camera can be used to find and recognize the user when meeting the robot.

(H) **Music Player:**
This is an optional accessory that could be useful to people while walking. It is easy to integrate and could just make the journey more enjoyable.

7. Conclusion

As part of this project, a P3DX robot was programmed in order to find an optimum path and avoid obstacles while doing so. In addition to this,interactive features were added to the robot in order to make it user-friendly and comfortable to use. Simulation softwares that are available made the programming and testing very convenient. Remote control was enabled to avoid any accidents while testing.
While some features were added as a part of my thesis work such as audio features and charge awareness, a lot of work needs to be done to make the robot truly perfect. The addition of a robot arm will ensure that users retrieve the objects that they may have dropped while walking along the path. When an object is dropped on a soft surface, it does not always make a sound. So it may happen that the person using the P3DX may not realize that he or she has dropped something. The P3DX must be made capable of first detection of a falling object and then retrieving it using the robot arm. The second addition will be the self-charging ability. This is the next step for the charge awareness feature. While the robot can identify a low charge, it must also be able to plug itself into a charging port in the future. This is tedious because of the small size of the charging port and no available mounting facility as yet. Besides these useful features, options for listening to music or the news can be incorporated to make the experience more enjoyable for the user.
While working on this project I have had the opportunity to amass a lot of knowledge about the working of the robot, the software required and also the way to present my work. It has helped me develop many skills not just specific to the project but for any kind of research work. I hope to continue developing my skills so that some day I could make a valuable contribution to the field of assistive technology.

References

[1] http://www.mobilerobots.com/ResearchRobots/PioneerP3DX.aspx
[2] http://www.inf.ufrgs.br/~prestes/Courses/Robotics/manual_pioneer.pdf
[3] http://robots.mobilerobots.com/Aria/docs/main.html#intro
[4] https://www.societyofrobots.com/programming_differentialdrive.shtml
[5] https://www.societyofrobots.com/programming_PID.shtml
[6] https://www.societyofrobots.com/schematics_h-bridgedes.shtml
[7] http://playerstage.sourceforge.net/wiki/Main_Page
[8] http://playerstage.sourceforge.net/
[9] http://sourceforge.net/projects/playerstage/files/
[10] http://playerstage.sourceforge.net/doc/Player-svn/player/install.html
[11] http://playerstage.sourceforge.net/doc/playerstage_instructions_2.0.pdf
[12] http://gazebosim.org/tutorials?tut=install_ubuntu&cat=install
[13] http://gazebosim.org/tutorials?cat=connect_player
[14] http://gazebosim.org/tutorials?tut=quick_start&cat=get_started
[15]http://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/16311/www/s16/syllabus/ppp/Lec13-Localization.pdf
[16]
http://ocw.mit.edu/courses/aeronautics-and-astronautics/16-412j-cognitive-robotics-spring-2005/projects/1aslam_blas_repo.pdf
[17] http://robots.mobilerobots.com/wiki/MobileSim
[18] http://robots.mobilerobots.com/wiki/ARIA#Download_Aria
[19] http://ubuntuforums.org/showthread.php?t=751169
[20] Victor Marmol,Balajee Kannan, and M Bernardine Dias "Market-based Coordination of Recharging Robots" tech. report CMU-RI-TR-12-28, Robotics Institute, Carnegie Mellon University, September, 2012
[21]Stephanie, (2010, April 5), *12 Ingenious gadgets and technologies designed for the blind.* [Online]. Available:http://weburbanist.com/2010/04/05/12-ingenious-gadgets-technologies-for-the-blind/
[22]J. D. Warren, T. D. Griffiths. "Distinct Mechanisms for Processing Spatial Sequnces and Pitch Sequences in the Human Auditory Brain". *The Journal of Neuroscience,* vol. 13, pp. 5799-5804, July 2003.
[23] Ron Kurtus. (2002, August 22), *Hearing Direction and Distance (2nd ed.)* [Online]. Available:
http://www.school-for-champions.com/senses/hearing_direction.htm#.VmSYvXyrQ8o
[24]Paul A. Taylor, Alan Black, and Richard Caley. The Architechture of the festival speech synthesis system. In *The Third ESCA Workshop in Speech Synthesis,* pages 147-151, Jenolan Caves, Australia, 1998.

[25]Android Developers. “Accessibility”.
Internet:http://developer.android.com/design/patterns/accessibility.html
[26]Brian Edward Johnson, “The speed and accuracy of voice recognition software-assisted transcription versus the listen-and-type method: a research note”. *“Qualitative Research”,* vol. 11(1), pp. 91-97, Jan. 2011.
[27]Carl M. Rebman et. al. “Speech Recognition in the Human-Computer Interface”. *Information & Management,* vol. 40(6), pp. 509-519, June 2013.
[28]*How to Guide people with Sight Problems,* Royal National Institute for Blind, London, UK, RC- 226227, March
[29]Chih-Hung King, Tiffany L. Chen, Zhengqin Fan, Jonathan D. Glass, and Charls C. Kemp. Dusty:An Assistive Mobile Manipulator that Retrieves Dropped Objects for People with Motor Impairments, Disability and Rehabilitation: Assistive Technology, 2011