Northeastern University

Boston, Massachusetts

EECE-5698 Wireless Sensor Networks:

Dr. Francesco Restuccia

A Project

report on

mmWave Communicaion for people counting with Camera integration

Project Work
Submitted By

Shreekant Kodukula-001817642
Vaibhav Srivastava-001816739
Abhishek Sharma- 001224161

TABLE OF CONTENTS

Abstract

A type of communication, which helps us detect objects with high levels of accuracy and precision. It does not involve any type of contact. The spectrum range is usually 30-300 GHz. It can operate in very less wavelengths and can penetrate even materials like plastic, concrete walls etc. Usual wavelengths in this type of communication are around 76-81 MHz. A complete mmWave radar system includes a transmit antenna (the number can vary, depending upon the type of evaluation board being used), receiving antenna (again, this also depends on the type of evaluation board being used), radio frequency (RF) components, analog to digital converter (ADC), microcontrollers (MCU's), digital signal processors (DSP's).

We integrated a camera to a mmWave sensor evaluation board and then sent the data through a raspberry pi board using a TCP/IP communication.

Objective

We can implement mmWave communication using a specific type of radar, and the evaluation boards, which are on offer at Texas Instruments, make use of a frequency modulated continuous wave (FMCW) radar. It helps us measure the angle and the velocity. This usually differs from the traditional radars, which transmit pulses periodically and not continuously like the FMCW.

The main RF components of FMCW radar are a synthesizer, mixer, receiving antenna and a transmitting antenna. The radar operates as follows:

The chirp signal is generated with the help of a synthesizer.

• Then it is transmitted by a transmit antenna.

• It gets reflected from the objects, generating a reflected chirp.

• Then it is captured by a receiving antenna.

Now we can make use of this mmWave technology and integrate it with a camera, and correlate the points of both the planes. The points for the camera axis' will be in X,Y plane, whereas the mmWave sensor will record in Z,X plane. We will need to transform the points using a transformation matrix.

Once we have the transformation matrix, we can send the data of the points through a TCP/IP socket connection.

Board Design and sub systems

IWR vs AWR Evaluation Boards for mmWave ommunication: for this type of communication, we can use the evaluation boards provided by Texas Instrumetns to conduct further tests. But for that we need to choose which type of application we are focussing our research on and what components are needed for it.

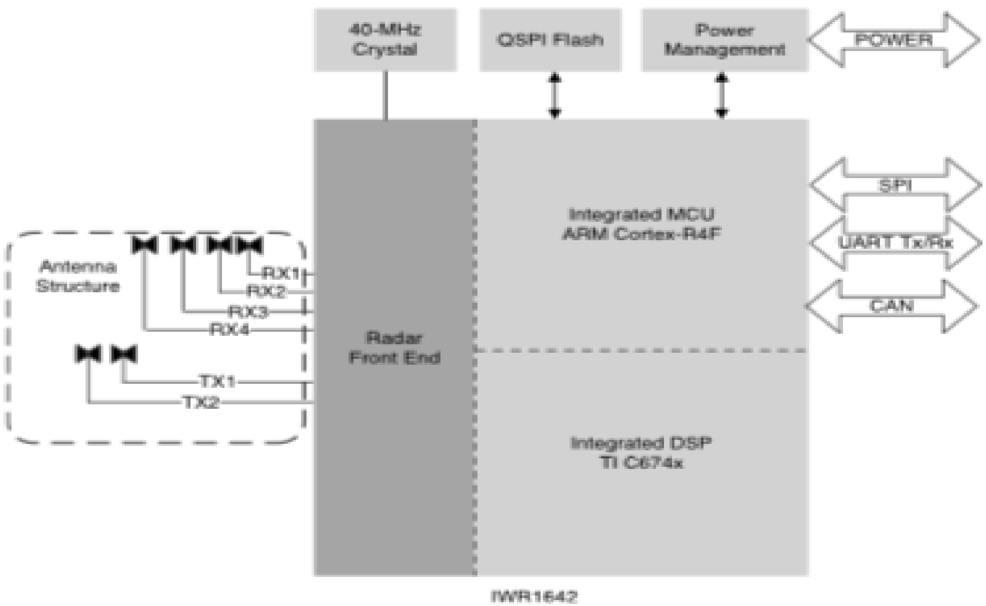


Fig 2: IWR 1642 evaluation Board design.

There are two types of boards available at Texas instruments, namely IWR, which is for industrial applicatons and AWR, which is for automotive applications.
The main notable differences between the two types of sensors are:

- AWRxxxx device support wider range of temperature junctions, as compared to IWRxxxx. -40 to 125C as compared to -40 to 105C.
- AWR1642 has two CAN inerfaces, where as IWR1642 only has one CAN inteface. The second interface on AWR1642 is CAN-FD, which is used for additional memory and faster data rates.

Design specifications of IWR1642: The evaluation board consists of various sub systems like transmit sub suystem, DSP sub system, which has a digital signal processor C674x, Microcontroler sub system, which has a microcontroller Arm Cortex-R4F.
The DSP subsystem usually is used to perform operations like FFT. MSS subsystem is used to send messages to other subsystems in order to schedule tasks and perform operations. In IWR1642, there are a totol of 2 transmitting antennas and 4 receiving antennas.

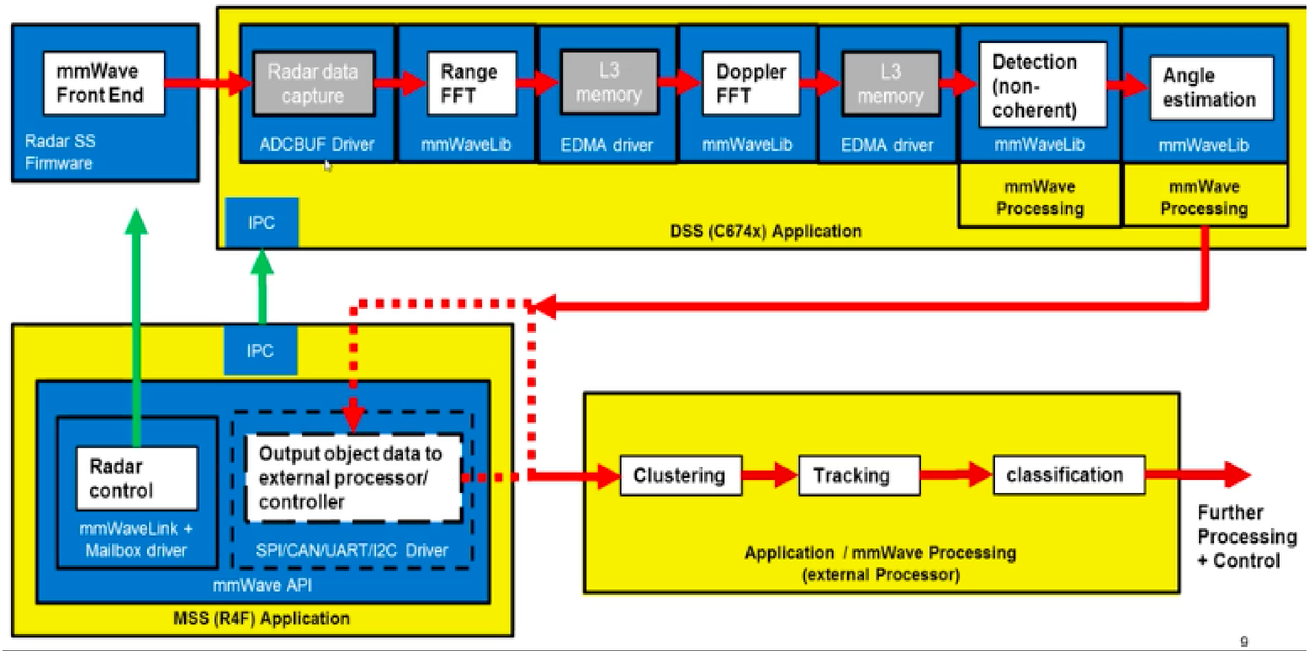


Fig 3: Processing chain in IWR1642.

Range measurement: In FMCW radar, the range is measured with the help of a chirp signal. Chirp signal is a signal, whose frequency increases linearly with time. As we can see from figure 1, the frequency increases with time. FMCW radar uses these kinds of signals and then the reflections from objects, when chirp signals are transmitted are utilized in localizing an object.
For two sinusoidal inputs x1 and x2, where**, x1 = sin(w1t + ϕ1)** and **x2 = sin(w2t + ϕ2)**, the output **xout** has an instantaneous frequency equal to the difference of these two signals.

Range Resolution: The ability of a radar to disinguish between two different objects is called as the range resolution of a radar. But whenever two objects move closer towards the radar, it cannot detect both of those objects as two different ones. This problem can be solved by increasing the length of the IF signal. It has some consequences though, since the increase in the IF signal will result in increase in the bandwidth as well, and the spectrum will have two different peaks. This concept is stated by the Fourier Transform Theory. The phase change is usually due to the differential change in distance. The phase change can be derived using the equation 1: $\Delta\boldsymbol{\phi} = \frac{\boldsymbol{2\pi\Delta d}}{\boldsymbol{\lambda}}$. Thus the angle of arrival θ can be computed with the help of $\Delta\phi$ with the help of this equation 2: $\boldsymbol{\theta = \sin^{-1}\frac{\lambda\Delta\phi}{2\pi l}}$.

Maximum angular field of view: It is usually the maximum Angle of arrival, that a radar can measure. The maximum unambiguous range of angle requires the $\Delta\boldsymbol{\omega}$ **< 180.** This corresponds to the equation $\frac{\boldsymbol{2\pi l \sin(\theta)}}{\boldsymbol{\lambda}} < \boldsymbol{\pi}$**.** The maximum field of view for two antennas placed length, l apart is: $\boldsymbol{\theta max = \sin^{-1}\frac{\lambda}{2l}}$. So the maximum field of view can be ahcieved with an angle of 90 degrees.

Velocity Measurement

The measurement of velocity by an FMCW radar is done by transmitting two chirps separated by a T_c (inter-chirp duration). The range FFT of both the chirps is calculated. The phase difference in the Range-FFT of the chirps is utilized to calculate the velocity of the object as follows:

$$\Delta\Phi = 4\pi v T_c/\lambda$$
$$\text{Hence } v = \Delta\Phi\ \lambda\ /\ (4\pi T_c)$$
$$v(max) = \lambda/4T_c$$

After our detailed research regarding the algorithms needed for the working of different lab experiments, we found some coding files, which are necessary for every example. The coding files are categorized based on the components attached inside the IWR1642 EVM, namely MSS and DSS. Here MSS refers to microcontroller subsystem and DSS refers to DSP (digital signal processing) sub system.

Inside the DSS folder, there are .cc files related to the communication (to MSS) and working. Similarly, vice versa for the MSS folder.

- DSS Sub sytem

  - dssmain.cc

    It has coding for the tasks running for a particular lab, and in this case, for the people counting experiment.Tasks include mmwave_dssinit_task, mmwave_dss_DataPath_task, mmwave_dssmmwave_control_Task.

    It also has two interrupts defined, which are chirphandler interrupt and framestart handler.

  - dss_mmwave.h

    It has a total of 3 states defined:

    Dss_state = 0; (after the data path is initialized).

    Dss_state_started;

    Dss_state_stopped;

    It also has a number of configuration events:

    Configevt;

    Startevt;

    Closeevt;

    Stopevt;

    Chirpcounterevt;

  - dss_config_edma_util.cc

    It has the configuration for four types of EDMA:

    Type 1

Type 1a
Type 2a
Type 3

- <u>dss_data_path.cc</u>
  Here inside this file, some parameters are defined such as the buffer size, scratch size etc.
  MSS_ADC_Buffer size = 0x4000U;
  DSS_L2_scratch_size = 0x1000U;
  MMW_L1_scratch size = 0x4000U;
  DSS_L2_Buffer_size = 0x3000U;
  DSS_L3_RAM_buff = 0x4000U;

  Also we can also infer from this file about the functions defined in order to detect 1D and 2D data.
  VoiddataPathwait1DInputData;
  VoiddataPathwait1DOutputData;
  VoiddataPathwait2DInputData;
  VoiddataPathwait2DOutputData;

- <u>mmw_config.h</u>
  It has a structure defined in order to define the system clock frequency, logging baud rate, common baud rate etc.

- <u>mmw_messages.h</u>
  It contains all the messages, which are exchanged between the MSS to DSS such as:

  MmwDEMO_MSS2DSS_GUI_CFG = 0xFEED0001;
  MmwDEMO_MSS2DSS_CFAR_CFG;
  MmwDEMO_MSS2DSS_DOA_CFG;
  MmwDEMO_MSS2DSS_DBSCAN_CFG; MmwDEMO_MSS2DSS_TRACKING_CFG;
  MmwDEMO_MSS2DSS_DETOBJECT_SHIPPED;
  MmwDEMO_MSS2DSS_SET_DATALOGGER;
  MmwDEMO_MSS2DSS_ADCBUFFER_CFG;

- <u>radar_process.h</u>
  We can infer about the number of points, transmitters and receivers count etc.

  Number of points = 250;
  Tx count = 2;
  Rx count = 4;

Experiment performed

For the better understanding of concepts, we performed one lab experiment in order to test out the mmWave accuracy and get a better understanding of the evaluation board.. The experiment we performed is listed below:

High accuracy detection:

Here in this example, the implementation of zoom FFT is done to detect objects within sub mm accuracy. In this lab demonstration, we made use of IWR1642, which demonstrates the capability of detecting a single peak inside the range specified by the user. The detection is done with an SNR of around 57 dB. Zoom FFT Algorithm is described below:

Here in this configuration, there is one transmitting and one receiving antenna. There are multiple chirps supported, which are accumulated before the coarse range FFT. A N size FFT can be re-expressed as:

Step1: N1 number of size-N2 FFTs,

Step 2: followed by additional twiddle multiplication,

Step 3: then N2 number of size-N1 FFTs, if N can be factorized as N = N1 x N2.

Memory optimized design algorithm:

Only peak object region has zoom FFT of size N, no need to generate complete

Generate 2 sets of twiddle: fine□□: □−□□□□/□ and Co□□□□□□: □−□□□□/□ for *k=0 N -1*

N=N1*N2 number of twiddle factor 1 *1*

The basic operation becomes finding the indices to the fineTw array and index to the CoarseTw array, then multiply a set of twiddles from table look-up and then multiply to the input signal.

Mainly AND and SHIFT operations are used in order to calculate the twiddle array.

Trade offs and Considerations: Since we made use of IWR1642, there are some tradeoffs as compared to other models of evaluation boards:

- The FFT size is 16K in 14xx models, whereas the size of FFT in 16xx is 512 x 512 = 256k because of the presence of a DSP.
- The power consumption is more in 16xx models as compared to the 14xx models.
- Overhead is more in 16xx models, as compared to 14xx models.

People counting

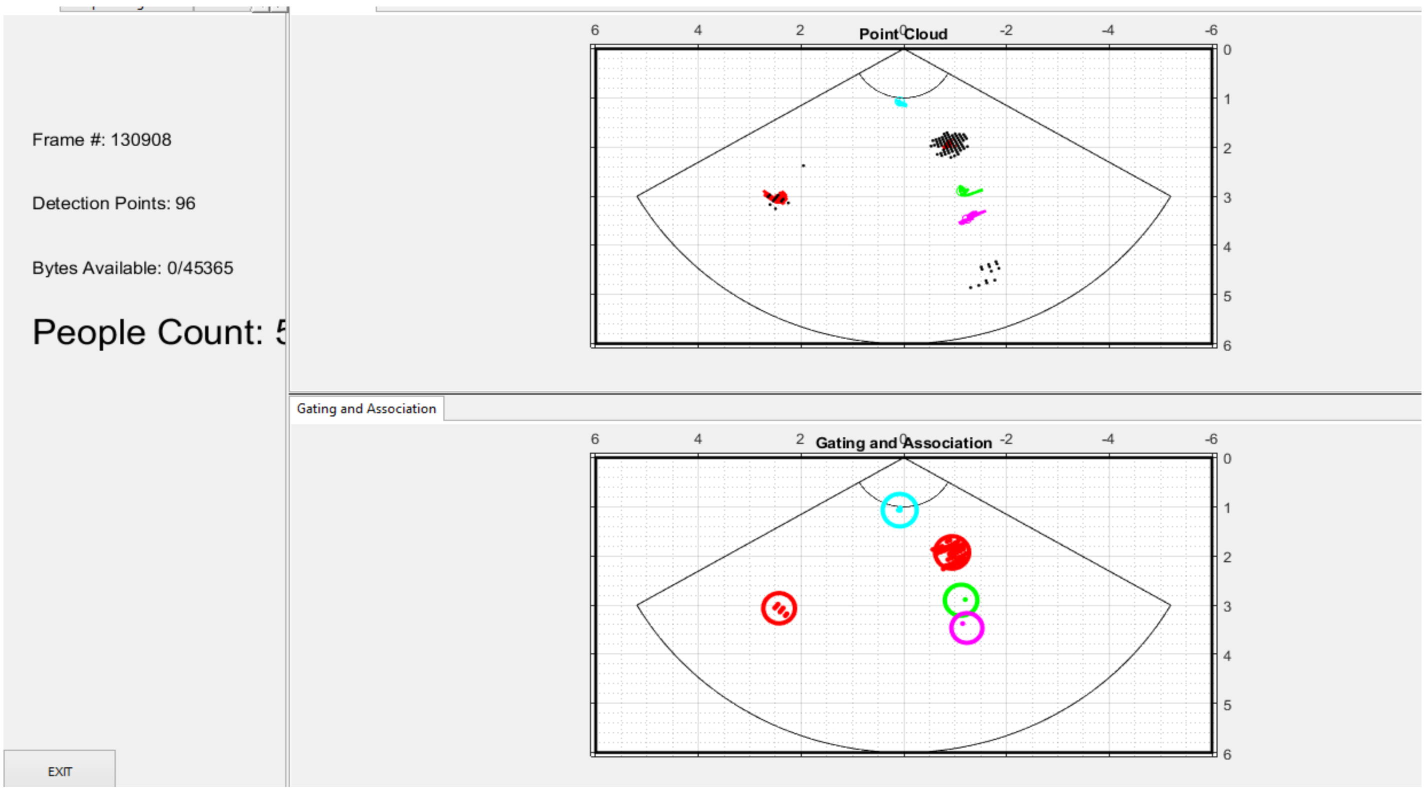


Fig: MATLAB people counting output

Processing Chain

In IWR1642 programming, there is a processing chain, in order for all the components to work accordingly with each other.

• Front end: it represents all the antenna and RF transceiver responsible for the implementation of FMCW radar.

• ADC: The ADC output samples are buffered in ADC output buffers for access by the digital part of the processing chain.

• EDMA controller: Its usually user programmed. The main purpose DMA is to move data from one memory location to another without using a secondary processor.

• C674 DSP: This is the digital signal-processing core that implements the configuration of the front end and executes the main signal processing operations on the data. This core has access to several memory resources as noted further in the design description.

• ARM R4F: This ARM MCU can execute application code including further signal processing operations and other higher-level functions. In this application the ARM R4F primarily relays visualization data to the UART interface. There is a shared memory visible to both the DSP and the R4F.

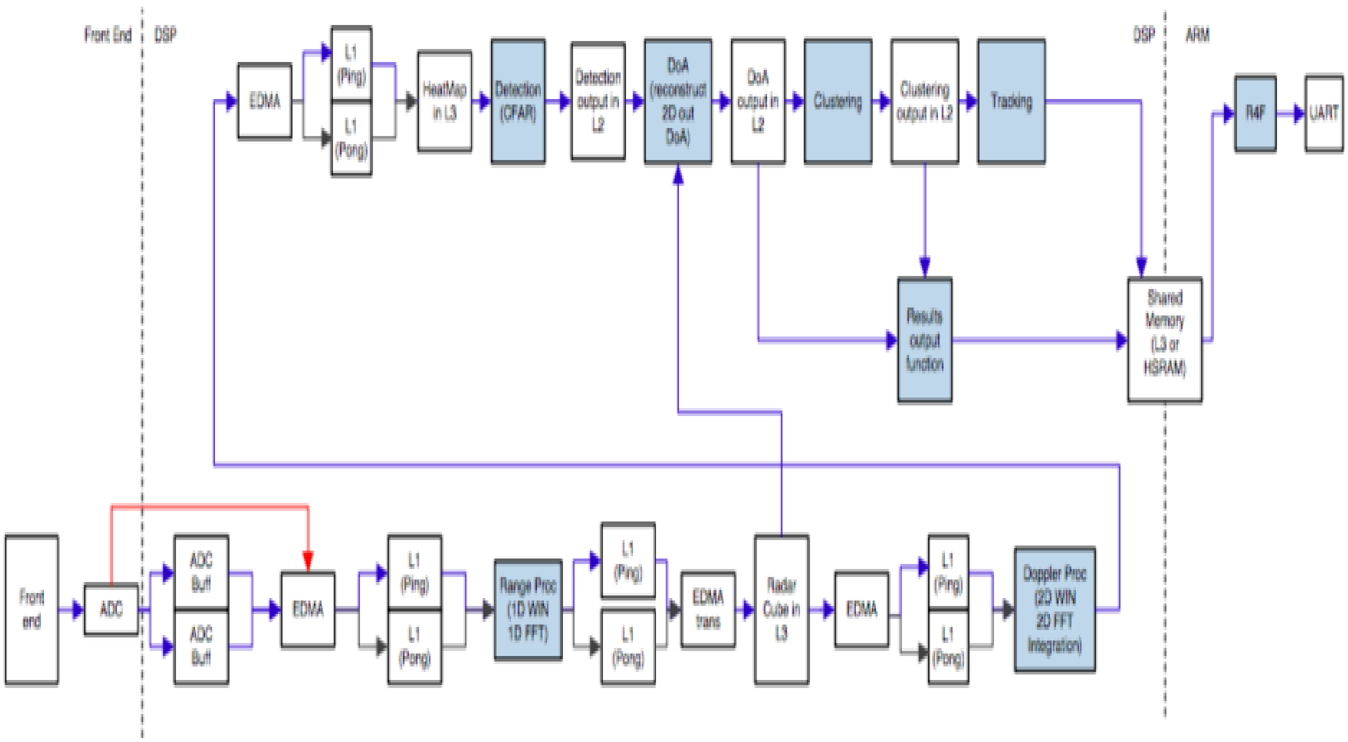


Fig: Processing chain for IWR 1642 sensor board.

The implementation of the traffic monitoring example in the signal-processing chain consists of the following blocks implemented as DSP code executing on the C674x core in the IWR1642:

• Range processing: For each antenna, EDMA is used to move samples from the ADC output buffer to DSP's local memory. A 16-bit, fixed-point 1D windowing and 16-bit, fixed-point 1D FFT are performed. EDMA is used to move output from DSP local memory to radar cube storage in layer three (L3) memory. Range processing is interleaved with active chirp time of the frame. All other processing happens each frame, except where noted, during the idle time between the active chirp time and the end of the frame.

• Doppler processing, antenna combining: For each antenna, EDMA transfers data between radar cube in L3 and DSP local L2 memory. The DSP operations are, 16-bit fixed point 2D windowing, formatting from 16-bit fixed-point IQ to floating-point IQ, floating-point 2D FFT, and non-coherent combining of received power across antennas in floating point. The output range-Doppler power signal or heat map is stored in L3 memory separate from the radar cube.

• Range-Doppler detection: An algorithm is applied to the range-Doppler power mapping to find detection points in range and Doppler space. The algorithm consists of a first pass along range axis using cell averaging smaller of (CASO) CFAR, and a second pass along Doppler axis using cell averaging (CA) CFAR. Due to the data access pattern, the detection code accesses the integrated signal in L3 memory through the L1D cache. The output detected point list is stored in L2 memory.

• Angle estimation: For each detected point in range and Doppler space, the input to angle estimation is reconstructed by recomputing per antenna Doppler data from radar cube and applying Doppler compensation for TDM-MIMO. A beamforming with multi-peak search algorithm returns one or more target angles for each input range-Doppler point. X and Y coordinates can then be calculated and stored, along with the Doppler and other attributes for each target point. The output is stored in the L2 memory.

• Clustering: Detected points are accumulated over four frames. Every fourth frame DBSCAN, a point density based clustering algorithm, is run. The clustering operation designates one or more subsets of the accumulated target points. Each of these subsets is interpreted as containing points belonging to the same object such as a vehicle. Other points not meeting the clustering criteria are left unassociated. This output data is stored in the L2 memory.

- Tracking: The output of clustering is fed to a basic tracking algorithm, which also updates every fourth frame. The basic tracking algorithm maintains state information for one or more tracked objects. The update operations consist of cluster association, track management, and Kalman filter based tracking of objects. During the update process, tracks may be created or deleted, and existing tracks are updated to include estimated object position, velocity, and other attributes.

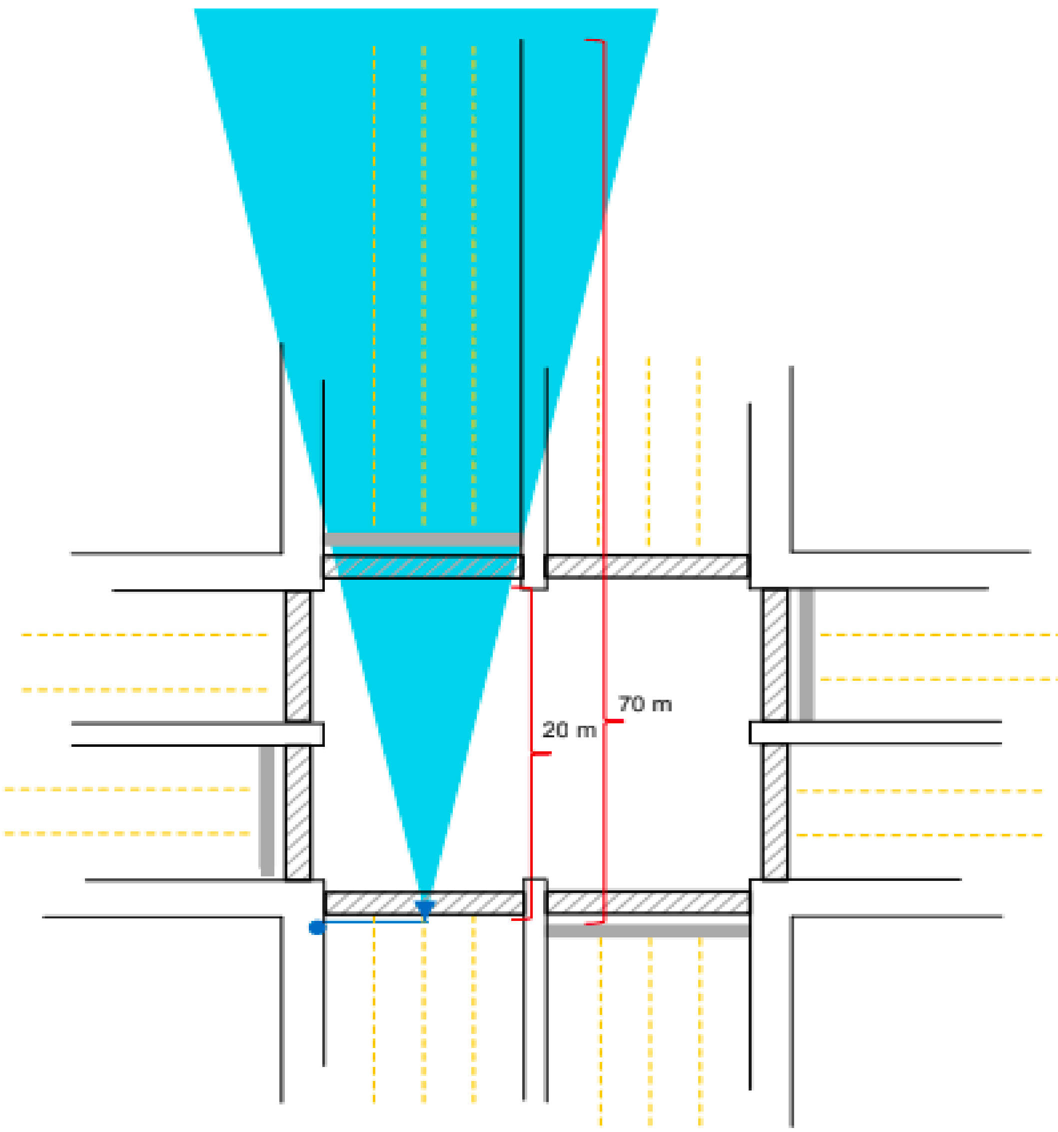


Fig: Range and angle of the covered area.

Clustering and Tracking Algorithms

Object detection and tracking are critical steps in understanding and analyzing the environment. DBSCAN is used, which is a distance-based clustering algorithm combined with Kalman filtering to track the objects.

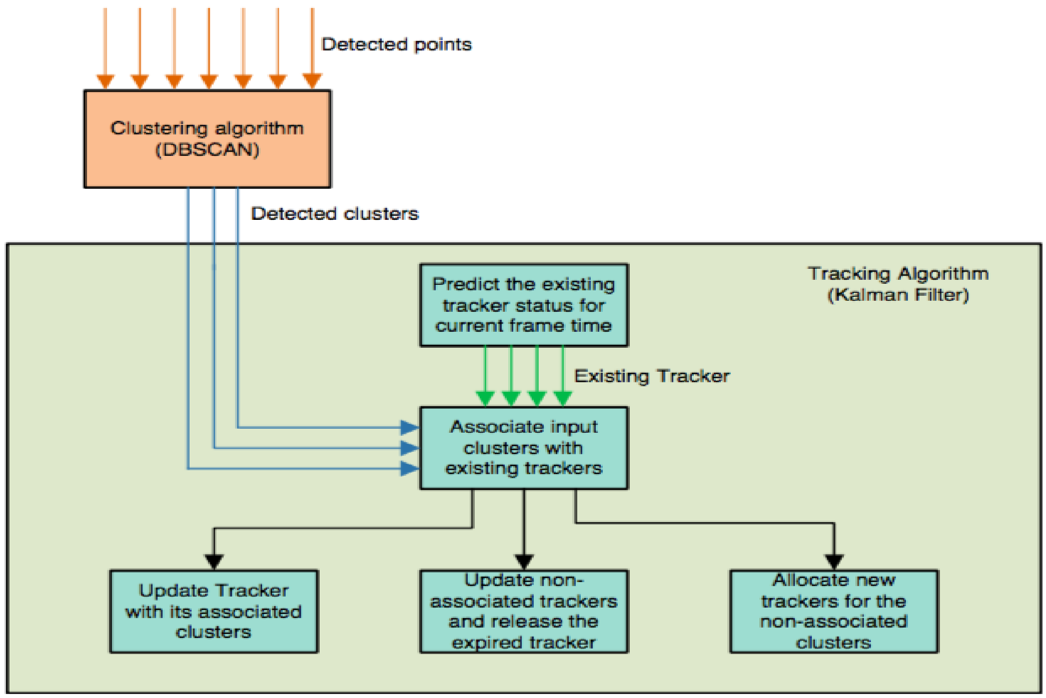


Fig: The Clustering and Tracking algorithm.

Clustering Algorithm

The clustering algorithm is used to separate the points based on whether they are closer in both space and speed. The distance metric is computed between the target point *a* and all other candidates. This distance metric is then compared with the adjusted distance threshold Epsilon to decide whether candidate *b* is qualified to join the *a-group*.

$$|a_{loc} - b_{loc}|^2 + weight' \, |a_{vel} - b_{vel}|^2 < Epsilon + weight' \times \min\left(|a_{vel}|, vFactor\right)^{\wedge 2}$$

Epsilon: The maximum space distance to be included in the cluster group

Weight: The weight between the space distance and speed differences in the distance matrix.

minPoints: Minimum Points in the group to claim a cluster.

vFactor: maximum speed which is to be added to the epilson.

REQUIRED PARAMETER	**MEANING**
xCenter	Average of x-location among all points in the cluster
yCenter	Average of y-location among all points in the cluster

xSize	Maximum delta between the x-location and xCenter
ySize	Maximum delta between the y-location and yCenter
avgVel	Average radial velocity for all points in the cluster
centerRangeVar	Average range variance among all points in the cluster
centerAngleVar	Average angle variance among all points in the cluster
centerDopplerVar	Variance of velocity among all points in the cluster

Tracking Algorithm

The clustering outputs are detected by the Kalman filter, and appear amongst different frames. The location and velocity of points X and Y are tracked through the Kalman filter. The input to the Kalman filter is the clustering algorithm. One important consideration for this filter implementation is the distance metric, between the clusters and existing trackers. The candidate closest to a particular cluster will be associated to that cluster. As long as the cluster center falls within the trackers range, it will be associated. The distance between both of them should satisfy this condition:

$$|LOC_{tracker} - LOC_{cluster}|^2 < (trackerAssociationThreshold^2 + 4' \text{ x } xSize^{2}_{tracker} + 4' \text{ x } YSize^2_{tracker})$$

If there arises a case that multiple clusters are associated with the same tracker, the combined cluster will be computed and used to update the tracker. The tracker without any association at this time will be updated on their status and has a potential risk to be expired. The cluster without any association will have a new tracker allocated. The expired tracker will be collected and reused later during new tracker allocation.

REQUIRED PARAMETER	MEANING
trackerID	Tracker identification
state	Tracker status
S_hat	[x_loc, y_loc, x_vel, y_vel]
xSize	Object size on x-direction
ySize	Object size on y-direction

Camera Integration algorithm

Use of Raspberry Pi –

We are using arducam embedded camera, which uses Camera Serial Interface to send the data to the microcontroller. Camera can take video frames at resolution of 640x480 RGB format and if we want to live stream at 30 FPS we want some powerful microcontroller. Raspberry Pi 3 is a very powerful mini computer, which also has support for embedded cameras. Raspberry Pi 3 has system call which supports frame capture from a camera we use the system call to capture the frames and send it over the network. But we need all our sensors and the remote server in the same network. So we create a network using the Raspberry Pi 3 as a Wifi Host and connect all other sensor to this network and then live stream from raspberry pi 3 on this network. Below are the steps for configuring raspberry pi 3 as Wifi Host.

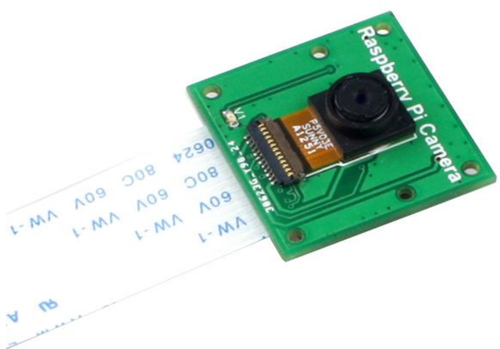


Fig: Adrucam camera module.

Configuration of Raspberry Pi as Access Point in the network -

1. Steps to configure raspberry pi as Wifi Hotspot :
   sudo apt-get install dnsmasq hostapd
2. Since the configuration files are not ready yet, turn the new software off as follows:
   sudo systemctl stop dnsmasq
   sudo systemctl stop hostapd

3. Configuring a static IP - We are configuring a standalone network to act as a server, so the Raspberry Pi needs to have a static IP address assigned to the wireless port. We will assign the server the IP address 192.168.4.1 and the wireless device being used is wlan0. To configure the static IP address, edit the dhcpcd configuration file with:
   Append - interface wlan0
   static ip_address=192.168.4.1/24 to the /etc/dhcpcd.conf file.
4. Configuring the DHCP server (dnsmasq) -
   Append - interface=wlan0
   dhcp-range=192.168.4.2,192.168.4.20,255.255.255.0,24h to /etc/dnsmasq.conf.

5. Configuring the access point host software (hostapd) : Edit the /etc/hostapd/hostapd.conf file to configure
   the SSID and Passphrase.
6. Append DAEMON_CONF="/etc/hostapd/hostapd.conf" into /etc/default/hostapd file.
7. Start the configurations again -

        sudo systemctl start hostapd
        sudo systemctl start dnsmasq

Camera -

The Camera which we am using is from Adru-cam which has 5 MegaPixel sensor and is capable of doing 640x480 at 60FPS. There are many ways to live stream on a network but I choose cvlc to do the network streaming for the frames I capture from the raspberry pi embedded camera. Raspbian is the OS which is supported by the raspberry pi 3. I use the raspbian stretch lite, which is only command line version of debian modified to support raspberry pi. We used a system call natively supported by the raspberry to capture frames from raspberry pi. The command is as follows –

<u>raspivid -o -hf -w 640-h 480 -fps 24</u>

To stream the frames over the network I redirected the STDOUT of the raspivid process to the STDIN of cvlc and started the streaming using over rstp protocol. Other supported protocols were HTTP and others. But In my testing rstp protocol was giving 5 more frames per second when compared to HTTP. The reason for using cvlc was that it compressed frames in h264 codec and Opencv has support for h264 codec. On the remote server we use Opencv to capture the live stream and process the live stream using the people detector model which we create in keras to detect people

Calibrations –

Camera Calibration -

As we later want to do tracking of people we would like to know the intrinsic parameters. Intrinsic Parameters are camera parameters that are internal and fixed to a particular camera/digitization setup. These allow a mapping between camera coordinates and pixel coordinates in the image frame and we will do tracking in the camera frame so we would like to project the pixel location (u,v) to location X, Y in the camera frame.

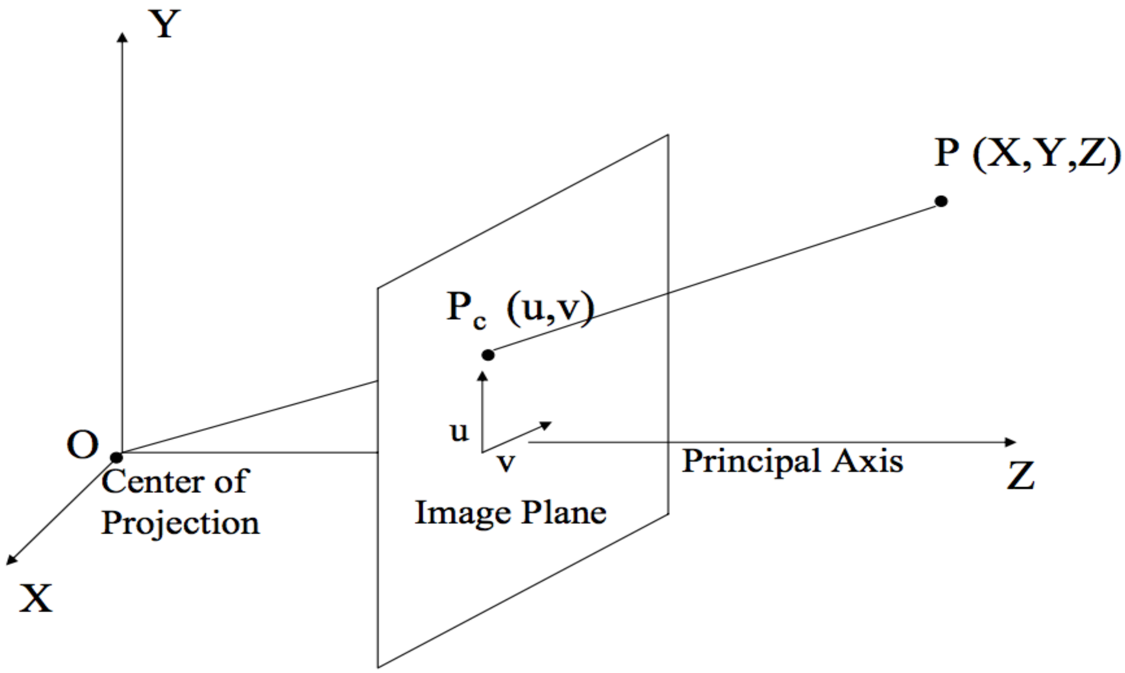


Figure: Image plane and point (X,Y,Z) in the camera frame.

To get the focal length of the camera we need to do the camera calibration. We use MATLAB to calibrate the camera by capturing 50 images of checkerboard of size 7x11 23.8 mm having black and white boxes. Few examples of the captured images are as follows –

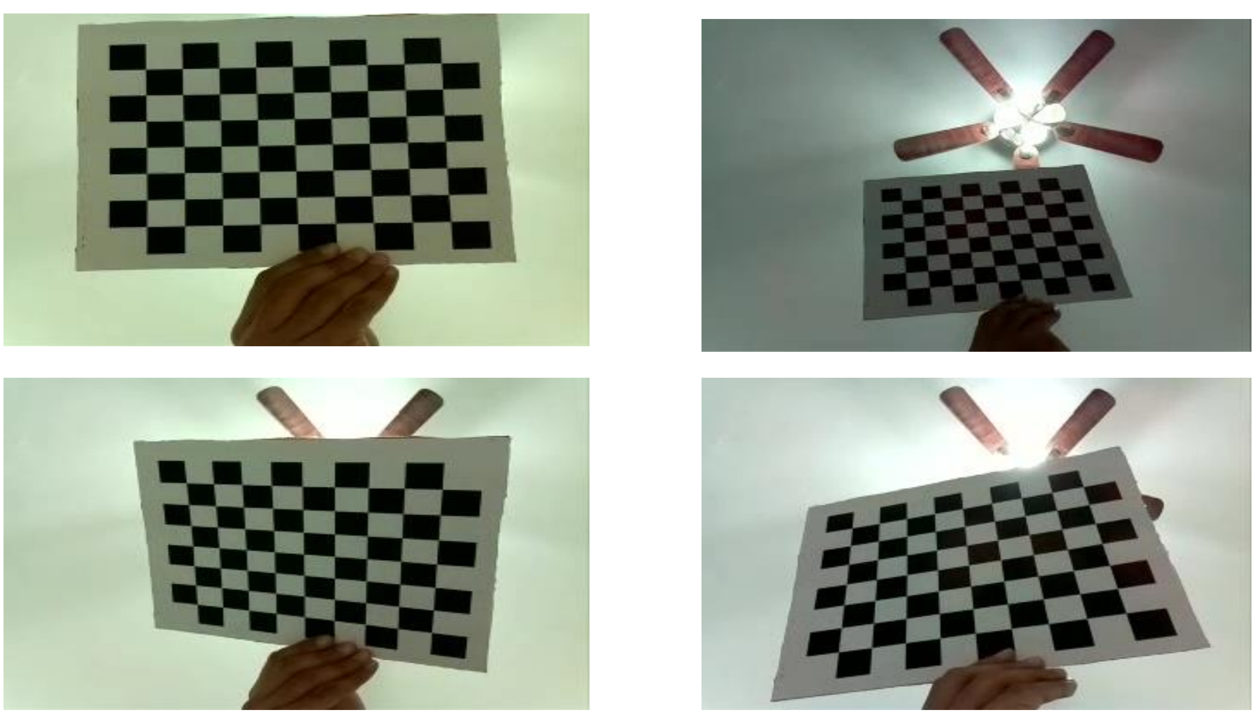
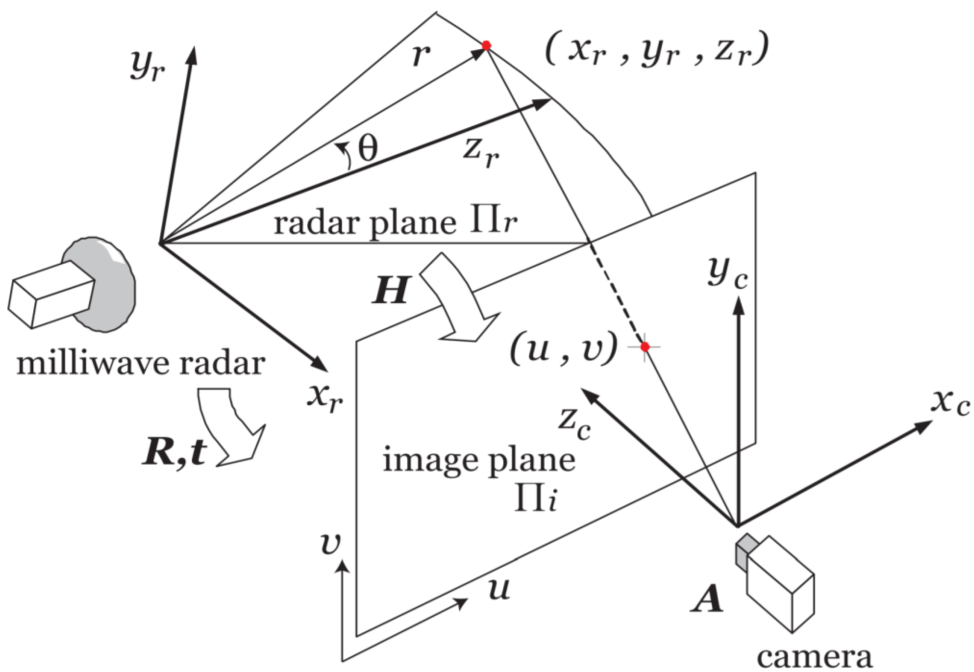


Figure : Example of checkerboard images to calibrate the camera.

After the camera calibration we get the intrinsic matrix as follows –

Focal Length: fc = [323.56520   302.97345] +/- [2.89704   2.64279]
Principal point: cc = [141.38520   120.80802] +/- [2.61234   2.25056]
Skew: alpha_c = [0.00000] +/- [0.00000]   => angle of pixel axes = 90.00000 Distortion:
kc = [0.16855   -0.51249   0.00099   -0.00771 0.00000] +/- [0.02314   0.08240   0.00297   0.00301 0.00000]
Pixel error: err = [0.20614   0.17077]
Now we can use the Focal Length in the transformations later.

Camera and mmWave Radar calibration -



We suppose that the radar scans in a plane, called the 'radar plane'. As shown in figure above (Xr, Yr, Zr) and (Xc, Yc, Zc)
be the radar and the camera coordinates respectively, and (u, v) be the image plane coordinates. Using homogeneous coordinates, we can describe the equation of transformation between (Xr, Yr, Zr, 1) and (u, v, 1). The equation between
this transformation can be given as -

$$[u\ v\ 1]^T = \mathbf{P}\ [Xr\ Yr\ Zr\ 1]^T$$

Where **P** is **A** [**R | t**]. In the above equation, the 3x3 matrix R and the 3x1 vector t denote, respectively, the rotation and translation between the sensor coordinates. The 3x3 matrix A denotes intrinsic camera parameters which was obtained
using the calibration shown in the previous section. Generally, calibration between the two sensors requires estimation of
the 3x4 matrix **P**, or all of the R,t,and A. On the contrary we use the transformation between the radar plane

Πr and the

image plane Πi. Considering that all radar data come from somewhere on the radar plane Yr = 0, the equation $[u\ v\ 1]^T$= **P** $[Xr\ Yr\ Zr\ 1]^T$ can be written as follows –

$$[u\ v\ 1]^T = \mathbf{H}\ [Xr\ \ Zr\ 1]^T$$

Where **H** is the 3x3 homography matrix. By estimating the H, the transformation between the radar plane Πr and the image plane Πi is determined without the need of **P**. We use the least squared estimation using seven data set of (u, v) and (Xr, Zr) to get an estimate of **H.**

The **H** we got using this estimation is =

$$H = \begin{matrix} -122.8303 & -0.2614 & 253.7836 \\ 9.0520 & 12.0507 & 187.7295 \\ 0.0000 & 0.0000 & 1.0000 \end{matrix}$$

Now once we have the point detected by the radar we can project the points (Xr, Zr) into the image to check if the point is a person or not. This is because using the mmWave radar we get a lot of false positives which can be refined if we use a camera. Camera uses visual features to detect people and its detection are much more reliable than mmWave sensor.

Now when we project the detection in the image plane then we can also know the distance of people from the camera, which we got from the radar.

People Detection in Camera -

To detect people in the camera I use Deep learning. We referenced the paper [6], fro further help. You Only Look Once (YOLO) is an architecture designed for object detection in images. YOLO reframes object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities.

A single convolutional network simultaneously predicts multiple bounding boxes and class probabilities for those boxes. YOLO trains on full images and directly optimizes detection performance. This unified model has several benefits over traditional methods of object detection. First, YOLO is extremely fast. YOLO simply runs neural network on a new image at test time to predict detections. YOLOs base network runs at 45 frames per second with no batch processing on a Titan X GPU and a fast version run at more than 150 fps. In our case we were able to run at 12FPS on GPU 940mx, which only has compute capability of 5.1.

Working example of YOLO -

YOLO divides up the image into a grid of 13 by 13 cells. Each of these cells is responsible for predicting 5 bounding boxes. A bounding box describes the rectangle that encloses an object.

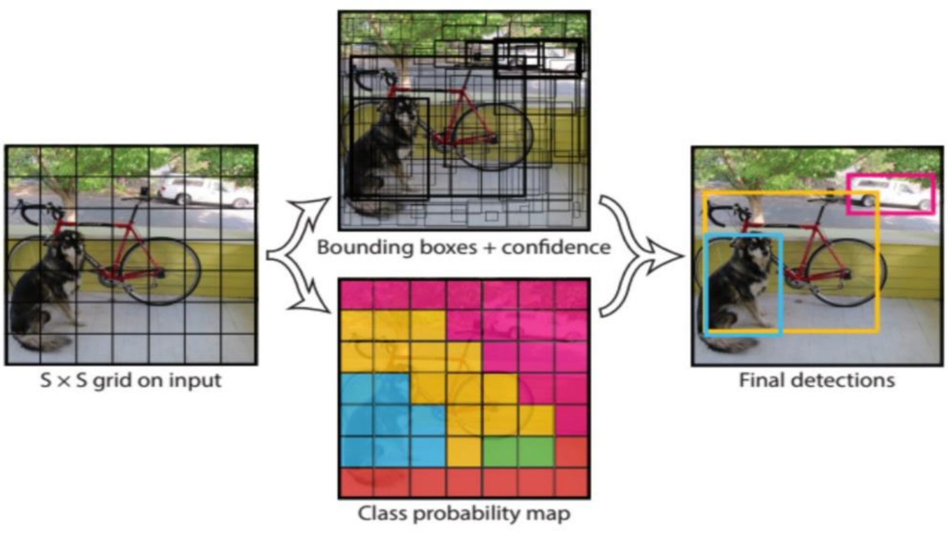
YOLO also outputs a confidence score that tells us how certain it is that the predicted bounding box actually encloses some object. This score doesn’t say anything about what kind of object is in the box, just if the shape of the box is any good.

The confidence score for the bounding box and the class prediction are combined into one final score that tells us the probability that this bounding box contains a specific type of object.

Since there are 13×13 = 169 grid cells and each cell predicts 5 bounding boxes, we end up with 845 bounding boxes in total. It turns out that most of these boxes will have very low confidence scores, so we only keep the boxes whose final score is 70% or more.

The architecture is as follows -

Layer	kernel	stride	output shape
Input			(416, 416, 3)
Convolution	3×3	1	(416, 416, 16)
MaxPooling	2×2	2	(208, 208, 16)
Convolution	3×3	1	(208, 208, 32)
MaxPooling	2×2	2	(104, 104, 32)
Convolution	3×3	1	(104, 104, 64)
MaxPooling	2×2	2	(52, 52, 64)
Convolution	3×3	1	(52, 52, 128)
MaxPooling	2×2	2	(26, 26, 128)
Convolution	3×3	1	(26, 26, 256)
MaxPooling	2×2	2	(13, 13, 256)
Convolution	3×3	1	(13, 13, 512)
MaxPooling	2×2	1	(13, 13, 512)
Convolution	3×3	1	(13, 13, 1024)
Convolution	3×3	1	(13, 13, 1024)
Convolution	1×1	1	(13, 13, 125)



<u>Fig: YOLO model</u>

The original model was created in own custom darknet framework, which was in c language. But we implemented the Yolo model using Keras Library in python, which uses Tensorflow as backend. Then we converted the weights from the original framework into the sensor flow and use those weights to start the detections. Once we are able to get some detection I refine the detection and project the bounding box over the image.

Results and conclusions

Fig: Final working model

The model with a mmWave sensor evaluation board and a raspberry pi with camera module has been implemented. Algorithms for clustering like DBSCAN, Kalvan filter for tracking and YOLO for camera module integration were implemented in MATLAB and Python respectively.

Challenges faced and future work

Challenges faced:

- Better equipment for more accurate results.
- Transforming the points between camera and sensor planes.
- False detections during clustering.
- mmWave sensor being oversensitive to the environment conditions.

Future Work

- Parking sensor algorithm in terms of timestamp.
- Implementation in 3-D.

References

1. http://www.ti.com/lit/wp/spyy005/spyy005.pdf
2. http://www.ti.com/lit/ds/symlink/iwr1642.pdf
3. https://e2e.ti.com/support/sensor/mmwave_sensors/f/1023/t/601728?AWR1642-Can-t-choose-between-AWR-and-IWR-range
4. http://www.ti.com/product/IWR1642/technicaldocuments
5. https://training.ti.com/introduction-mmwave-sdk
6. https://pjreddie.com/media/files/papers/yolo.pdf.

Appendix

Code for People counting algorithm (IWR1642):

```
clear, clc
t = tcpip('192.168.4.16', 10068);
fopen(t);
%setting the scene
sceneRun = 'GUI_Setup';

% 2. Specify COM ports
if (~strcmp(sceneRun,'GUI_Setup'))
   %%%%% EDIT COM PORTS %%%%%%
   controlSerialPort = 10;
   dataSerialPort = 11;
   loadCfg = 1;
end
%% Setup tracking scene

% Enables setting parameters by GUI
if(strcmp(sceneRun,'GUI_Setup'))

   % Call setup GUI
   hSetup = setup();
   close(hSetup.figure1);
   % Get parameters defined in GUI
   camIndex = hSetup.camIndex;
   hDataSerialPort = hSetup.hDataSerialPort;
   hControlSerialPort = hSetup.hControlSerialPort;
   wall = hSetup.wall;
   scene.azimuthTilt = hSetup.angle*pi/180;
   Params = hSetup.params;
   % Target box settings
   scene.numberOfTargetBoxes = 0; % Box counting not enabled for this mode

   % Define wall [BLx BLy W H]
   scene.areaBox = [wall.left wall.back abs(wall.left)+wall.right wall.front+abs(wall.back)];

   % Define plotting area [Lx Rx By Ty]
   scene.maxPos = [scene.areaBox(1)-0.1 scene.areaBox(1)+scene.areaBox(3)+0.1
scene.areaBox(2)-0.1 scene.areaBox(2)+scene.areaBox(4)+0.1];

   % Chirp config
   loadCfg = 0; %disabled because loaded in GUI
end
```

```
% Programmatically set scene. Includes example of setting boundary boxes to count in
if(strcmp(sceneRun,'Prgm_2box'))

    %Read Chirp Configuration file
    configurationFileName = 'mmw_pc_128x128_2box.cfg';
    cliCfg = readCfg(configurationFileName);
    Params = parseCfg(cliCfg);

    % Room Wall dimensions [m]
    % Measured relative to radar
    wall.left = -6; % signed: - required
    wall.right = 6;
    wall.front = 6;
    wall.back = -0; % signed: - required

    % define wall [BLx BLy W H]
    scene.areaBox = [wall.left wall.back abs(wall.left)+wall.right wall.front+abs(wall.back)];

    % Define two rectangles for specific counting in the region
    % Target box settings
    scene.numberOfTargetBoxes = 2;

    % Parameters to make it easier to define two rectangles of the same size
       % that are side by side
    % RO = Rectangle Origin. RO is measured relative to Left Back wall corner
    box.ROxtoLB = 1.0; % x distance from left wall
    box.ROytoLB = 1.5; % y distance from back wall
    box.height = 2;    % height of boxes
    box.sep = 0.6;     % seperation width of boxes
    box.width = 1;     % width of boxes
    box.RTXplot = box.ROxtoLB+wall.left;
    box.RTYplot = box.ROytoLB+wall.back;
    scene.targetBox = [box.RTXplot box.RTYplot box.width box.height;
                  (box.RTXplot+box.width+box.sep) box.RTYplot box.width box.height];


    % define plotting area as margin around wall
    margin = 0.1; %[m]
    scene.maxPos = [scene.areaBox(1)-margin ...
                scene.areaBox(1)+scene.areaBox(3)+margin ...
                scene.areaBox(2)-margin ...
                scene.areaBox(2)+scene.areaBox(4)+margin];

    % Azimuth tilt of radar.
    angle = +0; % Signed: + if tilted towards R wall, - if L, 0 if straight forward
    scene.azimuthTilt = angle*pi/180;
```

```
end


% Programmatically set scene. Includes example of setting boundary boxes to count in
if(strcmp(sceneRun,'Prgm_MaxFOV'))

    %Read Chirp Configuration file
    configurationFileName = 'mmw_pcdemo_default.cfg';
    cliCfg = readCfg(configurationFileName);
    Params = parseCfg(cliCfg);

    % Room Wall dimensions [m]
    wall.left = -6; % signed: - required
    wall.right = 6;
    wall.front = 6;
    wall.back = -0; % signed: - required

    % define wall [BLx BLy W H]
    scene.areaBox = [wall.left wall.back abs(wall.left)+wall.right wall.front+abs(wall.back)];

    % Define two rectangles for specific counting in the region
    % Target box settings
    scene.numberOfTargetBoxes = 0;


    % RO = Rectangle Origin. RO is measured relative to Left Back wall corner
    box.ROxtoLB = 1.0; % x distance from left wall
    box.ROytoLB = 1.5; % y distance from back wall
    box.height = 2;    % height of boxes
    box.sep = 0.6;     % seperation width of boxes
    box.width = 1;     % width of boxes
    box.RTXplot = box.ROxtoLB+wall.left;
    box.RTYplot = box.ROytoLB+wall.back;


    % Each row of targetBox specifies the dimensions of a rectangle for counting.
    % The # of rows of targetBox must match numberOfTargetBoxes.
    % The rectangles are specified by (x,y) coordinate and width and height
    % Custom rectangles can be defined instead if two side by side rects of
    % same size are not desired using [RTCx RTCy W H] convention
    scene.targetBox = [box.RTXplot box.RTYplot box.width box.height;
                  (box.RTXplot+box.width+box.sep) box.RTYplot box.width box.height];


    % define plotting area as margin around wall
    margin = 0.1; %[m]
```

```
    scene.maxPos = [scene.areaBox(1)-margin ...
                scene.areaBox(1)+scene.areaBox(3)+margin ...
                scene.areaBox(2)-margin ...
                scene.areaBox(2)+scene.areaBox(4)+margin];

    % Azimuth tilt of radar.
    angle = +0; % Signed: + if tilted towards R wall, - if L, 0 if straight forward
    scene.azimuthTilt = angle*pi/180;
end

if(strcmp(sceneRun,'My_Scene'))
end

%% Webcam setup
if (strcmp(sceneRun,'GUI_Setup'))
    if(~(camIndex == -1))
        enableWebcam = 1;
        cam = webcam(camIndex);
        resList = cam.AvailableResolution;
        cam.Resolution = resList{getWidestFOV(resList)};


        hWebcamFigure = figure('Name', 'Ground Truth','Tag','webcamFigure',...
            'Toolbar','none', 'Menubar','none',...
            'NumberTitle', 'Off', 'Interruptible', 'Off');
        axWebcam = axes('Parent', hWebcamFigure);
        hImage = image(axWebcam, snapshot(cam));
        axis(axWebcam, 'manual','off')


        % Set up the push buttons
        uicontrol('String', 'Play',...
            'Callback', 'preview(cam, hImage)',...
            'Units','normalized',...
            'Position',[0 0 0.15 .07]);
        uicontrol('String', 'Pause',...
            'Callback', 'closePreview(cam)',...
            'Units','normalized',...
            'Position',[.17 0 .15 .07]);
        uicontrol('String', 'Close',...
            'Callback', 'delete(hWebcamFigure)',...
            'Units','normalized',...
            'Position',[0.34 0 .15 .07]);


        axWebcam = axes('Parent', hWebcamFigure);
```

```
        hImage = image(axWebcam, snapshot(cam));
        axis(axWebcam, 'manual','off')


        res = cam.Resolution;
        ss = strsplit(res,'x');
        imWidth = str2num(ss{1});
        imHeight = str2num(ss{2});
        hImage = image( zeros(imHeight, imWidth, 3) );
        % Set up the update preview window function.
        setappdata(hImage,'UpdatePreviewWindowFcn',@mypreview_fcn);



        % Specify the size of the axes that contains the image object
        % so that it displays the image at the right resolution and
        % centers it in the figure window.
        figSize = get(hWebcamFigure,'Position');
        figWidth = figSize(3);
        figHeight = figSize(4);
        gca.unit = 'pixels';
        gca.position = [ ((figWidth - imWidth)/2)...
                ((figHeight - imHeight)/2)...
                imWidth imHeight ];


        hCam = preview(cam, hImage);
        pause(0.5); %allow webcam to load
    else
        enableWebcam = 0;
        hWebcamFigure = [];
    end
else
    %Progammatically configure webcam here
    enableWebcam = 0;
    hWebcamFigure = [];
end


%% Serial setup
if (~strcmp(sceneRun,'GUI_Setup'))
    %Configure data UART port with input buffer to hold 100+ frames
    hDataSerialPort = configureDataSport(dataSerialPort, 65536);

    %Send Configuration Parameters to IWR16xx
    if(loadCfg)
```

```
        mmwDemoCliPrompt = char('mmwDemo:/>');
        hControlSerialPort = configureControlPort(controlSerialPort);
        %Send CLI configuration to IWR16xx
        fprintf('Sending configuration from %s file to IWR16xx ...\n', configurationFileName);
        for k=1:length(cliCfg)
           fprintf(hControlSerialPort, cliCfg{k});
           fprintf('%s\n', cliCfg{k});
           echo = fgetl(hControlSerialPort); % Get an echo of a command
           done = fgetl(hControlSerialPort); % Get "Done"
           prompt  =  fread(hControlSerialPort,  size(mmwDemoCliPrompt,2));  %  Get  the  prompt
back
        end
        fclose(hControlSerialPort);
        delete(hControlSerialPort);
    end
end

%% Init variables
trackerRun = 'Target';
colors='brgcm';
labelTrack = 0;


%sensor parameters
sensor.rangeMax = 6;
sensor.rangeMin = 1;
sensor.azimuthFoV = 120*pi/180; %120 degree FOV in horizontal direction
sensor.framePeriod = 50;
sensor.maxURadialVelocity = 20;
sensor.angles = linspace(-sensor.azimuthFoV/2, sensor.azimuthFoV/2, 128);

hTargetBoxHandle = [];
peopleCountTotal = 0;
peopleCountInBox = zeros(1, scene.numberOfTargetBoxes);
rxData = zeros(10000,1,'uint8');

maxNumTracks = 20;
maxNumPoints = 250;

hPlotCloudHandleAll = [];
hPlotCloudHandleOutRange = [];
hPlotCloudHandleClutter = [];
hPlotCloudHandleStatic = [];
hPlotCloudHandleDynamic =[];
hPlotPoints3D = [];
```

```
clutterPoints = zeros(2,1);
activeTracks = zeros(1, maxNumTracks);

trackingHistStruct = struct('tid', 0, 'allocationTime', 0, 'tick', 0, 'posIndex', 0, 'histIndex', 0,
'sHat', zeros(1000,6), 'ec', zeros(1000,9),'pos', zeros(100,2), 'hMeshU', [], 'hMeshG', [],
'hPlotAssociatedPoints', [], 'hPlotTrack', [], 'hPlotCentroid', []);
trackingHist = repmat(trackingHistStruct, 1, maxNumTracks);

%% Setup figure

figHandle = figure('Name', 'Visualizer','tag','mainFigure');


clf(figHandle);
set(figHandle, 'WindowStyle','normal');
set(figHandle,'Name','Texas Instruments - People Counting','NumberTitle','off')

set(figHandle,'currentchar',' ')        % set a dummy character

warning off MATLAB:HandleGraphics:ObsoletedProperty:JavaFrame
jframe=get(figHandle,'javaframe');
set(figHandle, 'MenuBar', 'none');
set(figHandle, 'Color', [0 0 0]);
%set(figHandle, 'CloseRequestFcn', close(figHandle));
%set(figHandle, 'DeleteFcn', @close_main);
pause(0.00001);
set(jframe,'Maximized',1);
pause(0.00001);


% Background


figureTitles = {'Statistics', 'Point Cloud', 'Gating and Association', 'Chirp Configuration',
'Visualizer Options & Control'}; %, 'Heat Map', 'Feature Ext'};
figureGroup = [1, 2, 3, 1, 1];
numFigures = size(figureTitles, 2);
hFigure = zeros(1,numFigures);

hTabGroup(1) = uitabgroup(figHandle, 'Position', [0.0 0 0.2 1]);

if((wall.right-wall.left) > (wall.front - wall.back))
  hTabGroup(2) = uitabgroup(figHandle, 'Position', [0.2 0.5 0.8 0.5]);
  hTabGroup(3) = uitabgroup(figHandle, 'Position', [0.2 0.0 0.8 0.5]);
else
  hTabGroup(2) = uitabgroup(figHandle, 'Position', [0.2 0 0.4 1]);
```

```
  hTabGroup(3) = uitabgroup(figHandle, 'Position', [0.6 0 0.4 1]);
end

for iFig = 1:5
   hFigure(iFig) = uitab(hTabGroup(figureGroup(iFig)), 'Title', figureTitles{iFig});

   if(strcmp(figureTitles{iFig},'Point Cloud'))
      trackingAx = axes('parent', hFigure(iFig));

      plot(trackingAx, sensor.rangeMin*sin(sensor.angles+scene.azimuthTilt), sensor.rangeMin*cos(sensor.angles+scene.azimuthTilt), '-k');  hold on;
      plot(trackingAx, [0 sensor.rangeMax*sin(sensor.angles+scene.azimuthTilt) 0],[0 sensor.rangeMax*cos(sensor.angles+scene.azimuthTilt) 0], '-k');

      title(figureTitles{iFig},'FontUnits','Normalized', 'FontSize',0.05);
      axis equal;
      axis(scene.maxPos);
      camroll(180)

      % draw wall box
      rectangle(trackingAx, 'Position', scene.areaBox, 'EdgeColor','k', 'LineStyle', '-', 'LineWidth', 2);

      % draw target box
      for nBoxes = 1:scene.numberOfTargetBoxes
         hTargetBoxHandle(nBoxes)= rectangle('Parent', trackingAx, 'Position', scene.targetBox(nBoxes,:), 'EdgeColor', colors(nBoxes), 'LineWidth', 4);
      end


      grid on;
      hold on;
      grid minor;

   end

   if(strcmp(figureTitles{iFig},'Gating and Association'))
      gatingAx = axes('parent', hFigure(iFig));

      plot(gatingAx, sensor.rangeMin*sin(sensor.angles+scene.azimuthTilt), sensor.rangeMin*cos(sensor.angles+scene.azimuthTilt), '-k');  hold on;
      plot(gatingAx, [0 sensor.rangeMax*sin(sensor.angles+scene.azimuthTilt) 0],[0 sensor.rangeMax*cos(sensor.angles+scene.azimuthTilt) 0], '-k');

      title(figureTitles{iFig},'FontUnits','Normalized', 'FontSize',0.05);
      axis equal;
```

```
        axis(scene.maxPos);
        camroll(180)

        % draw wall box
        rectangle(gatingAx, 'Position', scene.areaBox, 'EdgeColor','k', 'LineStyle', '-', 'LineWidth',
2);

        % draw target box
        for nBoxes = 1:scene.numberOfTargetBoxes
          hTargetBoxHandle(nBoxes)= rectangle('Parent', gatingAx, 'Position',
scene.targetBox(nBoxes,:), 'EdgeColor', colors(nBoxes), 'LineWidth', 4);
        end

        grid on;
        hold on;
        grid minor;

    end

    if(strcmp(figureTitles{iFig},'Chirp Configuration'))
        %axes('parent', hFigure(iFig))
        tablePosition = [0.1 0.45 0.8 0.5];
        displayChirpParams(Params, tablePosition, hFigure(iFig));
    end

    if(strcmp(figureTitles{iFig},'Statistics'))
        axes('parent', hFigure(iFig))
        hStatGlobal(1) = text(0, 0.9, 'Frame # 0', 'FontSize',12, 'Visible', 'on');
        hStatGlobal(2) = text(0, 0.8, 'Detection Points: 0','FontSize',12, 'Visible', 'on');
        hStatGlobal(3) = text(0, 0.6, 'People Count:  0','FontSize',24);
        for i=1:scene.numberOfTargetBoxes
          hStatGlobal(end+1) = text(0, 0.6-0.15*i, 'Box Count','FontSize', 30, 'color', colors(i));
        end
        hStatGlobal(end+1) = text(0, 0.7, 'Bytes Available:  0/0','FontSize',12, 'Visible', 'on');
        axis off;
    end

    if(strcmp(figureTitles{iFig},'Visualizer Options & Control'))
        cFig = iFig;
        contW = 0.75;
        contH = 0.05;
        %imshow('tiLogo.jpg')
        hRbPause = uicontrol(hFigure(cFig),'style','checkbox', 'string', 'Pause',...
          'Units','Normalized', 'Position',[0.05 0.85 contW contH],...
          'FontSize', 15);
```

```
        hPlotTabs = uicontrol(hFigure(cFig),'style','checkbox','string','Consolidate plotting tabs',...
            'Units','Normalized', 'Position',[0.05 0.9 contW contH],'Value',0,...
            'FontSize', 15, 'Callback', {@checkPlotTabs,hTabGroup});

        hPbExit = uicontrol('Style', 'pushbutton', 'String', 'EXIT',...
           'Position', [10 10 100 40],'Callback', @exitPressFcn);
        setappdata(hPbExit, 'exitKeyPressed', 0);
    end
end


%% Data structures
syncPatternUINT64                                                                    =
typecast(uint16([hex2dec('0102'),hex2dec('0304'),hex2dec('0506'),hex2dec('0708')]),'uint64');
syncPatternUINT8                                                                     =
typecast(uint16([hex2dec('0102'),hex2dec('0304'),hex2dec('0506'),hex2dec('0708')]),'uint8');

frameHeaderStructType = struct(...
    'sync',             {'uint64', 8}, ... % See syncPatternUINT64 below
    'version',          {'uint32', 4}, ...
    'platform',         {'uint32', 4}, ...
    'timestamp',        {'uint32', 4}, ... % 600MHz clocks
    'packetLength',     {'uint32', 4}, ... % In bytes, including header
    'frameNumber',      {'uint32', 4}, ... % Starting from 1
    'subframeNumber',   {'uint32', 4}, ...
    'chirpMargin',      {'uint32', 4}, ... % Chirp Processing margin, in ms
    'frameMargin',      {'uint32', 4}, ... % Frame Processing margin, in ms
    'uartSentTime' ,    {'uint32', 4}, ... % Time spent to send data, in ms
    'trackProcessTime', {'uint32', 4}, ... % Tracking Processing time, in ms
    'numTLVs' ,         {'uint16', 2}, ... % Number of TLVs in thins frame
    'checksum',         {'uint16', 2});    % Header checksum

tlvHeaderStruct = struct(...
    'type',             {'uint32', 4}, ... % TLV object Type
    'length',           {'uint32', 4});    % TLV object Length, in bytes, including TLV header

% Point Cloud TLV object consists of an array of points.

pointStruct = struct(...
    'range',            {'float', 4}, ... % Range, in m
    'angle',            {'float', 4}, ... % Angel, in rad
    'doppler',          {'float', 4}, ... % Doplper, in m/s
    'snr',              {'float', 4});    % SNR, ratio
% Target List TLV object consists of an array of targets.
targetStruct = struct(...
    'tid',              {'uint32', 4}, ... % Track ID
```

```
    'posX',            {'float', 4}, ... % Target position in X dimension, m
    'posY',            {'float', 4}, ... % Target position in Y dimension, m
    'velX',            {'float', 4}, ... % Target velocity in X dimension, m/s
    'velY',            {'float', 4}, ... % Target velocity in Y dimension, m/s
    'accX',            {'float', 4}, ... % Target acceleration in X dimension, m/s2
    'accY',            {'float', 4}, ... % Target acceleration in Y dimension, m/s
    'EC',              {'float', 9*4}, ... % Tracking error covariance matrix, [3x3], in
range/angle/doppler coordinates
    'G',               {'float', 4});

frameHeaderLengthInBytes = lengthFromStruct(frameHeaderStructType);
tlvHeaderLengthInBytes = lengthFromStruct(tlvHeaderStruct);
pointLengthInBytes = lengthFromStruct(pointStruct);
targetLengthInBytes = lengthFromStruct(targetStruct);
indexLengthInBytes = 1;

exitRequest = 0;
lostSync = 0;
gotHeader = 0;
outOfSyncBytes = 0;
runningSlow = 0;
maxBytesAvailable = 0;
point3D = [];

frameStatStruct = struct('targetFrameNum', [], 'bytes', [], 'numInputPoints', 0,
'numOutputPoints', 0, 'timestamp', 0, 'start', 0, 'benchmarks', [], 'done', 0, ...
    'pointCloud', [], 'targetList', [], 'indexArray', []);
fHist = repmat(frameStatStruct, 1, 10000);
%videoFrame = struct('cdata',[],'colormap', []);
%F = repmat(videoFrame, 10000,1);
optimize = 1;
skipProcessing = 0;
frameNum = 1;
frameNumLogged = 1;
fprintf('------------------\n');

update = 0;
%% Main
while(isvalid(hDataSerialPort))


    while(lostSync == 0 && isvalid(hDataSerialPort))

        frameStart = tic;
        fHist(frameNum).timestamp = frameStart;
        bytesAvailable = get(hDataSerialPort,'BytesAvailable');
```

```
if(bytesAvailable > maxBytesAvailable)
   maxBytesAvailable = bytesAvailable;
end
fHist(frameNum).bytesAvailable = bytesAvailable;
if(gotHeader == 0)
   %Read the header first
   [rxHeader, byteCount] = fread(hDataSerialPort, frameHeaderLengthInBytes, 'uint8');
end
fHist(frameNum).start = 1000*toc(frameStart);

magicBytes = typecast(uint8(rxHeader(1:8)), 'uint64');
if(magicBytes ~= syncPatternUINT64)
   reason = 'No SYNC pattern';
   lostSync = 1;
   break;
end
if(byteCount ~= frameHeaderLengthInBytes)
   reason = 'Header Size is wrong';
   lostSync = 1;
   break;
end
if(validateChecksum(rxHeader) ~= 0)
   reason = 'Header Checksum is wrong';
   lostSync = 1;
   break;
end

frameHeader = readToStruct(frameHeaderStructType, rxHeader);

if(gotHeader == 1)
   if(frameHeader.frameNumber > targetFrameNum)
      targetFrameNum = frameHeader.frameNumber;
      disp(['Found sync at frame ',num2str(targetFrameNum),'(',num2str(frameNum),'),
after ', num2str(1000*toc(lostSyncTime),3), 'ms']);
      gotHeader = 0;
   else
      reason = 'Old Frame';
      gotHeader = 0;
      lostSync = 1;
      break;
   end
end

% We have a valid header
targetFrameNum = frameHeader.frameNumber;
fHist(frameNum).targetFrameNum = targetFrameNum;
```

```
        fHist(frameNum).header = frameHeader;

        dataLength = frameHeader.packetLength - frameHeaderLengthInBytes;

        fHist(frameNum).bytes = dataLength;
        numInputPoints = 0;
        numTargets = 0;
        mIndex = [];

        if(dataLength > 0)
            %Read all packet
            [rxData, byteCount] = fread(hDataSerialPort, double(dataLength), 'uint8');
            if(byteCount ~= double(dataLength))
                reason = 'Data Size is wrong';
                lostSync = 1;
                break;
            end
            offset = 0;

            fHist(frameNum).benchmarks(1) = 1000*toc(frameStart);

            % TLV Parsing
            for nTlv = 1:frameHeader.numTLVs
                tlvType = typecast(uint8(rxData(offset+1:offset+4)), 'uint32');
                tlvLength = typecast(uint8(rxData(offset+5:offset+8)), 'uint32');
                if(tlvLength + offset > dataLength)
                    reason = 'TLV Size is wrong';
                    lostSync = 1;
                    break;
                end
                offset = offset + tlvHeaderLengthInBytes;
                valueLength = tlvLength - tlvHeaderLengthInBytes;
                switch(tlvType)
                    case 6
                        % Point Cloud TLV
                        numInputPoints = valueLength/pointLengthInBytes;
                        if(numInputPoints > 0)
                            % Get Point Cloud from the sensor
                            p = typecast(uint8(rxData(offset+1: offset+valueLength)),'single');

                            pointCloud = reshape(p,4, numInputPoints);
%                           pointCloud(2,:) = pointCloud(2,:)*pi/180;

                            posAll = [pointCloud(1,:).*sin(pointCloud(2,:));
pointCloud(1,:).*cos(pointCloud(2,:))];
```

```
                snrAll = pointCloud(4,:);

                % Remove out of Range, Behind the Walls, out of FOV points
                inRangeInd = (pointCloud(1,:) > 1) & (pointCloud(1,:) < 6) & ...
                  (pointCloud(2,:) > -50*pi/180) &  (pointCloud(2,:) < 50*pi/180) & ...
                  (posAll(1,:) > scene.areaBox(1)) & (posAll(1,:) < (scene.areaBox(1) + 
scene.areaBox(3))) & ...
                  (posAll(2,:) > scene.areaBox(2)) & (posAll(2,:) < (scene.areaBox(2) + 
scene.areaBox(4)));
                pointCloudInRange = pointCloud(:,inRangeInd);
                posInRange = posAll(:,inRangeInd);
%{
                % Clutter removal
                staticInd = (pointCloud(3,:) == 0);
                clutterInd = ismember(pointCloud(1:2,:)', clutterPoints', 'rows');
                clutterInd = clutterInd' & staticInd;
                clutterPoints = pointCloud(1:2,staticInd);
                pointCloud = pointCloud(1:3,~clutterInd);
%}
                numOutputPoints = size(pointCloud,2);
              end
              offset = offset + valueLength;

            case 7
              % Target List TLV
              numTargets = valueLength/targetLengthInBytes;
              TID = zeros(1,numTargets);
              S = zeros(6, numTargets);
              EC = zeros(9, numTargets);
              G = zeros(1,numTargets);
              for n=1:numTargets
                TID(n)  = typecast(uint8(rxData(offset+1:offset+4)),'uint32');      %1x4=4bytes
                S(:,n)  = typecast(uint8(rxData(offset+5:offset+28)),'single');     %6x4=24bytes
                EC(:,n) = typecast(uint8(rxData(offset+29:offset+64)),'single');    %9x4=36bytes
                G(n)    = typecast(uint8(rxData(offset+65:offset+68)),'single');    %1x4=4bytes
                offset = offset + 68;
              end

            case 8
              % Target Index TLV
              numIndices = valueLength/indexLengthInBytes;
              mIndex = typecast(uint8(rxData(offset+1:offset+numIndices)),'uint8');
              offset = offset + valueLength;
          end
        end
      end
```

```
    if(numInputPoints == 0)
        numOutputPoints = 0;
        pointCloud = single(zeros(4,0));
        posAll = [];
        posInRange = [];
    end
    if(numTargets == 0)
        TID = [];
        S = [];
        EC = [];
        G = [];
    end

    fHist(frameNum).numInputPoints = numInputPoints;
    fHist(frameNum).numOutputPoints = numOutputPoints;
    fHist(frameNum).numTargets = numTargets;
    fHist(frameNum).pointCloud = pointCloud;
    fHist(frameNum).targetList.numTargets = numTargets;
    fHist(frameNum).targetList.TID = TID;
    fHist(frameNum).targetList.S = S;
    display(fHist(frameNum))

    if(~optimize)
        fHist(frameNum).targetList.EC = EC;
    end
    fHist(frameNum).targetList.G = G;
    fHist(frameNum).indexArray = mIndex;

    % Plot pointCloud
    fHist(frameNum).benchmarks(2) = 1000*toc(frameStart);

    if(get(hRbPause, 'Value') == 1)
        pause(0.01);
        continue;
    end

    %if frameNum/100 == 0

        Data = fHist(frameNum).targetList.S;
        s1s = "Q[";
%       string1 = strcat(s1s,d1s);
        for i =1:numTargets
        s1s =   s1s+"[" + num2str(Data(1,i))+","+num2str(Data(2,i))+"],";

        %display(strlength(s1s))
```

```
    %   fwrite(t, num2str(Data(1,i)));
        end
        fwrite(t, s1s+"]");

    % Delete previous points
    if(ishandle(hPlotCloudHandleAll))
        delete(hPlotCloudHandleAll);
    end
    if(ishandle(hPlotCloudHandleOutRange))
        delete(hPlotCloudHandleOutRange);
    end
    if(ishandle(hPlotCloudHandleClutter))
        delete(hPlotCloudHandleClutter);
    end
    if(ishandle(hPlotCloudHandleStatic))
        delete(hPlotCloudHandleStatic);
    end
    if(ishandle(hPlotCloudHandleDynamic))
        delete(hPlotCloudHandleDynamic);
    end

    if(size(posAll,2))
        % Plot all points
        if(snrAll*10 > 0)
            if(~optimize)
                hPlotCloudHandleAll = scatter(trackingAx, posAll(1,:),
posAll(2,:),'.k','SizeData',snrAll*10);
            else
                hPlotCloudHandleAll = plot(trackingAx, posAll(1,:), posAll(2,:),'.k');
            end
        else
            reason = 'SNR value is wrong';
            lostSync = 1;
            break;
        end
        % Cross out out-of-Range
        if(~optimize)
            hPlotCloudHandleOutRange = plot(trackingAx, posAll(1,~inRangeInd),
posAll(2,~inRangeInd), 'xr');
        end
    end


    if(size(posInRange,2))
        % Cross out Clutter
```

```
        %       hPlotCloudHandleClutter       =       plot(trackingAx,       posInRange(1,clutterInd),
posInRange(2,clutterInd), 'xk');
        % Indicate Static
%             hPlotCloudHandleStatic = plot(trackingAx, posInRange(1,staticInd & ~clutterInd),
posInRange(2,staticInd & ~clutterInd), 'ok');
        % Indicate Dynamic
       %         hPlotCloudHandleDynamic     =     plot(trackingAx,     posInRange(1,~staticInd),
posInRange(2,~staticInd), 'ob');
    end

    fHist(frameNum).benchmarks(3) = 1000*toc(frameStart);

    switch trackerRun
        case 'Target'
            if(numTargets == 0)
                TID = zeros(1,0);
                S = zeros(6,0);
                EC = zeros(9,0);
                G = zeros(1,0);
            end
    end

    fHist(frameNum).benchmarks(4) = 1000*toc(frameStart);

    if nnz(isnan(S))
        reason = 'Error: S contains NaNs';
        lostSync = 1;
        break;
    end
    if nnz(isnan(EC))
        reason = 'Error: EC contains NaNs';
        lostSync = 1;
        break;
    end

    tNumC = length(TID);
    peopleCountTotal = tNumC;
    peopleCountInBox = zeros(1, scene.numberOfTargetBoxes);

    if(size(mIndex,1))
        mIndex = mIndex + 1;
    end

    % Plot previous frame's 3D points
    if(size(point3D,2))
        if isempty(hPlotPoints3D)
```

```
            %hPlotPoints3D = plot(gatingAx, point3D(1,:), point3D(2,:), '.k');%, point3D(3,:),'.k');
         else
            %set(hPlotPoints3D, 'XData', point3D(1,:),'YData', point3D(2,:)); %, 'ZData',
point3D(3,:));
         end
      end

      for n=1:tNumC

         tid = TID(n)+1;
         if(tid > maxNumTracks)
            reason = 'Error: TID is wrong';
            lostSync = 1;
            break;
         end

         if( (size(mIndex,1) > 0) && (size(mIndex,1) == size(point3D,2)) )
            tColor = colors(mod(tid,length(colors))+1);
            ind = (mIndex == tid);
            if nnz(ind)
               %trackingHist(tid).hPlotAssociatedPoints = plot(gatingAx, point3D(1,ind),
point3D(2,ind), 'o', 'color', tColor)
               if (isempty(trackingHist(tid).hPlotAssociatedPoints) ||
~ishandle(trackingHist(tid).hPlotAssociatedPoints))
                  trackingHist(tid).hPlotAssociatedPoints = plot(gatingAx, point3D(1,ind),
point3D(2,ind), '.', 'MarkerSize', 10, 'color', tColor);
               else
                  if ishandle(trackingHist(tid).hPlotAssociatedPoints)
                     set(trackingHist(tid).hPlotAssociatedPoints, 'XData', point3D(1,ind),'YData',
point3D(2,ind));
                  end
               end
            end
         end

         %g = G(n);
         centroid = computeH(1, S(:,n));
         ec = reshape(EC(:,n),3,3);
         if(nnz(ec)>1)
            dim = getDim(gatingAx, 1, centroid, ec);

            if isempty(trackingHist(tid).hMeshU)
               trackingHist(tid).hMeshU = circle(gatingAx,S(1,n), S(2,n),dim);
               if(labelTrack)
                  trackingHist(tid).hMeshU.UserData = text(gatingAx,S(1,n), S(2,n),
char(65+mod(tid,26)),'HorizontalAlignment','center','FontUnits','normalized','FontSize',0.5/sce
```

```
ne.maxPos(4)*0.75);
          end
            if(~optimize)
            trackingHist(tid).hMeshU.EdgeColor = [0.5 0.5 0.5];
          else
            trackingHist(tid).hMeshU.EdgeColor = colors(mod(tid,length(colors))+1);
          end
          trackingHist(tid).hMeshU.FaceColor = 'none';
        else
          %set(trackingHist(tid).hMeshU,   'XData',   xU.*sin(yU),'YData',xU.*cos(yU),   'ZData',
zU);
          trackingHist(tid).hMeshU.Position = updateCenter(S(1,n), S(2,n),dim);
          if(labelTrack)
            trackingHist(tid).hMeshU.UserData.Position = [S(1,n), S(2,n)];
          end
        end
      end

      if(~optimize)
        if(g ~= 0)
          [xG, yG, zG, vG] = gatePlot3(gatingAx, g, centroid, ec);
          if isempty(trackingHist(tid).hMeshG)
            trackingHist(tid).hMeshG = mesh(gatingAx, xG.*sin(yG),xG.*cos(yG), zG);
            trackingHist(tid).hMeshG.EdgeColor = colors(mod(tid,length(colors))+1);
            trackingHist(tid).hMeshG.FaceColor = 'none';
          else
            set(trackingHist(tid).hMeshG,   'XData',   xG.*sin(yG),'YData',xG.*cos(yG),   'ZData',
zG);
          end
        end
      end

      if(activeTracks(tid) == 0)
        activeTracks(tid) = 1;
        trackingHist(tid).tid = TID(n);
        trackingHist(tid).allocationTime = targetFrameNum;
        trackingHist(tid).tick = 1;
        trackingHist(tid).posIndex = 1;
        trackingHist(tid).histIndex = 1;
        trackingHist(tid).sHat(1,:) = S(:,n);
        trackingHist(tid).pos(1,:) = S(1:2,n);
        trackingHist(tid).hPlotTrack = plot(trackingAx, S(1,n), S(2,n), '-', 'LineWidth', 2, 'color',
colors(mod(tid,length(colors))+1));
        trackingHist(tid).hPlotCentroid   =   plot(trackingAx,   S(1,n),   S(2,n),   'o',   'color',
colors(mod(tid,length(colors))+1));
      else
```

```
            activeTracks(tid) = 1;
            trackingHist(tid).tick = trackingHist(tid).tick + 1;

            trackingHist(tid).histIndex = trackingHist(tid).histIndex + 1;
            if(trackingHist(tid).histIndex > 1000)
               trackingHist(tid).histIndex = 1;
            end
            trackingHist(tid).sHat(trackingHist(tid).histIndex,:) = S(:,n);
            if(~optimize)
               trackingHist(tid).ec(trackingHist(tid).histIndex,:) = EC(:,n);
            end
            trackingHist(tid).posIndex = trackingHist(tid).posIndex + 1;
            if(trackingHist(tid).posIndex > 100)
               trackingHist(tid).posIndex = 1;
            end
            trackingHist(tid).pos(trackingHist(tid).posIndex,:) = S(1:2,n);

            if(trackingHist(tid).tick > 100)
               set(trackingHist(tid).hPlotTrack,                                                         'XData',
[trackingHist(tid).pos(trackingHist(tid).posIndex+1:end,1);
trackingHist(tid).pos(1:trackingHist(tid).posIndex,1)], ...
                  'YData',[trackingHist(tid).pos(trackingHist(tid).posIndex+1:end,2);
trackingHist(tid).pos(1:trackingHist(tid).posIndex,2)]);
            else
               set(trackingHist(tid).hPlotTrack,                                                         'XData',
trackingHist(tid).pos(1:trackingHist(tid).posIndex,1), ...
                  'YData',trackingHist(tid).pos(1:trackingHist(tid).posIndex,2));
            end
            set(trackingHist(tid).hPlotCentroid,'XData',S(1,n),'YData',S(2,n));

            for nBoxes = 1:scene.numberOfTargetBoxes
               if( (S(1,n) > scene.targetBox(nBoxes,1)) && (S(1,n) < (scene.targetBox(nBoxes,1) +
scene.targetBox(nBoxes,3))) && ...
                     (S(2,n)       >       scene.targetBox(nBoxes,2))       &&       (S(2,n)       <
(scene.targetBox(nBoxes,2)+scene.targetBox(nBoxes,4))) )
                  peopleCountInBox(nBoxes) = peopleCountInBox(nBoxes) +1;
               end
            end
         end
      end

      iDelete = find(activeTracks == 2);
      for n=1:length(iDelete)
         ind = iDelete(n);
         delete(trackingHist(ind).hPlotTrack);
         delete(trackingHist(ind).hPlotCentroid);
```

```
        delete(trackingHist(ind).hMeshU.UserData);
        delete(trackingHist(ind).hMeshU);
        delete(trackingHist(ind).hMeshG);
        delete(trackingHist(ind).hPlotAssociatedPoints);
        trackingHist(ind).hMeshU = [];
        trackingHist(ind).hMeshG = [];
        activeTracks(ind) = 0;
    end

    iReady = (activeTracks == 1);
    activeTracks(iReady) = 2;

    fHist(frameNum).done = 1000*toc(frameStart);

    string{1} = sprintf('Frame #: %d', targetFrameNum);
    string{2} = sprintf('Detection Points: %d', numOutputPoints);
    string{3} = sprintf('People Count: %d', peopleCountTotal);
    for i=1:scene.numberOfTargetBoxes
        string{3+i} = sprintf('Box %d Count: %d', i, peopleCountInBox(i));
    end
    string{3+scene.numberOfTargetBoxes+1} = sprintf('Bytes Available: %d/%d',
bytesAvailable, maxBytesAvailable);

    for n=1:length(hStatGlobal)
        set(hStatGlobal(n),'String',string{n});
    end


    for nBoxes = 1:scene.numberOfTargetBoxes
        if(peopleCountInBox(nBoxes))
            set(hTargetBoxHandle(nBoxes), 'LineWidth', 14);
        else
            set(hTargetBoxHandle(nBoxes), 'LineWidth', 4);
        end
    end

    if(getappdata(hPbExit, 'exitKeyPressed') == 1)
        if(frameNumLogged > 10000)
            fHist = [fHist(frameNum+1:end) fHist(1:frameNum)];
        else
            fHist = fHist(1:frameNum);
        end
        % changes to the code here
        save('fhistRT.mat','fHist');
        disp('Saving data and exiting');
        close_main()
```

```
            return;


        end

        frameNum = frameNum + 1;
        frameNumLogged = frameNumLogged + 1;
        if(frameNum > 10000)
            frameNum = 1;
        end

        point3D = [posAll; pointCloud(3,:)];

        if(bytesAvailable > 32000)
            runningSlow  = 1;
        elseif(bytesAvailable < 1000)
            runningSlow = 0;
        end

        if(runningSlow)
            % Don't pause, we are slow
        else
            pause(0.01);
        end
    end

    if(targetFrameNum)
        lostSyncTime = tic;
        bytesAvailable = get(hDataSerialPort,'BytesAvailable');
        disp(['Lost sync at frame ', num2str(targetFrameNum),'(', num2str(frameNum), '), Reason:
', reason, ', ', num2str(bytesAvailable), ' bytes in Rx buffer']);
    else
        errordlg('Port sync error: Please close and restart program');
    end
%{
    % To catch up, we read and discard all uart data
    bytesAvailable = get(hDataSerialPort,'BytesAvailable');
    disp(bytesAvailable);
    [rxDataDebug, byteCountDebug] = fread(hDataSerialPort, bytesAvailable, 'uint8');
%}
    while(lostSync)
        for n=1:8
            [rxByte, byteCount] = fread(hDataSerialPort, 1, 'uint8');
            if(rxByte ~= syncPatternUINT8(n))
                outOfSyncBytes = outOfSyncBytes + 1;
                break;
```

```
        end
      end
      if(n == 8)
        lostSync = 0;
        frameNum = frameNum + 1;
        if(frameNum > 10000)
          frameNum = 1;
        end

        [header, byteCount] = fread(hDataSerialPort, frameHeaderLengthInBytes - 8, 'uint8');
        rxHeader = [syncPatternUINT8'; header];
        byteCount = byteCount + 8;
        gotHeader = 1;
      end
    end
  end

%% Helper functions

Display Chirp parameters in table on screen
function h = displayChirpParams(Params, Position, hFig)

  dat =  {'Start Frequency (Ghz)', Params.profileCfg.startFreq;...
        'Slope (MHz/us)', Params.profileCfg.freqSlopeConst;...
        'Samples per chirp', Params.profileCfg.numAdcSamples;...
        'Chirps per frame',  Params.dataPath.numChirpsPerFrame;...
        'Frame duration (ms)',  Params.frameCfg.framePeriodicity;...
        'Sampling rate (Msps)', Params.profileCfg.digOutSampleRate / 1000;...
        'Bandwidth (GHz)', Params.profileCfg.freqSlopeConst *
Params.profileCfg.numAdcSamples /...
                Params.profileCfg.digOutSampleRate;...
        'Range resolution (m)', Params.dataPath.rangeResolutionMeters;...
        'Velocity resolution (m/s)', Params.dataPath.dopplerResolutionMps;...
        'Number of Rx (MIMO)', Params.dataPath.numRxAnt; ...
        'Number of Tx (MIMO)', Params.dataPath.numTxAnt;};
  columnname =   {'Chirp Parameter (Units)      ', 'Value'};
  columnformat = {'char', 'numeric'};

  h = uitable('Parent',hFig,'Units','normalized', ...
        'Position', Position, ...
        'Data', dat,...
        'ColumnName', columnname,...
        'ColumnFormat', columnformat,...
        'ColumnWidth', 'auto',...
        'RowName',[]);
end
```

```
function [P] = parseCfg(cliCfg)
   P=[];
   for k=1:length(cliCfg)
      C = strsplit(cliCfg{k});
      if strcmp(C{1},'channelCfg')
         P.channelCfg.txChannelEn = str2double(C{3});
         P.dataPath.numTxAzimAnt = bitand(bitshift(P.channelCfg.txChannelEn,0),1) +...
                          bitand(bitshift(P.channelCfg.txChannelEn,-1),1);
         P.dataPath.numTxElevAnt = 0;
         P.channelCfg.rxChannelEn = str2double(C{2});
         P.dataPath.numRxAnt = bitand(bitshift(P.channelCfg.rxChannelEn,0),1) +...
                       bitand(bitshift(P.channelCfg.rxChannelEn,-1),1) +...
                       bitand(bitshift(P.channelCfg.rxChannelEn,-2),1) +...
                       bitand(bitshift(P.channelCfg.rxChannelEn,-3),1);
         P.dataPath.numTxAnt = P.dataPath.numTxElevAnt + P.dataPath.numTxAzimAnt;

      elseif strcmp(C{1},'dataFmt')
      elseif strcmp(C{1},'profileCfg')
         P.profileCfg.startFreq = str2double(C{3});
         P.profileCfg.idleTime =  str2double(C{4});
         P.profileCfg.rampEndTime = str2double(C{6});
         P.profileCfg.freqSlopeConst = str2double(C{9});
         P.profileCfg.numAdcSamples = str2double(C{11});
         P.profileCfg.digOutSampleRate = str2double(C{12}); %uints: ksps
      elseif strcmp(C{1},'chirpCfg')
      elseif strcmp(C{1},'frameCfg')
         P.frameCfg.chirpStartIdx = str2double(C{2});
         P.frameCfg.chirpEndIdx = str2double(C{3});
         P.frameCfg.numLoops = str2double(C{4});
         P.frameCfg.numFrames = str2double(C{5});
         P.frameCfg.framePeriodicity = str2double(C{6});
      elseif strcmp(C{1},'guiMonitor')
         P.guiMonitor.detectedObjects = str2double(C{2});
         P.guiMonitor.logMagRange = str2double(C{3});
         P.guiMonitor.rangeAzimuthHeatMap = str2double(C{4});
         P.guiMonitor.rangeDopplerHeatMap = str2double(C{5});
      end
   end
   P.dataPath.numChirpsPerFrame = (P.frameCfg.chirpEndIdx -...
                              P.frameCfg.chirpStartIdx + 1) *...
                              P.frameCfg.numLoops;
   P.dataPath.numDopplerBins = P.dataPath.numChirpsPerFrame / P.dataPath.numTxAnt;
   P.dataPath.numRangeBins = pow2roundup(P.profileCfg.numAdcSamples);
   P.dataPath.rangeResolutionMeters = 3e8 * P.profileCfg.digOutSampleRate * 1e3 /...
            (2 * P.profileCfg.freqSlopeConst * 1e12 * P.profileCfg.numAdcSamples);
```

```
    P.dataPath.rangeIdxToMeters = 3e8 * P.profileCfg.digOutSampleRate * 1e3 /...
                (2 * P.profileCfg.freqSlopeConst * 1e12 * P.dataPath.numRangeBins);
    P.dataPath.dopplerResolutionMps = 3e8 / (2*P.profileCfg.startFreq*1e9 *...
                                (P.profileCfg.idleTime + P.profileCfg.rampEndTime) *...
                                1e-6 * P.dataPath.numDopplerBins * P.dataPath.numTxAnt);
end

function [] = dispError()
    disp('Serial Port Error!');
end

function exitPressFcn(hObject, ~)
    setappdata(hObject, 'exitKeyPressed', 1);
end

function checkPlotTabs(hObject, eventData, hTabGroup)
    if(hObject.Value)
        % get children
        children = hTabGroup(3).Children;
        hTabGroup(3).UserData = children; %save children to restore

        % combine tab group
        for t=1:length(children)
            set(children(t),'Parent',hTabGroup(2));
        end

        % resize tab group
        hTabGroup(2).UserData = hTabGroup(2).Position; %save position to restore
        hTabGroup(2).Position = [0.2 0 0.8 1];
        hTabGroup(3).Visible = 'off';
    else
        % restore children
        children = hTabGroup(3).UserData;

        % move tab group
        for t=1:length(children)
            set(children(t),'Parent',hTabGroup(3));
        end

        % resize tab group
        hTabGroup(2).Position = hTabGroup(2).UserData;
        hTabGroup(3).Visible = 'on';
    end

end
```

```
function [sphandle] = configureDataSport(comPortNum, bufferSize)
    if ~isempty(instrfind('Type','serial'))
        disp('Serial port(s) already open. Re-initializing...');
        delete(instrfind('Type','serial'));  % delete open serial ports.
    end
    comPortString = ['COM' num2str(comPortNum)];
    sphandle = serial(comPortString,'BaudRate',921600);
    set(sphandle,'Terminator', '');
    set(sphandle,'InputBufferSize', bufferSize);
    set(sphandle,'Timeout',10);
    set(sphandle,'ErrorFcn',@dispError);
    fopen(sphandle);
end

function [sphandle] = configureControlPort(comPortNum)
    %if ~isempty(instrfind('Type','serial'))
    %    disp('Serial port(s) already open. Re-initializing...');
    %    delete(instrfind('Type','serial'));  % delete open serial ports.
    %end
    comPortString = ['COM' num2str(comPortNum)];
    sphandle = serial(comPortString,'BaudRate',115200);
    set(sphandle,'Parity','none')
    set(sphandle,'Terminator','LF')
    fopen(sphandle);
end

function config = readCfg(filename)
    config = cell(1,100);
    fid = fopen(filename, 'r');
    if fid == -1
        fprintf('File %s not found!\n', filename);
        return;
    else
        fprintf('Opening configuration file %s ...\n', filename);
    end
    tline = fgetl(fid);
    k=1;
    while ischar(tline)
        config{k} = tline;
        tline = fgetl(fid);
        k = k + 1;
    end
    config = config(1:k-1);
    fclose(fid);
end
```

```
function length = lengthFromStruct(S)
   fieldName = fieldnames(S);
   length = 0;
   for n = 1:numel(fieldName)
      [~, fieldLength] = S.(fieldName{n});
      length = length + fieldLength;
   end
end

function [R] = readToStruct(S, ByteArray)
   fieldName = fieldnames(S);
   offset = 0;
   for n = 1:numel(fieldName)
      [fieldType, fieldLength] = S.(fieldName{n});
      R.(fieldName{n}) = typecast(uint8(ByteArray(offset+1:offset+fieldLength)), fieldType);
      offset = offset + fieldLength;
   end
end
function CS = validateChecksum(header)
   h = typecast(uint8(header),'uint16');
   a = uint32(sum(h));
   b = uint16(sum(typecast(a,'uint16')));
   CS = uint16(bitcmp(b));
end

function [H] = computeH(~, s)
   posx = s(1); posy = s(2); velx = s(3); vely = s(4);
   range = sqrt(posx^2+posy^2);
   if posy == 0
      azimuth = pi/2;
   elseif posy > 0
      azimuth = atan(posx/posy);
   else
      azimuth = atan(posx/posy) + pi;
   end
   doppler = (posx*velx+posy*vely)/range;
   H = [range azimuth doppler]';
end

function [XX, YY, ZZ, v] = gatePlot3(~, G, C, A)
   %Extract the ellipsoid's axes lengths (a,b,c) and the rotation matrix (V) using singular value
decomposition:
   [~,D,V] = svd(A/G);

   a = 1/sqrt(D(1,1));
   b = 1/sqrt(D(2,2));
```

```
    c = 1/sqrt(D(3,3));
    v = 4*pi*a*b*c/3;

    % generate ellipsoid at 0 origin
    [X,Y,Z] = ellipsoid(0,0,0,a,b,c);
    XX = zeros(size(X));
    YY = zeros(size(X));
    ZZ = zeros(size(X));
    for k = 1:length(X)
        for j = 1:length(X)
            point = [X(k,j) Y(k,j) Z(k,j)]';
            P = V * point;
            XX(k,j) = P(1)+C(1);
            YY(k,j) = P(2)+C(2);
            ZZ(k,j) = P(3)+C(3);
        end
    end
end

function [maxDim] = getDim(~, G, C, A)
    %Extract the ellipsoid's axes lengths (a,b,c) and the rotation matrix (V) using singular value
decomposition:
    [~,D,V] = svd(A/G);

    a = 1/sqrt(D(1,1));
    b = 1/sqrt(D(2,2));
    c = 1/sqrt(D(3,3));

    maxDim = max([a,b]);

end

function [y] = pow2roundup (x)
    y = 1;
    while x > y
        y = y * 2;
    end
end

function h = circle(ax, x,y,r)
d = r*2;
px = x-r;
py = y-r;
dim = [px py d d];
h = rectangle(ax, 'Position',dim,'Curvature',[1,1], 'LineWidth',3);
daspect([1,1,1])
```

```
end

function h = updateCenter(x,y,r,offset)
d = r*2;
px = x-r;
py = y-r;
h = [px py d d];
end

function close_main()
   %helpdlg('Saving and closing');
   open_port = instrfind('Type','serial','Status','open');
   for i=1:length(open_port)
      fclose(open_port(i));
      delete(open_port(i));
   end
   clear all
   delete(findobj('Tag', 'mainFigure'));

end

function mypreview_fcn(obj,event,himage)
% Example update preview window function.

% Display image data.
himage.CData = fliplr(event.Data);
end

function [resInd] = getWidestFOV(resList)
maxR = 1;
resInd = 1;
   for i=1:length(resList)
      ss = strsplit(resList{i},'x');
      imWidth = str2num(ss{1});
      imHeight = str2num(ss{2});
      r = imWidth/imHeight;
      if (r>maxR)
         maxR = r;
         resInd = i;
      end
   end
end
```

Code for Camera module integration:

```
def scale_boxes(boxes, image_shape):
  """ Scales the predicted boxes in order to be drawable on the image"""
  height = image_shape[0]
  width = image_shape[1]
  image_dims = K.stack([height, width, height, width])
  image_dims = K.reshape(image_dims, [1, 4])
  boxes = boxes * image_dims
  return boxes

def read_classes(classes_path):
  with open(classes_path) as f:
      class_names = f.readlines()
  class_names = [c.strip() for c in class_names]
  return class_names

def read_anchors(anchors_path):
  with open(anchors_path) as f:
      anchors = f.readline()
      anchors = [float(x) for x in anchors.split(',')]
      anchors = np.array(anchors).reshape(-1, 2)
  return anchors


def yolo_filter_boxes(box_confidence, boxes, box_class_probs, threshold = .6):

  # Compute box scores
  box_scores = box_confidence*box_class_probs

  # Find the box_classes thanks to the max box_scores, keep track of the corresponding score
  box_classes = K.argmax(box_scores, -1)
  box_class_scores = K.max(box_scores, -1)


  filtering_mask = box_class_scores >= threshold

  scores = tf.boolean_mask(box_class_scores, filtering_mask)
  boxes =  tf.boolean_mask(boxes, filtering_mask)
  classes =tf.boolean_mask(box_classes, filtering_mask)

  return scores, boxes, classes


def iou(box1, box2):
```

```
  xi1 = max(box1[0], box2[0])
  yi1 = max(box1[1], box2[1])
  xi2 = min(box1[2], box2[2])
  yi2 = min(box1[3], box2[3])
  inter_area = max(yi2-yi1,0)*max(xi2-xi1,0)

  box1_area = (box1[2]-box1[0])*(box1[3]-box1[1])
  box2_area = (box2[2]-box2[0])*(box2[3]-box2[1])
  union_area = box1_area + box2_area - inter_area

  iou = inter_area/union_area

  return iou



def yolo_non_max_suppression(scores, boxes, classes, max_boxes = 10, iou_threshold = 0.5):

  max_boxes_tensor = K.variable(max_boxes, dtype='int32')     # tensor to be used in
tf.image.non_max_suppression()
  K.get_session().run(tf.variables_initializer([max_boxes_tensor]))  # initialize variable
max_boxes_tensor

  nms_indices = tf.image.non_max_suppression(boxes,scores,max_boxes,iou_threshold)

  scores = K.gather(scores, nms_indices)
  boxes = K.gather(boxes, nms_indices)
  classes = K.gather(classes, nms_indices)

  return scores, boxes, classes


def yolo_eval(yolo_outputs, image_shape = (720., 1280.), max_boxes=10,
score_threshold=.75, iou_threshold=.5):


  box_confidence, box_xy, box_wh, box_class_probs = yolo_outputs
  boxes = yolo_boxes_to_corners(box_xy, box_wh)
  scores, boxes, classes = yolo_filter_boxes(box_confidence, boxes, box_class_probs,
score_threshold)
  boxes = scale_boxes(boxes, image_shape)
  scores, boxes, classes = yolo_non_max_suppression(scores, boxes, classes, max_boxes,
iou_threshold )

  return scores, boxes, classes
```

```
image = np.zeros((416,416,3))

def readImages(path):
  cap = cv2.VideoCapture(path)
  global image
  while(True):
  ret, frame = cap.read()
  print(ret)
  if(ret):
        image = np.array(cv2.resize(frame, (416, 416), interpolation = cv2.INTER_CUBIC), dtype='float32')/255.
  else :
        break

class CameraVideoStream :
  def __init__(self, src = 0, width = 416, height = 416) :
  self.stream = cv2.VideoCapture(src)
  # self.stream.set(3,width)
  # self.stream.set(4,height)
  (self.grabbed, self.frame) = self.stream.read()
  self.started = False
  self.read_lock = Lock()

  def start(self) :
  if self.started :
        print ("already started!!")
        return None
  self.started = True
  self.thread = Thread(target=self.update, args=())
  self.thread.start()
  return self

  def update(self) :
  while self.started :
        (grabbed, frame) = self.stream.read()
        self.read_lock.acquire()
        self.grabbed, self.frame = grabbed, frame
        self.read_lock.release()

  def read(self) :
  self.read_lock.acquire()
  frame = self.frame.copy()
  self.read_lock.release()
  image = np.array(cv2.resize(frame, (416, 416), interpolation = cv2.INTER_CUBIC), dtype='float32')/255.
  return image
```

```
  def stop(self) :
  self.started = False
  if self.thread.is_alive():
          self.thread.join()

  def __exit__(self, exc_type, exc_value, traceback) :
  self.stream.release()


class Server :
  def __init__(self, host, port):
  self.host = host
  self.port = port
  self.data = 0
  self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
  self.sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
  self.sock.bind((self.host, 10068))
  self.read_lock = Lock()
  self.started = False
  self.stopped = False

  print("Waiting for connection")

  def start(self) :
  self.thread = Thread(target=self.listen, args=())
  self.thread.start()
  return self


  def listen(self):
  self.sock.listen(1)
  while True:
          time.sleep(2)
          if(self.stopped):
                  break
          client, address = self.sock.accept()
          print(client)
          client.settimeout(60)
          Thread(target = self.listenToClient,args = (client,address)).start()


  def listenToClient(self, client, address):
  size = 1024
  while True:
          try:
```

```
                data = client.recv(size)
                self.started = True
                if data:
                    self.read_lock.acquire()
                    self.data = data.decode('utf-8')
                    self.read_lock.release()
                else:
                    raise error('Client disconnected')
            except:
                client.close()
                return False

    def readData(self):
        if self.started:
            # pass
            self.read_lock.acquire()
            data = self.data
            self.read_lock.release()
            return data

    def stop(self) :
        self.stopped = True
        self.sock.close()




sess = K.get_session()

class_names = read_classes("model_data/coco_classes.txt")
anchors = read_anchors("model_data/yolov2_anchors.txt")
image_shape = (416., 416.)


# Loading a pretrained model


yolo_model = load_model("model_data/yolov2.h5")
# yolo_model.summary()
yolo_outputs = yolo_head(yolo_model.output, anchors, len(class_names))
scores, boxes, classes = yolo_eval(yolo_outputs, image_shape)
# cap = cv2.VideoCapture(0)
# cap = cv2.VideoCapture('http://192.168.4.1:8160')
# cap.set(5, 20)
server = Server('192.168.4.16',10067).start()
videoStream = CameraVideoStream(src='http://192.168.4.1:8160').start()
```

```
H = np.array([[-122.8303,-0.2614,253.7836],[9.0520,12.0507,187.7295],[0.,0.,1.]])
def predict(sess):
  while(True):
  # image, image_data = preprocess_image("images/" + image_file, model_image_size = (416, 416))
  # ret, frame = cap.read()
  # print("got frame")
  image = videoStream.read()
  image_data = np.expand_dims(image, 0)
  # plt.imsave(os.path.join("out", image_file), image)
  out_scores, out_boxes, out_classes = sess.run([scores, boxes, classes], feed_dict={yolo_model.input: image_data, K.learning_phase(): 0})
  # print(out_classes)
  # print('Found {} boxes for {}'.format(len(out_boxes), image_file))

  # colors = generate_colors(class_names)
  peopleCenter = []
  for i in range(len(out_boxes)):
        c = out_classes[i]
        if c == 0:
              score = out_scores[i]
              predicted_class = class_names[c]
              label = '{} {:.2f}'.format(predicted_class, score)
              box = out_boxes[i]
              top, left, bottom, right = box
              top = max(0, np.floor(top + 0.5).astype('int32'))
              left = max(0, np.floor(left + 0.5).astype('int32'))
              bottom = min(image.shape[1], np.floor(bottom + 0.5).astype('int32'))
              right = min(image.shape[0], np.floor(right + 0.5).astype('int32'))
              peopleCenter.append(np.array([(left+right)/2, (top+bottom)/2]))

              cv2.rectangle(image, (left, top), (right, bottom), (255,0,0), 2)
              # data = server.readData()
              # if data:
              #     data = data.split("Q")[1][1:][:-2]
              #     for everyCoordinate in data.replace("],[", "] [").split():
              #           b = np.array([float(x) for x in everyCoordinate.strip("[").rstrip("]").split(',')])
              #           UV = np.matmul(H,np.array([b[0], b[1], 1]))
              #           if  left < UV[0] and UV[0] < right and top < UV[1] and UV[1] < bottom :
              #                 cv2.putText(image,"Depth = "+str(b[1]), (int(UV[0]), int(UV[1])), cv2.FONT_HERSHEY_SIMPLEX, 0.6,(255,255,0),1)

              # cv2.putText(image,label, (left-5, top-5), cv2.FONT_HERSHEY_SIMPLEX, 0.6,(255,255,0),1)
```

```
                    # print("Coordinate X = ", (left+right)/2)
                    # print("Coordinate Y = ", (top+bottom)/2)

    maxpeople = len(peopleCenter)
    radarDetections = []
    radarPeople = []
    if maxpeople :
            data = server.readData()
            if data:
                    print("Data ",data)
                    data = data.split("Q")[1][1:][:-2]
                    for everyCoordinate in data.replace("],[", "] [").split():
                            b = np.array([float(x) for x in
everyCoordinate.strip("[").rstrip("]").split(',')])
                            radarDetections.append(b)
                            UV = np.matmul(H,np.array([b[0], b[1], 1]))
                            radarPeople.append(np.array([UV[0], UV[1]]))

            radarPeople = np.array(radarPeople)
            cost = []
            for i in range(maxpeople):
                    diff = np.linalg.norm(peopleCenter[i] - radarPeople.reshape(-1,2), axis=1)
                    cost.append(diff)

            cost = np.array(cost)*0.1
            row, col = linear_sum_assignment(cost)

            assignment = [-1]*maxpeople
            for i in range(len(row)):
                    assignment[row[i]] = col[i]

            for i in range(len(assignment)):
                    if assignment[i] != -1:
                            print(radarDetections[assignment[i]])
                            cv2.putText(image,"Depth = "+str(radarDetections[assignment[i]][1]),
(int(radarPeople[assignment[i]][0]), int(radarPeople[assignment[i]][1])),

    cv2.FONT_HERSHEY_SIMPLEX, 0.6,(255,255,0),1)


    # draw_boxes(image, out_scores, out_boxes, out_classes, class_names, colors)
    # plt.imsave(os.path.join("out", image_file), image)
    cv2.imshow('frame',image)
    # image.save(os.path.join("out", image_file), quality=90)
    # output_image = scipy.misc.imread(os.path.join("out", image_file))
    # imshow(output_image)
```

```
  if cv2.waitKey(1) & 0xFF == ord('q'):
        break
  videoStream.stop()
  # server.stop()
  cv2.destroyAllWindows()
  sys.exit(0)

  return out_scores, out_boxes, out_classes

# results = pool.map(readImages, predict, ['http://192.168.4.1:8160', sess])
out_scores, out_boxes, out_classes = predict(sess)
```

Individual Contributions

Shreekant Kodukula and **Vaibhav Srivastava** are responsible for carrying out experiments and all the activities with the mmWave sensor. The task of configuring the sensor was also done in unison. Each of us wrote some or the other part of the code together in MATLAB for the GUI for obtaining and displaying point cloud,corresponding clusters and tracking of the clusters. **Abhishek Sharma** is responsible for implementing people detection using YOLO model, by embedding a camera into a rapsberry pi module. It involved the sensor fusion.
All three of us contributed and actively participated to the final part of the project, which involves sending data over the IP and deriving the Transformation matrix to detect the object and assign an approximate value of depth.