Design and development of Intelligent Mobile Robots (IMRs) for disaster mitigation and firefighting

Fifth Deliverable

Report submitted on 25th July 2017

PI: Dr. Muhammad Bilal Kadri

Co-PI: Dr. Tariq Mairaj Rasool Khan

Table of Contents

Chapter 1. Introduction

This report is the fifth deliverable for the National ICTR&D funded research project titled *"Design and development of Intelligent Mobile Robots (IMRs) for disaster mitigation and firefighting"*.

Chapter 2. Using ROS with the robot

We have studied, learned and used ROS to interface sensors on the robot for use in the IMR Lab work. However to harness maximum advantage from ROS and utilize its features for sensor transformations we have been having some difficulty implementing the features. In order to move forward with problems and issues related to ROS we selected to study similar robots and how they work in the ROS environment to get a better understanding for our own work.

2.1 Research on ROS Compatible Robots

There is a comprehensive list of ROS compatible robots on ROS official website with relevant links and related information. This list is composed of many different categories of robots including aerial, grounded and marine robots etc. Grounded robots are further categories into fixed, two, three and four-wheeled robots. We found few ROS compatible robots which are much similar in functionality to the IMR robot with respect to sensors and actuators, i.e. ultrasonic sensors, LIDAR, inertial sensors, GPS and cameras.

2.2 List of similar ROS compatible robots

1. TurtleBot 2
2. Mobile Research Platform (MRP2)
3. PMB - 2
4. Evarobot

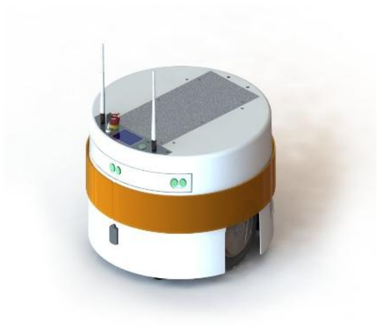
2.2.1 TurtleBot 2

TurtleBot is the official ROS compatible robot. Designed and Developed by willow Garage, who also have a critical role in development of ROS. It is open source hardware platform and mobile base. When powered by ROS software, TurtleBot can handle vision, localization, communication and mobility. It can autonomously move while avoiding obstacles and reaching its goal positions. Turtlebot 2 uses Kobuki Base which is a ROS compatible base and work as it's brain.
Link: http://learn.turtlebot.com/2015/02/01/1/

2.2.1 Mobile Research Platform (MRP2)

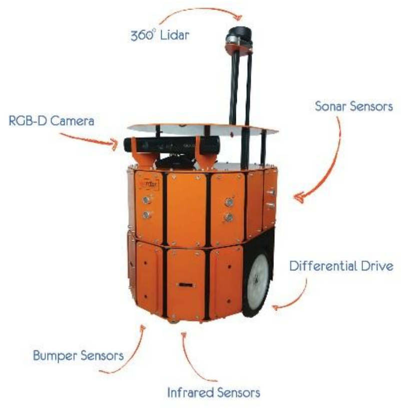
MRP2 is an easy to use mobile research robot developed by Milvus Robotics. The robot can easily do mapping with its already available navigation software and can position itself on the map with a few centimeters accuracy. MRP2, with its powerful hardware and software infrastructure, enables users to make developments quickly and comfortably.
Link: http://milvusrobotics.com/en/products/mrp2

2.2.2 PMB - 2



PMB-2 is an indoor mobile robot designed with minimum footprint and maximum payload in mind. Its primary sensors are a laser range finder and an IMU, but optional additions include ultrasound sensors and microphones. PMB-2 features built-in speakers so the robot can inform its surroundings via sound. This robot runs on ROS Hydro internally.
Link: http://wiki.ros.org/Robots/PMB-2

2.2.4 Evarobot



The Evarobot is a mobile robot platform built by Inovasyon Muhendislik. Usually used for indoors localization and navigation due its sensors limitations. Evarobot uses Raspberry Pi as its main processing brain and has almost entire software toolchain available as open source. Further, Evarobot also provides a ready to use Raspberry Pi OS Image.
Link: http://inovasyonmuhendislik.com/ and http://wiki.ros.org/Robots/Evarobot

We selected Evarobot for further and detailed analysis and study as this robot is the most similar to our IMR Lab robots and on top of that provides the software repositories as well. The use of Raspberry Pi, usage of LIDAR and use of UltraSonic sensors is very similar to our use case.

2.3 Evarobot Detailed analysis

Evarobot has most more similarities to IMR than other studied robots. As it uses LIDAR, ultrasonic sensors, camera, bumper sensors, inertial sensors while also having a differential drive mechanism similar to the IMR robots. The other reason for choosing this robot for further in-depth study is its open-source software and code. Tutorials for gazebo simulations are also provided and will be helpful because we don't have Evarobot hardware.

2.3.1 Choice of OS

Studying Evarobot, we came to know that Evarobot uses Raspbian as its operating system with ROS indigo installed. ROS Indigo is not latest version and there have been 2 newer versions but Indigo is an LTS version, meaning it has long term support fomr the community.

We are using Ubuntu Mate and ROS Kinetic with Raspberry Pi at IMR Lab. Although, for the time being it works, but Ubuntu is a sluggish running on the Raspberry Pi, while Raspbian is optimized for running on the pi. Our code is becoming more and more complex and requires fast and real time response, which is not possible with a slow operating system. The sluggish system also hinders and slows down the development process as well. On the other side, ROS does not officially support Raspbian but through source installation ROS can be installed. Furthermore, Evarobot also provides a Raspbian image file with ROS indigo installed which will be helpful as a starting point.

2.3.2 Network configuration

To communicate and operate Evarobot with PC it also requires a specific network configuration. It uses static IP network which is helpful as it means no time wasted in discovery of network and finding connected devices or guessing IP addresses.

On the other hand, the whenever IMR is connected to network it is assigned different IP. This results in us having to hunt for its IP from connection information which is a time wasting procedure. At the moment, we are working with dynamic IP network which is not suitable and optimal solution for working with multiple robot network.

2.3.3 Sensor Integration

Every sensor used in Evarobot has its own ROS package and a Kernel driver associated with it. This results in the sensors always available for processing and being available as Kernel devices, any programming language and framework can be used to read sensors allowing for easy programming and debugging.

Our method of interfacing hardware with Raspberry Pi and ROS is based on Python script files which are not handled as proper ROS nodes and hence some debugging and testing tools are not available to us. Our hardware is also connected to Pi via a separate Arduino and some sensors are directly connected but the Python is a bit slow in reading the hardware.

2.3.4 Simulation with URDF

The main reason for studying different ROS compatible robots is their open source simulation models and examples. Evarobot gives a comprehensive and multiple simulation examples for building map of unknown environment and navigation through it. By studying the Evarobot model, we came to know the importance of URDF file in simulating with a 3D robot model. In addition to simulation, one of the many benefits of ROS is Sensor TRANSFORMS, uptill now, we had not been able to utilize transforms for our sensors on IMR Robot, studying the Evarobot examples have cleared many issues and problems we were facing. We have now started work on creating a basic URDF file for our IMR robots, this will allow us to have the sensors values be transformed to proper coordinate frames according to robot body and the robot location. Another ROS tool is Robot State Publisher which is responsible for reading URDF files and transforming the sensor data accordingly.

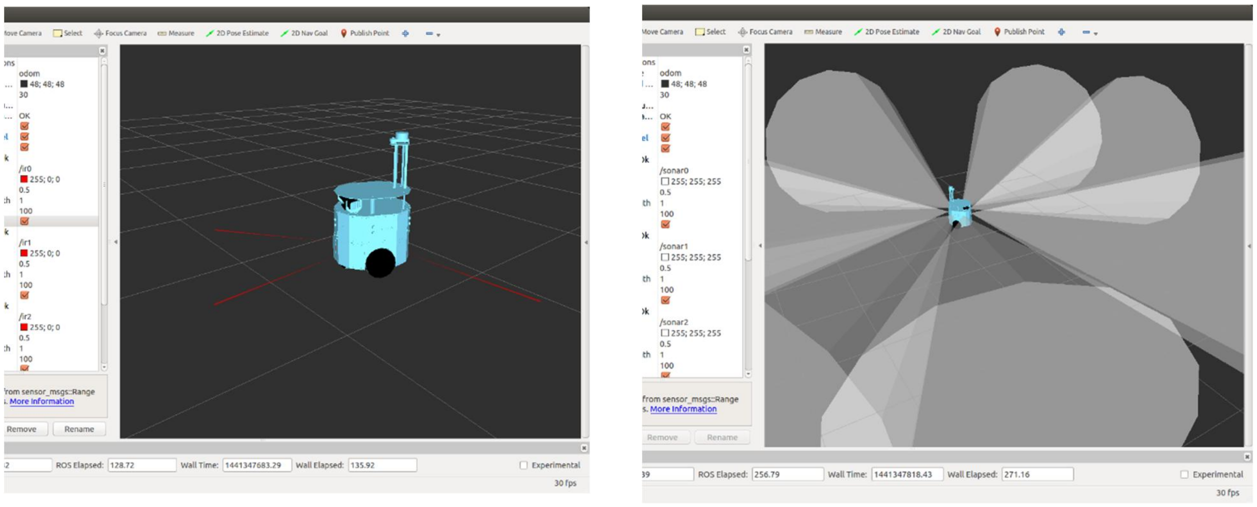


Figure 1 ROS visualization tool RVIZ

2.4 Improvements to our ROS Methodology

After in-depth analysis of Evarobot and other ROS compatible robots we found some flaws and improvements to be done in our ROS methodology like lacking of single launch file to start the operation of robot, robot sensor transforms, static IP network etc.

2.4.1 Work done uptill now

So far, we had successfully interfaced inertial sensors, ultrasonic sensors, camera and LIDAR with ROS separately, and published odometry results into ROS. All sensors are tested independently and produce satisfactory results. Independently testing of sensors does not require any robot transform and URDF file. For combined test of sensors require a robot state publisher and a URDF file which have robot description. Now, for future work it is very necessary that we start from robot state publisher and URDF file to combine those separately tested sensors.

2.4.2 Kernel Drivers

Most of the studied robots uses kernel drivers to interface with sensors which is more convenient method than separately invoking/calling every sensor within single script, and proper separate ROS packages will also be helpful.

2.4.3 Robot State Publisher and Robot URDF

As described above, for separate integration of sensors with ROS deos not require any robot transform to publish. But, as we are advancing into next phase of combined sensor integration we require a robot transform that tells us which component is attached at which position with respect to robot base.

2.4.4 C/C++ Language

In order to fully utilize Raspberry Pi and sensors hardware, we need to convert our code from Python to C, which will give us marked improvement in sensor performance and interfacing capability. Currently we are relying on Python code/libraries which are not very optimized nor are designed to work with multiple sensor scenario.

2.4.5 Testing with ROS indigo

Up till now, we are working with Ubuntu Mate and ROS Kinetic which is good enough for simple use. Unfortunately, we faced some problems as we are progressing in ROS like compilation errors, slow response due to sluggish operating system and many ROS packages are still only available for ROS Indigo. Due to this, we have decided to move from ROS Kinetic to ROS Indigo for compatibility and support reasons.

Chapter 3. Calibration of UT and Thermal Imaging Camera

3.1 UT Detection Testing:

We have tested the detection range of MaxSonar sensors. For this purpose, we have created an experimental site of 4x10 boxes in NDT lab as shown in figure.

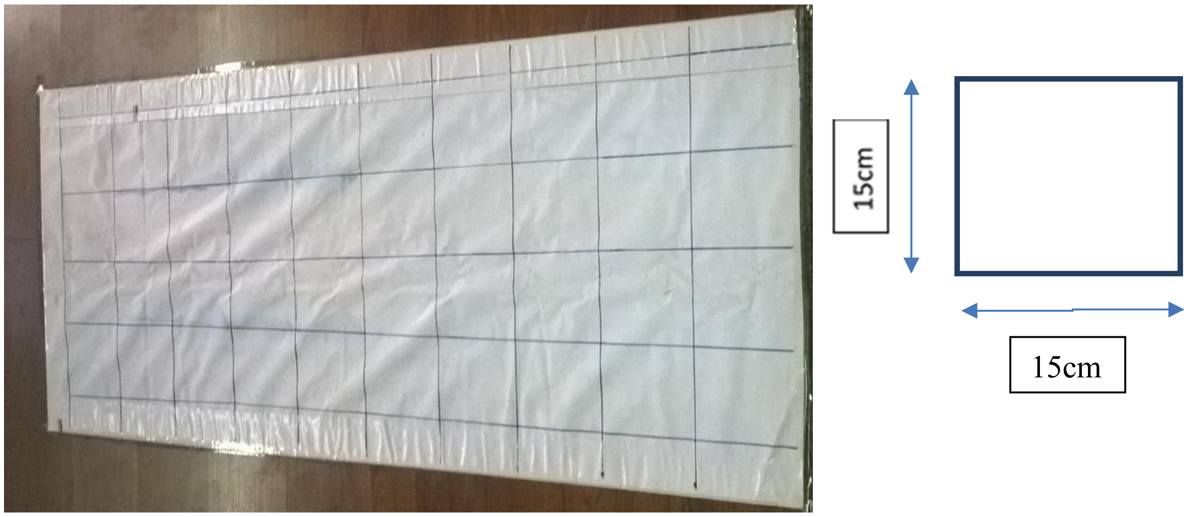


Figure 2: Experiment site made in NDT center

We tested the behavior of MaxSonar sensor (MB7380 LR) for:

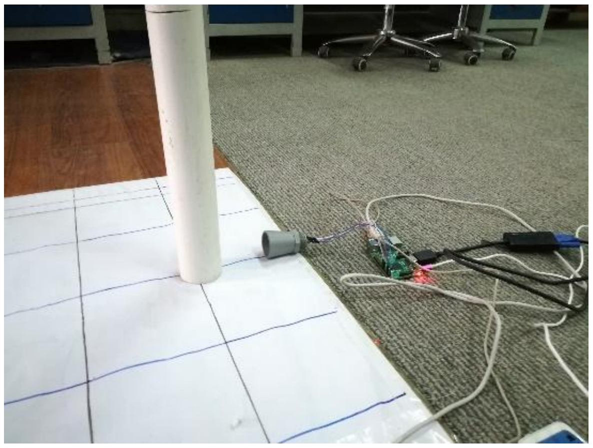
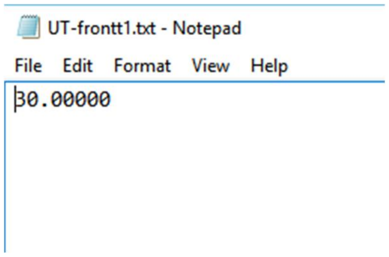
1) Objects of Different Size
2) Objects at Different Distances
3) Objects of Different Heights
4) Sensor at Different Altitudes

Objects or obstacles which we used for this test were;

- PVC Pipe with diameter of 2.5 inch and height of 24 inch
- Steel Pipe with diameter of 3.5 inch and height of 16.5 inch
- PVC Pipe with diameter of 6.75 inch and height of 9.5 inch
- 13 inch wide wooden board and height of 9.4 inch

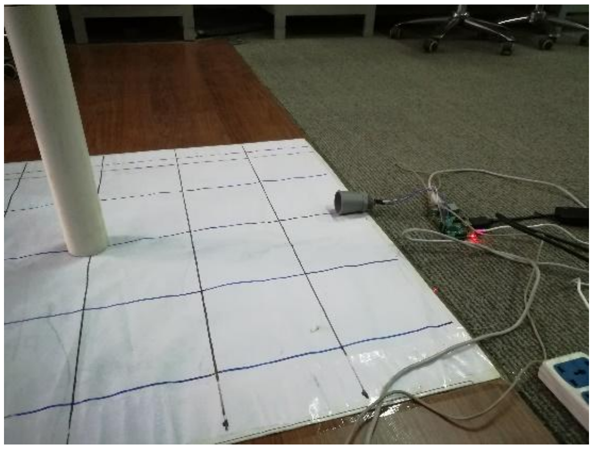
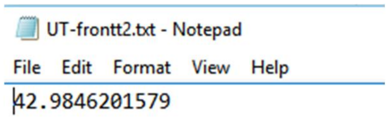
3.1.1 PVC Pipe with diameter of 2.5 inch

First set of tests were performed using a PVC Pipe of 2.5 inch. The pipe was moved over the grid from left towards right and the response of MaxSonar sensor was observed.

```
30.00000
```

Figure 3: PVC Pipe with center at distance of 6 inch from sensor

```
42.9846201579
```

Figure 4: PVC Pipe with center at distance of 18 inch from sensor

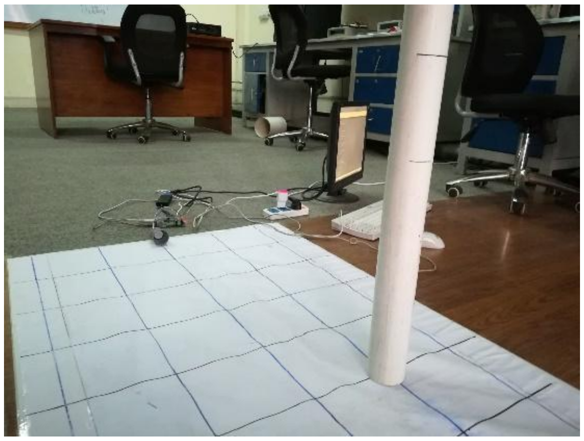
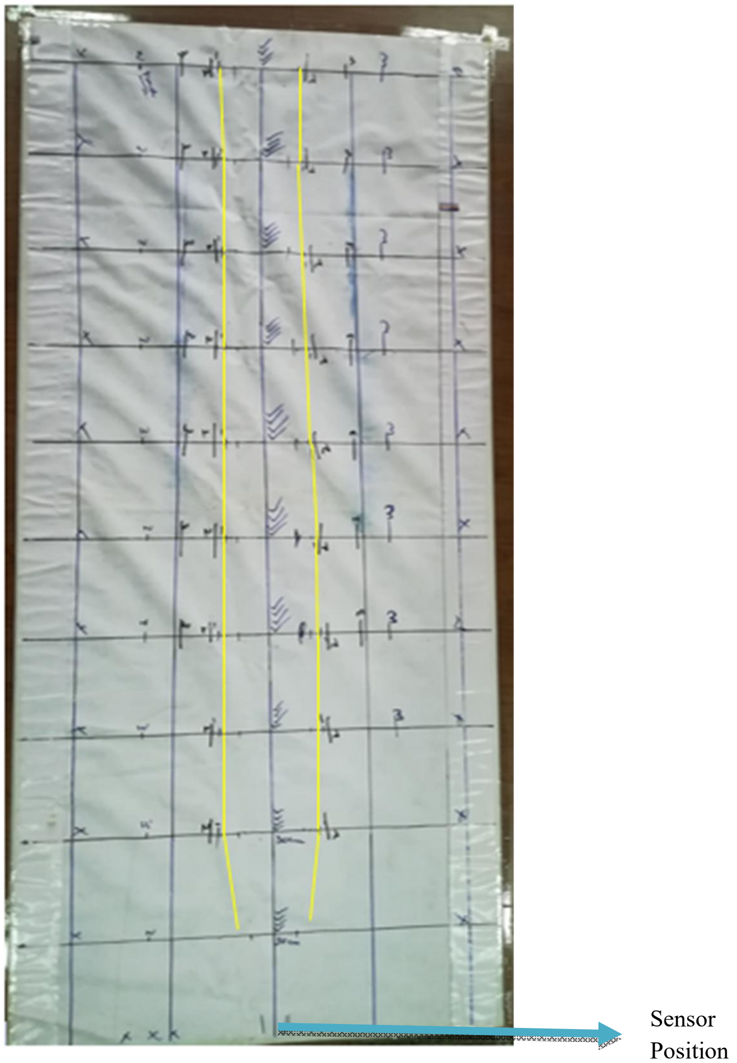


Figure 5: PVC Pipe placed at distance of 30 inch towards front and 6 inch towards left

The overall range of testing of steel pipe of 2.5 inch diameter and of 24 inch height is shown in below figure



3.1.2 Steel Pipe with diameter of 3.5 inch

Second obstacle used for testing was a steel pipe with diameter of 3.5 inches. This obstacle was also moved across the grid and the results were then noted. Some images representing this activity are attached under;

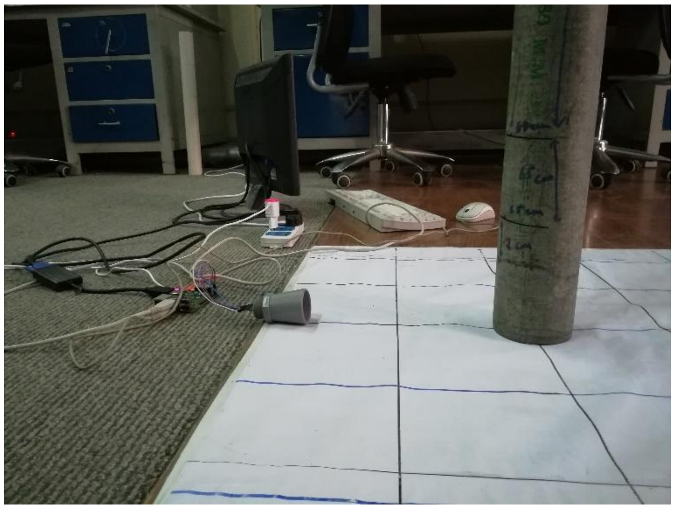


Figure 6: Steel pipe place at front at distance of 12 inch from center

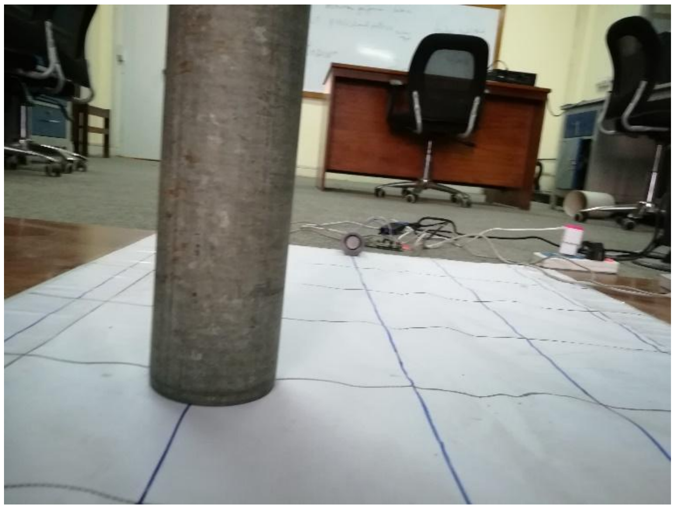


Figure 7: Steel pipe place distance of front (24 inch) and right (6 inch)

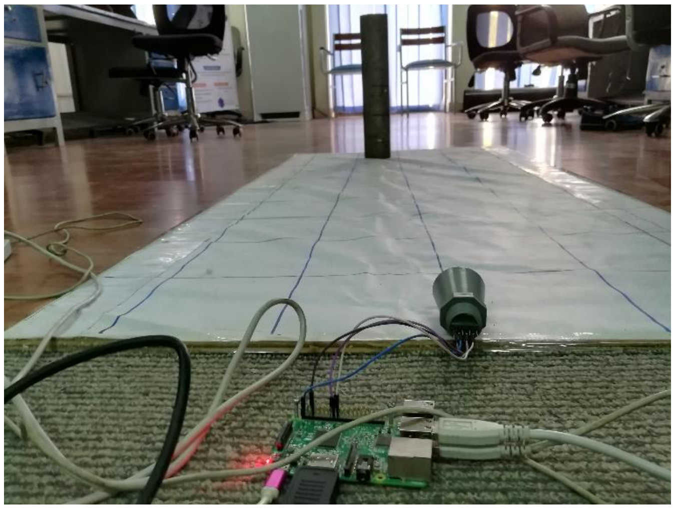
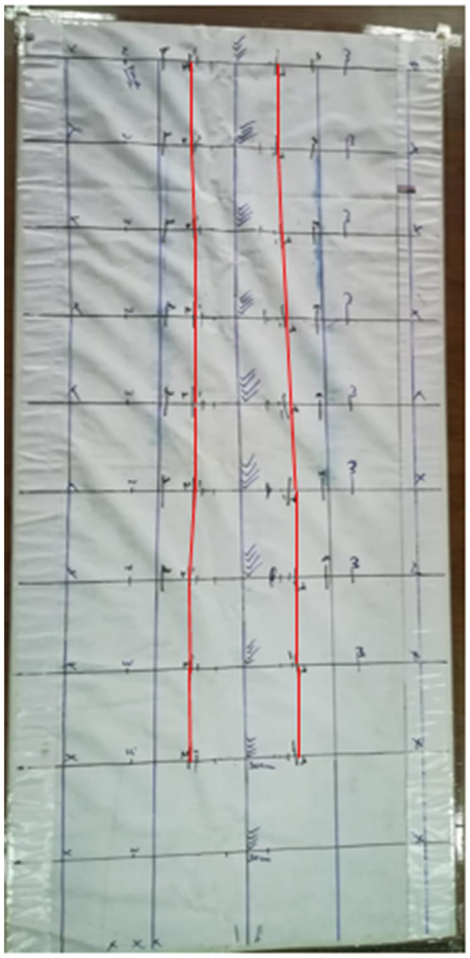
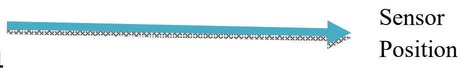


Figure 8: Steel pipe place at distance of front (54 inch) and right (3 inch)

The overall range of testing of detection for a steel pipe of 2.5 inch diameter and of 24 inch height is shown in below figure;





3.1.3 PVC Pipe with diameter of 6.75 inch

Third obstacle which we used was a PVC pipe with diameter of 6.75 inch. Same testing procedure was used for this obstacle as well.

A few images of this are also attached under;

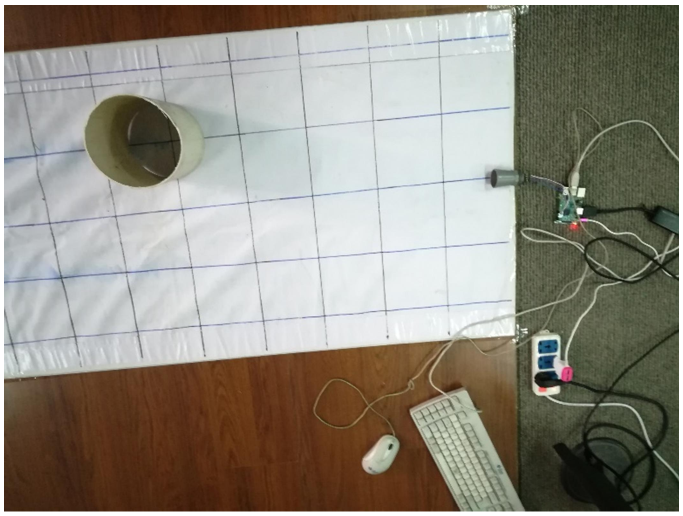


Figure 9: PVC Pipe of 6.75 inch placed at front (30 inch) and left (6 inch)

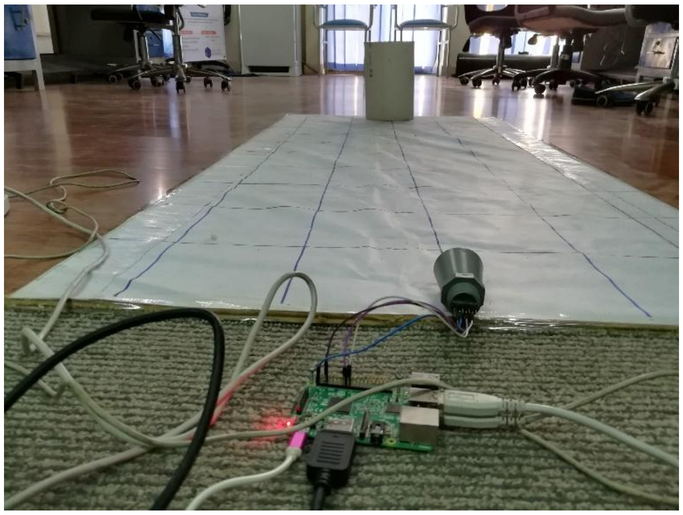
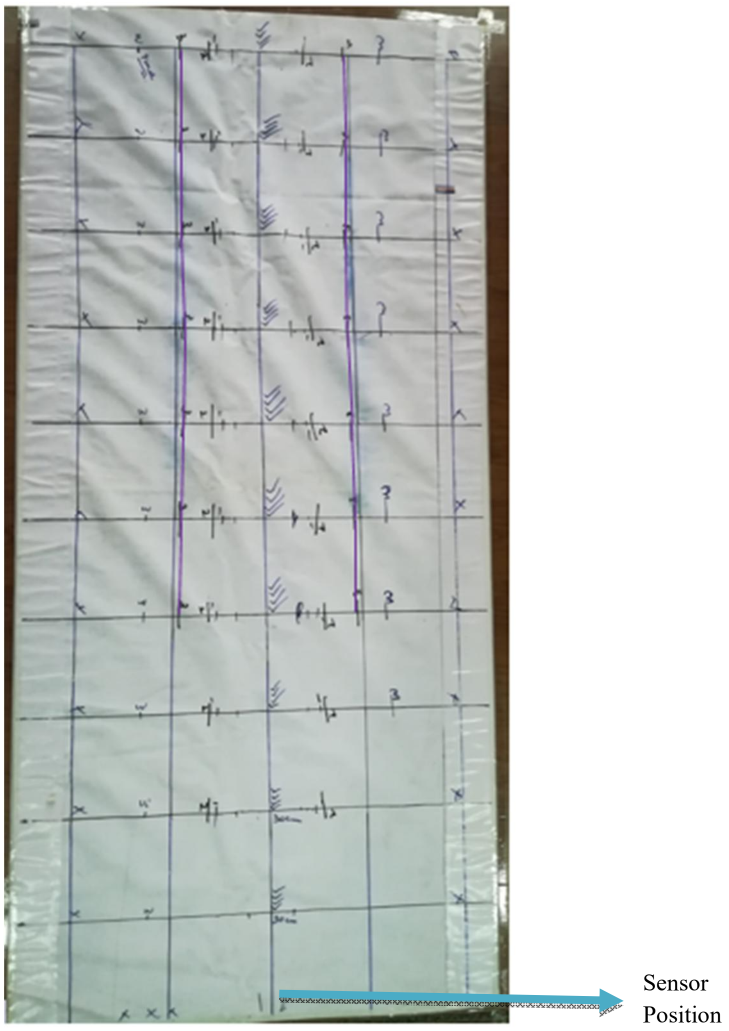


Figure 10: PVC Pipe of 6.75 inch diameter place at distance of 60 inch towards front

The overall range of testing of steel pipe of 2.5 inch diameter and of 24 inch height is shown in below figure



3.1.4 13 inch wide wooden board

In last scenario, a 13 inch wide wooden board was used as an obstacle. This wooden was also placed at all possible location on the board and results were noted. Images depicting the conduct of this test are also attached under;

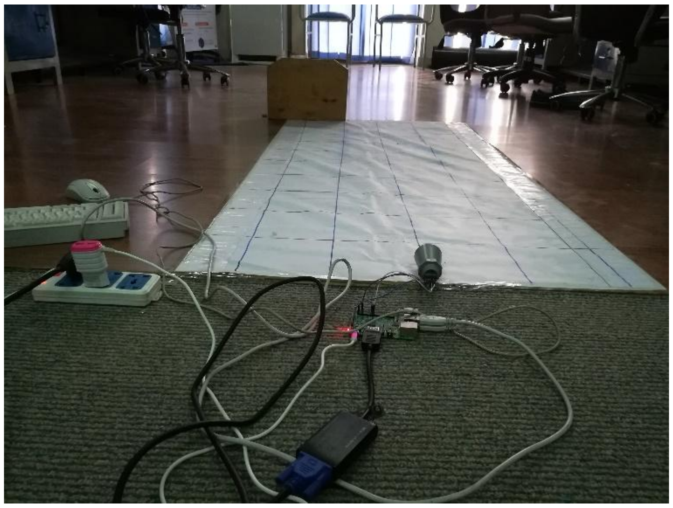


Figure 11: 13 inch wide wooden board placed at front (60 inch) and left (12 inch from center)

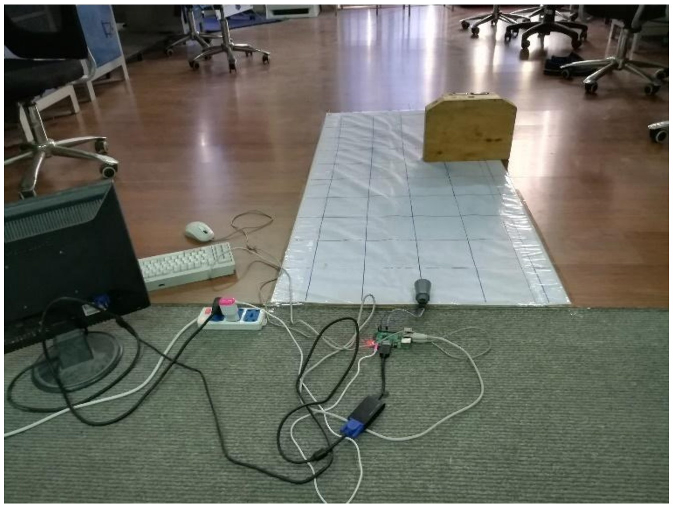


Figure 12: 13 inch wide wooden board placed at front (30 inch) and right (9 inch from center)

The overall range of testing of steel pipe of 2.5 inch diameter and of 24 inch height is shown in below figure

3.1.5 Sensor at different altitudes

The height of sensor was then changed from the ground by the use of the chair and all the experiments were then repeated again. Multiple obstacles were also used to test the performance of MaxSonar sensor. Results of maxsonar sensor were satisfactory with multiple obstacle in its range.

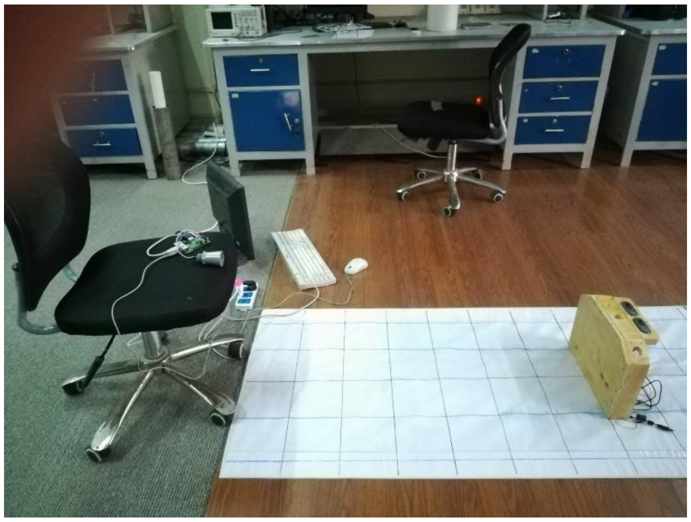


Figure 13: Sensor placed at different hieght

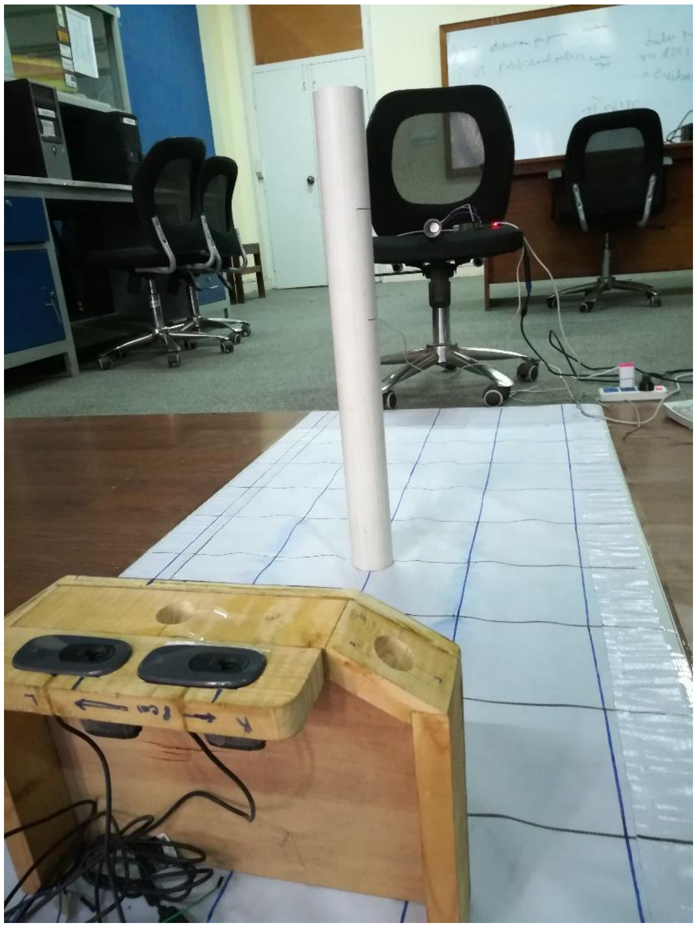


Figure 14: 2 obstacles in range of Sonar sensor

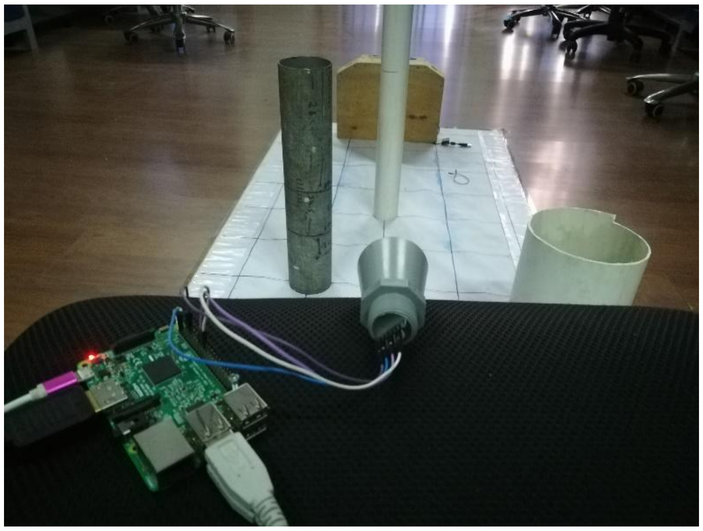


Figure 15: 4 obstacles in range of Sonar sensor

3.1.6 Results:

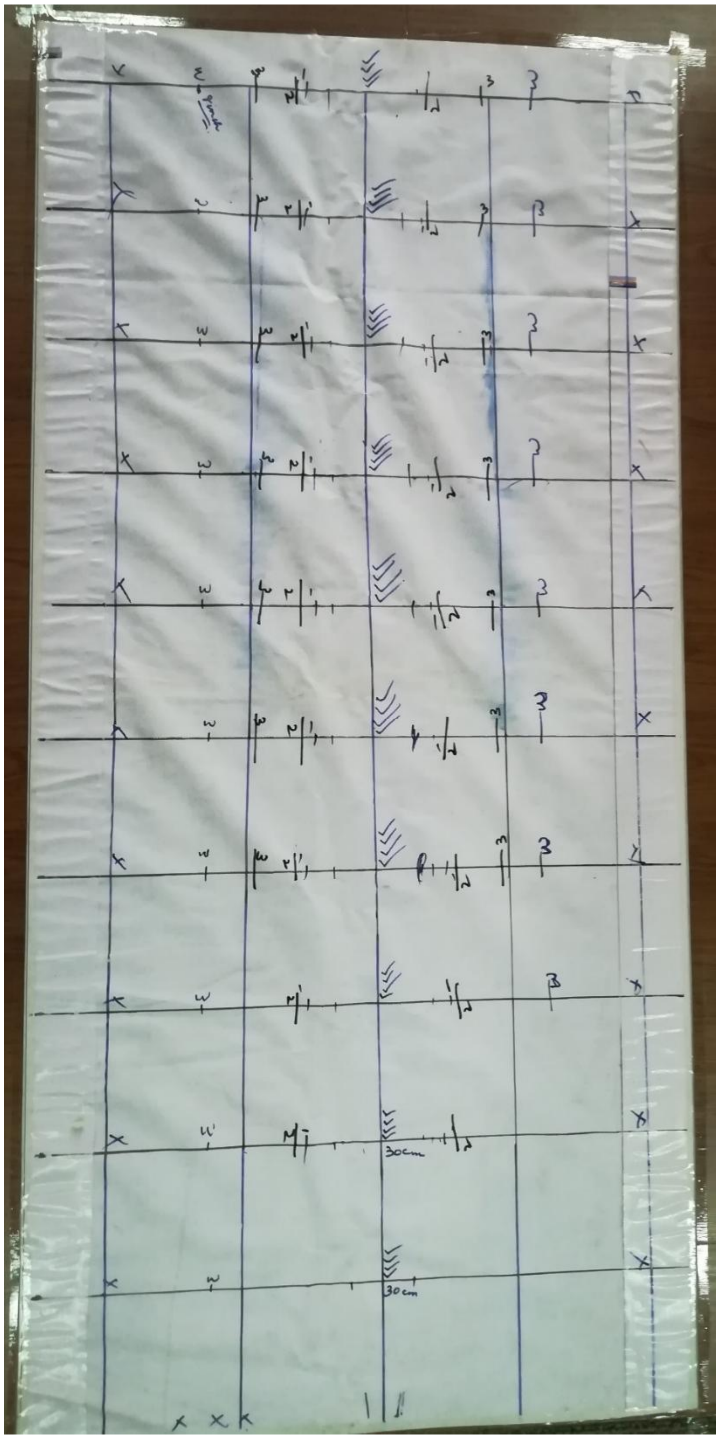


Figure 16: Max detection range from different obstacles

These maximum detection points from different obstacles were then joint to depict the actual detection range. As shown in the figure below, where;

- Pink lines represent the detection range of 13 inch wide wooden board
- Purple lines represent the detection range of 6.75 inch diameter PVC Pipe
- Red lines represent the detection range of 3.5 inch diameter steel pipe
- Yellow lines represent the detection range of 2.5 inch diameter PVC pipe
- Green lines represent the actual detection range of MaxSonar sensor based on all the testing performed.

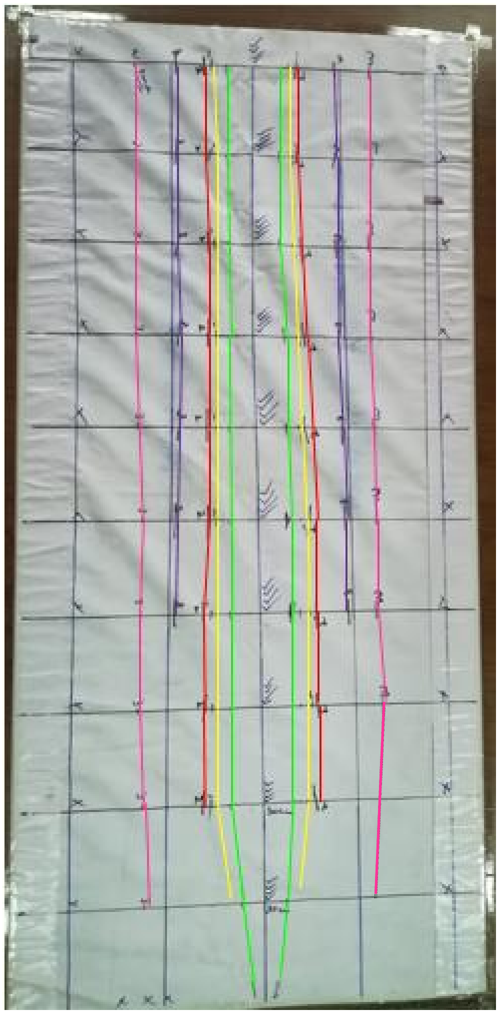


Figure 17: detection range of different Obstacles in front of MaxSonar sensor

3.1.7 Conclusion:

Based on the testing performed and the results which we got when we placed different obstacles in front of the sensor. Following things can be concluded;

- It can detect the objects 1mm a part.
- It can measure a minimum distance of 30cm. Therefore the objects near to 30cm are typically reported as 30cm.
- The HRXL-MaxSonar-WR sensor line compensates for target size differences. This means that, provided an object is large enough to be detected, the sensor will report the same distance, regardless of target size.
- Smaller targets can have additional detection noise that may limit this feature. In addition, targets with small or rounded surfaces may have an apparent distance that is slightly farther, where the distance reported may be a composite of the sensed object(s).

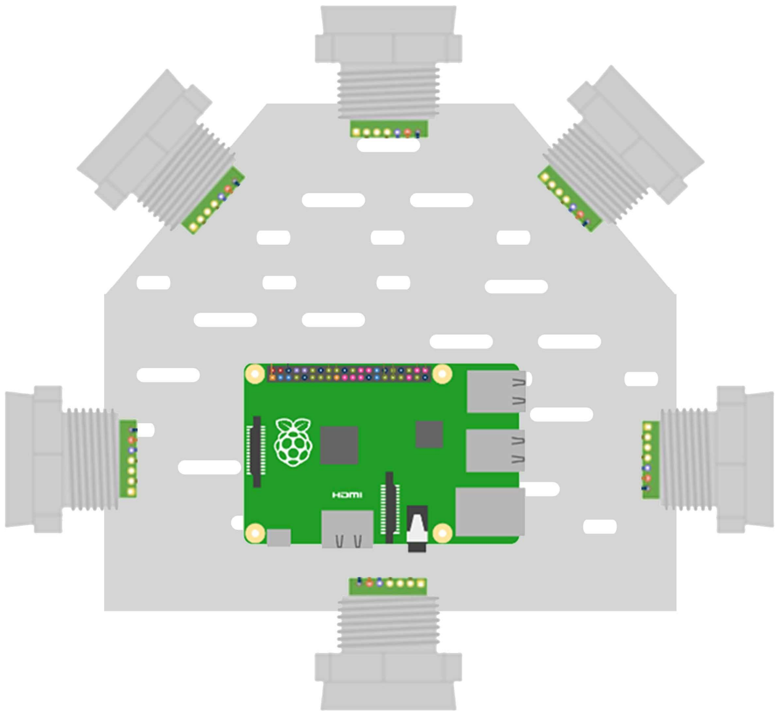
Chapter 4. Fusion of UT and LIDAR in ROS

4.1 UT Data in ROS

4.1.1 UT integration with raspberry pi

After proper calibration of the data of single MaxSonar sensor on raspberry pi, we then interfaced six MaxSonar sensors from maxbotix on a single raspberry pi and obtained there data simultaneously in real-time.

The orientation of these sensors on the robot head is attached under;



Sensors Orientation on robot

4.1.2 UT data Publish to ROS network

The data of all these ultrasonic sensors (six sensors on single robot) was then published in the ROS network. For publishing the MaxSonar data in ROS network, we used three different machines, two of them were Linux operating machines with Robotics Operating System installed on them (One PC and one raspberry pi) while the remaining system was a normal windows operating system with MATLAB 16.0 installed on them. We integrated all six ultrasonic sensors with one of the Linux operating machine (raspberry pi with ROS kinetic installed on it).

We then went into the catkin workspace which we created in raspberry pi for building and running the already build packages. For this, we used the following command;

- ⇨ $cd ~/catkin_ws

A bash file was then created to enter to ROS network using the under mentioned command;

- ⇨ Source devel/setup.bash

To enter into the bash file containing folder;

- ⇨ Roscd

The data all ultrasonic sensors was then published in the ROS network using under mentioned command;

- ⇨ Rosrun ultrasonic_ros

4.1.3 UT data subscribed in ROS network

To subscribe to the data of ultrasonic sensors which we published from other machine (import the data from robot) in real time, the remaining Linux operating machine was used which was already inside the ROS network.

For this purpose, at first we went through all the topics that are being published in the ROS network. Following command is used for this purpose;

- ⇨ Rostopic list

From the list of all these topics, user can access any of the data being published in the ROS network in real time. To import the data from raspberry pi (robot) to base station, we used following command;

- ⇨ Rostopic echo /topic_name

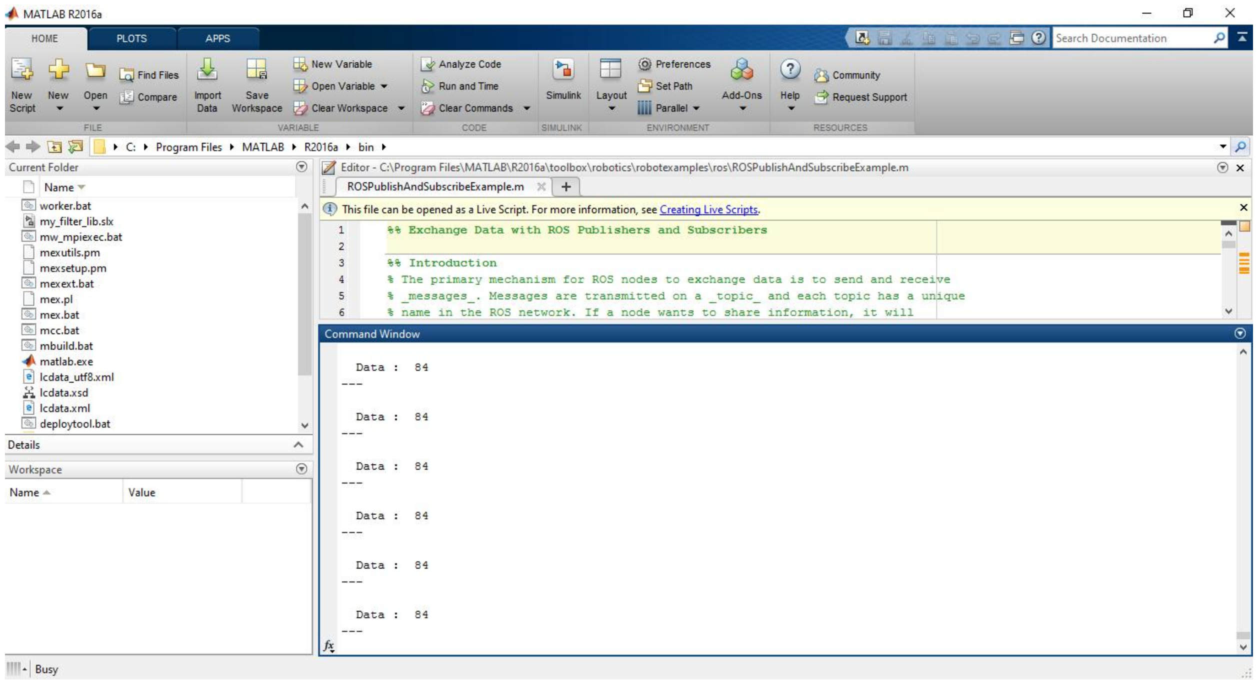
Where, the topic name will be replace the name of any topic which user want to access from the list of topics being published. As in this case, it is;

- ⇨ Rostopic echo ultrasonic_ros

4.1.4 UT data accessed in MATLAB

This published data was then brought into MATLAB in real time by running a Node Host at MATLAB using MATLAB’s Robotics Toolbox.

The commands which we used for this purpose were the same as mentioned above in the subscriber topic. Attached following the screenshot of the data which we got in MATLAB in real time.


MATLAB R2016a
HOME
PLOTS
APPS
Current Folder
Editor - C:\Program Files\MATLAB\R2016a\toolbox\robotics\robotexamples\ros\ROSPublishAndSubscribeExample.m
ROSPublishAndSubscribeExample.m
Command Window
Busy

This file can be opened as a Live Script. For more information, see Creating Live Scripts.

```
%% Exchange Data with ROS Publishers and Subscribers

%% Introduction
% The primary mechanism for ROS nodes to exchange data is to send and receive
% _messages_. Messages are transmitted on a _topic_ and each topic has a unique
% name in the ROS network. If a node wants to share information, it will
```

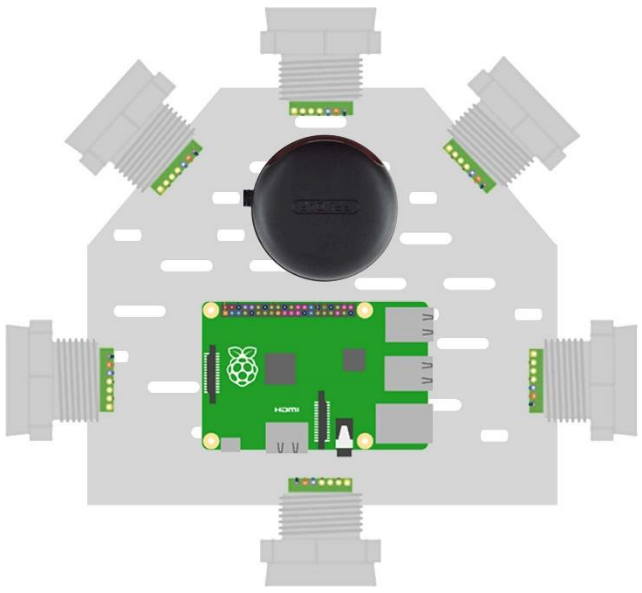
```
  Data :  84
---

  Data :  84
---

  Data :  84
---

  Data :  84
---

  Data :  84
---

  Data :  84
---
```

Real time Ultrasonic data in MATLAB's command window

4.2 LiDAR data in ROS

4.2.1 LiDAR location on Robot

LiDAR was mounted at the top of the front location of robot and was connected to raspberry pi through USB port.



LiDAR and ultrasonic sensors location on robot

4.2.2 LiDAR data published to ROS network

The data of 2D point cloud which we got from rplidar was then published into the ROS network. For this purpose we used three different machines, two of them with Robotics Operating System installed on them while the remaining system was a normal windows operating system with MATLAB 16.0 installed on them. We integrated rplidar with one of the Linux operating machine (raspberry pi with ROS kinetic installed on it).

We then went into the catkin workspace which we created in raspberry pi for building and running the already build packages. For this, the following command was used;

```
⇨ $cd ~/catkin_ws
```

A bash file was then created to enter to ROS network using the under mentioned command;

```
⇨ Source devel/setup.bash
```

To enter into the bash file containing folder;

```
⇨ Roscd
```

Rplidar node was then launched to run in ROS network;

```
⇨ Roslaunch rplidar_ros rplidar.launch
```

The data rplidar was then published in the ROS network using under mentioned command;

```
⇨ Rosrun rplidar_ros rplidarNodeClient
```

4.2.3 LiDAR data subscribed from ROS network

To subscribe this data (import the data from robot) in real time, the remaining Linux operating machine was used which was already inside the ROS network.

For this purpose, at first we went through all the topics that are being published in the ROS network. Following command is used for this purpose;

⇨ Rostopic list

From the list of all the user can access any of the data being published in the ROS network in real time. For this purpose, following command is generally used;

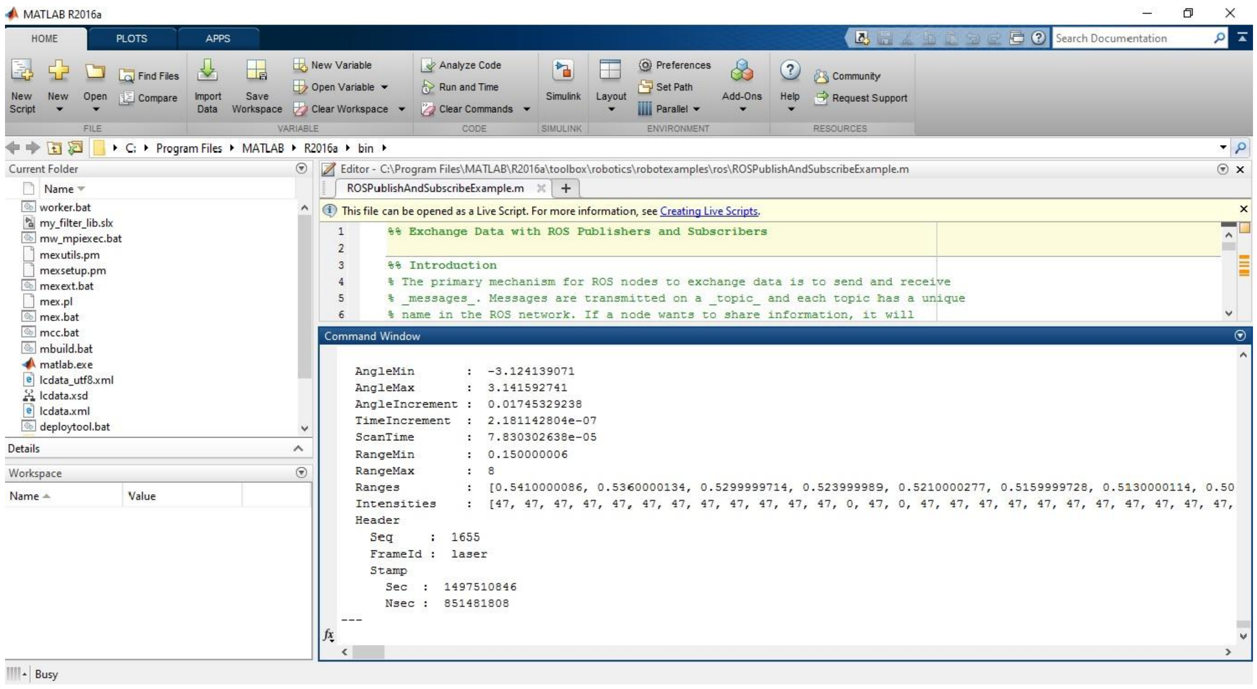
⇨ Rostopic echo /topic_name

Where, the topic name will be replace the name of any topic which user want to access from the list of topics being published.

4.2.4 LiDAR data imported to MATLAB command window

This published data was then brought into MATLAB in real time by running a Node Host at MATLAB using MATLAB's Robotics Toolbox.

The commands which we used for this purpose were the same as mentioned above. Attached following the screenshot of the data which we got in MATLAB in real time.



```
Editor - C:\Program Files\MATLAB\R2016a\toolbox\robotics\robotexamples\ros\ROSPublishAndSubscribeExample.m
%% Exchange Data with ROS Publishers and Subscribers

%% Introduction
% The primary mechanism for ROS nodes to exchange data is to send and receive
% _messages_. Messages are transmitted on a _topic_ and each topic has a unique
% name in the ROS network. If a node wants to share information, it will
```

```
  AngleMin       :  -3.124139071
  AngleMax       :  3.141592741
  AngleIncrement :  0.01745329238
  TimeIncrement  :  2.181142804e-07
  ScanTime       :  7.830302638e-05
  RangeMin       :  0.150000006
  RangeMax       :  8
  Ranges         :  [0.5410000086, 0.5360000134, 0.5299999714, 0.523999989, 0.5210000277, 0.5159999728, 0.5130000114, 0.50
  Intensities    :  [47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 0, 47, 0, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47,
  Header
    Seq     :  1655
    FrameId :  laser
    Stamp
      Sec  :  1497510846
      Nsec :  851481808
---
```

Real time LiDAR inside the command window of MATLAB

4.3 Data Fusion in MATLAB for Map plotting

The data which we have in MATLAB in real time will be fused to generate a better and more accurate graph can then be achieved. As the increase in temperature decreases the accuracy of LiDAR decreases while on the other hand, ultrasonic sensors are relatively slower as compared to LiDAR. Therefore system should generate more accurate graphs in lesser possible time.

We have got the data of MaxSonar ultrasonic sensors and LiDAR data in MATLAB in real time. We are now working on making appropriate algorithms for the fusion of both sensor’s data.

Chapter 5. PCB Designs

This chapter discusses the work done on the electronics for robots made at IMR lab and the electronics work that is to be done to get the final robots up and running. In order to build up to the final robots, there were several iterations and designs that were made, upgraded, discarded and re-made to satisfy the necessary requirements of the robots and the requirements of the project. Most importantly, the IMR robots are not a one step process but that these have evolved from simple lab based robots MUSAFIR bots to a bigger and more robust sensor platform MUSAFIR v2 and then finally the IMR Robots which are currently being made are full size and use multiple 90W motors.

5.1 MUSAFIR, MUSAFIR v2

The first robots to come out of the IMR Lab was the MUSAFIR project, this was a great achievement for the Lab as majority of the circuitry, design and code was written from scratch and a single MUSAFIR Robot, which costed about 10 times less than its similar counterparts in other parts of the world, allowed us to gauge and test several algorithms and gave us insight into how multiple robots can be controlled in a dynamic and unstructured environment.

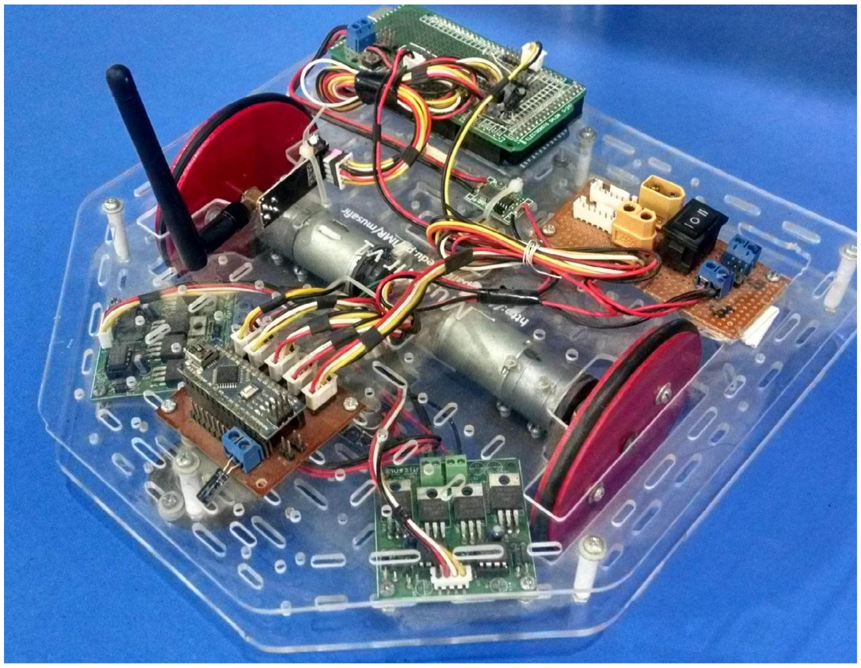


Figure 18 MUSAFIR Robot version 1.0

One of the downsides of MUSAFIR was that it could not move on uneven terrain and not even on road or cemented flooring, this severely limited our testing capabilities and we went back to the drawing board to design a better variant which would be able to move on cement and road terrain. The MUSAFIR v2 was born. MUSAFIR v2 was also the robot which utilized ROS and had a single board computer Raspberry Pi 3 on board for most of the processing. Using the MUSAFIR v2 we were able to generate maps of unknown environments using the LIDAR Sensor and the capabilities of the Raspberry Pi computer and the ROS framework.

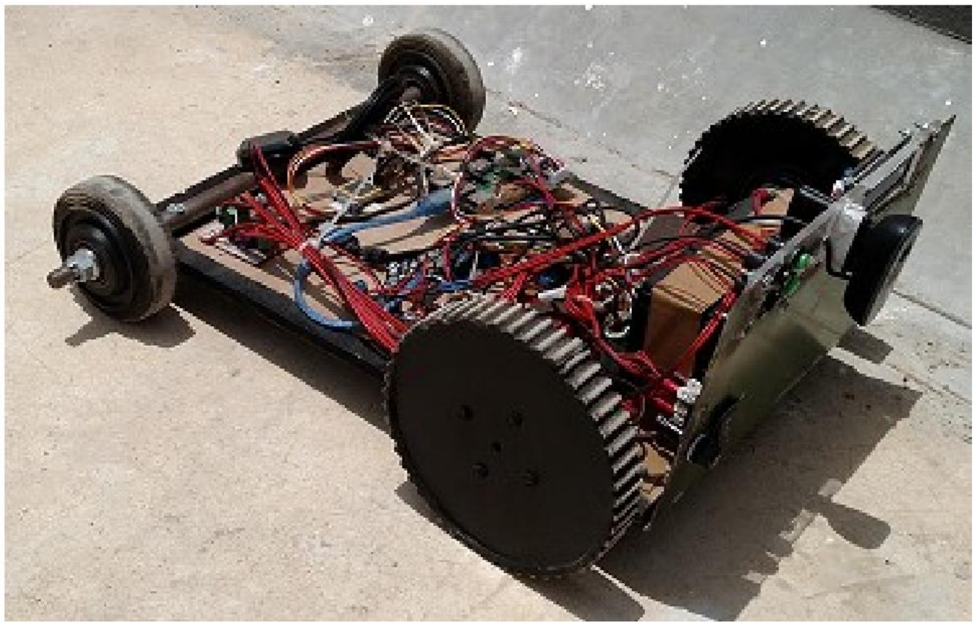


Figure 19 MUSAFIR version 2.0

5.2 MUSAFIR Robot Design

The MUSAFIR robots used completely designed and fabricated in-house with observing similar robots. This section discusses the design decisions that were taken and how those affected the outcomes, the issues we faced and what solutions were proposed and taken to counter the problems faced.

5.3 MUSAFIR v1

MUSAFIR v1 was built with plastic acrylic sheets, LASER Cut to our required size and shape. The motors used had built-in encoders which allowed us to do odometery whereas onboard IMU provided acceleration and velocity readings. MUSAFIR v1 had distributed control, with two micro-controller boards on the robot, one for odometery and velocity control which implemented low level PID control for both motors and the other controller which communicates with the main computer using a 2.4GHz transceiver and also uses an SD Card for data logging.

The Electronics onboard MUSAFIR v1 is listed below:

- Lipo Battery, 4S, 5200mAh
- Buck Converters – 5V for Controllers and 12V for Motor Drivers
- Motor H-Bridge v1
- 12V DC Geared Motor with Encoder (1500CPR)
- Motor Controller v1 – Arduino Nano interface board
- Arduino Mega – for communication
- MPU-9150 – 9-Axis IMU

- nRF24L01+ 2.4GHz Transceiver
- SD Card Module

MUSAFIR v1 was our first implementation and we managed to create 2 identical MUSAFIR v1 robots which were controlled by a computer using a USB to Serial 2.4GHz Transceiver Bridge that we had made, using Arduino and nRF module.

5.3.1 MUSAFIR v1 Issues

While working with MUSAFIR v1 we realized several things that needed to either be replaced or improved upon, these are listed below.

- Having 2 micro controllers on board and managing code of both was a very tedious task. Solution: Have 1 micro-controller on the robot to do all low level tasks.
- nRF24L01+ transceivers worked most of the time, but when they didn't much time was wasted. Also there was no proper mechanism of confirming ACTIVE LINK, error checking and acknowledgements. Implementing those would have taken much time. Solution: Replace nRF with a better Wireless system.
- Arduino MEGA although having enough memory for code storage and RAM, but it still is ultimately a single micro-controller and giving it all the higher level tasks of communication, data storage, IMU filtering was a huge task on its own and although we could do all these individually, doing all these things simultaneously was not possible for us to achieve in the given time and with required accuracy and speed. Solution: Use a better processor, managed system, like a single board computer, Raspberry Pi for higher level processing, communication and logging.

5.4 MUSAFIR v1.5

MUSAFIR v1.5 used the same bodies/chassis of the MUSAFIR v1, basically MUSAFIR v1s were repurposed to make v1.5. This variant resolved the issued faced with v1 and allowed us to start and implement ROS on the robots.

The main difference in the onboard electronics of MUSAFIR v1.5 was the replacement of Arduino MEGA with Raspberry Pi 3, as Raspberry Pi 3 has built-in WiFi, we also had no longer need of the 2.4GHz Transceiver modules.

5.4.1 MUSAFIR v1.5 Issues

As with any research based effort, iterations provide improvements over previous versions but new problems and issues are discovered and noted for further improvements in the work. Issues with MUSAFIR v1.5 are discussed below and some solutions are suggested.

- WiFi connectivity issues and IP related matters, connecting to Raspberry Pi when there is no display or keyboard/mouse attached, i.e. HEADLESS MDOE, requires that IP and network to which Raspberry Pi is connected to is known in advance. Solution: Have a python script check the connected WiFi Network and connect to the preferred network if needed, the script also would show the information on an LCD connected to Raspberry Pi.

- WiFi Connectivity allowed us to use ROS and ROS allowed publishing and subscribing of the data coming in from sensors and encoders in a very reliable and robust manner. However connectivity with MATLAB remained an issue and getting the ROS data in MATLAB.
  Solution: Use the Robotics ToolBox in MATLAB 2016b or 2017 which includes ability to create ROS nodes which connect with a ROS Master system and can get the data being published by other ROS nodes in the network.
- Arduino and ROS Compatibility issues, as we were using an Arduino Nano for the low level control of motors, we were unable to completely implement ROS, down to the Arduino level as Arduino Nano does not have enough space in it. Solution: Use an intermediate layer for data transmission between Arduino and and Raspberry Pi. OR use an Arduino which can support ROS, like Arduino Due or Mega.

5.5 MUSAFIR v2

MUSAFIR v2 as seen in the pictures before, was a complete remake of the robot with better tyres for gripping on rough surfaces and a bigger body accommodate all the new sensor modules and the circuitry. The electronics on MUSAFIR v2 is mostly same as MUSAFIR v1.5, with addition of 2 VGA Cameras at the front and also space for some Ultra Sonic Sensors to be added later to the front of the Robot and a LIDAR on top.

The Electronics onboard MUSAFIR v2 is listed below:

- Sealed Lead Acid Battery, 12V, 7Ah
- Buck Converters – 5V for Raspberry Pi and LIDAR
- Motor H-Bridge v2
- 12V DC Geared Motor with Encoder (1500CPR)
- Motor Controller v2 – Arduino Nano interface board
- MPU-9150 – 9-Axis IMU
- RP LIDAR A2
- 2x LogiTech USB Cameras
- TPLINK WiFi Module
- uBlox Neo 6m GPS Module
- I2C 16x2 Text LCD Module
- Analog Battery Voltage Monitoring Circuit
- Power/Switch/Charger Circuitry

The MUSAFIR v2 body was made of sheet metal, while the base was made of wood, wheels being made out of Teflon. The motors used for MUSAFIR v2 had a very high gear ratio along with encoders, resulting in slow but precise movement of the robot. We tested the MUSAFIR v2 in the field, outside of the LAB environment by driving the robot on cement, asphalt and road surface where the performance of the robot and the wheels was satisfactory (after some modifications were made, as mentioned below).

5.5.1 MUSAFIR v2 Issues

As MUSAFIR .v2 was more of a mechanical re-design than an electrical/electronic one, most of the issues and problems we observed were based on the mechanical work. The differential drive mechanism was a bit flawed, the robot motors had enough power, but due to being light, the wheels weren't getting proper grip on the floor and there were issues with working with the onboard Electronics.

- We had designed the robot motor placement to work as differential drive robot, however, unlike MUSAFIR v1 and v1.5 where the drive motors were in the center and the caster wheels at both ends of the robot, this time, we had the drive motors at the back and at the front side, 2 free wheels on each side of the robot. These free wheels not connected to the motor resulting in additional friction skidding of the robot. Solution: Have both front and back wheels synced using a chain mechanism in the next iteration of the robot, which is what we have also observed in all other robot designs on the Internet which use differential skid drive.
- Weight of the robot was an issue and in a way that we had not anticipated, our robot wheels were not able to form proper grip on the floor as the weight was very less, we tested by adding dead weight at different places on the robot and observed the changes, which were positive making the robot move and turn properly. Solution: We added a Dead Weight at the back of the robot, just in between the drive wheels/motors, the weight was made from a metal piece, MS, solid rectangular piece.
- As all the electronics was inside the metal chassis of the robot, we had to use long connecting wires for sensors which were placed in the robot upper body going to the base, and this was also needed for power switch, indicator LEDs and the charging port. Solution: We cut the metal cover in 2 pieces to allow for easy access to the electronics, this way, we did not need to replace the entire metal cover to work on the electronics.
- Robot Status is something that was needed time and again, we needed to know the IP address, if the robot was even currently working or stuck in some processing loop and also needed to know the battery status. Solution: Add a Text LCD and indicator LEDs which showed us the current status of the robot and the updating of the LCD meant that the Raspberry Pi was working as well.

5.6 MUSAFIR Robot Electronics Overview

The MUSAFIR robots used Electronics that was designed in house and then the circuit designs were sent to China for PCB Fabrication. This section will discuss the Electronics used on MUSAFIR v1, v1.5 and MUSAFIR v2, very briefly as these have already been discussed in previous reports.

- Sensor Shield
- Motor Controller v1
- Motor Controller v2
- Motor Driver – H-Bridge v1
- Motor Driver – H-Bridge v2

5.7 Components of Robot

Level 0 - Robot Base: Mechanical, ElectroMechanical, Electrical, Core Safety

- Robot Base
- Wheels
- Drive Motors + Encoders
- Bumper Switches
- Front Head lights
- Battery
- DC DC Boost Converter
- Connector Board - Bumper/Encoder/Front Head Light
- Power Panel
    - Battery Terminal
    - Switch - ON-OFF-Charge
    - On, Battery Good/Bad, Charge Indicator
    - Fuse
    - E-Stop Switch
    - Battery Brown Out Detector

Level 0 to Level 1 - INTERFACE

- Electrical Power Wires
    - 4x Motor Wires
    - 2x Battery Terminal, 12V
    - 2x Boost Converter Output, 24V
    - 2x Front Headlight Wires
- Electronics Signal wires
    - 4x Encoder Interface
    - 2x 3.3V Supply / GND
    - 2x Bumper Switches

Level 1 - Motor Control: Electrical, Electronics, Odometery, Velocity Control

- Motor Controller
  Module: Arduino Mega
  https://www.arduino.cc/en/Main/arduinoBoardMega
    - Odometry
        - Dead Reckoning using Encoder Counts
        - Raw Encoder Count Interface
    - Control Methodology
        - Velocity based via PID
        - PWM Based
    - Motor Driver Current/Temperature Sensing/Threshold
    - Motor Driver Control Signals
    - HeadLights Control

    - Bumper Sensor Interface
- Motor Driver
    - H-Bridge
      Mosfet: IRLB8743
      https://goo.gl/Rhn5jc
    - Thermal Sensor
      IC: LM35
      http://www.ti.com/product/LM35
    - Current Sensor
      IC: ACS712 20a
      http://www.allegromicro.com/en/Products/Current-Sensor-ICs/Zero-To-Fifty-Amp-Integrated-Conductor-Sensor-ICs/ACS712.aspx
    - Active Cooling

Level 1 to Level 2 - INTERFACE

- Electrical Power Wires
    - 1x Battery Terminal, 12V
- Interface for Motor Controller - via USB

Level 2 - Sensor Head: Core Process Electronics, Sensors

- Raspberry Pi 3
  https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

- Sensor Interface PCB
    - RTC
      IC:DS1307
      https://www.maximintegrated.com/en/products/digital/real-time-clocks/DS1307.html
    - Battery Level Monitoring
      IC: LM324 (Op-amp based)
    - 8-Bit MicroController for Analog/Digital IO via I2C
      MicroController: Atmega8
      http://www.microchip.com/wwwproducts/en/ATmega8
    - 3x Buck Converters - 12V to 5V
        - For Raspberry Pi
        - For Sensor Interface PCB Power
        - For USB Hub

      https://goo.gl/MVx56J
    - Connectors and Signal Conditioning Circuitry
        - GPS - UART via GPIO
          https://goo.gl/YvJ5Q9
        - 6x (upto) UltraSonic Sensors - TRIG and PWM ECHO via GPIO
          Model No: MB7380
          http://www.maxbotix.com/Ultrasonic_Sensors/MB7380.htm

        - 9-Axis IMU - I2C
          Model No: MPY9250
          https://goo.gl/sknx9o
        - Thermal Camera - I2C and SPI
        - 4x (upto) Button Inputs - via GPIO
        - 2x16 Text LCD - I2C
          https://www.sparkfun.com/products/255
        - DS18B20 Temeprature Sensor
          https://www.maximintegrated.com/en/products/analog/sensors-and-sensor-interface/DS18B20.html
        - 8x (upto) Digital I/O - via I2C
        - 8x (upto) Analog I/O - via I2C
- Interface to Motor Controller - via USB
- Interface to ARM Stepper Controller - via USB
    - Stepper controller - via USB
    - Stepper driver
      Model No: tb6600
      https://goo.gl/z9MJMt
- Powered USB Hub
    - 2x (upto) Camera - via USB
      Model No: C270h
      http://www.logitech.com/en-in/product/hd-webcam-c270h
    - RP LIDAR A2 - UART via USB
      https://www.seeedstudio.com/RPLIDAR-A2-The-Thinest-LIDAR-p-2687.html
    - WiFi Module - via USB
      Model No: WN7200ND
      http://www.tp-link.com/lk/download/TL-WN7200ND.html

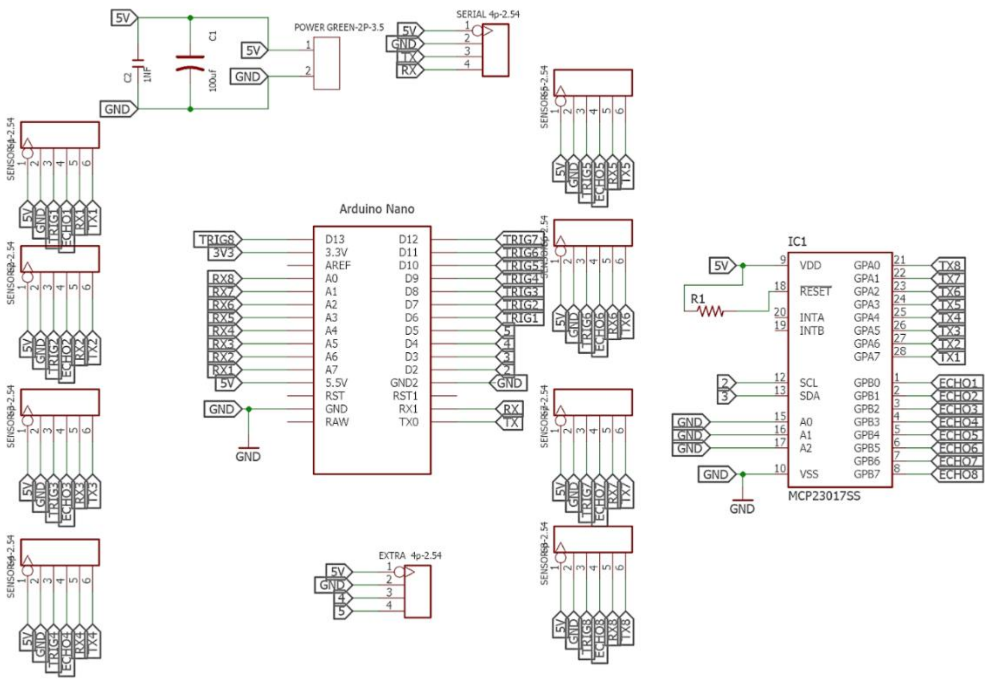


Figure 20 MUSAFIR ver 1.0 schematic of sensor shield

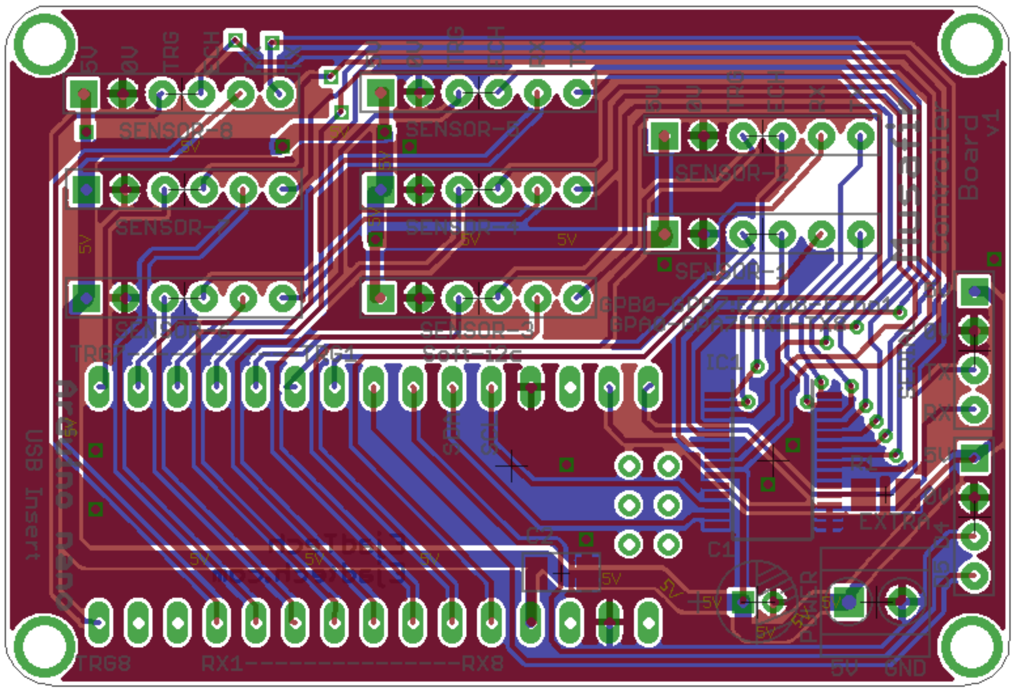


Figure 21 MUSAFIR ver 1.0 layout of sensor shield

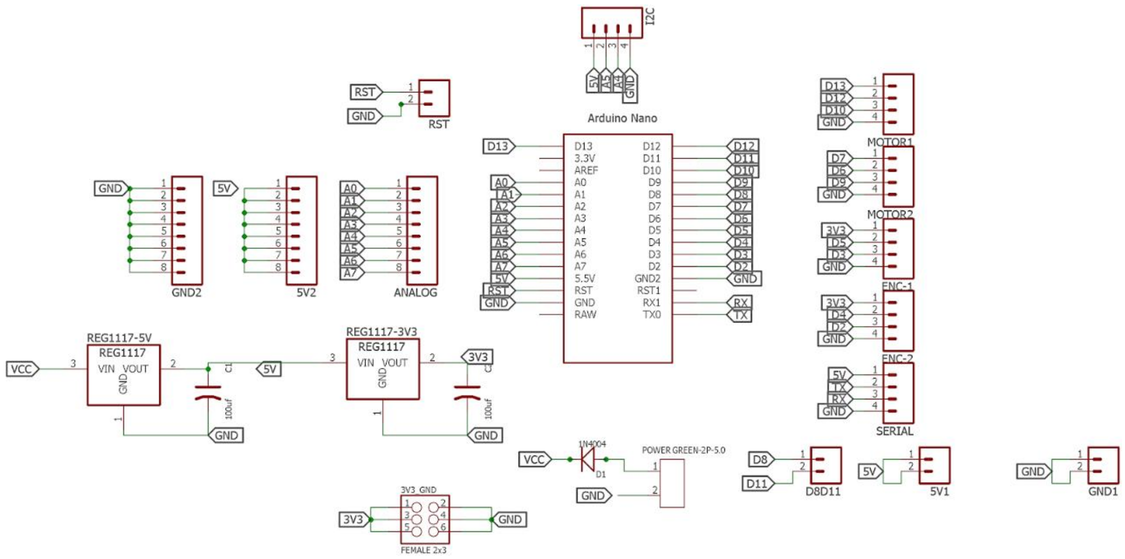


Figure 22 Schematic of motor controller for MUSAFIR robot

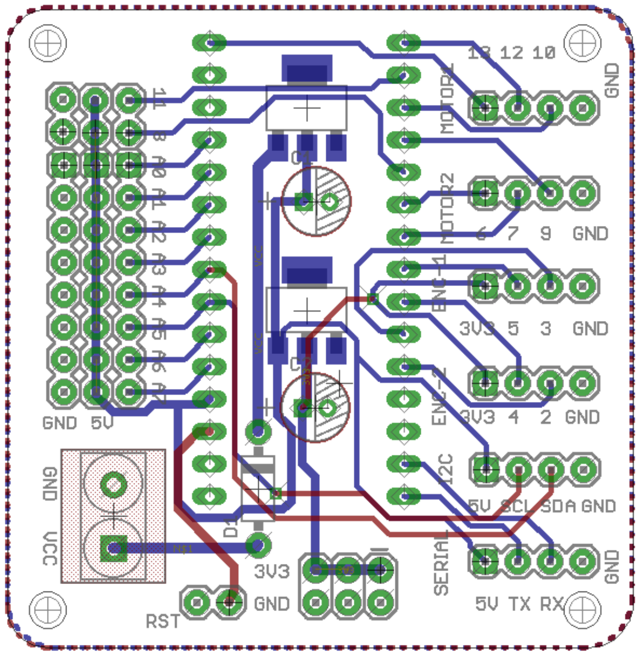


Figure 23 Layout of motor controller for MUSAFIR robot

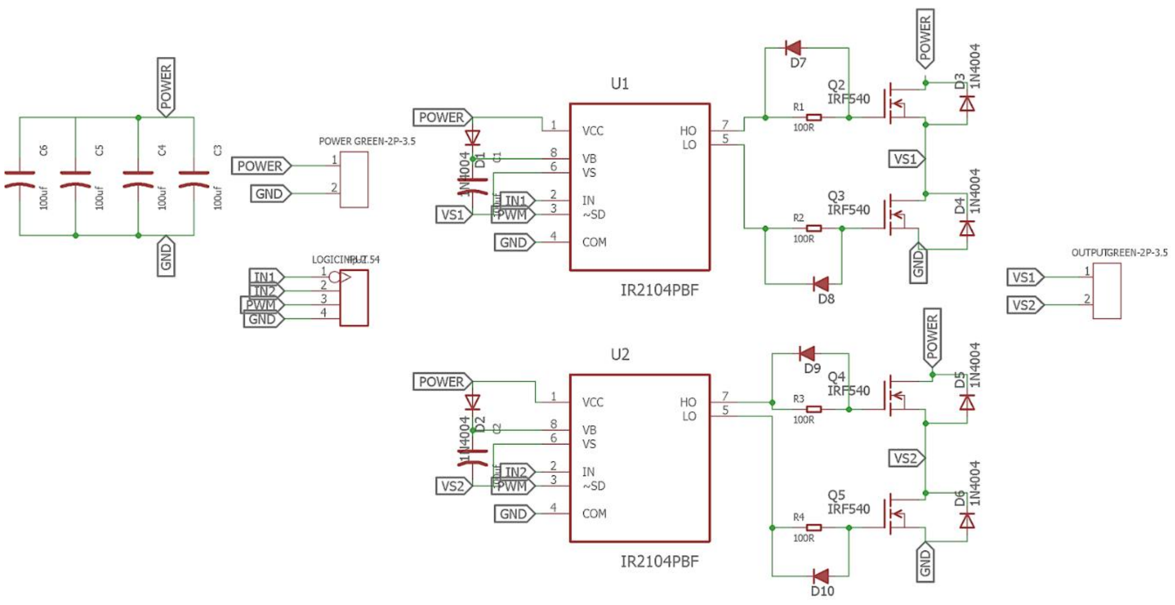


Figure 24 Schematic of motor driver for MUSAFIR robot ver 1.0

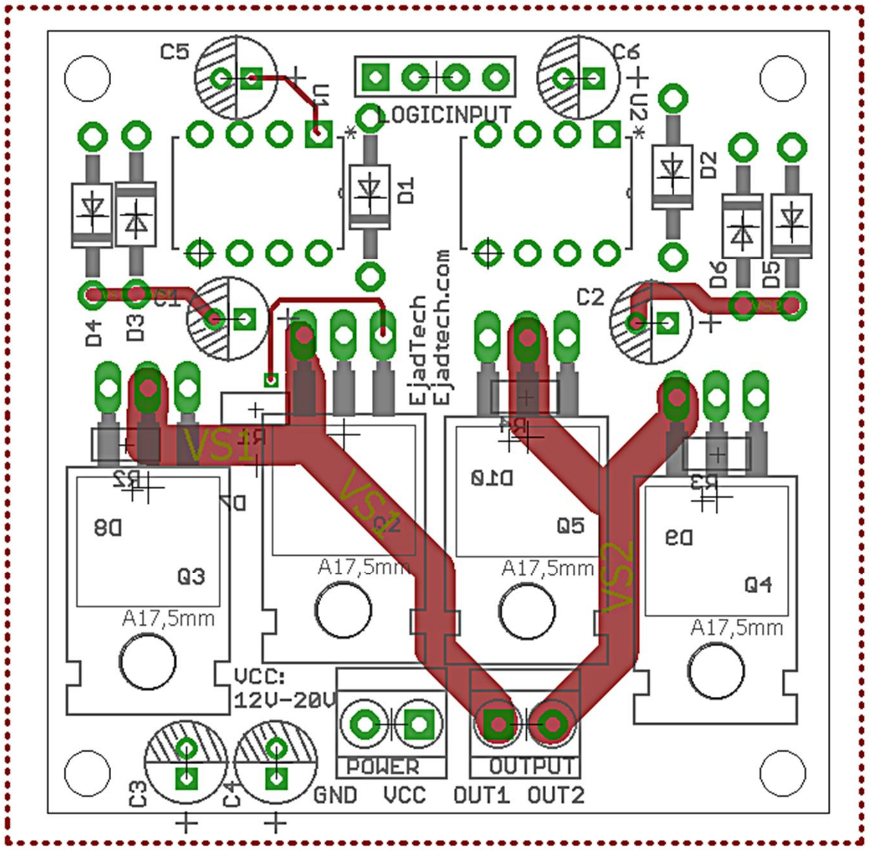


Figure 25 Layout of motor driver for MUSAFIR robot ver 1.0

Chapter 6. Sensor Fusion Techniques for determining the exact geo-location

This chapter presents an approach for the determination of the robot position using both indoor as well as outdoor sensors: the INS, the GPS and the odometer. The proposed technique combines the signals from these sensors through sensor fusion. Here, the Kalman Filter has been to combine the position data obtained from INS and GPS, the KF filtered position data is then input to the complementary filter where it is combined with odometer based position data. For the case where GPS signal becomes unavailable, the Multi-layer Perceptron Neural Networks are employed to predict the robot's position. The MLP neural networks are trained on the input-target data when the GPS signal and the KF signal is available and tested to make the prediction for the robot's location during the phase when GPS is off, i.e, during GPS outages. The position data predicted by the MLP NN is then fed to the complementary filter where this data is combined with the odometer based robot position data to obtain the final output for robot's position. The proposed technique also takes into account the slippage problem with odometer. A fuzzy logic system is used here, to overcome the odometer slippage problem by appropriately adjusting the weights of the complementary filter whenever the condition for slippage occurs. The simulations for the proposed system are carried out in V-REP and MATLAB environments.

6.1 Introduction

Robot localization is an important aspect in R&D research areas where there is a need to control a robot. This paper proposes a technique for obtaining a robot's location by utilizing the signals from various indoor as well as outdoor sensors. Several schemes have been proposed for the determination of precise information of an object's position.

N. Musavi and J. Keighobadi in their works have designed an adaptive fuzzy neuro-observer (AFNO) to apply on the integrated SINS-GPS system [1]. In this scheme, the outputs of INS from 3-axis accelerometers and 3-axis gyroscopes are used to construct an input vector. This vector along with the estimated state vector comprising of position velocity and attitude is provided as input to the fuzzy neural network (FNN) that generates an approximated uncertainty term and as a next step the full state linearized state space models are constructed. When the GPS signal is available, the dynamics of the estimated states are found using the state transition and input matrices, approximated uncertainty term obtained from FNN and the error between measured and outputs obtained from estimated states. This vector is then integrated to obtain the estimated state vector from which the output vector is calculated that contains the position and velocity information of an object. The error between the output vector and the GPS measurement vector is the used to update the weights of FNN. When the GPS is lost, the dynamics of the estimated states are found using the state transition and input matrices, approximated uncertainty term obtained from FNN and the previous values of estimated state vector. A transformation is then applied on this vector using diffeomorphism. This transformed vector is integrated to obtain estimated state vector comprising of position and velocity of the object. The major draw-back of the scheme is that it is complex and difficult to implement.

Further, it assumes a constant GPS outage of 1second only and does not propose a solution for longer GPS outages or signal loss. Further, the proposed scheme provides a solution for outdoor localization applications only.

Jing Li et al. have developed a localization scheme based on Kalman Filters and artificial neural networks [2]. In their proposed scheme, the Kalman Filters and SINS to provide position information when the GPS is present and as the GPS signal becomes unavailable, the neural networks are used to train on the SINS and KF data stored in the database that was available during the presence of GPS signal and tested to predict the robot's position based on the current SINS data. This scheme is effective during the presence and absence of GPS signal loss but the proposed scheme has some major drawbacks: in their proposed method, they have assumed fixed intervals of training and testing of the neural networks. However in practical situations, the availability of the GPS signal is not that certain. The time intervals for the GPS signals vary time to time and therefore, the neural network training and testing times cannot be assumed to be always fixed. Secondly this scheme does not provide a solution for very large GPS outages or when the GPS fails to provide position and velocity information for a long time. In such a case, the neural network may not be able to predict the correct position after a particular elapsed time. This is the case that can be considered for purely indoor applications where a robot enters a building where GPS signal is completely blocked. Our proposed scheme is based on the work by Jing Li et al. Our scheme removes the draw-backs of this scheme. It is simple to implement and is applicable for both indoor and outdoor applications.

6.2 The Inertial navigation system

6.2.1 The Data from INS:

INS stands for Inertial Navigation System. It comprises of tri-axis accelerometers and gyroscopes. The accelerometers provide acceleration values in x, y and z directions. The gyroscope gives angular rates. Using acceleration values, an object's velocity can be calculated by integration. Similarly, the object's attitude information can be obtained by integrating the angular rates provided by the gyroscopes. The accelerations and angular rates obtained from the INS are in body frame co-ordinate system. Therefore, these values should be rotated to convert these values to any appropriate world co-ordinate system before further processing. The conversion of INS values from body frame to world co-ordinate frame is discussed as follows.

6.2.2 Converting INS Data From Body Frame To World Co-ordinates:

As discussed above, an inertial navigation system is composed of both accelerometers and gyroscopes. The data from accelerometers give values in body frame, which is the frame attached to the body of the sensor. To convert these acceleration values from body frame to world co-ordinates, a direction cosine matrix (DCM) is required which is denoted by C_b^n. In this paper, the geographic co-ordinate system – East-North-Up (ENU) frame is used. The DCM for converting from body frame to ENU frame is as follows:

$$C_b^{\,n} = \begin{bmatrix} c(\psi)c(\phi) - s(\psi)s(\theta)s(\phi) & -s(\psi)c(\theta) & c(\psi)s(\phi) + s(\psi)s(\theta)c(\phi) \\ s(\psi)c(\phi) + c(\phi)s(\theta)s(\psi) & c(\psi)c(\theta) & s(\psi)s(\phi) - c(\psi)s(\theta)c(\phi) \\ -c(\theta)s(\phi) & s(\theta) & c(\theta)c(\phi) \end{bmatrix} \text{------ (1)}$$

Here, 'c' represents trigonometric function, 'cosine' and 's' represents trigonometric function, 'sine'. Also, ϕ, θ, ψ are roll pitch and yaw angles respectively also called 'Euler Angles'.

The body frame accelerations or velocities are then converted to world co-ordinates by vector multiplication with the DCM as described by equation (2) and (3) respectively.

$$a_n = C_b^{\,n}\, a_b \quad \text{----- (2)}$$

$$v_n = C_b^{\,n}\, v_b \quad \text{------ (3)}$$

Where,

a_n	represents acceleration vector in inertial navigation frame
a_b	represents acceleration vector in body frame
v_n	represents acceleration vector in inertial navigation frame
v_b	represents acceleration vector in body frame

The block diagram in fig 23 shows the conversion process from body frame to ENU inertial navigation frame.

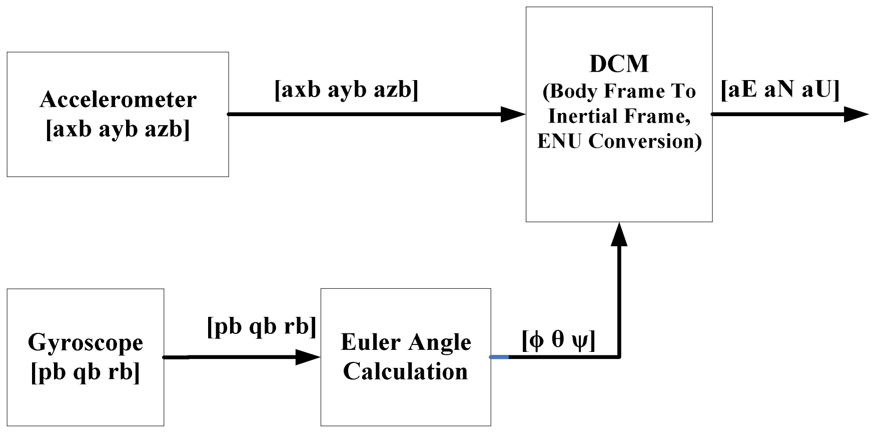


Figure 26 *Conversion from Body frame to World Co-ordinates*

From equation (1) it can be noticed that calculation of DCM requires the euler angles: ϕ, θ, ψ .

These Euler angles can be calculated using the angular rate measurements from gyroscopes as follows [3]:

1. First the initial values of four Euler parameters, e_0, e_1, e_2, e_3 are calculated using the initial values of roll, pitch and yaw angles $\varphi_i, \theta_i, \psi_i$ as:

$$e_{oi} = c(\frac{\psi_i}{2})c(\frac{\theta_i}{2})c(\frac{\varphi_i}{2}) + s(\frac{\psi_i}{2})s(\frac{\theta_i}{2})s(\frac{\varphi_i}{2}) \quad \text{---- (4)}$$

$$e_{1i} = c(\frac{\psi_i}{2})c(\frac{\theta_i}{2})s(\frac{\varphi_i}{2}) - s(\frac{\psi_i}{2})s(\frac{\theta_i}{2})c(\frac{\varphi_i}{2}) \quad \text{--- (5)}$$

$$e_{2i} = c(\frac{\psi_i}{2})s(\frac{\theta_i}{2})c(\frac{\varphi_i}{2}) + s(\frac{\psi_i}{2})c(\frac{\theta_i}{2})s(\frac{\varphi_i}{2}) \quad \text{---- (6)}$$

$$e_{3i} = c(\frac{\psi_i}{2})s(\frac{\theta_i}{2})s(\frac{\varphi_i}{2}) + s(\frac{\psi_i}{2})c(\frac{\theta_i}{2})c(\frac{\varphi_i}{2}) \quad \text{--- (7)}$$

Here, 'c' represents trigonometric function, 'cosine' and's' represents trigonometric function, 'sine'

2. Using the above initial values of euler parameters, the time trajectory calculated as:

$$\dot{e}_0 = -\frac{1}{2}(e_1 p + e_2 q + e_3 r) \quad \text{---} \quad (8)$$

$$\dot{e}_1 = \frac{1}{2}(e_o p + e_2 r - e_3 q) \quad \text{----} \quad (9)$$

$$\dot{e}_2 = \frac{1}{2}(e_o q + e_3 p - e_1 r) \quad \text{----} \quad (10)$$

$$\dot{e}_3 = \frac{1}{2}(e_o r + e_1 q - e_2 p) \text{-----} \quad (11)$$

3. Finally the Euler angles are calculated using the above calculated Euler parameters:

$$\theta = \sin^{-1}(-2[e_1 e_3 + e_o e_2]) \text{ ---- (12)}$$

$$\varphi = \cos^{-1}\frac{[e_o{}^2 - e_1{}^2 - e_2{}^2 + e_3{}^2]}{\sqrt{(1-4(e_1 e_3 - e_o e_2)^2)}} sign(2[e_2 e_3 + e_o e_1]) \text{ --- (13)}$$

$$\psi = \cos^{-1}\frac{[e_o{}^2 - e_1{}^2 - e_2{}^2 - e_3{}^2]}{\sqrt{(1-4(e_1 e_3 - e_o e_2)^2)}} sign(2[e_1 e_2 + e_o e_3]) \text{--- (14)}$$

Using the above Euler angles, the DCM matrix in (1) can be calculated.

6.3 Global Positioning System (GPS)

GPS stands for Global Positioning system. We can now determine the position of an object anywhere on the globe using the GPS. The GPS is able to provide accurate position and velocity information but the response of GPS is subjected to errors when there are situations like satellite blockage .Also in indoors, GPS signals are blocked by enclosed spaces such as tunnels or buildings [4].The position components obtained from the GPS have Gaussian noise characteristics [5] and therefore, for simulation purpose, the GPS based position can be modeled according to the following equation [6]:

$$GPS\_position = Actual\_position + \sigma * Gaussian\_Noise \text{ --- } (15)$$

Here, ' σ ' is the standard deviation whose value can be varied in accordance with the availability or strength of the GPS signal [6].

According to the proposed scheme, whenever the GPS signal is available, to account for the errors in the GPS signal, the GPS based position is filtered using the Kalman filter sensor fusion scheme, where the GPS data combined with the INS data to obtain a better estimate of a robot's position., the KF filtered position data is then combined with the odometer based position information using complimentary filter to obtain the final output. A detailed discussion of sensor fusion of GPS-INS using the Kalman Filter and KF – Odometer integration is presented in Section V.

And when the GPS signal is unavailable, the Multi-layer perceptron network is trained and tested to predict the position output in place of the GPS and the final output is obtained by fusing the MLP and odometer based position outputs using complimentary filter. This is presented in detail in Section IV.

6.4 Odometer

The odometer or (wheel sensor) is a sensor that provides the velocity information of a robot. Here, a robot's movement is sensed with the wheel encoder. The velocity obtained from odometer can be integrated to obtain a robot's position [6].An odometer is capable of providing position and velocity information in both the indoors and outdoors. However, the position obtained using odometer has large integration errors which should be corrected using some other mechanisms for localization. Therefore, in the proposed scheme, the odometer position data is combined with KF or MLP predicted position information in the presence and absence of GPS data respectively to obtain the final position output from the system.

The position can be obtained from odometer based velocities as follows [4]:

$$\dot{x} = v\cos(\psi) \text{ ---------- } (16)$$
$$\dot{y} = v\sin(\psi) \text{ ------------ } (17)$$

Where, ' ψ 'is the yaw or heading angle of the robot.

Like the INS the odometer data should also be converted from body frame to inertial navigation frame, i.e, ENU or other frame. Where, the superscript 'b' represents body frame and C_b^n is the DCM matrix given in equation (1)

$$V^{ENU}{}_{OD} = C_b^n V^b{}_{OD} \text{ ----- } (18)$$

Where, the superscript 'b' represents body frame and C_b^n is the DCM matrix given in equation (1)

A drawback with odometer based positioning system is that when it suffers from wheel slip it often results in large localization estimation errors. To correct these localization errors, a system based on fuzzy rules is designed to find the weight of the fusion parameter that is used to fuse the odometer based position data with either the Kalman filter or MLP based position data depending on the availability of the GPS signal.

6.5 GPS-INS Sensor fusion

Sensor Fusion is the process of combining the output signals from different so as to obtain a more acceptable signal than the one obtained from the individual sensors. Sensor fusion reduces noise and combines the advantages of different sensors [7]. In this scheme, sensor fusion of GPS and INS is accomplished using Kalman Filter. The Kalman Filter (KF) algorithm and Kalman Filter formulation for the GPS-INS sensor fusion are discussed as follows [8]:

6.5.1 The Kalman Filter:

The Kalman filter is a well-known sensor fusion technique that comprises of two important mechanisms: prediction and update. The steps involved in Kalman Filter algorithm are described as follows:

1**. Initialize the values of states and error covariance matrices**

$$x\hat{o}, Po \text{ ----- } (19)$$

2. Predict the state and error covariance.

$$\hat{x}_k = A\hat{x}_{k-1} + Bu \text{ ----- } (20)$$

$$P_k^- = AP_{k-1}A^T + Q \text{ ----- } (21)$$

Where, A is the state transition matrix and B is the control input matrix which applies the effect of the input 'u' to the state vector. Q is the process noise covariance matrix and P is the error covariance matrix.

3. Find the Kalman Gain

$$K_K = P_K^- H^T (HP_K^- H^T + R)^{-1} \text{ ----- } (22)$$

Where, H is the observation matrix and R is the measurement noise covariance matrix

4. Calculate the state estimate:

$$\hat{x}_k = \hat{x}_k^- + K(z - H\hat{x}_k^-) \text{ ----- } (23)$$

5. Calculate the error covariance:

$$P_k = P_k^- - KHP_k^- \text{----- (24)}$$

6.5.2 Kalman Filter Formulation for GPS –INS Sensor Fusion:

For combing the data from GPS and INS for better position estimation, a 6 state Kalman filter is implemented. The designed Kalman filter matrices are as follows:

1. The State Matrix (X):

The state matrix is:

$$X = [\, a_E \;\; a_N \;\; v_E \;\; v_N \;\; x_E \;\; y_E \,] \text{ ----- (25)}$$

Where,

a_E	is the robot acceleration along east.
a_N	is the robot acceleration along north.
v_E	is the robot velocity along east
v_N	is the robot velocity along north
x_E	is the robot position in the east direction
y_E	is the robot position in the north direction

The system states are initialized to all zeros assuming no prior knowledge:

$$X_0 = [0_{1X6}] \text{----- (26)}$$

2. The State Transition Matrix (A):

The state transition matrix has been assigned the following values:

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ \Delta t & 0 & 1 & 0 & 0 & 0 \\ 0 & \Delta t & 0 & 1 & 0 & 0 \\ \frac{1}{2}\Delta t^2 & 0 & \Delta t & 0 & 1 & 0 \\ 0 & \frac{1}{2}\Delta t^2 & 0 & \Delta t & 0 & 1 \end{bmatrix} \text{----- (27)}$$

Where Δt is the sampling time between GPS signal.

3. The Input Matrix- (B):

Since, there is no input, the input matrix ‘B’ is zero.

$$B = [0_{6x1}] \text{----- (28)}$$

4. The Measurement Matrix (Z):

The measurement matrix ‘Z’ is composed of sensor measurements from accelerometers and GPS position co-ordinates.

$$\mathrm{z} = [\, v_E \;\; v_N \;\;\; x_E \;\; y_N \,] \text{ ----- (29)}$$

Where, $v_E \;\; v_N \;\;\; x_E \;\; y_N$ have the same definitions as described above.

4. The Observation Matrix (H):
The observation matrix ‘H’ is:

$$H = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \text{----- (30)}$$

4. The Process Noise Covariance Matrix (Q):
The process noise covariance matrix is assigned the following values:

$$Q = \left[0.1_{6X6}\right] \text{----- (31)}$$

5. The Measurement Noise Covariance Matrix (R):
The measurement noise covariance matrix R is:

$$R = \begin{bmatrix} \sigma^2_{a_E} & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma^2_{a_N} & 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma^2_{v_E} & 0 & 0 & 0 \\ 0 & 0 & 0 & \sigma^2_{v_N} & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma^2_{x_E} & 0 \\ 0 & 0 & 0 & 0 & 0 & \sigma^2_{y_N} \end{bmatrix} \text{----- (32)}$$

6. The Error Covariance Matrix (P)
The error covariance matrix P is initialized to:

$$P_0 = Q$$

6.5.3 KF-Odometer Integration:

The final output obtained from the system in the presence of GPS signals is obtained by combing the position outputs estimated from the KF and odometer using the complementary filter as follows.

Final Output = α ***(KFPosition) + (1-***α ***) (Odometer Position)*** ----- (33)

Where 'α' is the weighting/fusion parameter which is used here to fuse the two position outputs from KF and odometer. The range of 'α' is: $0 \le \alpha \le 1$

6.6 Multilayer Perceptron Networks for prediction position data

In the proposed scheme, the Multi-layer perceptron neural networks (MLPNN) are employed to predict the robot's position in the absence of GPS signals. The structure of MLP neural networks and the MLP -Odometer integration scheme is also discussed in this section.

6.6.1 The Multi-Layer Perceptron Architecture:

The structure of MLP neural networks is shown in Fig 24.Its structure consist of an input layer, one or more hidden layers for processing and one output layer. Each node in the MLPNN is connected with weights. MLPNN are an effective tool for regression, classification and non-linear function fitting problems [2].

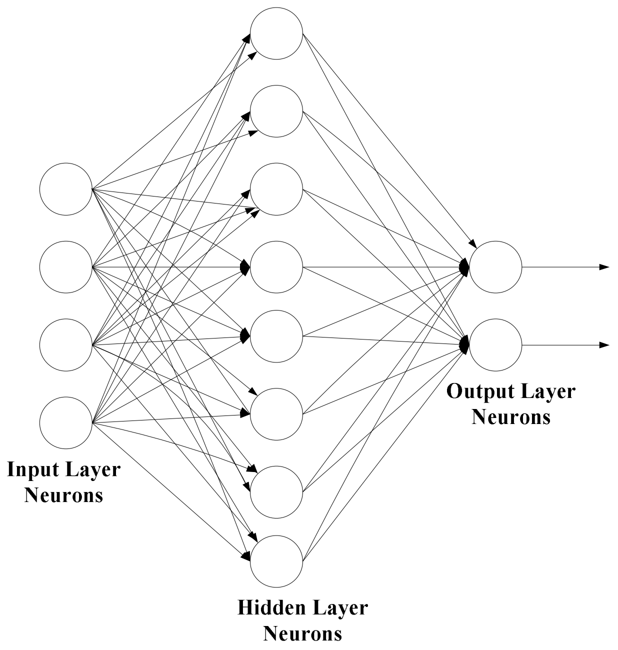


Figure 27 The MLP Architecture

The output of each node is given by:

$$y_i = f(w_i^1 x_1 + w_i^2 x_2 + w_i^3 x_3 \ldots\ldots w_i^m x_m) \text{ ----- (34)}$$

$$\text{Or } y_i = \sum_j w_i^j x_j \text{ ----- (35)}$$

Where, ' x_i ' is the input from the node 'j' of the input data layer ' w_i^j ' is the weight connecting input node 'j' to a neuron 'i' in the next layer. 'f' is the activation function.

6.6.2 MLP-INS Integration Scheme during the unavailability of GPS signal:

When no GPS signals are available, i.e, during GPS outages, the Kalman filter cannot be used to predict the robot's position data. And we are left with the INS system only. In this phase, however, we utilize the accelerometer and gyroscope data obtained from the inertial navigation system to train MLPNN. The multi-layer perceptron neural network will be trained on the input-output data when the GPS was available. And the trained MLP network is then tested to predict the robot's position when there's no GPS and Kalman filtered signals available. In this scheme, two Multi-layer Perceptron Neural Networks have been created, one for predicting the x-component of the robot's position and another for predicting the y-component of the robot's position. The designed architecture for both the MLP neural networks is [8 x 15 x 1].i.e, for both the MLPNN, there are 8 inputs , one hidden / processing layer comprising of 15 neurons and 1 output layer neuron. Levenberg-Marquardt back-propagation algorithm is used to train the MLP networks.

The MLP networks used to predict the position has the following inputs and targets.

A. Input and Targets for MLP 1:

Inputs: $[v_E \; v_N \; v_U \sum v_E \sum v_N \sum v_U \; \theta \sum \theta \;]$

Targets: $[x]$

B. Input and Targets for MLP 2:

Inputs: $[v_E \; v_N \; v_U \sum v_E \sum v_N \sum v_U \; \theta \sum \theta \;]$

Targets: $[y]$

Where,

$v_E \; v_N \; v_U$	are velocities in ENU inertial frame
$\sum v_E \sum v_N \sum v_U$	are cumulative velocities in ENU inertial frame
θ	is the yaw angle
$\sum \theta$	is the cumulative yaw angle

x	is the robot's position component along East direction in the ENU frame
y	is the robot's position component along North direction in the ENU frame

The training and testing phases of Multilayer Perceptron Networks are shown in Fig 25 and Fig 26 respectively.

6.6.3 MLP-Odometer Integration:

The final output obtained from the system in the absence of GPS signals is obtained by combing the position outputs from the MLP and odometer using the complementary filter as follows.

Final Output = α ***(MLP Position)*** + ***(1-***α ***) (Odometer Position)*** ----- (36)

Where 'α' is the weighting/fusion parameter which is used here to fuse the two position outputs from MLP and odometer. The range of 'α' is: $0 \le \alpha \le 1$

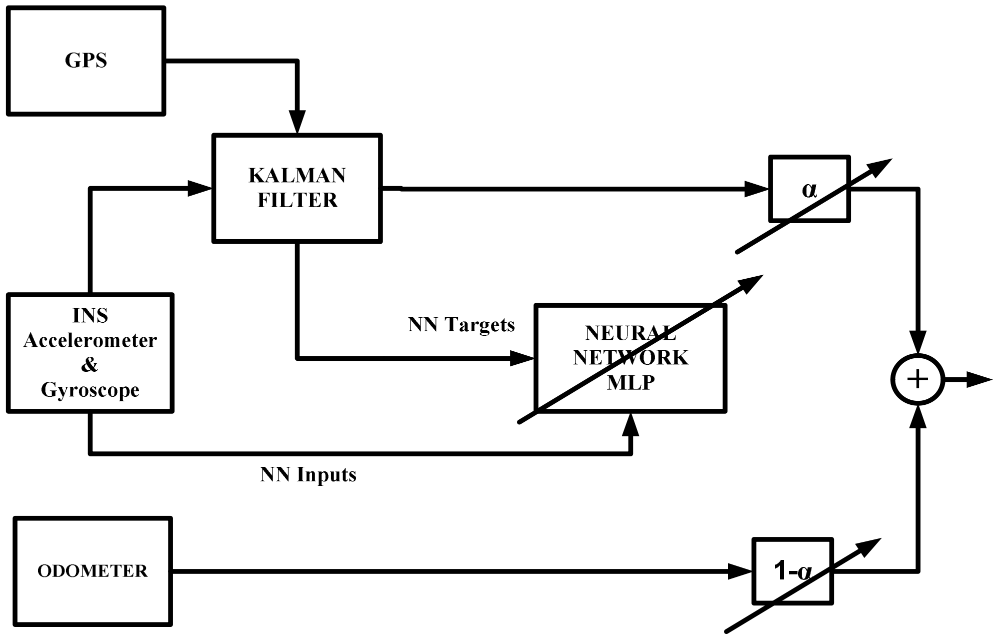


Figure 28 MLP Training Phase

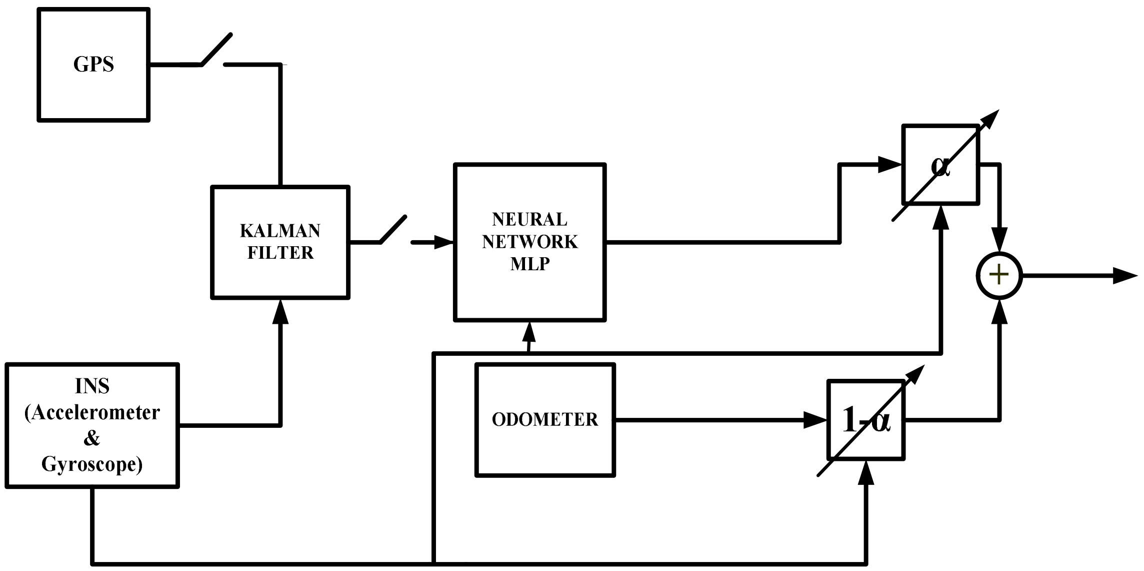


Figure 29 MLP Testing Phase

6.7 Fuzzy System to incorporate Robot Slippage

As described in Section IV, when the odometer suffers from wheel slippage, it registers wrong position readings because the wheels are constantly moving but the object's position almost remain constant .This position error is eliminated with a fuzzy system that compares the high pass filtered velocity outputs of both the INS and odometer by calculating the error between the two sensor readings and generates a suitable value of the fusion parameter as output.

A. Fuzzy System Inputs and Outputs:

The designed fuzzy logic system has the following inputs and outputs

Input(s): error between high pass filtered velocity outputs of both the INS and odometer (e).

Output(s): the fusion parameter /(α) ;

The range of 'α' is between '0' and '1'.

B. Fuzzy Rule Base:

The rule base for the fuzzy logic system is as follows:

Rule 1*:* If error is zero then fusion parameter is medium.

Rule 2: If error is large then fusion parameter is small.

Rule 3: If error is small then fusion parameter is large.

C. Fuzzy Membership Functions:

The membership functions for input and ouput are as follows:

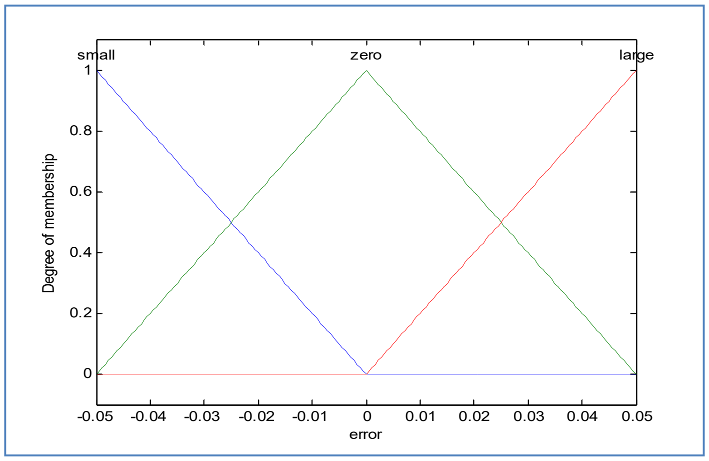


Figure 30 Membership Function for input error(e)

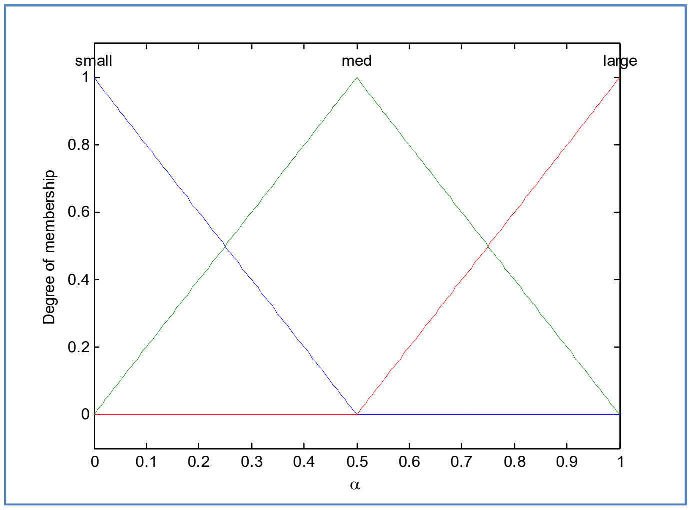


Figure 31 Membership Function for output-fusion parameter (α)

6.8 Simulation results

The simulation results are generated for the following three test cases:

6.8.1 Case-I: KF/GPS signal is available and there's odometer wheel slippage:

Fig 29(a) and (b) show the simulation result for the case where the GPS and so the KF signal is available, the KF and odometer fused output is shown in magenta .It can be the odometer wheel slippage is detected by the fuzzy logic system correctly. This is shown by the cyan color superimposed on the final output in magenta.

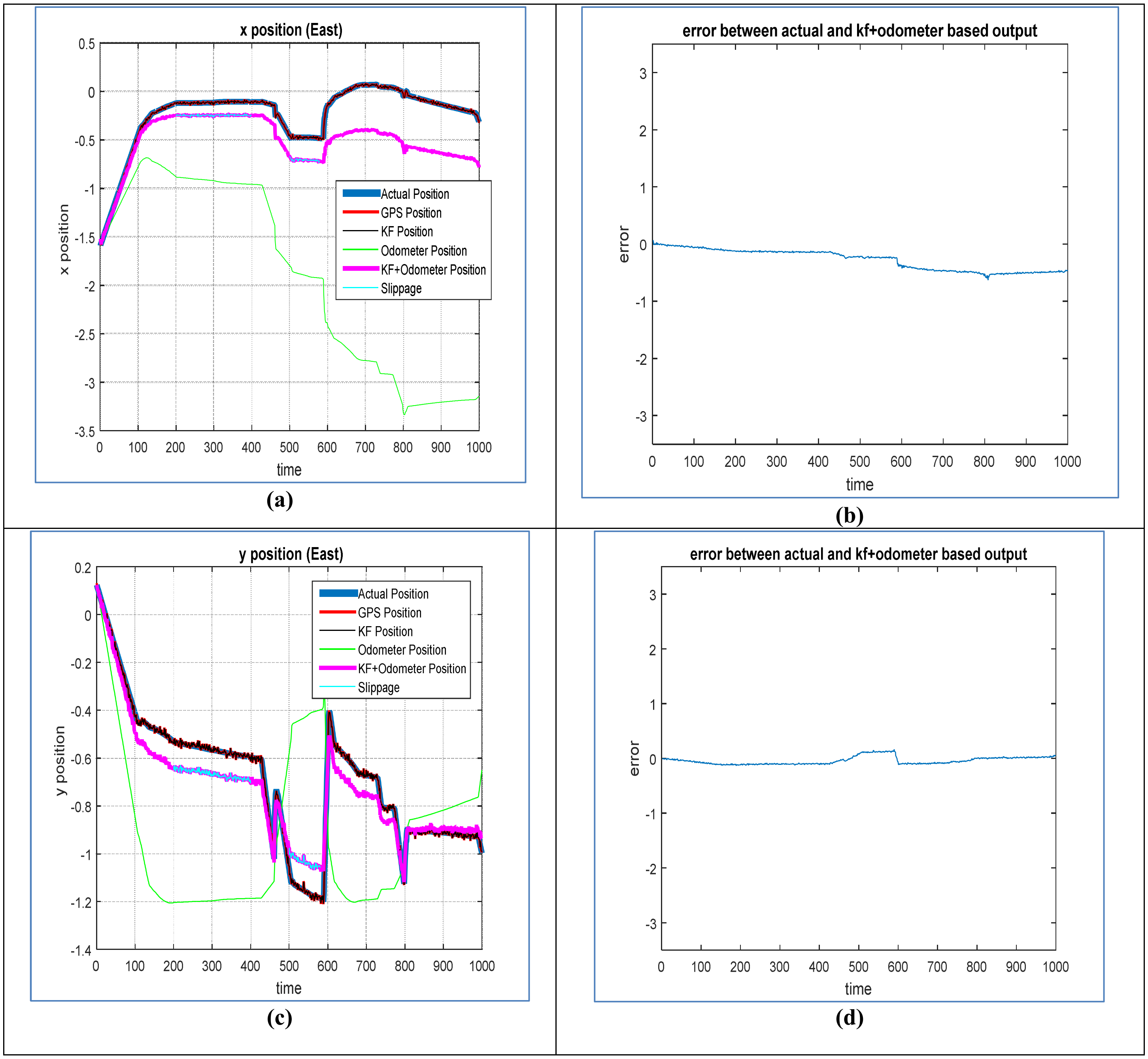
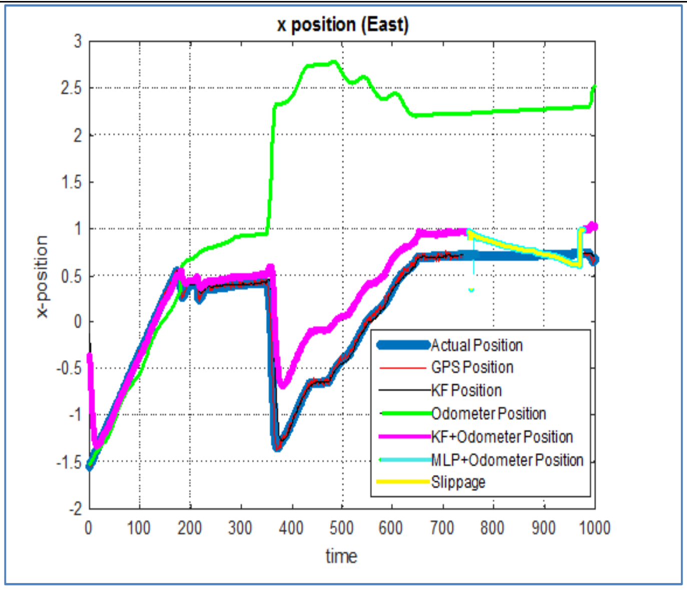


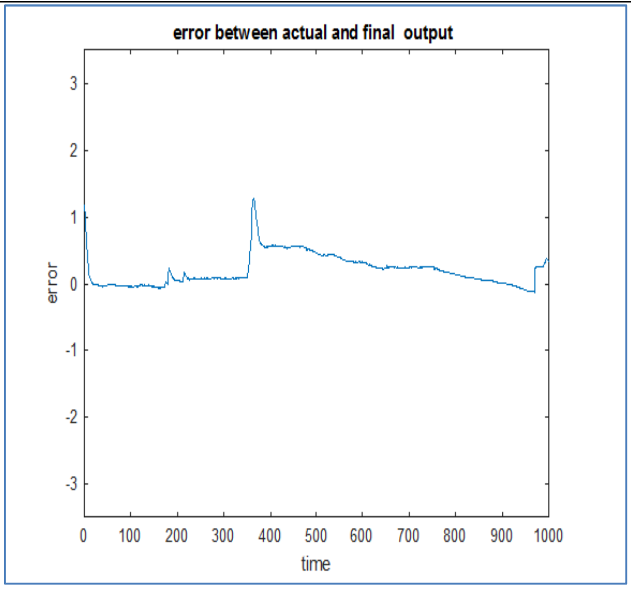
Figure 32 Simulation results for case 1

6.8.2 Case-II: KF/GPS signal is unavailable and there's odometer wheel slippage:

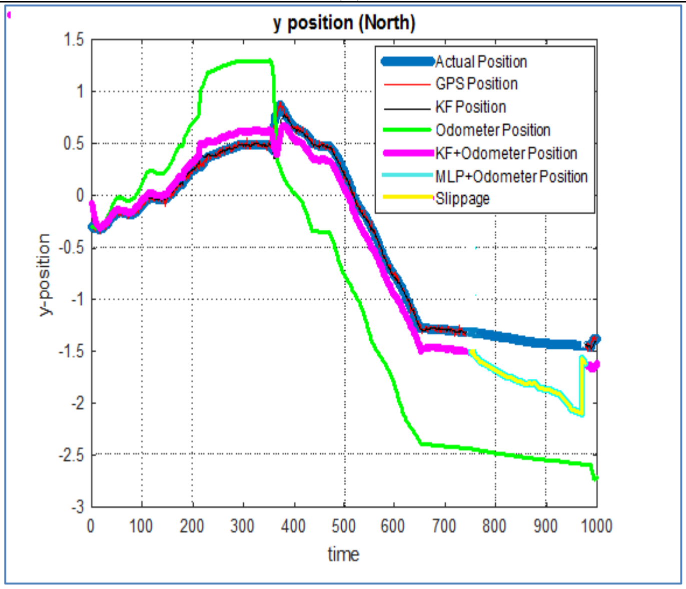
Fig 30(a) and (b) show the simulation result for the case where the GPS and so the KF signal is available for time t = 0 to 750 seconds and the GPS signal becoming unavailable for the time t = 751 to 950 seconds, the KF and odometer fused output is shown in magenta .During this time interval, MLP network is trained and tested to predict the robot's position whereas in this time interval, the odometer slippage is also considered. the MLP and odometer fused output is shown in cyan It can be observed that the odometer wheel slippage is detected by the fuzzy logic system correctly. This is shown by the yellow colored line superimposed on the final output in cyan.



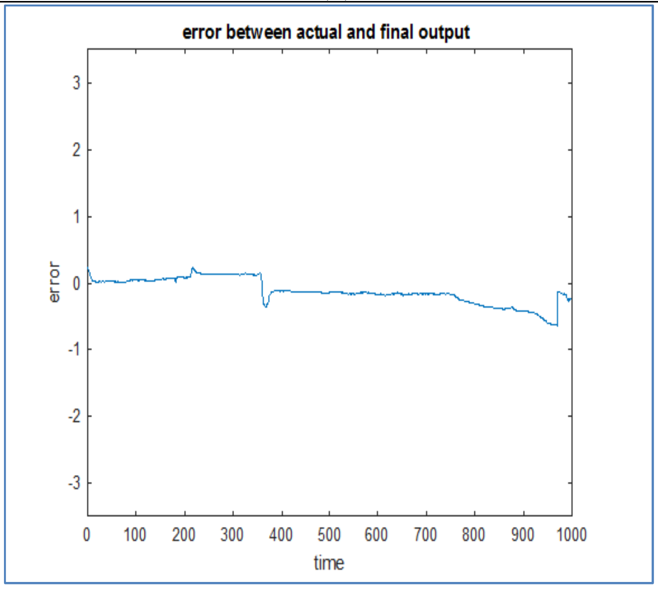
(a)



(b)



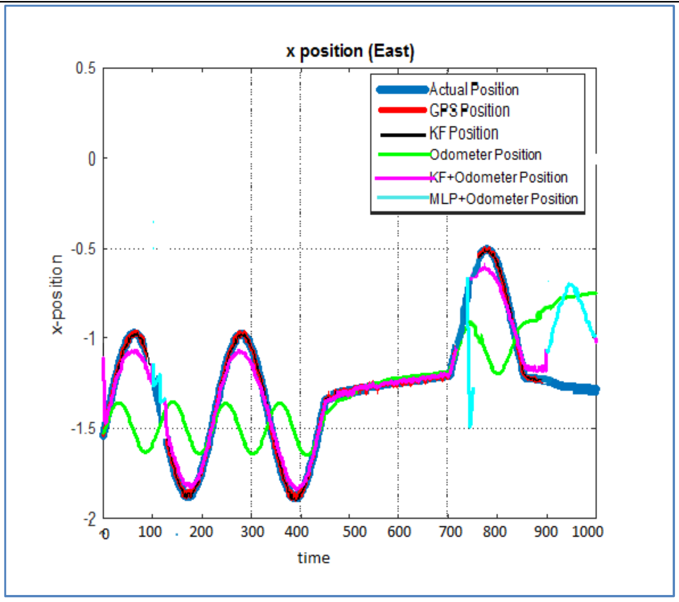
(c)



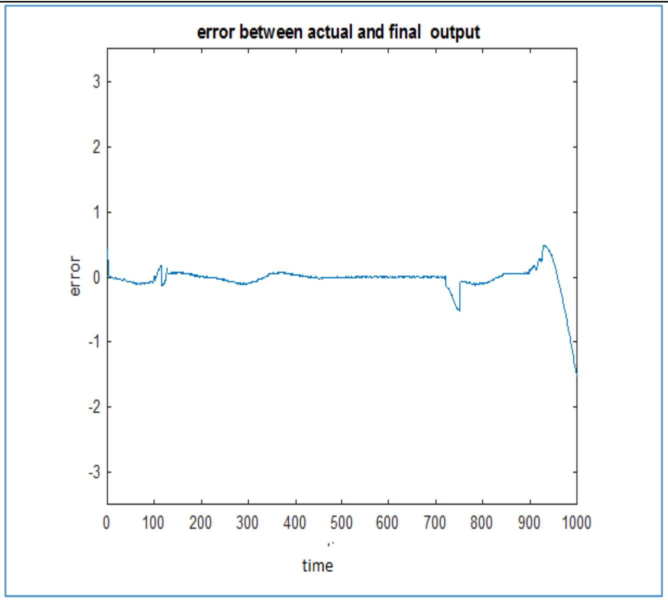
(d)

Figure 33 Simulation results for case 2

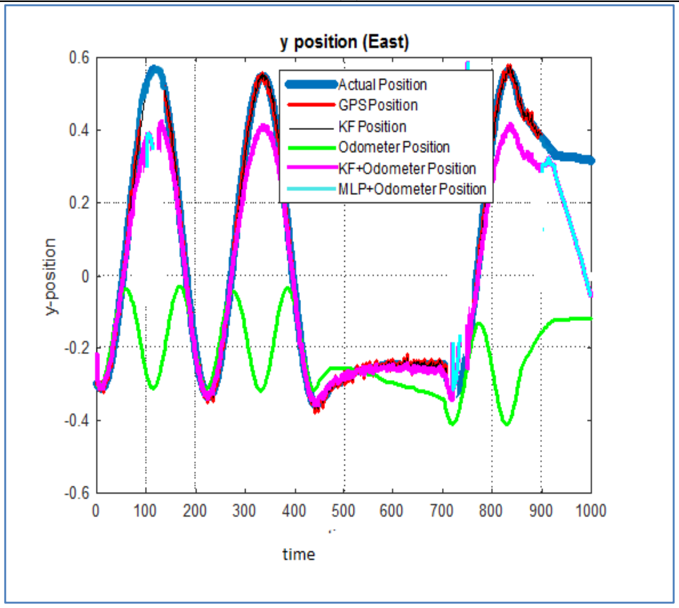
6.8.3 Case-III: KF /GPS signal is unavailable and there's no odometer wheel slippage:



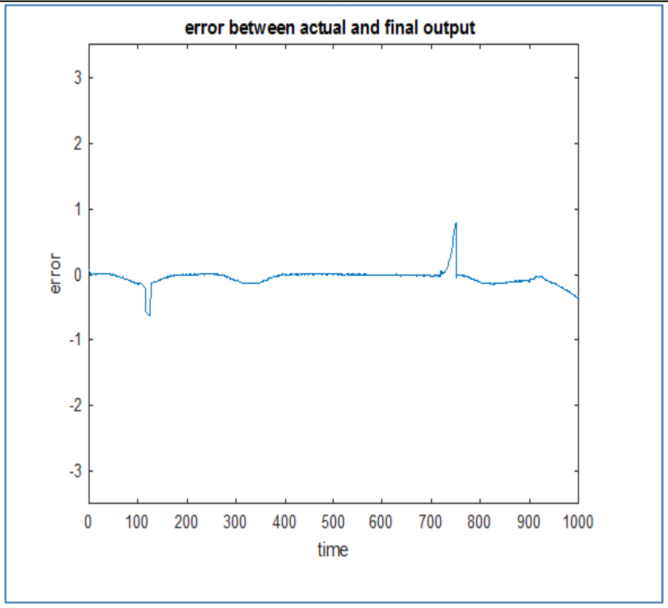
(a)



(b)



(c)



(d)

Figure 34 Simulation results for case 3

Fig 31(a) and (b) show the simulation result for the case where the GPS and so the KF signal is unavailable for time t = 100 to 125,t= 720 to 750 and t = 900 to 1000 seconds and during these time intervals no wheel slippage is considered. The KF and odometer fused output is shown in magenta and the MLP and odometer fused output is shown in magenta .And the fuzzy logic system correctly identifies the no slippage condition and adjusts the fusion parameter accordingly.

6.9 Conclusion

In this chapter, we have proposed a technique for the precise localization of a robot. The proposed scheme is simple to implement and provides localization solution in both indoor and outdoor applications. Further a fuzzy logic Mamdani system has been developed to minimize the positioning errors in the odometer due to slippage condition. The algorithm was successfully applied on the data obtained from sensors in V-REP environment.

6.10 References

[1] N.Musavi, J.Keighobadi, "*Adaptive fuzzy neuro-observer Applied to low cost INS/GPS*", Appl. Soft Comput.J. (2014)

[2] L.Jing, S.Ningfang, Y.Gongliu et al. "*Improving position accuracy of vehicular navigation system during GPS outages utilizing ensemble learning algorithms*." Information Fusion, pp-1-10, 2017.

[3]K.N.Vikas, '*Integration of Inertial Navigation System and Global Positioning System Using Kalman Filter*', MS Thesis , Dept. of Aerospace Engineering, Indian Institute of Technology, Mumbai, 2004.

[4] M. David, '*Multi-rate Sensor Fusion for GPS Navigation Using Kalman Filtering',* MS Thesis , Dept. of Electrical Engineering, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, May 1999.

[5] A. Noureldin, A. El-Shafi, M. Bayoumi, "*GPS/INS integration utilizing dynamic neural networks for vehicular navigation*", Information Fusion 12 (2011) 48–57

[6] K.M. Chinmaya, B.K.Mishra, J.S Jebakumar,"EKF *based GPS/Odometer Data Fusion for Precise Robot Localization*," IOSR Journal of Mechanical and Civil Engineering, Vol.2,pp. 70-75,April 2014.

[7]N S.Jaewon, K.L.Hyung, G.L.Jang, et al. "*Lever Arm Compensation for GPS/INS and Odometer Integrated System*," International Journal of Control, Automation, and Systems, Vol.4, pp.247-254, April 2006.

[8] G.Welch, G. Bishop, "*An Introduction to the Kalman Filter*," Technical Report: TR95-041, University of North Carolina, 2006.

Chapter 7. Improvements in the design of the mechanical structure of IMR

7.1 Tank based design ver 1.0

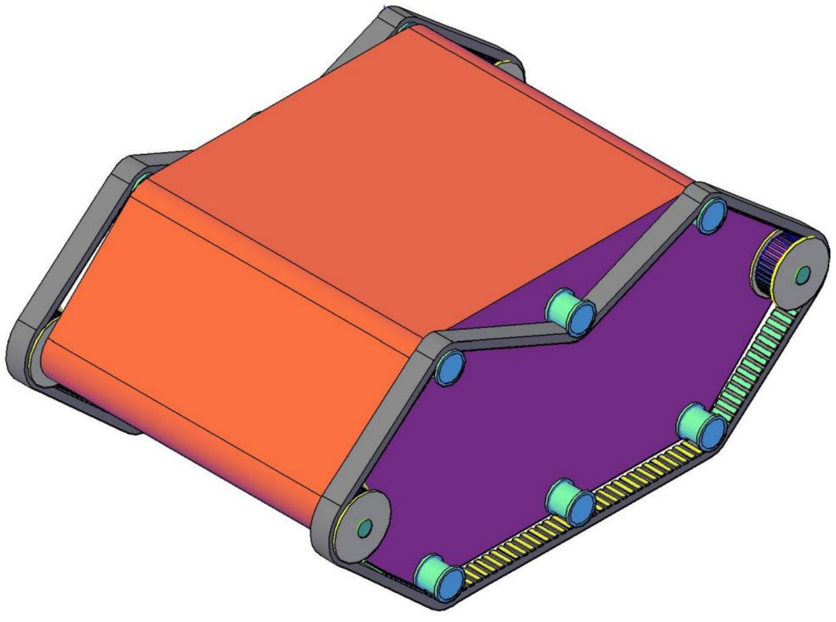


Figure 35 3D Model of tank based robot version 1

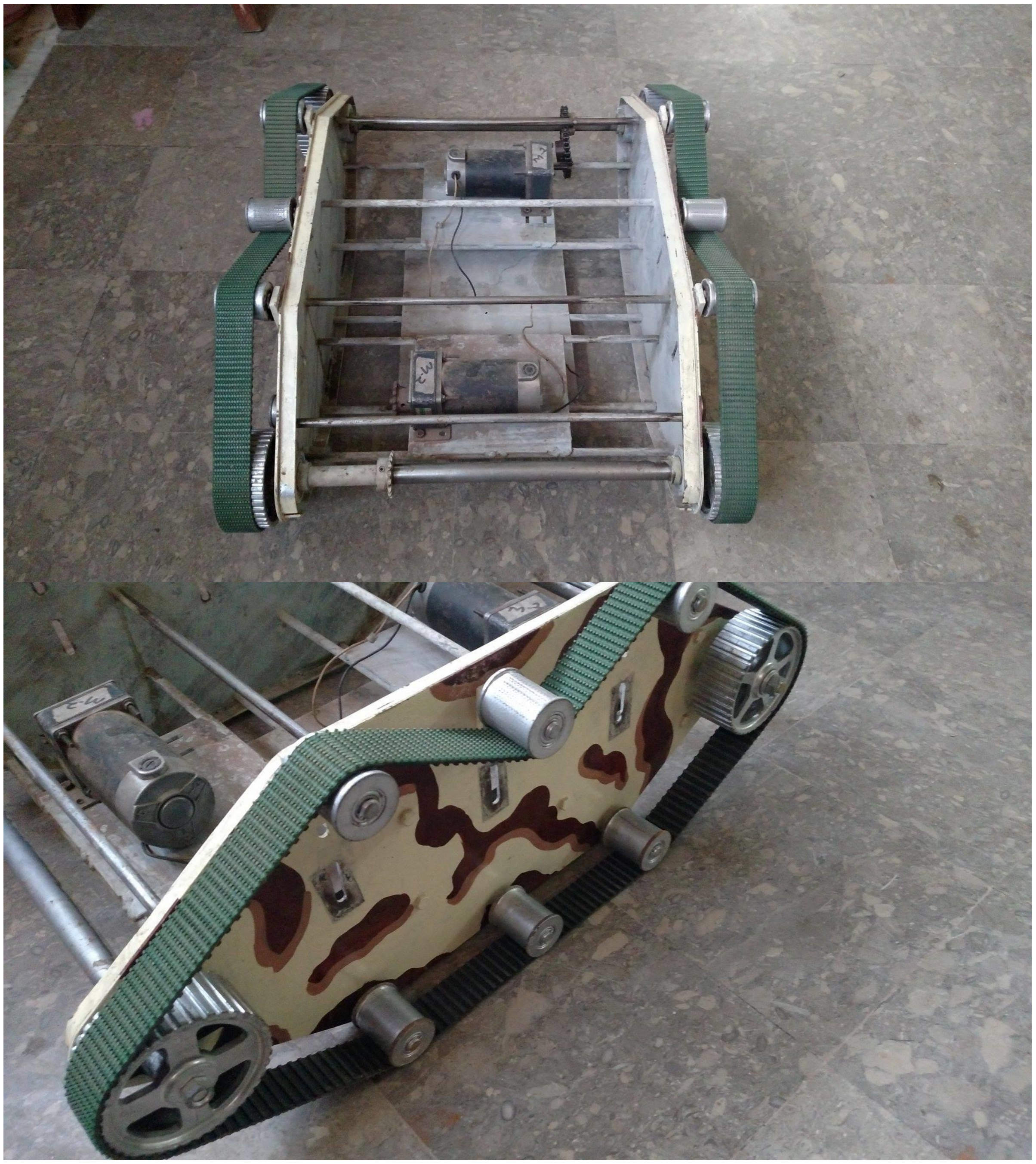


Figure 36   Actual model of tank based robot version 1.0

7.2   Tank based design ver 2.0

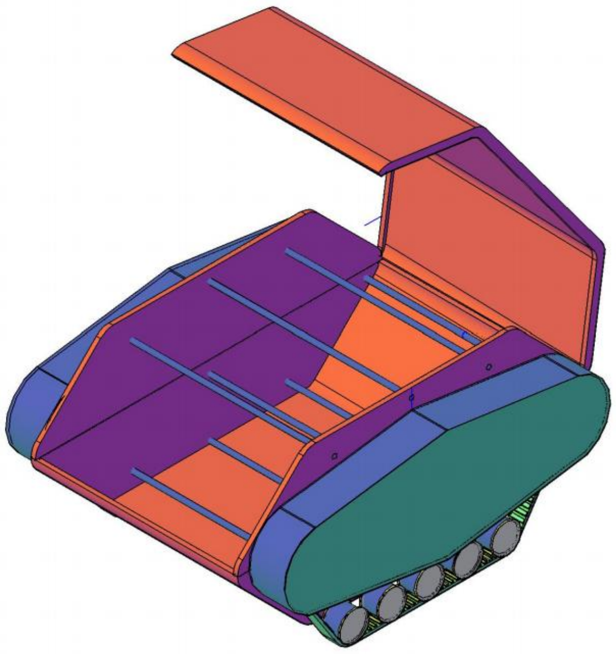


Figure 37 3D Model of tank based robot version 2

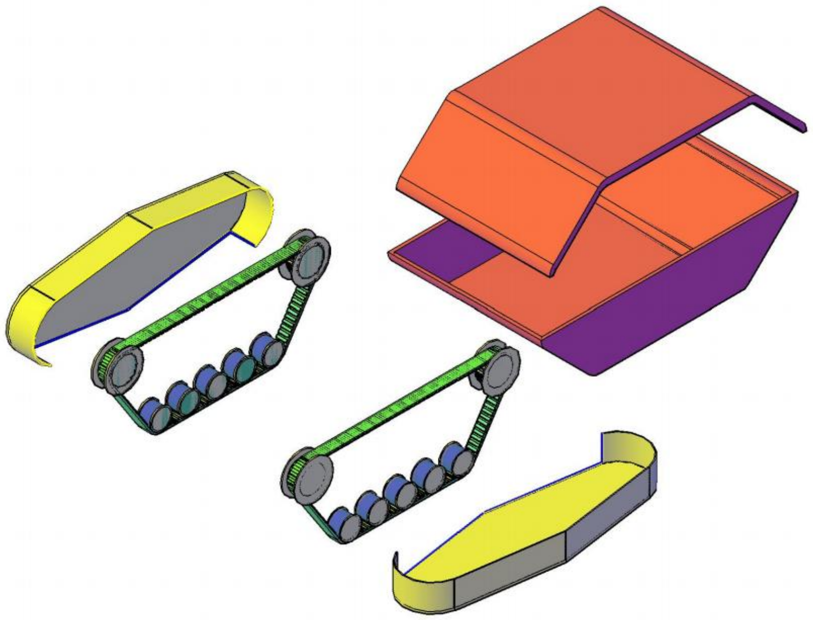


Figure 38 Exploded view

7.3 Problems and challenges faced in the initial porotype

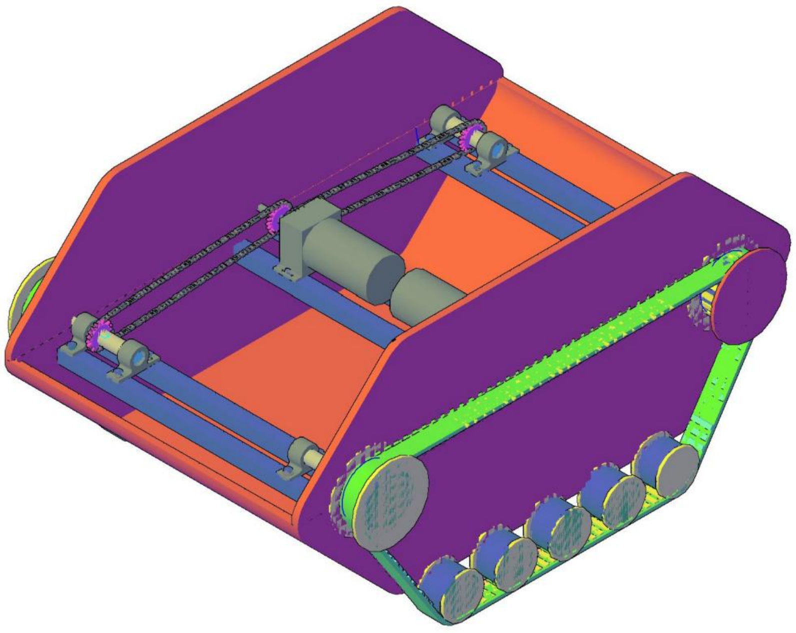
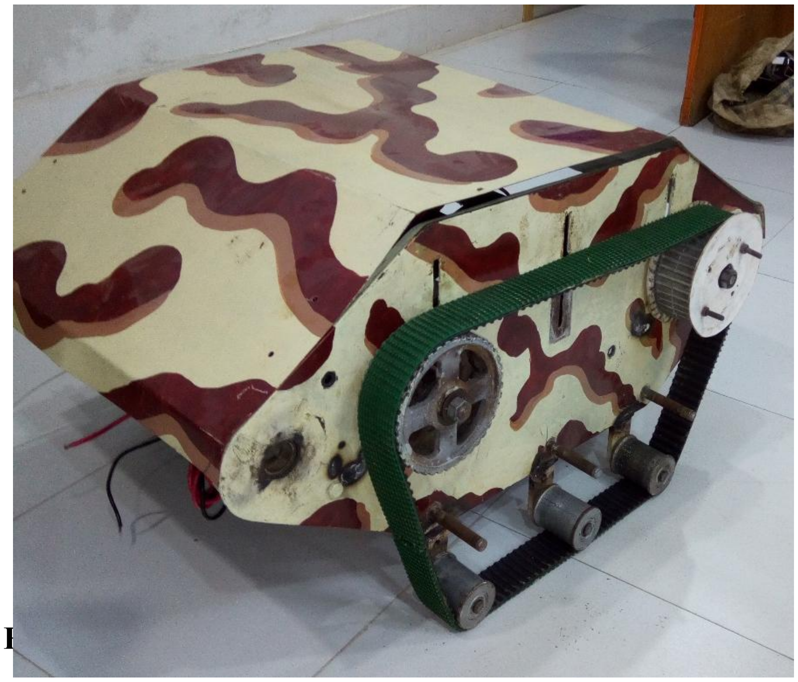


Figure 39 Internal diagram with motors



F

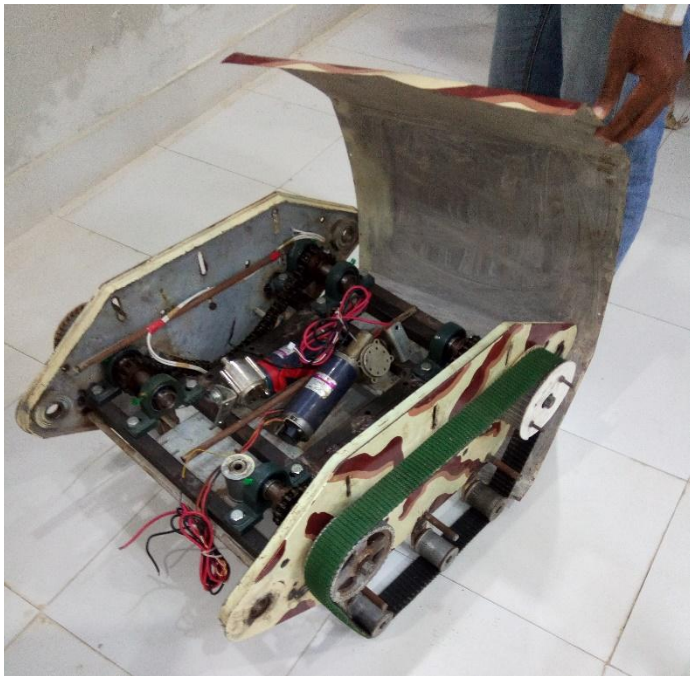


Figure 41 Actual internal design of tank based robot version 2

7.4 Wheeled based robot version 1.0

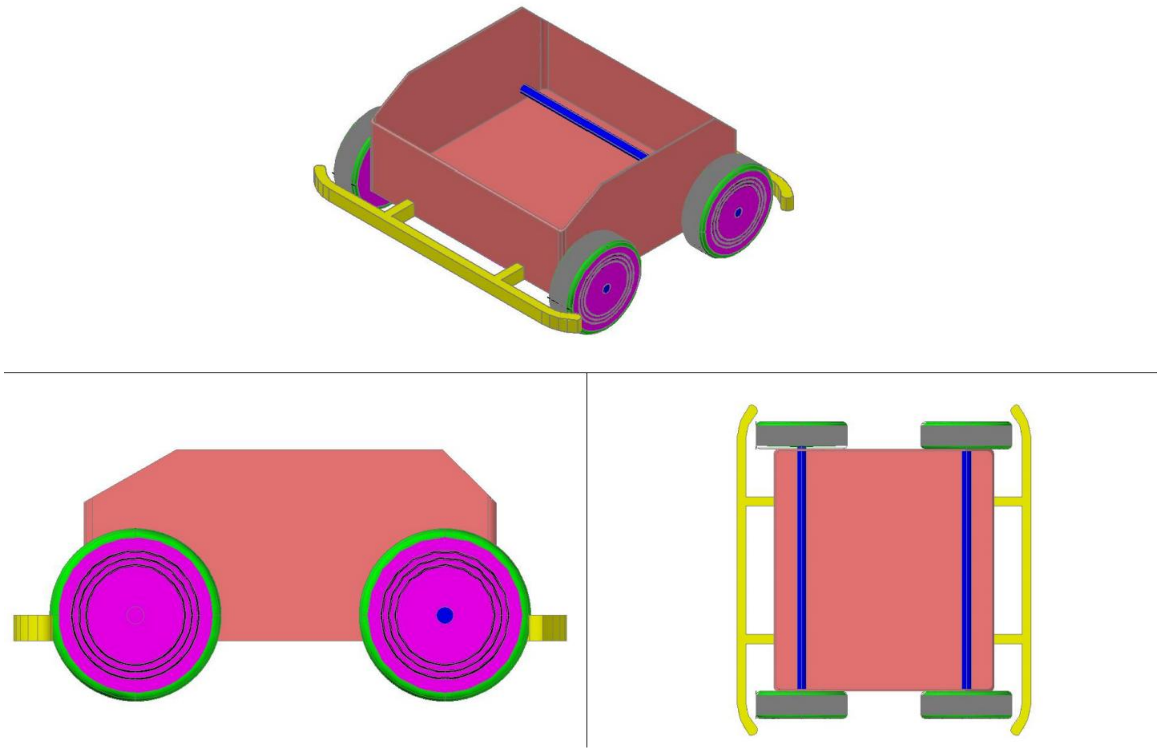


Figure 42 Wheeled based robot version 1.0

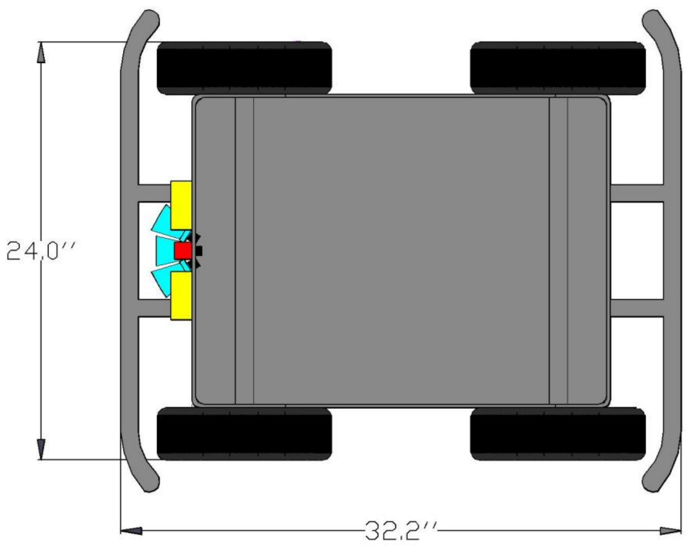


Figure 43 Wheeled based robot version 1.0 with sensor placement

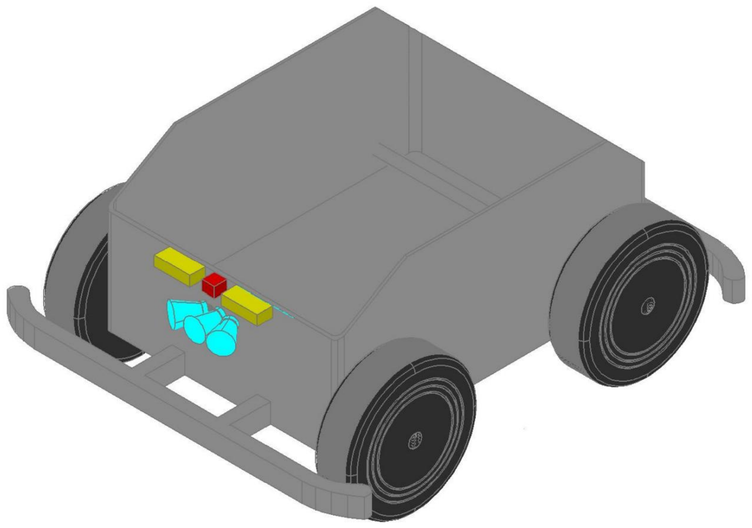


Figure 44 3D view of wheeled based robot version 1.0 with sensor placement

7.5 Wheeled based robot version 2.0

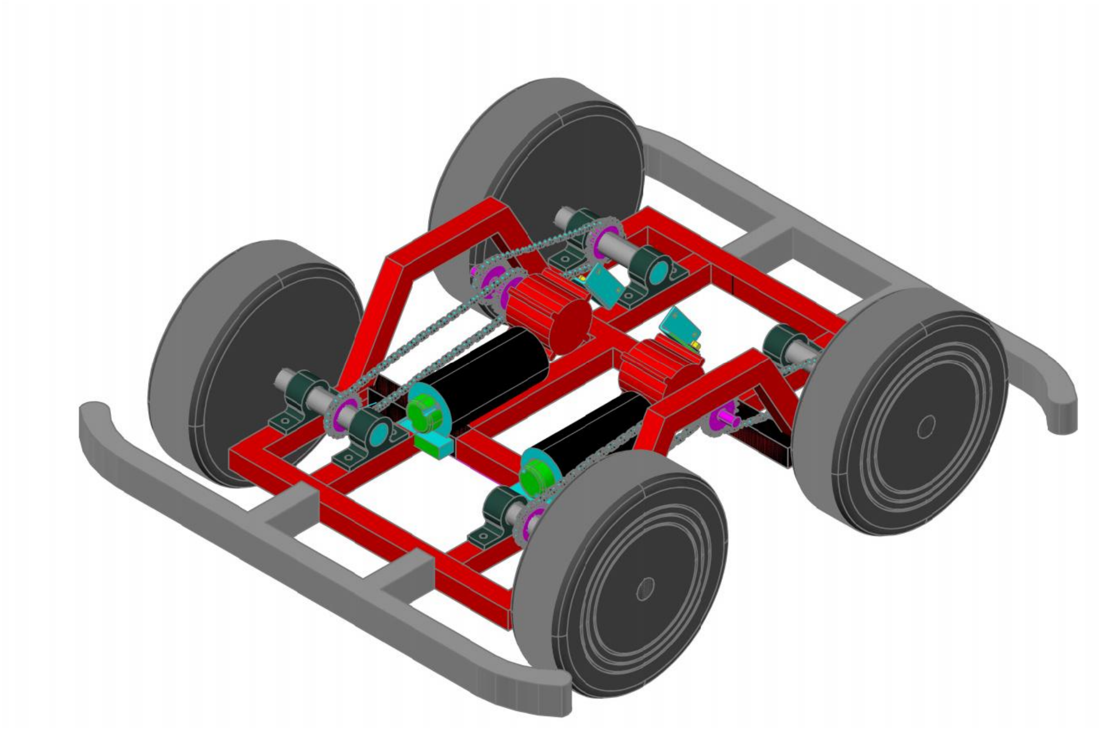


Figure 45 Wheeled based robot version 2.0

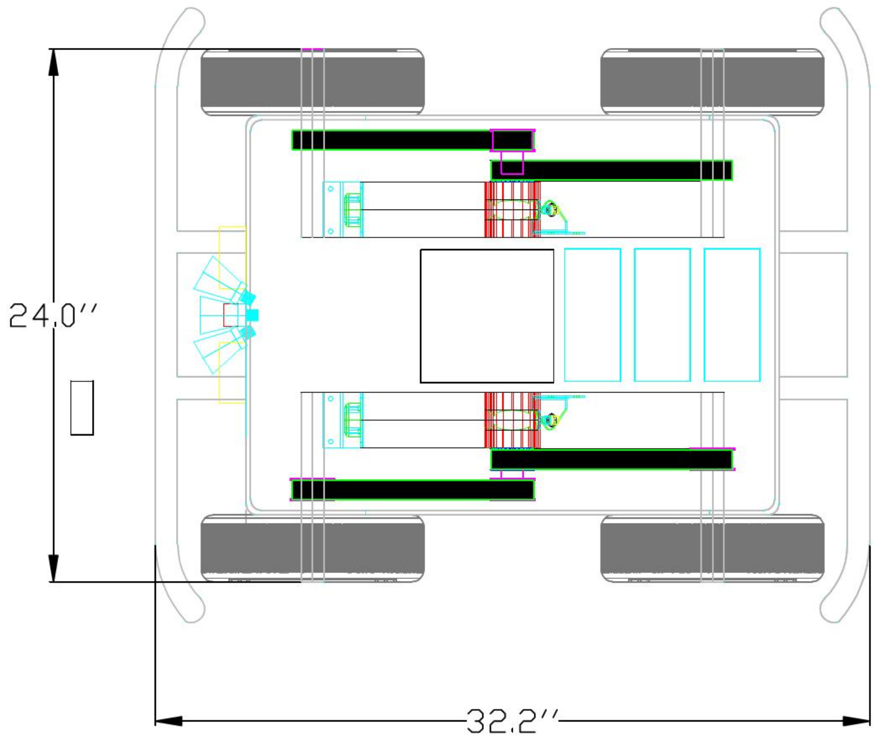


Figure 46 Wheeled based robot version 2.0 with sensor placement

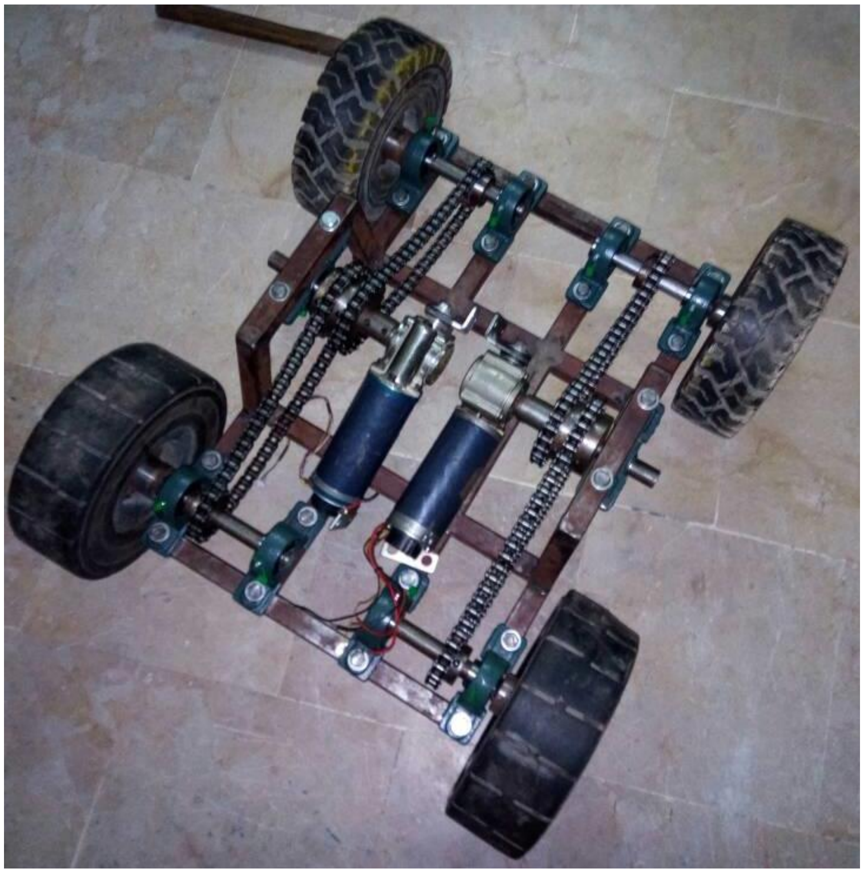


Figure 47   Actual model of wheeled based robot version 2.0