

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

Fourth Deliverable

Report submitted on 5th May 2017

PI: Dr. Muhammad Bilal Kadri

Co-PI: Dr. Tariq Mairaj Rasool Khan

Table of Contents

Chapter 1.	Introduction.....	4
Chapter 2.	Using ROS with MUSAFIR (the miniature robot).....	5
2.1	Robotics for Academia.....	5
2.1.1	ROS (Robot Operating System).....	5
2.1.1	MUSAFIR Robot.....	6
2.1.2	MUSAFIR and ROS Integration Goals	7
2.2	System Requirements.....	7
2.2.1	System Level Components	7
2.2.2	Network Requirements	8
2.2.3	Using with Windows PC.....	8
2.2.4	Using with Linux PC	8
2.3	ROS Setup and Configuration.....	9
2.3.1	IP Address Configuration	9
2.3.2	IP Address Configuration	10
2.3.3	ROS Initialization	10
2.4	TEST Run.....	11
2.4.1	TeleOperation of Robot	11
2.4.2	Robot Pose Visualization.....	11
2.5	ROS and MATLAB	12
2.5.1	roscore on MATLAB.....	12
2.5.2	MATLAB as ROS Node.....	12
Chapter 3.	Problems and challenges with the design of MUSAFIR	13
3.1	MUSAFIR Robot	13
3.1.1	Features	13
3.2	MUSAFIR Variants.....	14
3.3	MUSAFIR V1 (a and b).....	14
3.3.1	MUSAFIR V1 (a) Specifications.....	14
3.3.2	MUSAFIR v1 (b) Specifications/Features.....	15

3.4	MUSAFIR V1 Issues	16
3.4.1	Uneven Surface	16
3.4.2	Odometry Error	16
3.4.3	Power Circuitry	16
3.4.4	nRF Modules	16
3.4.5	Acrylic Base	16
3.4.6	Road Grip	16
3.4.7	Outdoor Capability	16
3.5	MUSAFIR v2	16
3.5.1	Addition of Cameras	17
3.5.2	Addition of Back Panel	17
3.5.3	Sonar Sensor Panel	17
3.5.4	Tire & Wheel	17
3.5.5	Motors	17
3.5.6	Size	17
3.5.7	Weight	17
3.5.8	Power Circuitry	17
3.6	MUSAFIR v2 Issues	17
3.6.1	Speed	17
3.6.2	Weight	17
3.6.3	LIDAR issues	18
3.6.4	Wiring Issues	18
3.6.5	Drive Mechanism	18
3.6.6	Ground Clearance	18
Chapter 4.	Sensor fusion of UT, TIC and LIDAR for map generation	19
4.1	Image processing using Stereo Vision	19
4.1.1	Experimental Setup	19
4.1.2	Results	21
4.1.3	Live Video Capturing	21
4.1.4	Two VGA Cameras on single Pi board	22
4.1.5	Real time Stereo Depth Perception	22

4.1.6	Results of Real Time Stereo Vision.....	22
4.1.7	Data Transmission	23
4.2	Stereo Results at actual fire site	23
4.2.1	Map Genration based on MaxSonar sensor	23
4.2.2	Three MaxSonar Sensors on single raspberry pi	26
4.3	Flir Thermal Imaging Camera.....	27
4.3.1	Progress with the sensor.....	28
4.4	Sensors Fusion for implementation of Control algorithms:	28
4.5	Development of 3D environment for Mobile Robots using UT and TIC:	28
Chapter 5.	Problems and challenges with the design of MUSAFIR	30
5.1	Introduction	30
5.2	Redesigning the MUSAFIR	30
Chapter 6.	Sensor Fusion Techniques for determining the exact geo-location	31
Chapter 7.	Problems in design and development of the mechanical structure of IMR	47
7.1	Initial design of the IMR	48
7.2	Proposed modifications in the design of the IMR.....	50

Chapter 1. Introduction

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

1) Problems faced in interfacing the DC motor with the IMR.

Test performance of the IMR in an artificial environment where some objects will be put on fire.

Complete design of the IMR which is capable of performing basic fire fighting tasks.

A prototype capable for initial testing.

2) All the bottlenecks faced in the hardware interfacing will be reported.

Chapter 2. Using ROS with MUSAFIR (the miniature robot)

2.1 Robotics for Academia

Robotics is a multi-disciplinary field and for any researcher to do robotics related work, there comes a time when it is time to move from simulation to the real world and verify that the system behaves as it is supposed to and is controllable and repeatable. For such hardware based work, there are several options which depend on the need, complexity requirement, cost factor and allocated budget. As any such ready to use solutions are not made in the country and buying from US or other foreign countries means spending exorbitant sums of money for very few and small robots. Also, buying readymade robotic kits results in no advancement of knowledge with respect to design and development of the robotics hardware.

In this chapter, we are providing detailed information about the robots that we have developed and step by step procedure for using them for research work. Our work comprises of hardware robots that we have indigenously designed in our lab and are called “MUSAFIR” Robots. The robots are controllable by several different systems and languages as the Robots are ROS compatible, this allows for use of C, C++, Python, MATLAB among many other options for command and control of the MUSAFIR Robots.

2.1.1 ROS (Robot Operating System)

ROS or Robot Operating System is a software framework for working with robots and automated systems. ROS allows for integration of multiple complex systems and computers into one network where all relevant information of sensors, actuators and controllers is accessible to all services and running processes. A generic block diagram of a ROS based robot is given in figure 2.1 as an example.

ROS Control

Data flow of controllers

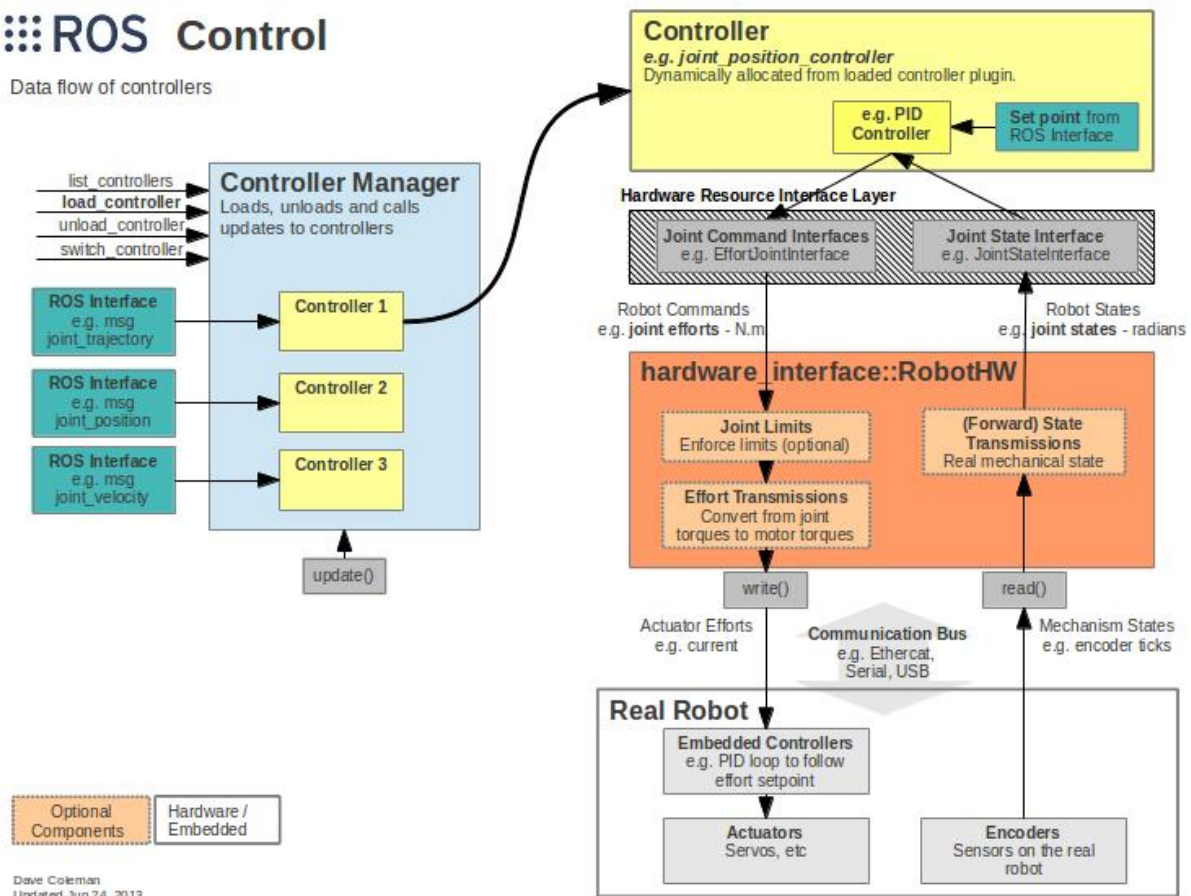


Figure 2.1: Generic ROS based Robotics System Components\

2.1.1 MUSAFIR Robot

MUSAFIR Robots are indigenous robots developed at IMR LAB PAF KIET for the purpose of academic research work. These robots are developed as either blank platforms with motor control and wireless communication or as full-fledged sensory platforms for indoor and outdoor localization and mapping based research work.

The Sensor/Actuator suite integrated with MUSAFIR Robot are:

- 2x DC Motors, Geared with Encoders for Differential Drive
- 2x Motor Drivers
- Odometry via Encoders
- 3-Axis Accelerometer
- 3-Axis Gyroscope
- 3-Axis Magnetometer
- GPS Module

- 360 Degree LIDAR Sensor - RP LIDAR A2
- Ultrasonic Rangefinders
- Microphone
- Speaker
- Temperature Sensors
- Thermal Imaging Camera
- 2x Camera Stereo Configuration on Front
- Raspberry PI 3 Computer for High Level Processing
- Wi-Fi Connectivity
- 8-Bit Microcontroller for Low-Level Control
- 12V Recharge-able Battery
- Power Failure Protection Circuitry - Over Current / Under Voltage

2.1.2 MUSAFIR and ROS Integration Goals

As mentioned above, the MUSAFIR robot is a complex system with high integration of various modules and components. Writing control software for such a system from scratch is a huge project on its own. However, we tested each component and module individually and then integrated these to serve as one robotic platform by means of integrating with the ROS Framework. Our goal is to make MUSAFIR fully ROS compatible and have a ready to use driver available in ROS repository for the MUSAFIR robot.

2.2 System Requirements

In order to be able to use MUSAFIR in a lab environment, the following requirements must be met and it is expected that the user is familiar with programming languages (C, C++, Python), familiarity with Linux Environment, Scripting, MATLAB and Computer Networking concepts.

2.2.1 System Level Components

For working with MUSAFIR Robots, the minimum requirement is that there be a working Wi-Fi Network to which the MUSAFIR Robot will connect and publish all information, as shown in figure 2.2. A USER PC is also required which will provide a user interface to the user. An additional LINUX Server system with high performance is recommended for most of high level processing, map making and data logging.

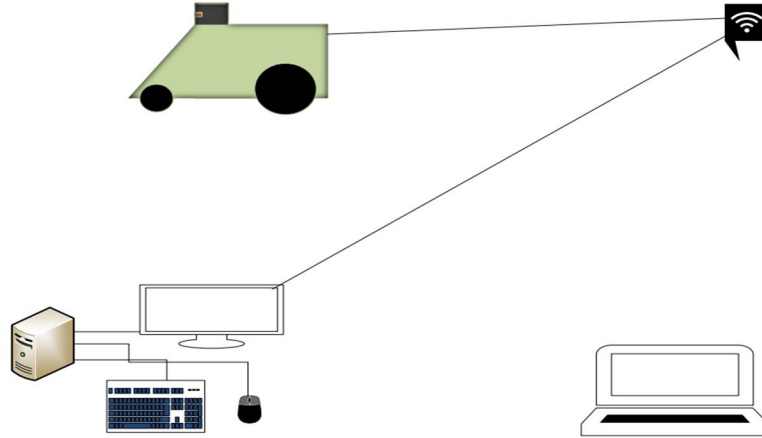


Figure 2.2: Minimum Requirement for MUSAFIR/ROS based System

2.2.2 Network Requirements

A working Wi-Fi Network with Internet access is required. ROS requires proper network connectivity between all systems with bi-directional communication and no lag or latency issues. We have tested the system with a D-Link Wireless Router DIR-822 in the lab and a Zong 4G Modem/Wi-Fi Router in outdoor setting. From ROS Wiki, the network requirements are given as;

ROS is a distributed computing environment. A running ROS system can comprise dozens, even hundreds of nodes, spread across multiple machines. Depending on how the system is configured, any node may need to communicate with any other node, at any time.

As a result, ROS has certain requirements of the network configuration:

- There must be complete, bi-directional connectivity between all pairs of machines, on all ports.
- Each machine must advertise itself by a name that all other machines can resolve.

2.2.3 Using with Windows PC

As ROS itself is mainly supported only on Linux based systems, running ROS in Windows requires MATLAB with Robotics Toolbox installed, we have tested with MATLAB 2015b and 2016b both with Robotics Toolbox installed, which has a ROS node built-in and can make a connection between the MATLAB environment and the running ROS Nodes and Topics.

2.2.4 Using with Linux PC

ROS installation on Linux is straightforward, depending on the Linux distribution that is in use, the installation can be from either the apt repositories where a pre-compiled version is available or by compiling from source files directly available at the online official ROS repository.

ROS suggests installation of Kinetic Kame, the current latest variant, which also has LTS (Long Term Support) on Ubuntu 16.04, which is also the latest LTS version from Canonical. To install ROS, follow the below steps.

Run these commands in a Terminal Window

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu
$(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-
latest.list'

sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80
--recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116

sudo apt-get update
sudo apt-get install ros-kinetic-desktop-full
sudo rosdep init
rosdep update

echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc
source ~/.bashrc
sudo apt-get install python-rosinstall
```

2.3 ROS Setup and Configuration

After installation ROS dependencies need to be added to the environment and other ROS variables needed to be updated and added/appended to the system PATH variables for proper working of ROS executables.

Run these commands in a Terminal Window

```
sudo rosdep init
rosdep update
echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc
source ~/.bashrc
sudo apt-get install python-rosinstall
```

Running these commands will initialize the ROS dependencies in the system and will also additionally add the ROS variables to the bashrc file which loads whenever a new terminal is opened so that ROS variables are always available. python-rosinstall allows for easy installation of ROS packages from the official online repositories.

2.3.1 IP Address Configuration

IP Address of the system needs to be known where the roscore will run, roscore is the main ROS executable which does the handshaking between all the nodes and keeps a list of running nodes and topics. To get the IP address of the system, use the command `ifconfig` on Linux. Make

sure to note IP address of the correct Ethernet/Wi-Fi interface as in a typical system there are several network interfaces available.

For the rest of this report, we will assume that the main system where ROS (roscore) is running has the IP address: 192.168.100.161, the user Laptop which has Linux has the IP address 192.168.100.162 and the Musafir Bot has the IP address 192.168.100.165

2.3.2 IP Address Configuration

For ROS, there needs to be one system which will run roscore and the other systems including the system (Raspberry Pi) on the robot needs to be in the same network and IPs of all systems known. The main system IP address where roscore will run needs to be added/exported as environment variable to Linux terminals as ROS_MASTER_URI using the below command

```
export ROS_MASTER_URI=http://IP OF SYSTEM:11311
```

(where 11311 is the default port number of ROS)

2.3.3 ROS Initialization

For starting with ROS, on every system in the network where any ROS node will be working, there needs to be some environment variables present, hence the below commands should be done on the systems whenever a new terminal is started when working with ROS

On Main LINUX System – SERVER - IP is 192.168.100.161

```
export ROS_MASTER_URI=http://192.168.100.161:11311
export ROS_HOSTNAME=192.168.100.161
source ~/catkin_ws/devel/setup.bash
```

On PI, do this - IP of Pi is 192.168.100.165

```
export ROS_MASTER_URI=http://192.168.100.161:11311
export ROS_HOSTNAME=192.168.100.165
source ~/catkin_ws/devel/setup.bash
```

On USER LAPTOP LINUX System – USER - IP is 192.168.100.162

```
export ROS_MASTER_URI=http://192.168.100.161:11311
export ROS_HOSTNAME=192.168.100.162
source ~/catkin_ws/devel/setup.bash
```

Upon running the above commands starting ROS nodes on any of the systems or visualizing data becomes straightforward. On Main SERVER System, we run the roscore and the mapping server, all in different terminals.

```
roscore
roslaunch hector_mapping mapping_default.launch
```

On the Robot System, Raspberry Pi, we run the following nodes, all in different terminals.

```

cgps -s
roslaunch i2c_imu i2c_imu_auto.launch
roslaunch gspd_client gspd_client
roslaunch rplidar_ros rplidar.launch
python ~/catkin_ws/src/musafir/motor.py

```

On the USER Laptop System, we run the teleoperation node and then we can also run any visualizations with rqt plot and rviz nodes.

```

roslaunch teleop_twist_keyboard teleop_twist_keyboard.py

```

2.4 TEST Run

After running the previous commands in the various terminal windows across the different systems, the MUSAFIR Bot is ready for a test run,.

2.4.1 TeleOperation of Robot

The TeleOp Node terminal window provides the input required for the user to move the robot manually.

```

Terminal File Edit View Search Terminal Tabs Help
roscore http://192.168.8.100:11311/
imrlab@IMRLAB1: ~
imrlab@IMRLAB1:~$ roslaunch teleop_twist_keyboard teleop_twist_keyboard.py
Reading from the keyboard and Publishing to Twist!
-----
Moving around:
  u   i   o
  j   k   l
  m   ,   .

For Holonomic mode (strafing), hold down the shift key:
-----
  U   I   O
  J   K   L
  M   <  >

t : up (+z)
b : down (-z)

anything else : stop

q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%

CTRL-C to quit

currently:   speed 0.5      turn 1

```

The TeleOp node publishes velocity commands as standard Twist Messages.

Published Topic

/cmd_vel

Type: geometry_msgs/Twist -

http://docs.ros.org/api/geometry_msgs/html/msg/Twist.html

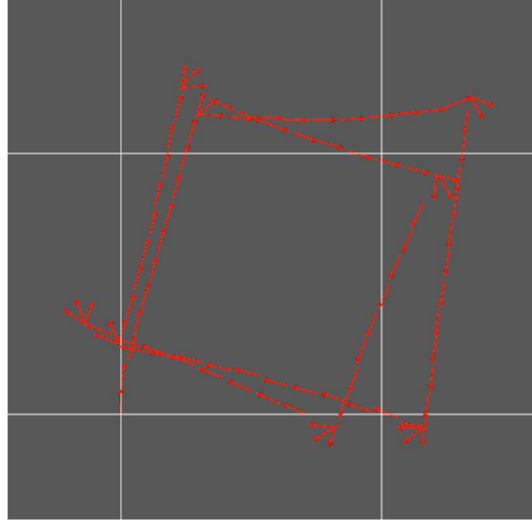
This expresses velocity in free space broken into its linear and angular parts.

Vector3 linear

Vector3 angular

2.4.2 Robot Pose Visualization

The robot Pose can be visualized in the rviz application. The robot Pose is visualized as an arrow in 2D space, pointing towards where the robot front is facing and starting from where the robot base is located.



2.5 ROS and MATLAB

Running ROS along with MATLAB requires that either MATLAB be the main ROS node, roscore or be part of the ROS network as a regular ROS node. Installing MATLAB with the Robotics ToolBox also installs ROS support in MATLAB versions 2015 and 2016.

2.5.1 roscore on MATLAB

The MATLAB command for running ROS as the MASTER node or as a normal ROS node is the same, depending on the parameters, the node behaves differently.

```
rosinit()
```

initializes ROS with roscore on the same system as MATLAB

2.5.2 MATLAB as ROS Node

By providing an IP address to the rosininit command and NodeHost parameter with IP of MATLAB system, the rosininit creates a MATLAB ROS node and connects to the MASTER ROS system.

```
rosinit('192.168.100.161', 'NodeHost', '192.168.100.166')
```

Chapter 3. Problems and challenges with the design of MUSAFIR

In order to design and build robots for the IMR project, we need to test several algorithms and work on several different sensing techniques. In order to be able to reliably and repeatedly test the algorithms, we needed a test robotic platform. Musafir is the result of that need.

3.1 MUSAFIR Robot

3.1.1 Features

The Musafir design is mainly inspired from some similar academic commercial bots, which are too expensive. We selected to mimic Khepera robots as these robots are also the ones used by other robotics labs around the world.

Musafir robot is expected to be built in a fraction of the cost and we are working to make 4 such robots, so that most of the collaborative control, SLAM and swarm algorithms can be tested, most configurations require at least 3 to 4 robots.

In order to design and build a new robotics platform, specifications need to be written and laid out before starting the work. Musafir robots are expected to have several built-in sensors and communication methods.

- Re-Chargeable Battery
- Round/Circular/Symmetric Design
- IR/Sonar sensor skirt around the bot
- Differential Drive Motors with Encoders
- IMU (Gyro + Accelerometer) + Compass Sensor
- Wireless Communication
- Motor Driver - MOSFET Based
- On-board computer/camera for further processing and SLAM – OPTIONAL

With these specifications, the design was broken into 3 main steps, initially a very basic robotic platform with motors and wireless control and minimum sensors will be implemented, then the design will be refined and more sensors (sensor skirt) will be added to the bot, finally, in the third stage the robot will be equipped with a computer (Raspberry Pi) and a camera, given time and budget constraints, this step will be done last and is optional.

3.2 MUSAFIR Variants

Over the course of this project, IMR Lab has produced three variants of MUSAFIR up till now, named as MUSAFIR V1a, MUSAFIR V1b and MUSAFIR V2 respectively. In the next section, we will go through all of them sequentially highlighting specifications of them and the issues faced during implementation phase.

3.3 MUSAFIR V1 (a and b)

There are two models of MUSAFIR robot, v1 and v2, however while making v1, we implemented changes and improvements as we were designing and using the MUSAFIR robots, hence even the v1 has two variants, v1 (a) and v1 (b).

3.3.1 MUSAFIR V1 (a) Specifications

3.3.1.1 Accurate Odometry

The motors selected to be used in Musafir are DC Gear motors with built-in encoders, these motors were selected after a thorough selection process and criteria refining, we needed the motor to run at 12V and have strong torque with slow outer speed. For odometry calculation, encoders were necessary. Encoders can be externally connected to a motor as well, but with built-in encoders, high precision is achieved in determining motor shaft rotation. The selected motors were then imported from china.

3.3.1.2 RF Link

The Radio Frequency link is being operated at 2.4 GHz with data rate of 250 Kilobits per second. The module in use is nRF24L01+. Range of the transceiver is 0.8 KM.

3.3.1.3 Battery Capacity

In MUSAFIR V1, 4S 2800 mAH batteries were used. These are Lithium polymer RC batteries (as indicated in the picture).

3.3.1.4 2-Tiered 8-Bit Controller

The Musafir electronics were designed in a very modular manner with the ease of usage and assembly in mind along with separation of tasks among boards and the different controllers, so that programming the firmware would then be distributed as separate tasks. The electronics is basically distributed into Power, Motor Drivers, Motor Controller, Sensor Controller, Main Controller and RF Controller (on PC Side).

3.3.1.5 9-axis IMU

MPU-9250 is a multi-chip module integrated into a single QFN package. One die houses the 3-Axis gyroscope and the 3-Axis accelerometer. The other die houses the AK8963 3-Axis magnetometer. Hence, the MPU-9250 is a 9-axis Motion Tracking device that combines a 3-axis

gyroscope, 3-axis accelerometer, 3-axis magnetometer and a Digital Motion Processor. MPU-9250 features three 16-bit analog-to-digital converters (ADCs) for digitizing the gyroscope outputs, three 16-bit ADCs for digitizing the accelerometer outputs, and three 16-bit ADCs for digitizing the magnetometer outputs.

3.3.1.6 Motor Modes

With improved motor driving circuit, there are two modes of operation: PWM without feedback, and constant velocity mode)

3.3.2 MUSAFIR v1 (b) Specifications/Features

3.3.2.1 Addition of Raspberry Pi

The major change from MUSAFIR V1A to V1B is the addition of Raspberry Pi, a small palm-sized computer. This addition made other arduino boards redundant so the Robot Controller Arduino Mega and RF Controller Arduino which was being used for serial communication between PC and MUSAFIR has been removed from the platform.

3.3.2.2 Added GPS

In MUSAFIR V1B, GPS has been added and interfaced with Raspberry Pi to get better positioning indoor and outdoor. After 10-12 seconds of triangulation process it produces lat-long data with +/-10m accuracy.

3.3.2.3 Added LIDAR

The RPLIDAR A2 360° Laser Scanner is the next generation of 2D LIDAR. The RPLIDAR A2 adopts low cost laser triangulation measurement system developed by SLAMTEC, and therefore has excellent performance in all kinds of indoor environments and outdoor environments without direct sunlight exposure. It can take up to 4000 samples of laser ranging per second with high rotation speed.

3.3.2.4 IMU filters

An inertial measurement unit, or IMU, measures accelerations and rotation rates, and possibly earth's magnetic field, in order to determine a body's attitude. Three basic filter approaches are applied, e.g. the complementary filter, the Kalman filter (with constant matrices), and the Mahony&Madgwick filter.

3.3.2.5 Sensor Fusion

In MUSAFIR V1B, we have been capable of fusing IMU, odometry and GPS data together in hardware, to get accurate whereabouts of the robot.

3.3.2.6 Tire Width Increased

To obtain grip on indoor environment, tire contact area with floor has been doubled.

Size & Weight

Length: 257 MM, Width: 255 MM, Height: 160 MM

3.4 MUSAFIR V1 Issues

3.4.1 Uneven Surface

Upon testing the MUSAFIR outside lab environment, we have noticed that the slippage increases in uneven surfaces and in some places MUSAFIR V1 gets stuck.

3.4.2 Odometry Error

Due to motors backlash, there were unavoidable accumulated errors in the odometry readings present after some time.

3.4.3 Power Circuitry

While testing, it has been noticed that MUSAFIR V1 A had rudimentary power circuit due to which one Raspberry Pi got burnt. Therefore, in MUSAFIR V1 B, power circuitry has been optimized to avoid any surges during the stall mode.

3.4.4 nRF Modules

In MUSAFIR V1 A, nRF Modules has been used for communication link layer. While nRF is a choice in the first iteration, it has later been realized that the connection issues were present so in MUSAFIR V1 B, nRF has been replaced with on-board Wi-Fi present in Raspberry Pi, thus this issue has been resolved successfully.

3.4.5 Acrylic Base

As we know, the base of MUSAFIR V1 Platform was made by Acrylic plastic sheet and hence was only suitable for limited indoor testing purposes only as the base was not strong enough to withstand any possible collision with obstacles or other bots.

3.4.6 Road Grip

Due to lightweight structure and toothless tires, MUSAFIR V1 does not have the required road grip.

3.4.7 Outdoor Capability

As stated earlier, MUSAFIR V1 was workable only inside a lab environment with tiled surface. When exposed to daylight, the proximity sensors produced erroneous output. So the MUSAFIR V1 outdoor operating capability is limited.

3.5 MUSAFIR v2

MUSAFIR v2 Specifications

3.5.1 Addition of Cameras

To obtain visual feedback from front of MUSAFIR V2, 2 Cameras are placed side by side to get increased field of view. These cameras are interfaced with ROS and can be viewed over the network.

3.5.2 Addition of Back Panel

On the back of MUSAFIR V2, one camera with tilting mechanism, LCD and push buttons for reset and power has been added.

3.5.3 Sonar Sensor Panel

Three Sonar sensors added mounted on the front panel of MUSAFIR V2 for optimum obstacle detection and avoidance.

3.5.4 Tire & Wheel

To provide additional grip and traction, acrylic wheel has been replaced with customized Teflon wheel with rubber grip tires on top. This enhancement solves the skid issue to much extent.

3.5.5 Motors

In MUSAFIR V2, the motors have been replaced from generic DC motors to a much powerful German made Faulhaber 3257L024CR. This gives extra torque and power required to move heavier structure of the next MUSAFIR.

3.5.6 Size

Length: 440 MM, Width: 390 MM, Height: 270 MM

3.5.7 Weight

MUSAFIR V2 Weighs approximately 11.4 KG.

3.5.8 Power Circuitry

In the improved circuitry for power management, low voltage detection has been introduced and hence there are no abrupt restarts during the operation.

3.6 MUSAFIR v2 Issues

3.6.1 Speed

The MUSAFIR V2 speed is low as of writing this report. i.e. 2.5 kmph. Progress is underway to increase the speed in 0 degree inclination.

3.6.2 Weight

Albeit the MUSAFIR V2 is heavier than its precedent, there is still a slight skidding issues observed during the movement. This can be reduced by introducing dead weight mounted on the robot chassis.

3.6.3 LIDAR issues

In outdoor daylight, LIDAR gave erroneous readings, this could be resolved by putting a shelter on top of LIDAR.

3.6.4 Wiring Issues

Initially there were some wiring issues but those have been resolved afterwards.

3.6.5 Drive Mechanism

In MUSAFIR V2, there is lack of differential drive, and a mid-point between two back wheels is the center of rotation.

3.6.6 Ground Clearance

Chassis of MUSAFIR V2 is held low and due to which there is not much ground clearance available for outdoor movement.

Chapter 4. Sensor fusion of UT, TIC and LIDAR for map generation

Image processing results for stereo vision, UT sensor, LIDAR and FLIR will be discussed. The main objective is to help the IMR navigate in an unknown environment.

4.1 Image processing using Stereo Vision

4.1.1 Experimental Setup

In this work, the actual stereo cameras were not used. We use two different cameras of same specifications separated by a baseline distance B as shown in fig. 1.

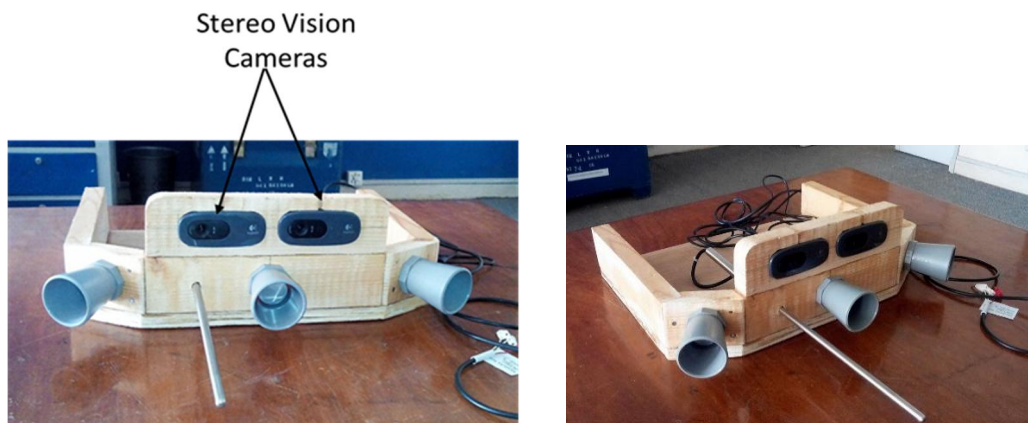


Fig. 1. VGA Cameras separated by a baseline distance B

The images are captured from the camera. They are then converted into grayscale for proper matching. Left and right images from left and right camera respectively are used to determine the disparity between the images. This disparity is determined by using the correlation and block matching mechanism as shown in fig. 2

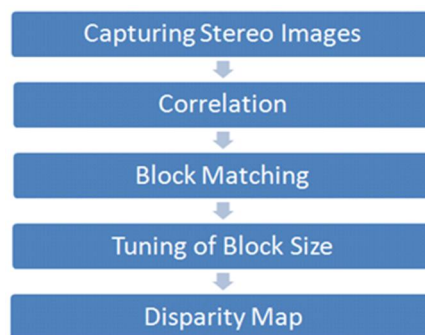


Fig. 2. Process flow

The images taken from left side and right side cameras are shifted horizontally because of the baseline distance (B) between the cameras. The disparity between these images can be calculated using this horizontal shift. The next step is to perform the basic block matching. For every pixel in the right image, we extract the 7-by-7-pixel block around it and search along the same row in the left image for the block that best matches it. The basic block matching does well, as the correct shape of the stereo scene is recovered. However, there are noisy patches and bad depth estimates everywhere, especially on the ceiling. These are caused when no strong image features appear inside of the 7-by-7-pixel windows being compared. Then the matching process is subject to noise since each pixel chooses its disparity independently of all the other pixels. A noisy disparity image can be improved by introducing a smoothness constraint using dynamic programming.

The depth (Z) from the image pair can be estimated using (1)

$$Z = \frac{fB}{x1 + x2} \quad (1)$$

Where f is the focal length of the lens used, B is the baseline distance between the two cameras, (x1+x2) is the disparity as shown in fig. 4.

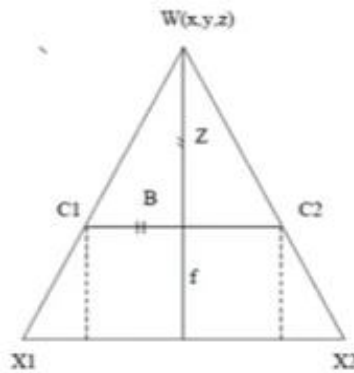


Figure 4 Triangulation

The left and right images are compared using the Sum of Square Difference (SSD) between each pixel of the window as in (2).

$$SSD = \sum (I_{left} - I_{right})^2 \quad (2)$$

Where Ileft and Iright are the left and right images respectively

4.1.2 Results

The test image taken in the lab is shown in fig. 9 and there corresponding depth image is shown in fig. 10. The pictures taken at lab showed promising results of disparity map. The white color shows object is nearer however black color maps to farther objects.



Fig. 9 Test image taken in the lab



Fig. 10 Depth image of the test image of figure 9

4.1.3 Live Video Capturing

After successful image processing on the images which we captured from both VGA cameras serially, we then captured live video from both the VGA cameras separately so that we can reduce the processing time of the stereo depth perception.

The Python code for live video capturing from the VGA camera is attached in **Annexure-A**.

4.1.4 Two VGA Cameras on single Pi board

Two VGA cameras were then run on a single raspberry pi and the frames from both these cameras were then captured serially. The delay between the frames captured from both the cameras was 3mSec.

The Python code for running two VGA cameras on single raspberry pi board and capturing frames from those VGA cameras is attached in **Annexure-B**.

4.1.5 Real time Stereo Depth Perception

Instead of saving those captured frames into the memory of raspberry pi, we used those frames directly into depth perception code. So that we can increase the processing speed and can obtain the maximum frames possible in least possible time.

The Python code for Stereo Depth perception is attached in **Annexure-C**.

4.1.6 Results of Real Time Stereo Vision

The results of stereo depth perception are as under,



Fig 11(a): Real time Stereo Results



Fig 11(b): Real time Stereo Results

4.1.7 Data Transmission

The disparity results which we got from VGA cameras were then transferred to the base station over the WiFi in the real time. These results were then saved at base station with real time stamp. These images were then used to populate the map grid in real time.

4.2 Stereo Results at actual fire site

The results obtained at real outdoor environment are shown below:

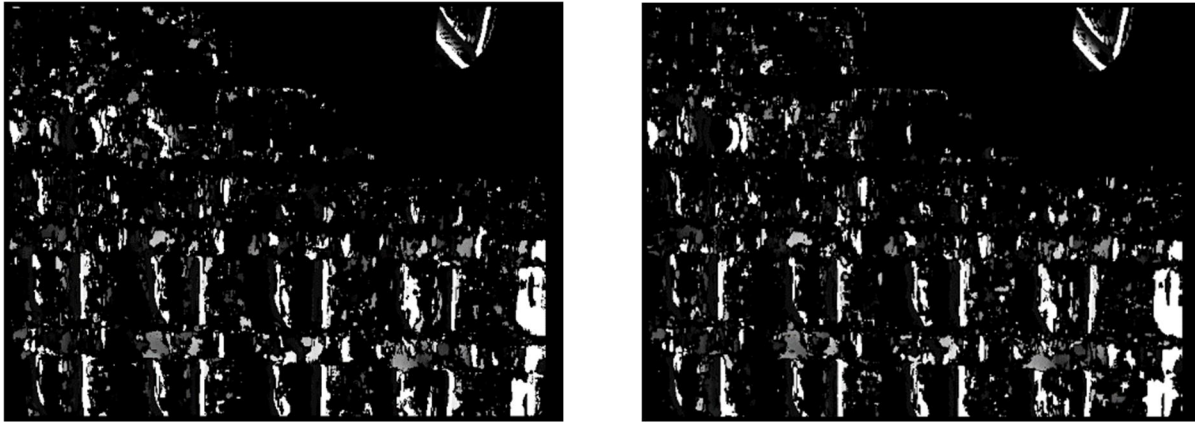


Fig: Stereo results in Outdoor environment

The results in the outdoor environment were not accurate as compared to the lab environment. The main reason for this inaccuracy is that the computer vision techniques are not generic. These techniques/codes need to be tuned according to the environment. They only work in suitable environment. The inaccuracy occurs due to stereo correspondence. An image is comprised of hundreds of thousands of pixels. For a given pixel in the reference image, once its corresponding point is located in the target image, the disparity for that pixel is derived. A complete stereo correspondence algorithm will compute disparity for each pixel that appeared on both images. Moreover, a dense disparity map due to shadow effect usually contains background noise that makes it complicated for obstacle-specific depth calculation in smoky environment.

4.2.1 Map Generation based on MaxSonar sensor

MaxSonar sensor mounted on robot face was then rotated over an angle of 180° from 0° to 180° with step size of 10° to generate initial maps of the environment.

The actual image of the environment is as shown



Fig 4.8: MaxSonar Testing Environment

The Map which we generated based on the readings from MaxSonar sensor at its initial position was as

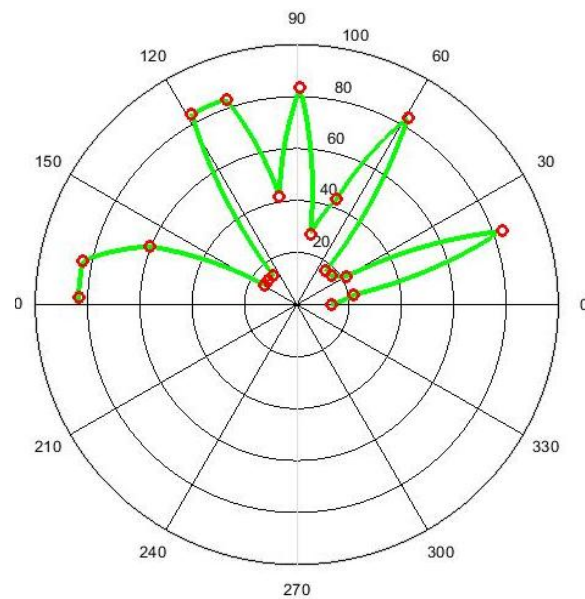


Fig 4.9: Initial Map based on UT sensor

The robot was then moved 6 inch further and a new set of readings were taken from 0° to 180° with step size of 10° . The Map which we got this time was as shown under

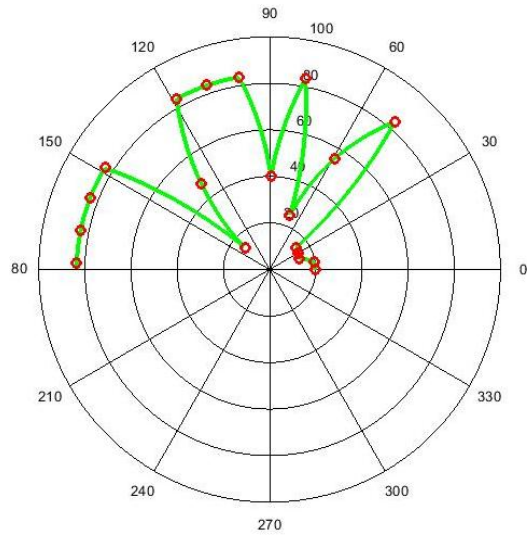


Fig 4.10: Map at Position 1

The same procedure was repeated at four unique positions of the robot and the results which we got are as shown

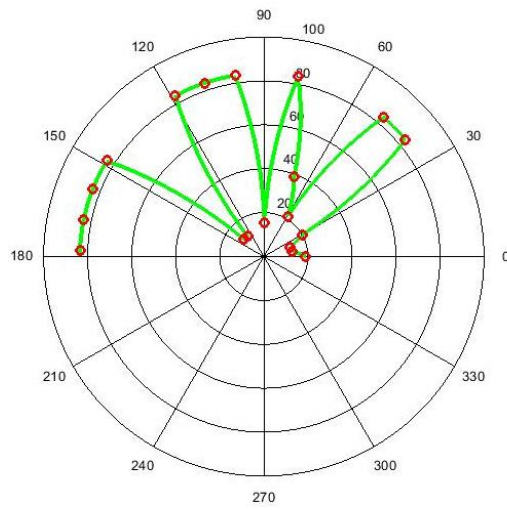


Fig 4.11: Map at Position 2

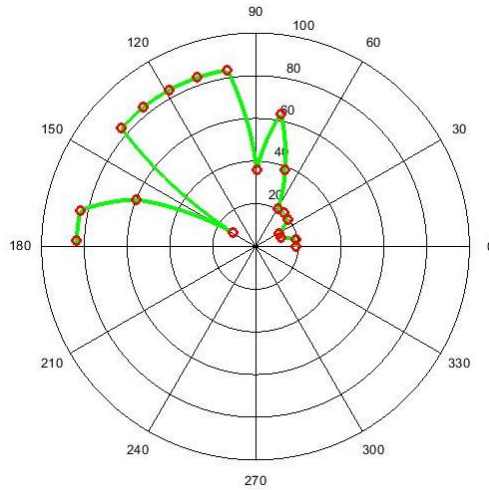


Fig 4.12: Map at Position 3

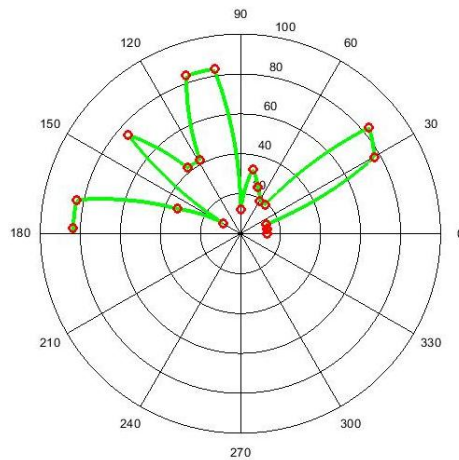


Fig 4.13: Map at Position 4

4.2.2 Three MaxSonar Sensors on single raspberry pi

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

The wiring diagram of three MaxSonar sensors with raspberry pi 3 is as attached in fig 4.14

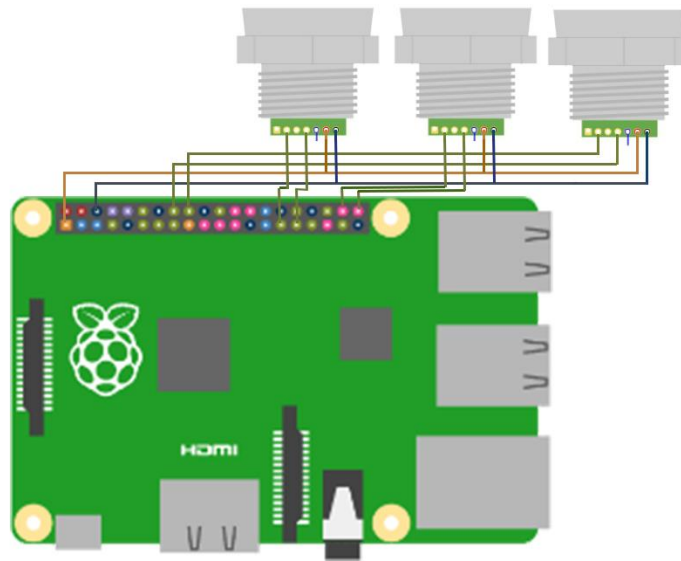
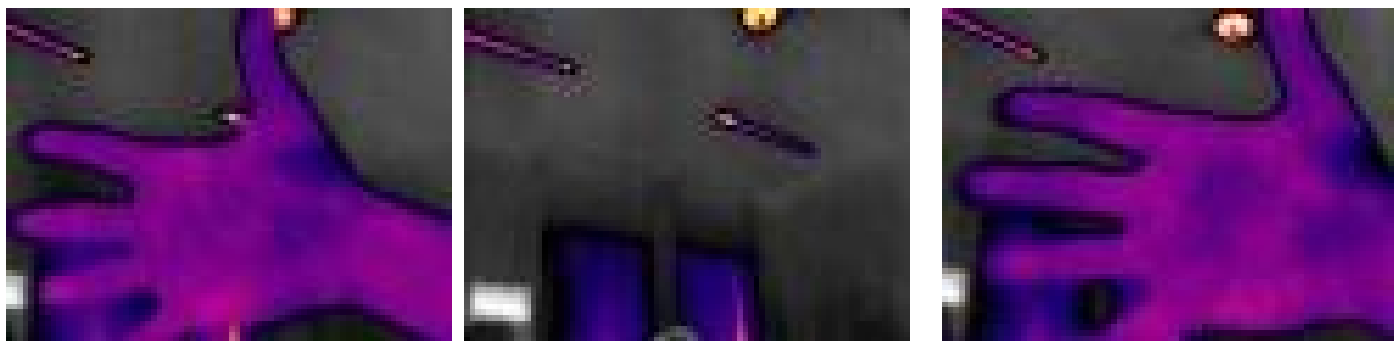


Fig 4.14: three MaxSonar sensors interface with raspberry pi

The python scripts which we used for running three ultrasonic sensors on single pi board is attached in **Annexure-F**.

4.3 Flir Thermal Imaging Camera

The thermal imaging camera has the sensitivity of $<50\text{mk}$, which means it can detect the slightest change in outside temperature. This thermal imaging camera is integrated on the raspberry pi, which is mounted on the IMR. This will detect images of various scenarios in front of the camera. The images will be processed to take out the norm, taking two images at a time and comparing them. The one with the highest norm will enable the robot to move towards the hotter region from its current position. High norm mean that the image contains higher temperature region. This thermal imaging camera will thus tell the robot where the highest temperature region is and robot will move towards that. A test image of thermal imaging camera is shown below:



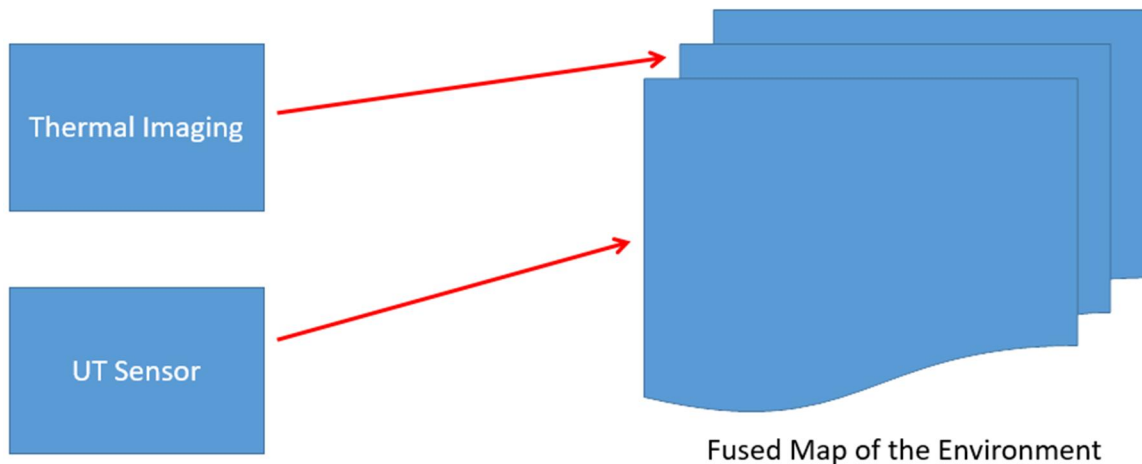
4.3.1 Progress with the sensor

Our progress with this sensor is that we have modified the available codes of thermal imaging camera to work with our conditions. Now the sensor can take images as per the user desire or pre-defined delay in the code. Currently, we have used the delay of two seconds in our code of thermal imaging camera. So that our camera takes imaging after every two seconds in its directory where the images are exported to take out the norm. We have successfully written the code in python for taking out the norm from the images captured by Flir thermal imaging camera. The code is attached in **Annexure-E**.

Based on these norm values and the distance values which we got from MaxSonar sensor, a 3D map of the environment was generated which tells the temperature of different regions in the environment around the robot.

4.4 Sensors Fusion for implementation of Control algorithms:

Feature Based Fusion Algorithm has been design to fuse UT and TIC data. It requires the extraction of salient features which are depending on their environment such as pixel intensities, and edges. These features from input images are fused to form a new image. The obtained information from fused image is then combined applying decision rules to reinforce common interpretation.

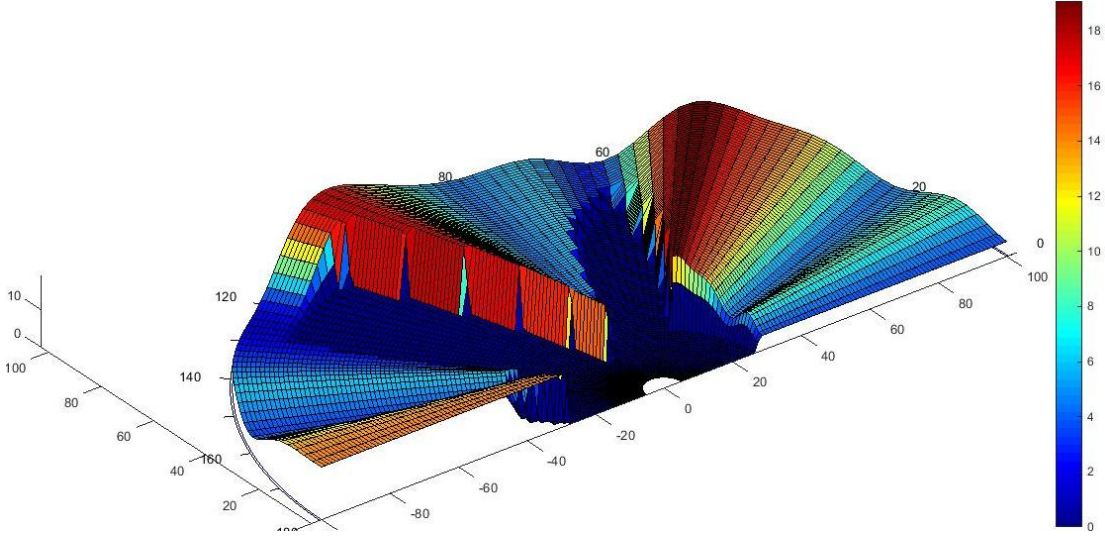


4.5 Development of 3D environment for Mobile Robots using UT and TIC:

For the development of 3D environment for the implementation of control algorithms for IMR, we fused the data from all our sensors in such a way that we took thermal gradient of thermal images in real time. So that it can be easier for robot for making decisions for it next movement.

The initial which we made by diffusing the data from Flir thermal imaging camera and MaxSonar sensor is shown in the figure below.

The temperature in different regions is represented by different color, where dark red color represent the highest temperature or most likely fire region while other colors represent the regions with relatively lesser values



Chapter 5. Problems and challenges with the design of MUSAFIR

5.1 Introduction

This chapter contains the modeling, control strategies and simulation results of the following four modules of a mobile robot.

5.2 Redesigning the MUSAFIR

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

5.1 INTRODUCTION:

The term, “Robot localization” refers to the methods and techniques that are employed to determine the exact position of the robot. In order to navigate autonomously and to perform useful tasks, a robot needs to know its position and orientation. Robot localization is therefore, a key problem in providing autonomous capabilities to a mobile robot.

5.2 ROBOT LOCALIZATION USING VARIOUS SENSORS:

Various on-board sensors can be used to determine the exact position of mobile robot, such as IMU – Inertial Measurement Units, which are composed of GYROSCOPES and ACCELEROMETERS. Similarly, an odometer can also be used to determine the robot’s position information. Since, the position is calculated by integrating values from these sensors, the robot localization errors grow with time. Therefore, these sensors cannot provide accurate position estimation of the robot in the long run due to accumulated errors by integration.

Another method for robot localization uses satellite based signals, e.g. GPS-Global Positioning System. The GPS provides accurate position information of robot but it has response slower than the sensors mentioned above. Also situations may arise when the GPS signals may not be available for some time. For e.g., when the robot is using GPS for localization and moves in enclosed spaces such as buildings etc.

The issues discussed above for various sensors provide motivation towards a technique that combines the features of various sensors eliminating the noise factor at the same time. Such a technique is termed as “Sensor Fusion”.

5.3 SENSOR FUSION:

Sensor Fusion is the combining of sensory data or data derived from sensory data such that the resulting information is in some sense better than would be possible when these sources were used individually.

A sensor fusion technique combines the data coming from various sensors so as to produce acceptable values of desired variables. A **Kalman Filter** is a well-known sensor fusion tool that is used for this purpose. Other methods use **complementary filters** and **artificial neural networks** as well. In the next section, different techniques for determination of exact geo-location are discussed.

5.4 SENSOR FUSION TECHNIQUES FOR ROBOT LOCALIZATION:

This section describes different sensor fusion techniques that combine data from various sensors to produce exact position information of a mobile robot. These techniques include the following:

(1) Sensor Fusion of INS and GPS using Kalman Filter

(2) Sensor Fusion of INS, GPS and Odometer using Kalman Filter and Complimentary Filter

(3) Sensor Fusion of INS, GPS and Odometer using Kalman Filter, Complimentary Filter and Artificial Neural Networks-ANN

5.4.1. SENSOR FUSION OF INS AND GPS USING KALMAN FILTER:

In this scheme of robot localization, only GPS (Global Positioning System) and INS (Inertial Navigation System) is used for determining the position of robot. The data from both the sensors is fused together using “Kalman Filter” as a sensor fusion technique to produce the desired position of robot.

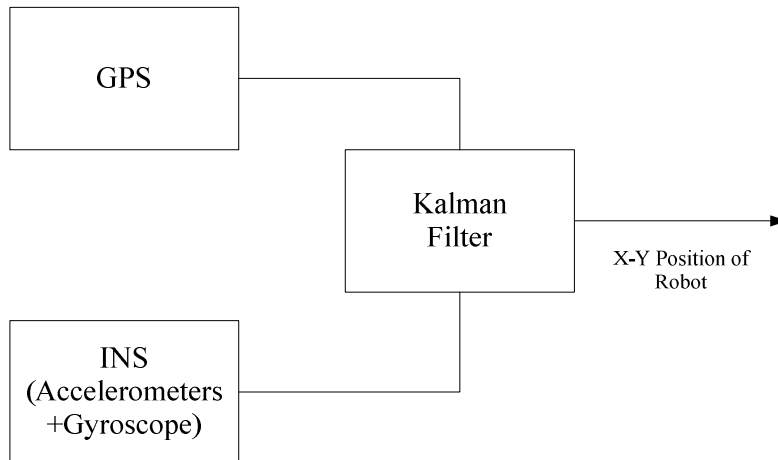


Fig5.1 Sensor Fusion of GPS/INS Only

Inertial navigation system is composed of two sensors: accelerometers and gyroscopes and depending upon the application, these sensors could be dual-axis or tri-axis. Double integration of accelerometer data produces position information. However, the position obtained from accelerometers does not have long-term accuracy, therefore, the position data from INS is combined with GPS using KF.

5.4.1.1-ADVANTAGES OF THE SCHEME:

Due to Kalman filter, the external disturbances and noise are considerably removed and an acceptable result is obtained.

5.4.1.2-DRAWBACKS OF THE SCHEME:

The two major draw-backs of the scheme that must be highlighted here are:

1. The scheme fails to work when there is no GPS signal as well as during GPS outages. When there is no GPS signal, the GPS/INS integration scheme becomes equivalent to INS only.i.e, at that time when there's no GPS signal, we are left with INS based position data only which contains accumulated errors due to integration.
2. The scheme cannot be used for indoor positioning applications as GPS works in outdoor environments only.

Therefore, due to the limitations of above scheme not suitable for indoor environments such as inside buildings etc. A system capable of working indoors and outdoors is developed. This scheme is discussed in the next section.

5.4.2. SENSOR FUSION OF INS, GPS AND ODOMETER USING KALMAN FILTER AND COMPLEMENTARY FILTER:

The motivation behind the development of such a scheme that is capable of working in indoor as well as outdoor environment is that the scheme presented in section 5.2.1 fails to estimate position in the absence of GPS.

In this scheme, data from three sensors namely, INS, GPS and Odometer are combined to address the problem of position estimation of a mobile robot.GPS provides position information in outdoors where odometry is employed here to give position information in indoor environment. The data coming from the three sensors is combined in two steps:

- 1. Sensor Fusion of Odometer and INS Using Kalman Filter**
- 2. Sensor Fusion of Kalman Filter Position Output and GPS Using Complementary Filter**

The block diagram summarizing this scheme is shown in Fig. 5.2

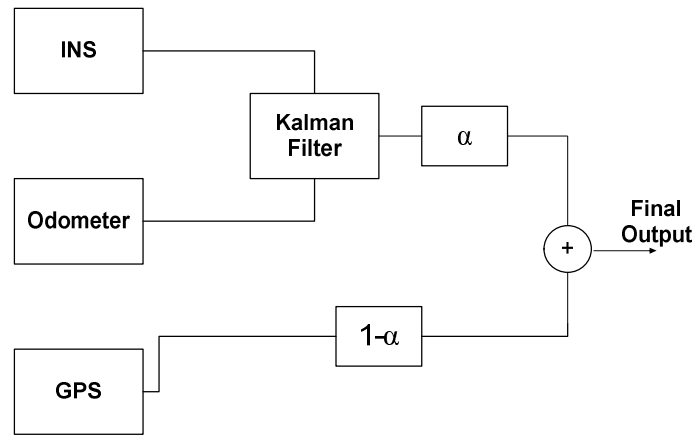


Fig5.2 Sensor Fusion of GPS/INS and odometer

In the above diagram, it is shown that the weighting parameter ‘ α ’ is used to fuse the data from Kalman Filter and odometer in order to obtain the exact position result, i.e:

$$\mathbf{Final\ Position} = (1-\alpha) \mathbf{GPS} + \alpha (\mathbf{Kalman\ Filter\ Position\ Output}) \quad \text{---- eqn.(1)}$$

The filter of the form: $z = a * x + (1 - a) * y$ is called a “*complimentary filter*”.

From above equation , **eqn.(1)**,for final output, it can be seen that the system still works when there are no GPS signals , that is , in indoor environment as well as when there are GPS signals present ,that is, in outdoor environments. The only parameter that needs to be adjusted is parameter, ‘ α ’.

5.4.2.1 ADJUSTING WEIGHTING/FUSION PARAMETER, α :

1. The value of weighting parameter ‘ α ’ is adjusted in accordance with the availability of GPS signals. If GPS signals are available, then α is assigned the following values:

$$0 \leq \alpha \leq 0.1$$

2. If the values of GPS signals are not available, then the weighting parameter ‘ α ’ is assigned the values in the range:

$$0.9 \leq \alpha \leq 1$$

Range of ‘ α ’:

The range of ‘ α ’ is between ‘0’ and ‘1’. i.e,

$$0 \leq \alpha \leq 1$$

5.4.2.2: ADVANTAGES OF THE SCHEME:

1. The scheme is simple to implement as the third sensor, odometer is fused using complementary filter.
2. The scheme provides a solution for position estimation in indoor environments.

5.4.2.3: DISADVANTAGES OF THE SCHEME:

The disadvantage of this scheme is that while odometer was easy to implement, it suffers from common dead reckoning error due to wheel slippage. This will cause error in calculation of position.

The scheme presented in the next section presents a modification of the above scheme, making use of artificial neural networks along with odometry data to provide localization solution in indoor environments, i.e, during the unavailability of GPS signals.

5.4.3. SENSOR FUSION OF INS, GPS AND ODOMETER USING KALMAN FILTER , COMPLIMENTARY FILTER AND ARTIFICIAL NEUTRAL NETWORKS-ANN:

We all learn that the cheapest, simplest and most useful solution for navigation and localization is the GPS system. But when you use a GPS sensor inside a house or building, a wide variety of barriers and interference make it difficult for GPS devices to work particularly well indoors. Given this, we have to forget the GPS navigation system for indoor use and try other methods.

A solution for robot localization for indoor environment was discussed in section 5.4.2. However, the wheel slippage problem in odometer arises as an important issue in the accuracy of position estimation of robot.

In this section, a scheme employing the artificial neural networks in addition to Kalman and complementary filters is presented to overcome the problems mentioned in the above paragraphs. We show that this strategy proves useful when the robot is using GPS to localize itself as well as when GPS becomes unavailable for some time.

5.4.3.1. DETAILS OF THE SCHEME:

The scheme is represented in block diagrams in Fig3 and Fig4. The details of the scheme are as follows:

(a) WHEN THE GPS IS AVAILABLE:

As shown in Fig3, when GPS data is available, a multi-layered perceptron neural network is trained. First the position data from INS and GPS is fused together using Kalman Filter (KF) sensor fusion algorithm. This removes any uncertainty or noise in the GPS data. The KF works only when the GPS signal is present. For e.g., if the GPS sampling frequency is 2Hz, the KF provides position estimation after every 0.5 seconds. Meanwhile the neural network is also trained with the dataset presented to it after every 0.5 seconds. Multi-layered perceptron neural networks, (MLP-NN) are employed here for training the data-set. Multi Layer perceptron (MLP) is a feed-forward neural network with one or more layers between input and output layer. Feed-forward means that data flows in one direction from input to output layer (forward). The block diagram showing feed-forward MLP-NN is shown in Fig5. The implementation details of Kalman Filter and MLP-NN are discussed in section 5.4.3.2.

FINAL OUTPUT:

The final output during the MLP-NN training phase or in other words, when the GPS signal is available, is obtained by fusing the KF output with odometer based position data using complimentary filter as:

$$Final\ Output = \alpha (KF\ Position) + (1-\alpha) (Odometer\ Position) \text{ ---- eqn.(2)}$$

Where ‘ α ’ is the fusion parameter. The range of ‘ α ’ is between ‘0’ and ‘1’.

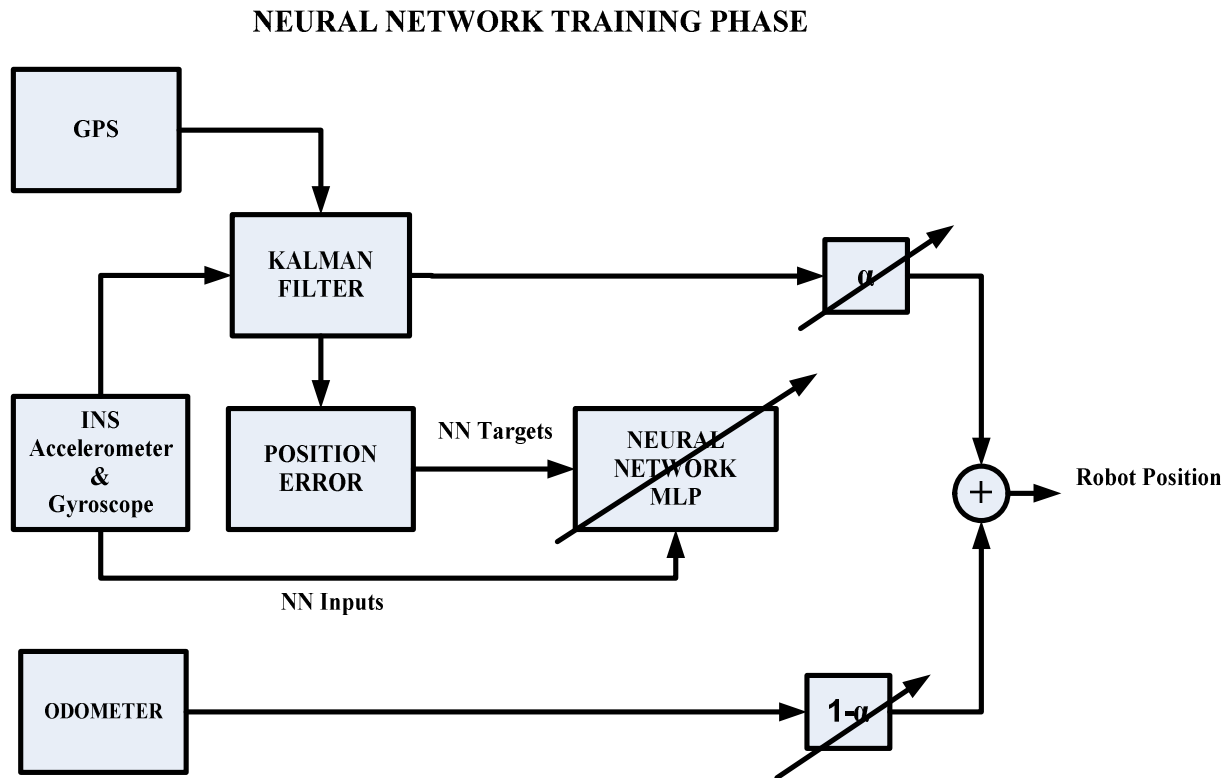


Fig3. MLP-NN Training when GPS is available

(b) WHEN THE GPS IS UNAVAILABLE:

The block diagram of the system when GPS signals are not available is shown in Fig4. When the GPS signals are not available, the trained MLP-NN is tested, the data from INS is presented as input vector to the trained MLP which predicts the change in position ' Δp ', and the current position of robot ' p_c ' is found using the change in position ' Δp ' predicted by MLP-NN and last saved GPS position ' p_o ' as:

$$p_c = p_o + \Delta p$$

FINAL OUTPUT:

The final output of the system when the GPS signal is unavailable is obtained by the fusion of current position estimate calculated using MLP-NN output and odometer, using complimentary filter as:

$$\mathbf{Final\ Output} = \alpha (\mathbf{MLP-NN\ Position}) + (1-\alpha) (\mathbf{Odometer\ Position}) \text{ ---- eqn.(3)}$$

Where ' α ' is the fusion parameter. The range of ' α ' is between '0' and '1'.

NEURAL NETWORK TESTING PHASE

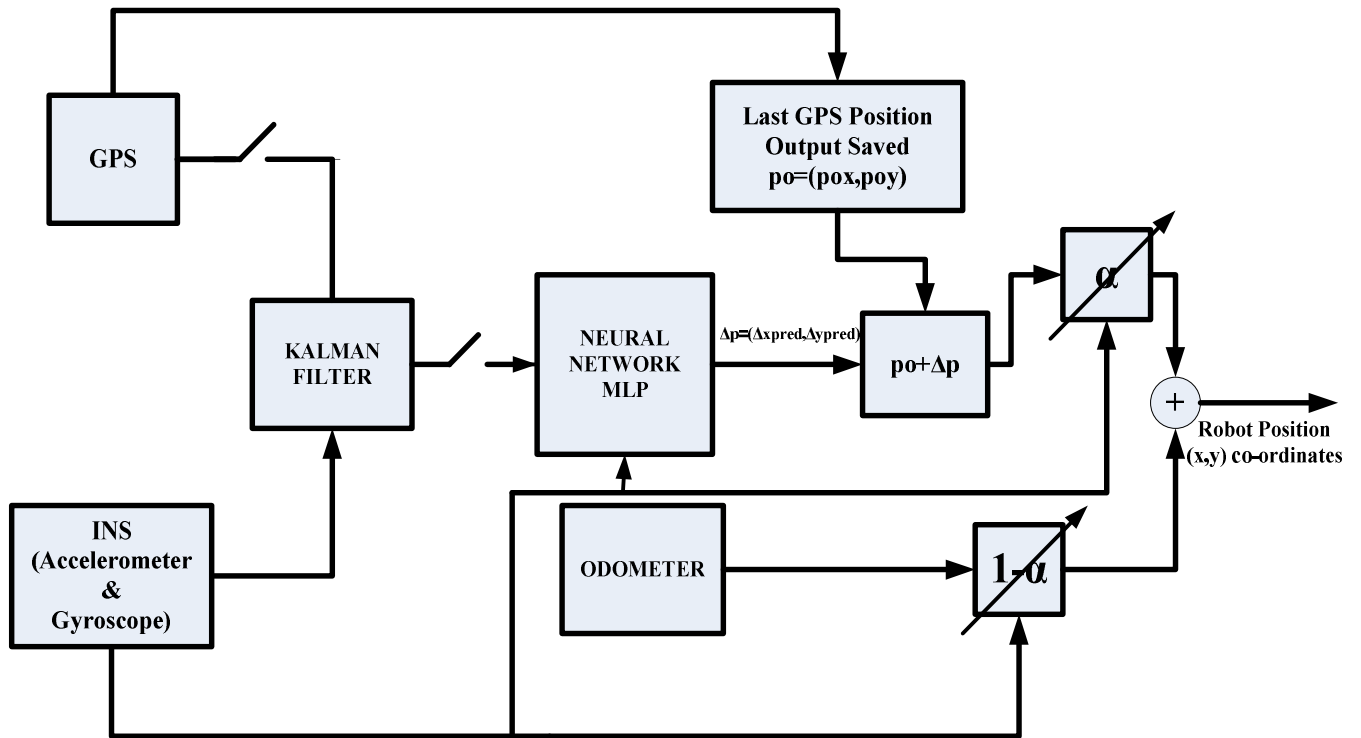


Fig4. MLP-NN Testing when GPS is unavailable

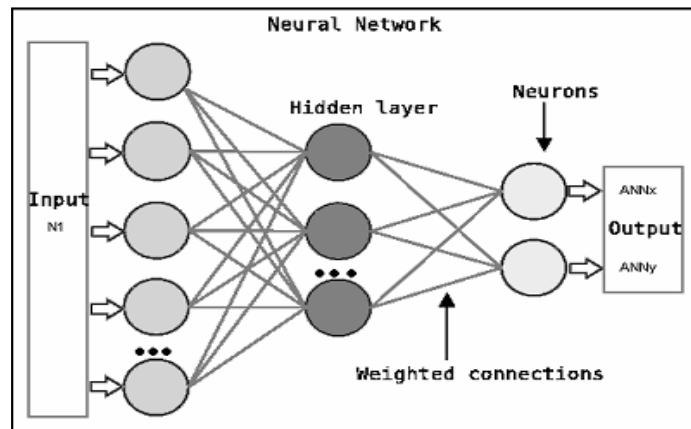


Fig5. MLP-NN Architecture

5.4.3.2. IMPLEMENTATION:

In this section, the implementation of Kalman filter algorithm followed by the details of Multilayer perceptron architecture and MLP-NN training data set are presented.

(a) KALMAN FILTER:

As described in section 5.4.3.1, the INS and GPS data is combined together using sensor fusion scheme called ‘Kalman Filter’. The main steps of Kalman filter are described as follows:

STEPS OF KALMAN FILTER ALGORITHM:

Kalman Filter is a recursive algorithm. The main steps of KF algorithm are highlighted here:

1. Initialize the values of states and error covariance matrices

$$\hat{x}_0, P_0 \text{ ---- eqn.(4)}$$

2. Predict the state and error covariance.

$$\hat{x}_k = A\hat{x}_{k-1} + Bu \text{ ---- eqn.(5)}$$

$$P_k^- = AP_{k-1}A^T + Q \text{ ---- eqn.(6)}$$

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

$$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1} \text{ ---- eqn.(7)}$$

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

4. Calculate the state estimate:

$$\hat{x}_k = \hat{x}_k^- + K(z - H\hat{x}_k^-) \text{ ---- eqn.(8)}$$

5. Calculate the error covariance:

$$P_k = P_k^- - KHP_k^- \text{ ---- eqn.(9)}$$

The algorithm then iterates through steps 2 to 5 predicting the states \hat{x}_k in each iteration.

KALMAN FILTER FORMULATION FOR THE FUSION GPS AND INS:

As shown in Fig3, the position data from GPS and INS is combined using Kalman Filter. In order to combine INS-GPS data, an 8 state Kalman filter is implemented:

STATE VECTOR:

The state vector is:

$$x = [a_N \ a_E \ a_D \ V_N \ V_E \ V_D \ \lambda \ \mu] \text{ ---- eqn.(10)}$$

Where,

a_N is the robot acceleration along north

a_E is the robot acceleration along east

a_D is the robot acceleration in down-ward direction

V_N is the robot velocity along north

V_E is the robot velocity along east

V_D is the robot velocity in down-ward direction

λ is the geodetic latitude

μ is the geodetic longitude

STATE TRANSITION MATRIX:

The state transition matrix has been assigned the following values:

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ \Delta T & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & \Delta T & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & \Delta T & 0 & 0 & 1 & 0 & 0 \\ \frac{\Delta T}{R_M} & 0 & 0 & \frac{1}{R_M} & 0 & 0 & 1 & 0 \\ 0 & \frac{\Delta T}{R_N \cos(\lambda(k))} & 0 & 0 & \frac{1}{R_N \cos(\lambda(k))} & 0 & 0 & 1 \end{bmatrix} \text{ ---- eqn.(11)}$$

Where, ΔT is the sampling time, R_M and R_N are radius of curvature of earth in meridian and prime vertical respectively, and are given by:

$$R_M = \frac{a(1-e^2)}{(1-e^2 \sin^2 \phi)^{3/2}} \text{ ---- eqn.(12)}$$

$$R_N = \frac{a}{(1-e^2 \sin^2 \phi)^{1/2}} \text{ ---- eqn.(13)}$$

Where, ‘ a ’ is the equatorial radius of earth, given by:

$$a = 6,378,137,0 \text{ m}$$

and ‘ e ’ is the eccentricity having the value:

$$e = 0.08181919$$

INPUT MATRIX:

Since, there is no input, the input matrix 'B' is zero.

$$B = [0_{8 \times 1}] \text{ ---- eqn.(14)}$$

MEASUREMENT MATRIX:

The measurement matrix 'z' is composed of sensor measurements from accelerometers and GPS position co-ordinates.

$$z = [a_N \ a_E \ a_D \ \lambda \ \mu] \text{ ---- eqn.(15)}$$

OBSERVATION MATRIX:

The observation matrix 'H' is:

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \text{ ---- eqn.(16)}$$

PROCESS NOISE COVARIANCE MATRIX:

The process noise covariance matrix is assigned the following values.

$$Q = [0.1_{8 \times 8}] \text{ ---- eqn.(17)}$$

MEASUREMENT NOISE COVARIANCE MATRIX:

The measurement noise covariance matrix *R* is:

$$R = \begin{bmatrix} \sigma_{aN}^2 & 0 & 0 & 0 & 0 \\ 0 & \sigma_{aE}^2 & 0 & 0 & 0 \\ 0 & 0 & \sigma_{aD}^2 & 0 & 0 \\ 0 & 0 & 0 & \sigma_{\lambda}^2 & 0 \\ 0 & 0 & 0 & 0 & \sigma_{\mu}^2 \end{bmatrix} \text{ ---- eqn.(18)}$$

INITIAL VALUES OF ERROR NOISE COVARIANCE AND STATE VECTOR:

The error covariance matrix P is initialized to:

$$P_0 = Q \text{ ---- eqn.(19)}$$

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

$$X_0 = [0_{1 \times 8}] \text{ ---- eqn.(20)}$$

IMPLEMENTATION OF MULTI-LAYER PERCEPTRON NEURAL NETWORK:

As shown in Fig3, during the availability of GPS signals the Multilayer Perceptron neural network is trained. When the GPS signal loses, the trained MLP network works instead of GPS to predict the current position of the robot. This is depicted in Fig4. The architecture of Multi-layer Perceptron network and details of training data-set are as follows:

MLP-NN ARCHITECTURE:

Two Multi-layer Perceptron neural networks are implemented here for predicting the change in position of robot (change in latitude and longitude). Both the MLP networks are implemented in the following architecture:

$$\text{MLP1} = 12 \times 50 \times 40 \times 1$$

$$\text{MLP 2} = 12 \times 50 \times 40 \times 1$$

In both the MLP networks, there is one input layer, two hidden or processing layers having 50 and 40 neurons respectively and one output neuron.

TRAINING DATASET:

The training data sets for MLP1 and MLP2 are as follows:

TRAINING DATASET FOR MLP1:

. Input and Targets for MLP 1:

$$\text{Inputs: } [a_N \ a_E \ a_D \ \sum a_N \ \sum a_E \ \sum a_D \ v_N \ v_E \ v_D \ \phi \ \theta \ \psi]$$

$$\text{Target(s): } [\Delta\lambda] \text{ (change in latitude)}$$

TRAINING DATASET FOR MLP2:

b. Input and Targets for MLP 2:

$$\text{Inputs: } [a_N \ a_E \ a_D \ \sum a_N \ \sum a_E \ \sum a_D \ v_N \ v_E \ v_D \ \phi \ \theta \ \psi]$$

Target(s): $[\Delta\mu]$ (change in longitude)

Where,

$a_N a_E a_D$ are accelerations in NED inertial frame

$\sum a_N \sum a_E \sum a_D$ are cumulative accelerations in NED inertial frame

$v_N v_E v_D$ Velocities in NED inertial frame

$\phi \theta \psi$ are Euler Angles

All the variables in the input vector are obtained from INS – Inertial Navigation System only ,as shown in fig3 and fig4.The inertial navigation system comprises of accelerometers and gyroscopes. The accelerometers provide accelerations in three dimensions from which we calculate cumulative values of accelerations. The velocity is calculated by integrating the acceleration values. Since the acceleration values are in sensor frame initially, therefore, these values were transformed from sensor body frame to NED reference inertial frame. The gyroscope gives angular velocities, using these angular velocities, euler angles can be calculated using the quaternion using the following steps.

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:

$$e_{0i} = \cos\left(\frac{\psi_i}{2}\right)\cos\left(\frac{\theta_i}{2}\right)\cos\left(\frac{\varphi_i}{2}\right) + \sin\left(\frac{\psi_i}{2}\right)\sin\left(\frac{\theta_i}{2}\right)\sin\left(\frac{\varphi_i}{2}\right) \text{ ---- eqn.(21)}$$

$$e_{1i} = \cos\left(\frac{\psi_i}{2}\right)\cos\left(\frac{\theta_i}{2}\right)\sin\left(\frac{\varphi_i}{2}\right) - \sin\left(\frac{\psi_i}{2}\right)\sin\left(\frac{\theta_i}{2}\right)\cos\left(\frac{\varphi_i}{2}\right) \text{ ---- eqn.(22)}$$

$$e_{2i} = \cos\left(\frac{\psi_i}{2}\right)\sin\left(\frac{\theta_i}{2}\right)\cos\left(\frac{\varphi_i}{2}\right) + \sin\left(\frac{\psi_i}{2}\right)\cos\left(\frac{\theta_i}{2}\right)\sin\left(\frac{\varphi_i}{2}\right) \text{ ---- eqn.(23)}$$

$$e_{3i} = \cos\left(\frac{\psi_i}{2}\right)\sin\left(\frac{\theta_i}{2}\right)\sin\left(\frac{\varphi_i}{2}\right) + \sin\left(\frac{\psi_i}{2}\right)\cos\left(\frac{\theta_i}{2}\right)\cos\left(\frac{\varphi_i}{2}\right) \text{ ---- eqn.(24)}$$

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

$$\dot{e}_0 = -\frac{1}{2}(e_1 p + e_2 q + e_3 r) \text{ ---- eqn.(25)}$$

$$\dot{e}_1 = \frac{1}{2}(e_0 p + e_2 r - e_3 q) \text{ ---- eqn.(26)}$$

$$\dot{e}_2 = \frac{1}{2}(e_0 q + e_3 p - e_1 r) \text{ ---- eqn.(27)}$$

$$\dot{e}_3 = \frac{1}{2}(e_o r + e_1 q - e_2 p) \text{ ---- eqn.(28)}$$

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

$$\theta = \sin^{-1}(-2[e_1 e_3 + e_o e_2]) \text{ ---- eqn.(29)}$$

$$\phi = \cos^{-1} \frac{[e_o^2 - e_1^2 - e_2^2 + e_3^2]}{\sqrt{(1-4(e_1 e_3 - e_o e_2)^2)}} \text{ sign}(2[e_2 e_3 + e_o e_1]) \text{ ---- eqn.(30)}$$

$$\psi = \cos^{-1} \frac{[e_o^2 - e_1^2 - e_2^2 - e_3^2]}{\sqrt{(1-4(e_1 e_3 - e_o e_2)^2)}} \text{ sign}(2[e_1 e_2 + e_o e_3]) \text{ ---- eqn.(31)}$$

5.4.3.4. SIMULATION RESULTS:

The simulation results for the above scheme are shown in Fig6to Fig9.

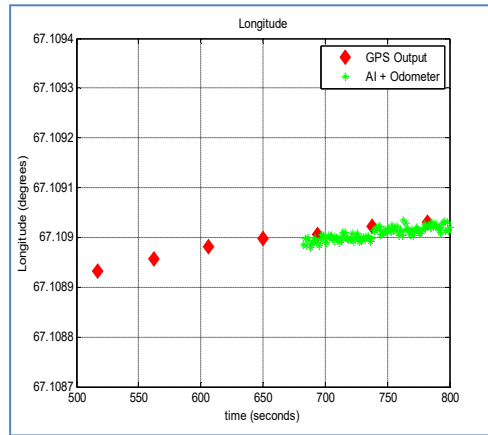


Fig6. GPS Output and AI (MLP) + Odometer Output (For longitude)

Explanation:

When GPS is on, firstly, the MLP is trained using the Kalman Filter outputs as targets, the KF works only when the GPS is on .After training with KF position data, the MLP is tested to predict position (longitude) in the absence of GPS data points, as shown in the above figure. The graph plotted in green shows the MLP predicted position output fused with odometer based position output using the complementary filter. The GPS points in red are also shown on the figure. It can be observed that after training, the MLP is able to predict position in the time instants where the GPS based position data is not available.

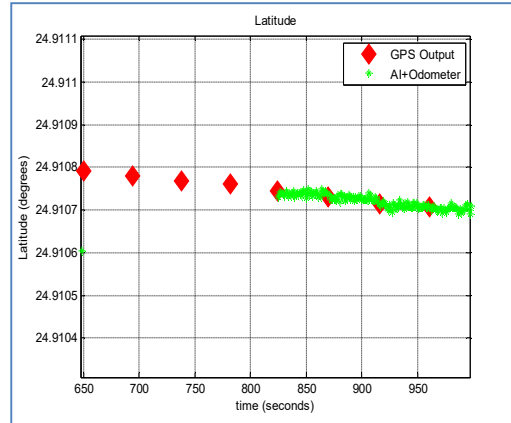


Fig7. GPS Output and AI (MLP) + Odometer Output (For latitude)

Explanation:

When GPS is on, firstly, the MLP is trained using the Kalman Filter outputs as targets, the KF works only when the GPS is on .After training with some KF position data, the MLP is tested to predict position (latitude) in the absence of GPS data points, as shown in the above figure. The graph plotted in green shows the MLP predicted position output fused with odometer based position output using the complementary filter. The GPS points in red are also shown on the figure. It can be observed that after training, the MLP is able to predict position in the time instants where the GPS based position data is not present.

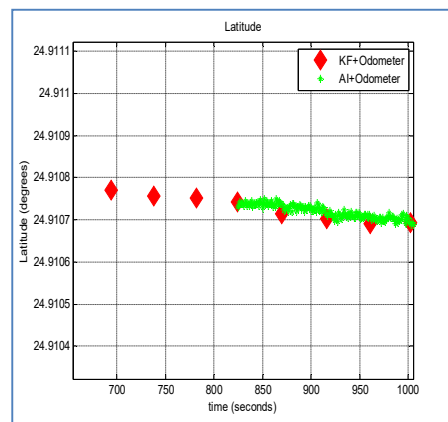


Fig8. KF + Odometer and AI (MLP) + Odometer Output (For latitude)

Explanation:

When GPS is on, firstly, the MLP is trained using the Kalman Filter outputs as targets, the KF works only when the GPS is on .After training with KF position data, the MLP is tested to predict position (latitude) in the absence of GPS data points, as shown in the above figure. The graph plotted in green shows the MLP predicted position output fused with odometer based position output using the complementary filter. The KF position fused with odometer based position are also shown on the figure in red. It can be observed that after training, the MLP is able to predict position in the time instants where the KF based position data is not available.

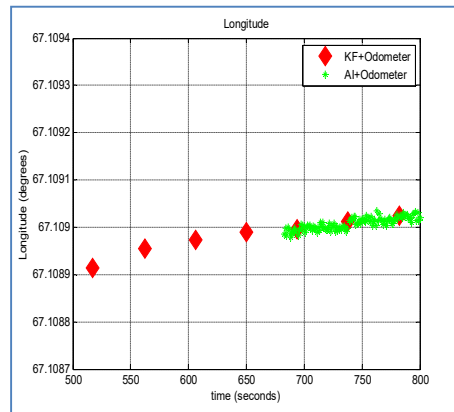


Fig9. KF + Odometer and AI (MLP) + Odometer Output (For longitude)

Explanation:

When GPS is on, firstly, the MLP is trained using the Kalman Filter outputs as targets, the KF works only when the GPS is on .After training with some KF position data, the MLP is tested to predict position (longitude) in the absence of GPS data points, as shown in the above figure. The graph plotted in green shows the MLP predicted position output fused with odometer based position output using the complementary filter. The KF position fused with odometer based position are also shown on the figure in red. It can be observed that after training, the MLP is able to predict position in the time instants where the KF based position data is not available.

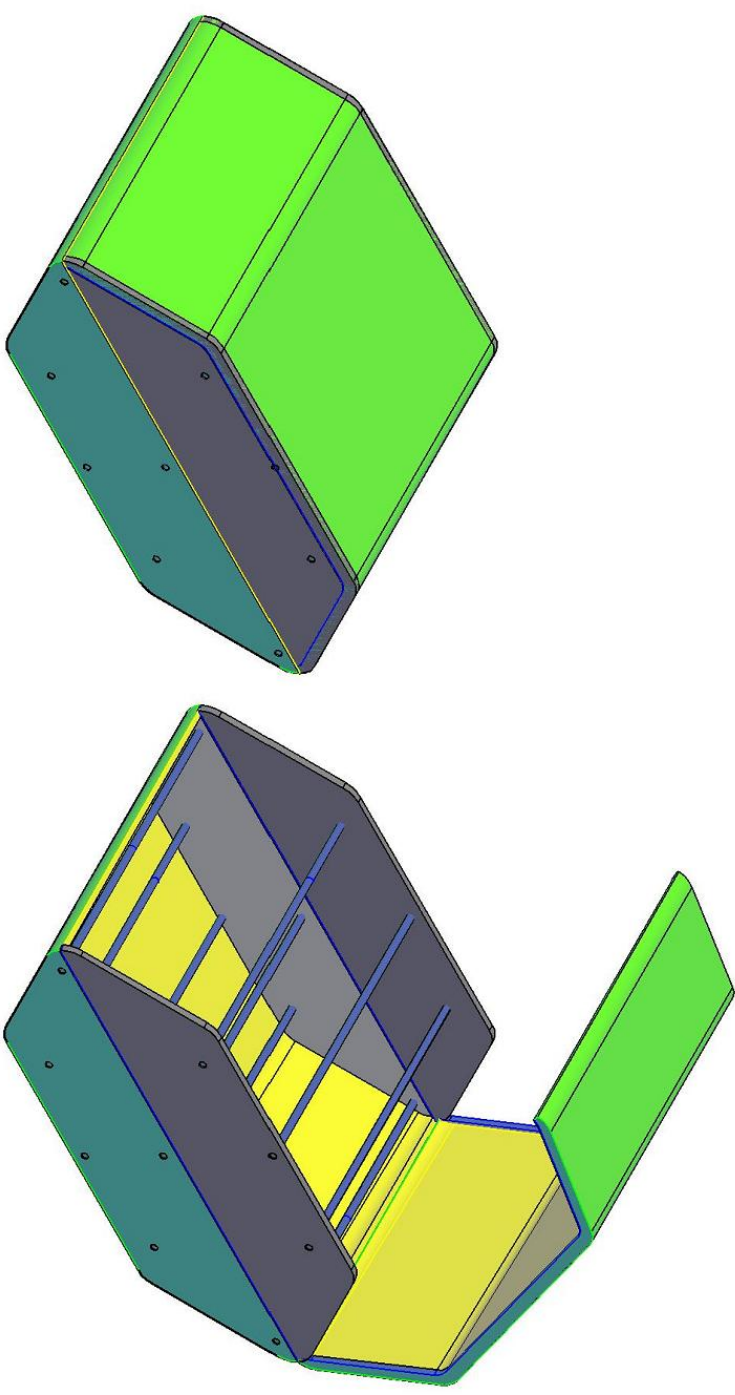
5.4.3.5. CONCLUSION:

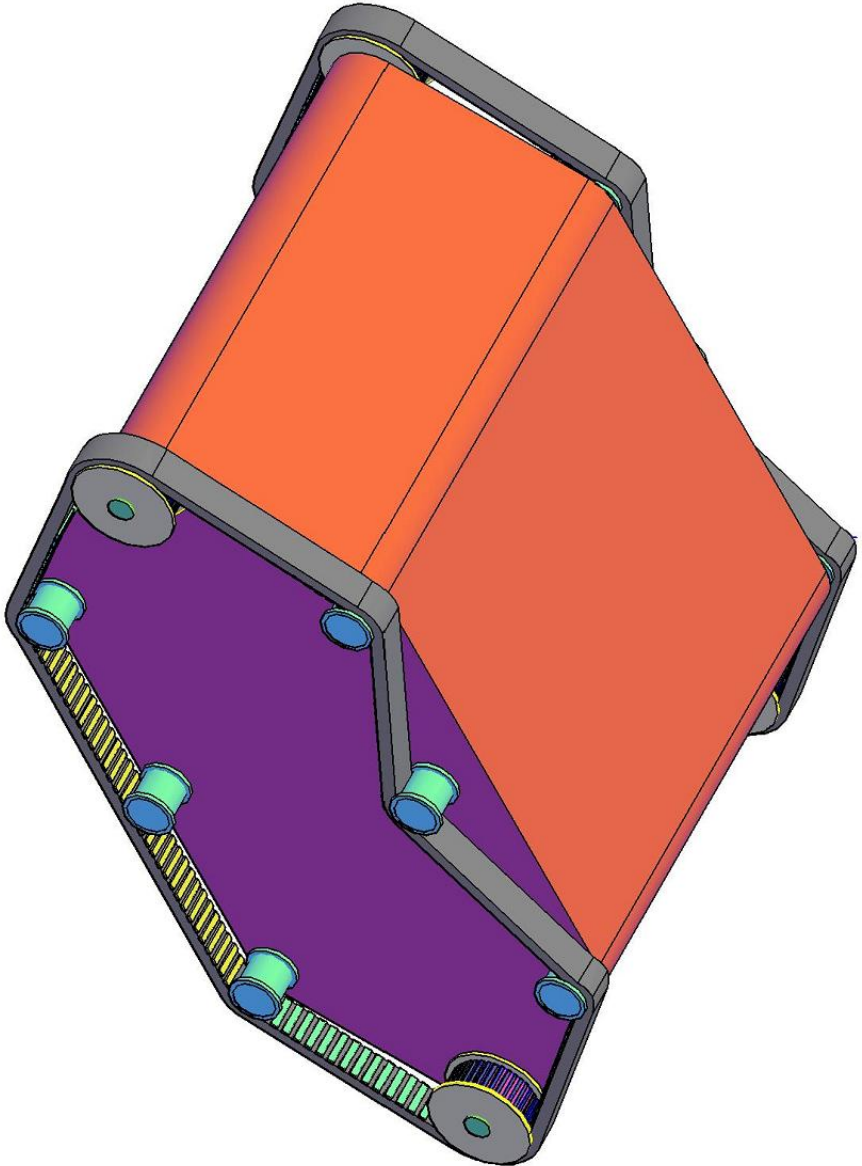
In the above scheme, data fusion of three sensors is presented namely GPS, INS and odometer. The proposed scheme provides the position data both during the availability and unavailability of the GPS signals. Neural network based on multi-layer perceptron is employed to predict the GPS information during GPS signal loss. In order to make the scheme more reliable future work is directed towards developing a scheme for online updating the fusion weighting parameter ' α '. With the proposed scheme absolute robot position to a greater degree of accuracy can be obtained in indoor as well outdoor environments.

Chapter 7. Problems in design and development of the mechanical structure of IMR

Two different types of design for the IMR have been explored. In the initial design treads were utilized in the mechanical structure. There were maneuverability issues with the initial design. In the second approach wheel based robots have been designed. Currently we are working on the improvement of both the designs. Mechanical drawings of both the models are included in this chapter.

7.1 Initial design of the IMR

<p>PAF (KIET) COLLEGE OF ENGINEERING ENGINEERING WORKSHOP</p>	<p>Fire fighting Robot</p>	<p>CAD Design Muhammad Waqas Assistant Manager Mechatronics</p> <p>DATE 26 /04/ 2017</p>
		



PAF (KIET)
COLLEGE OF ENGINEERING
ENGINEERING WORKSHOP

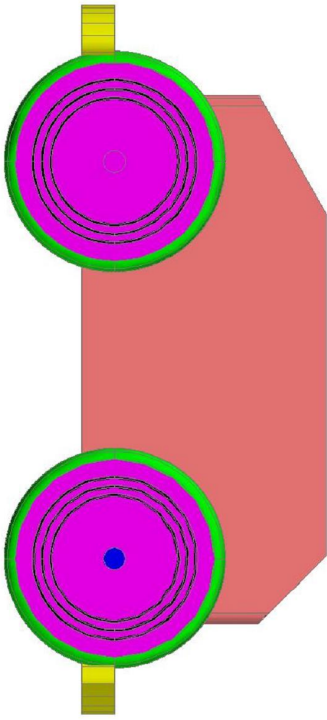
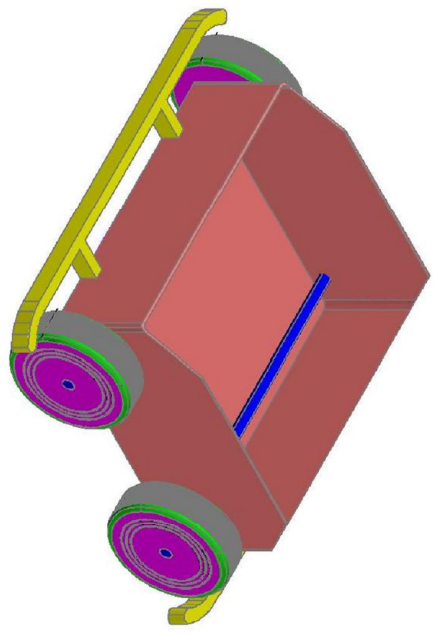
Fire fighting Robot

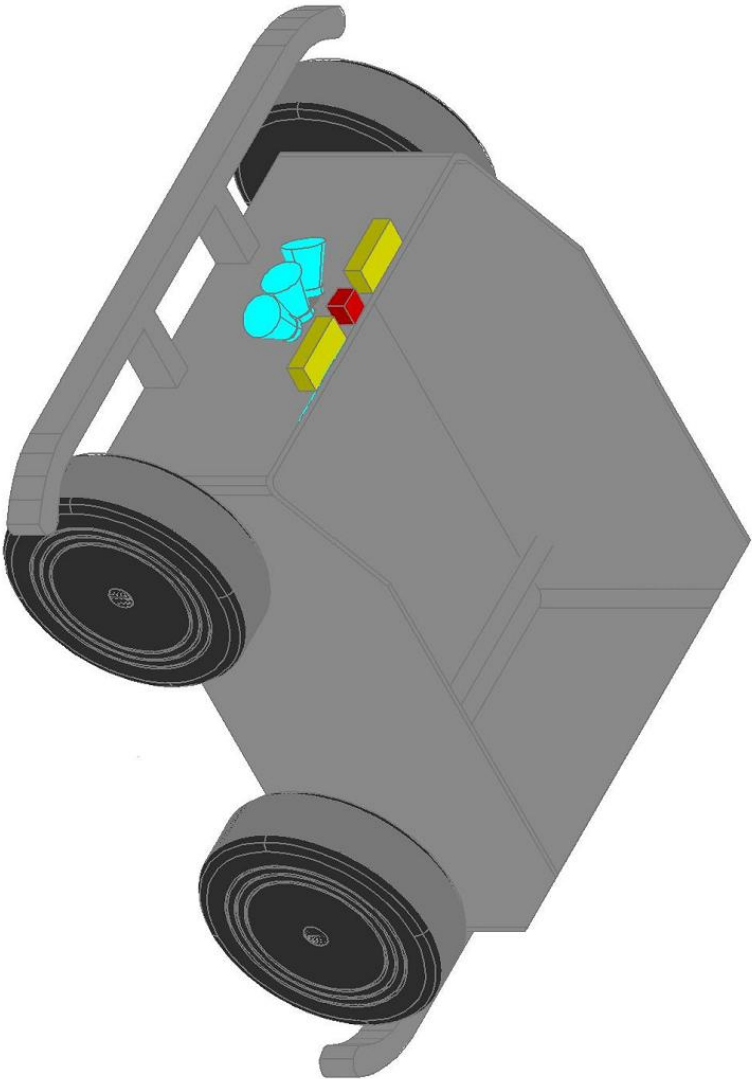
CAD Design

Muhammad Waqas
Assistant Manager Mechatronics

DATE 26 /04 /2017

7.2 Proposed modifications in the design of the IMR

<p>PAF (KIET) COLLEGE OF ENGINEERING ENGINEERING WORKSHOP</p>	<p>IMR BOT</p>	<p>CAD Design Muhammad Waqas Assistant Manager Mechatronics</p> <p>DATE 27 /04/ 2017</p>		
---	----------------	--	--	---



PAF (KIET)
COLLEGE OF ENGINEERING
ENGINEERING WORKSHOP

IMR BOT

CAD Design

Muhammad Waqas
Assistant Manager Mechatronics

DATE 02/05/2017

