**Summary of RADAR**

The term RADAR is an acronym for "Radio Detection and Ranging " which means the detection and measurement of distance by radio waves.

A Radar is a device developed in an army capable of electromagnetic capture and broadcasting that helps detect object within its range. With the radar appearance, the human was able to observe beyond the capability of the human eye. The Radar also allows you to detect through obstructions such as smoke, dust, clouds, rain, foliage and, especially, to create observation abilities in all weather, throughout the day and night.

During World War II, the concept of Radar became familiar and was known as an essential equipment used in the military to enable the participants to detect moving objects or illegal intrusions from the enemies.

Radar is the indispensable device in the current air traffic control, which was invented by Scottish scientist Robert Watson Watt about 100 years ago.

The Radar is a type of positioning vehicle based on the reflections of the electromagnetic waves of interference from interference on the transmission line.

Along with the development of science and engineering, the radar has been continuously improved and developed to serve not only the military, but also for civilian purposes. Today, with its advantages, the radar still acts as a professional supervisor used on marine vessels, planes, ... Or used in the weather forecast, check the speed of the vehicles, ... From there go deep into serving human life. Looking back at the stage of the radar and development history will give us a more pronounced view of this device

**History of Radar development**

The historical phases of the radar were from early discoveries of the electromagnetic field theory to a wide range of practical radar trials extending from the 19th century until the early 20th century before it was heavily researched and developed during and after the Second World War.

In 1842, Christian Andreas Doppler announced his discovery of sound waves. He noticed that when a transmitter moved near a stationary receiver, the receiver obtained a sound signal with higher frequency than the frequency of the transmitter and vice versa when the sound transmitter moved away from the stationary recording power, the receiver obtained the sound signal has a lower frequency than the frequency of the transmitter. The phenomenon is also proven true to electromagnetic waves and also known as the Doppler effect. [1]

In 1864, the equation on electromagnetic theory was given by James Clark Maxwell, also known as Maxwell's equation. [2]

In 1886, Maxwell's theories were carried out in fact and were proven by Heinrich Hertz to demonstrate the above theories with both electromagnetic waves and light waves. [3]

In 1888, Heinrich Hertz proved that radio waves could be reflected when encountering metallic objects or dielectric materials. [3]

In 1900, the inventor Nikola Tesla made the idea of radar-like devices. [4]

Tests for detecting objects with radio wave were first made in 1904 by the German inventor Christian H ̈ulsmeyer. He demonstrated the ability to detect a ship in dense fog conditions but was unable to determine the distance from the generator. He was patented for this invention in January 4/1904 and the invention was subsequently improved by H ̈ulsmeyer with the ability to approximate the distance to the ship. [5]

In 1922, Guglielmo Marconi proved to be able to detect sea vessels and communications through the continental radio waves [6]. At the same time, Albert Hoyt Taylor and Leo C. Young used CW (Continuous Wave) Radar to spot a wooden ship on the sea. [7]

Throughout the years 1920 to 1930, the US, Germany, France, the Soviet Union and especially Britain were focusing on radar research and the technology was considered a military secret. However, despite having spent a lot of time researching, the best radar systems can now only provide information on the direction of large objects appearing in a close distance. The distance and altitude parameters compared to the sea surface are not yet able to be calculated.

Robert Watson Watt-a scientific consultant in the field of communication was invited to the British War Board (BWC-British War Council) to assess about a beam of death (death ray – theoretically a particle or an electromagnetic weapon). Here he invented a complete radar device, used in military and on 26/2/1935, his invention was patented. [8]

Shortly after its inception, the radar was promoted to its strategic effect in the British air Battle of 1940. Although it was only 10 miles (16 km) away, the system was sufficiently large enough to detect that a bomber or fighter was up close. More importantly, the system was used to instruct British fighters against the Luftwaffe directly from the ground while German planes had to search for air targets.

The actual breakthrough only appears when a modern identification radar system is created thanks to the invention of the extremely short (micro) wave used in the home or precisely from the device that generates the microwave-magnetron. Magnetron was invented by John Randall and Harry Boot in 1940 at the University of Birmingham [9], although the radar's range is not yet large, just over 80 km.

The Radar today has evolved in a variety of waves, such as sound waves, radio waves, optics and lazer. In addition to giant radars, super mini radars were also pre-built for super-small reconnaissance aircraft. Each system is tailored to different missions. In addition, technological advancements can change the way a radar system works.

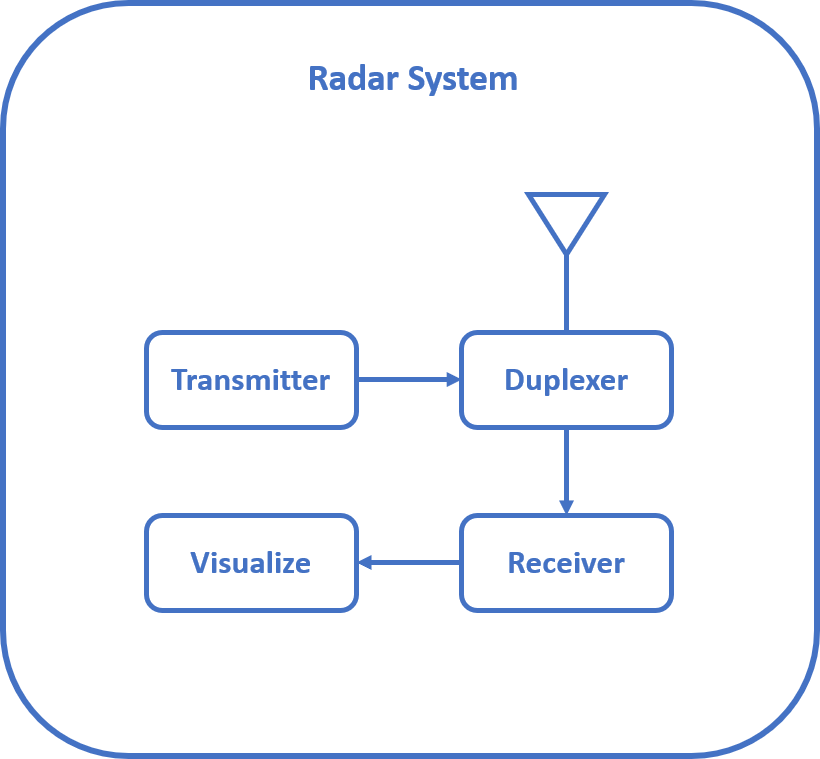
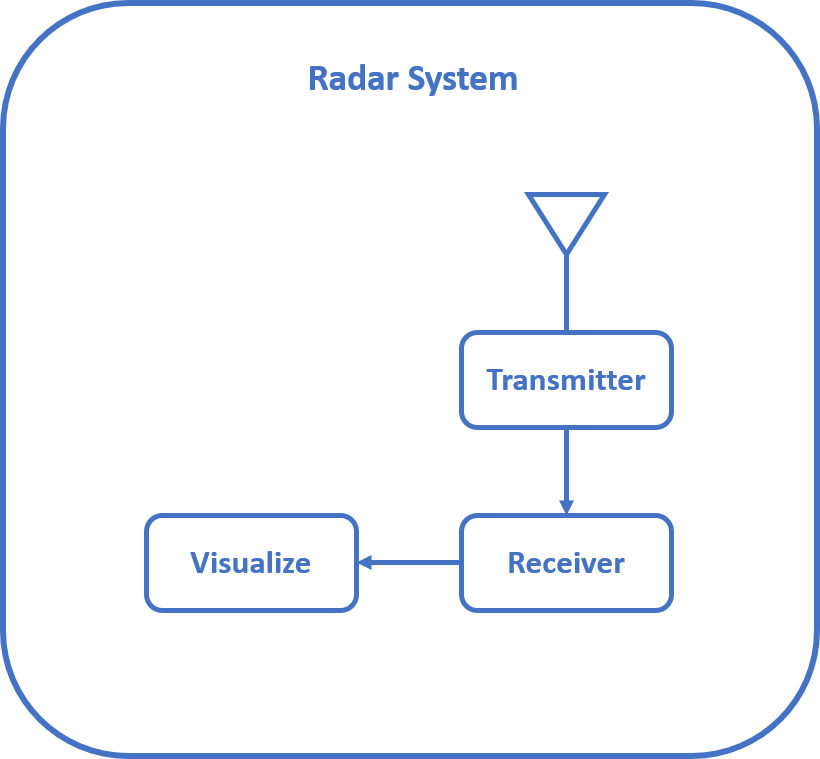
Along with the strong development of the science and Technology of electronic engineering, information technologies, mechanical-electronic industry, new materials, automation, nanotechnology... Today's Radar has also been newly developed and developed in diverse range of waves. The Radar is also designed and deployed both in civil applications.

**Basic knowledge of RADAR**

The Radar operates based on radiation and reflexes of electromagnetic waves in space. Electromagnetic waves spread in straight-line space and at an approximate speed equal to the speed of light in a vacuum (3x108 m/s). Specifically, the radar generates an amount of electromagnetic energy from radiation into space and analyzes the electromagnetic energy from the reflex back from the barrier within its range. Taking a simple example, when we stood in a cave we could hear our voice reflected from the wall, or the bat's ability to detect its path by analyzing the ultrasonic.

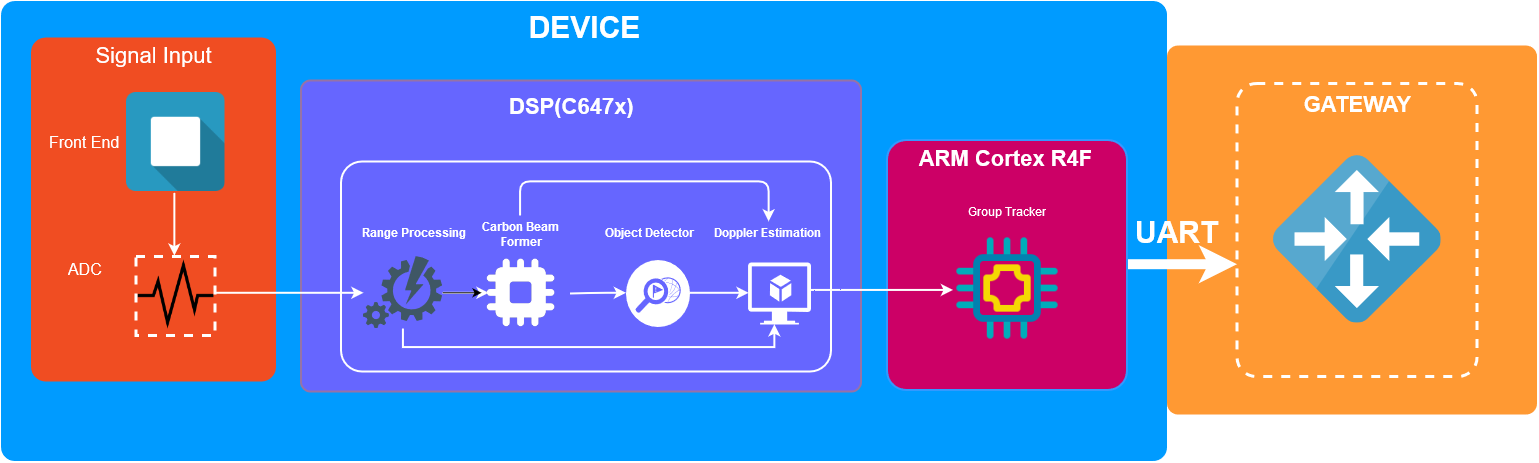
A simple radar system is illustrated in Figure \_\_\_\_\_ . The above radar system consists of a transmitter, a receiver and 2 antennas with the ability to radiated electromagnetic energy on the Transmitter side and to collect the electromagnetic energy from the object in the Receiver. In particular, the system will generate an RF signal (Radio Frequency) on the side of the transmitter and radiation to the space thanks to the antenna in the transmitter side. The signal on when encountering the barrier will be reflected in many different directions which will have the signal reflected back towards the receiver antenna. From there, the antenna on the receiver side will receive a reflex signal and then switch to the receiver, the receiver analyzes the difference between the transmitting signal and the receiver to give information about the distance, velocity, movement direction of the object, ...

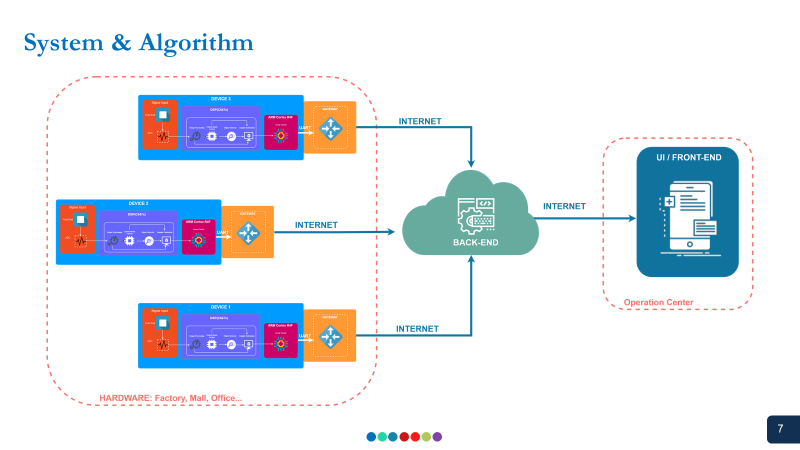
An improvement to the radar system above it is instead of using 2 antennas (1 for the transmitter side and 1 for the receiver side) the radar system in Figure \_\_\_\_ using only 1 antenna for the receiver and play. However, with this system, due to the use of only 1 antenna, it is necessary to isolate the transmitting signal and the receiving signal by the duplexer, thereby reducing the bulky system of the systems compared to the 2 antennas above. Duplexer also helps to protect the receiver from large power signals from the transmitter. In the next section, we will explore more detail about the components that make up a radar system.

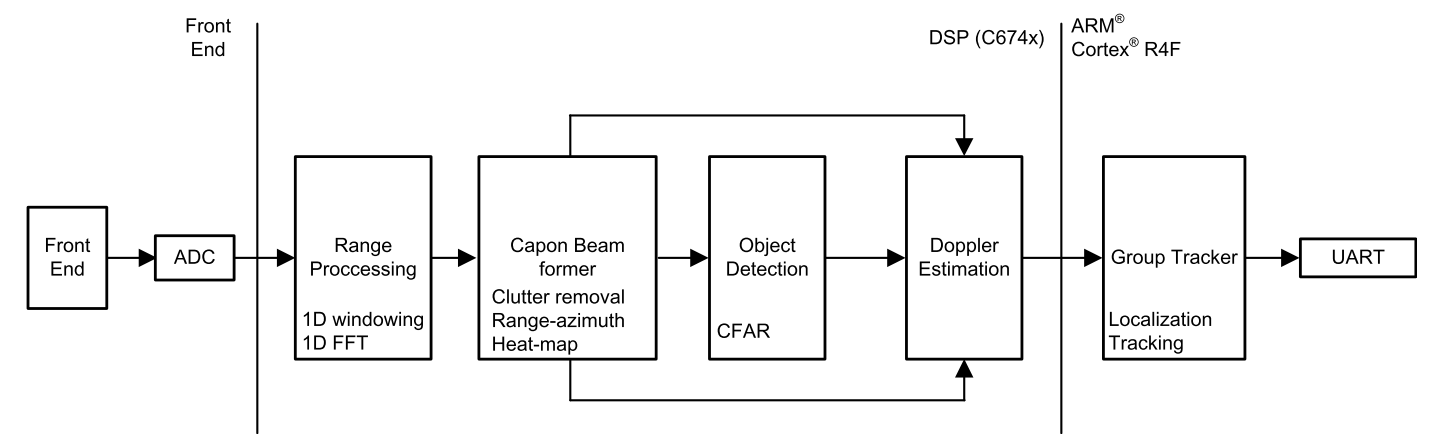


**2.2 Components of a simple RADAR System**

**Introduction of IWR1642**



****

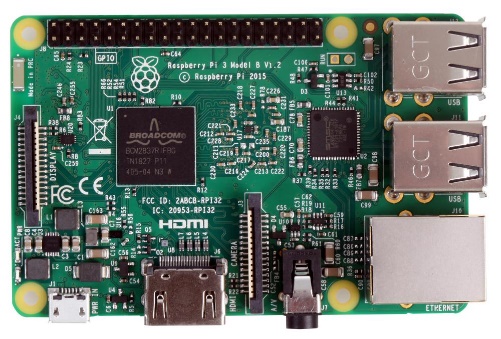
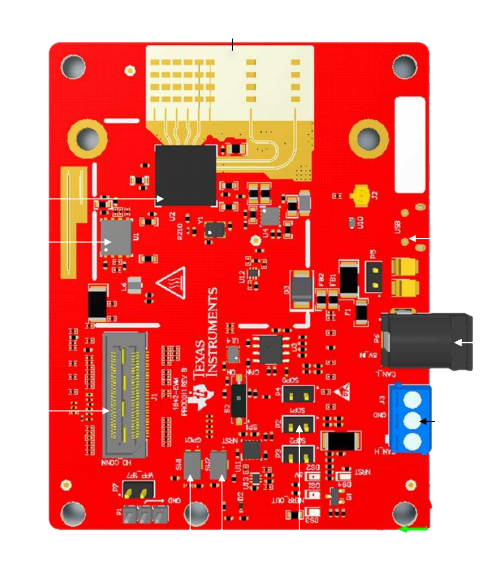


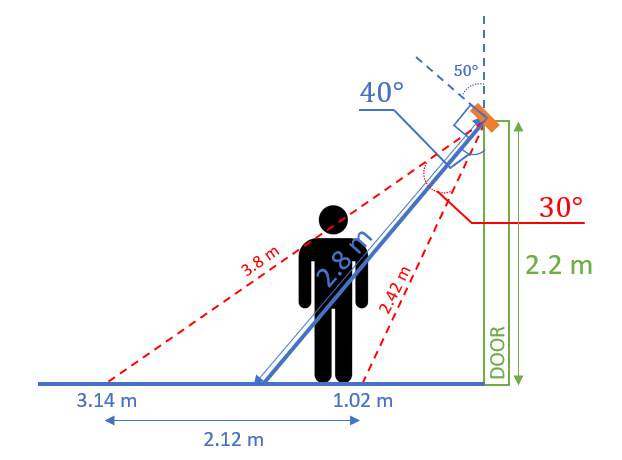
the implementation of the people-counting application demo on the IWR6843 consists of a signal chain running on the C674x DSP, and the tracking module running on the ARM® Cortex®-R4F processor.

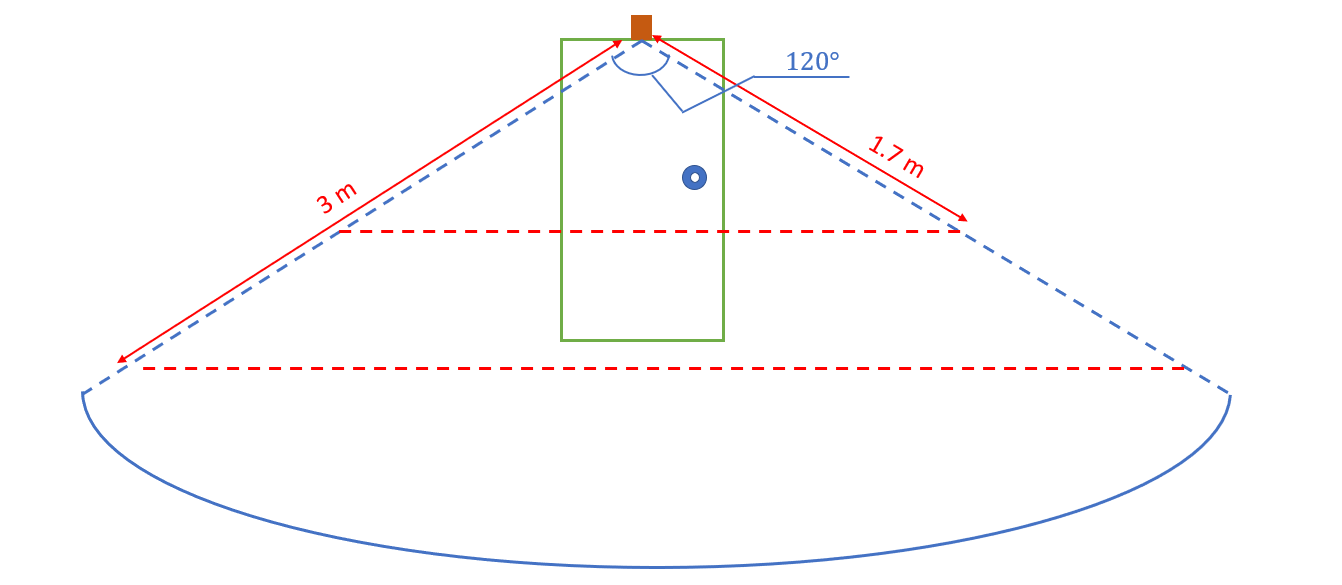
* Range processing: – For each antenna, 1D windowing, and 1D fast Fourier transform (FFT) – Range processing is interleaved with the active chirp time of the frame
* Capon beam forming: – Static clutter removal – Covariance matrix generation, inverse-angle spectrum generation, and integration is performed – Outputs range-angle heat map
* CFAR detection algorithm: – Two-pass, constant false-alarm rate – First pass cell averaging smallest of CFAR-CASO in the range domain, confirmed by second pass cell averaging smallest of CFAR-CASO in the angle domain, to find detection points.
* Doppler estimation: – For each detected [range, azimuth] pair from the detection module, estimate the Doppler by filtering the range bin using Capon beam-weights, and then run a peak search over the FFT of the filtered range bin.
* Tracking: – Perform target localization, and report the results. – Output of the tracker is a set of trackable objects with certain properties like position, velocity, physical dimensions, and point density

**Idea installing the system**

The system have two board the IWR1642 and the Raspberry PI 3. The Idea is to connect the IWR1642 with the Raspberry Pi in order to let the Raspberry auto config and upload the data (Wifi) to the server via MQTT protocol.

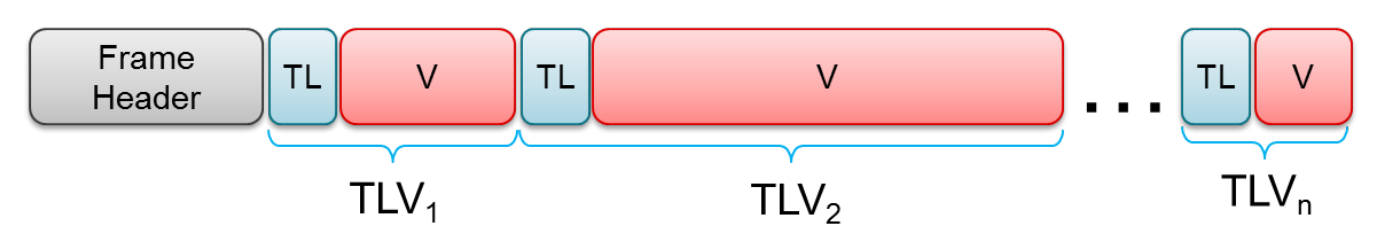
** **

****

****

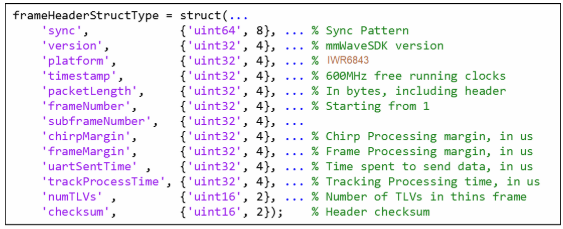
**Data Formats**

A TLV (type-length-value) encoding scheme is used with little endian byte order. For every frame, a packet is sent consisting of a fixed sized Frame Header and then a variable number of TLVs depending on what was detected in that scene. The TLVs can be of types representing the 2D point cloud, target list object, and associated points.

****

**Frame Header (52 bytes)**

The frame header is a fixed size (52 bytes) and has following structure (using MATLAB® notation, with name, type, and length in bytes). The header is designed to self-describe the content, and allow the user application to operate in a lossy environment. The header fields are protected with the checksum,

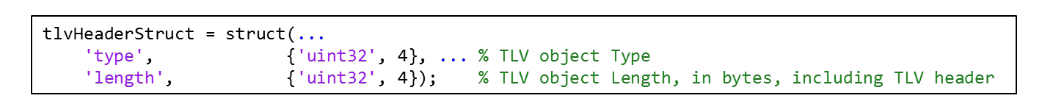
****

**TLVs**

The TLVs can be of type **POINT\_CLOUD\_2D**, **TARGET\_LIST\_2D**, or **TARGET\_INDEX**.

**TLV Header (8 bytes)**

Each TLV has a fixed header (8 bytes) followed by a TLV-specific payload. Figure 11 shows the TLV header.

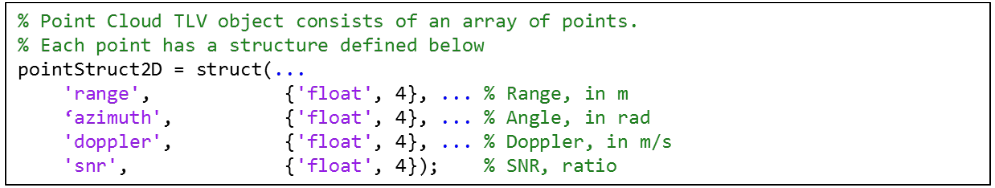


Three TLVs are supported at this time, as follows:

• Point cloud TLV

– Type = POINT\_CLOUD\_2D

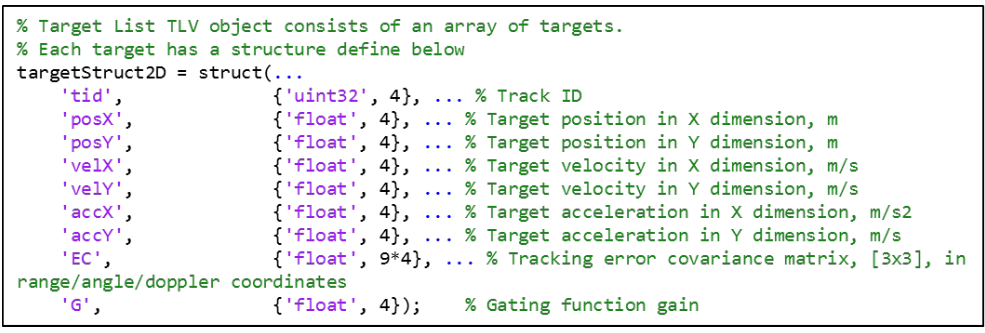
– Length = sizeof (tlvHeaderStruct) + sizeof (pointStruct2D) × numberOfPoints



• Target list TLV

– Type = TARGET\_LIST\_2D

– Length = sizeof (tlvHeaderStruct) + sizeof (targetStruct) × numberOfTargets



• Target Index TLV

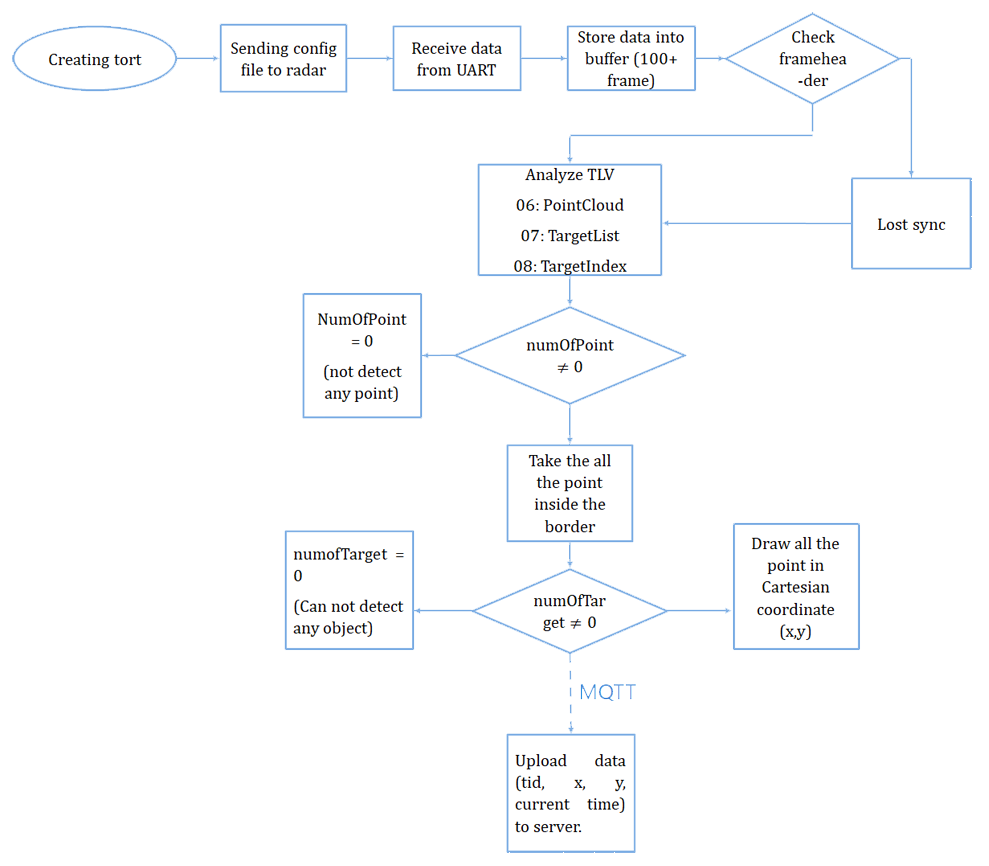
– Type = TARGET\_INDEX

– Length = sizeof (tlvHeaderStruct) + numberOfPoints.

– Payload is a byte array, each byte represents a tracking ID.

**NOTE**: The target index TLV received in the N-th frame indices the point cloud in (N-1)-th frame.

The txrack ID is a byte. Values 0 to 249 are supported. Values 250 to 255 are reserved.

****



Frame Header

Point Cloud TLV

Target List TLV

Target Index TLV

Type Length Header

***For the Frame Header***

* Sync byte of the first frame: 02 01 04 03 06 05 08 07
* Vesion: byte 13 đến byte 16 (42 16 0A 00)
* Packet length: from byte 21 to byte 24 (4A 01 00 00) convert to unit32 which equal 330
* Framenumber: 8D 5E 00 00
* numTLVs: 03 00 (which mean it contain 3 types of TLV: pointCloud\_TLV, targetList\_TLV, targetIndex\_TLV)

***For the pointCloud\_TLV:***

PointCloud\_TLVheader: 06 00 00 00 38 00 00 00 (06 is an TLV type, 338 is the length of of PointCloud\_TLV, which converted to Hex equal 56)

If we remove the length of the header we will have the PointStruct2D = 56 – 8 = 48 bytes.

Each PointStruct2D have the length of 16 bytes, so (numberOfPoint)

Consider 16 bytes of the first PointStruct2D (Float 32 bit converted):

* Range: 6C D6 8F 3F

The distance estimated: 1.1237 (m)

* Azimuth: DB 0F C9 3D

The angle estimated: 0.0982 (rad)

* Doppler: B3 15 A6 3D

The Doppler estimated: 0.0811 (m/s)

* SNR: 1B 30 0A 41

The SNR estimated: 8.6367

Consider the next 16 bytes of the second PointStruct2D (Float 32 bit converted):

* Range: 59 99 A2 3F

The distance estimated: 1.2703 (m)

* Azimuth: 92 0A 86 3D

The angle estimated: 0.0654 (rad)

* Doppler: B3 15 A6 BD

The Doppler estimated: -0.0811 (m/s)

* SNR: 49 6D 18 41

The SNR estimated: 9.5267

Consider the last 16 bytes of the third PointStruct2D (Float 32 bit converted):

* Range: 52 DA A8 3F

The distance estimated: 1.3192 (m)

* Azimuth: 92 0A 86 3D

The angle estimated: 0.0654 (rad)

* Doppler: B3 15 A6 BD

The Doppler estimated: -0.0811 (m/s)

* SNR: 38 26 02 41

The SNR estimated: 8.1343

***For the targetList\_TLV:***

* Type\_TLV: 07 (which is an TLV type)

The Length of the PointCloud\_TLV: D4 which convert to HEX equal 212.

If we remove the header we will have the targetStruct= 212 – 8 = 204 bytes.

Each targetStruct have the fixed length is 68 bytes so (numberOfTarget)

Consider the first targetStruct:

* Track ID: 00 00 00 00

Which is an ID of the target (ID = 0)

* PosX: 7B BA A3 3D

The X coordinate of the target = 0.0799 (m)

* PosY: 83 4F 98 3F

The Y coordinate of the target = 1.1899 (m)

* VelX: FE 47 0A BE

The velocity base on X coordinate of the target = -0.135 (m/s)

* VelY: 00 B0 77 38

The velocity base on Y coordinate of the target = (m/s)

* AccX: 1E 9F D9 BE

The acceleration base on X coordinate of the target = -0.425 (𝑚/𝑠2)

* AccY: 80 BB B0 3A

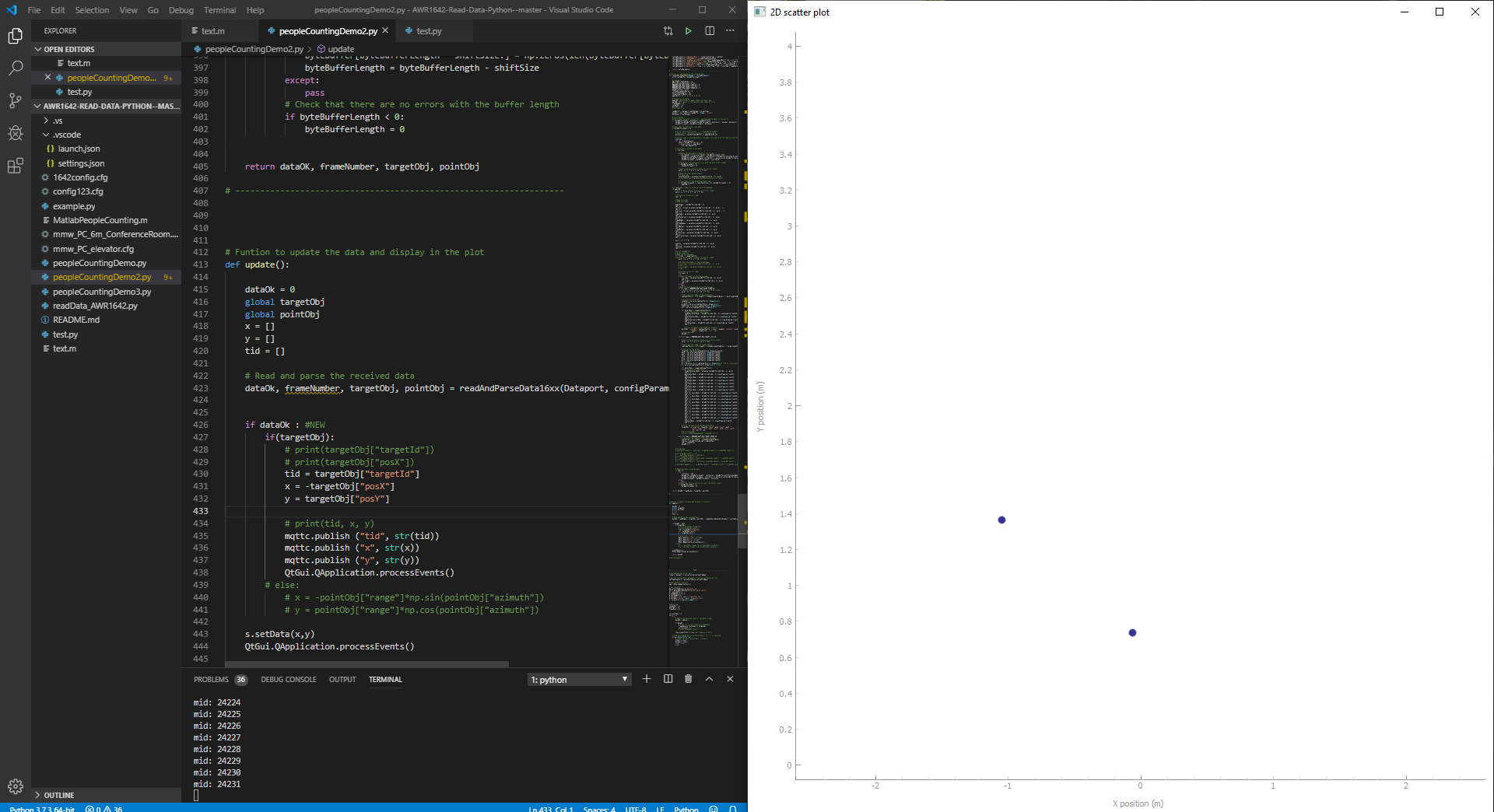
The acceleration base on Y coordinate of the target = 0.0013 (m/s2)

* Gaiting: C8 79 A0 40

Gaiting that combine with the PointCloud = 5.0149

We examine the same with the other TargetStruct.

**THE RESULT OF THE PEOPLE COUNTING APPLICATION IN PYTHON**



The output data we want is:

X, Y, tid (target ID that maximum is 20 target from 0 -> 20), Date&Time

**Source Code**

import serial

import time

import numpy as np

import paho.mqtt.client as mqtt

import datetime

import pyqtgraph as pg

from pyqtgraph.Qt import QtGui

#Setting for currentime

#----------------------------------------------------------------------

datetime.datetime.now()

datetime.datetime(2009, 1, 6, 15, 8, 24)

#----------------------------------------------------------------------

# MQTTconfig

#----------------------------------------------------------------------

broker\_url = "soldier.cloudmqtt.com"

broker\_port = 11141

username = "spoeosqs"

password = "ac\_554KtH4Gr"

mqttc = mqtt.Client()

mqttc.username\_pw\_set (username, password)

mqttc.connect(broker\_url,broker\_port)

# Define event callbacks

def on\_connect(client, userdata, flags, rc):

    print("rc: " + str(rc))

def on\_message(client, obj, msg):

    print(msg.topic + " " + str(msg.qos) + " " + str(msg.payload))

def on\_publish(client, obj, mid):

    print("mid: " + str(mid))

# Assign event callbacks

mqttc.on\_message = on\_message

mqttc.on\_connect = on\_connect

mqttc.on\_publish = on\_publish

#-----------------------------------------------------------------

# Change the configuration file name

configFileName = 'mmw\_PC\_elevator.cfg'

CLIport = {}

Dataport = {}

byteBuffer = np.zeros(2\*\*15,dtype = 'uint8')

byteBufferLength = 0;

# -----------------------------------------------------------------------------

# -----------------------------------------------------------------------------

# -----------------------------------------------------------------------------

# Function to configure the serial ports and send the data from

# the configuration file to the radar

def serialConfig(configFileName):

    global CLIport

    global Dataport

    # Open the serial ports for the configuration and the data ports

    # Raspberry pi

    # CLIport = serial.Serial('/dev/ttyACM0', 115200)

    # Dataport = serial.Serial('/dev/ttyACM1', 921600)

    # Windows

    CLIport = serial.Serial('COM4', 115200)

    Dataport = serial.Serial('COM3', 921600)

    # Read the configuration file and send it to the board

    config = [line.rstrip('\r\n') for line in open(configFileName)]

    for i in config:

        CLIport.write((i+'\n').encode())

        print(i)

        time.sleep(0.01)

    return CLIport, Dataport

# -----------------------------------------------------------------------------

# -----------------------------------------------------------------------------

# -----------------------------------------------------------------------------

# Function to parse the data inside the configuration file

def parseConfigFile(configFileName):

    configParameters = {} # Initialize an empty dictionary to store the configuration parameters

    # Read the configuration file and send it to the board

    config = [line.rstrip('\r\n') for line in open(configFileName)]

    for i in config:

        # Split the line

        splitWords = i.split(" ")

        # Hard code the number of antennas, change if other configuration is used

        numRxAnt = 4

        numTxAnt = 2

        # Get the information about the profile configuration

        if "profileCfg" in splitWords[0]:

            startFreq = int(float(splitWords[2]))

            idleTime = int(splitWords[3])

            rampEndTime = float(splitWords[5])

            freqSlopeConst = float(splitWords[8])

            numAdcSamples = int(splitWords[10])

            digOutSampleRate = int(splitWords[11])

            numAdcSamplesRoundTo2 = 1;

            while numAdcSamples > numAdcSamplesRoundTo2:

                numAdcSamplesRoundTo2 = numAdcSamplesRoundTo2 \* 2;

            digOutSampleRate = int(splitWords[11]);

        # Get the information about the frame configuration

        elif "frameCfg" in splitWords[0]:

            chirpStartIdx = int(splitWords[1]);

            chirpEndIdx = int(splitWords[2]);

            numLoops = int(splitWords[3]);

            numFrames = int(splitWords[4]);

            framePeriodicity = int(splitWords[5]);

    # Combine the read data to obtain the configuration parameters

    numChirpsPerFrame = (chirpEndIdx - chirpStartIdx + 1) \* numLoops

    configParameters["numDopplerBins"] = numChirpsPerFrame / numTxAnt

    configParameters["numRangeBins"] = numAdcSamplesRoundTo2

    configParameters["rangeResolutionMeters"] = (3e8 \* digOutSampleRate \* 1e3) / (2 \* freqSlopeConst \* 1e12 \* numAdcSamples)

    configParameters["rangeIdxToMeters"] = (3e8 \* digOutSampleRate \* 1e3) / (2 \* freqSlopeConst \* 1e12 \* configParameters["numRangeBins"])

    configParameters["dopplerResolutionMps"] = 3e8 / (2 \* startFreq \* 1e9 \* (idleTime + rampEndTime) \* 1e-6 \* configParameters["numDopplerBins"] \* numTxAnt)

    configParameters["maxRange"] = (300 \* 0.9 \* digOutSampleRate)/(2 \* freqSlopeConst \* 1e3)

    configParameters["maxVelocity"] = 3e8 / (4 \* startFreq \* 1e9 \* (idleTime + rampEndTime) \* 1e-6 \* numTxAnt)

    return configParameters

# -----------------------------------------------------------------------------

# -----------------------------------------------------------------------------

# -----------------------------------------------------------------------------

# Funtion to read and parse the incoming data

def readAndParseData16xx(Dataport, configParameters):

    global byteBuffer, byteBufferLength

    # Constants

    OBJ\_STRUCT\_SIZE\_BYTES = 12;

    BYTE\_VEC\_ACC\_MAX\_SIZE = 2\*\*15;

    MMWDEMO\_UART\_MSG\_POINT\_CLOUD\_2D = 6;

    MMWDEMO\_UART\_MSG\_TARGET\_LIST\_2D = 7;

    MMWDEMO\_UART\_MSG\_TARGET\_INDEX\_2D = 8;

    maxBufferSize = 2\*\*15;

    tlvHeaderLengthInBytes = 8;

    pointLengthInBytes = 16;

    targetLengthInBytes = 68;

    magicWord = [2, 1, 4, 3, 6, 5, 8, 7]

    # Initialize variables

    magicOK = 0 # Checks if magic number has been read

    dataOK = 0 # Checks if the data has been read correctly

    frameNumber = 0

    targetObj = {}

    pointObj = {}

    #read buffer from the board

    readBuffer = Dataport.read(Dataport.in\_waiting)

    byteVec = np.frombuffer(readBuffer, dtype = 'uint8')     # interpret the buffer in an array

    byteCount = len(byteVec)                                 # Count number in the array

    # Check that the buffer is not full, and then add the data to the buffer

    if (byteBufferLength + byteCount) < maxBufferSize:

        byteBuffer[byteBufferLength:byteBufferLength + byteCount] = byteVec[:byteCount]

        byteBufferLength = byteBufferLength + byteCount

    # Check that the buffer has some data

    if byteBufferLength > 16:

        # Check for all possible locations of the magic word

        possibleLocs = np.where(byteBuffer == magicWord[0])[0]

        # Confirm that is the beginning of the magic word and store the index in startIdx

        startIdx = []

        for loc in possibleLocs:

            check = byteBuffer[loc:loc + 8]

            if np.all(check == magicWord):

                startIdx.append(loc)

        # Check that startIdx is not empty

        if startIdx:

            # Remove the data before the first start index

            if startIdx[0] > 0 and startIdx[0] < byteBufferLength:

                byteBuffer[:byteBufferLength - startIdx[0]] = byteBuffer[startIdx[0]:byteBufferLength]

                byteBuffer[byteBufferLength-startIdx[0]:] = np.zeros(len(byteBuffer[byteBufferLength-startIdx[0]:]),dtype = 'uint8')

                byteBufferLength = byteBufferLength - startIdx[0]

            # Check that there have no errors with the byte buffer length

            if byteBufferLength < 0:

                byteBufferLength = 0

            # word array to convert 4 bytes to a 32 bit number

            word = [1, 2 \*\* 8, 2 \*\* 16, 2 \*\* 24]

            # Read the total packet length

            # Truong code

            totalPacketLen = np.matmul(byteBuffer[20:20 + 4], word)

            # totalPacketLen = np.matmul(byteBuffer[12:12 + 4], word)

            # Check that all the packet has been read

            if (byteBufferLength >= totalPacketLen) and (byteBufferLength != 0):

                magicOK = 1

    # If magicOK is equal to 1 then process the message

    if magicOK:

        # word array to convert 4 bytes to a 32 bit number

        word = [1, 2 \*\* 8, 2 \*\* 16, 2 \*\* 24]

        # Initialize the pointer index

        idX = 0

        # Read the header

        magicNumber = byteBuffer[idX:idX + 8]

        idX += 8

        version = format(np.matmul(byteBuffer[idX:idX + 4], word), 'x')

        idX += 4

        platform = format(np.matmul(byteBuffer[idX:idX + 4], word), 'x')

        idX += 4

        timeStamp = np.matmul(byteBuffer[idX:idX + 4], word)

        idX += 4

        totalPacketLen = np.matmul(byteBuffer[idX:idX + 4], word)

        idX += 4

        frameNumber = np.matmul(byteBuffer[idX:idX + 4], word)

        idX += 4

        subFrameNumber = np.matmul(byteBuffer[idX:idX + 4], word)

        idX += 4

        chirpMargin = np.matmul(byteBuffer[idX:idX + 4], word)

        idX += 4

        frameMargin = np.matmul(byteBuffer[idX:idX + 4], word)

        idX += 4

        uartSentTime = np.matmul(byteBuffer[idX:idX + 4], word)

        idX += 4

        trackProcessTime = np.matmul(byteBuffer[idX:idX + 4], word)

        idX += 4

        word = [1, 2 \*\* 8]

        numTLVs = np.matmul(byteBuffer[idX:idX + 2], word)

        idX += 2

        checksum = np.matmul(byteBuffer[idX:idX + 2], word)

        idX += 2

        # print (byteBuffer)

        # print (numTLVs)

        # Read the TLV messages

        for tlvIdx in range(numTLVs):

        # word array to convert 4 bytes to a 32 bit number

            word = [1, 2 \*\* 8, 2 \*\* 16, 2 \*\* 24]

            # # Check the header of the TLV message

            # tlv\_type = np.matmul(byteBuffer[idX:idX + 4], word)

            # idX += 4

            # tlv\_length = np.matmul(byteBuffer[idX:idX + 4], word)

            # idX += 4

            # Initialize the tlv type

            tlv\_type = 0

            try:

            # Check the header of the TLV message#######################################################################################

                tlv\_type = np.matmul(byteBuffer[idX:idX + 4], word)

                idX += 4

                tlv\_length = np.matmul(byteBuffer[idX:idX + 4], word)

                idX += 4

            except:

                pass

            # Read the data depending on the TLV message

            if tlv\_type == MMWDEMO\_UART\_MSG\_POINT\_CLOUD\_2D: #######################################################################################

                # word array to convert 4 bytes to a 16 bit number

                word = [1, 2 \*\* 8, 2 \*\* 16, 2 \*\* 24]

                # Calculate the number of detected points

                numInputPoints = (tlv\_length - tlvHeaderLengthInBytes) // pointLengthInBytes

                # Initialize the arrays

                rangeVal = np.zeros(numInputPoints, dtype=object)

                #truongCODE

                azimuth = np.zeros(numInputPoints, dtype=object)

                dopplerVal = np.zeros(numInputPoints, dtype=object)

                snr = np.zeros(numInputPoints, dtype=object)

                for objectNum in range(numInputPoints):

                    # Read the data for each object

                    rangeVal[objectNum] = byteBuffer[idX:idX + 4].view(dtype=np.float32)

                    idX += 4

                    azimuth[objectNum] = byteBuffer[idX:idX + 4].view(dtype=np.float32)

                    idX += 4

                    dopplerVal[objectNum] = byteBuffer[idX:idX + 4].view(dtype=np.float32)

                    idX += 4

                    snr[objectNum] = byteBuffer[idX:idX + 4].view(dtype=np.float32)

                    idX += 4

                    # Store the data in the detObj dictionary

                pointObj = {"numObj": numInputPoints, "range": rangeVal, "azimuth": azimuth,\

                            "doppler": dopplerVal, "snr": snr}

                # print(pointObj)

                dataOK = 1

            elif tlv\_type == MMWDEMO\_UART\_MSG\_TARGET\_LIST\_2D:

                # word array to convert 4 bytes to a 16 bit number

                word = [1, 2 \*\* 8, 2 \*\* 16, 2 \*\* 24]

                # Calculate the number of target points

                numTargetPoints = (tlv\_length - tlvHeaderLengthInBytes) // targetLengthInBytes

                # Initialize the arrays

                targetId = np.zeros(numTargetPoints, dtype=np.uint32)

                posX = np.zeros(numTargetPoints, dtype=np.float32)

                posY = np.zeros(numTargetPoints, dtype=np.float32)

                velX = np.zeros(numTargetPoints, dtype=np.float32)

                velY = np.zeros(numTargetPoints, dtype=np.float32)

                accX = np.zeros(numTargetPoints, dtype=np.float32)

                accY = np.zeros(numTargetPoints, dtype=np.float32)

                #TruongCODE

                EC = np.zeros((3, 3, numTargetPoints), dtype=object)  # Error covariance matrix np

                G = np.zeros(numTargetPoints, dtype=object)  # Gain

                for objectNum in range(numTargetPoints):

                # Read the data for each object

                    try:

                        targetId[objectNum] = np.matmul(byteBuffer[idX:idX + 4], word)

                        idX += 4

                        posX[objectNum] = byteBuffer[idX:idX + 4].view(dtype=np.float32)

                        idX += 4

                        posY[objectNum] = byteBuffer[idX:idX + 4].view(dtype=np.float32)

                        idX += 4

                        velX[objectNum] = byteBuffer[idX:idX + 4].view(dtype=np.float32)

                        idX += 4

                        velY[objectNum] = byteBuffer[idX:idX + 4].view(dtype=np.float32)

                        idX += 4

                        accX[objectNum] = byteBuffer[idX:idX + 4].view(dtype=np.float32)

                        idX += 4

                        accY[objectNum] = byteBuffer[idX:idX + 4].view(dtype=np.float32)

                        idX += 4

                        EC[0, 0, objectNum] = byteBuffer[idX:idX + 4].view(dtype=np.float32)

                        idX += 4

                        EC[0, 1, objectNum] = byteBuffer[idX:idX + 4].view(dtype=np.float32)

                        idX += 4

                        EC[0, 2, objectNum] = byteBuffer[idX:idX + 4].view(dtype=np.float32)

                        idX += 4

                        EC[1, 0, objectNum] = byteBuffer[idX:idX + 4].view(dtype=np.float32)

                        idX += 4

                        EC[1, 1, objectNum] = byteBuffer[idX:idX + 4].view(dtype=np.float32)

                        idX += 4

                        EC[1, 2, objectNum] = byteBuffer[idX:idX + 4].view(dtype=np.float32)

                        idX += 4

                        EC[2, 0, objectNum] = byteBuffer[idX:idX + 4].view(dtype=np.float32)

                        idX += 4

                        EC[2, 1, objectNum] = byteBuffer[idX:idX + 4].view(dtype=np.float32)

                        idX += 4

                        EC[2, 2, objectNum] = byteBuffer[idX:idX + 4].view(dtype=np.float32)

                        idX += 4

                        G[objectNum] = byteBuffer[idX:idX + 4].view(dtype=np.float32)

                        idX += 4

                    except:

                        pass

                # Store the data in the detObj dictionary

                targetObj = {"targetId": targetId, "posX": posX, "posY": posY, \

                             "velX": velX, "velY": velY, "accX": accX, "accY": accY, \

                             "EC": EC, "G": G}

            elif tlv\_type == MMWDEMO\_UART\_MSG\_TARGET\_INDEX\_2D: #######################################################################################

                # Calculate the length of the index message

                numIndices = tlv\_length - tlvHeaderLengthInBytes

                indices = byteBuffer[idX:idX + numIndices]

                idX += numIndices

                dataOK = 1

        # Remove already processed data

        if idX > 0:

            try:

                shiftSize = totalPacketLen     # try change idX to totalPacketLen

                byteBuffer[:byteBufferLength - shiftSize] = byteBuffer[shiftSize:byteBufferLength]

                byteBuffer[byteBufferLength - shiftSize:] = np.zeros(len(byteBuffer[byteBufferLength - shiftSize:]),dtype = 'uint8') #NEW

                byteBufferLength = byteBufferLength - shiftSize

            except:

                pass

            # Check that there are no errors with the buffer length

            if byteBufferLength < 0:

                byteBufferLength = 0

    return dataOK, frameNumber, targetObj, pointObj

# ------------------------------------------------------------------

# Funtion to update the data and display in the plot

def update():

    dataOk = 0

    global targetObj

    global pointObj

    global mqttCount

    x = []

    y = []

    tid = []

    # Read and parse the received data

    dataOk, frameNumber, targetObj, pointObj = readAndParseData16xx(Dataport, configParameters)

    if dataOk : #NEW

        if(targetObj):

            tid = targetObj["targetId"]

            x = -targetObj["posX"]

            y = targetObj["posY"]

            tid\_comma = list(tid)

            x\_comma = list(x)

            y\_comma = list (y)

            if mqttCount == 10:

                mqttc.publish ("date & time", str(datetime.datetime.now()))

                mqttc.publish ("tid", str(tid\_comma))

                mqttc.publish ("x", str(x\_comma))

                mqttc.publish ("y", str(y\_comma))

                mqttCount = 0

            else:

                mqttCount = mqttCount + 1

            QtGui.QApplication.processEvents()

    s.setData(x,y)

    QtGui.QApplication.processEvents()

    return dataOk

# -----------------------------------------------------------------------------

# -------------------------    MAIN   -----------------------------------------

# -----------------------------------------------------------------------------

# Configurate the serial port

CLIport, Dataport = serialConfig(configFileName)

# Get the configuration parameters from the configuration file

configParameters = parseConfigFile(configFileName)

#START QtAPPfor the plot

app = QtGui.QApplication([])

# Set the plot

pg.setConfigOption('background','w')

win = pg.GraphicsWindow(title="2D scatter plot")

p = win.addPlot()

p.setXRange(-2.5,2.5)

p.setYRange(0,4)

p.setLabel('left',text = 'Y position (m)')

p.setLabel('bottom', text= 'X position (m)')

s = p.plot([],[],pen=None,symbol='o')

# Main loop

targetObj = {}

pointObj = {}

frameData = {}

mqttCount = 0

currentIndex = 0

while True:

    try:

        # Update the data and check if the data is okay

        dataOk = update()

        if dataOk:

            # Store the current frame into frameData

            frameData[currentIndex] = targetObj

            currentIndex += 1

            # print(targetObj["posX"])

        time.sleep(0.044) # Sampling frequency of 30 Hz

    # Stop the program and close everything if Ctrl + c is pressed

    except KeyboardInterrupt:

        # CLIport.write(('sensorStop\n').encode())

        CLIport.close()

        Dataport.close()

        # win.close()

        break

**MQTT DEFINITION**

MQTT is one of the most commonly used protocols in IoT projects. It stands for Message Queuing Telemetry Transport.

In addition, it is designed as a lightweight messaging protocol that uses publish/subscribe operations to exchange data between clients and the server. Furthermore, its small size, low power usage, minimized data packets and ease of implementation make the protocol ideal of the “machine-to-machine” or “Internet of Things” world.

**WHY USE MQTT ?**

MQTT has unique features you can hardly find in other protocols, like:

* It’s a lightweight protocol. So, it’s easy to implement in software and fast in data transmission.
* It’s based on a messaging technique. Of course, you know how fast your messenger/WhatsApp message delivery is. Likewise, the MQTT protocol.
* Minimized data packets. Hence, low network usage.
* Low power usage. As a result, it saves the connected device’s battery.
* It’s real time! That’s is specifically what makes it perfect for IoT applications.

**HOW MQTT WORKS?**

MQTT is based on clients and a server. Likewise, the server is the guy who is responsible for handling the client’s requests of receiving or sending data between each other.

MQTT server is called a broker and the clients are simply the connected devices.  
So:

* When a device (a client) wants to send data to the broker, we call this operation a “publish”.
* When a device (a client) wants to receive data from the broker, we call this operation a “subscribe”.



**MQTT compare with HTTP**

**Design and Messaging**

MQTT is data centric whereas HTTP is document-centric. HTTP is request-response protocol for client-server computing and not always optimized for mobile devices. Main solid benefits of MQTT in these terms are lightweightness (MQTT transfers data as a byte array) and publish/subscribe model, which makes it perfect for resource-constrained devices and help to save battery.

Besides, publish/subscribe model provides clients with independent existence from one another and enhance the reliability of the whole system. When one client is out of order the whole system can keep on working properly.

**Speed and Delivery**

According to measurements in 3G networks, throughput of MQTT is 93 times faster than HTTP’s.

Besides, in comparison to HTTP, MQTT Protocol ensures high delivery guarantees. There are 3 levels of Quality of Services:

- at most once: guarantees a best effort delivery.

- at least once: guaranteed that a message will be delivered at least once. But the message can also be delivered more than once.

- exactly once: guarantees that each message is received only once by the counterpart

MQTT also provides users with options of Last will & Testament and Retained messages. The first means that in case of unexpected disconnection of a client all subscribed clients will get a message from a broker. Retained message means that a newly subscribed client will get an immediate status update.

HTTP Protocol has none of these abilities.

**Complexity and Message Size**  
  
MQTT (Message Queuing Telemetry Transport) has pretty short specification. There are only CONNECT, PUBLISH, SUBSCRIBE, UNSUBSCRIBE and DISCONNECT types that are significant for developers. Whereas HTTP specifications are much longer.

MQTT has a very short message header and the smallest packet message size of 2 bytes. Using text message format by HTTP protocol allows it to compose lengthy headers and messages. It helps to eliminate troubles because it can be read by humans, but at the same time it’s needless for resource-constrained devices.

**Conclusion**

MQTT Protocol is easy of use. It is essential when response time, throughput, lower battery and bandwidth usage are on the first place for future solutions. It’s also perfect in case of intermittent connectivity.

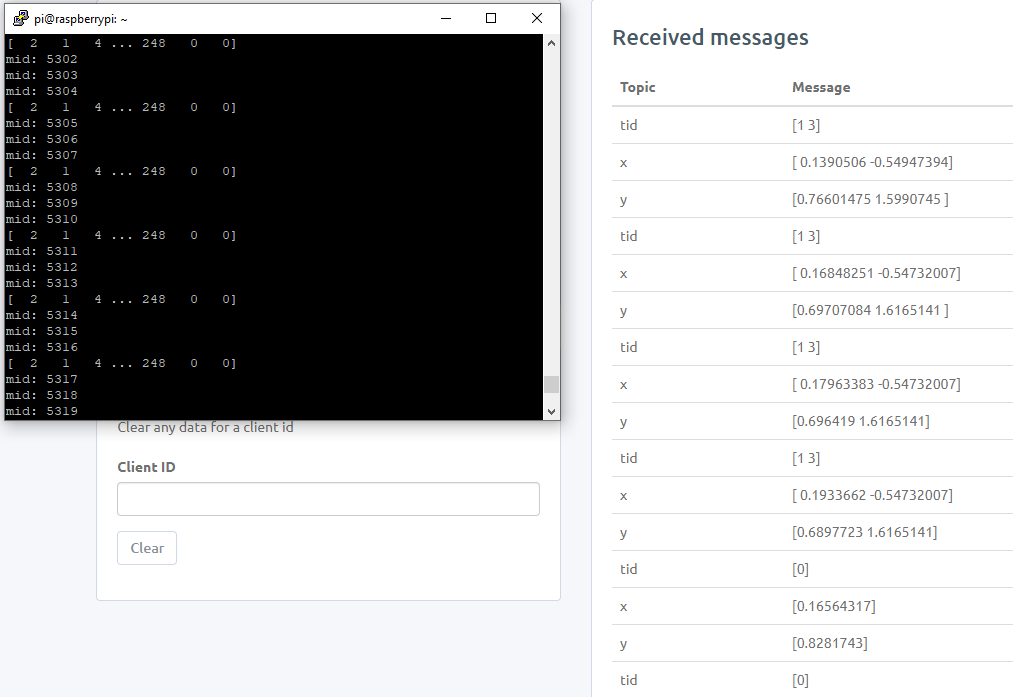
HTTP is worthy and extendable. But MQTT is more suitable when it is referred to IoT development.

**CloudMQTT Broker**

**CloudMQTT are managed Mosquitto servers in the cloud.** Mosquitto implements the MQ Telemetry Transport protocol, MQTT, which provides lightweight methods of carrying out messaging using a publish/subscribe message queueing model.

Also CloudMQTT broker does support **Heroku**

The result after upload the data into CloudMQTT



Let take the first group of data published

**tid = [1 3]**

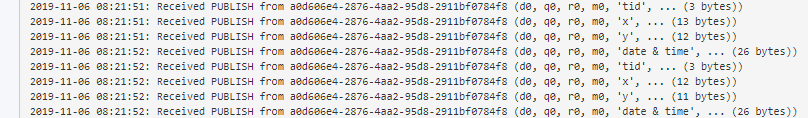
which mean detect 2 target which the first target have an target id is 1 and the second target have target id is 3.

**x = [0.1390506 -0.57347394]**

**y = [0.76601475 1.5990745]**

which mean the fir target have the coordinate (0.1390506 , 0.76601475) and the second target have the coordinate (-0.57347394 , 1.5990745)

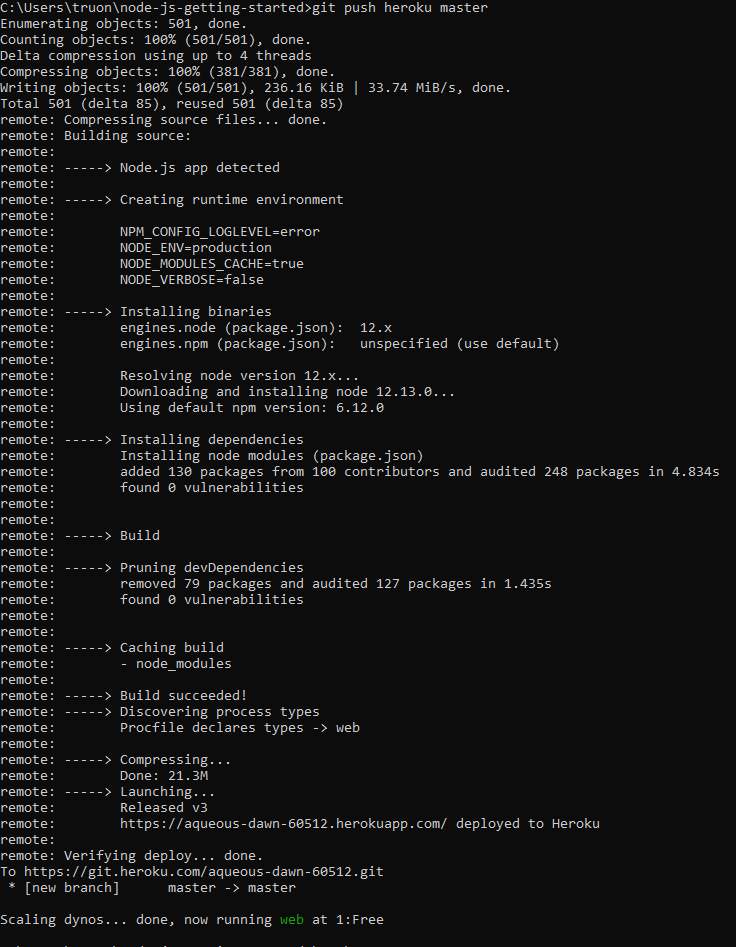
The LOG of publish data into server:



**Heroku**

Heroku deploy app

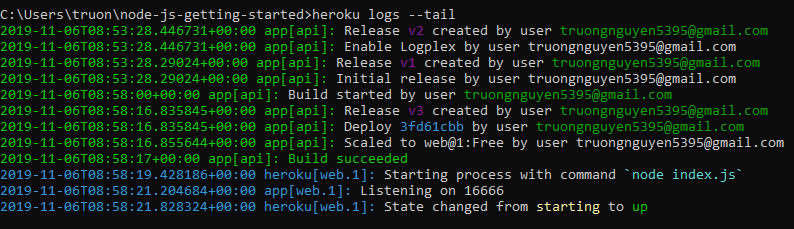
****

****

Check the app is running

heroku ps:scale web=1

View LOG



START A CONSOLE

To get a real feel for how dynos work, you can create another one-off dyno and run the bash command, which opens up a shell on that dyno. You can then execute commands there. Each dyno has its own ephemeral filespace, populated with your app and its dependencies - once the command completes (in this case, bash), the dyno is removed.

$ heroku run bash

Running `bash` attached to terminal... up, run.3052

~ $ ls

Procfile README.md composer.json composer.lock vendor views web

~ $ exit

exit

**PROCFILE**

Heroku apps include a **Procfile** that specifies the commands that are executed by the app on startup.

A Procfile declares its process types on individual lines, each with the following format:

<process type>: <command>

* <process type> is an alphanumeric name for your command, such as web, worker, urgentworker, clock, and so on.
* <command> indicates the command that every dyno of the process type should execute on startup, such as rake jobs:work.

**DYNO**

All Heroku applications run in a collection of lightweight Linux containers called **dynos**. This article describes dyno conventions on the Heroku platform

Every dyno belongs to one of the three following configurations:

* **Web**: Web dynos are dynos of the “web” process type that is defined in your [Procfile](https://devcenter.heroku.com/articles/procfile). Only web dynos receive HTTP traffic from the [routers](https://devcenter.heroku.com/articles/http-routing).
* **Worker**: Worker dynos can be of any process type declared in your Procfile, other than “web”. Worker dynos are typically used for background jobs, queueing systems, and timed jobs. You can have multiple kinds of worker dynos in your application. For example, one for urgent jobs and another for long-running jobs. For more information, see [Worker Dynos, Background Jobs and Queueing](https://devcenter.heroku.com/articles/background-jobs-queueing).
* **One-off**: One-off dynos are temporary dynos that can run detached, or with their input/output attached to your local terminal. They’re loaded with your latest release. They can be used to handle administrative tasks, such as database migrations and console sessions. They can also be used to run occasional background work, as with [Heroku Scheduler](https://devcenter.heroku.com/articles/scheduler). For more information, see [One-Off Dynos](https://devcenter.heroku.com/articles/one-off-dynos).

Once a web or worker dyno is started, the [dyno formation](https://devcenter.heroku.com/articles/scaling#dyno-formation) of your app will change (the number of running dynos of each process type) - and subject to dyno lifecycle, Heroku will continue to maintain that dyno formation until you change it. One-off dynos, on the other hand, are only expected to run a short-lived command and then exit, not affecting your dyno formation.

.

**NoSQL**

NoSQL is an alternative to conventional relational databases in which data is put in tables and the data structure is carefully designed before the database is created. It is mainly helpful for working with huge sets of distributed data. NoSQL databases are scalable, high performant and flexible in nature.

**#1) Column:** Wide column stores and arranges the data tables as columns rather than as rows.

They can query a large volume of data very quickly than the traditional databases. They can be employed for recommendation engines, catalogs, fraud detection, etc.

**Examples:** Cassandra, HBase, Google BigTable, Scylla, Vertica, etc.

**#2) Document:** Document databases, aka document stores and keeps the semi-structured data along with its description in the document format.

Each document has a unique key through which it is addressed. They are helpful for content management and mobile application data handling. They are widely used along with JSON and JavaScript. Document databases also offer an API and query language through which the documents can be fetched based on their contents.

**Examples:** Apache, MongoDB, MarkLogic, CouchDB, BaseX, IBM Domino, etc.

**#3) Key-value:** Key value databases have their data model based on an associative array (map or a dictionary) in which the data has represented a collection of key-value pairs. They are highly suitable for session management and caching in web applications.

**Examples:** Aerospike, Berkeley DB, Apache ignites, Dynamo, Redis, Riak, ZooKeeper, etc.

**#4) Graph:** In graph stores, data is organized as nodes and edges.

You can think of a node as a record and edge as a relationship between the records in the relational database. This model supports a richer representation of data relationships. They are useful for customer relationship Management systems, road maps, reservation systems, etc.

**Examples:** AllegroGraph, InfiniteGraph, MarkLogic, Neo4j, IBM graph, Titan, etc.

**PROBLEMS**

The Program still have some error that stop unexpectedly

**VISUAL ON WEB**

<h3>{{viewTitle}}</h3>

<form action="/location" method="POST" autocomplete="off">

    <div class="row">

        <div class="col-md-6">

            <canvas id = "targetCanvas" width = "750" height="425"></canvas>

            <script>

                window.onload = function() {

                var canvas = document.getElementById("targetCanvas");

                var ctx = canvas.getContext("2d");

                var image = new Image();

                image.src = "../images/image1.jpg";

                ctx.drawImage(image, 0,0, canvas.width, canvas.height);

                }

            </script>

            <img id="myImage" src="../images/image1.jpg" alt="piooop" style="display: none;">

        </div>

        <div class="col-md-6">

            <canvas id="myChart" width="400" height="230"></canvas>

            <script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.8.0/Chart.js"></script>

            <script>

            var ctx = document.getElementById('myChart').getContext('2d');

            var myChart = new Chart(ctx, {

                type: 'line',

                data: {

                    labels: ['0', '1 minute', '2 minute', '3 minute', '4 minute', '5 minute'],

                    datasets: [{

                        label: 'Number of people enter',

                        data: [12, 10, 3, 5, 2, 3],

                        backgroundColor: [

                            'rgba(54, 162, 235, 0.2)',

                            'rgba(255, 99, 132, 0.2)',

                            'rgba(255, 206, 86, 0.2)',

                            'rgba(75, 192, 192, 0.2)',

                            'rgba(153, 102, 255, 0.2)',

                            'rgba(255, 159, 64, 0.2)'

                        ],

                        borderColor: [

                            'rgba(255, 99, 132, 1)',

                            'rgba(54, 162, 235, 1)',

                            'rgba(255, 206, 86, 1)',

                            'rgba(75, 192, 192, 1)',

                            'rgba(153, 102, 255, 1)',

                            'rgba(255, 159, 64, 1)'

                        ],

                        borderWidth: 1

                    }]

                },

                options: {

                    scales: {

                        yAxes: [{

                            ticks: {

                                beginAtZero: true

                            }

                        }]

                    }

                }

            });

            </script>

        </div>

    </div>

    <div class="form-group">

        <button type="submit" class="btn btn-info"><i class="fa fa-database"></i> View Database</button>

    </div>

    <div class="form-group">

        <label>tid</label>

        <input type="text" class="form-control" name="tid" placeholder="ID">

    </div>

    <div class="form-group">

        <label>xAxis</label>

        <input type="text" class="form-control" name="xAxis" placeholder="xAxis" >

    </div>

    <div class="form-group">

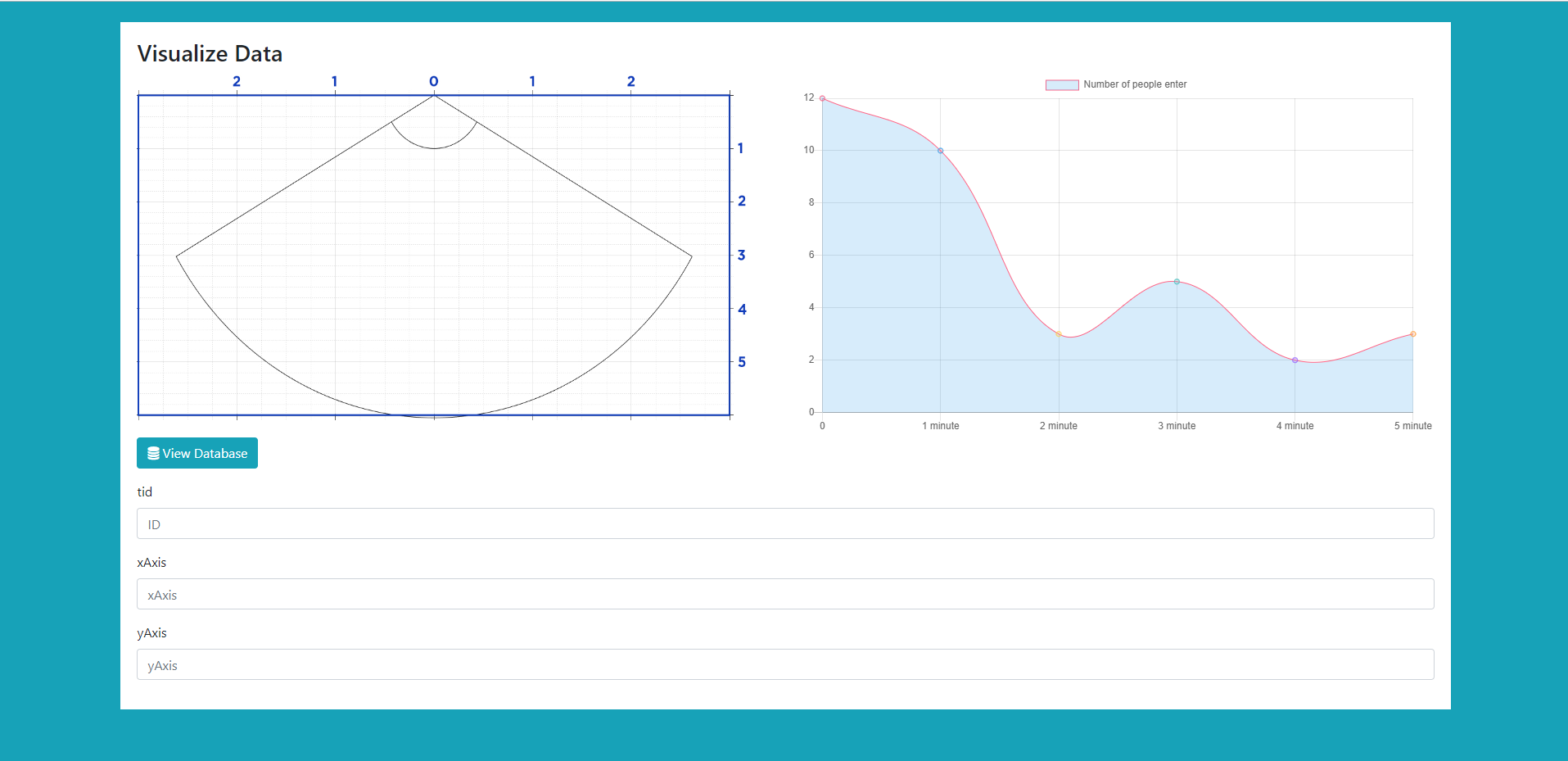
        <label>yAxis</label>

        <input type="text" class="form-control" name="yAxis" placeholder="yAxis" >

    </div>

 </form>

**RESULT**



**Connect between Device and web server via Cloud MQTT, store the data in mongoDB**

CONNECT MQTT WITH NODE.JS

//------------------------CloudMQTT---------------------------------//

var mqtt = require('mqtt');

var Topic = '#';   // Sub to any topic

var err = Error

var options = {

    port: 11141,

    host: 'mqtt://soldier.cloudmqtt.com',

    clientId: 'mqttjs\_' + Math.random().toString(16).substr(2, 8),

    username: 'spoeosqs',

    password: 'ac\_554KtH4Gr',

    keepalive: 60,

    reconnectPeriod: 1000,

    clean: true,

    encoding: 'utf8'

};

var client = mqtt.connect('mqtt://soldier.cloudmqtt.com', options);

client.on('connect',mqtt\_connect);

client.on('reconnect', mqtt\_reconnect);

client.on('error', mqtt\_error);

client.on('message', mqtt\_messsageReceived);

client.on('close', mqtt\_close);

function mqtt\_connect()

{

    console.log("Connecting MQTT");

    client.subscribe(Topic, mqtt\_subscribe);

}

function mqtt\_subscribe(err,grant)

{

    console.log ("Subcribed to " + Topic);

    if (err) {console.log(err);}

}

function mqtt\_reconnect(err)

{

    console.log("Reconnect MQTT");

    if (err) {console.log(err);}

    client  = mqtt.connect('mqtt://soldier.cloudmqtt.com', options);

}

function mqtt\_error(err)

{

    console.log("Error!");

    if (err) {console.log(err);}

}

function mqtt\_messsageReceived(topic, message, packet)

{

    console.log('Topic=' +  topic + '  Message=' + message);

}

function mqtt\_close()

{

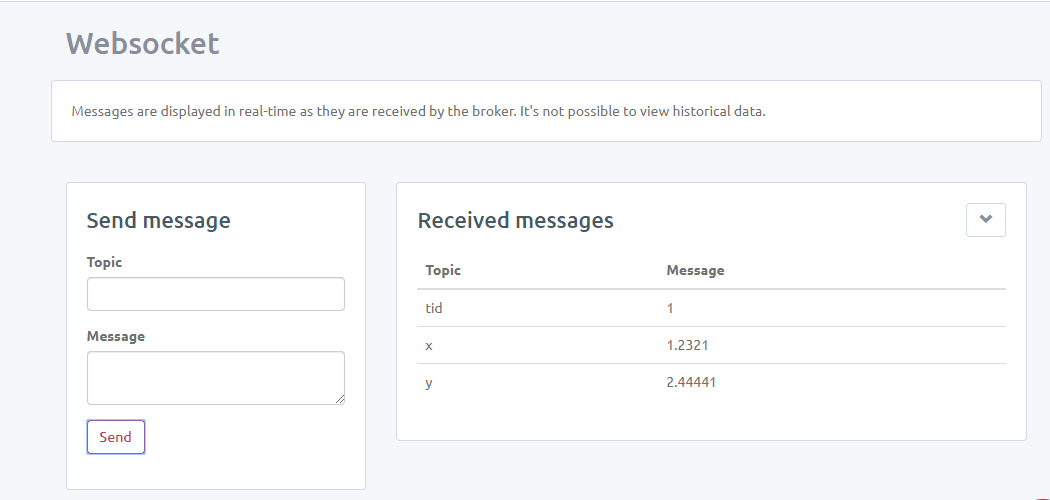
    console.log("Close MQTT");

}

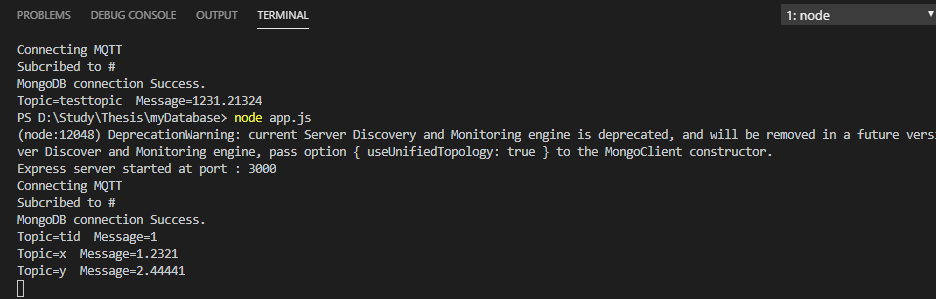
//-----------------------------------------------------------------------/

**RESULT**

Message from CloudMQTT Broker



The app server received



**INSERT DATA INTO MONGODB**

function mqtt\_messsageReceiv(topic, message, packet)

{

    insertRecord(topic, message);

    console.log( topic + " " + message);

}

...

function insertRecord(a, b, c) {

    var location = new Location();

    location.tid = a;

    location.xAxis = b;

    location.yAxis = c;

    location.save((err, doc) => {

        if (!err)

           console.log("Good Job");// res.redirect('/location/list');

        else {

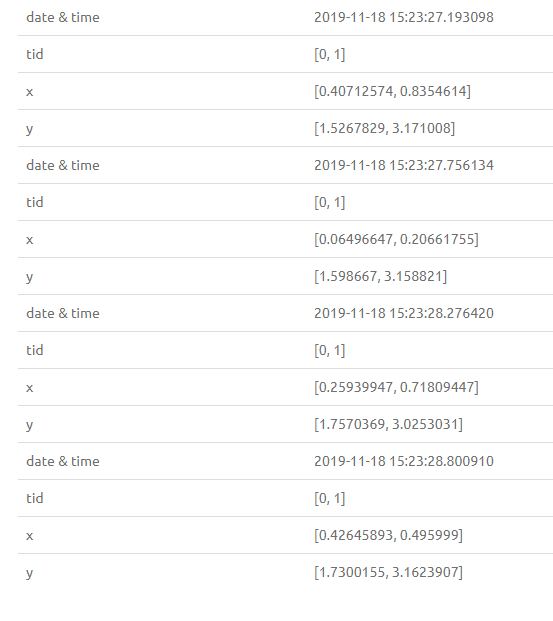
            console.log ('Error During recprd insert: ' + err);

        }

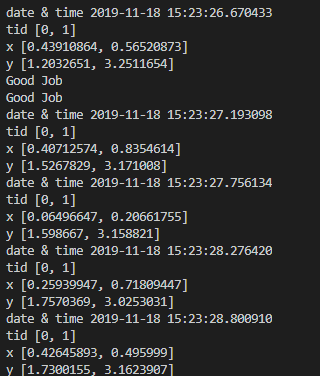
    });

}

Data received from CloudMQTT



Receive on Terminal



Store in MongoDB



**Visualize the point**

            <script>

                var xfield = 6;

                var yfield = 6;

                var xpixel = 750;

                var ypixel = 425;

                var offset = 375;

                var tid = [1, 2, 3];

                var xPosition = [2, 1, 0];

                var yPosition = [3, 2, 5];

                var canvas = document.getElementById("targetCanvas");

                var ctx = canvas.getContext("2d");

                var image = new Image();

                function init() {

                    image.src = "../images/image1.jpg";

                    window.requestAnimationFrame(draw);

                }

                function drawBall(tag\_xcale, tag\_ycale) {

                    xLocation = (tag\_xcale / xfield) \* xpixel + offset;

                    yLocation = (tag\_ycale / yfield) \* ypixel;

                    ctx.beginPath();

                    ctx.arc(xLocation, yLocation, 10, 0, 2 \* Math.PI);

                    ctx.fillStyle = "#2775f2";

                    ctx.fill();

                    ctx.closePath();

                }

                function draw() {

                    ctx.drawImage(image, 0, 0, canvas.width, canvas.height);

                    for (count = 0; count < tid.length; count++) {

                        drawBall(xPosition[count], yPosition[count]);

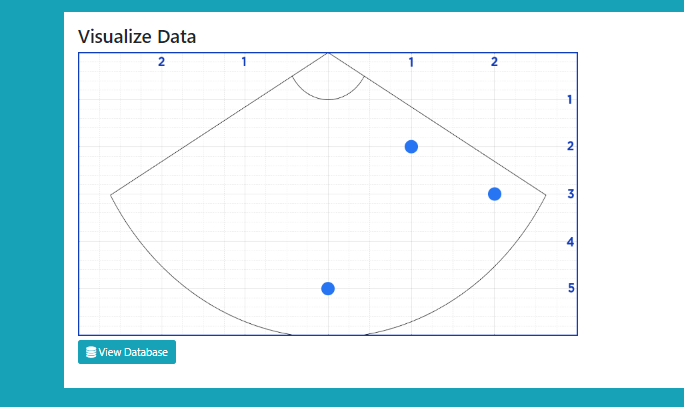
                    }

                    window.requestAnimationFrame(draw);

                };

                init();

            </script>

****