

# ACKNOWLEDGEMENT

I would like to avail the opportunity to acknowledge my ongoing project as a part of Electronics and Communication Engineering course under the department of Electronics and Communication Engineering, Tezpur University.

My deepest sense of gratitude to **Dr. Nabarun Bhattacharya**, Senior Director and Centre Head, C-DAC, Kolkata, for his never-ending guidance and direction through valuable suggestions.

I express my heartfelt thanks to **Mr. Alokesh Ghosh**, Associate Director, C-DAC, Kolkata. **Dr. Hena Ray**, Joint Director. **Mr. Ravi Sankar**, Project Engineer. **Mr. Angshuman Chakraborty**, Project Assistant, C-DAC, Kolkata in the completion of project for guiding and correcting various documents, readings, result with utmost attention and care. They have taken the effort to go through the project and make necessary corrections when needed.

I owe my appreciation to the helpful people at C-DAC, Kolkata for their support.

I also express my gratitude to **Prof. Partha Pratim Sahu**, HoD, Dept. of ECE, Tezpur University for extending his support and giving me the opportunity to undergo my internship at C-DAC, Kolkata.

I would also like to thank my Institution and faculty members, without them, this project would have been a distant reality. I also extend my heartfelt thanks to my family and well-wishers.

# ABSTRACT

With the advancement of science and technology, the importance of robotic applications and IoT (Internet of Things) is increasing day by day. Considering that we have decided to work on the design and development of a robot with the help of microcontrollers and Wi-Fi based IoT application. Internet of Things (IoT) provides a strong platform to connect objects to the Internet for facilitating Machine to Machine (M2M) communication and transferring data using standard network protocols like TCP/IP.

The agriculture industry is being revolutionized by robotics and automation. With the help of robotics and IoT, farmers address the issue of a dwindling workforce and allow them to work more efficiently while saving money on labor. Spraying pesticides and weed killers onto fields is not only wasteful, it can severely harm the environment. Robots provide a much more efficient method in such cases.

# CONTENTS

Serial No.	TOPIC	Page No.
CHAPTER 1: WORKING IN THE LINUX ENVIRONMENT		4-13
1.1	INTRODUCTION	5
1.2	USER, NAME AND PASSWORD	5
1.3	SOME TERMINOLOGIES	5
1.4	SOME COMMAND LINES USED IN TERMINAL	6
1.5	VI EDITOR	8
1.6	GEDIT	8
1.7	MCEDIT	8
1.8	GCC	9
1.9	G++	10
1.10	FILE TRANSFER PROTOCOL	12
1.11	FILEZILLA	12
CHAPTER 2: TURTLEBOT3 WITH ROS		14-30
2.1	INTRODUCTION	15
2.2	HARDWARE DESCRIPTION OF TURTLEBOT3 BURGER	15
2.3	WORKING WITH ROS	16
2.4	MAPPING, LOCALIZATION & NAVIGATION	24
2.5	SLAM	25
2.6	PUBLISHER & SUBSCRIBER NODE USING DWM	26
2.7	DECAWAVE MODULE (DWM1001)	28
CHAPTER 3: DESIGN OF OUR OWN NAVIGATION ROBOT		31-61
3.1	INTRODUCTION	32
3.2	HARDWARE DESCRIPTION	32
3.3	ASSEMBLY OF THE ROBOT	43
3.4	WORKING PRINCIPLE OF THE MOTOR DRIVER BOARD	44
3.5	WORKING PRINCIPLE OF THE NVIDIA JETSON TX2 BOARD	45
3.6	SOFTWARE DESCRIPTION	46
3.7	DECAWAVE MODULE (DWM1001) INTERFACE	46
3.8	I2C PROTOCOL	48
3.9	IMU INTERFACE	49
3.10	GY-271 HMC5883L DIGITAL ELECTRONIC COMPASS 3 AXIS MAGNETIC FIELD SENSOR	54
3.11	FILE HANDLING IN C	58
3.12	SOCKET PROGRAMMING USING TCP/IP FOR COMMUNICATION BETWEEN REMOTE PC AND BOT	59
CHAPTER 4: IoT BASED TEMPERATURE AND HUMIDITY SENSING MODULE		61-80
4.1	INTRODUCTION	62
4.2	ARDUINO UNO	62
4.3	DHT22	63
4.4	ESP8266	63
4.5	BIDIRECTIONAL LEVEL SHIFTER	64
4.6	UART/SERIAL COMMUNICATION	65
4.7	PARALLEL COMMUNICATION	65
4.8	DIFFERENCE BETWEEN SERIAL AND PARALLEL COMMUNICATION	65
4.9	CONNECTING DHT22 SENSOR TO ESP8266 NODEMCU	69
4.10	INSTALLING ESP8266 BOARD IN ARDUINO IDE	70
4.11	INSTALLING DHT SENSOR LIBRARY	70
4.12	CREATING ESP8266 NODEMCU WEB SERVER USING WI-FI AP MODE	71
4.13	ACCESSING THE WEB SERVER	74
4.14	DETAILED CODE EXPLANATION	74

# **Chapter 1:**

## **WORKING IN THE LINUX ENVIRONMENT**

## 1.1 Introduction

Linux is a free version of UNIX (or UN\*X). The free part is not meant in money terms but rather that the source code for Linux is freely available for inspection, modification and what we feel we can/should do.

Linux is a **multitasking** and **multiuser** operating system. Now, a little explanation of this terminology.

An **operating system** is a collection of programs that runs in a computer so that a person can easily access the hardware and all resources of the computers. The operating system is the big program that makes our computer life easy (or difficult, if the operating system is a bad one).

A **multitasking** operating system is capable of doing several tasks at the same time (well, not quite so, but it seems like that from the human point of view).

A **multiuser** operating system has a concept of “userquot;”, a way to identify the person that is using the system, and can allow different users to perform different tasks in the computer, and protect one user's tasks from interfering with another user's programs.

## 1.2 Username and Password

So, when we want to work in a computer running Linux, the first thing we need is a user name (or logging name). That will be given to us by the person that administrates the system.

We will also need a password to be sure that only we use our account. An initial password might be given to us by the system administrator; we can change it later to something that we can easily remember. But we will try some password that is difficult to guess, or any one will be able to use our account (and remove our files, send email on our name, etc.).

## 1.3 Some terminologies

There are a few other terms that will help us understand the rest of the manual:

### **Shell:**

This is a program in the system that allows us to give the commands that we want to execute. It is the basic program that connects us to the operating system.

### **Process:**

Any task that we run in the system is called a process.

### **File:**

A part of the hard disk that contains data owned by a user of the system.

## **X-windows (or simply windows):**

This is a mode of Linux where the screen (monitor) can be split in small "parts" called windows, that allows us to do several things at the same time (or rather change from one task to another easily) and view graphics in a nice way.

## **Terminal:**

The terminal is an interface in which we can type and execute text-based commands. It can be much faster to complete some tasks using a Terminal than with graphical applications and menus.

Another benefit is allowing access to many more commands and scripts.

A common terminal task of installing an application can be achieved within a single command, compared to navigating through the Software Centre or Synaptic Manager.

## **Session:**

The time we spend between logging on in the system and logging out of the system.

# **1.4 Some command lines used in terminal**

## **CP**

cp file1 file2

-i : takes permission, -r : copies all subsidiaries, -v : for verbose

## **RM**

To remove (or delete) a file or directory in Linux from the command line, use the rm (remove) command.

- To delete a single file use, the rm command followed by the file name:  
\$ rm filename
- If the file is write protected we will be prompted for confirmation as shown below. To remove the file type y and hit Enter. Otherwise, if the file is not write protected it will be deleted without prompting.  
\$ rm: remove write-protected regular empty file 'filename'?
- To delete multiple files at once use the rm command followed by the file names separated by space.  
\$ rm filename1 filename2 filename3
- We can also use a wildcard (\*) and regular expansions to match multiple files. For example to remove all .pdf files in the current directory, use the following command:  
\$ rm \*.pdf

- We can use the `-i` option to confirm each file before deleting it:

```
$ rm -i filename(s)
```

- To remove files without prompting even if the files are write-protected use the `-f` (force) option:

```
$ rm -f filename(s)
```

- We can also combine `rm` options. For example, to remove all `.txt` files in the current directory without a prompt in verbose mode, use the following command:

```
$ rm -fv *.txt
```

## **SUDO Permission:**

It prompts to enter their own password. The `sudo` command is a handy tool that allows me as a sysadmin with root access to delegate responsibility for all or a few administrative tasks to other users of the computer. It allows me to perform the delegation without compromising the root password.

## **GKSUDO**

Syntax:

```
$ gksu
```

```
$ gksu [-u <user>] [options] <command>
```

```
$ gksudo [-u <user>] [options] <command>
```

**gksu** is a frontend to **su** and **gksudo** is a frontend to `sudo`. Their primary purpose is to run graphical commands that need root without the need to run an X terminal emulator and using **su** directly.

## **CHMOD**

The `chmod` command is used to change the access mode of a file.

Syntax:

```
$ chmod [reference][operator][mode] file
```

<b><u>Reference</u></b>	<b><u>Class</u></b>	<b><u>Description</u></b>
u	owner	file's owner
g	group	users who are members of the file's group
o	others	users who are neither the file's owner nor members of the file's group
a	all	All three of the above, same as u, g, o

## **CHOWN**

Changes ownership of files and directories in a file system.

Permissions:

- a) User permissions
- b) Group permissions
- c) Other permissions: These users are known as others or the world.

## **1.5 VI EDITOR**

The default editor that comes with the UNIX operating system is called vi (visual editor). Using vi editor, we can edit an existing file or create a new file from scratch. we can also use this editor to just read a text file.

Syntax:

```
$ vi filename
```

Modes of Operation in vi editor There are three modes of operation in vi:

- Command Mode: When vi starts up, it is in Command Mode. This mode is where vi interprets any characters we type as commands and thus does not display them in the window. This mode allows us to move through a file, and to delete, copy, or paste a piece of text. To enter into Command Mode from any other mode, it requires pressing the [Esc] key. If we press [Esc] when we are already in Command Mode, then vi will beep or flash the screen.
- Insert mode: This mode enables us to insert text into the file. Everything that's typed in this mode is interpreted as input and finally, it is put in the file. The vi always starts in command mode. To enter text, we must be in insert mode. To come in insert mode, we simply type i. To get out of insert mode, press the Esc key, which will put us back into command mode.
- Last Line Mode (Escape Mode): Line Mode is invoked by typing a colon [:], while vi is in Command Mode. The cursor will jump to the last line of the screen and vi will wait for a command. This mode enables us to perform tasks such as saving files, executing commands.

## **1.6 GEDIT**

Gedit is the GNOME text editor. While aiming at simplicity and ease of use, gedit is a powerful general-purpose text editor.

Installation:

```
$ sudo apt-add-repository ppa:mc3man/older
```

```
$ sudo apt-get update && sudo apt-get install gedit gedit-plugins
```

gedit-common Syntax:

```
$ gedit filename(with extension)
```



## 1.7 MCEDIT

Internal file editor of GNU Midnight Commander.

Syntax:

```
$ mcedit [-bcCdfhstVx?] [+lineno] [file1] [file2] ...
```

```
$ mcedit [-bcCdfhstVx?] file1:lineno[:] file2:lineno[:] ...
```

mcedit is a link to mc, the main GNU Midnight Commander executable. Executing GNU Midnight Commander under this name runs the internal editor and opens files specified on the command line. The editor is based on the terminal version of cooedit - standalone editor for X Window System.

### Options

+lineno: Go to the line specified by number (do not put a space between the + sign and the number). Several line numbers are allowed but only the last one will be used, and it will be applied to the first file only.

-b: Force black and white display.

-c: Force ANSI color mode on terminals that don't seem to have color support.

-C: <keyword>=<fgcolor>,<bgcolor>,<attributes>:<keyword>= ...

-d: Disable mouse support.

-f: Display the compiled-in search path for GNU Midnight Commander data files.

-t: Force using termcap database instead of terminfo. This option is only applicable if GNU Midnight Commander was compiled with S-Lang library with terminfo support.

-V: Display the version of the program.

-x: Force xterm mode. Used when running on xterm-capable terminals (two screen modes, and able to send mouse escape sequences).

## 1.8 GCC

GCC stands for GNU Compiler Collections which is used to compile mainly C and C++ language. It can also be used to compile Objective C and Objective C++. The most important option required while compiling a source code file is the name of the source program, rest every argument is optional like a warning, debugging, linking libraries, object file etc. The different options of gcc command allow the user to stop the compilation process at different stages.

Syntax:

```
$ gcc [-c|-S|-E] [-std=standard]
```

Example: This will compile the source.c file and give the output file as a.out file which is default name of output file given by gcc compiler, which can be executed using ./a.out \$ gcc source.c

□ **Most useful options with Examples: Here source.c is the C program code file.**

**-o opt:** This will compile the source.c file but instead of giving default name hence executed using ./opt, it will give output file as opt. -o is for output file option.

```
$ gcc source.c -o opt
```

**-Werror:** This will compile the source and show the warning if any error is there in the program, -W is for giving warnings.

```
$ gcc source.c -Werror -o opt
```

**-Wall:** This will check not only for errors but also for all kinds warning like unused variables errors, it is good practice to use this flag while compiling the code.

```
$ gcc source.c -Wall -o opt
```

**-ggdb3:** This command give us permissions to debug the program using gdb which will be described later, -g option is for debugging.

```
$ gcc -ggdb3 source.c -Wall -o opt
```

**-lm:** This command link math.h library to our source file, -l option is used for linking particular library, for math.h we use -lm.

```
$ gcc -Wall source.c -o opt -lm
```

**-std=c11:** This command will use the c11 version of standards for compiling the source.c program, which allows to define variable under loop initializations also using newer standards version is preferred.

```
$ gcc -Wall -std=c11 source.c -o opt
```

**-c:** This command compile the program and give the object file as output, which is used to make libraries.

**-v :** This option is used for the verbose purpose.

## 1.9 G++

g++ command is a GNU c++ compiler invocation command, which is used for preprocessing, compilation, assembly and linking of source code to generate an executable file. The different “options” of g++ command allow us to stop this process at the intermediate stage.

Check g++ compiler version information:

```
$ g++ --version
```

Compile a CPP file to generate executable target file: g++ file\_name command is used to compile and create an executable file a.out (default target name).

Example: Given a simple program to print “Hello World” on standard output with file name hello.cpp

```
$ g++ hello.cpp
```

This compiles and links hello.cpp to produce a default target executable file a.out in present working directory. To run this program, type ./a.out where ./ represents present working directory and a.out is the executable target file.

```
$ ./a.out
```

**g++ -S file\_name:** is used to only compile the file\_name and not assembling or linking. It will generate a file\_name.s assembly source file.

Example:

```
$ g++ -S hello.cpp
```

**g++ -c file\_name:** is used to only compile and assemble the file\_name and not link the object code to produce executable file. It will generate a file\_name.o object code file in present working directory.

Example:

```
$ g++ -c hello.cpp
```

**g++ -o target\_name file\_name:** Compiles and links file\_name and generates executable target file with target\_name (or a.out by default).

Example:

```
$ g++ -o main.exe hello.cpp
```

**g++ -Wall file\_name:** It prints all warning messages that are generated during compilation of file\_name.

Example:

```
$ g++ -Wall hello.cpp
```

## 1.10 FILE TRANSFER PROTOCOL

File Transfer Protocol (FTP) is the commonly used protocol for exchanging files over the Internet. FTP uses the Internet's TCP/IP protocols to enable data transfer. FTP uses a client-server architecture, often secured with SSL/TLS. FTP promotes sharing of files via remote computers with reliable and efficient data transfer.

### How FTP Works?

FTP works in the same way as HTTP for transferring Web pages from a server to a user's browser and SMTP for transferring electronic mail across the Internet in that, like these technologies.

FTP uses a client-server architecture. Users provide authentication using a sign-in protocol, usually a username and password, however some FTP servers may be configured to accept anonymous FTP logins where we don't need to identify ourselves before accessing files. Most often, FTP is secured with SSL/TLS. SSL stands for Secure Sockets Layer and TLS stands for Transport Layer Security.

### How to FTP?

Files can be transferred between two computers using FTP software. The user's computer is called the local host machine and is connected to the Internet. The second machine, called the remote host, is also running FTP software and connected to the Internet.

- The local host machine connects to the remote host's IP address.
- The user would enter a username/password (or use anonymous).
- FTP software may have a GUI, allowing users to drag and drop files between the remote and local host. If not, a series of FTP commands are used to log in to the remote host and transfer files between the machines.

### Common Uses of FTP

FTP is most commonly used to download a file from a server using the Internet or to upload a file to a server (e.g., uploading a web page file to a Web server).

## 1.11 FILEZILLA

The FileZilla software program is a free-to-use (open source) FTP utility, allowing a user to transfer files from a local computer to a remote computer. FileZilla is available as a client version and a server version.



## FileZilla features:

FileZilla has dozens of features, including some of the below popular features.

- **Site Manager** - create and store a list of FTP servers and associated connection data.
- **Directory Comparison** - allows a user to compare the contents of a local and remote directory.
- **File and Folder View** - similar to a file manager, allowing a user to modify files and folders and providing a drag-and-drop capability between local and remote directories.
- **Transfer Queue** - displays the status of file transfers in progress or waiting to process.

## Installing FileZilla in Ubuntu 16.04:

First, we need to add PPA file in our system and then we can install it. Use following commands to complete the installation.

```
$ sudo add-apt-repository ppa:n-muench/programs-ppa
```

```
$ sudo apt-get update
```

```
$ sudo apt-get install filezilla
```

To open FileZilla: We can use GUI icon to start filezilla in our system or using below command on terminal

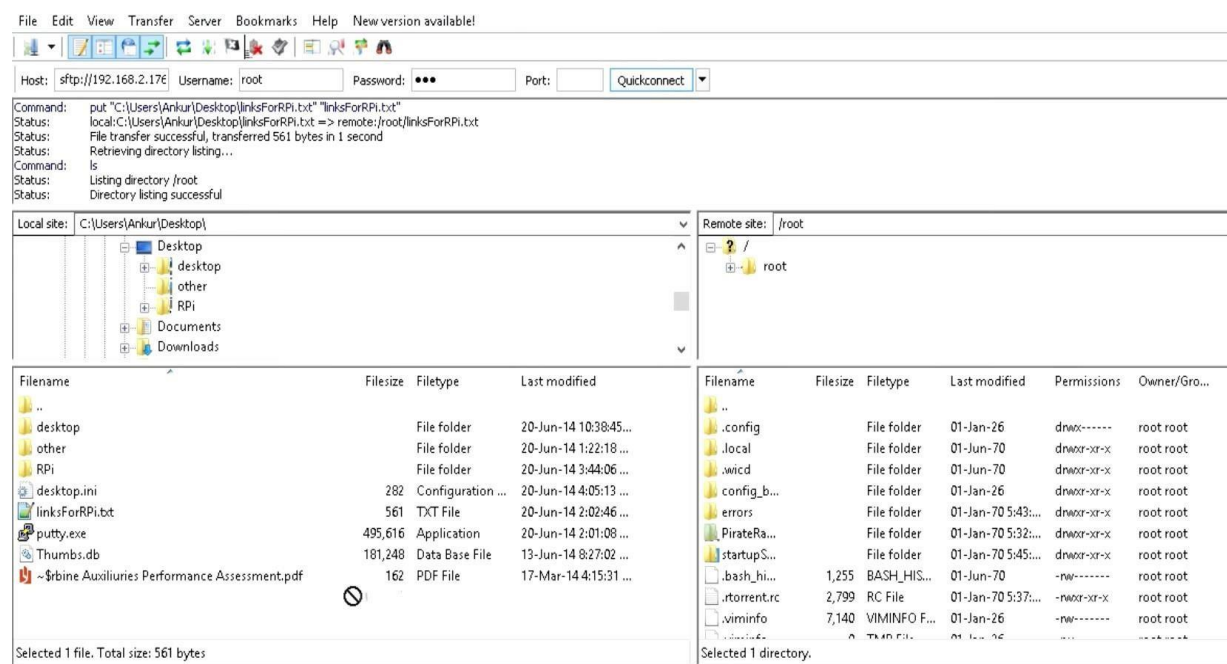
```
$ filezilla @
```

## Transferring files from remote pc to server

Fill in the Host address, Username, Password and Port number. Then click on Quickconnect.

To transfer any file, open the location folder where the file is present and the target location folder on the other side. Double click the file to transfer.

This can be done to transfer files in both directions.



## Chapter 2:

# TURTLEBOT3 WITH ROS

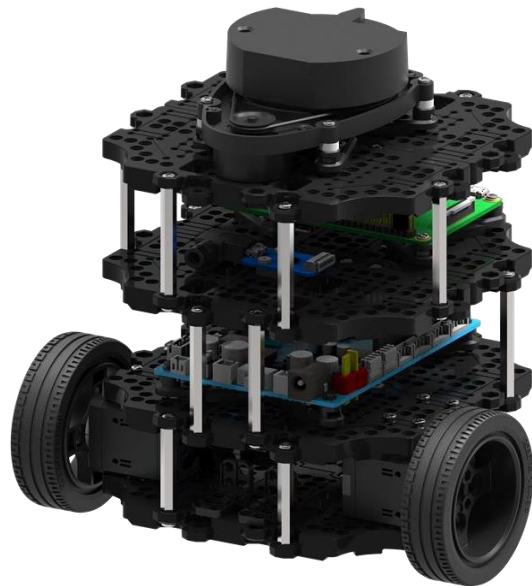
## 2.1 Introduction:

**ROS (Robot Operating System)** is an open-source, meta-operating system for robot. It provides the services which would be expected from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers.

## 2.2 Hardware Description of TurtleBot3 Burger:

TurtleBot is a ROS standard platform robot. Turtle is derived from the Turtle robot, which was driven by the educational computer programming language Logo in 1967. There are 3 versions of the TurtleBot series. TurtleBot3 is a small, affordable, programmable, ROS-based mobile robot for use in education, research, hobby, and product prototyping. The goal of TurtleBot3 is to dramatically reduce the size of the platform and lower the price without having to sacrifice its functionality and quality, while at the same time offering expandability. The TurtleBot3 can be customized into various ways depending reconstruction of the mechanical parts and use optional parts such as the

computer and sensor. In addition, TurtleBot3 is evolved with cost-effective and small-sized SBC that is suitable for robust embedded system, 360 degree distance sensor and 3D printing technology. The TurtleBot3's core technology is SLAM, Navigation and Manipulation, making it suitable for home service robots. The TurtleBot can run SLAM (simultaneous localization and mapping) algorithms to build a map and can drive around our room. Also, it can be controlled remotely from a laptop, joystick or Android-based smartphone.



### Hardware Specification:

Maximum translational velocity	0.22 m/s
Maximum rotational velocity	2.84 rad/s (162.72 deg/s)
Maximum payload	15kg
Size (L x W x H)	138mm x 178mm x 192mm
Weight (+ SBC + Battery + Sensors)	1kg
SBC (Single Board Computers)	Raspberry Pi 3 Model B and B+
MCU	32-bit ARM Cortex®-M7 with FPU (216 MHz, 462 DMIPS)
Actuator	Dynamixel XL430-W250

Power adapter (SMPS)	Input : 100-240V, AC 50/60Hz, 1.5A @max Output : 12V DC, 5A
LDS(Laser Distance Sensor)	360 Laser Distance Sensor LDS-01
IMU	Gyroscope 3 Axis Accelerometer 3 Axis Magnetometer 3 Axis
Power connectors	3.3V / 800mA 5V / 4A 12V / 1A
Expansion pins	GPIO 18 pins Arduino 32 pin
Buttons and Switches	Push buttons x 2, Reset button x 1, Dip switch x 2
Battery	Lithium polymer 11.1V 1800mAh / 19.98Wh 5C

## 2.3 Working with ROS:

The Robot Operating System (ROS) is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms.

### Why was ROS built?

Because creating truly robust, general-purpose robot software is hard. From the robot's perspective, problems that seem trivial to humans often vary wildly between instances of tasks and environments. Dealing with these variations is so hard that no single individual, laboratory, or institution can hope to do it on their own.

As a result, ROS was built from the ground up to encourage collaborative robotics software development. For example, one laboratory might have experts in mapping indoor environments, and could contribute a world-class system for producing maps. Another group might have experts at using maps to navigate, and yet another group might have discovered a computer vision approach that works well for recognizing small objects in clutter. ROS was designed specifically for groups like these to collaborate and build upon each other's work, as is described throughout this site.

### Install ROS on Remote PC:

Run the following command in a terminal window.

```
$ sudo apt-get update
```

```
$ sudo apt-get upgrade
```

```
$ wget https://raw.githubusercontent.com/ROBOTIS-GIT/robotis_tools/master/install_ros_kinetic.sh && chmod 755 ./install_ros_kinetic.sh && bash ./install_ros_kinetic.sh
```

NOTE: In order to check which packages are installed, please check this link out [\*install\\_ros\\_kinetic.sh\*](#)

### Install Dependent ROS Packages on Remote PC –



The next step is to install dependent packages for TurtleBot3 control on Remote PC. So we have to run the following commands in the terminal window.

```
$ sudo apt-get install ros-kinetic-joy ros-kinetic-teleop-twist-joy ros-kinetic-teleop-twist-keyboard ros-kinetic-laser-proc ros-kinetic-rgbd-launch ros-kinetic-depthimage-to-laserscan ros-kinetic-rosserial-arduino ros-kinetic-rosserial-python ros-kinetic-rosserial-server ros-kinetic-rosserial-client ros-kinetic-rosserial-msgs ros-kinetic-amcl ros-kinetic-map-server ros-kinetic-move-base ros-kinetic-urdf ros-kinetic-xacro ros-kinetic-compressed-image-transport ros-kinetic-rqt-image-view ros-kinetic-gmapping ros-kinetic-navigation ros-kinetic-interactive-markers
```

```
$ mkdir ~/catkin_ws/
```

```
$ cd ~/catkin_ws/
```

```
$ mkdir src
```

```
$ cd src
```

```
$ git clone https://github.com/ROBOTIS-GIT/turtlebot3_msgs.git
```

```
$ git clone https://github.com/ROBOTIS-GIT/turtlebot3.git
```

```
$ cd ~/catkin_ws && catkin_make
```

If catkin\_make command is completed without any errors, the preparation for TurtleBot 3 is done.

### Network Configuration on Remote PC-



ROS requires IP addresses in order to communicate between TurtleBot 3 PC and the remote PC. The remote PC and TurtleBot 3 PC should be connected to the same Wi-Fi router. Enter the below command on the terminal window of the remote PC to find out the IP address of the remote PC.

```
$ ifconfig
```

```
enp3s0  Link encap:Ethernet  HWaddr d8:cb:8a:f3:d3:00  
        inet addr:192.168.0.165  Bcast:192.168.0.255  Mask:255.255.255.0  
        inet6 addr: fe80::b5ed:414a:b396:f212/64 Scope:Link  
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1  
        RX packets:118368 errors:0 dropped:0 overruns:0 frame:0  
        TX packets:62573 errors:0 dropped:0 overruns:0 carrier:0  
        collisions:0 txqueuelen:1000  
        RX bytes:114480539 (114.4 MB)  TX bytes:8118317 (8.1 MB)  
        Interrupt:19  
  
lo      Link encap:Local Loopback  
        inet addr:127.0.0.1  Mask:255.0.0.0  
        inet6 addr: ::1/128 Scope:Host  
        UP LOOPBACK RUNNING  MTU:65536  Metric:1  
        RX packets:8912 errors:0 dropped:0 overruns:0 frame:0  
        TX packets:8912 errors:0 dropped:0 overruns:0 carrier:0  
        collisions:0 txqueuelen:1  
        RX bytes:1713294 (1.7 MB)  TX bytes:1713294 (1.7 MB)  
  
wlp2s0  Link encap:Ethernet  HWaddr ac:2b:6e:6d:08:ee  
        inet addr:192.168.0.100  Bcast:192.168.1.255  Mask:255.255.255.0  
        inet6 addr: fe80::7a:d5c:9ca8:bd9c/64 Scope:Link  
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1  
        RX packets:468 errors:0 dropped:0 overruns:0 frame:0  
        TX packets:630 errors:0 dropped:0 overruns:0 carrier:0  
        collisions:0 txqueuelen:1000  
        RX bytes:107986 (107.9 KB)  TX bytes:89522 (89.5 KB)
```

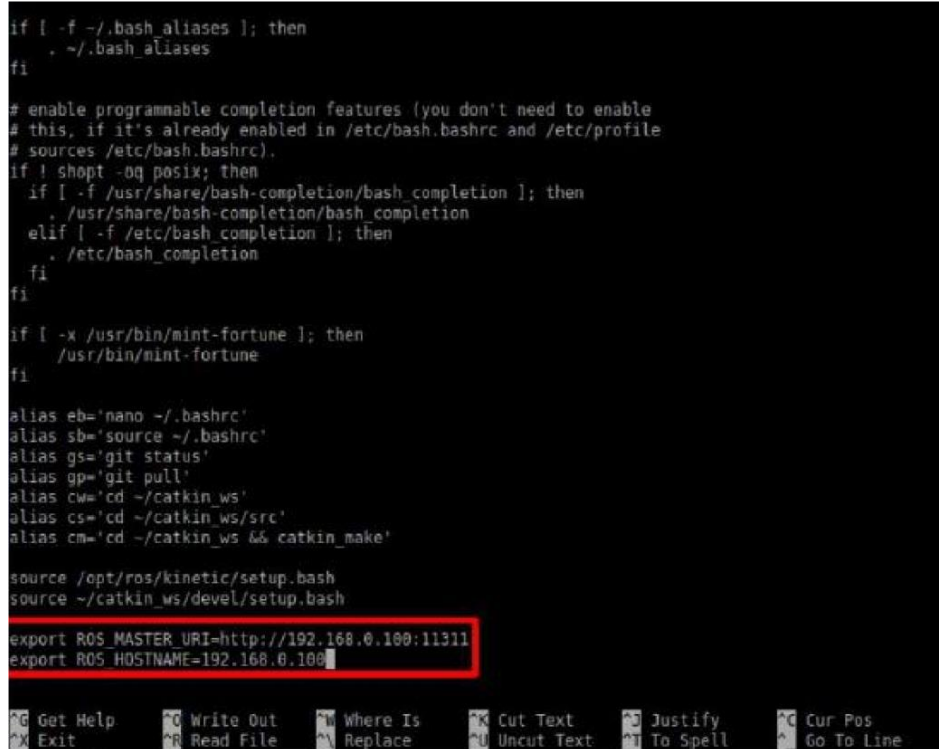
Text strings in the rectangle is the IP address of the Remote PC.

Enter the below command for edit.

```
$ nano ~/.bashrc
```

Then we have to press alt+/ to end line of the file.

Modify the address of localhost in the ROS\_MASTER\_URI and ROS\_HOSTNAME with the IP address acquired from the above terminal window.



```
if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
    if [ -f /usr/share/bash-completion/bash_completion ]; then
        . /usr/share/bash-completion/bash_completion
    elif [ -f /etc/bash_completion ]; then
        . /etc/bash_completion
    fi
fi

if [ -x /usr/bin/mint-fortune ]; then
    /usr/bin/mint-fortune
fi

alias eb='nano ~/.bashrc'
alias sb='source ~/.bashrc'
alias gs='git status'
alias gp='git pull'
alias cw='cd ~/catkin_ws'
alias cs='cd ~/catkin_ws/src'
alias cm='cd ~/catkin_ws && catkin_make'

source /opt/ros/kinetic/setup.bash
source ~/catkin_ws/devel/setup.bash

export ROS_MASTER_URI=http://192.168.0.100:11311
export ROS_HOSTNAME=192.168.0.100
```

Here we have to modify both ROS\_MASTER\_URI and ROS\_HOSTNAME with the same IP of the remote PC IP. Example-

```
export ROS_MASTER_URI=http://REMOTE_PC_IP:11311
```

```
export ROS_HOSTNAME= REMOTE_PC_IP
```

Then, source the bashrc with below command.

```
$ source ~/.bashrc
```

## Difference between library and executable?

The main difference between an **executable** and a **library** is that an **executable** has a fixed starting point. A **library** can have multiple entry points, and there is no single point defined as the starting point. An **executable** could contain any code that could exist **in a library**.

Location: /opt/ros

## ROS Packages and manifest

A package can contain programs written in Python or C++. Another class of packages in ROS called metapackages.

### ROS manifest

Each package contains a manifest named package.xml Extensible Markup Language(xml)

The manifest defines properties about the package such as the package name, version numbers, authors, maintainers and any dependencies on other packages.

## **Exploring the ROS packages**

1. **rospack**: used for info about a package.
2. **roscd**: used to navigate the ROS directories.
3. **rosls**: used to list directories and files in a package directory.

## **Common User Tools**

Location: `opt/ros/kinetic`

The following tools are built when a top-level make is called in `$ROS_ROOT`. They are installed to `$ROS_ROOT/bin`, which should have been added to `PATH` variable as part of the installation process.

### **rosbag**

`rosbag` is a command-line tool for performing various operations on ROS bag files, including playing, recording, and validating.

### **rosbash**

`rosbash` is not a command, but rather a suite of commands and functionality. It requires that we source the contents of the `rosbash` file

```
source $ROS_ROOT/tools/rosbash/rosbash
```

which, if we followed the installation guide, should already be done by our `bashrc` file. `rosbash` provides the commands `roscd` and `roscd`, in addition to adding correct tab-completion functionality to `roscd`, `roscd`, `rosmake`, and `rosls`.

### **roscd**

`roscd` is part of the `rosbash` suite. It allows us to `cd` directly to a package, stack, or common location by name rather than having to know the package path.

Usage:

```
roscd locationname[/subdir]
```

Example:

```
roscd roscpp/include
```

`roscd` without an argument will take us to `$ROS_ROOT`. In addition to our packages and stacks, there are some common locations, "log", and "test\_results" which will take us directly to those locations.

For advanced users, we can extend `roscd` with our own keywords by modifying the `$ROS_LOCATIONS` environment variable to contain a colon-separated list of keys and locations that will be included in the `roscd` path. For example,

```
export $ROS_LOCATIONS="rospkg=/path/to/rospkg:stairpkg=/path/to/stairpkg"
```

Prior to ROS-0.8, we could `roscd` to the first directory on our `$ROS_PACKAGE_PATH` using the `pkg` keyword. This functionality can be restored with:

```
export $ROS_LOCATIONS="pkg=$ROS_PACKAGE_PATH"
```

## **rosclean**

Cleanup filesystem resources (e.g. log files) created by ROS.

## **roscore**

[roscore](#) runs the ROS Core Stack ([Master](#), [Parameter Server](#), [roscout](#), etc.)

## **rosdep**

Documented at [rosdep](#), this installs system dependencies

Usage:

```
rosdep install PACKAGE_NAME
```

## **roscd**

`roscd` is part of the [rosbash](#) suite. It allows us to directly edit a file within a package by package name rather than having to know the package path.

Usage:

```
roscd packagename filename
```

Example:

```
roscd roscpp ros.h
```

If the filename is not uniquely defined within the package, a menu will prompt us to choose which of the possible files we want to edit. `roscd` will open the editor defined in our `$EDITOR` environmental variable, or else default to `vim`.

## **roscrcat-pkg**

`roscrcat-pkg` creates common [Manifest](#), [CMakeLists](#), [Doxygen](#) and other files necessary for a new ROS package. It is part of the [roscrcat](#) package.

## **roscrcat-stack**

`roscrcat-stack` creates common [Stack Manifest](#), [CMakeLists](#) and other files necessary for a new ROS stack. It is part of the [roscrcat](#) package.

## **roslaunch**

`roslaunch` allows us to run an executable in an arbitrary package without having to `cd` (or `roscd`) there first.

Usage:

```
roslaunch package executable
```

Example:

```
roslaunch roscpp_tutorials talker
```

## **roslaunch**

[roslaunch](#) launches a set of nodes from an XML configuration file and includes support for

launching on remote machines. More documentation is available on the [roslaunch page](#).

### **rosllocate**

[rosllocate](#) finds the repository that a ROS package is stored in, e.g. `rosllocate svn tf`. It makes it easy to quickly checkout the source of a package: `svn co `rosllocate svn tf``. More documentation is available on the [rosllocate page](#).

### **rosmake**

It is a ROS dependency aware build tool which can be used to build all dependencies in the correct order.

### **rosmmsg**

[rosmmsg](#) displays Message data structure definitions. More documentation is available on the [rosmmsg page](#).

### **rosmnode**

[rosmnode](#) displays runtime node information and lets we ping nodes to check connectivity. More documentation is available on the [rosmnode page](#).

### **rosmpack**

Command line tool to retrieve information about ROS packages available on the file system. Commands ranging from locating ROS packages in the filesystem, to calculating the dependency tree of packages.

### **rosmparam**

[rosmparam](#) enables getting and setting parameter server values from the command-line using YAML encoded text.

### **rosmrv**

[rosmrv](#) displays Service [srv](#) data structure definitions. More documentation is available on the [rosmrv page](#).

### **rosmervice**

[rosmervice](#) displays run-time information about [Services](#) and also lets we print out messages being sent to a topic. More documentation is available on the [rosmervice page](#).

### **rosmstack**

For retrieving information about ROS stacks available on the file system.

### **rostopic**

[rostopic](#) displays run-time information about [Topics](#) and also lets we print out messages being sent to a topic. More documentation is available on the [rostopic page](#).

### **rosmversion**

Reports the version of a ROS stack.

## □ Graphical tools

The ROS graphical tools often require additional dependencies before they can be used, such as graphviz and Python GTK. We can use `bash < (rosdep satisfy PACKAGE_NAME)` to quickly install the dependencies for these tools.

### **rqt\_bag**

[rqt\\_bag](#) is a graphical tool for viewing data in ROS [bag files](#).

### **rqt\_deps**

`rqt_deps` generates a PDF of ROS dependencies.

### **rqt\_graph**

`rqt_graph` displays an interactive graph of ROS nodes and topics. See the [rosgraph](#) package for documentation.

### **rqt\_plot**

[rqt\\_plot](#) plots numerical data on a ROS topic over time.

## **Less-used tools**

The following tools may be commonly used by internal tools, but aren't often used by end users.

### **gendeps**

loads the dependency tree for a specified message.

## **grep command in Unix/Linux**

The `grep` filter searches a file for a particular pattern of characters, and displays all lines that contain that pattern. The pattern that is searched in the file is referred to as the regular expression (`grep` stands for globally search for regular expression and print out).

**Syntax:**

**grep [options] pattern [files]**

### **Options Description**

- c** : This prints only a count of the lines that match a pattern
- h** : Display the matched lines, but do not display the filenames.
- i** : Ignores, case for matching
- l** : Displays list of a filenames only.
- n** : Display the matched lines and their line numbers.
- v** : This prints out all the lines that do not matches the pattern
- e exp** : Specifies expression with this option. Can use multiple times.
- f file** : Takes patterns from file, one per line.
- E** : Treats pattern as an extended regular expression (ERE)
- w** : Match whole word
- o** : Print only the matched parts of a matching line, with each such part on a separate output line.

## Searching for a string in a list of files and storing the output in another file:

```
$ grep -r "string" /address | tee ABC.txt
```

## Finding the main function in roslaunch: `grep -iw "main"`

/home/dns/Desktop/1/Roslaunch (Find the path to ROSlaunch) o/p:

```
/opt/ros/kinetic/Lab/Python2.7/dist-packages/roslaunch/config.py:432
```

```
cd /opt/ros/kinetic/Lab/Python2.7/dist-packages/roslaunch/config.py:432
```

using the command below we can get file name of the main file

```
$ grep -Hri "main("
```

to get the path

```
$ grep -ril "main("
```

## Installing RVIZ

```
$ sudo apt-get update
```

```
$ sudo apt-get install rviz
```

## 2D Nav button

Need to find the 2D Nav main function <https://github.com/ros-visualisation/rviz/tree/kinetic->  
version of RVIZ is 1.17.12 (rosversion)

- To debug the source file, we need to install **qt creator**.

```
$ sudo apt-get update
```

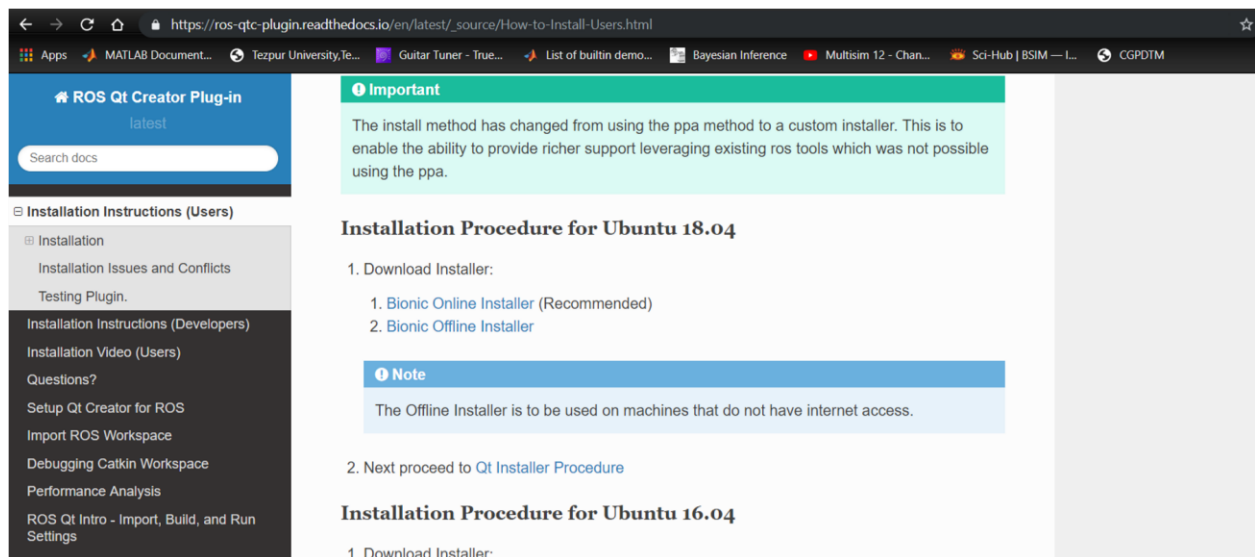
```
$ sudo apt-get install qtcreator
```

## Installing ROS plugin

[https://ros-qtc-plugin.readthedocs.io/en/latest/\\_source/How-to-Install-Users.html](https://ros-qtc-plugin.readthedocs.io/en/latest/_source/How-to-Install-Users.html) Scroll down to

Installation Procedure for Ubuntu 16.04

Head to Xenial Offline installer



- How to import existing project in qt creator?

New Project >> Import Project >> Import Existing Project

Then fill up the below details

Project Name:

Location:

File Selection procedure: Select all >> Next >> Finish

Additional things to learn:

1. How to use CMake with qt creator for building C++ Programs – Youtube.
2. Programming Knowledge -Youtube Channel.

Problems:

1. While debugging CMakeLists.txt, following error occurs:  
-No rule to make target 'all' stop.
2. The process “/usr/bin/make” exited with code 2.

## 2.4 Mapping, Localization and Navigation:

To navigate a robot we need

1. A map
2. A localization module
3. A path planning module



## **Map:**

A map is a representation of the environment where the robot is operating. It should contain enough information to accomplish a task of interest.

### **Representations of a map:**

1. Metric
  - (i) Grid Based
  - (ii) Feature Based
2. Hybrid
3. Topological
4. Hybrid

## **Localization:**

To determine the current robot position, measurements up to the current instant and a map.

## **Path Planning:**

To determine (if it exists) a path to reach a given goal location given a localized robot and a map of traversable regions.

## **ROS Navigation**

A 2D navigation stack takes in information from odometry, sensor streams, and a goal pose and outputs safe velocity commands that are sent to a mobile base.

Odometry is the use of data from motion sensors to estimate change in position over time. It is used in robotics by some legged or wheeled robots to estimate their position relative to a starting location.

The Navigation Stack needs to be configured for the shape and dynamics of a robot to perform at a high level. There are three main hardware requirements that restrict its use:

1. It is meant for both differential drive and holonomic wheeled robots only.
2. It requires a planar laser mounted somewhere on the mobile base. This laser is used for map building and localization.
3. The Navigation Stack was developed on a square robot, so its performance will be best on robots that are nearly square or circular.

As a pre-requisite for navigation stack use:

1. The robot must be running ROS
2. Have a tf transform tree in place
3. Publish sensor data using the correct ROS Message types.

## **2.5 SLAM:**

In navigation, robotic mapping and odometry for virtual reality or augmented reality, simultaneous localization and mapping (SLAM) is the computational problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's

location within it.

SLAM estimates the map of the environment and the trajectory of a moving device using a sequence of sensor measurements.

There are hundreds of SLAM algorithms around. ROS uses GMapping, which implements a particle filter to track the robot trajectories.

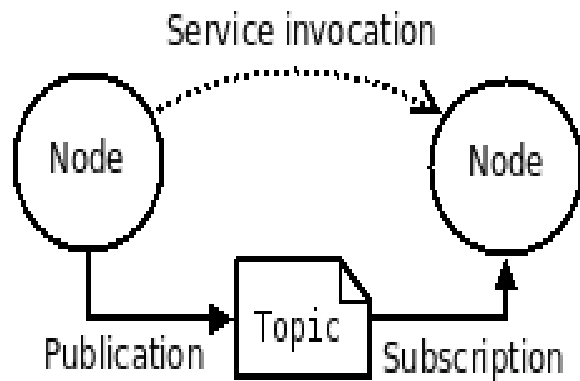
### **Steps for Navigation of the TurtleBot3:**

1. Setup the Navigation Stack for TurtleBot
2. SLAM Map Building with TurtleBot
3. Autonomous Navigation of a Known Map with TurtleBot
4. Setup the Navigation Stack for TurtleBot
5. SLAM Map Building with TurtleBot
6. Autonomous Navigation of a Known Map with TurtleBot

## **2.6 Publisher & Subscriber Node using DWM:**

‘Node’ is the ROS term for an executable that is connected to the ROS network. A publisher is a "talker" node which continually broadcasts a message via a transport system called ‘topic’. A subscriber is a "listener" node which continually receives the message broadcasted by the subscriber on the same topic.

The message here simply refers to a data structure, comprising typed fields. Standard primitive types (integer, floating point, boolean, etc.) are supported, as are arrays of primitive types.



In a topic-based system, messages are published to "topics" or named logical channels. Subscribers in a topic-based system will receive all messages published to the topics to which they subscribe, and all subscribers to a topic will receive the same messages. The publisher is responsible for defining the classes of messages to which subscribers can subscribe.

### **How to perform ‘Publishing’ and ‘Subscribing’:**

#### **1. Creating a package in ROS:**

```
$ catkin_create_pkg beginner_tutorials std_msgs roscpp
```

#### **2. Building a catkin workspace and sourcing the setup file:**

```
$ cd ~/catkin_ws
```

```
$ catkin_make
```

### **3. Change directory to beginner tutorials:**

```
$ roscd beginner_tutorials
```

### **4. Writing codes for the Publisher and Subscriber nodes:**

Please refer to DVD for the C++ codes for publisher as “talker.cpp” and subscriber as “listener.cpp”.

### **5. Building the nodes:**

Inside the beginner\_tutorials open the CMakeList.txt file and add the lines below:

```
add_executable(talker src/talker.cpp)

target_link_libraries(talker ${catkin_LIBRARIES})

add_dependencies(talker beginner_tutorials_generate_messages_cpp)

add_executable(listener src/listener.cpp)

target_link_libraries(listener ${catkin_LIBRARIES})

add_dependencies(listener beginner_tutorials_generate_messages_cpp)
```

This will create two executables, talker and listener, which by default will go into package directory of devel space.

### **6. Setting permissions:**

```
$ sudo chmod -R 777 /dev/ttyACM0
```

### **7. Finally compile and run the programs:**

```
$ catkin_make
```

```
$ rosrn beginner_tutorials talker.cpp
```

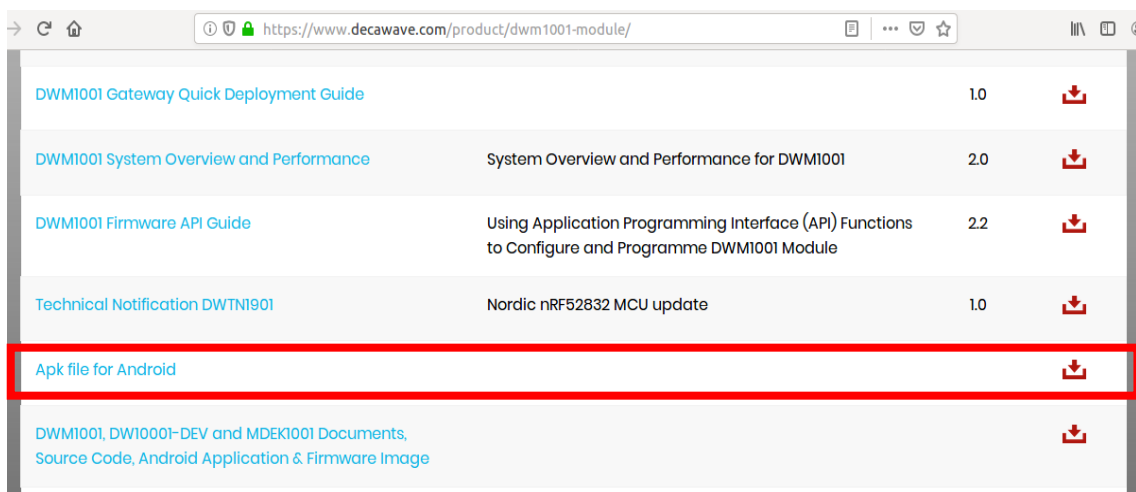
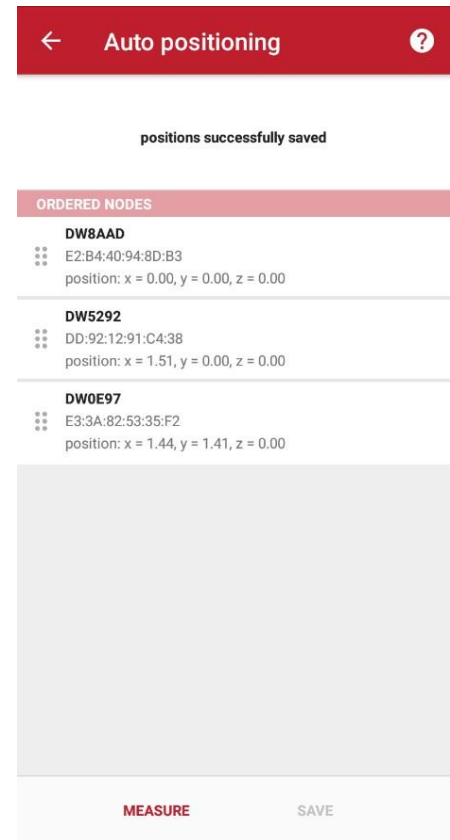
## 2.7 DecaWave Module (DWM1001):

The DWM1001 is a device that works on Ultra Wide Band (UWB). Ultra-Wide Band (UWB) is a communication method used in wireless networking that uses low power consumption to achieve high bandwidth connections. In other words, it's meant to transmit a lot of data over a short distance without using too much power.

In our project we used the UART bus to read from the decawave module through a USB to serial converter. And the communication program used is Minicom. Minicom is a text-based serial port communications program used to talk to external RS-232 devices such as mobile phones, routers, and serial console ports.

We have used four DWM for the UWB communication. The tag is attached to the PC, while the three anchor are place at three different corners of the room. One of the anchors is set at a position of  $x = 0.00$ ,  $y = 0.00$ ,  $z = 0.00$  which is achieved through the mobile app DRTL5 from Decawave (refer to figure on the right). This is to be noted that the minimum number of anchors required for the UWB communication is three, like GPS.

A host USB driver must use one of the existing control models. The Abstract control model or ACM, lets the modem hardware perform the analog functions. When developping on a USB-enabled embedded microcontroller that needs to exchange data with a computer over USB, it is tempting to use a standardized way of communication which is well supported by virtually every operating system.



## Reading from DecaWave:

### 1. Installing Minicom:

```
$ sudo apt-get install minicom
```

### 2. Setting up minicom:

The -s option is used to setup minicom. Type the following command at shell prompt:

```
$ minicom -s
```

### 3. Running Minicom in terminal

On the welcome screen, hit double enter. Type **les**

```
Last login: Tue Jul  2 17:45:17 2019 from 10.240.9.100
-bash: /opt/ros/kinetic/setup.bash: No such file or directory
ubuntu@tegra-ubuntu:~$ sudo minicom -s
[sudo] password for ubuntu:
```

```
+-----+
| A -   Serial Device       : /dev/ttyACM0 |
| B - Lockfile Location    : /var/lock     |
| C -   Callin Program     :               |
| D -   Callout Program    :               |
| E -   Bps/Par/Bits       : 115200 8N1    |
| F - Hardware Flow Control : Yes          |
| G - Software Flow Control : No          |
|                                     |
|   Change which setting? █             |
+-----+
| Screen and keyboard |
| Save setup as dfl   |
| Save setup as..    |
| Exit                |
| Exit from Minicom  |
+-----+
```

```
Welcome to minicom 2.7

OPTIONS: I18n
Compiled on Nov 15 2018, 20:20:44.
Port /dev/ttyACM0, 18:18:24

Press CTRL-A Z for help on special keys


DWM1001 TWR Real Time Location System

Copyright : 2016-2019 LEAPS and Decawave
License   : Please visit https://decawave.com/dwm1001_license
Compiled  : Mar 27 2019 03:35:59

Help      : ? or help

dwm> les█
```

Data read from DWM has the following format:

[ pos.x, pos.y, pos.z, pos.qf ]

```
8AAD[0.00,0.00,0.00]=0.80 5292[1.51,0.00,0.00]=1.44 0E97[1.44,1.41,0.00]=1.41 le_us=671 est[0.28,0.64,0.23,62]
8AAD[0.00,0.00,0.00]=0.83 5292[1.51,0.00,0.00]=1.41 0E97[1.44,1.41,0.00]=1.41 le_us=671 est[0.29,0.64,0.27,58]
8AAD[0.00,0.00,0.00]=0.76 5292[1.51,0.00,0.00]=1.36 0E97[1.44,1.41,0.00]=1.41 le_us=671 est[0.30,0.63,0.29,69]
8AAD[0.00,0.00,0.00]=0.77 5292[1.51,0.00,0.00]=1.42 0E97[1.44,1.41,0.00]=1.41 le_us=823 est[0.29,0.64,0.29,67]
8AAD[0.00,0.00,0.00]=0.77 5292[1.51,0.00,0.00]=1.42 0E97[1.44,1.41,0.00]=1.41 le_us=671 est[0.29,0.64,0.29,67]
8AAD[0.00,0.00,0.00]=0.80 5292[1.51,0.00,0.00]=1.44 0E97[1.44,1.41,0.00]=1.36 le_us=671 est[0.29,0.67,0.26,72]
8AAD[0.00,0.00,0.00]=0.82 0E97[1.44,1.41,0.00]=1.47 5292[1.51,0.00,0.00]=1.42 le_us=732 est[0.29,0.65,0.08,52]
8AAD[0.00,0.00,0.00]=0.81 0E97[1.44,1.41,0.00]=1.42 5292[1.51,0.00,0.00]=1.37 le_us=701 est[0.31,0.64,-0.04,60]
8AAD[0.00,0.00,0.00]=0.78 5292[1.51,0.00,0.00]=1.42 0E97[1.44,1.41,0.00]=1.38 le_us=671 est[0.30,0.65,0.02,71]
8AAD[0.00,0.00,0.00]=0.79 0E97[1.44,1.41,0.00]=1.48 5292[1.51,0.00,0.00]=1.42 le_us=701 est[0.30,0.64,-0.09,53]
8AAD[0.00,0.00,0.00]=0.78 5292[1.51,0.00,0.00]=1.38 0E97[1.44,1.41,0.00]=1.41 le_us=701 est[0.30,0.63,0.01,67]
8AAD[0.00,0.00,0.00]=0.79 5292[1.51,0.00,0.00]=1.41 0E97[1.44,1.41,0.00]=1.35 le_us=671 est[0.30,0.65,0.06,74]
8AAD[0.00,0.00,0.00]=0.77 0E97[1.44,1.41,0.00]=1.46 5292[1.51,0.00,0.00]=1.39 le_us=732 est[0.31,0.63,-0.06,60]
8AAD[0.00,0.00,0.00]=0.79 5292[1.51,0.00,0.00]=1.41 0E97[1.44,1.41,0.00]=1.38 le_us=671 est[0.30,0.65,0.02,70]
8AAD[0.00,0.00,0.00]=0.78 5292[1.51,0.00,0.00]=1.40 0E97[1.44,1.41,0.00]=1.38 le_us=671 est[0.31,0.65,0.09,71]
8AAD[0.00,0.00,0.00]=0.78 0E97[1.44,1.41,0.00]=1.41 5292[1.51,0.00,0.00]=1.40 le_us=701 est[0.31,0.65,-0.02,65]
8AAD[0.00,0.00,0.00]=0.81 5292[1.51,0.00,0.00]=1.40 0E97[1.44,1.41,0.00]=1.43 le_us=671 est[0.31,0.64,0.09,59]
8AAD[0.00,0.00,0.00]=0.82 5292[1.51,0.00,0.00]=1.46 0E97[1.44,1.41,0.00]=1.41 le_us=671 est[0.30,0.66,0.14,62]
8AAD[0.00,0.00,0.00]=0.78 0E97[1.44,1.41,0.00]=1.37 5292[1.51,0.00,0.00]=1.36 le_us=732 est[0.31,0.65,0.04,74]
8AAD[0.00,0.00,0.00]=0.79 0E97[1.44,1.41,0.00]=1.47 5292[1.51,0.00,0.00]=1.39 le_us=701 est[0.31,0.63,-0.08,54]
8AAD[0.00,0.00,0.00]=0.78 5292[1.51,0.00,0.00]=1.40 0E97[1.44,1.41,0.00]=1.38 le_us=671 est[0.31,0.64,-0.00,72]
8AAD[0.00,0.00,0.00]=0.76 5292[1.51,0.00,0.00]=1.40 0E97[1.44,1.41,0.00]=1.37 le_us=671 est[0.31,0.65,0.04,76]
8AAD[0.00,0.00,0.00]=0.78 0E97[1.44,1.41,0.00]=1.41 5292[1.51,0.00,0.00]=1.39 le_us=732 est[0.31,0.65,-0.04,68]
8AAD[0.00,0.00,0.00]=0.84 5292[1.51,0.00,0.00]=1.43 0E97[1.44,1.41,0.00]=1.38 le_us=671 est[0.31,0.66,0.05,62]
```

```
CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7 | VT102 | Offline | ttyACM0
```

The position of the DWM set as the tag is given with respect to the DWMs set as the anchors.

Publishing the x and y coordinates of the tag read from DWM using C++

The data read from DWM is stored in the /dev/ttyACM0 file in Linux in the format as shown in fig: showing the terminal window. We write a C++ code to read only the x and y coordinates of the tag and ultimately publish it.

## **Chapter 3:**

# **Design of Our Own Navigation Robot**

### 3.1 Introduction:

Since working with ROS is not so comfortable for us in this project, so we decided to shift our working base from ROS to a simple microcontroller-based robot using the NVIDIA Jetson Tx2 module. In this robot, we assembled four DC motors, which are controlled by the Atmega16 microcontroller and fitted on a chassis. The Atmega16 microcontroller sits on a PCB along with its other integrating components. On top of the chassis, the NVIDIA Jetson Tx2 module is fitted, which controls the robot as a whole. A DWM1001 Development Board is added to the system for the navigation purpose. Moreover, the bot is accompanied by an IMU sensor for orientation and movement.

### 3.2 Hardware Description:

#### Components:

1. Chassis
2. Geared DC motors and wheels
3. Atmega 16 microcontroller board
4. Nvidia Jetson Tx2 board
5. DWM1001 development board
6. IMU sensor
7. USB to Serial Port

#### Chassis:

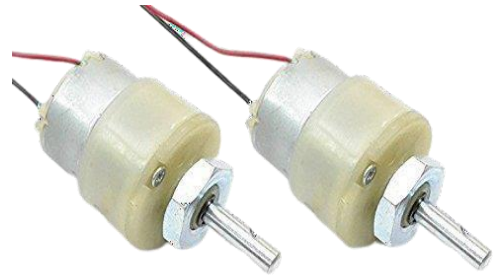
The chassis is the backbone of the ROBOT. Without a proper chassis the robot is not fit for the tracks. Great emphasis has to be laid on the design of a structurally sound chassis design which will not give in under the extreme pressures a robot has to go through. In our robot, we have used a chassis of dimension 22×15×6 cm where four motors can be mounted.





## Geared DC Motors and Wheels:

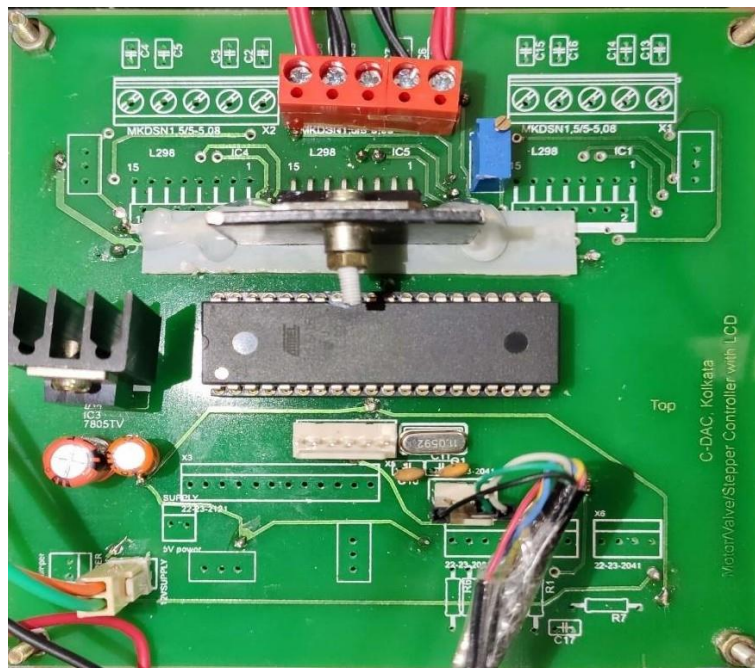
A Geared DC Motor is made by attaching a gear assembly to an ordinary DC motor. This will increase the torque by decreasing the speed of motor. The speed of motor is counted in terms of RPM, rotations of shaft per minute. The speed can be reduced to any desired RPM by using the correct combination of gears.



In this robot, we have used four motors which are geared down to 45 RPM. The gears are fixed on hardened steel spindles. The output shaft rotates in a plastic bushing. The whole assembly is covered with a plastic ring. Gearbox is sealed and lubricated with lithium grease. The motor is screwed to the gear box from inside.

The motors are connected with four plastic tracked wheels of diameter 11cm. The wheels are tightened to the motor shaft with screws.

## AVR microcontroller-based motor driver board:



AVR is family of microcontrollers developed by Atmel in 1996. It is a single chip computer that comes with CPU, ROM, RAM, EEPROM, Timers, Counters, ADC and four 8-bit ports called PORTA, PORTB, PORTC, PORTD where each port consists of 8 I/O pins. Atmega16 is a 40-pin low power microcontroller which is developed using CMOS technology. CMOS is an advanced technology which is mainly used for developing integrated circuits. It comes with low power consumption and high noise immunity. Atmega16 is an 8-bit controller based on AVR advanced RISC (Reduced Instruction Set Computing) architecture.

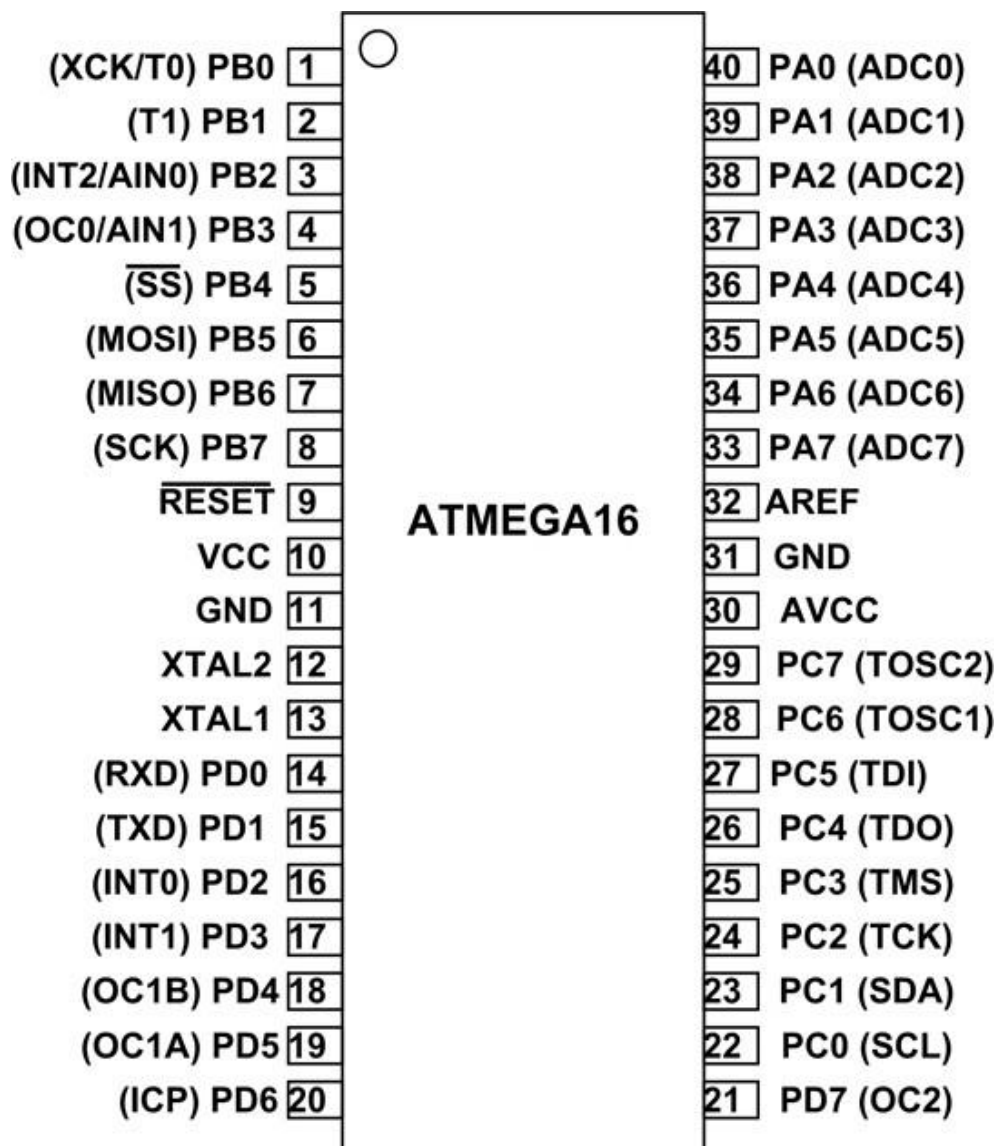


### **Atmega16 microcontroller:**

Atmega16 has built-in registers that are used to make a connection between the CPU and external peripherals devices.

CPU has no direct connection with external devices. It can take input by reading registers and give output by writing registers. Atmega16 comes with two 8-bit timers and one 16-bit timer. All these timers can be used as counters when they are optimized to count the external signal. It can work on a maximum frequency of 16MHz.

### **Pin configuration of Atmega16 microcontroller:**



## Pin Description of Atmega16 microcontroller:

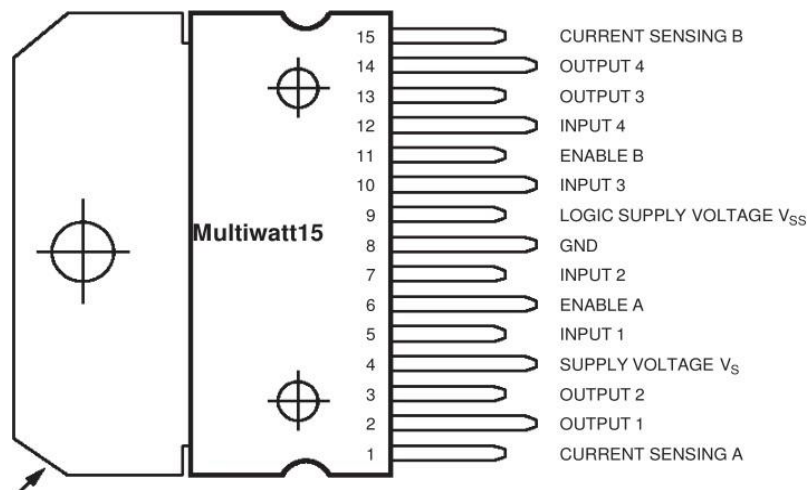
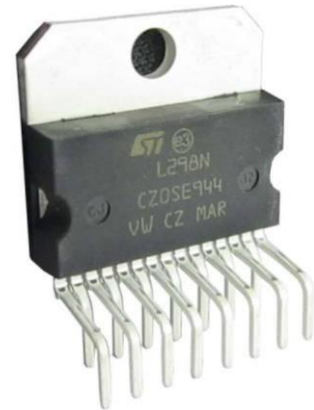
PIN1	I/O , T0 ( Timer0 External Counter Input) , XCK : USART External Clock I/O
PIN2	I/O, T1 (Timer1 External Counter Input)
PIN3	I/O, AIN0: Analog Comparator Positive Input , INT2: External Interrupt 2 Input
PIN4	I/O, AIN1: Analog Comparator Negative Input, OC0 : Timer0 Output Compare Match Output
PIN9	Reset Pin, Active Low Reset
PIN10	VCC=+5V
PIN11	GND
PIN12	XTAL2
PIN13	XTAL1
PIN14	(TXD) ,I/O Pin 1,USART Serial Communication Interface
PIN15	(INT0),I/O Pin 2, External Interrupt INT0
PIN16	(INT1),I/O Pin 3, External Interrupt INT1
PIN17	(INT1),I/O Pin 3, External Interrupt INT1

PIN18	(OC1B),I/O Pin 4, PWM Channel Outputs
PIN19	(OC1A),I/O Pin 5, PWM Channel Outputs
PIN20	(ICP), I/O Pin 6, Timer/Counter1 Input Capture Pin
PIN21	(OC2),I/O Pin 7,Timer/Counter2 Output Compare Match Output
PIN22	(SCL),I/O Pin 0,TWI Interface
PIN23	(SDA),I/O Pin 1,TWI Interface
PIN24-27	JTAG INTERFACE
PIN28	(TOSC1),I/O Pin 6,Timer Oscillator Pin 1
PIN29	(TOSC2),I/O Pin 7,Timer Oscillator Pin 2
PIN30	AVCC (for ADC)
PIN31	GND (for ADC)
PIN33-40	PAX: I/O, ADCx (Where x is 7 – 0)

## Other board elements:

### **L298N:**

The L298N is an integrated monolithic circuit in a 15- lead multi-watt and PowerSO20 packages. It is a high voltage, high current dual full-bridge driver de-signed to accept standard TTL logic level sand drive inductive loads such as relays, solenoids, DC and stepping motors. Two enable inputs are provided to enable or disable the device independently of the in-put signals. The emitters of the lower transistors of each bridge are connected together rand the corresponding external terminal can be used for the connection of an external sensing resistor. An additional Supply input is provided so that the logic works at a lower voltage.



### **PIN FUNCTION:**

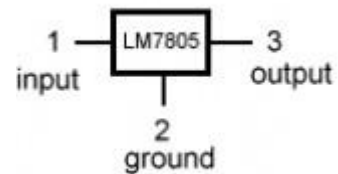
MW.15	PowerSO	Name	Function
1;15	2;19	Sense A; Sense B	Between this pin and ground is connected the sense resistor to control the current of the load.
2;3	4;5	Out 1; Out 2	Outputs of the Bridge A; the current that flows through the load connected between these two pins is monitored at pin 1.
4	6	V <sub>S</sub>	Supply Voltage for the Power Output Stages. A non-inductive 100nF capacitor must be connected between this pin and ground.
5;7	7;9	Input 1; Input 2	TTL Compatible Inputs of the Bridge A.
6;11	8;14	Enable A; Enable B	TTL Compatible Enable Input: the L state disables the bridge A (enable A) and/or the bridge B (enable B).
8	1,10,11,20	GND	Ground.
9	12	V <sub>SS</sub>	Supply Voltage for the Logic Blocks. A100nF capacitor must be connected between this pin and ground.
10; 12	13;15	Input 3; Input 4	TTL Compatible Inputs of the Bridge B.
13; 14	16;17	Out 3; Out 4	Outputs of the Bridge B. The current that flows through the load connected between these two pins is monitored at pin 15.

## LM7805:

Voltage sources in a circuit may have fluctuations resulting in not providing fixed voltage outputs. A voltage regulator IC maintains the output voltage at a constant value. 7805 IC, a member of 78xx series of fixed linear voltage regulators used to maintain such fluctuations, is a popular voltage regulator integrated circuit (IC). The xx in 78xx indicates the output voltage it provides. 7805 IC provides +5 volts regulated power supply with provisions to add a heat sink.



### Pin Details of 7805 IC



Pin No.	Pin	Function	Description
1	INPUT	Input voltage (7V-35V)	In this pin of the IC positive unregulated voltage is given in regulation.
2	GROUND	Ground (0V)	In this pin where the ground is given. This pin is neutral for equally the input and output.
3	OUTPUT	Regulated output; 5V (4.8V-5.2V)	The output of the regulated 5V volt is taken out at this pin of the IC regulator.



## Crystal (11.0592 MHz):

This is a low cost crystal oscillator with oscillation frequency of 11.0592 Mhz. Crystal are normally required to provide clock pulses to us microcontroller or other IC's which require external clock source. An oscillator crystal has two electrically conductive plates, with a slice or tuning fork of quartz crystal sandwiched between them. The crystal oscillator circuit sustains oscillation by taking a voltage signal from the quartz resonator, amplifying it, and feeding it back to the resonator.

Quartz has the further advantage that its elastic constants and its size change in such a way that the frequency dependence on temperature can be very low.



## Nvidia Jetson Tx2 Board:



The Jetson TX2 Developer Kit gives a fast, easy way to develop hardware and software for the Jetson TX2 AI supercomputer on a module. It exposes the hardware capabilities and interfaces of the developer board, comes with design guides and other documentation, and is pre-flashed with a Linux development environment. It also supports NVIDIA Jetpack—a complete SDK that includes the BSP, libraries for deep learning, computer vision, GPU computing, multimedia processing, and much more.

## Processing Components

- ☐ Dual-core NVIDIA Denver2 + quad-core ARM Cortex-A57
- ☐ 256-core Pascal GPU
- ☐ 8GB LPDDR4, 128-bit interface
- ☐ 32GB eMMC
- ☐ 4kp60 H.264/H.265 encoder & decoder
- ☐ Dual ISPs (Image Signal Processors)

- 1.4 gigapixel/sec MIPI CSI camera ingest



## Ports & Peripherals

- HDMI 2.0
- 802.11a/b/g/n/ac 2x2 867Mbps Wi-Fi
- Bluetooth 4.1
- USB3, USB2
- 10/100/1000 BASE-T Ethernet
- 12 lanes MIPI CSI 2.0, 2.5 Gb/sec per lane
- PCIe gen 2.0, 1x4 + 1x1 or 2x1 + 1x2
- SATA, SD card
- dual CAN bus
- UART, SPI, I2C, I2S, GPIOs

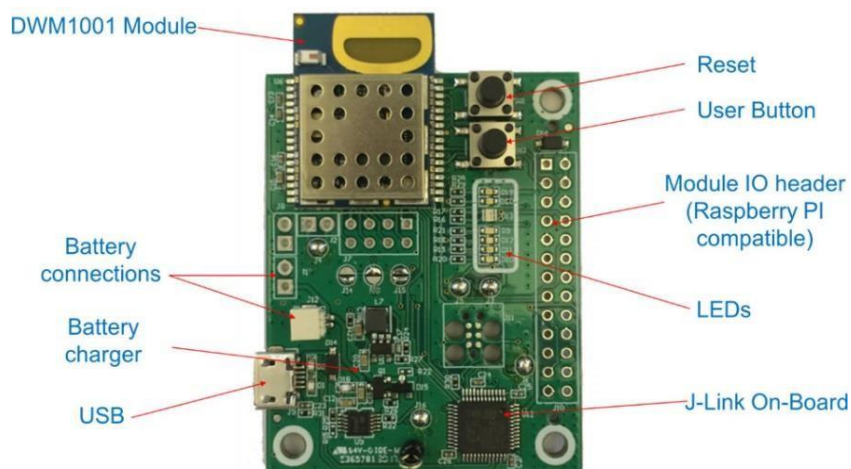
## Form-Factor

- 400-pin Samtec board-to-board connector
- dimensions: 50x87mm (1.96" x 3.42")
- Thermal Transfer Plate (TTP), -25C to 80C operating temperature
- mass: 85 grams, including TTP
- 5.5-19.6VDC input power (consuming 7.5W under typical load)

## DWM1001 development board:



The DWM1001 module is based on Decawave's DW1000 Ultra Wideband (UWB) transceiver IC, which is an IEEE 802.15.4- 2011 UWB implementation. It integrates UWB and Bluetooth® antenna, all RF circuitry, Nordic Semiconductor nRF52832 and a motion sensor.



### Key Features

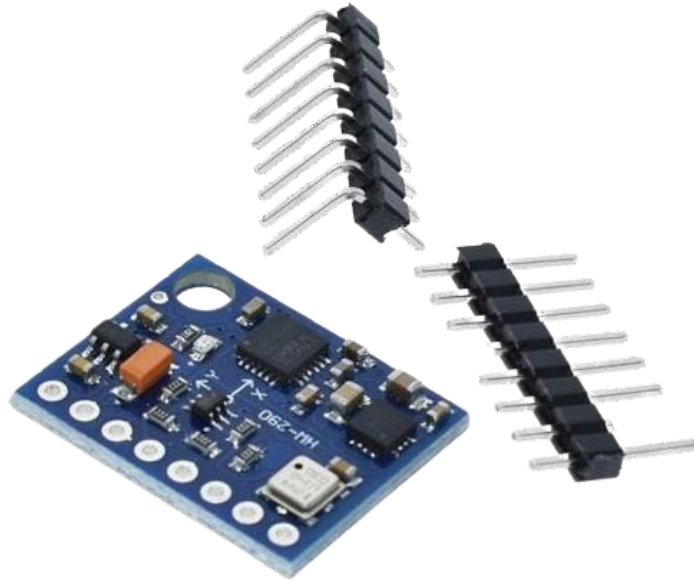
- Ranging accuracy to within 10cm.
- UWB Channel 5 printed PCB antenna (6.5 GHz)
- 6.8 Mbps data rate IEEE 802.15.4-2011 UWB Compliant
- Nordic Semiconductor nRF52832
- Bluetooth® connectivity
- Bluetooth® chip antenna
- Motion sensor: 3-axis accelerometer
- Supply voltage: 2.8 V to 3.6 V
- Size: 19.1 mm x 26.2 mm x 2.6 mm

Its main key benefit is it enables anchors, tags & gateways to quickly get an entire RTLS system up-and-running.



## IMU sensor:

IMU (Inertial Measurement Unit) sensors are used in self-balancing robots, UAVs, smartphones, and more. IMU sensors help us get the position of an object attached to the sensor in three-dimensional space. These values are usually in angles to help us to determine its position. They are used to detect the orientation of smartphones, or in wearable gadgets like the Fitbit, which uses IMU sensors to track movement.



### Specifications:

Type	GY-87
Driver Chip	MPU6050+HMC5883L+BMP180
Operating Voltage(VDC)	3 to 5
Gyro range(°/s)	$\pm 250, 500, 1000, 2000$
Acceleration range(g)	$\pm 2 \pm 4 \pm 8 \pm 16$
Communication	I2C Protocol
Length (mm)	22
Width (mm)	17
Weight (gm)	5
Shipment Weight	0.02 kg
Shipment Dimensions	3.5 x 3 x 1.5 cm

### Description of Driver Chips:

#### MPU6050:

The MPU6050 devices combine a 3-axis gyroscope and a 3-axis accelerometer on the same silicon together with an onboard Digital Motion Processor (DMP) capable of processing complex 9-axis Motion Fusion algorithms.

The parts' integrated 9-axis Motion Fusion algorithms access external magnetometers or other sensors through an auxiliary master I2C bus, allowing the device to gather a full set of sensor data without intervention from the system processor. For precision tracking of both fast and slow motion, the parts feature a user-programmable gyro full-scale range of  $\pm 250, \pm 500, \pm 1000$ , and  $\pm 2000^\circ/\text{sec}$  (DPS) and a user-programmable accelerometer full-scale range of  $\pm 2g, \pm 4g, \pm 8g$ , and  $\pm 16g$ .

### **HMC5883:**

The HMC5883 is a surface mount multi-chip module designed for low field magnetic sensing with a digital interface for applications such as low cost compassing and magnetometry. The HMC5883 includes our state of the art, high-resolution HMC118X series magneto-resistive sensors plus Honeywell developed ASIC containing amplification, automatic degaussing strap drivers, offset cancellation, 12-bit ADC that enables 1° to 2° compass heading accuracy.

### **BMP180:**

This precision sensor is the best low-cost sensing solution for measuring barometric pressure and temperature. Barometric pressure sensors measure the absolute pressure of the air around them. This pressure varies with both the weather and altitude. The BMP180 is the next-generation of sensors and replaces the BMP085. The sensor is soldered onto a PCB with a 3.3V regulator, I2C level shifter and pull-up resistors on the I2C pins.

### **USB to Serial converter:**

A microcontroller can communicate with a computer or another device by using a standard method of communication that has been worked out ahead of time. Serial communication is a very common means of communication. Devices using communication send and receive a series of digital pulses between them at a rate that's been decided upon before communication starts. That agreed-upon signaling is called a protocol. A USB to Serial Port adapter is a type of protocol converter which is used



for converting USB data signals to and from other communications standards. Commonly, USB adapters are used to convert USB data to standard serial port data and vice versa. Most commonly the USB data signals are converted to either RS232, RS485, RS422, or TTL-level UART serial data.

In our robot, we have used USB to TTL-level UART serial data converter. Asynchronous serial communication is sometimes referred to as Transistor-Transistor Logic (TTL) serial, where the high voltage level is logic 1, and the low voltage equates to logic 0. The rate at which a serial communication takes place is called a baud rate. Baud rates are measured in bits per second (bps) and some standard baud rates are 110, 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 5600, 57600, 115200, 128000, 153600, 230400, 256000, 460800, and 921600 bps. For simple communication, such as communicating with a microcontroller to program it from a host PC, the baud rate is usually 9600 bps or so. At 9600 baud, a transmitter sends one bit every 1/9600th of a second and the receiving device will receive at the same rate. Microcontrollers typically sense a high voltage level at +3.3 or +5.0 volts for serial communication. Thus, every 1/9600<sup>th</sup> of a second, a receiver will look at the line and determine if it's high or low-level voltage and translate that into a bit or 1 or 0. In TTL series communication, a low voltage level means a 0-bit value and a high voltage level means a 1-bit value.

### 3.3 Assembly of the robot:

#### Jetson TX2 Board:

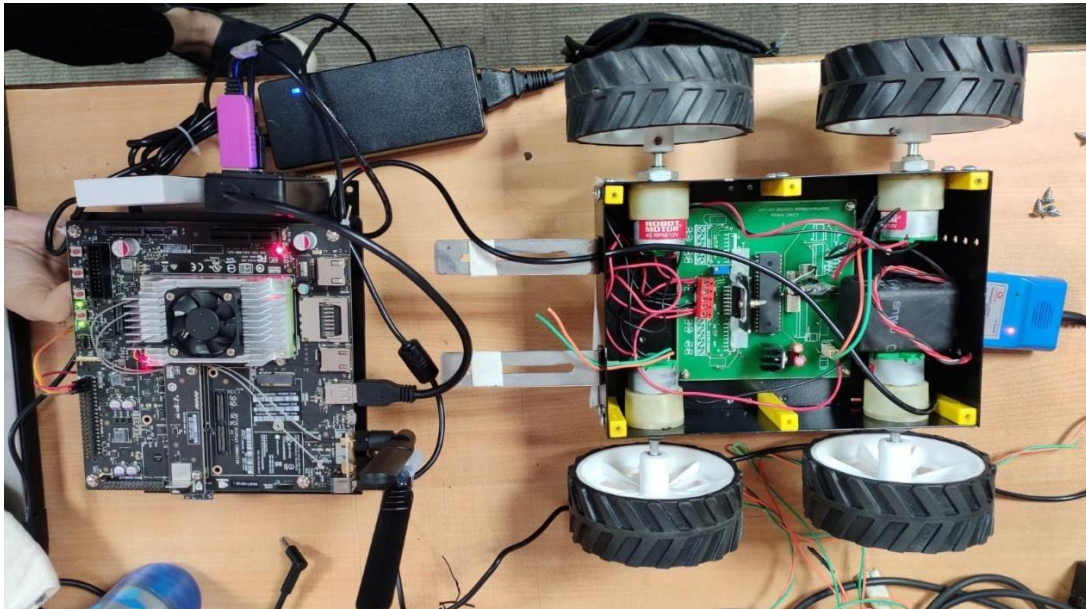
The computing board Jetson tx2 is fitted on top of the chassis. It is the brain of the Robot. It is given a power of 19V through a power adapter.

- The motor driver board which controls the motors is connected with the Jetson through serial port interface. A serial port is a serial communication interface through which information transfers in or out one bit at a time.
- The Decawave module is connected through USB.
- The HMC5883L and the IMU modules are connected through I2C bus and uses I2C Protocol to communicate with the Jetson Board.

#### Motor Driver Board:

The motor driver board is given a power supply of 12V through a rechargeable battery.

- The motors (each of 45rpm) are connected to the Microcontroller ATmega 16 through the motor driver L298N.
- The output pins of the motor driver is connected to the motor as below:
- The input pins of the motor driver are connected to the pins of Port-D of ATmega 16 as shown below:
- A crystal oscillator (crystal-11.0592, capacitor-22 pF(2)) is mounted on the motor control driver board to generate clock pulses required for the synchronization of all the internal operations.



### 3.4 Working Principle of the Motor Driver Board:

As mentioned earlier, the motor driver board is mainly consisting of two ICs- Atmega 16 Microcontroller IC and L298N IC. The Atmega16 IC has four ports viz. PortA, PortB, PortC & PortD. We have used PortD for controlling the motors. PD4, PD5, PD6 & PD7 are connected to the input pins of L298N. The output pins of the L298N IC are connected to the positive & negative terminals of the motor as shown in fig: A. The logic supply voltage,  $V_{ss}$  is given 5V and the supply voltage,  $V_s$  is given 12V. EnableA & EnableB is set at logic 1. Current sensing A & Current sensing B pins are short circuited and connected to the ground through a resistance of 0.47 Ohm. A diagrammatic representation of the connections is shown below.

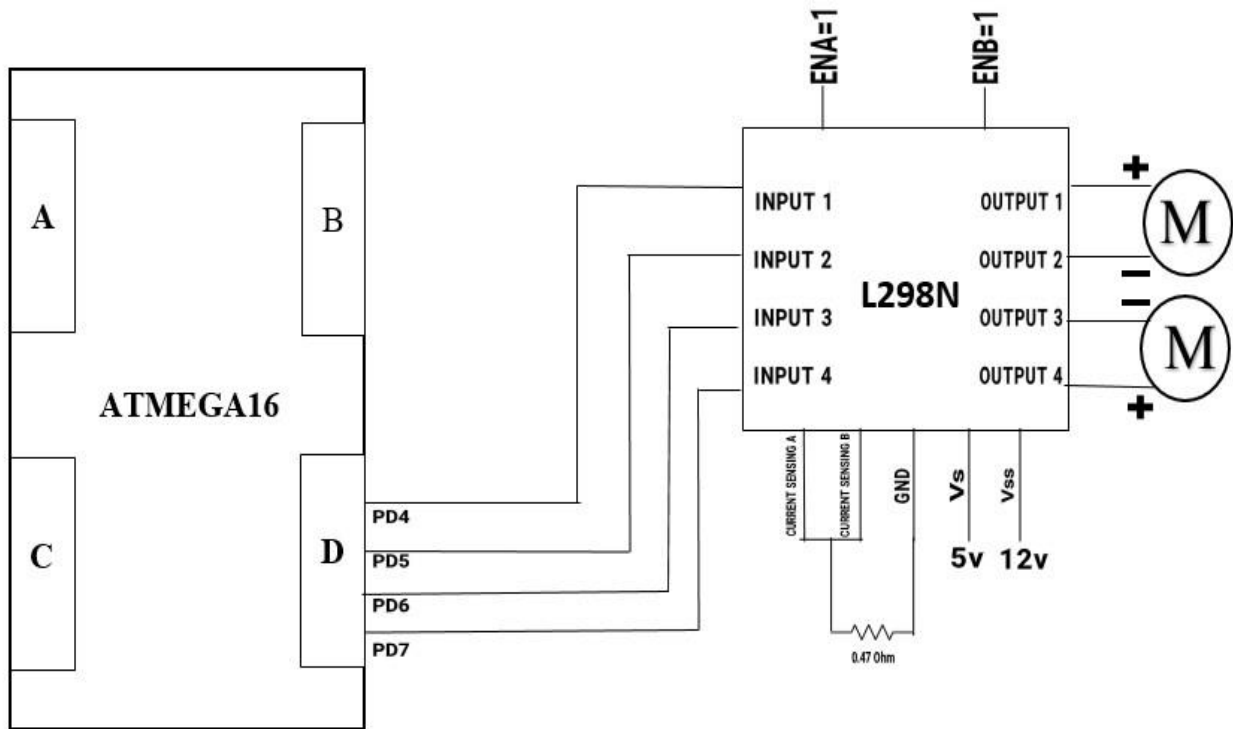


Fig: A

The entire process of sending a command to the robot <sup>0.47 Ohm</sup> from PC to the movement of the robot can be explained with the help of the table below.

PC Command	MCU Command	A	B		C	D		Movement
S	Xs	1	1		1	1		Stop
F	Xf	1	0	↑	0	1	↑	Front
B	xB	0	1	↓	1	0	↓	Back
R	xR	0	1	↑	0	1	↓	Right
L	Xl	1	0	↓	1	0	↑	Left

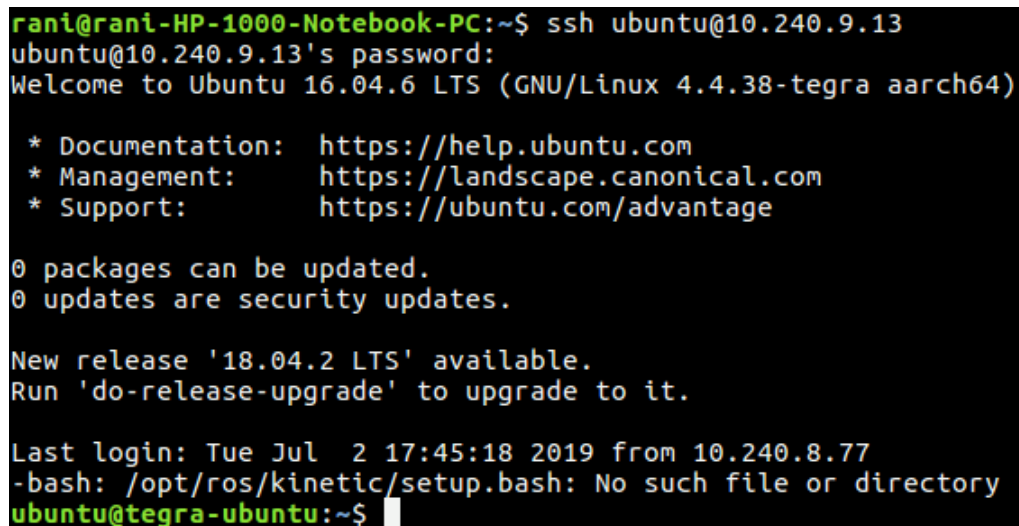
### 3.5 Working Principle of the Nvidia Jetson Tx2 Board:

Nvidia Jetson Tx2 is a power efficient embedded AI computing device. This supercomputer is connected to the Decawave DWM1001 module, HMC5883L module, IMU Sensor and the AVR motor driver board. A remote PC is connected to it via Wi-Fi. The supercomputer is accessed through SSH (refer to figure B).

The Jetson board communicates with the IMU and Magnetic sensor through I2C bus and the I2C protocol. It communicates with DWM1001 through USB port through ACM (Abstract Control Module).

Appropriate programs are written and executed to read from the DWM1001 module, HMC5883L module and the IMU sensor and to write into the AVR motor driver board as per the requirement.

On execution of the programs appropriate commands are send to the microcontroller board. Consequently, the microcontroller ATmega 16 sends signals to the motor driver which in turn rotates the motors.



```
rani@rani-HP-1000-Notebook-PC:~$ ssh ubuntu@10.240.9.13
ubuntu@10.240.9.13's password:
Welcome to Ubuntu 16.04.6 LTS (GNU/Linux 4.4.38-tegra aarch64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

0 packages can be updated.
0 updates are security updates.

New release '18.04.2 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Tue Jul  2 17:45:18 2019 from 10.240.8.77
-bash: /opt/ros/kinetic/setup.bash: No such file or directory
ubuntu@tegra-ubuntu:~$
```

Figure: B

### 3.6 Software Description:

#### A. Reading from Decawave Module

The data read by the tag DWM is stored in the Linux file /dev/ttyACM0. Following are the functions and system calls used to read the x- and y- coordinates of the tag:

1. **open()** : The **open()** system call opens the file specified by pathname in double inverted commas. If the specified file does not exist, it may optionally be created by **open()**. The return value of **open()** is a file descriptor, a small, nonnegative integer that is used in subsequent system calls. The file descriptor returned by a successful call will be the lowest-numbered file descriptor not currently open for the process.

#### Syntax:

```
int fd = open("/dev/ttyACM0", O_RDWR | O_NOCTTY | O_NDELAY);
```

2. **read( )** : read() attempts to read up to count bytes from file descriptor fd into the buffer starting at buf. On files that support seeking, the read operation commences at the file offset, and the file offset is incremented by the number of bytes read. If the file offset is at or past the end of file, no bytes are read, and read() returns zero.

**Syntax:**

function declaration

```
ssize_t read(int fd, void *buf, size_t count);
```

3. **write()** : write() writes up to count bytes from the buffer starting at buf to the file referred to by the file descriptor fd.

**Syntax:**

```
ssize_t write(int fd, const void *buf, size_t count);
```

4. **close()**: It closes a file descriptor, so that it no longer refers to any file and may be reused. **close()** returns zero on success. On error, -1 is returned, and *errno* is set appropriately.

**Syntax:**

```
close(fd);  
//fd is the file descriptor
```

Note: To use the above system calls the unistd.h header file has to be included.

## **B. Reading from HMC 5883L**

**ioctl()** : An ioctl (input-output control) is a kind of device-specific system call. There are only a few system calls in Linux (300-400), which are not enough to express all the unique functions devices may have. So a driver can define an ioctl which allows userspace application to send it orders.

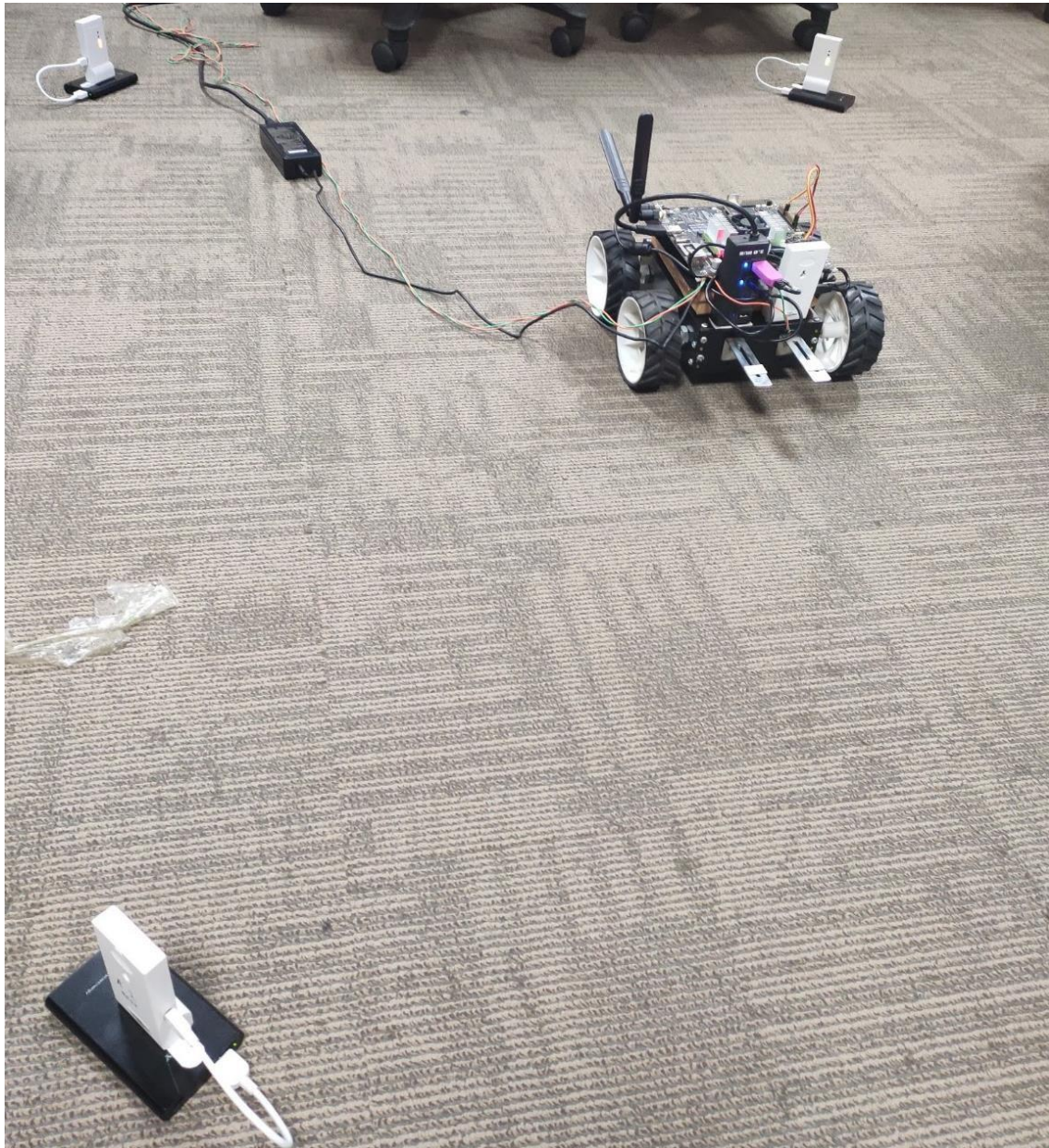
The ioctl() system call manipulates the underlying device parameters of special files. In particular, many operating characteristics of character special files (e.g., terminals) may be controlled with ioctl() requests. The argument fd must be an open file descriptor. The second argument is a device-dependent request code. The third argument is an untyped pointer to memory.

## **3.7 DecaWave Module (DWM1001) Interface**

The DWM1001 is a device that works on Ultra Wide Band (UWB). Ultra-Wide Band (**UWB**) is a communication method used in wireless networking that uses low power consumption to achieve high bandwidth connections. In other words, it's meant to transmit a lot of data over a short distance without using too much power.



We have used four DWM for the UWB communication. The tag is attached to the Jetson Board, while the three anchor are placed at three different corners of the room. This is to be noted that the minimum number of anchors required for the UWB communication is three, like GPS.



### **Algorithm for reading from DWM interface**

1. Open the file `/dev/ttyACM0`
2. On successful open, the file is read, a fixed byte at a time and stored in a character buffer.
3. The x- and y- coordinates are extracted from the char array and converted into float data.
4. The float data are ultimately used to localize the bot and for the further movement of the bot.

## 3.8 I2C Protocol

An Inter-Integrated Circuit protocol is one of the serial communication protocols that is used for the chip to chip communication. I2C is the synchronous communication where both master and slave use the shared clock which is produced by the master.

IMU sensor (MPU6050) or low-field magnetic sensor (HMC5883L) is used for obtaining precise heading information. Both of them have digital interfacing and is controlled by the Jetson through I2C communication.

### Feature of I2C Bus

- In I2C only two buses are required for the communication, the serial data bus (SDA) and serial clock bus (SCL).
- Each component in I2C bus is software addressable by a unique address, this unique address is used by the master to communicate with a particular slave.
- Always a master and slave relationships exist in I2C.
- In I2C, communication is started by the master.
- The I2C bus provides the ability of the arbitration and collision detection.
- I2C is the 8-bit oriented serial bidirectional communication, there are following speed mode in I2C

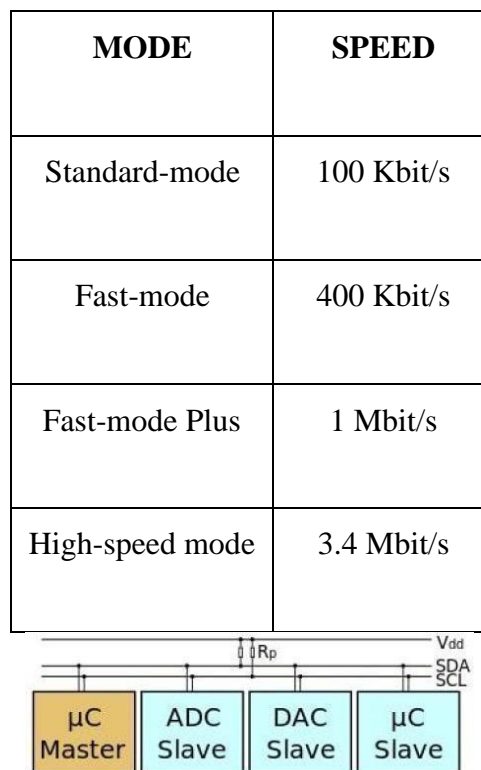


Fig1: I2C communication of single master and multiple slaves



## I2C WRITE OPERATION



## I2C READ OPERATION



Sent by the slave



Sent by the Master

## 3.9 IMU Interface:

An inertial measurement unit (IMU) consists of accelerometer (measures acceleration), gyroscope (measures angular velocity) and magnetometer (measures magnetic intensity). The maximum possible degrees of freedom is 6, which would include 3 degrees of translation (flat) movement across a straight plane/along each axis (front/back, right/left, up/down) and 3 degrees of rotational movement across the x, y and z axes/about each axis.

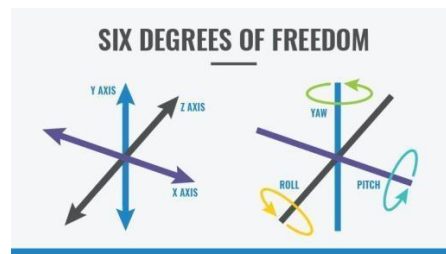


Fig. 2: DOF of IMU

## IMU6050

For precision tracking of both fast and slow motion, the MPU-60X0 features a user-programmable gyroscope full-scale range of  $\pm 250$ ,  $\pm 500$ ,  $\pm 1000$ , and  $\pm 2000^\circ/\text{sec}$  (dps). The parts also have a user programmable accelerometer full-scale range of  $\pm 2g$ ,  $\pm 4g$ ,  $\pm 8g$ , and  $\pm 16g$ .

## REGISTER MAP

Addr (Hex)	Addr (Dec.)	Register Name	Serial I/F	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0D	13	SELF_TEST_X	R/W	XA_TEST[4-2]			XG_TEST[4-0]				
0E	14	SELF_TEST_Y	R/W	YA_TEST[4-2]			YG_TEST[4-0]				
0F	15	SELF_TEST_Z	R/W	ZA_TEST[4-2]			ZG_TEST[4-0]				
10	16	SELF_TEST_A	R/W	RESERVED		XA_TEST[1-0]		YA_TEST[1-0]		ZA_TEST[1-0]	
19	25	SMPLRT_DIV	R/W	SMPLRT_DIV[7:0]							
1A	26	CONFIG	R/W	-	-	EXT_SYNC_SET[2:0]			DLPF_CFG[2:0]		
1B	27	GYRO_CONFIG	R/W	-	-	-	FS_SEL [1:0]		-	-	-
1C	28	ACCEL_CONFIG	R/W	XA_ST	YA_ST	ZA_ST	AFS_SEL[1:0]				
23	35	FIFO_EN	R/W	TEMP_FIFO_EN	XG_FIFO_EN	YG_FIFO_EN	ZG_FIFO_EN	ACCEL_FIFO_EN	SLV2_FIFO_EN	SLV1_FIFO_EN	SLV0_FIFO_EN
24	36	I2C_MST_CTRL	R/W	MULT_MST_EN	WAIT_FOR_ES	SLV_3_FIFO_EN	I2C_MST_P_NSR	I2C_MST_CLK[3:0]			
25	37	I2C_SLV0_ADDR	R/W	I2C_SLV0_RW	I2C_SLV0_ADDR[6:0]						
26	38	I2C_SLV0_REG	R/W	I2C_SLV0_REG[7:0]							
27	39	I2C_SLV0_CTRL	R/W	I2C_SLV0_EN	I2C_SLV0_BYTE_SW	I2C_SLV0_REG_DIS	I2C_SLV0_GRP	I2C_SLV0_LEN[3:0]			
28	40	I2C_SLV1_ADDR	R/W	I2C_SLV1_RW	I2C_SLV1_ADDR[6:0]						
29	41	I2C_SLV1_REG	R/W	I2C_SLV1_REG[7:0]							
2A	42	I2C_SLV1_CTRL	R/W	I2C_SLV1_EN	I2C_SLV1_BYTE_SW	I2C_SLV1_REG_DIS	I2C_SLV1_GRP	I2C_SLV1_LEN[3:0]			
2B	43	I2C_SLV2_ADDR	R/W	I2C_SLV2_RW	I2C_SLV2_ADDR[6:0]						
2C	44	I2C_SLV2_REG	R/W	I2C_SLV2_REG[7:0]							
2D	45	I2C_SLV2_CTRL	R/W	I2C_SLV2_EN	I2C_SLV2_BYTE_SW	I2C_SLV2_REG_DIS	I2C_SLV2_GRP	I2C_SLV2_LEN[3:0]			
2E	46	I2C_SLV3_ADDR	R/W	I2C_SLV3_RW	I2C_SLV3_ADDR[6:0]						
2F	47	I2C_SLV3_REG	R/W	I2C_SLV3_REG[7:0]							
30	48	I2C_SLV3_CTRL	R/W	I2C_SLV3_EN	I2C_SLV3_BYTE_SW	I2C_SLV3_REG_DIS	I2C_SLV3_GRP	I2C_SLV3_LEN[3:0]			
31	49	I2C_SLV4_ADDR	R/W	I2C_SLV4_RW	I2C_SLV4_ADDR[6:0]						
32	50	I2C_SLV4_REG	R/W	I2C_SLV4_REG[7:0]							
33	51	I2C_SLV4_DO	R/W	I2C_SLV4_DO[7:0]							
34	52	I2C_SLV4_CTRL	R/W	I2C_SLV4_EN	I2C_SLV4_INT_EN	I2C_SLV4_REG_DIS	I2C_MST_DLY[4:0]				
35	53	I2C_SLV4_DI	R	I2C_SLV4_DI[7:0]							
36	54	I2C_MST_STATUS	R	PASS_THROUGH	I2C_SLV4_DONE	I2C_LOST_ARB	I2C_SLV4_NACK	I2C_SLV3_NACK	I2C_SLV2_NACK	I2C_SLV1_NACK	I2C_SLV0_NACK
37	55	INT_PIN_CFG	R/W	INT_LEVEL	INT_OPEN	LATCH_INT_EN	INT_RD_CLEAR	FSYNC_INT_LEVEL	FSYNC_INT_EN	I2C_BYPASS_EN	-
38	56	INT_ENABLE	R/W	-	-	-	FIFO_OFLOW_EN	I2C_MST_INT_EN	-	-	DATA_RDY_EN
3A	58	INT_STATUS	R	-	-	-	FIFO_OFLOW_INT	I2C_MST_INT	-	-	DATA_RDY_INT
3B	59	ACCEL_XOUT_H	R	ACCEL_XOUT[15:8]							
3C	60	ACCEL_XOUT_L	R	ACCEL_XOUT[7:0]							
3D	61	ACCEL_YOUT_H	R	ACCEL_YOUT[15:8]							
3E	62	ACCEL_YOUT_L	R	ACCEL_YOUT[7:0]							
3F	63	ACCEL_ZOUT_H	R	ACCEL_ZOUT[15:8]							
40	64	ACCEL_ZOUT_L	R	ACCEL_ZOUT[7:0]							
41	65	TEMP_OUT_H	R	TEMP_OUT[15:8]							
42	66	TEMP_OUT_L	R	TEMP_OUT[7:0]							
43	67	GYRO_XOUT_H	R	GYRO_XOUT[15:8]							
44	68	GYRO_XOUT_L	R	GYRO_XOUT[7:0]							
45	69	GYRO_YOUT_H	R	GYRO_YOUT[15:8]							
46	70	GYRO_YOUT_L	R	GYRO_YOUT[7:0]							
47	71	GYRO_ZOUT_H	R	GYRO_ZOUT[15:8]							

48	72	GYRO_ZOUT_L	R	GYRO_ZOUT[7:0]							
49	73	EXT_SENS_DATA_00	R	EXT_SENS_DATA_00[7:0]							
4A	74	EXT_SENS_DATA_01	R	EXT_SENS_DATA_01[7:0]							
4B	75	EXT_SENS_DATA_02	R	EXT_SENS_DATA_02[7:0]							
4C	76	EXT_SENS_DATA_03	R	EXT_SENS_DATA_03[7:0]							
4D	77	EXT_SENS_DATA_04	R	EXT_SENS_DATA_04[7:0]							
4E	78	EXT_SENS_DATA_05	R	EXT_SENS_DATA_05[7:0]							
4F	79	EXT_SENS_DATA_06	R	EXT_SENS_DATA_06[7:0]							
50	80	EXT_SENS_DATA_07	R	EXT_SENS_DATA_07[7:0]							
51	81	EXT_SENS_DATA_08	R	EXT_SENS_DATA_08[7:0]							
52	82	EXT_SENS_DATA_09	R	EXT_SENS_DATA_09[7:0]							
53	83	EXT_SENS_DATA_10	R	EXT_SENS_DATA_10[7:0]							
54	84	EXT_SENS_DATA_11	R	EXT_SENS_DATA_11[7:0]							
55	85	EXT_SENS_DATA_12	R	EXT_SENS_DATA_12[7:0]							
56	86	EXT_SENS_DATA_13	R	EXT_SENS_DATA_13[7:0]							
57	87	EXT_SENS_DATA_14	R	EXT_SENS_DATA_14[7:0]							
58	88	EXT_SENS_DATA_15	R	EXT_SENS_DATA_15[7:0]							
59	89	EXT_SENS_DATA_16	R	EXT_SENS_DATA_16[7:0]							
5A	90	EXT_SENS_DATA_17	R	EXT_SENS_DATA_17[7:0]							
5B	91	EXT_SENS_DATA_18	R	EXT_SENS_DATA_18[7:0]							
5C	92	EXT_SENS_DATA_19	R	EXT_SENS_DATA_19[7:0]							
5D	93	EXT_SENS_DATA_20	R	EXT_SENS_DATA_20[7:0]							
5E	94	EXT_SENS_DATA_21	R	EXT_SENS_DATA_21[7:0]							
5F	95	EXT_SENS_DATA_22	R	EXT_SENS_DATA_22[7:0]							
60	96	EXT_SENS_DATA_23	R	EXT_SENS_DATA_23[7:0]							
63	99	I2C_SLV0_DO	R/W	I2C_SLV0_DO[7:0]							
64	100	I2C_SLV1_DO	R/W	I2C_SLV1_DO[7:0]							
65	101	I2C_SLV2_DO	R/W	I2C_SLV2_DO[7:0]							
66	102	I2C_SLV3_DO	R/W	I2C_SLV3_DO[7:0]							
67	103	I2C_MST_DELAY_CTRL	R/W	DELAY_ES_SHADOW	.	.	I2C_SLV4_DLY_EN	I2C_SLV3_DLY_EN	I2C_SLV2_DLY_EN	I2C_SLV1_DLY_EN	I2C_SLV0_DLY_EN
68	104	SIGNAL_PATH_RESET	R/W	.	.	.	.	.	GYRO_RESET	ACCEL_RESET	TEMP_RESET
6A	106	USER_CTRL	R/W	.	FIFO_EN	I2C_MST_EN	I2C_IF_DIS	.	FIFO_RESET	I2C_MST_RESET	SIG_COND_RESET
6B	107	PWR_MGMT_1	R/W	DEVICE_RESET	SLEEP	CYCLE	.	TEMP_DIS	CLKSEL[2:0]		
6C	108	PWR_MGMT_2	R/W	LP_WAKE_CTRL[1:0]		STBY_XA	STBY_YA	STBY_ZA	STBY_XG	STBY_YG	STBY_ZG
72	114	FIFO_COUNTH	R/W	FIFO_COUNT[15:8]							
73	115	FIFO_COUNTL	R/W	FIFO_COUNT[7:0]							
74	116	FIFO_R_W	R/W	FIFO_DATA[7:0]							
75	117	WHO_AM_I	R	.	WHO_AM_I[6:1]						.

The reset value is 0x00 for all registers other than the registers below.

- Register 107: 0x40.
- Register 117: 0x68.

## **ALGORITHM**

- Calling of a mpu\_initialization function to set the device address (0x68).
- Accessing the power management register address and setting the bits accordingly. Set register 0x6B to 0x00.

Register 107 – Power Management 1

**Type: Read/Write**

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
6B	107	DEVICE_RESET	SLEEP	CYCLE	-	TEMP_DIS	CLKSEL[2:0]		

An internal 8MHz oscillator, gyroscope based clock, or external sources can be selected as the MPU-60X0 clock source.

**DEVICE\_RESET** When set to 1, this bit resets all internal registers to their default values. The bit automatically clears to 0 once the reset is done.

**SLEEP** When set to 1, this bit puts the MPU-60X0 into sleep mode.

**CYCLE** When this bit is set to 1 and SLEEP is disabled, the MPU-60X0 will cycle between sleep mode and waking up to take a single sample of data from active sensors at a rate determined by LP\_WAKE\_CTRL (register 108).

**TEMP\_DIS** When set to 1, this bit disables the temperature sensor.

**CLKSEL** 3-bit unsigned value. Specifies the clock source of the device.

- Accessing the 0x1B and 0x1C and setting both the register value as 0x00 so that the gyroscope LSB sensitivity is 131 and accelerometer LSB sensitivity is 16384.
- Calling i2c\_read function to read the accelerometer values for x, y, z axis which returns the values in a character array.

**Type: Read Only**

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
3B	59	ACCEL_XOUT[15:8]							
3C	60	ACCEL_XOUT[7:0]							
3D	61	ACCEL_YOUT[15:8]							
3E	62	ACCEL_YOUT[7:0]							
3F	63	ACCEL_ZOUT[15:8]							
40	64	ACCEL_ZOUT[7:0]							

- Typecasting the character into int and shifting left the values in 3B, 3D, 3F by 8 bits and then OR-ing them bitwise with values in 3C, 3E, 40 to obtain 16 bits AccX, AccY, AccZ.

### Calculating roll and pitch values from accelerometer data:

$\text{accAngleX} = (\text{atan}(\text{AccY} / \sqrt{\text{pow}(\text{AccX}, 2) + \text{pow}(\text{AccZ}, 2)}) * 180 / \text{PI}) - 0.58;$   
 $\text{accAngleY} = (\text{atan}(-1 * \text{AccX} / \sqrt{\text{pow}(\text{AccY}, 2) + \text{pow}(\text{AccZ}, 2)}) * 180 / \text{PI}) + 1.58;$

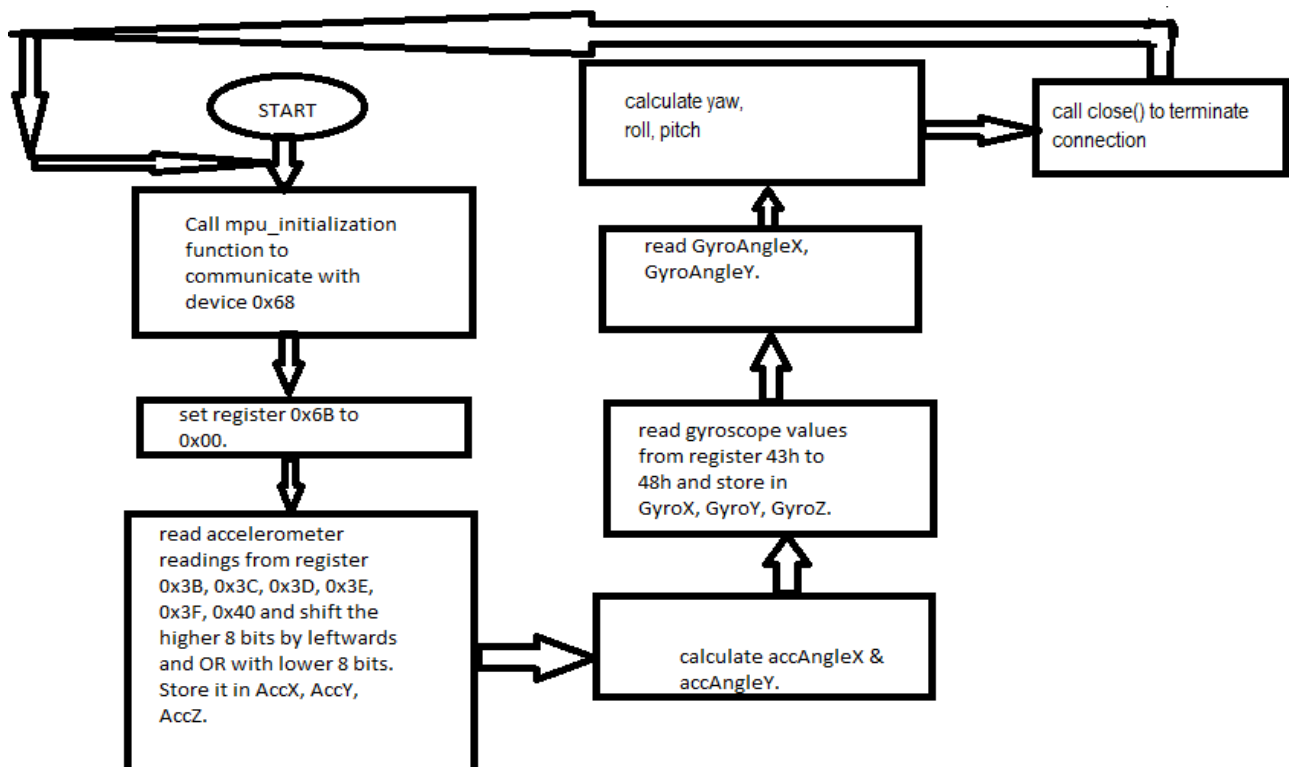
- Calling a timer function which returns the elapsed\_time.
- Calling i2c\_read function to read the gyroscope values for x, y, z axis which returns the values in a character array.

### Type: Read Only

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
43	67								GYRO_XOUT[15:8]
44	68								GYRO_XOUT[7:0]
45	69								GYRO_YOUT[15:8]
46	70								GYRO_YOUT[7:0]
47	71								GYRO_ZOUT[15:8]
48	72								GYRO_ZOUT[7:0]

- Typecasting the character into int and shifting left the values in 43, 45, 47 by 8 bits and then OR-ing them bitwise with values in 44, 46, 48 to obtain 16 bits GyroX, GyroY, GyroZ.  
 $\text{gyroAngleX} = \text{gyroAngleX} + \text{GyroX} * \text{elapsedTime}$   
 $\text{gyroAngleY} = \text{gyroAngleY} + \text{GyroY} * \text{elapsedTime}$
- $\text{yaw} = \text{yaw} + \text{GyroZ} * \text{elapsedTime}$
- $\text{roll} = 0.96 * \text{gyroAngleX} + 0.04 * \text{accAngleX}$
- $\text{pitch} = 0.96 * \text{gyroAngleY} + 0.04 * \text{accAngleY}$
- ending the transmission by calling the close function

### FLOWCHART



## 3.10 GY-271 HMC5883L Digital Electronic Compass 3 Axis Magnetic Field Sensor:

The Honeywell HMC5883L is a surface-mount, multi-chip module designed for low-field magnetic sensing with a digital interface for applications such as low cost compassing and magnetometry.

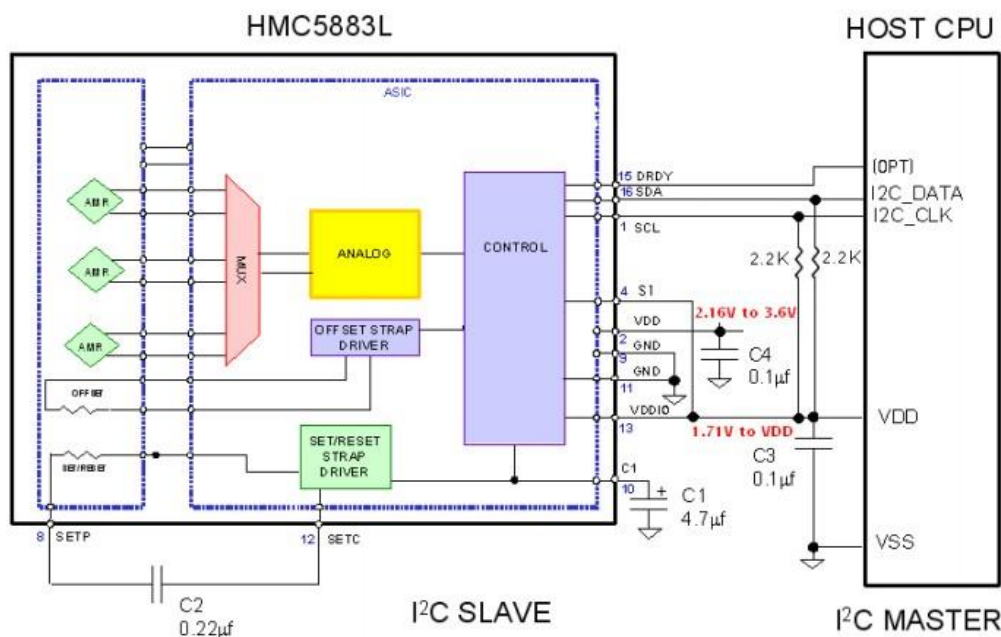
### FEATURES

- ▶ 3-Axis Magnetoresistive Sensors and ASIC in a 3.0x3.0x0.9mm LCC Surface Mount Package
- ▶ 12-Bit ADC Coupled with Low Noise AMR Sensors Achieves 2 milli-gauss Field Resolution in  $\pm 8$  Gauss Fields
- ▶ Built-In Self Test
- ▶ Low Voltage Operations (2.16 to 3.6V) and Low Power Consumption (100  $\mu$ A)
- ▶ Built-In Strap Drive Circuits
- ▶ I<sup>2</sup>C Digital Interface
- ▶ Lead Free Package Construction
- ▶ Wide Magnetic Field Range ( $\pm 8$  Oe)
- ▶ Software and Algorithm Support Available
- ▶ Fast 160 Hz Maximum Output Rate

### BENEFITS

- ▶ Small Size for Highly Integrated Products. Just Add a Micro-Controller Interface, Plus Two External SMT Capacitors Designed for High Volume, Cost Sensitive OEM Designs Easy to Assemble & Compatible with High Speed SMT Assembly
- ▶ Enables 1° to 2° Degree Compass Heading Accuracy
- ▶ Enables Low-Cost Functionality Test after Assembly in Production
- ▶ Compatible for Battery Powered Applications
- ▶ Set/Reset and Offset Strap Drivers for Degaussing, Self Test, and Offset Compensation
- ▶ Popular Two-Wire Serial Data Interface for Consumer Electronics
- ▶ RoHS Compliance
- ▶ Sensors Can Be Used in Strong Magnetic Field Environments with a 1° to 2° Degree Compass Heading Accuracy
- ▶ Compassing Heading, Hard Iron, Soft Iron, and Auto Calibration Libraries Available
- ▶ Enables Pedestrian Navigation and LBS Applications

### INTERNAL SCHEMATIC DIAGRAM HMC5883L





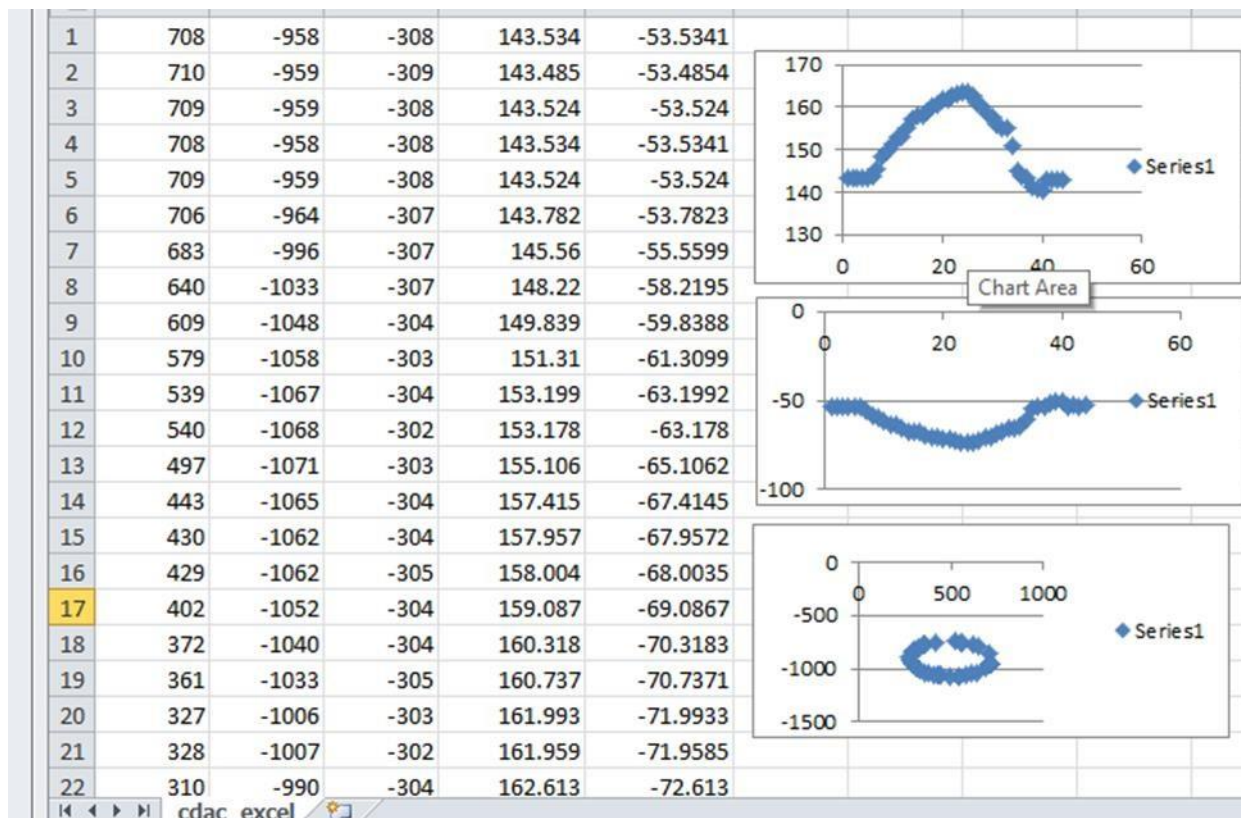
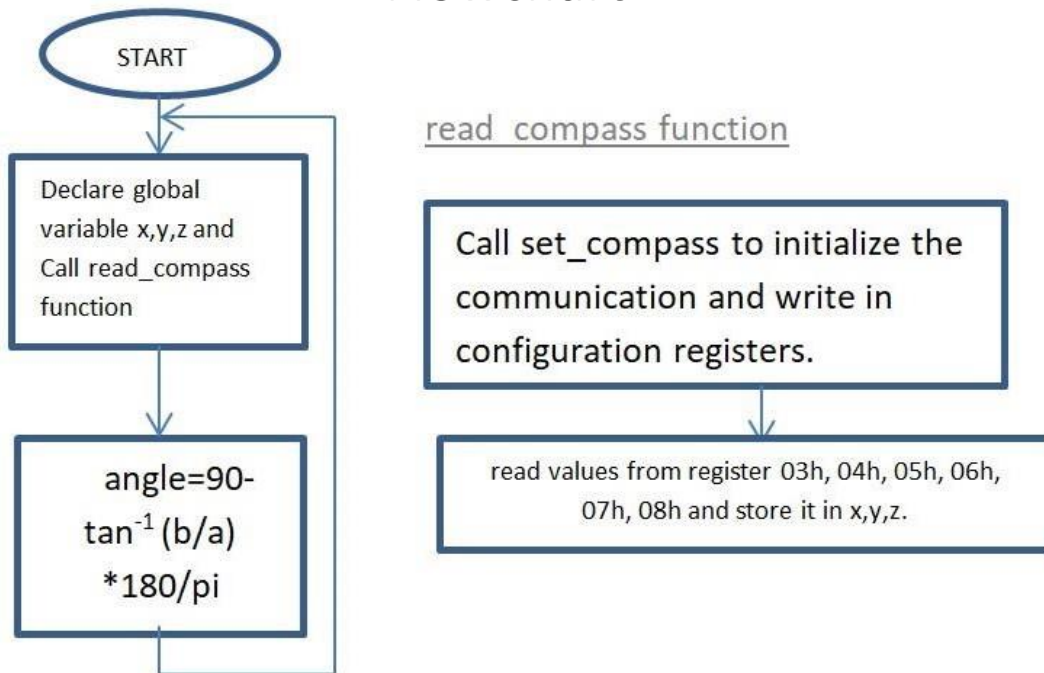
## **REGISTER MAP**

Address Location	Name	Access
00	Configuration Register A	Read/Write
01	Configuration Register B	Read/Write
02	Mode Register	Read/Write
03	Data Output X MSB Register	Read
04	Data Output X LSB Register	Read
05	Data Output Z MSB Register	Read
06	Data Output Z LSB Register	Read
07	Data Output Y MSB Register	Read
08	Data Output Y LSB Register	Read
09	Status Register	Read
10	Identification Register A	Read
11	Identification Register B	Read
12	Identification Register C	Read

## **ALGORITHM:**

- Calling the read\_compass function to read the values along x, y, z axis and storing the respective values in a, b, c
  1. Calling the set\_compass function :  
Calling the i2c\_init function to access the device (0x1E).  
Writing 0x70 in register 0x00.  
Writing 0x20 in register 0x01.  
Writing 0x00 in register 0x02.
  2. Reading the 16 bits values a, b, c from registers  
0x03,0x04,0x07,0x08,0x05,0x06.  
Typecasting the float into short signed and shifting left the values in  
0x03,0x07,0x05 by 8 bits and then OR-ing them bitwise with values in  
0x04,0x08,0x06. To obtain 16 bits a, b, c.
- Calculating the angle  
Formula:  $90 - \text{atan2}(b, a) * 180/M\_PI$

# Flowchart



In the above figure, the first column corresponds to values of x-axis, second column corresponds to y-axis, third column corresponds to z-axis, fourth and fifth column correspond to angles. The first graph is the plot for fourth column, second graph is the plot for fifth column and third graph is plot of y-axis values vs. x-axis values.



## Compass Calibration

From the above result, it can be interpreted that without compass calibration, the difference between the maximum angle and minimum angle for one single rotation is very small and it varies. To make it vary from 0 to 360-degree, calibration is done.

### **Working:**

In the compass C file a function `calibrate_compass()` is defined. From the main function

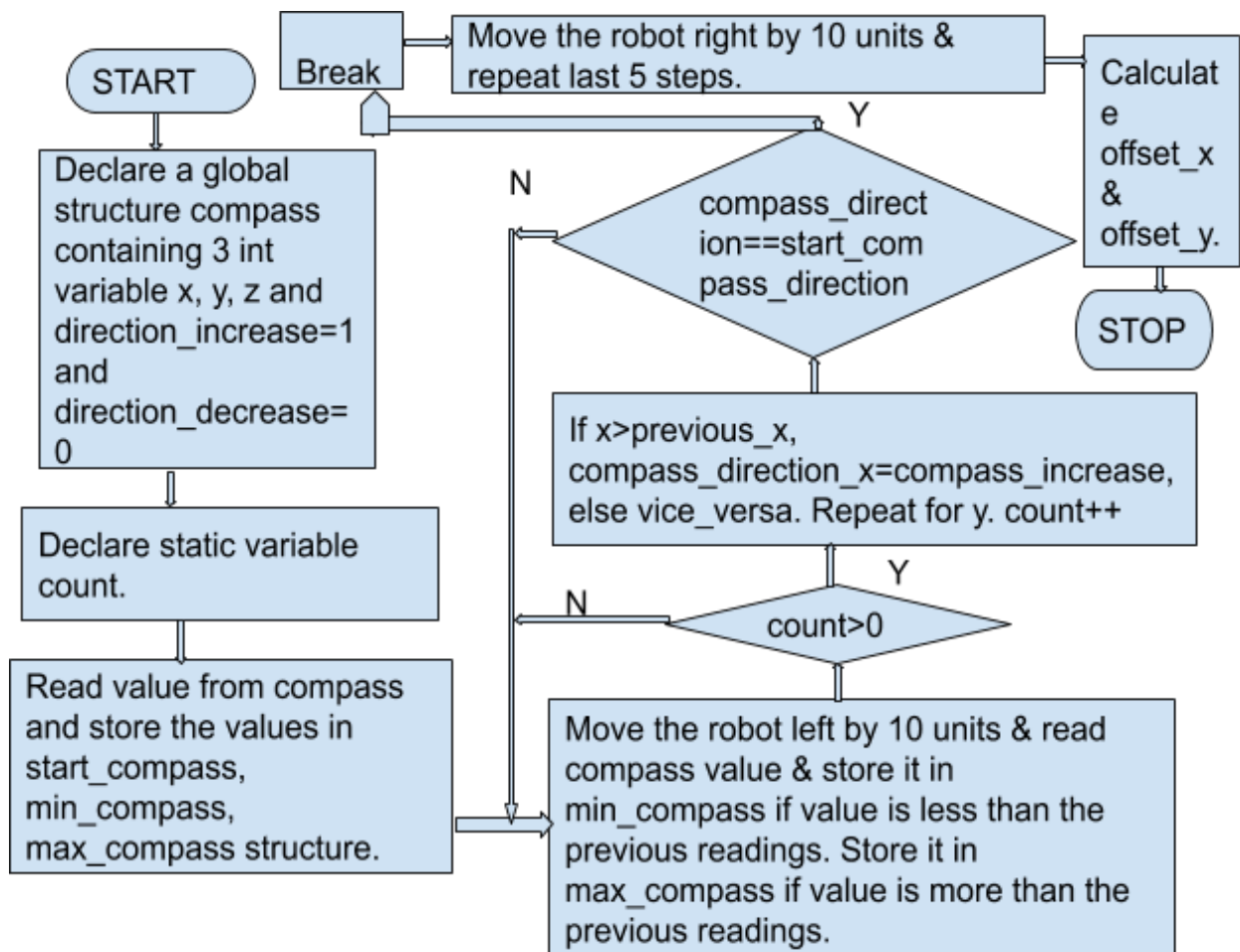
(`main.cpp`) the `calibrate_compass()` is called and the address of `offset_x`, `offset_y`, `offset_z` variable is passed. In `calibrate_compass()`, robot is rotated clockwise as well as anti-clockwise & the (`max_compass.x`, `max_compass.y`) as well as (`min_compass.x`, `min_compass.y`) is obtained.

$\text{offset\_x} = (\text{min\_x} + \text{max\_x}) / 2$

$\text{offset\_y} = (\text{min\_y} + \text{max\_y}) / 2$

In the main function the `offset_x` and `offset_y` variable is added with the actual (x,y) values read from the compass ( `x=x+offset_x`, `y=y+offset_y` ). These x and y values are then used to find out the angle.

### Flowchart



## 3.11 File Handling in C

File handling in C enables us to create, update, read, and delete the files stored on the local file system through our C program.

### Advantage:

When a variable is declared in a program, it occupies a memory in the process block. However, its life span is only till the program is being executed. Through file handling, the variable can be stored in a file in the secondary memory and later on can be retrieved.

### Working:

All the file handling functions in C are included in `<stdio.h>`.

Declare a file pointer which will point to a structure "FILE": `FILE *fp`

```
fp=fopen("file_name", "operating mode");
```

**fopen():** brings file (known as buffer in RAM) into process block in RAM. A char pointer in FILE points to the buffer. The address of the FILE structure is returned to fp.

**fputc(buf,fp):** writes a single character to the file.

**fgetc(buf,fp):** reads from file and stores the single character in char variable buf.

**fputs(buf,fp):** writes the string buf into the file.

**fputc(buf,sizeof(buf),fp):** reads string from file and stores it in buf.

**fwrite(const void \*ptr, size of each element in bytes, no. of elements to be written, fp):** ptr is the pointer to the array of elements which is to be written.

**fread(const void \*ptr, size of each element in bytes, no. of elements to be read, fp):** ptr is the pointer to the array of elements which is to be read.

**fprintf(fp, const char \*format [, argument, ...]):** same as printf() but the output is not displayed and is written in the file.

**fscanf(FILE \*stream, const char \*format [, argument, ...]):** same as scanf() but the input is taken from the file and is not the user input.

### ▪ Operating Modes:

"r"

Reads the file. If the file does not exist fopen() returns NULL.

"w"

Overwrites the content of the file. On calling fopen(), if file does not exist, a new file is created. The file pointer points at the first character in the beginning.

“a”

Appends the content in the existing content of the file. On calling fopen(),

if file does not exist, a new file is created. The file pointer points at the end of file in the beginning.

“r+”

Although main aim is to read from file, both reading and writing can be done. If the file does not exist fopen() returns NULL.

“w+”

Same as “w”. Although main aim is to perform write operation, both reading and writing can be done.

“a+”

Same as “a”. Although main aim is to perform append operation, both reading and appending can be done.

### **Use of file handling in the project**

Using file handling, one can easily store the value of the file descriptor(fd) in a file and can fetch it whenever required. It increases the robot’s efficiency as whenever one needs to read the values from different sensors (like IMU, compass, DecaWave), one doesn’t need to run the initialization process to get the fd value. Further, whenever the value is being read from a sensor and the ongoing process is terminated before closing the file descriptor, it can show an error while accessing the device file next time. So, before the device file is accessed the next time, the fd is read from a text file and is closed.

The functions fprintf() is used to perform write operation and fscanf() is used to read from the file.

## **3.12 Socket Programming using TCP/IP for communication between remote PC and Bot:**

Socket programs are used to communicate between various processes usually running on different systems. One socket (node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection. Server forms the listener socket while client reaches out to the server.

The communication over the network in TCP/IP model takes place in form of a client-server architecture. i.e., the client begins the communication and establish a connection with a server.

## **Steps to create a client using TCP/IP API**

- Create a socket using the `socket()` function in c.
- Initialize the socket address structure as per the server and connect the socket to the address of the server using the `connect()`;
- Receive and send the data using the `recv()` and `send()` functions.  
Close the connection by calling the `close()` function.

## **Steps to create a server using TCP/IP API**

- Create a socket using the `socket()` function in c.
- Initialize the socket address structure and bind the socket to an address using the `bind()` function.
- Listen for connections with the `listen()` function.
- Accept a connection with the `accept()` function system call. This call typically blocks until a client connects to the server.
- Receive and send data by using the `recv()` and `send()` function in c.
- Close the connection by using the `close()` function.

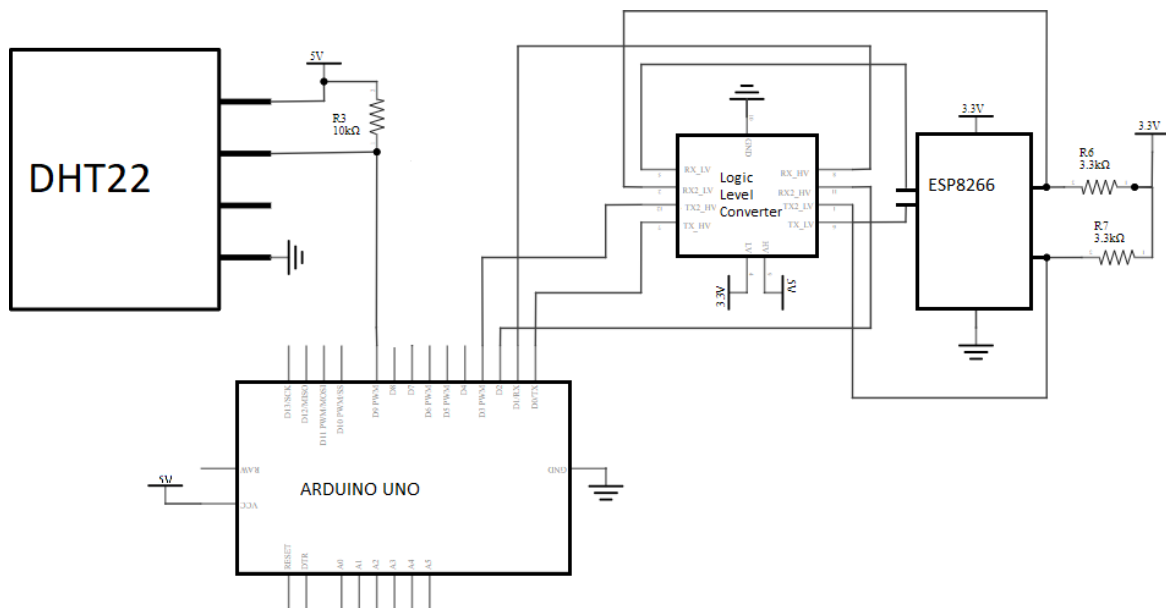
## **CHAPTER 4**

# **IoT BASED TEMPERATURE AND HUMIDITY SENSING MODULE**

## 4.1 Introduction:

This project is about a module that senses the temperature and humidity using a DHT22 temperature and humidity sensor which is connected to an Arduino UNO, which sends the data directly to an android application using an ESP8266 Wi-Fi module which is also connected to the Arduino Uno board. This module also anticipated to be enhanced with a PWM controlled LED and Fan.

## HARDWARE DESIGN AND CONFIGURATION:



## 4.2 Arduino Uno:

The Arduino UNO is an open-source microcontroller board based on the Microchip ATmega328P microcontroller and developed by Arduino.cc. The board is equipped with sets of digital and analog input/output (I/O) pins that may be interfaced to various expansion boards (shields) and other circuits. The board has 14 Digital pins, 6 Analog pins, and programmable with the Arduino IDE (Integrated Development Environment) via a type B USB cable. It can be powered by the USB cable or by an external 9-volt battery, though it accepts voltages between 7 and 20 volts.

### Specifications:

#### **Microcontroller ATmega328:**

Operating Voltage = 5V

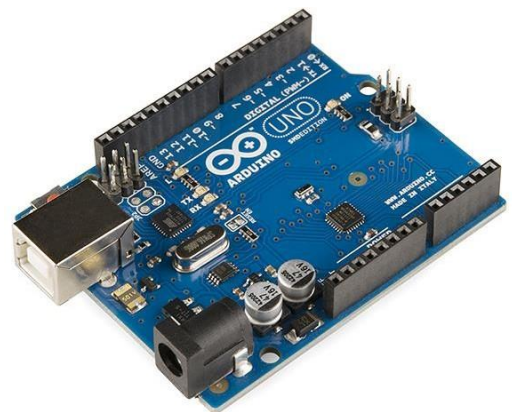
Input Voltage (recommended) = 7-12V

Input Voltage (limits) = 6-20V

Digital I/O Pins = 14 (of which 6 provide PWM output)

Analog Input Pins = 6

DC Current per I/O Pin = 40 mA



DC Current for 3.3V Pin = 50 mA

Flash Memory = 32 KB (ATmega328) of which 0.5 KB used by bootloader

SRAM = 2 KB (ATmega328)

EEPROM = 1 KB (ATmega328)

Clock Speed = 16 MHz

## 4.3 DHT22:

It utilizes exclusive digital-signal-collecting-technique and humidity sensing technology, assuring its reliability and stability. Its sensing elements is connected with 8-bit single-chip computer. Every sensor of this model is temperature compensated and calibrated in accurate calibration chamber and the calibration-coefficient is saved in type of program in OTP memory, when the sensor is detecting, it will cite coefficient from memory. Small size & low consumption & long transmission distance (20m) enable DHT22 to be suited in all kinds of harsh application occasions. Single-row packaged with four pins, making the connection very convenient.

### Specifications:

#### Model DHT22:

Power supply=3.3-6V DC

Output signal= digital signal via single-bus

Sensing element= Polymer capacitor

Operating range= humidity 0-100%RH; temperature -40~80Celsius

Accuracy= humidity  $\pm 2\%$  RH(Max  $\pm 5\%$  RH); temperature  $< \pm 0.5$  Celsius

Resolution or sensitivity= humidity 0.1%RH; temperature 0.1Celsius

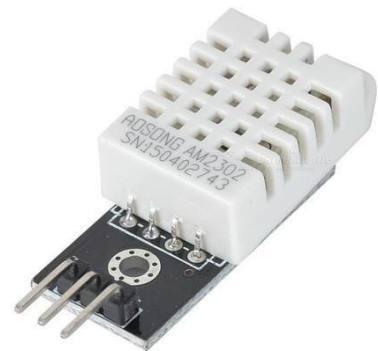
Repeatability= humidity  $\pm 1\%$  RH; temperature  $\pm 0.2$  Celsius

Humidity hysteresis=  $\pm 0.3\%$  RH

Long-term Stability=  $\pm 0.5\%$  RH/year

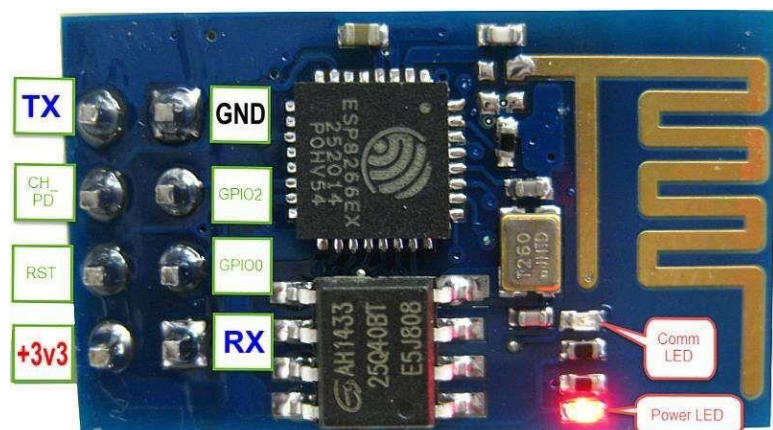
Sensing period= Average: 2s

Interchangeability= fully interchangeable



## 4.4 ESP8266:

ESP8266 delivers highly integrated Wi-Fi SoC solution to meet users' continuous demands for efficient power usage, compact design and reliable performance in the Internet of Things industry. With the complete and self-contained Wi-Fi networking capabilities, ESP8266 can perform either as a standalone application or as the slave to a host MCU. When ESP8266 hosts the



application, it promptly boots up from the flash. The integrated high speed cache helps to increase the system performance and optimize the system memory. Also, ESP8266 can be applied to any microcontroller design as a Wi-Fi adaptor through SPI/SDIO or UART interfaces. ESP8266 integrates antenna switches, RF balun, power amplifier, low noise receive amplifier, filters and power management modules. The compact design minimizes the PCB size and requires minimal external circuitries. Besides the Wi-Fi functionalities, ESP8266 also integrates an enhanced version of Tensilica's L106 Diamond series 32-bit processor and on-chip SRAM. It can be interfaced with external sensors and other devices through the GPIOs. Software Development Kit (SDK) provides sample codes for various applications. Espressif Systems' Smart Connectivity Platform (ESCP) enables sophisticated features including:

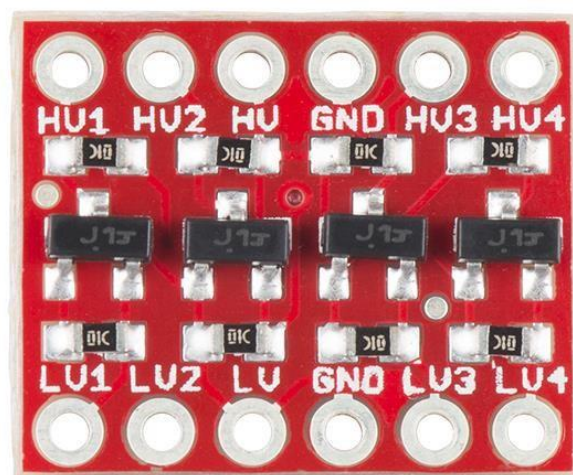
- Fast switch between sleep and wakeup mode for energy-efficient purpose;
- Adaptive radio biasing for low-power operation
- Advance signal processing
- Spur cancellation and RF co-existence mechanisms for common cellular, Bluetooth, DDR, LVDS, LCD interference mitigation.

### **Specifications:**

Parameters	Conditions	Min	Typical	Max	Unit
Operating Temperature Range	-	-40	Normal	125	°C
Maximum Soldering Temperature	IPC/JEDEC J-STD-020	-	-	260	°C
Working Voltage Value	-	2.5	3.3	3.6	V
I/O	$V_{IL}$	-0.3	-	$0.25V_{IO}$	V
	$V_{IH}$	$0.75V_{IO}$	-	3.6	
	$V_{OL}$	-	-	$0.1V_{IO}$	
	$V_{OH}$	$0.8V_{IO}$	-	-	
	$I_{MAX}$	-	-	12	mA
Electrostatic Discharge (HBM)	TAMB=25°C	-	-	2	KV
Electrostatic Discharge (CDM)	TAMB=25°C	-	-	0.5	KV

## **4.5 Bi-Directional Level Shifter:**

A level shifter in digital electronics, also called logic-level shifter or voltage level translation, is a circuit used to translate signals from one logic level or voltage domain to another, allowing compatibility between ICs with different voltage requirements, such as TTL and CMOS. Many modern full featured systems use level shifters to bridge domains between processors, logic, sensors, and other circuits. In recent years, the three most common logic levels are 1.8V, 3.3V, 5V, though other levels exist above and below these voltages too.





## 4.6 UART/Serial Communication:

UART stands for Universal Asynchronous Receiver/Transmitter. It's not a communication protocol like SPI and I2C, but a physical circuit in a microcontroller, or a stand-alone IC. A UART's main purpose is to transmit and receive serial data. The connection here is **TX to RX and RX to TX**.

## 4.7 Parallel communication:

In data transmission, parallel communication is a method of conveying multiple binary digits (bits) simultaneously. It contrasts with serial communication, which conveys only a single bit at a time; this distinction is one way of characterizing a communications link. The connection here is **Tx to Tx and Rx to Rx**.

## 4.8 Difference between Serial and Parallel communication:

Serial communication sends only one bit at a time. so, these require fewer I/O (input-output) lines. Hence, occupying less space and more resistant to cross-talk. The main advantage of serial communication is, the cost of the entire embedded system becomes cheap and transmits the information over a long distance. Serial transfer is used in DCE (Data Communication Equipment) devices like a modem.

In parallel communication, a chunk of data (8,16 or 32 bit) is sent at a time. So, each bit of data requires a separate physical I/O line. The advantage of parallel communication is it is fast but its drawback is it uses a greater number of I/O (input-output) lines. Parallel transfer is used in PC (personal computer) for interconnecting CPU (central processing unit), RAM (random access memory), modems, audio, video and network hardware.

**Note:** If the Integrated Circuit or processor supports less amount of Input/output pins it is better to opt serial communication.

Serial Communication	Parallel Communication
Sends data bit by bit at one clock pulse	Transfers a chunk of data at a time
Requires one wire to transmit the data	Requires 'n' number of lines for transmitting 'n' bits
Communication speed is slow	Communication speed is fast

Installation cost is low

Installation cost is high

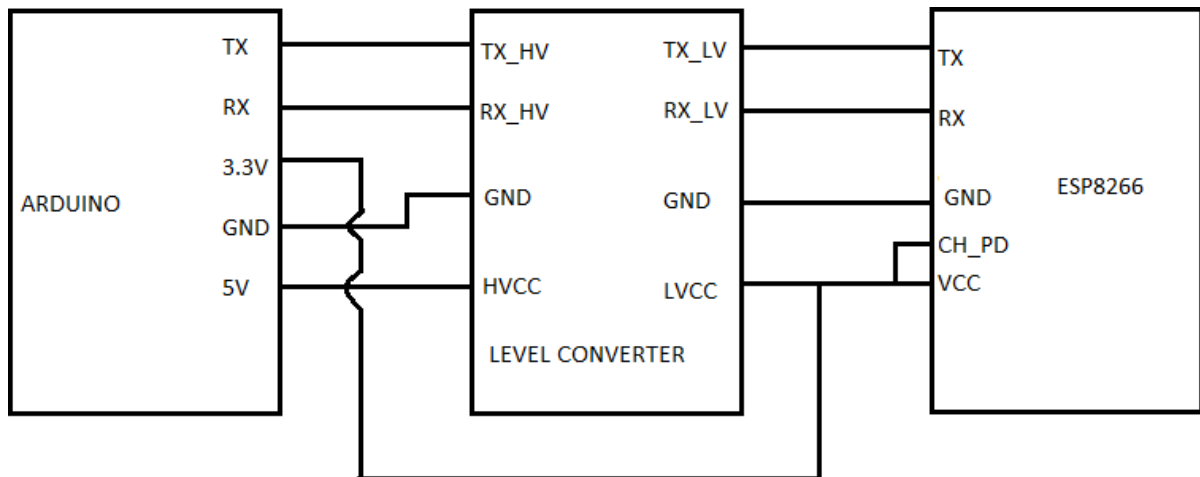
Preferred for long distance communication

Used for short distance communication

Example: Computer to Computer

Computer to multi function printer

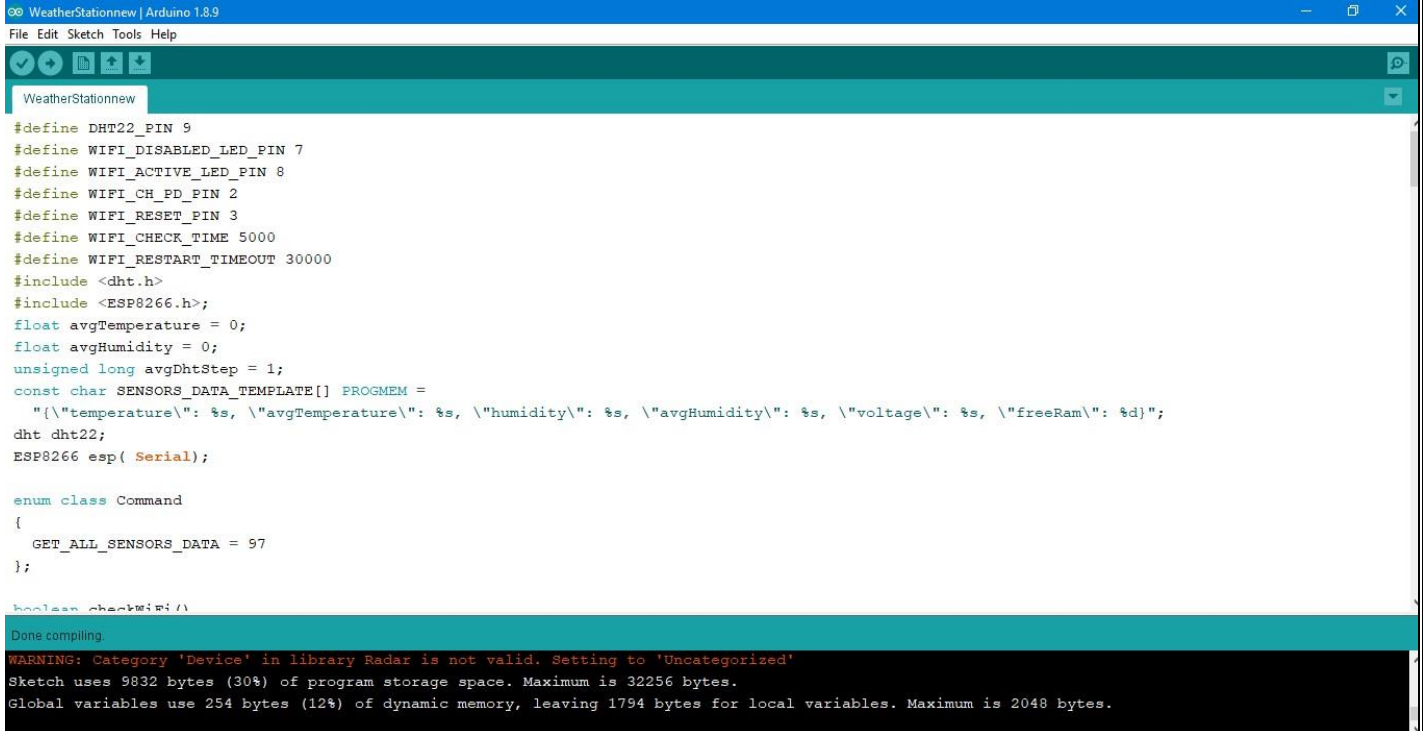
## **CONNECTING THE MODULES WITH LEVEL CONVERTER:**



### **Note:**

The above connection has been tested, verified and edited for the making of the project. And it has led to the compilation and uploading of the Arduino code and compilation of the android code only. Even though the code is getting compiled and uploaded, there is an issue (not error) during the run time in the serial monitor of the Arduino IDE. **The connection is however PARALLEL and not UART/SERIAL (i.e. Tx to Tx and Rx to Rx).** The output generated and as displayed in serial monitor is AT after every loop. It is unlike the anticipated result AT followed by TRUE or FALSE. The source code of the project is added.

## COMPILING



The screenshot shows the Arduino IDE interface with the sketch 'WeatherStationnew' open. The code is as follows:

```
#define DHT22_PIN 9
#define WIFI_DISABLED_LED_PIN 7
#define WIFI_ACTIVE_LED_PIN 8
#define WIFI_CH_PD_PIN 2
#define WIFI_RESET_PIN 3
#define WIFI_CHECK_TIME 5000
#define WIFI_RESTART_TIMEOUT 30000
#include <dht.h>
#include <ESP8266.h>;
float avgTemperature = 0;
float avgHumidity = 0;
unsigned long avgDhtStep = 1;
const char SENSORS_DATA_TEMPLATE[] PROGMEM =
    "{\"temperature\": %s, \"avgTemperature\": %s, \"humidity\": %s, \"avgHumidity\": %s, \"voltage\": %s, \"freeRam\": %d}";
dht dht22;
ESP8266 esp( Serial);

enum class Command
{
    GET_ALL_SENSORS_DATA = 97
};

bool checkWifi()
```

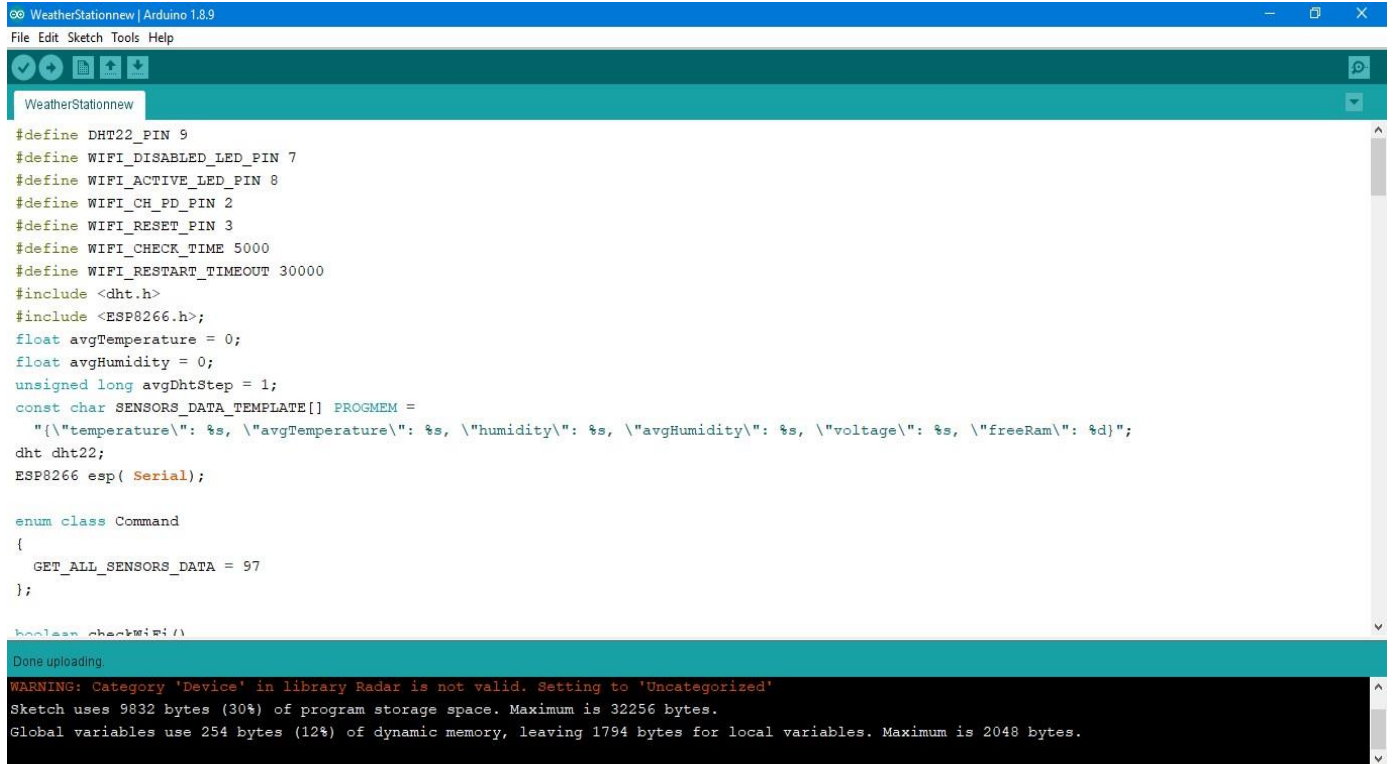
The status bar at the bottom indicates 'Done compiling.' Below the code editor, a warning message is displayed:

WARNING: Category 'Device' in library Radar is not valid. Setting to 'Uncategorized'

Sketch uses 9832 bytes (30%) of program storage space. Maximum is 32256 bytes.

Global variables use 254 bytes (12%) of dynamic memory, leaving 1794 bytes for local variables. Maximum is 2048 bytes.

## UPLOADING



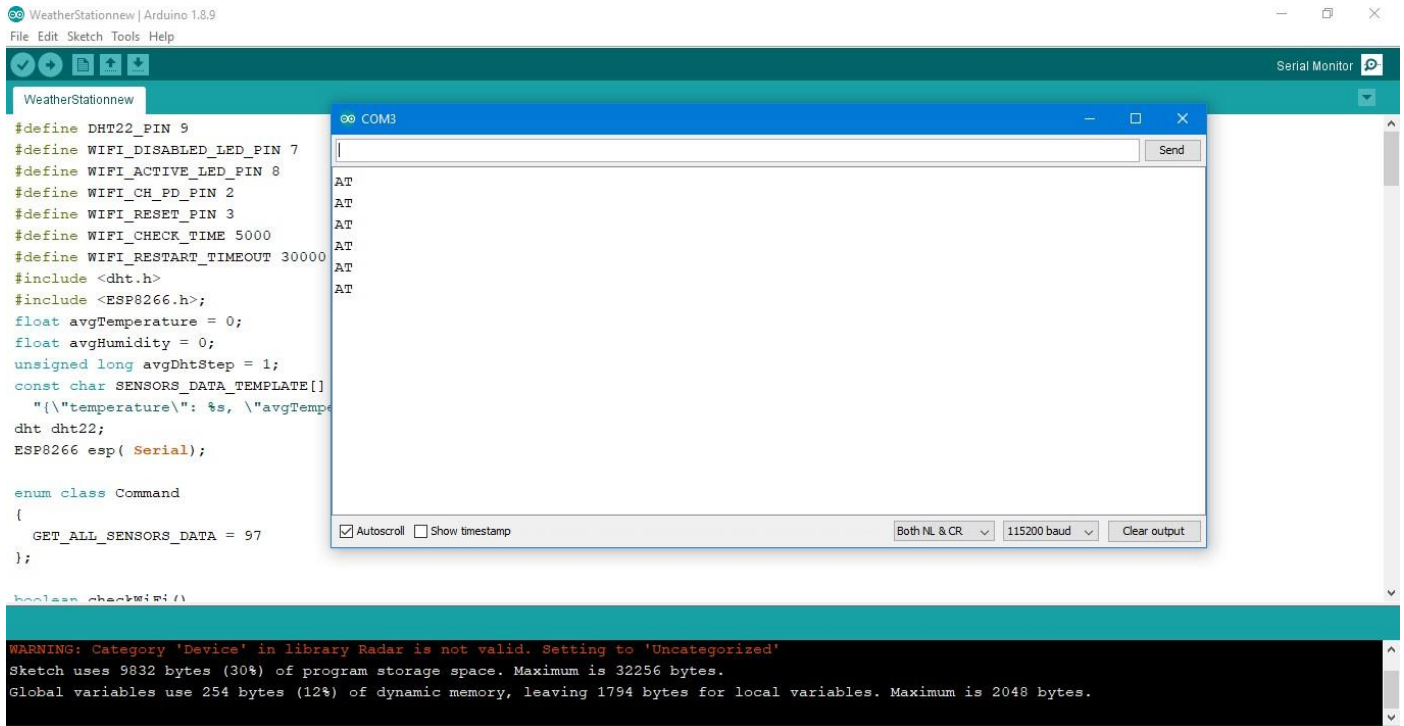
The screenshot shows the same Arduino IDE interface as the compilation step. The code is identical. The status bar at the bottom indicates 'Done uploading.' Below the code editor, the same warning message is displayed:

WARNING: Category 'Device' in library Radar is not valid. Setting to 'Uncategorized'

Sketch uses 9832 bytes (30%) of program storage space. Maximum is 32256 bytes.

Global variables use 254 bytes (12%) of dynamic memory, leaving 1794 bytes for local variables. Maximum is 2048 bytes.

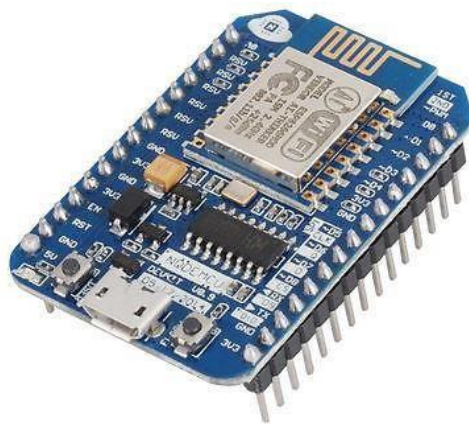
# SERIAL MONITOR



Since, the ESP8266 needs an additional Arduino board to function, we have shifted to NodeMCU.

## NodeMCU

NodeMCU is an open source [LUA](#) based firmware developed for ESP8266 wifi chip. By exploring functionality with ESP8266 chip, NodeMCU firmware comes with ESP8266 Development board/kit i.e. NodeMCU Development board.



**Fig: NodeMCU Development Board/kit v0.9 (Version1)**

Since NodeMCU is open source platform, their hardware design is open for edit/modify/build.

NodeMCU Dev Kit/board consist of ESP8266 wifi enabled chip. The **ESP8266** is a low-cost [Wi-Fi](#) chip developed by Espressif Systems with TCP/IP protocol. For more information about ESP8266, you can refer [ESP8266 WiFi Module](#).

There is Version2 (V2) available for NodeMCU Dev Kit i.e. **NodeMCU Development Board v1.0 (Version2)**, which usually comes in black colored PCB.



**Fig: NodeMCU Development Board/kit v1.0 (Version2)**

Now we're going to interface DHT22 with NodeMCU using Web Server.

## 4.9 Connecting DHT22 sensor to ESP8266 NodeMCU

Connecting DHT22 sensor to ESP8266 NodeMCU is simple. We start by placing the NodeMCU on to our breadboard, ensuring each side of the board is on a separate side of the breadboard.

Now we place the sensor on to our breadboard besides NodeMCU. We then connect VCC pin on the sensor to the 3.3V pin on the

NodeMCU and ground to ground. Also,

we connect Data pin on the sensor to D7

pin of the ESP8266 NodeMCU. Finally,

we need to place a pull-up resistor of

10K $\Omega$  between VCC and data line to keep

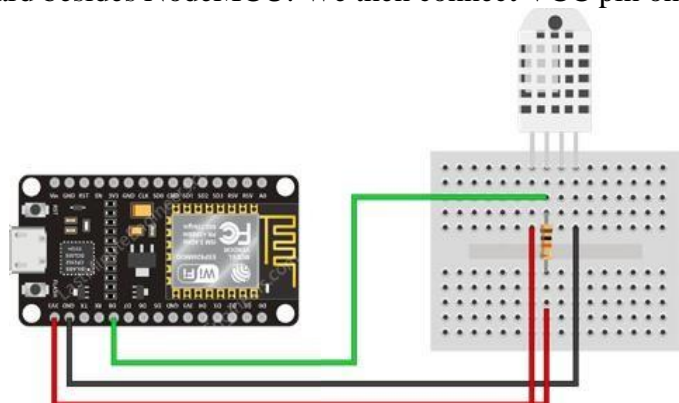
it HIGH for proper communication

between sensor and NodeMCU. If you

happen to have a breakout board of the

sensor, you need not add any external pull-up. It comes with a built-in pull-up resistor. When

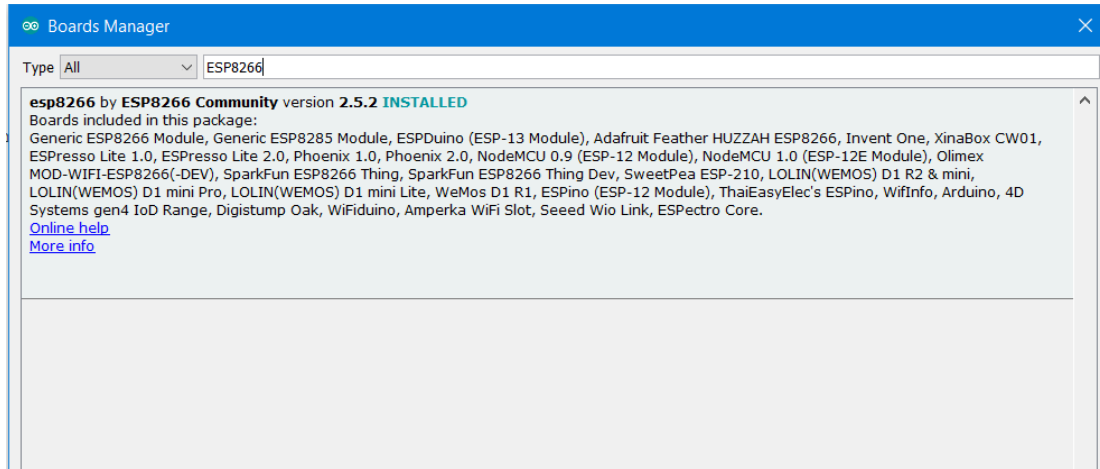
we're done, we should have something that looks similar to the illustration shown below.



Wiring DHT22 Temperature Humidity Sensor with ESP8266 NodeMCU

## 4.10 Installing ESP8266 Board

Navigate to Tools >> Boards >> Boards Manager >> Search “ESP8266”



Click on **Install** to download and install the esp8266 board by ESP8266 Community. This lets us to upload the codes to NodeMCU and communicate with it.

## 4.11 Installing DHT Sensor Library

Communicating with DHT22 sensors is a bunch of work, as they have their own single wire protocol for data transfer. And this protocol requires precise timing. Fortunately, we don't have to worry much about this because we are going to use the DHT library from Adafruit which takes care of almost everything. The library is so powerful that it runs on both Arduino and ESP architecture.

To install the library, navigate to the Sketch > Include Library > Manage Libraries. Wait for Library Manager to download libraries index and update list of installed libraries.

We will look for DHT sensor library by Adafruit. We click on that entry, and then we select Install.

The DHT sensor library uses the [Adafruit Sensor support backend](#). So, search the library manager for Adafruit Unified Sensor and install that too (you may have to scroll a bit)

## 4.12 Creating ESP8266 NodeMCU Web Server using Wi-Fi (AP) mode

We are going to configure our ESP8266 NodeMCU into Access Point (AP) mode, and create a web server to serve up web pages to any connected client under existing network.

Before we head for uploading the sketch, we need to make some changes to make it work for us. We need to modify the following two variables with our network credentials, so that another device can establish a connection with ESP8266 NodeMCU.

Once we are done, we will try the sketch out and then we will dissect it in some detail.

```
#include <ESP8266WiFi.h>

#include <ESP8266WebServer.h>

#include "DHT.h"

#define DHTTYPE DHT22 // DHT 22 (AM2302), AM2321

/*Put SSID & Password*/

const char* ssid = "CDAC_IOT"; // Enter SSID here
const char* password = "1234567890"; //Enter Password here

ESP8266WebServer server(80);

uint8_t DHTPin = D7;    // DHT Sensor Pin allocation

DHT dht(DHTPin, DHTTYPE);    // Initialize DHT sensor.
float Temperature, Humidity;

void setup()
{
  Serial.begin(115200);
  delay(100);
  pinMode(DHTPin, INPUT);
  dht.begin();
```

```

Serial.println("Configuring AP: ");
WiFi.softAP(ssid, password);      //begin WiFi connection
IPAddress myIP = WiFi.softAPIP();
Serial.print("AP IP address: ");
Serial.println(myIP);
server.on("/", handle_OnConnect);
server.onNotFound(handle_NotFound);
server.begin();
Serial.println("HTTP server started");
}

void loop()
{
  Temperature = dht.readTemperature();
  Serial.print("Temperature: ");
  Serial.print(Temperature);
  Serial.print("\t");
  Humidity = dht.readHumidity();
  Serial.print("Humidity: ");
  Serial.print(Humidity);
  Serial.print("\n");
  delay(1000);
  server.handleClient();
}

void handle_OnConnect()
{
  Temperature = dht.readTemperature(); // Gets the values of the temperature
  Humidity = dht.readHumidity(); // Gets the values of the humidity
  server.send(200, "text/html", SendHTML(Temperature,Humidity));
}

```



```
}
```

```
void handle_NotFound()
```

```
{
```

```
    server.send(404, "text/plain", "Not found");
```

```
}
```

```
String SendHTML(float Temperaturestat,float Humiditystat)
```

```
{
```

```
    String ptr = "<!DOCTYPE html> <html>\n";
```

```
    ptr +="<head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0, user-scalable=no\">\n";
```

```
    ptr +="<title>ESP8266 Weather Report</title>\n";
```

```
    ptr +="<style>html { font-family: Helvetica; display: inline-block; margin: 0px auto; text-align: center;}\n";
```

```
    ptr +="body{margin-top: 50px;} h1 {color: #444444;margin: 50px auto 30px;}\n";
```

```
    ptr +="p {font-size: 24px;color: #444444;margin-bottom: 10px;}\n";
```

```
    ptr +="</style>\n";
```

```
    ptr +="<script>\n";
```

```
    ptr +="setInterval(loadDoc,200);\n";
```

```
    ptr +="function loadDoc() {\n";
```

```
    ptr +="var xhttp = new XMLHttpRequest();\n";
```

```
    ptr +="xhttp.onreadystatechange = function() {\n";
```

```
    ptr +="if (this.readyState == 4 && this.status == 200) {\n";
```

```
    ptr +="document.getElementById(\"webpage\").innerHTML =this.responseText}\n";
```

```
    ptr +="};\n";
```

```
    ptr +="xhttp.open(\"GET\", \"^\", true);\n";
```

```
    ptr +="xhttp.send();\n";
```

```
    ptr +="}\n";
```

```
    ptr +="</script>\n";
```

```
    ptr +="</head>\n";
```

```
    ptr +="<body>\n";
```

```

ptr += "<div id=\"webpage\">\n";
ptr += "<h1>CDAC - Temperature & Humidity</h1>\n";
ptr += "<p>Temperature: ";
ptr += (float)Temperaturestat;
ptr += "°C</p>";
ptr += "<p>Humidity: ";
ptr += (float)Humiditystat;
ptr += "%</p>";
ptr += "</div>\n";
ptr += "</body>\n";
ptr += "</html>\n";

return ptr;
}

```

## 4.13 Accessing the Web Server

After uploading the sketch, we will open the Serial Monitor at a baud rate of 115200. And press the RESET button on the NodeMCU. If everything is OK, it will output the dynamic IP address obtained from your router and show HTTP server started message.

Next, load up a browser and point it to the IP address shown on the serial monitor. The ESP8266 NodeMCU should serve up a web page showing temperature and relative humidity.

## 4.14 Detailed Code Explanation

The sketch starts by including ESP8266WiFi.h library. This library provides ESP8266 NodeMCU specific Wi-Fi methods we are calling to connect to network. Following that we also



### ESP8266 Weather Report

Temperature: 28°C

Humidity: 95%

include the ESP8266WebServer.h library, which has some methods available that will help us setting up a server and handle incoming HTTP requests without needing to worry about low level implementation details. Finally, we include DHT.h library.

```
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include "DHT.h"
```

Next, we need to define the type of DHT sensor we are using. Uncomment one of the lines below accordingly!

```
#define DHTTYPE DHT22 // DHT 22
```

As we are configuring ESP8266 NodeMCU in Station (STA) mode, it will join existing Wi-Fi network. Hence, we need to provide it with your network's SSID & Password.

```
/*Put your SSID & Password*/
const char* ssid = "NetworkName"; // Enter SSID here
const char* password = "Password"; //Enter Password here
```

Next, we declare an object of ESP8266WebServer library, so we can access its functions. The constructor of this object takes port (where the server will be listening to) as a parameter. Since 80 is the default port for HTTP, we will use this value. Now you can access the server without needing to specify the port in the URL.

```
// declare an object of WebServer library
ESP8266WebServer server(80);
```

Next, we need to define the ESP8266 NodeMCU's pin number to which our sensor's Data pin is connected and create a DHT object. So, that we can access special functions related to the DHT library.

```
// DHT Sensor
uint8_t DHTPin = D7;
// Initialize DHT sensor.
DHT dht(DHTPin, DHTTYPE);
```

Two float variables viz. Temperature & Humidity are declared to store respective values.

```
float Temperature, Humidity;
```

## ➤ Inside Setup() Function

Inside Setup() Function we configure our HTTP server before actually running it. First of all, we open a serial connection for debugging purpose and set GPIO ports to INPUT. We also need to initialize the DHT object using `begin()` function.

```
Serial.begin(115200);  
delay(100);  
pinMode(DHTPin, INPUT);
```

```
dht.begin();
```

In order to handle incoming HTTP requests, we need to specify which code to execute when a URL is hit. To do so, we use `on` method. This method takes two parameters. First one is a URL path and second one is the name of function which we want to execute when that URL is hit.

The code below indicates that when a server receives an HTTP request on the root (/) path, it will trigger the `handle_OnConnect` function. Note that the URL specified is a relative path.

```
server.on("/", handle_OnConnect);
```

We haven't specified what the server should do if the client requests any URL other than specified with `server.on`. It should respond with an HTTP status 404 (Not Found) and a message for the user. We put this in a function as well, and use `server.onNotFound` to tell it that it should execute it when it receives a request for a URL that wasn't specified with `server.on`

```
server.onNotFound(handle_NotFound);
```

Now, to start our server, we call the `begin` method on the server object.

```
server.begin();  
Serial.println("HTTP server started");
```

## ➤ Inside Loop() Function

To handle the actual incoming HTTP requests, we need to call the `handleClient()` method on the server object.

```
server.handleClient();
```

Next, we need to create a function we attached to root (/) URL with `server.on`. Remember? At the start of this function, we get the values of temperature and humidity from the sensor. In order to respond to the HTTP request, we use the `send` method. Although the method can be called with a different set of arguments, its simplest form consists of the HTTP response code, the content type and the content.

In our case, we are sending the code 200 (one of the [HTTP status codes](#)), which corresponds to the OK response. Then, we are specifying the content type as "text/html", and finally we are calling `SendHTML()` custom function which creates a dynamic HTML page containing values of temperature and humidity.

```
void handle_OnConnect()
{
  Temperature = dht.readTemperature(); // Gets the values of the temperature
  Humidity = dht.readHumidity(); // Gets the values of the humidity
  server.send(200, "text/html", SendHTML(Temperature, Humidity));
}
```

Likewise, we need to create a function to handle 404 Error page.

```
void handle_NotFound(){
  server.send(404, "text/plain", "Not found");
}
```

## ➤ Displaying the HTML Web Page

`SendHTML()` function is responsible for generating a web page whenever the ESP8266 NodeMCU web server gets a request from a web client. It merely concatenates HTML code into a big string and returns to the `server.send()` function we discussed earlier. The function takes values of temperature and humidity as a parameter to dynamically generate the HTML content.

The first text you should always send is the `<!DOCTYPE>` declaration that indicates that we're sending HTML code.

```
String SendHTML(float Temperaturestat, float Humiditystat) {
String ptr = "<!DOCTYPE html> <html>\n";
```

Next, the `<meta>` viewport element makes the web page responsive in any web browser, while title tag sets the title of the page.

```
ptr += "<head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0, user-
scalable=no\">\n";
ptr += "<title>ESP8266 Weather Report</title>\n";
```

## ➤ Styling the Web Page

Next, we have some CSS to style the web page appearance. We choose the Helvetica font, define the content to be displayed as an inline-block and aligned at the center.

```
ptr += "<style>html { font-family: Helvetica; display: inline-block; margin: 0px auto; text-align:
center; }\n";
```

Following code then sets color, font and margin around the body, H1 and p tags.

```
ptr += "body{margin-top: 50px;} h1 {color: #444444;margin: 50px auto 30px;} \n";
ptr += "p { font-size: 24px;color: #444444;margin-bottom: 10px;} \n";
```

```
ptr += "</style>\n";
ptr += "</head>\n";
ptr += "<body>\n";
```

## ➤ Setting the Web Page Heading

Next, heading of the web page is set; you can change this text to anything that suits your application.

```
ptr += "<div id=\"webpage\">\n";
ptr += "<h1>ESP8266 Weather Report</h1>\n";
```

## ➤ Displaying Temperature and Humidity on Web Page

To dynamically display values of Temperature & humidity, we put those values in paragraph tag. These values are converted to integer by type casting. To display degree symbol, we use HTML entity &deg;.

```
ptr += "<p>Temperature: ";
ptr += (float)Temperaturestat;
ptr += "°C</p>";
ptr += "<p>Humidity: ";
ptr += (float)Humiditystat;
ptr += "%</p>";
```

```
ptr += "</div>\n";
ptr += "</body>\n";
ptr += "</html>\n";
return ptr;
}
```

## ➤ Dynamically load Sensor Data with AJAX

Refreshing a web page isn't too practical if you have a heavy web page. A better method is to use [Asynchronous JavaScript And Xml](#) (AJAX) so that we can request data from the server asynchronously (in the background) without refreshing the page.

The [XMLHttpRequest](#) object within JavaScript is commonly used to execute AJAX on webpages. It performs the silent GET request on the server and updates the element on the page. AJAX is not a new technology, or different language, just existing technologies used in new ways. Besides this, AJAX also makes it possible to

- Request data from a server after the page has loaded

- Receive data from a server after the page has loaded
- Send data to a server in the background

Here is the AJAX script that we'll be using. Place this script just before you close `</head>` tag.

```
ptr += "<script>\n";
ptr += "setInterval(loadDoc,200);\n";
ptr += "function loadDoc() {\n";
ptr += "var xhttp = new XMLHttpRequest();\n";
ptr += "xhttp.onreadystatechange = function() {\n";
ptr += "if (this.readyState == 4 && this.status == 200) {\n";
ptr += "document.getElementById(\"webpage\").innerHTML =this.responseText}\n";
ptr += "};\n";
ptr += "xhttp.open(\"GET\", \"/\", true);\n";
ptr += "xhttp.send();\n";
ptr += "}\n";
ptr += "</script>\n";
```

The script starts with `<script>` tag, as AJAX script is nothing but a javascript so, we need to write it in `<script>` tag. In order for this function to be repeatedly called, we will be using the javascript `setInterval()` function. It takes two parameters – a function to be executed and time interval (in milliseconds) on how often to execute the function.

```
ptr += "<script>\n";
ptr += "setInterval(loadDoc,200);\n";
```

The heart of this script is a `loadDoc()` function. Inside this function, an `XMLHttpRequest()` object is created. This object is used to request data from a web server.

```
ptr += "function loadDoc() {\n";
ptr += "var xhttp = new XMLHttpRequest();\n";
```

The `xhttp.onreadystatechange()` function is called every time the `readyState` changes. The `readyState` property holds the status of the `XMLHttpRequest`. It has one of the following values.

- 0: request not initialized
- 1: server connection established
- 2: request received
- 3: processing request
- 4: request finished and response is ready



The status property holds the status of the XMLHttpRequest object. It has one of the following values.

- 200: "OK"
- 403: "Forbidden"
- 404: "Page not found"

When readyState is 4 and status is 200, the response is ready. Now, the content of element with id webpage (div holding values of temperature & humidity) is updated.

```
ptr += "xhttp.onreadystatechange = function() {\n";  
ptr += "if (this.readyState == 4 && this.status == 200) {\n";  
ptr += "document.getElementById(\"webpage\").innerHTML =this.responseText}\n";  
ptr += "};\n";
```

The HTTP request is then initiated via the open() and send() functions.

```
ptr += "xhttp.open(\"GET\", \"/\", true);\n";  
ptr += "xhttp.send();\n";  
ptr += "}\n";
```

## CONCLUSION

Here, at the end of the project we have built and developed an autonomous robotic platform which has intended application in agriculture. This robot is still under development to a scaled-up model which will be used directly in the fields and we can control it remotely by commanding it to start, stop and steer to desired position. The prototype robot (TurtleBot3 Burger) is anticipated to be controlled using ROS (Robotic Operating System).

We are also developing a module which displays the current temperature and humidity using a DHT22 temperature-humidity sensor which is connected to Arduino UNO through a web server. This module(server) sends the data to the client using an ESP8266 Wi-Fi or NodeMCU. This web page will also have a light and fan button. The fan will automatically start running as soon as the temperature crosses a limit. This module will be distributed to schools in remote places.