

# **Connected and Autonomous Robotic System**

Project Workbook

By

Jay Sureshbhai Parsana

Prashant Shushilbhai Gandhi

Saumil Shah

Vatsal Makani

7<sup>th</sup> November 2019

**Advisor:** Prof. Kaikai Liu

## Chapter 1. Literature Search, State of the Art

### Literature Search

Since the past decade research on autonomous systems like self-driving cars, robots, etc is on the rise [2] and the tech industry will continue to develop techniques and algorithms to achieve complete self-driving experience. Using such algorithms, “experiments have broadened their scope and autonomous driving for several kilometers and over long periods of time” have become commercialized and available to all [5]. In recent times, companies are using laser-based radar (LIDAR) as well as traditional radars in robotic systems in order to add the surrounding awareness capabilities in these systems. Vijay Subramanian, Thomas F. Burks & A.A. Arroyo (2006) explained the use of radar incorporated with computer vision in autonomous tractors for citrus grove applications [1]. The authors explain the implementation of the “PID control algorithm to control the tractor using the information from the machine vision system and laser radar”, but the only drawback was latency communication from radar to the platform that processes the radar data [1]. Zhengang Li, Yong Xiong, Lei Zhou (2017) explains the working of the indoor wheelchair performing the autonomous exploration based on inexpensive RGB camera and ROS. The authors explain that the wheelchair rotary systems are controlled by the robot operating system in the host computer and RGB camera is used for depth perception, indoor path planning and obstacle avoidance [4]. The Arduino microcontroller is then used to receive the data from the camera and ROS and accordingly sets the PWM signals for the motors controlling the navigation of the wheelchair.

There are many applications where LIDAR proves advantageous over radars. LIDARs are used in applications that involve short-range detection and radars are used for long-range tracking. LIDARs can be used to create a larger 3D field of view and accurate range information which is suitable for obstacle detection as well as pedestrian recognition as explained by Heng Wang, Bin Wang, Bingbing Liu, Xiaoli Meng & Guanghong Yang (2016) [2]. The authors also explain the integration of LIDARs with IMU sensors and filters for implementing localization and pedestrian recognition [2].

Some of the airborne automated systems require navigation control based on IMU sensor units like accelerometers, magnetometer, gyroscope, etc. integrated along with location data. Jan Wendel, Oliver Meister, Christian Schlaile, Gert F. Trommer (2006) explain an integrated navigation system for an airborne robotic system that uses GPS and MEMS IMU sensors [3]. The authors propose the solutions for two scenarios: first when GPS data is available and second when GPS is lost [3]. Another replacement for GPS can be ultrasonic beacon which can be used for indoor positioning. Dongho Kang & Young-Jin Cha (2018) explains the use of ultrasonic beacon in the scenarios where GPS fails to work for a UAVs. The authors explain the application of a “deep convolutional neural network (CNN) for damage detection and a geotagging method for the localization of damage” [8]. “Localization, mapping, and path planning” are the most important algorithms used for any system to be autonomous. Localization, motion planning and path planning plays an

important role in deciding the shortest path that any autonomous system can travel in minimum time [6].

If an autonomous system needs to monitor the surroundings keeping the functionalities such as localization, path planning, and mapping, it is mandatory to define two different architectures communicating via IoT. LoRA is another wide area network telecommunication system which is widely used for IoT Architecture [7]. LoRA provides higher bandwidth and is basically targeted for low power systems like embedded microcontrollers.

### **State-of-the-Art Summary**

The higher demand for autonomous and IoT based systems in the tech industry has accelerated the development of new techniques to optimize the current state-of-the-art solutions. The objective of the project is to provide a platform that can be utilized for any autonomous applications in robots, cars, warehouses, hospitals, etc. We plan on minimizing the time for developing such autonomous system by providing a base platform on which many OEM's can integrate their hardware and use our API's to get a head-start for their system. This project consists of a low-level architecture that takes care of sensing, positioning, and obstacle avoidance and high-level architecture to compute localization, path planning, and generate control signals for low-level architecture. Architectures talk to each other using IoT framework such as LoRA or Wi-Fi. The communication between these architectures occurs at a very high speed, thus solving the problem of slow communication faced by Vijay, Thomas, and Arroyo [1].

The implementation of low-level architecture can be based on ST Microelectronics' STM32 platform and PX4 platform by px4.io. Due to the integration of many sensors in low-level architecture, the use of FreeRTOS will help to achieve task synchronization. Each task is dedicated to each sensor that executes in real-time by using various scheduling algorithms. STM32 has inbuilt support of CMSIS v2 to add real-time capabilities. We researched various technologies for establishing an IoT link such as the Verizon IoT module, Wi-Fi, LoRa. The Wi-Fi module in the low-level architecture enables the system to send sensor data to cloud services like AWS, Google IoT, Microsoft Azure, etc. Verizon IoT combines modules, network services, and the ThingSpace management platform to support IoT applications with minimal annual fee which is ideal for any embedded application.

Decaware provides a DWM1004C module for tagging and tracking applications. DWM1004C offers high accuracy, real-time location capability with a 6.8Mbps data rate which is useful for tag applications that are like RFID. Another tracking module is TREK1000 which is a two-way ranging module for real-time positioning applications. This module works in 3 modes to enable tracking, geo-fencing and navigation schemes.

The high-level architecture uses Nvidia Jetson TX2 for complex calculations like localization, path planning, and motion planning. The high-level architecture is based on Linux. The high-level system does not guarantee real-time operations. The software platform explored for high-level architectures are Autoware.AI and traditional ROS.

Autware.AI is an open-source software that implements localization with the help of 3D maps and SLAM algorithms, detection based on LIDAR and cameras and prediction based on deep learning neural networks. ROS provides APIs with inherent support for localization, path planning, and robot control.

## References

1. Subramanian, V., Burks, T. F., & Arroyo, A. A. (2006). Development of machine vision and laser radar based autonomous vehicle guidance systems for citrus grove navigation. *Computers and electronics in agriculture*, 53(2), 130-143.

[“This paper discusses the development of an autonomous guidance system for use in a citrus grove using machine vision and LIDAR”]

2. Wang, H., Wang, B., Liu, B., Meng, X., & Yang, G. (2017). Pedestrian recognition and tracking using 3D LiDAR for autonomous vehicle. *Robotics and Autonomous Systems*, 88, 71-78.

[“This paper studies the pedestrian recognition and tracking problem for autonomous vehicles using a 3D LiDAR, a classifier trained by SVM (Support Vector Machine) is used to recognize pedestrians”]

3. Wendel, J., Meister, O., Schlaile, C., & Trommer, G. F. (2006). An integrated GPS/MEMS-IMU navigation system for an autonomous helicopter. *Aerospace Science and Technology*, 10(6), 527-533.

[“This paper addresses the development of an integrated navigation system based on MEMS inertial sensors and GPS for a VTOL-MAV. These MAVs show an inherent instability that makes at least an automatic stabilization necessary”]

4. Li, Z., Xiong, Y., & Zhou, L. (2017, December). ROS-Based Indoor Autonomous Exploration and Navigation Wheelchair. In *2017 10th International Symposium on Computational Intelligence and Design (ISCID)* (Vol. 2, pp. 132-135). IEEE.

[“Aiming at the current situation of high cost, complicated construction and poor reusability of the autonomous navigation system, an indoor wheelchair autonomous exploration and navigation system with low cost and high reusability is realized in this paper”]

5. Bresson, G., Alsayed, Z., Yu, L., & Glaser, S. (2017). Simultaneous localization and mapping: A survey of current trends in autonomous driving. *IEEE Transactions on Intelligent Vehicles*, 2(3), 194-220.

[“This paper proposes a survey of the simultaneous localization and mapping field when considering the recent evolution of autonomous driving, present the limits of classical

approaches for autonomous driving and review the methods where the identified challenges are solved”]

6. Fethi, D., Nemra, A., Louadj, K., & Hamerlain, M. (2018). Simultaneous localization, mapping, and path planning for unmanned vehicle using optimal control. *Advances in Mechanical Engineering*, 10(1), 1687814017736653.

[“In this paper, investigation of the path planning problem of unmanned ground vehicle is based on optimal control theory and simultaneous localization and mapping”]

7. Augustin, A., Yi, J., Clausen, T., & Townsley, W. (2016). A study of LoRa: Long range & low power networks for the internet of things. *Sensors*, 16(9), 1466.

[“This paper provides an overview of LoRa and an in-depth analysis of its functional components”]

8. Kang, D., & Cha, Y. J. (2018). Autonomous UAVs for Structural Health Monitoring Using Deep Learning and an Ultrasonic Beacon System with Geo-Tagging. *Computer-Aided Civil and Infrastructure Engineering*, 33(10), 885-902.

[“This paper proposes an autonomous UAV method using ultrasonic beacons to replace the role of GPS, a deep convolutional neural network (CNN) for damage detection, and a geo-tagging method for the localization of damage”]

9. Grewal, H., Matthews, A., Tea, R., & George, K. (2017, March). Lidar-based autonomous wheelchair. In *2017 IEEE Sensors Applications Symposium (SAS)* (pp. 1-6). IEEE.

[“This paper seeks to mitigate these problems by presenting a wheelchair capable of autonomous navigation, with minimal directive from the user”]

## **Chapter 2. Project Justification**

There has been an increasing research interest and demand in the autonomous self-driving systems and IoT connected devices for the past decade. Currently, in the field of robotics and autonomous self-driving systems, platforms such as Slamtec Slamware exist. These platforms do not provide complete end to end solution that can be deployed on a robotic or automotive system and thus cannot be used to quickly prototype a product. Thus, engineers need to research and test different hardware modules and their software compatibility which suits their application requirements. We thereby propose to solve this problem using a generalized connected and autonomous robotic platform.

The purpose of our project is to develop a foundation which can be used for various applications in medical, robotics, industrial and automotive domain and targets to reduce the time to market by providing a complete package of hardware and software solutions for autonomous systems. This robotic system could be deployed directly on systems like cars, wheelchairs, industrial robots, etc. to adapt to various autonomous applications. Our project is an amalgam of embedded systems, IoT, distributed sensor networks, deep learning applications, etc. Our project shall comprise of 2 levels of architecture where low-level architecture supporting the integration of various kinds of sensors like telemetry, environmental and proximity sensors with the microcontroller for monitoring the surrounding parameters like environmental or an obstacle in the proximity. The high-level architecture is a Linux based Nvidia Jetson TX2 which supports computation of complex frameworks responsible for making the robotic system autonomous. Our project shall also provide IoT/Cloud support for data storage and monitoring applications.

### Chapter 3. Identify Baseline Approaches

After intensive literature review of implementations of various autonomous systems and sensor integration, the baseline approaches for low-level architecture and high-level architecture were decided. There are few software platforms available in the market which can implement localization, path planning and control for the high-level architecture. Autoware.AI provides inherent support for these algorithms but it is compatible only with heavy computing hardware like Intel Core processors, x86 architectures and Nvidia Xavier development kits. Since Nvidia Xavier has similar architecture to Nvidia Jetson TX2, it is possible to evaluate the Autoware's software platform performance in TX2. The traditional ROS provides native support for APIs used for complex algorithms in robotics.

As a baseline product, we would like to implement the one explained in the paper "LIDAR based Autonomous wheelchair" [9]. The authors explained the implementation of LIDAR to navigate the wheelchair. LIDAR provides the surrounding measurements and mapping around the wheelchair, sends the navigation information to microcontrollers and rotary system to move the wheelchair in the dedicated path to avoid any obstacles with the help of ROS. Our project shall implement a similar base architecture with additional features such as IoT support and other sensor information. The authors suggest the use of PC based x86 architecture to implement a high-level system for ROS used for localization and object detection. Whereas our project proposes to use ARM-based NVIDIA Jetson TX2 board with GPU providing low powered operation and universally deployable platform. We also will be using an ARM-based STM32 low-level platform which is scalable, mature and widely used in the industry instead of Arduino based platform underpowered platform.

For low-level architecture, we researched PX4 and STM32 platforms, but integration of FreeRTOS with the PX4 platform turns out to be a time-consuming task since PX4 APIs are not developed to provide real-time performance and STM32 provides inherent support for FreeRTOS which makes it suitable for low power, real-time applications. We have found few research papers regarding the autonomous systems integrated with GPS, IMU sensors, deep learning neural networks, radars and LIDARs for applications like pedestrian detection, citrus grove navigation, oceanography, etc. The author of one of the research papers explains the application of a beacon when the GPS data is lost, the application of LIDAR for obstacle detection as well as prediction using deep learning neural networks.

The communication between low-level STM32 platform and high-level Nvidia Jetson platform can take place using any one of the device protocols like UART, SPI or I2C. The cloud and IoT support at a low level is used for remote monitoring the sensor data such as motion readings, GPS readings, environmental parameters, etc.

## Chapter 4. Dependencies and Deliverables

### Dependencies

Our platform will provide features like object detection, coordinates of system, localization and path planning. While implementing these features, we need to consider some of the dependencies:

1. Getting GPS coordinates in closed areas such as buildings, warehouses, hospitals, etc.
2. For object avoidance, LIDARS fail to detect smooth surfaces such as walls. LIDARS are also sensitive to excessive light, causing it to give false readings.
3. Object detection – Prepare, define and find dataset relevant to the applications of our project. Choose appropriate machine learning algorithms and models for the best accuracy.
4. Establish communication between STM32 and Jetson TX2. Detection of obstacles and co-ordination of that information to avoid it will be a challenge as both are independent.
5. Integration of obstacle data from the high-level and low-level system to make a coordinated action to avoid it.
6. Implement APIs that are simple to use and easy to understand as a product.
7. Design the system so that it can be compatible with any future modifications.

### Deliverables

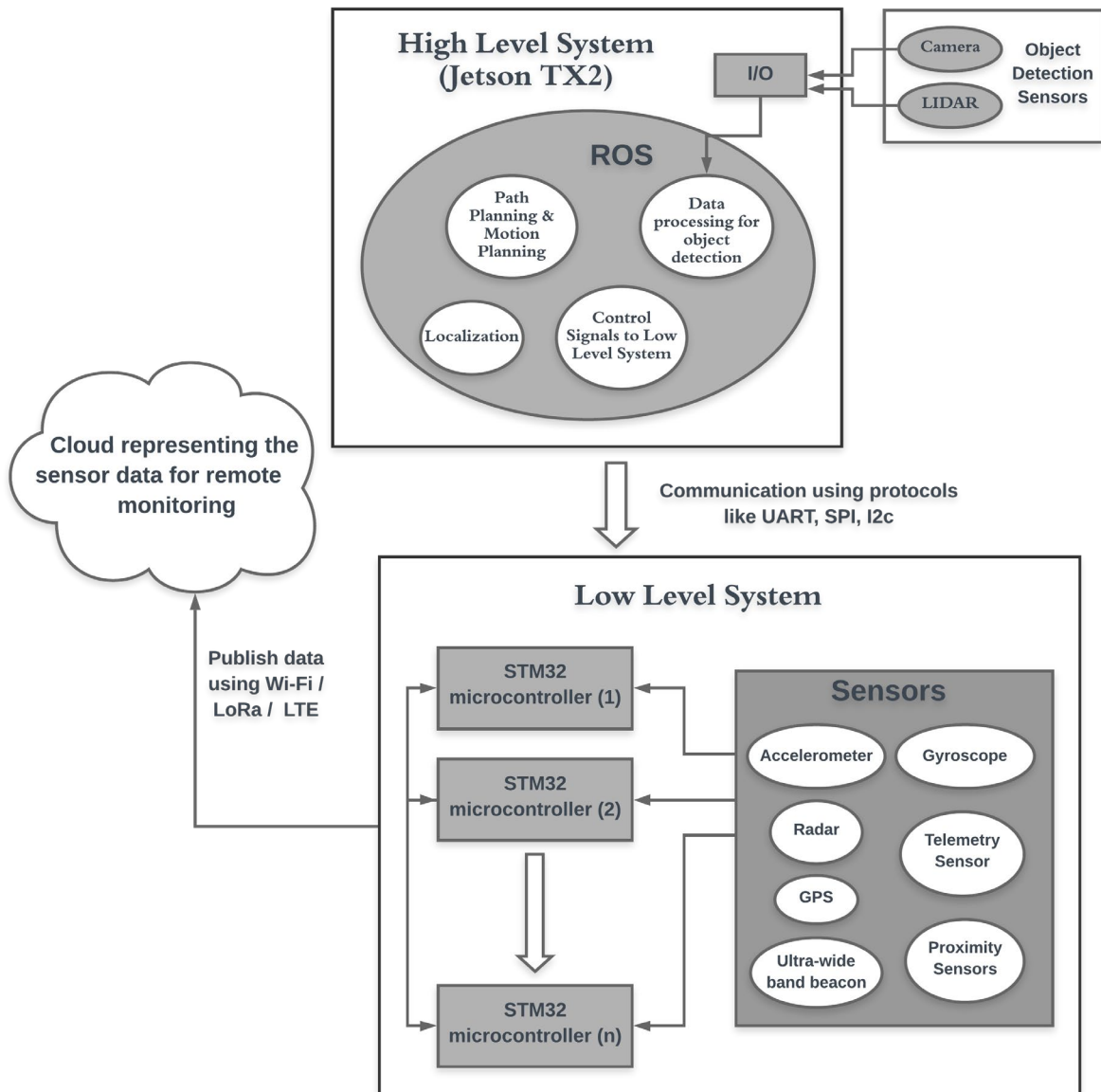
Our project's objective is to build a generic platform that can be used by different robotic systems. For low-level architecture we plan to interface sensors such as accelerometer, gyroscope, GPS, Ultra-wideband beacon, IoT integration, IMU sensors, proximity sensors, and radars. We will deploy multiple STM32 boards which will communicate with each other through a common bus. Some redundant controllers and sensors will also be added for the system to be fail safe.

A high-level system will comprise of a Nvidia Jetson TX2 board on ROS for processing data such as images from camera and point cloud from stereo camera. The system will compute path planning and motion planning algorithms using the data from camera and LIDAR. It will also be used to implement machine learning models and detect obstacles. The high-level architecture will generate some control signals and control the robotic system partially based on camera and LIDAR input.



## Chapter 5. Project Architecture

The block diagram below shows a clear picture of what we will be trying to achieve and how various systems will communicate with each other to achieve autonomy:



The project architecture will be divided into two levels:

- Low-level architecture
  - The low-level architecture consists of an embedded system which is the STM32 IoT discovery kit with FreeRTOS.
  - IMU sensors will be used to get feedback while doing the motion planning.
  - Radar and proximity sensors will detect the obstacles and a control action will be taken to avoid them.
  - Ultra-wideband beacon will be used for the localization of the robot. It will connect with the nearest beacon for reference to find out where is it located.
  - Telemetry sensors will be used to monitor the speed of the robotic vehicle and take control actions like reducing and increasing the speed of the robot.
  - IoT integration for STM32 will be done using Wi-Fi or LoRa. It will be used to publish sensor data on the cloud for monitoring it remotely.
  - A low-level system will get control signals from a high-level system. This control signal will be used to avoid obstacles. The communication link will be established using SPI, I2c or UART protocol.
- High-level architecture
  - The high-level architecture will run on the **Robot Operating System (ROS)**. It is an OS that has several API's for path planning and motion planning based on the camera and LIDAR.
  - The data from the camera and LIDAR will be processed parallelly by utilizing the GPU.
  - Obstacle detection by implementing a deep learning model will also be achieved by using the camera feed.
  - A 3D point cloud from LIDAR will also assist in obstacle avoidance.
  - All this data and algorithm implementation will, in turn, be converted into control signals. These signals will be communicated to the low-level system so that it can avoid obstacles.

## Chapter 6. Evaluation Methodology

As our project is to design a generic platform which can be deployed on any system to make them autonomous, there is one open source hardware available in the market which can make any flying vehicle autonomous. We can evaluate this hardware and take advantage of its architecture and if possible, we do some changes and use that hardware for other platform as well.

### PIXHAWK 4

Pixhawk 4 is an open source hardware design and it is an advanced autopilot system. It has STM32F765 processor as its main FMU processor. It has four on board sensor - Accelerometer/Gyroscope ICM-20689 for motion tracking of system, Accelerometer/Gyroscope BMI055, Magnetometer IST8310 and Barometer MS5611. It runs PX4 on top of NuttX OS. PX4 is an open source firmware. We can fork it into our GitHub account, and we can make the necessary changes into firmware as per our requirements. Following sections will give a detailed explanation of every necessary terms one needs to know before getting started with Pixhawk 4 autopilot system.

#### **NuttX OS:**

NuttX is a Real-Time Operating System (RTOS) developed for 8-bit to 32-bit microcontrollers. Its design is primary based on POSIX and ANSI standards, but it includes some of the functionalities of UNIX standards. Main features of NuttX OS are as below.

- POSIX/ANSI-like task controls, named message queues, counting semaphores, clocks/timers, signals, pthreads, robust mutexes, cancellation points, environment variables, filesystem.
- Realtime, deterministic, with support for priority inheritance.
- System logging.
- Realtime, deterministic, with support for priority inheritance.

NuttX has bash-like shell which is called Nuttshell. As it is based on POSIX standards it supports all shell commands which supported by POSIX.

#### **PX4**

PX4 is an open source software for flight control and specifically used for drones or any other unmanned vehicles. Features of PX4:

- It has a port-based architecture meaning if we add sensors or replace existing sensors, it doesn't affect its performance.
- It focuses on autonomy of system. It has over 1000 parameters which can be controlled easily by developers to make system more robust and smoother.
- It is written in C++ language.
- Supports many devices like GPS, Telemetry wifi and wifi modules whose APIs are already written.

- PX4 supports uORB messages.

Firmware for all sensors mentioned above, GPS, Telemetry wifi modules are written in PX4. PX4 runs on NuttX OS which is responsible for task scheduling. Using uORB messaging we can check different parameter's values which are being changed in particular task. uORB messages support Subscribe/Publish mechanism meaning if we want to monitor parameter from task X we need to subscribe for that parameter using Subscribe() APIs. Also, it is used of inter-thread/process communications. For example, if we want to subscribe for sensor data we can add following line in our sensor code: **int sensor\_sub\_fd = orb\_subscribe(ORB\_ID(sensor\_combined));**

### QGC (QGroundControl) :

QGC is a ground control software which helps us to upload firmware wirelessly using telemetry wifi module. We can calibrate Pixhawk 4 onboard sensors from QGC. We can monitor and change parameters from QGC without doing any change in the code itself. We can plan our mission in which we can set path of our vehicle its flying speed, hovering time at set points and altitude between two set points. Communication between QGC and PX4 is done using MAVLink wireless communication protocol.

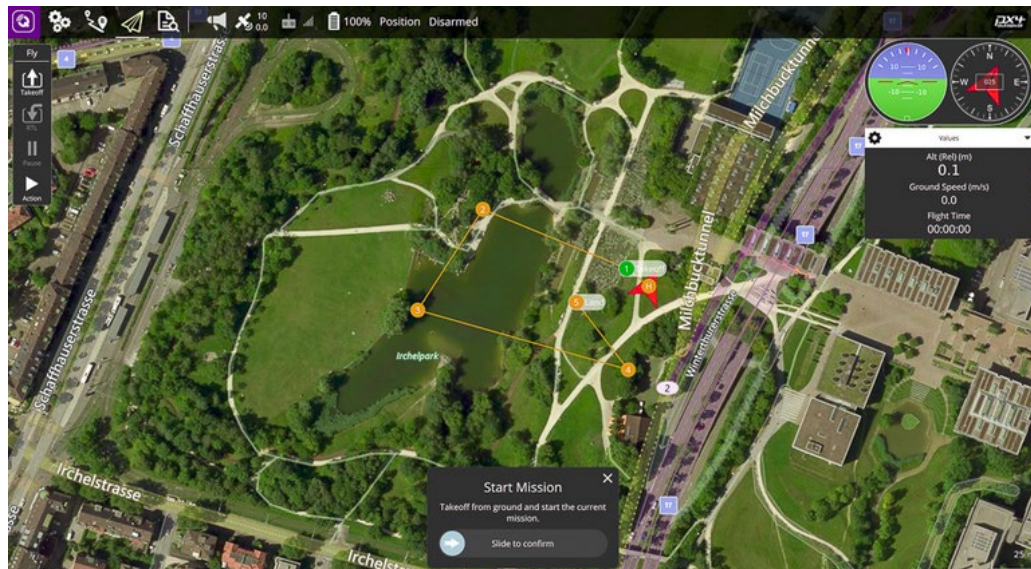


Figure 1 QGroundControl Screen

Based on the extensive research done on Pixhawk 4 open source hardware, we can conclude that this can be the best choice for our project as it is specially designed for autonomous flight. It also supports camera connectivity for object detection. Main drawback of Pixhawk 4 is that it is not generic platform meaning it is specifically designed to make drones and other flying vehicles autonomous. If we want to deploy it on RC car or other non-flying system, it will not work. But it is a best candidate for reference.

## MDEK 1001 PRODUCT AND DEVELOPMENT KIT

The MDEK 1001 development kit is used to deliver a flexible and compact real time location system) solution by providing the hardware platform, code and APIs and development environment using 12 pack DWM1001 modules. The DWM1001 module is based on ultra-wideband transceiver IC DW1000, which is based on IEEE 802.15.4 standard. It integrates UWB and Bluetooth to provide RTLS functionality and networking which are controlled using APIs. The firmware architecture packed in the DWM1001 module provides two functions: the PANS API and the PANS library. The PANS API includes the Generic API which includes different API sets for different peripheral interfaces which acts as a translator between the protocols like UART, SPI and BLE and the actual PANS library. Figure 2 shows the architecture of the DWM1001 firmware.

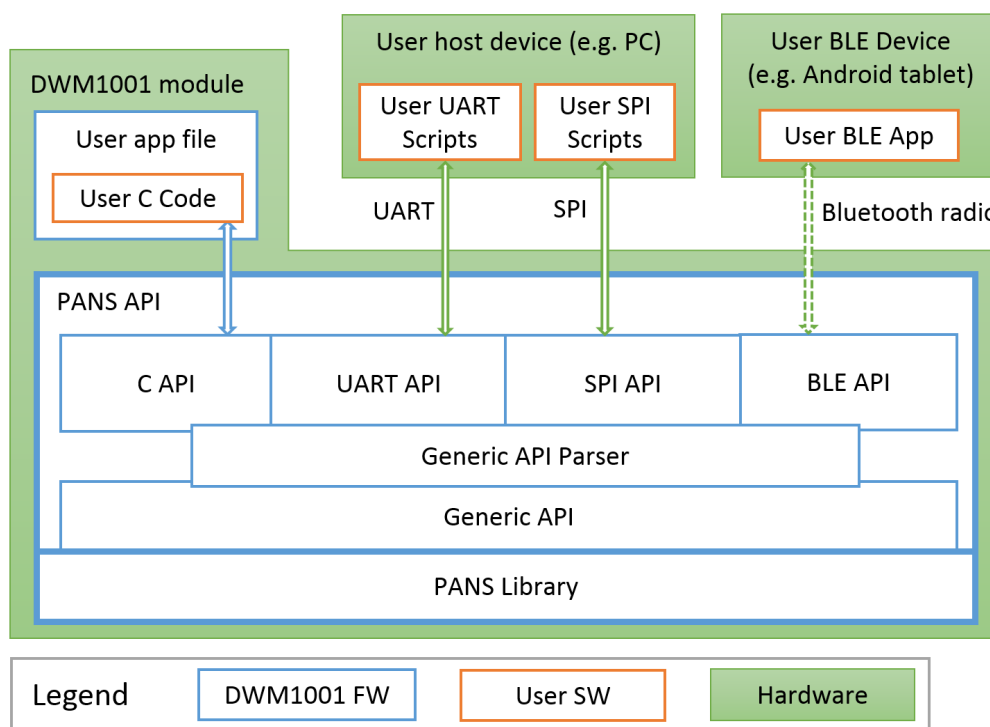
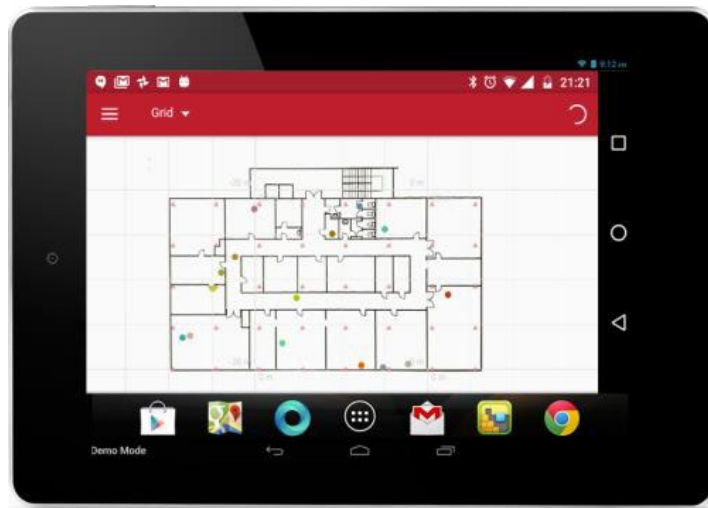


Figure 2 High-Level Architecture of DWM1001 Firmware

Few modules may act as anchors and one module can act as a tag which is attached along with a robotic system or a unit whose position needs to be monitored in the indoor conditions. The anchors and tag communicate with each other via UWB. There is an android application that controls and configures these sensors over BLE. This application shall be used to monitor the relative distance between the tag and the anchors to pin-point the exact coordinates of the tag.

Here is the example of the **evaluation steps** that was performed using the android application. Figure 3 shows the android device showing the location of tags and anchors



*Figure 3a Android application*

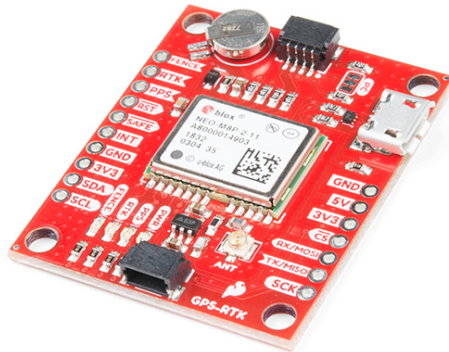
1. Mount at least 4 anchors on the wall as per the positioning mentioned in the instruction manual and assign remainder as the tags whose location needs to be monitored.
2. Start the device discovery and perform the configuration
3. Auto position these units and measure their distances with respect to the initiator.
4. Upload the floor plan, modify the units and change the refresh rate using the android app and repeat the above steps to measure the sensor performance.

#### **System performance:**

The system performance can be evaluated based on maximum tag location rate, X-Y location accuracy, point to point range, RTLS scheme range. The adaptive live location rate using motion sensor activity helps in increased battery life. The UWB Beacon can be used at the indoor conditions where most of the GPS fails to locate the robotic system.

### **RTK GPS**

- The RTK GPS will only be used when an autonomous system is to be deployed outdoors. For monitoring the position of Robotic system, we used GPS module to find the exact location of the system.
- GPS module comes in different forms, sizes and accuracy level. RTK GPS provides centimeter level accuracy for the robotic system. The GPS string coming from the UART port of GPS module is parsed using string functions on the STM32 low-level microcontroller.
- The parsed string contains latitude and longitude information to give the current position of the robotic system.
- This latitude and longitude information of the robotic system can be sent to the cloud by enabling IoT. We used AWS IoT platform for sending location information to the cloud.



*Figure 3b RTK GPS from Sparkfun website*

- AWS IoT provides FreeRTOS based software package to send sensor values such as location information of GPS sensor.
- The FreeRTOS platform allows multiple sensor readings to be uploaded to the cloud by managing many multi-threaded tasks. This allows easy integration of different sensors and software libraries to low level architecture.
- The GPS location obtained can be used to plot the location of Google Map giving the position of the robotic systems. When the destination of the path of travel is decided, the current location of robotic system can help to find the remaining distance of the robotic system from the destination.

## **IoT for data monitoring**

Since we are making the system connected, we will be uploading all the sensor data on the cloud. Any user with correct credentials would be able to monitor the system remotely. We are making use of proximity sensors, accelerometer, gyroscope, temperature, humidity and pressure sensors and eventually will be uploading this data on the cloud.

The proximity sensor will be mounted on the platform and will cover the entire 360 degrees to avoid any obstacles in its vicinity. The proximity sensor could be ultrasonic sensors which are able to detect the obstacles in the line of sight without any issues. The proximity sensors on STM32 were evaluated by writing its drivers on STM32 IoT Discovery Kit. The proximity sensors were only able to detect the obstacles within a range of 8 meters which is not sufficient if the system is moving at high speeds. The accuracy of the proximity sensor was  $\pm 2\%$  which is really good considering the small form factor of the sensor.

The accelerometer and gyroscope readings were also extracted from the onboard sensor. These sensor readings give X, Y and Z data. The X and Y readings are important if the system is maneuvering on the ground. The Z data is only useful if the system is on the slope or the system is deployed on a drone. The drivers are written for STM32 and the readings are extracted to make sense of how the system is moving in the real world so that we can get feedback and make necessary changes. The temperature, pressure and humidity sensor readings are added parameters that can be used to monitor the environment in which the autonomous system is deployed. The temperature reading kept on moving up as it caught the heat from the development board so we decided to make use of an external temperature module with a compensation circuit. The pressure and humidity sensor readings were tested in accordance with the readings from BME280 and the readings were in the range of 2% which is quite accurate.

To upload the sensor readings on the cloud we evaluated various LTE platforms such as SIM7600, SIM7000, BG96 and AT&T IoT Starter Kit. We make use of the MQTT protocol on the LTE module to communicate and publish sensor data on the cloud. We evaluated all the modules and selected SIM7600 because of its cost and convenience. The SIM7600 has an OpenLinux system but it does not support python. So we make use of AT commands over UART to communicate with the microcontroller. The MQTT protocol was configured on the LTE module and connection was established to the AWS cloud using IoT Core. The microcontroller was programmed to get all the sensor readings, connect to the MQTT using the AT commands and publish the values on the cloud using the AT publish command. These sensor values are then stored on a database and used to plot graphs and current location on the map for monitoring purpose.



## Chapter 7. System Design/Methodology

### PX4 Installation Guide:

PX4 code can be developed on Linux, MacOS and Windows, but it is recommended to use Linux. All target hardware including raspberry pi and Pixhawk 4 support linux. This guide is based on linux OS and target platform is Pixhawk 4.

Installation steps:

1. Download [ubuntu\\_sim\\_nuttx.sh](#)
2. Run above downloaded script in bash (Terminal in Linux).

**Command:** Source `ubuntu_sim_nuttx.sh`

3. Restart system.

PX4 Firmware is an open source and it is can be clone from GitHub repository [PX4/Firmware](#). It is recommended that first it should be fork and then clone it.

Building Firmware and upload:

1. Clone Firmware and go to **Firmware** directory.
2. Type command (for Pixhawk 4) : **make px4\_fmu-v5\_default**
3. On successful build below output will be generated.  
 - -Build files have been written to: /home/ikr/src/Firmware/build/px4\_fmu-v4\_default  
 [954/954]Creating/home/ik/src/Firmware/build/px4\_fmu-v4\_default/px4\_fmu-v4\_default.px4

Note: if error occurs try to update system using [sudo apt-get update].

4. To upload Firmware (Connect Pixhawk 4 to the system):

**Command:** `make px4_fmu-v5_default upload`

Writing first “Hello\_sky” program in PX4:

1. Create new directory - **Firmware/src/examples/px4\_hello\_sky.**
  2. Create new file - **px4\_hello\_sky.c**
- Code:** Below code is used for all codes available in Firmware repository.

```
#include <px4_log.h>
```

```
__EXPORT int px4_simple_app_main(int argc, char *argv[]);
```

```
int px4_hello_sky_main(int argc, char *argv[])
{
    PX4_INFO("Hello Sky!");
    return OK;
}
```

**Note:** main function should be name like this - **<module\_name>\_main**. In our case, it is **px4\_hello\_sky\_main**. **PX4\_INFO** is same as **printf**.

3. Create file name **CMakeLists.txt**. Copy below code in it.

```
px4_add_module(
    MODULE examples__px4_hello_sky
    MAIN px4_hello_sky
    STACK_MAIN 2000
    SRCS
        px4_hello_sky.c
    DEPENDS
)
```

4. Open cmake file which can be found in Firmware/boards/px4/fmu-v5/default.cmake and add following line somewhere in cmake file.

**examples/px4\_hello\_sky**

5. Build and upload as discussed above.

Connect Console:

1. Install screen: `sudo apt-get install screen`
2. Open the screen (Connect Pixhawk 4): `screen /dev/ttyXXX BAUDRATE 8N1`  
Hitting enter we can see: **nsh>**
3. Run application:  
**nsh> px4\_hello\_sky**

## MDEK 1001 PRODUCT AND DEVELOPMENT KIT:

The MDEK 1001 is a development kit for creating a small scale RTLS network with the help of 12 packed DWM 1001 UWB modules. Some of the main components of DWM1001 are shown in figure 4. DWM 1001 consists of few LEDs that indicate the communication status, battery status indication. LEDs D9-D12 are applicable when we use PANS firmware. The LED functionalities are mentioned in figure 5.

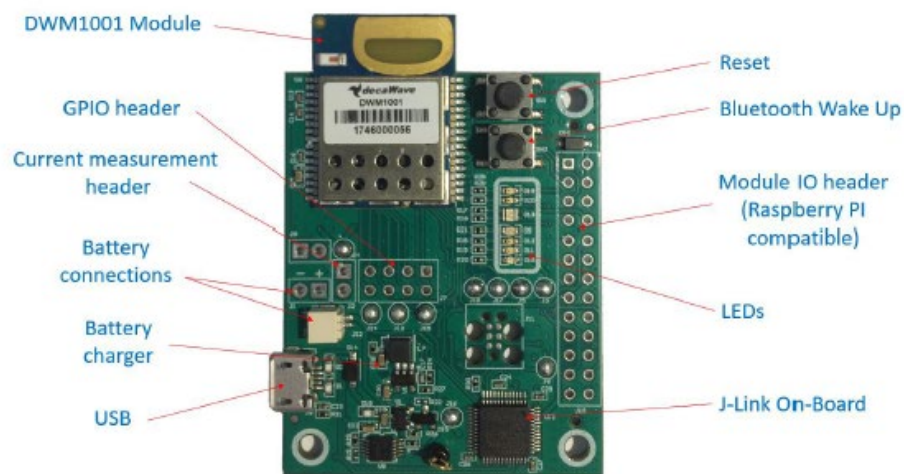


Figure 4 DWM1001 development board

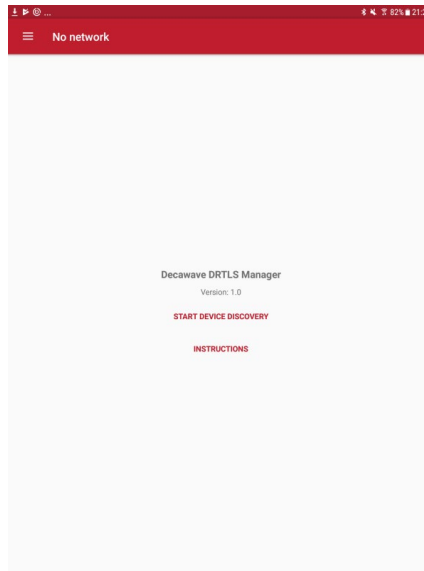


Figure 5 LED functionalities in DWM1001 development board

### Opening the Android application:

1. Download the Decawave RTLS Manager apk file from the decawave's website:  
<https://www.decawave.com/product/mdek1001-deployment-kit/>

2. Power on all the modules at their factory settings and start the device discovery as mentioned in the below screenshot:



*Figure 6 DECAWAVE DRTLS Manager APK Home screen*

3. The devices will be detected as tags and devices will be grouped as unassigned device, unassigned networks and networks as shown below. Select the devices and provide a network name and assign all the unassigned devices to these networks:

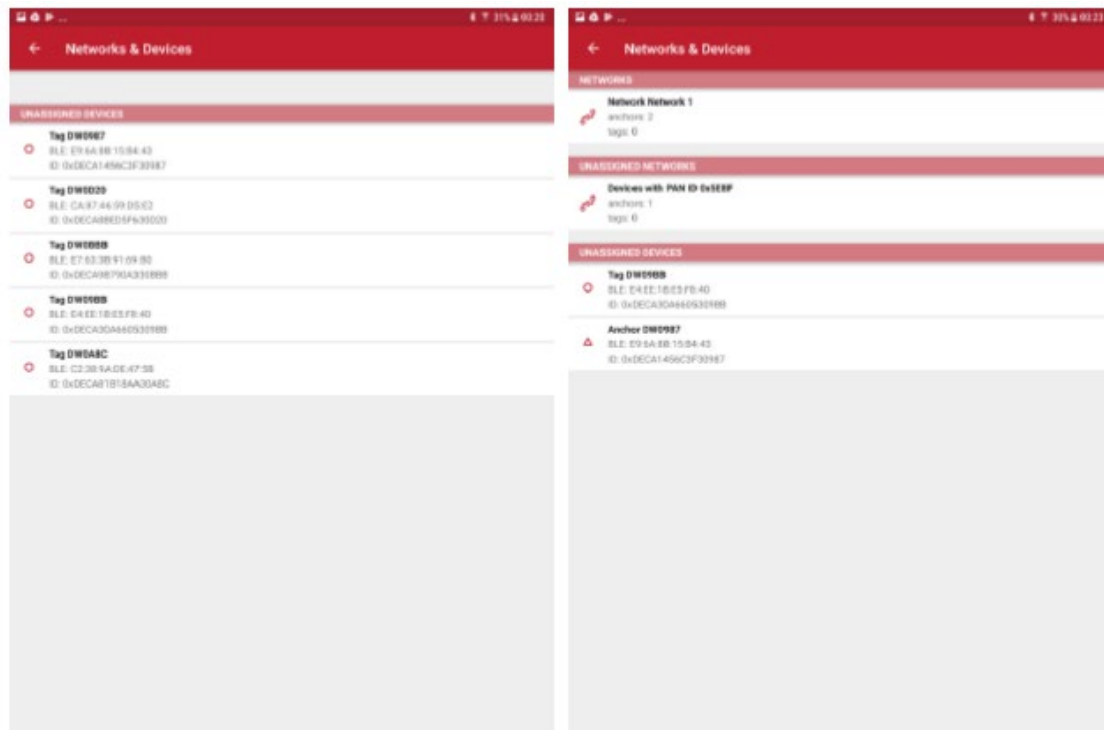


Figure 7 Devices detected as unassigned devices and unassigned networks

### Configuring an anchor:

1. Select some of the RTLS modules as anchors (minimum 4) and power them using the USB based power supply.
2. Mount these anchors into the wall at the same level but higher than the tags in order to ensure maximum coverage. Ensure the UWB is active.
3. The configuration of the anchor needs to be done as mentioned in figure 8 by clicking on the edit section. Select one of the anchors as an initiator and keep the value of x,y,z as (0,0,0).

### Configuring a tag:

1. Select one of the RTLS modules as tag and power them using the USB based power supply. You can attach it with the unit whose location needs to be monitored.
2. The configuration of the tag needs to be done as mentioned in figure 8 by clicking the edit button. Make sure the UWB and BLE is active.

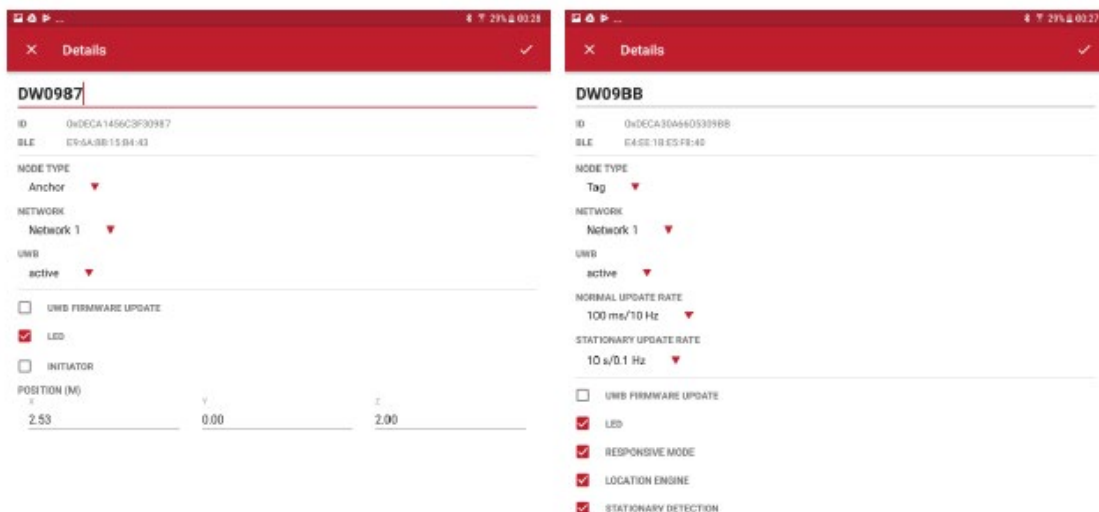


Figure 8 Configuring tags and anchors

### Auto positioning:

The auto positioning feature is used to automatically measure the distances of the anchors with respect to its initiator and set up an RTLS network. This may result in a small amount of error, but it is recommended that the line of sight between the anchors is maintained. Auto positioning can be selected as explained in figure 9. Once all the anchors are present in the list, reorder them from top to bottom and start measuring the distance by clicking on MEASURE. The auto positioning rules are explained in figure 10.

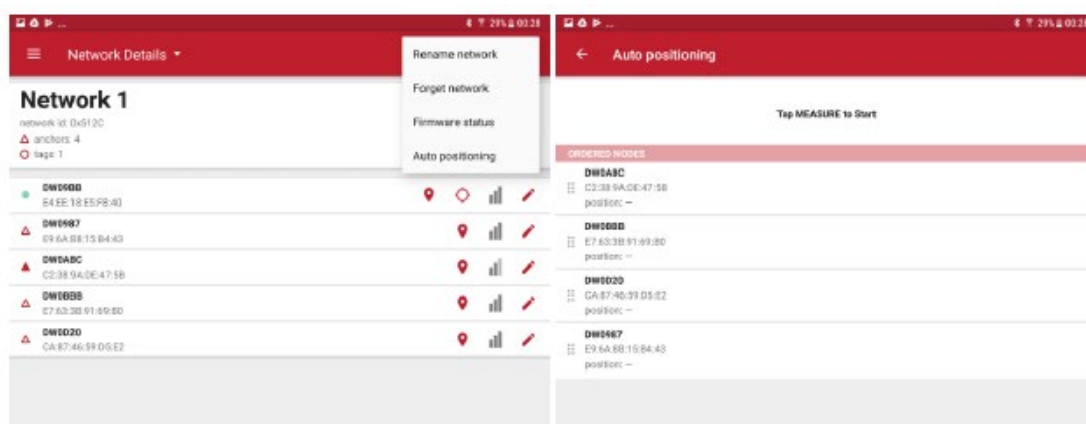


Figure 9 Entering the auto positioning mode

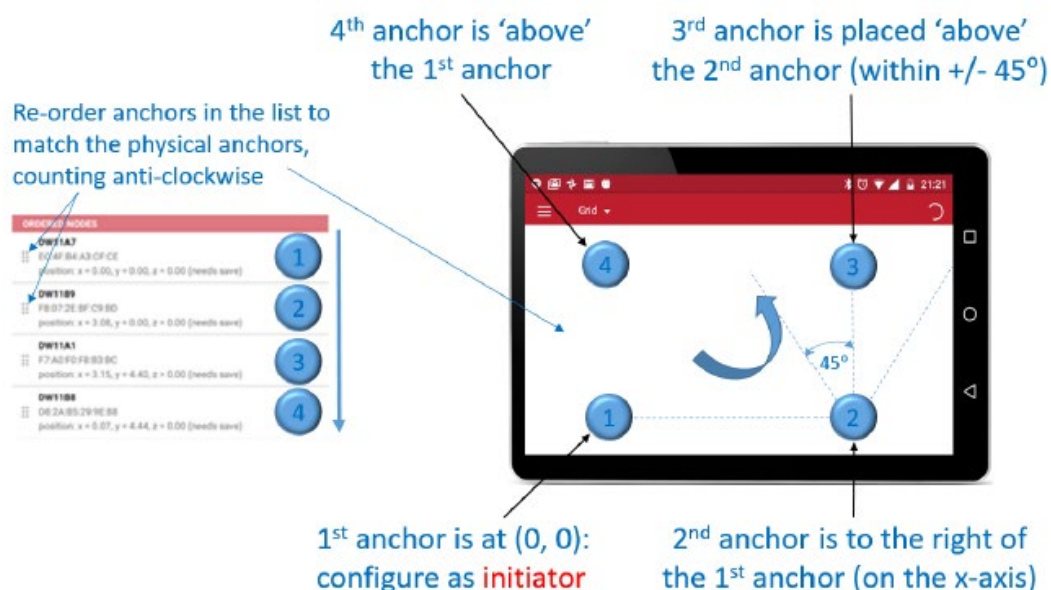


Figure 10 Auto positioning rules

Ranging starts in the auto positioning mode once all the anchors and tags are in the network and in the network settings, tap on the “GRID” option and see the dynamic update of the tag distance with respect to anchors and initiator. Figure 11 shows the demonstration of the RTLS network.

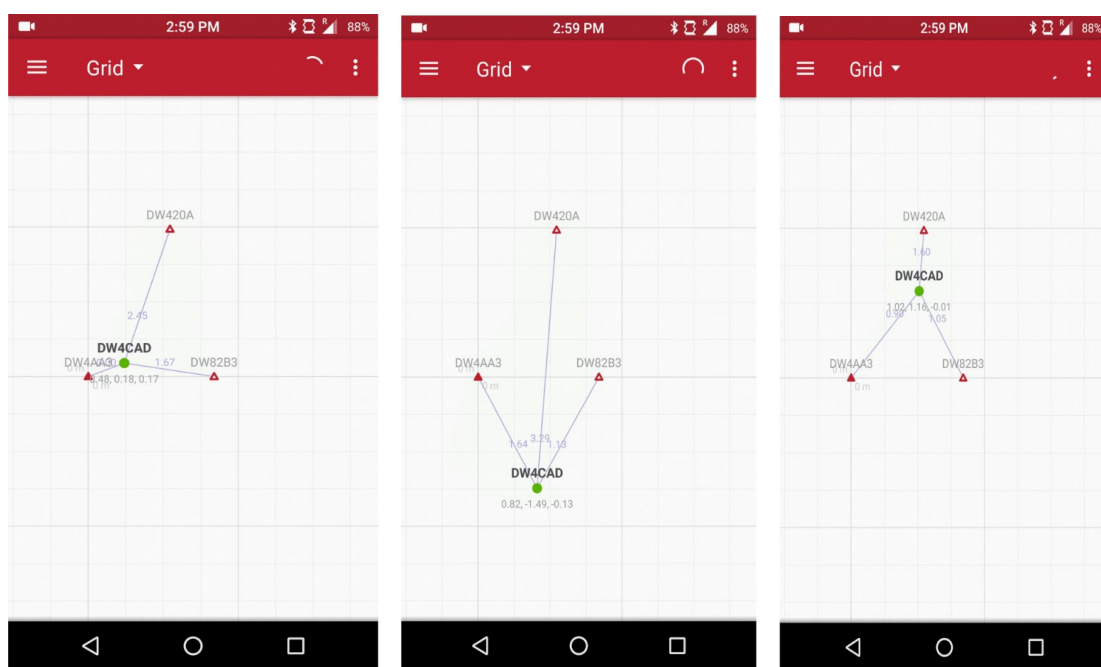


Figure 11 RTLS network demonstration

## RTK GPS

When GPS is connected to the UART port of the microcontroller we receive GPS NEMA strings which can be parsed to obtain latitude and longitude information.

```
$GNRMC,080154.00,A,3719.58030,N,12152.58517,W,0.007,,180419,,,A*7F
$GNVTG,,T,,M,0.007,N,0.013,K,A*38
$GNGGA,080154.00,3719.58030,N,12152.58517,W,1,08,1.28,48.8,M,-29.9,M,,*49
$GNGSA,A,3,10,27,21,,,,,,,,,2.51,1.28,2.16*13
$GNGSA,A,3,87,65,66,88,81,,,,,,,,,2.51,1.28,2.16*11
$GPGSV,1,1,04,08,30,308,09,10,69,344,25,21,45,116,17,27,52,271,22*7A
$GLGSV,2,1,08,65,68,354,32,66,34,242,27,72,27,036,,73,01,030,*65
$GLGSV,2,2,08,74,00,071,,81,27,329,18,87,50,154,34,88,78,319,21*6F
$GNLL,3719.58030,N,12152.58517,W,080154.00,A,A*64
$GNRMC,080155.00,A,3719.58026,N,12152.58516,W,0.068,,180419,,,A*71
$GNVTG,,T,,M,0.068,N,0.126,K,A*36
$GNGGA,080155.00,3719.58026,N,12152.58516,W,1,08,1.28,49.0,M,-29.9,M,,*47
$GNGSA,A,3,10,27,21,,,,,,,,,2.51,1.28,2.16*13
$GNGSA,A,3,87,65,66,88,81,,,,,,,,,2.51,1.28,2.16*11
$GPGSV,1,1,04,08,30,308,,10,69,344,25,21,45,116,16,27,52,271,21*71
$GLGSV,2,1,08,65,68,354,32,66,34,242,27,72,27,036,,73,01,030,*65
$GLGSV,2,2,08,74,00,071,,81,27,329,19,87,50,154,33,88,78,319,21*69
$GNLL,3719.58026,N,12152.58516,W,080155.00,A,A*63
$GNRMC,080156.00,A,3719.58020,N,12152.58517,W,0.064,,180419,,,A*79
$GNVTG,,T,,M,0.064,N,0.119,K,A*36
$GNGGA,080156.00,3719.58020,N,12152.58517,W,1,08,1.28,49.2,M,-29.9,M,,*41
```

*Figure 12 GPS Raw data*

On parsing the above string we get:

Latitude : 37.31 degree

Longitude: -121.866

The parsing function for GPS string is as follows:

```
int gps_parse_string(char *raw_gps_str, gps_parsed_s *gps_parsed)

int parse_success = 1;

char valid_invalid[2] = {0};
char dummy[10];
char south_or_north[2] = {0};
char east_or_west[2] = {0};
float longitude = 0;
float latitude = 0;
```



```

//If first gps string sequence is equal to $GPRMC
char gps_type[7] = {0};
strncpy(gps_type, raw_gps_str, 6);
if (!strcmp(gps_type, "$GPRMC"))
{
    sscanf(raw_gps_str, "$GPRMC,%[^,],%2[^,],%f,%2[^,],%f,
2[^,]",
           dummy, valid_invalid,
           &latitude, south_or_north, &longitude, east_or_west);

    strcpy(gps_parsed->valid_invalid, valid_invalid);
    gps_parsed->latitude = latitude;
    gps_parsed->longitude = longitude;
    strcpy(gps_parsed->south_or_north, south_or_north);
    strcpy(gps_parsed->east_or_west, east_or_west);
}
else
{
    parse_success = false;
}

return parse_success;

static void gps__process_Lat_Long(float *Lat, float *Long)

//Latitude format is in ddss.sssss
int dd = (int)(*Lat / 100.00); //4717.112671 dd = 47
float ss = *Lat - (float)(dd * 100);
*Lat = dd + ss / 60.0; //Lat = 47.28 --47 deg and 0.28 hr

//longitude format is in dddss.sssss
int ddd = (int)(*Long / 100); //00833.914843 dd = 8
ss = *Long - (float)(ddd * 100); //ss = 33.914..
*Long = ddd + ss / 60.0; //Long 8.56

void gps__get_Lat_Long(gps__parsed_s *gps_parsed, gps__position_s
location)

    gps__process_Lat_Long(&(gps_parsed->latitude), &(gps_parsed->
longitude));

```

```

location->latitude = gps_parsed->latitude;
location->longitude = gps_parsed->longitude;

//If direction is South/West co-ordinate is -ve
if (!strcmp(gps_parsed->south_or_north, "S"))
    location->latitude = -1 * location->latitude;

if (!strcmp(gps_parsed->east_or_west, "W"))
    location->longitude = -1 * location->longitude;

```

*Figure 13 RTK GPS source code*

Pseudo Code / Explanation of Code:

1. `gps__parse_string()`

This function receives raw GPS string such as

“\$GNRMC,080154.00,A,3719.58030,N,12152.58517,W,0.007,,180419,,,A\*7F” and it parses it using string functions and sscanf functions.

Thus, it separates various gps information such as latitude, longitude, altitude, distance, time, no of satellite.

2. `gps__process_Lat_Long()`

These functions is passed latitude and longitude as input in raw format. It converts this values in the lat and long with respect to degree and minute format, useful for plotting the coordinates on Google Maps.

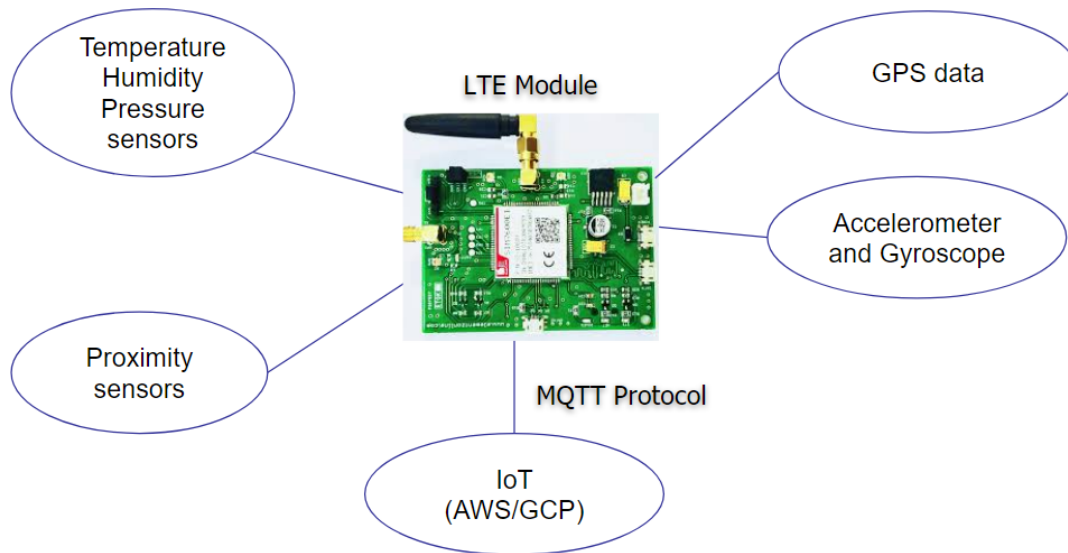
3. `gps__get_Lat_Long()`

This function sends latitude and longitude values to the caller function after the values are parsed in real-time. Considering, the direction of GPS coordinates the location information is multiplied by 1 or -1 indicated by north of southern direction.

Now we Send this location to AWS IoT Cloud using MQTT protocol. This location can be added to Google Maps API on an android app or a JavaScript based webpage to update the location of robotic system in real-time.

## IoT for data monitoring

The architecture of the IoT system is as follows:



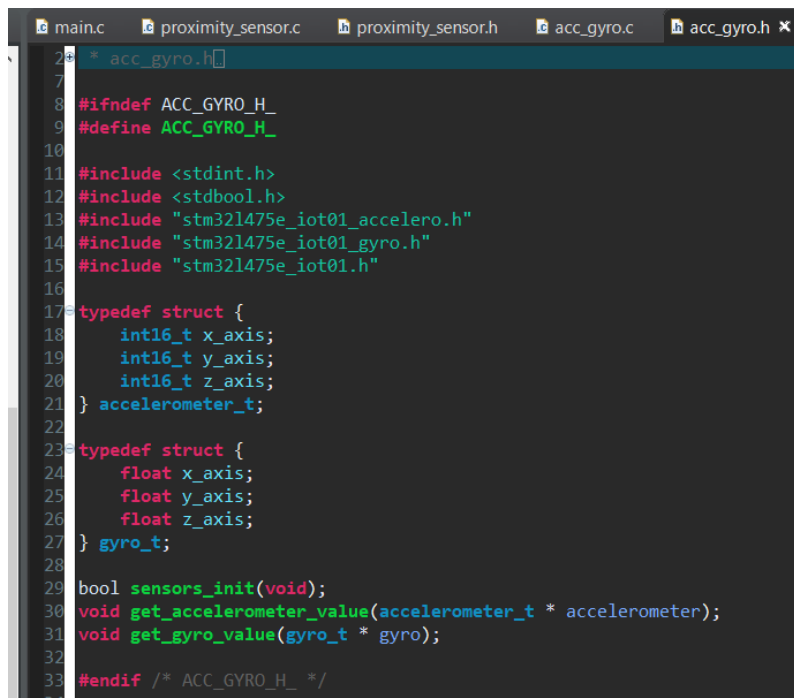
1. The proximity sensor drivers were developed, and API's were used to get the data easily. The header file for the same is as follows:

```

7
8 #ifndef PROXIMITY_SENSOR_H_
9 #define PROXIMITY_SENSOR_H_
10
11 #include <stdint.h>
12 #include <stdbool.h>
13
14 #include "vl53l0x/vl53l0x_def.h"
15 #include "vl53l0x/vl53l0x_api.h"
16 #include "vl53l0x/vl53l0x_tof.h"
17
18 void VL53L0X_PROXIMITY_MspInit(void);
19 uint16_t VL53L0X_PROXIMITY_GetDistance(void);
20 void VL53L0X_PROXIMITY_Init(void);
21
22 bool sensors_init(void);
23 void proximity_init(void);
24 uint16_t get_proximity_mm(void);
25 void get_proximity_cm(uint16_t *prox_cm);
26
27 #endif /* PROXIMITY_SENSOR_H_ */
28

```

- a. The proximity value can be accessed in mm and cm as we can see in the pseudo-code.'
  - b. The maximum value for the proximity sensor is 8 meters.
  - c. We do not need any calibration before using the sensor.\
2. The accelerometer and gyroscope readings were obtained with the following code:



```

24 * acc_gyro.h
7
8 #ifndef ACC_GYRO_H_
9 #define ACC_GYRO_H_
10
11 #include <stdint.h>
12 #include <stdbool.h>
13 #include "stm32l475e_iot01_accelero.h"
14 #include "stm32l475e_iot01_gyro.h"
15 #include "stm32l475e_iot01.h"
16
17 typedef struct {
18     int16_t x_axis;
19     int16_t y_axis;
20     int16_t z_axis;
21 } accelerometer_t;
22
23 typedef struct {
24     float x_axis;
25     float y_axis;
26     float z_axis;
27 } gyro_t;
28
29 bool sensors_init(void);
30 void get_accelerometer_value(accelerometer_t * accelerometer);
31 void get_gyro_value(gyro_t * gyro);
32
33 #endif /* ACC_GYRO_H_ */

```

- a. We need to initialize the sensors before reading the data.
  - b. The X, Y and Z readings are obtained using the following APIs.
3. The temperature, humidity and pressure readings can be obtained using BSP drivers which are part of STM code.
 

```

*TEMPERATURE_Value = BSP_TSENSOR_ReadTemp();
*HUMIDITY_Value = BSP_HSENSOR_ReadHumidity();
*PRESSURE_Value = BSP_PSENSOR_ReadPressure();

```
  4. The IoT module and the IoT discovery kit is connected using simple UART connections and providing supply and ground pins. The pin connection diagram is as follows:

STM32 IoT Discovery Kit	SIM7600
D0 (RX)	TXD
D1 (TX)	RXD
5V	5v
GND	GND
PWR	GPIO

- We can test if the module is working using the command “AT” and if we get “OK” back then we know that the device is working properly.
- The PWR is connected to a GPIO pin and it is usually high. A small high-to-low pulse will turn on the LTE module if it is in the OFF state.
- We will send a set of the following AT commands to turn on the MQTT and start publishing the data on the cloud.

```

AT+CCERTDOWN="clientcert.pem",1224
AT+CCERTDOWN="clientkey.pem",1679
AT+CCERTDOWN="cacert.pem",1208

AT+CSSSLCFG="sslversion",0,4
AT+CSSSLCFG="authmode",0,2
AT+CSSSLCFG="cacert",0,"cacert.pem"
AT+CSSSLCFG="clientcert",0,"clientcert.pem"
AT+CSSSLCFG="clientkey",0,"clientkey.pem"

AT+CMQTTSTART
AT+CMQTTACCQ=0,"SIMCom_7600_0",1
AT+CMQTTSSSLCFG=0,0

AT+CMQTTCONNECT=0,"tcp://a5f0mwk8yzpi0-ats.iot.us-east-2.amazonaws.com:8883",60,1

AT+CMQTTTOPIC=0,18
test_SIMCom_7600_0
AT+CMQTTPAYLOAD=0,21
Hello, this is a test
AT+CMQTTPUB=0,1,60

```

- In the AWS console, we need to subscribe to the topic so that we can listen to the messages sent by the LTE module.

## Chapter 8. Implementation Plan and Progress

1. The programming for all the sensors was done using STM32 Workbench which is an eclipse IDE modified for using it with STM products. The code with all the necessary pins initialized was developed using STM Cube MX.
2. Another important setup that needs to be done is the TeraTerm serial terminal which is used for monitoring the output of the sensor values.
3. The development kit STM32 IoT Discovery Kit was acquired and the sensors were already mounted on board.
4. We already had the external temperature sensor and BME280 and we used them as a reference for the on-board sensors on the STM32 development board.
5. All the sensor data i.e. temperature, humidity, pressure, proximity, accelerometer and gyroscope readings are being obtained properly:

```
Lat:0.000000 Long:-0.000000 ppb:1241 rh:23 Temp:26.422651 P
es:1018.859985 Hum:28.936838 AccX:19 AccY:3 AccZ:992 GyX:14
.000000 GyY:-1750.000000 GyZ:0.000000
```

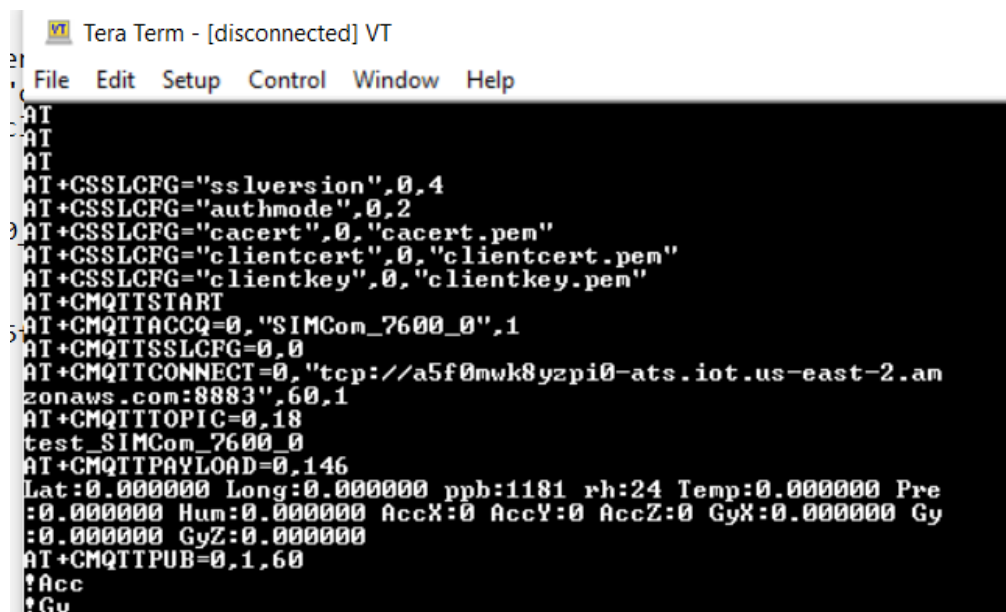
Figure 14. Temperature, Pressure, Humidity, Accelerometer, Gyroscope readings

```
STOP Obstacle detected at 127
STOP Obstacle detected at 89
STOP Obstacle detected at 85
STOP Obstacle detected at 80
STOP Obstacle detected at 79
STOP Obstacle detected at 84
STOP Obstacle detected at 87
STOP Obstacle detected at 83
STOP Obstacle detected at 84
STOP Obstacle detected at 86
STOP Obstacle detected at 85
STOP Obstacle detected at 85
```

Figure 15. Proximity readings

6. The next module that was acquired was the SIM7600 LTE module. Installed the drivers and updated the firmware on the board to ensure proper functioning.

7. Used the MQTT documentation provided by Waveshare for evaluating the module. After the successful implementation of the example, we set up the AWS IoT Core and added the IoT device.
8. Using the AT commands successfully published the data on the AWS cloud using the example and MQTT protocol with the certificate being added to the LTE module.
9. Connected the LTE module and IoT Discovery Kit and implemented a function which grabs the sensor readings and sends it on the AWS cloud.



```

Tera Term - [disconnected] VT
File Edit Setup Control Window Help
AT
AT
AT
AT+CSSLCFG="sslversion",0,4
AT+CSSLCFG="authmode",0,2
AT+CSSLCFG="cacert",0,"cacert.pem"
AT+CSSLCFG="clientcert",0,"clientcert.pem"
AT+CSSLCFG="clientkey",0,"clientkey.pem"
AT+CMQTTSTART
AT+CMQTTACCQ=0,"SIMCom_7600_0",1
AT+CMQTTSSLCFG=0,0
AT+CMQTTCONNECT=0,"tcp://a5f0mwk8yzpi0-ats.iot.us-east-2.amazonaws.com:8883",60,1
AT+CMQTTTOPIC=0,18
test_SIMCom_7600_0
AT+CMQTTPAYLOAD=0,146
Lat:0.000000 Long:0.000000 ppb:1181 rh:24 Temp:0.000000 Pre
:0.000000 Hum:0.000000 AccX:0 AccY:0 AccZ:0 GyX:0.000000 Gy
:0.000000 GyZ:0.000000
AT+CMQTTPUB=0,1,60
!Acc
!Gy
  
```

*Figure 16 Execution of AT commands for publishing the sensor values on cloud*

10. The functionality of ultra-wide band beacon DWM1001 needed to be tested. After studying datasheet, I explored various tutorials online to setup the MDEK1001. The firmware guide and quick start guide in the website were enough to operate this sensor.
11. The quick start guide explains to setup the sensors, configuring the parameters and positioning for correct operation. These sensors were controlled via bluetooth and an Android App named Decawave RTLS Manager.
12. One sensor was configured as tag and remaining sensors were configured as anchor and I was able to measure the relative distances between sensors dynamically via app using auto positioning feature.

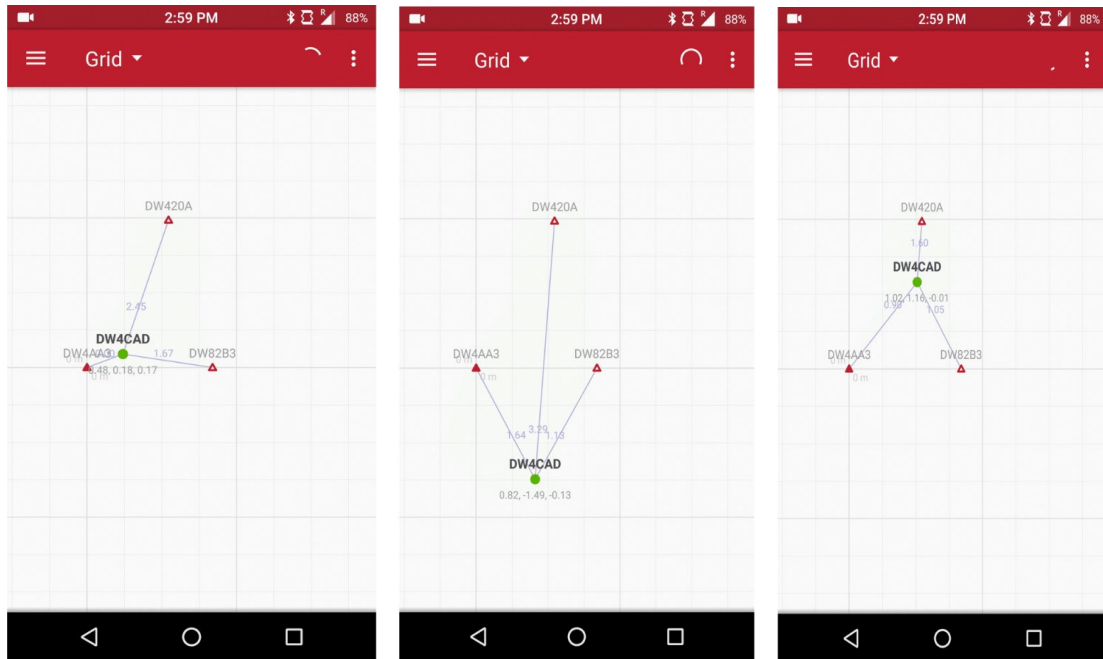


Figure 17 DWM1001 Demo

### 13. GPS string from RTK GPS is parsed using the U-Center software

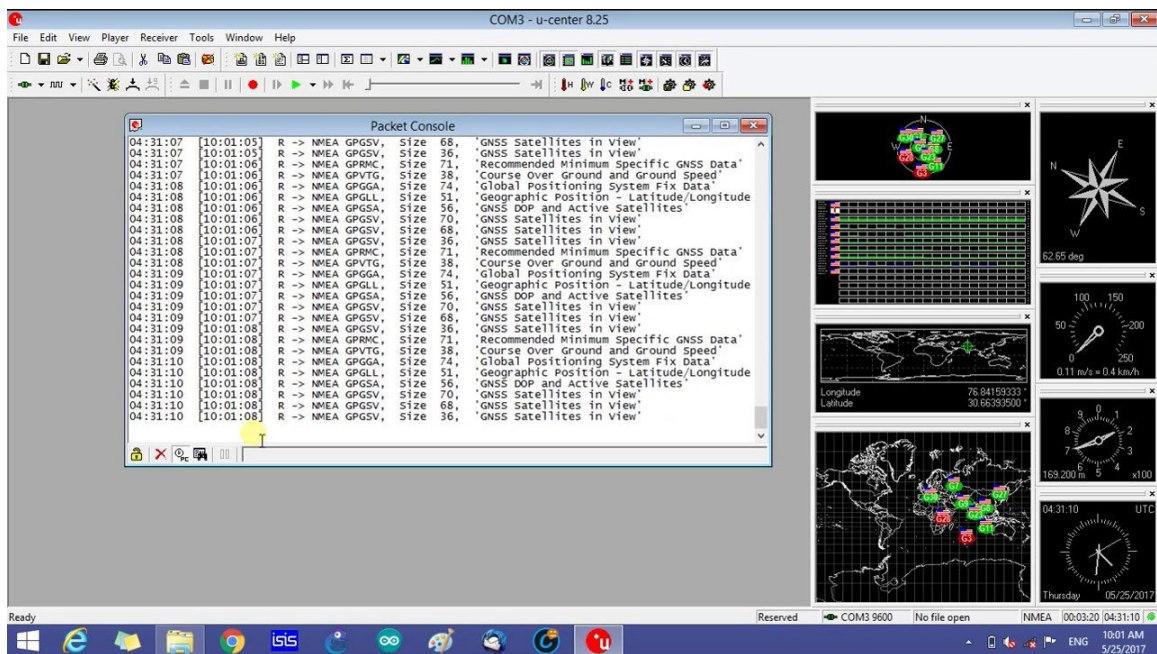
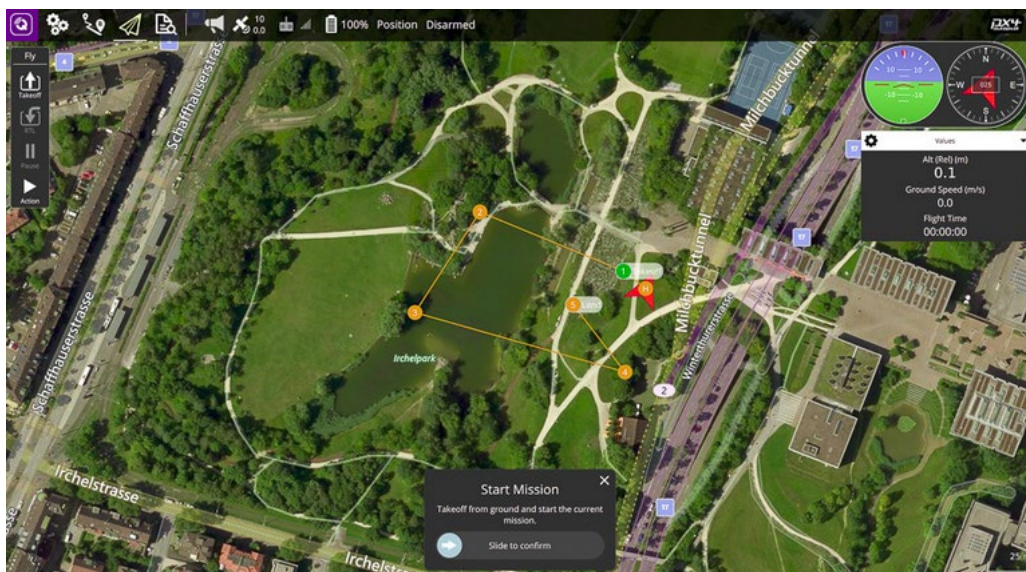


Figure 18 RTK GPS output



14. The parsed GPS string contains latitude and longitude is sent to the AWS IoT using STM32 microcontroller. The GPS location are then used to plot the position of robotic system on Google Map.
15. Evaluated the Pixhawk platform in detail by studying its hardware requirements, performance, missing aspects, portability towards ground autonomous systems.
16. Installed the Pixhawk 4 OS which is used for drone control on the PX4 development and successfully flashed it.
17. Executed an example program and evaluated the platform.
18. Tested all the API's with the available hardware such as ublox GPS.
19. For implementation of drones as autonomous system implemented QGC which acts as a base station and communicates with the drone. Implemented it on an actual drone.
20. Evaluated the mission control by programming the mission details on the drone and tested it on field.
21. Also tested the support for camera, which is used to implement obstacle detection when the drone is in autonomous mode.



*Figure 19 Flight mission control input*

## Chapter 9. Project Schedule

Sr. No.	Description	Team members (2 member team for each task)
1.	Finalize the details of the low-level and high-level architecture with the advisor.	Low-level architecture research – Jay and Saumil  High-Level architecture research – Vatsal and Prashant
2.	Get started with the development environment and familiarize ourselves with hardware for low level and high-level architecture.	Setup the environment for STM32 and research about different sensors available in the market – Jay, and Saumil  High-Level research: motion planning– Vatsal and Prashant
2.	Add FreeRTOS for the STM32 development board along with peripheral and software libraries to be used for Low-level architecture platform.	Acquire hardware and run all the software examples for sensor modules on STM32 board – Jay and Saumil  High-Level research: path planning– Vatsal and Prashant
4.	Interface sensors such as accelerometer, gyroscope, GPS, Ultra-wideband beacon, LTE module, IMU sensors.	Develop drivers for GPS (parsing function), telemetry sensors and beacon – Jay and Saumil  Develop drivers for Radar, ultrasonic sensor and IMU sensors (motion data capture)
5.	Create Separate tasks for each sensor and synchronize them to manage their data collection forming a reliable sensor network.	Collaborate the code and make the code work after collaborating effectively without any issues – All members
6.	As per the project requirement create PCBs and hardware enclosures for the low-level hardware architecture.	Collaborate and integrate all the sensors – Jay and Saumil  Work on PCB, finalize it and collaborate for integration of sensors –

		Vatsal and Prashant
7.	Connect the low-level hardware system to the cloud enabling the Internet of Things by using services such as AWS IoT or Google Cloud to monitor and analyze the sensor data.	<p>Publish all the sensor data on the cloud – Vatsal and Saumil</p> <p>Assemble the hardware and check the integration of hardware and sensors – Jay and Prashant</p>
8.	Create an Ubuntu Image for Nvidia Jetson TX2 board and load the OS. Familiarize with the functioning and usage of Nvidia Jetson TX2.	<p>Check the Camera on Jetson with the sample codes – Vatsal and Jay</p> <p>Check the LIDAR on Jetson with the sample codes – Saumil and Prashant</p>
9.	Port ROS on Nvidia Jetson TX2 high-level platform and accompanying packages required for the creation of a Robotic System.	Understand ROS and explore the API's from ROS on Jetson – All team members
10.	Interface Camera and Lidar to Nvidia Jetson TX2 and receive data from them in a successful manner.	All team members
11.	Create 3D Maps for localization and deep learning applications for object detection, obstacle avoidance, and image segmentation.	<p>Work on camera and object detection and deep learning models – Vatsal and Saumil</p> <p>Work on LIDAR for obstacle detection and develop a mechanism to create a signal for</p>
12.	Deploy the low-level system and high-level system in a Robotic platform, Drone or autonomous vehicle to test and evaluate the performance of the system.	Work on path planning and motion planning and integrate the entire low level and high-level part of the project to make it as a complete package – All team members



[illegible][illegible]