# Drone Using Navio2 and Raspberry Pi along with Amazon AWS IoT

*Faizan Mansuri (98753163)*
*Department of Electrical and Computer Engineering*
*University of British Columbia*
*Vancouver, Canada*

## *Introduction*

The objective of this project was to develop a base infrastructure for operation of Drones as IoT device with the use of Raspberry Pi along with Amazon AWS Services. This project would provide the baseline for all future drone swarm developments along with adding flexible controls of the drone and its peripheral sensors using a HTML based frontend with secure authentication. All the project code can be accessed at https://github.com/mbiuki/minicloud/tree/drone_AWS

Architecture used in this project is shown below:

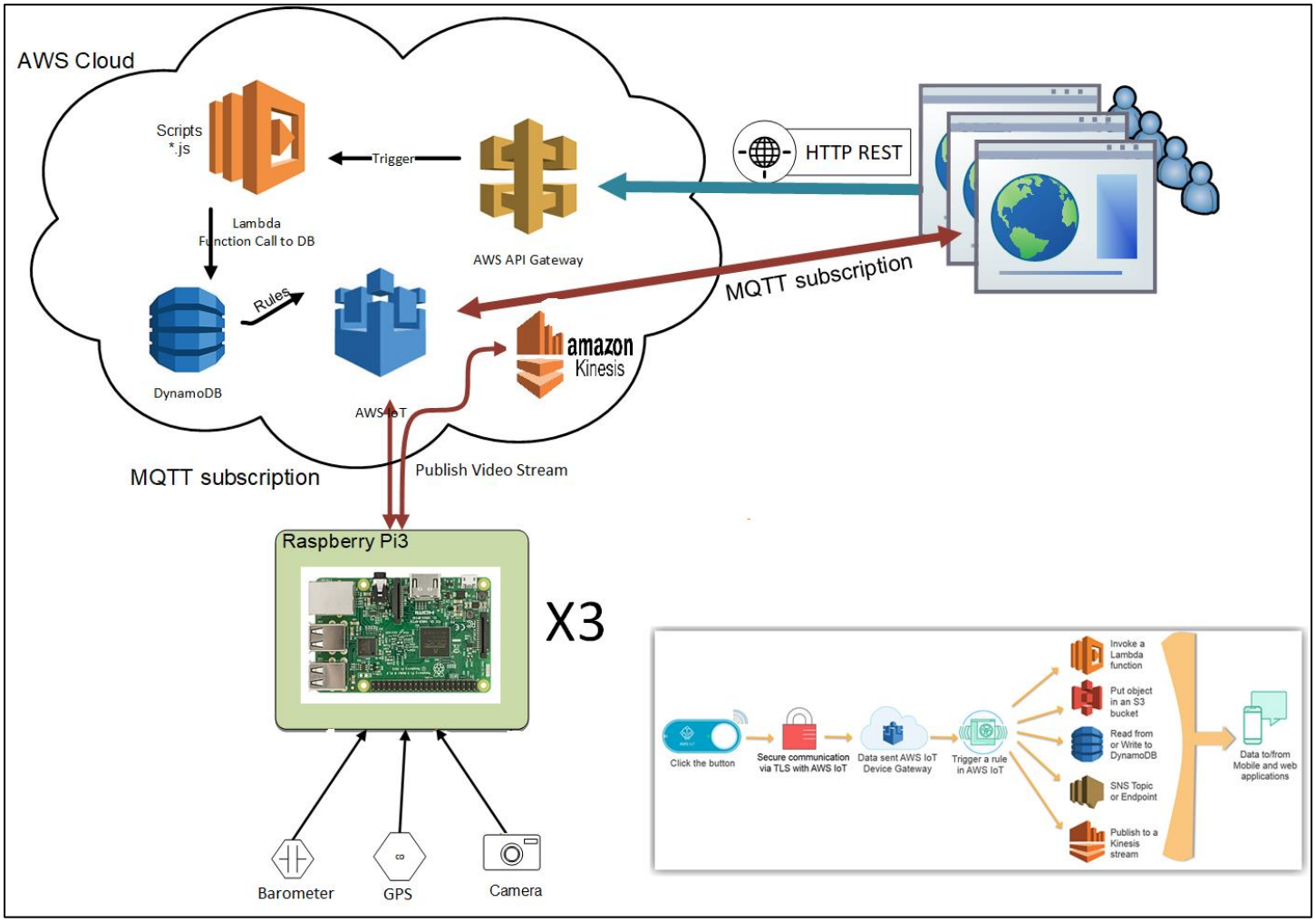



Figure: AWS Drone IoT Architecture

This project includes following tasks:

1) Stream real-time video from Drone to Front-End Website.
2) Publish Drone Sensor Data.
3) Design a front-end for collecting Drone data as well as Plot the respective sensor data charts.

## *Drone Hardware Specifications*

This project utilizes two primary control boards, namely, ***Raspberry Pi 3B+*** and ***Navio2 Emlid Board*** as Ardupilot Flight Controller. Navio2 is a flight controller hat designed for Raspberry Pi. The specifications of a Navio2 Board are listed below:

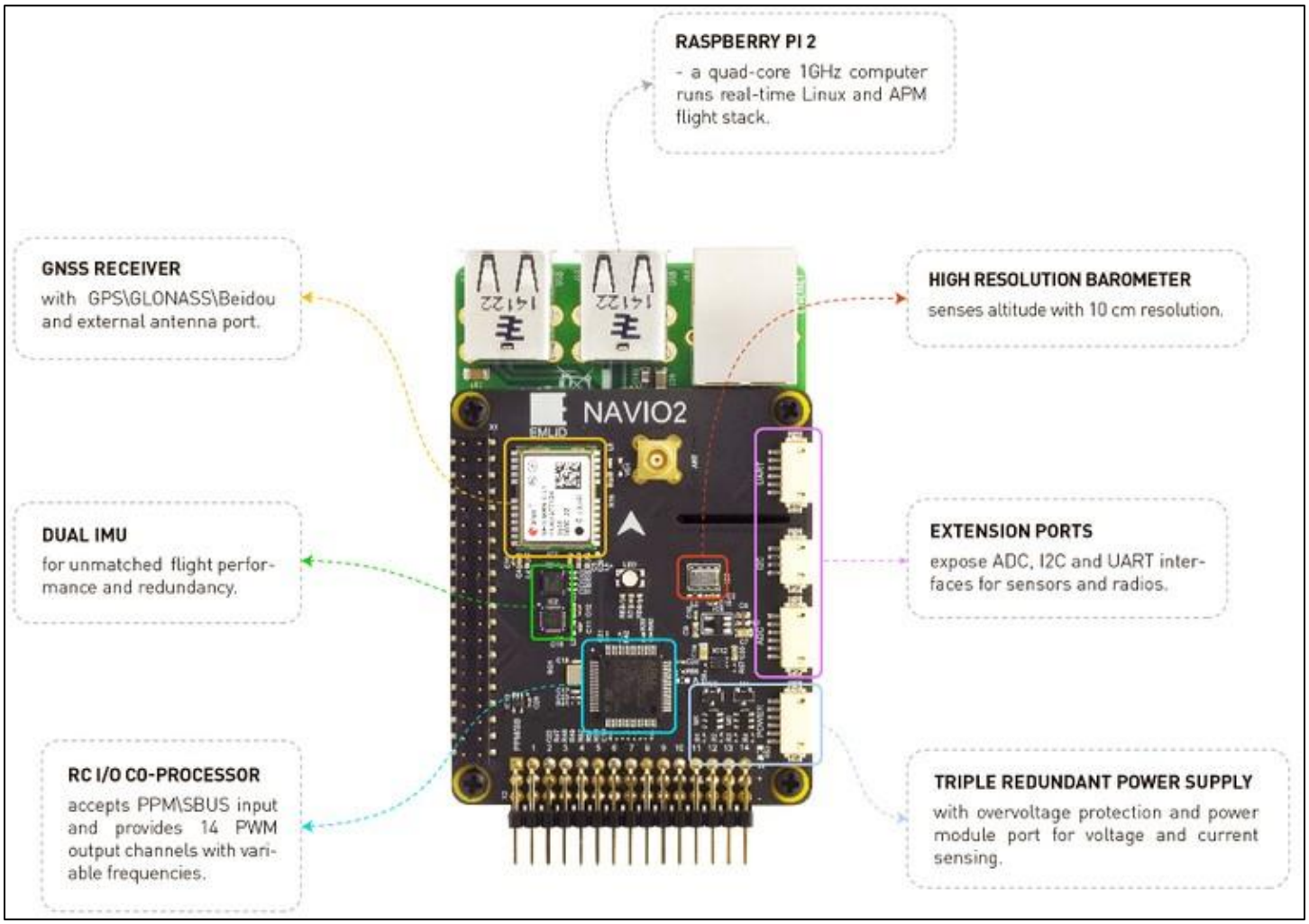



Figure: Navio2 + Raspberry Pi

- *Processor (Raspberry PI 3)*
  - 1.2GHz 64-bit quad-core ARMv8 CPU
  - 1GB RAM

- *Sensors*
  - MPU9250 9DOF IMU
  - LSM9DS1 9DOF IMU
  - MS5611 Barometer
  - U-blox M8N Glonass/GPS/Beidou
  - RC I/O co-processor
- *Power*
  - Triple redundant power supply
- *Interfaces*
  - UART, I2C, ADC for extensions
  - PWM/S.Bus input
  - 14 PWM servo outputs

The Project uses Raspberry Pi Camera Module V2 for video streaming. PiCamera is equipped with Sony IMX219 8 MP image sensor, has a maximum photo resolution of 3280 x 2464 pixels and supports Video Resolutions of 1080p30, 720p60 and 640x480p90.

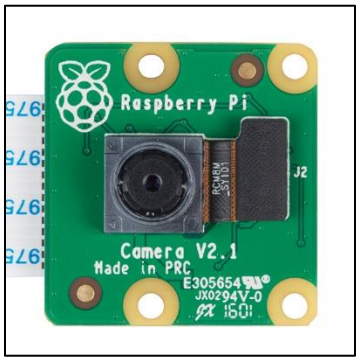
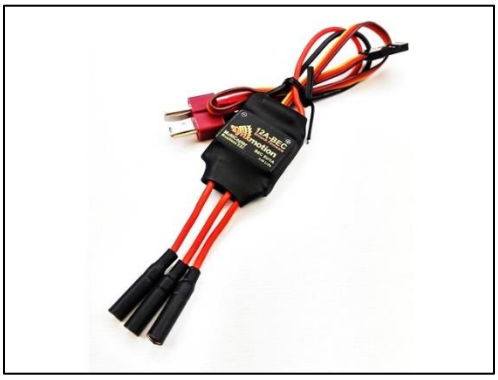



Figure: Raspberry Pi Camera V2 Module

The drone assembly is Crazy2Fly from Lynxmotion. It provides a durable 3D printed drone assembly along with ESCs, motors and Propellers.

Drone Assembly

15A / 1000Kv brushless Motor



12A Electronic Speed Controllers

Drone is equipped with Telemetry modules for communication with Ground Station. The telemetry module used is a 915 MHz operated module which is connected using UART Protocol with Navio2 board. This module helps the drone to connect to Ground Control Station, namely, Mission Planner to obtain the drone parameters real-

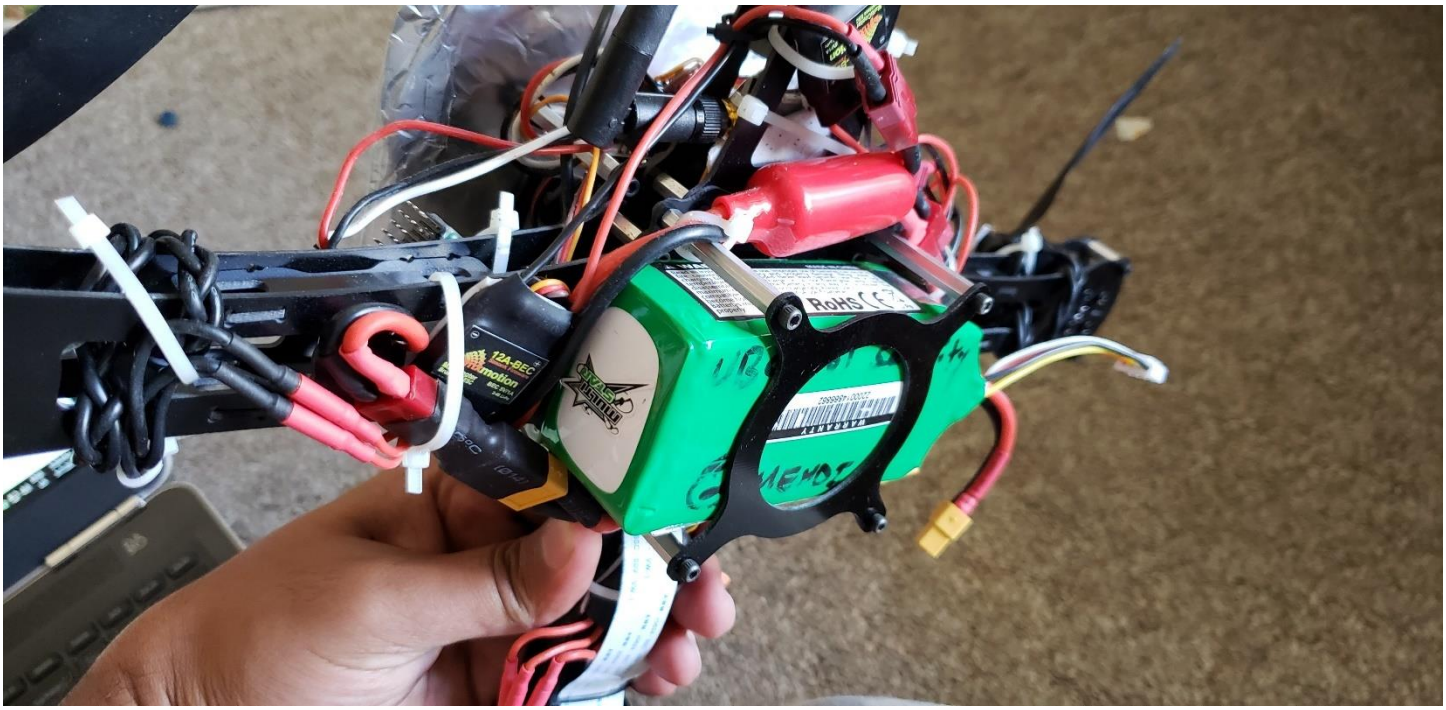
time as well as to program missions into the drone for autonomous flying. https://www.robotshop.com/en/ttl-3dr-radio-telemetry-915mhz-kit.html

TTL 3DR Radio Telemetry 915Mhz

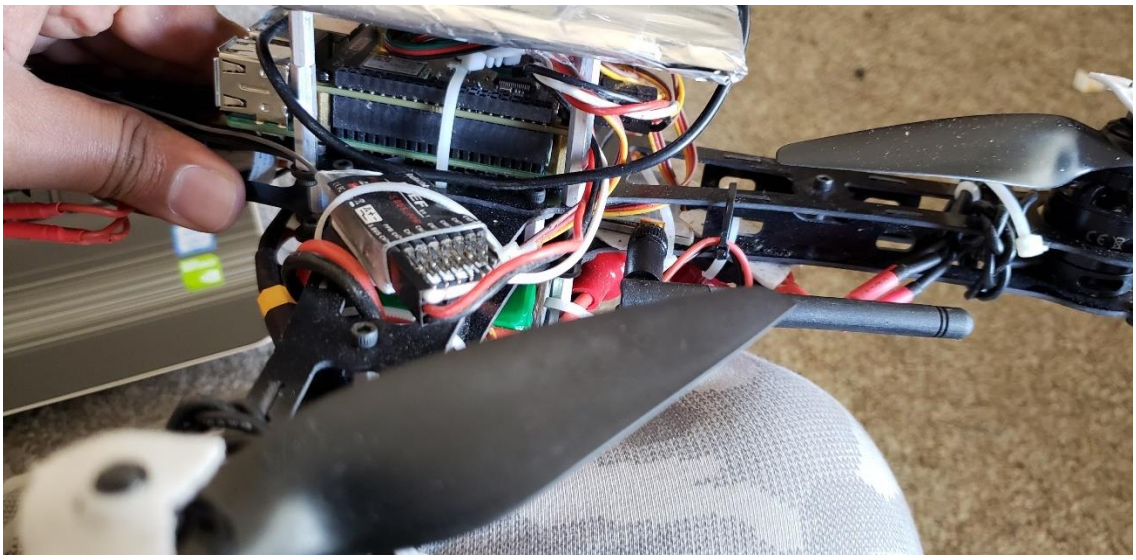
Battery used for the drone is a 5200 MAH 3S Lipo Battery of 11.2V and the radio used for this project is an 8-channel transmitter and receiver from RadioLink T8FB which works on PPM and PWM channel outputs. The operation of T8FB can be determined from the User Manual. This project uses PPM to connect Navio2 and Receiver connection.

*Drone Assembly:*

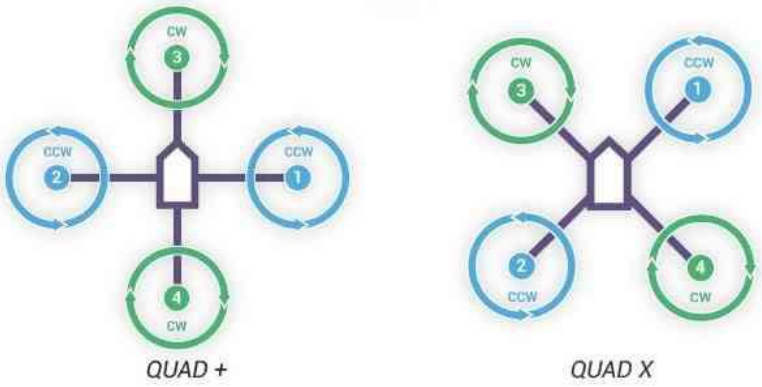
1) Assemble the Lynxmotion frame according to the User Manual provided with the drone kit.
2) Place the battery in the lower compartment



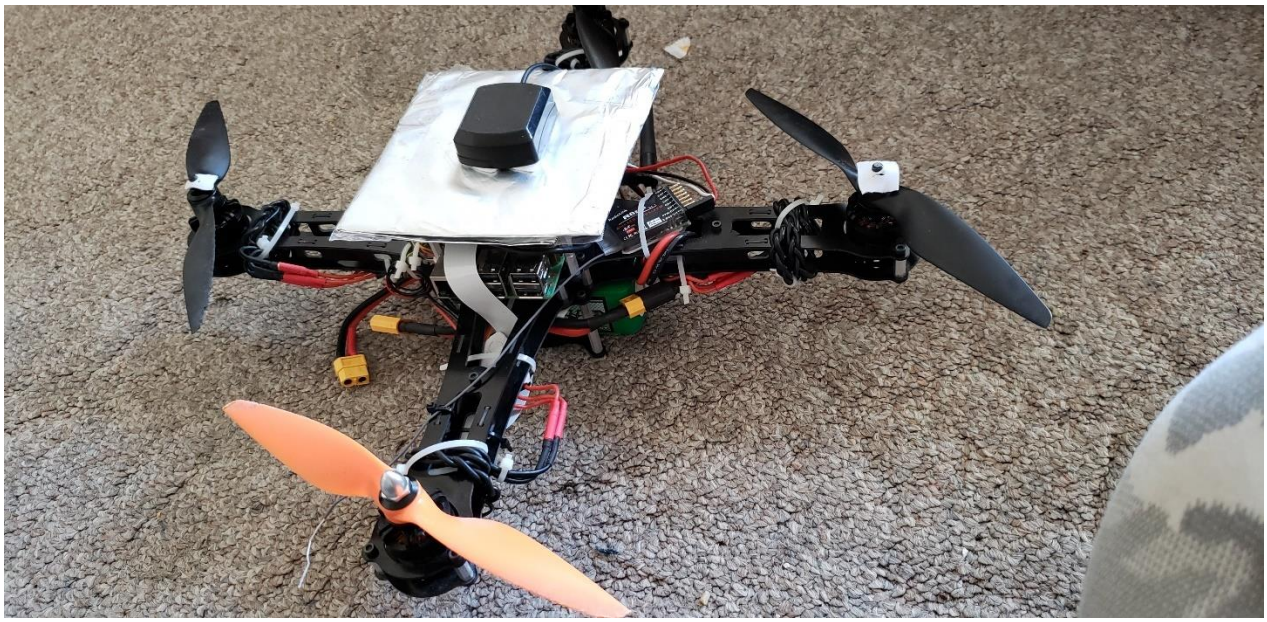
3) Place the Navio2 and Raspberry Pi in the center chamber of the drone



4) Attach the ESCs, motors, propellers and connect the motors to the Navio2 using below layout




5) The final drone assembly with GPS connection is shown below, for detailed tutorial on Navio2, refer to https://docs.emlid.com/navio2/



# *Drone Software Setup*

## *Preparing Raspberry Pi with Navio 2:*

To prepare flight controller:

1) Burn SD card for Raspberry Pi with latest Emlid software from here: https://docs.emlid.com/navio2/Navio-APM/configuring-raspberry-pi/
2) I used Etcher utility. Before installing SD card in Raspberry Pi, change "wpa_supplicant.conf" and specify Wi-fi SSID and Password.
3) Attach Raspberry to Navio2 according to the manual. Check twice that Navio2 is installed tightly, because that's the reason of having a lot of issues while configuring autopilot. Install SD card.
4) Power RPi with USB-cable or using Navio2 Power Module.
5) Install Nmap and Putty to login to Raspberry Pi. In NMap type "navio" and make search to find your RPi IP address. Type found IP in Putty and connect via SSH to RPi. Login is "pi", password is "raspberry".

## *Starting ArduPilot on Boot:*

1) Download ardupilot

```
sudo apt-get update && sudo apt-get install apm-navio2
```

2) Make ardupilot start automatically on boot by editing /etc/rc.local file on Raspberry pi

```
cd ../..

cd etc/

sudo nano rc.local

sudo nohup ArduCopter-quad -A udp:192.168.1.25:14550 -C /dev/ttyAMA0
> /home/pi/startup_log &
```

3) Connect to Mission Planner, Ground Control System using USB telemetry connection. This is used to calibrate the drone as well as plan missions in autonomous mode.

## *Sensor Code in Raspberry Pi:*

This project uses Python 2.7 for development. I cloned the Navio2 library, https://github.com/emlid/Navio2/tree/master/Python, which has all the required device drivers for sensors on Navio board. Using these drivers, a python script was developed which would read the sensor data as threads and publish the data to IoT server using MQTT Protocol. This IoT server then serves the data for post processing AWS services as well as to the frontend website accessible at http://www.ece.ubc.ca/~mkarimib/Drone/index.html.

## *Front-End website:*

This project included broadcasting the live sensor data, as well as, live Raspberry Pi Camera video to a frontend which is accessible to public. This website was designed using HTML5 and CSS. I have used graph to plot sensor graphs with respect to time, aws-sdk for displaying Video Stream from AWS Kinesis and Google OAuth API for authenticating the accesses to the drone. All write commands from the front-end are controlled using google authentication making the drone secure from malicious attacks. The website is accessible at http://www.ece.ubc.ca/~mkarimib/Drone/index.html.

## *Amazon Web Services:*

I have utilized a lot of services from Amazon for this project, namely, Kinesis Video Stream, IoT Core, lambda, API Gateway, DynamoDB, etc. Amazon provides handful of services at very low cost making it suitable for this project, which required controlling and accessing Drones using cloud IoT.

Drone is subscribed as an IoT device under IoT core, this https://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html tutorial gives enough details on how to connect the IoT devices to IoT core. I also used IoT rules which was responsible for updating and writing to the DynamoDB with the Sensor data for backup and further data processing. IoT devices are secured using keys and certificates from Amazon which can be obtained by following steps shown in the above link.

DynamoDB is a type of database provided by Amazon, where one can save dynamic data and then query through it using SQL. This database was used to store sensor data like, current data as well as logged history of each sensor with time-stamp. This logged history was used by the front-end to generate the graphs.

Kinesis Video Stream, is an AWS provided, stream engine where the Raspberry Pi can publish the video. All the real-time video from raspberry pi is published using aws-kinesis-video-stream-sdk to a specific stream defined on Kinesis. I used https://aws.amazon.com/kinesis/video-streams/raspberry-pi-tutorial/ tutorial to setup the stream and control the connections. This video stream is then utilized by the front-end to display the real-time video on the website. The front-end is designed using video-player javascript as well as aws-sdk for connections and data streaming. This video stream can be accessed at http://www.ece.ubc.ca/~mkarimib/Drone/kinesis.html.

I used API Gateway to provide API access to AWS services. All the accesses from front-end as well as publishing from raspberry pi is done using these APIs. API Gateway provides a flexible way of defining custom API connections and functions. Each API access is then used to trigger the respective lambda function. Lambda functions are basic function modules defined in Node.js and Python 2.7, these modules contain code that gets executed when a certain API request is made which in-turn executes the lambda function. For basic understanding, API Gateway provides a way to connect and trigger a function whereas lambda function contains the actual code that gets executed due to those API triggers. API Gateway can be configured using https://aws.amazon.com/api-gateway/getting-started/ tutorial.

| Function | Description | API Gateway Example Usage |
|---|---|---|
| GetSensorCurrStatus | Gets the current status of the sensors. No parameters. | GET https://uniqueid.execute-api.us-west-2.amazonaws.com/dev/GetSensorCurrStatus |
| GetDroneSensorStatus | Query past data for a given `sensorId` (path parameter) and time range (`timestart`, `timeend` in unix time as query parameters). | GET https://unique.execute-api.us-west-2.amazonaws.com/dev/GetDroneSensorStatus/barometer?timestart=0&timeend=1512024820868 |
| UpdateBarometer | Update to the cloud the temperature and pressure. Requires `temp` and `pressure` parameters in the body. | PUT https://uniqueid.execute-api.us-west-2.amazonaws.com/dev/UpdateBarometer |
| UpdateGPS | Update to the cloud the latitude, longitude and altitude. Requires `latitude`, `longitude` and `altitude` parameters in the body. | PUT https://uniqueid.execute-api.us-west-2.amazonaws.com/dev/UpdateGPS |
| CheckGoogleOauth | This Lambda function is automatically invoked prior to each API request to send a command. It checks the provided Google OAuth token in the `authorizationToken` header field to see if it is valid. If so, decrement the number of commands the user has available. Otherwise, reject the call. Set the `CLIENT_ID` environment variable to the client ID from the created Google API Project. Set the `PASSWORD` environment variable to the admin password. Requires | Not meant to be used as an API Gateway endpoint. |

| Function | Description | API Gateway Example Usage |
|---|---|---|
| | the `google-auth-library` node module, so this Lambda Function needs to be zipped up with the module, and cannot be pasted into the editor. | |
| CheckToken | Similar to CheckGoogleOAuth, but only allows for admin password instead. Set the `PASSWORD` environment variable to the admin password. | Not meant to be used as an API Gateway endpoint. |

Table: Endpoints for API Gateway

### *MQTT Protocol:*

MQTT stands for MQ Telemetry Transport. It is a publish/subscribe, extremely simple and lightweight messaging protocol, designed for constrained devices and low-bandwidth, high-latency or unreliable networks. The design principles are to minimise network bandwidth and device resource requirements whilst also attempting to ensure reliability and some degree of assurance of delivery. These principles also turn out to make the protocol ideal of the emerging “machine-to-machine” (M2M) or “Internet of Things” world of connected devices, and for mobile applications where bandwidth and battery power are at a premium.

The basic MQTT operation involves defining topics for communication. These topics can be viewed as simple topic for communication, like, sensor/gps/payload in our case. The IoT devices then subscribe to this topic, this can be done using MQTT Python library. Once the devices are subscribed, they can communicate by publishing and reading from the topic. Basically, one device can publish some data on the topic, like raspberry pi publish the GPS data to sensor/gps/payload and AWS IoT Core listens to the topic for any incoming message. Once the message is received, the IoT rules are triggered. Similarly, IoT Core can publish any message on the topic which is then heard by raspberry pi and the code is executed on raspberry pi based on the message payload as well as the topic on which the message was received.

## *Future Work*

This project provides a basic infrastructure for developing IoT based drones, enabling further developments like drone swarms, automatic control of drones using master and save configuration, object detection using Kinesis Video Stream and AWS Rekognition, geotagging of drone using Google Map API with the front-end. Other functionalities like attitude control of the drone can be governed using Front-End of using MQTT with other drones, i.e., one drone can instruct another drone to fly accordingly. The possibilities of advancements is high due to the flexible and scalable infrastructure defined in this project.

# Conclusion

There have been a lot of recent advancements in Drone technologies, with this project I was able to create and design an infrastructure which could be scaled to n number of drones and allowing each drone to communicate real-time with MQTT communication using IoT Core from AWS. I also achieved the communication as well as real-time drone data using Robot Operating System and MAVROS link connection. The architecture above can be scaled and used for various other autopilot projects. This project can be a baseline to show the interdependencies of AWS services and interlinking as well as interactions amongst the AWS services.

# Acknowledgment

I would like to thank Dr. Andre Ivanov for his continuous guidance and supervision throughout the project. I would also thank Mehdi Karimi for being a well-learnt and well-informed mentor as well as for helping me in places where I got stuck with the drone. He also provided me with the AWS account to work on for this project as well as for hosting my website on his ECE domain at UBC.

# References

1) http://mqtt.org/documentation
2) http://ardupilot.org/rover/docs/common-navio2-overview.html
3) https://docs.emlid.com/navio2/
4) https://github.com/mbiuki/minicloud
5) https://docs.aws.amazon.com/index.html?nc2=h_ql_doc