School of Electronic
Engineering and
Computer Science

MSc Internet Of Things
Project Report 2019

People counting and tracking based on LIDAR data

SURENDER
SAMPATH

August 2019

DISCLAIMER

This report is submitted as part requirement for the degree of MSc at the University of London. It is the product of my labour except where indicated in the text. The report may be freely copied and distributed provided the source is acknowledged.

Signature:

Name: Surender Sampath

Date: 21 August 2019

ACKNOWLEDGMENTS

The project has been possible with the help of many great people who provided me with inspiration and direction. It has been a thorough enjoyment and an amazing learning experience from all the time I spent on this project. Every step of the project had different challenges and involved several key learning points from varied domains. Foremost, I would like to thank my supervisor, Dr. Eliane Bodanese, who introduced me to this exciting topic and guided me through research and the completion of this project which would not have been possible otherwise. I would also like to extend my gratitude to Mr. Fei Luo, a Ph.D. student, for his support and mentoring during the project.

ABSTRACT

Indoor localization and tracking of humans demand accurate position of subjects which can be of great use in several applications that includes activity classification and autonomous person tracking. The main objective of this project is to distinguish individual humans to count and track the subject from LIDAR (Light Detection and Ranging) data. The primary objective of this project is to develop a code to detect, track people in the environment from passive LIDAR data.

In this report, it aims to make use of a clustering algorithm for robust detection and tracking from laser scanners placed at hip height level and distinguish the humans from other objects through supervised machine learning techniques. The project involves the collection of data from HOKUYO UST-30LX scanning laser rangefinder [1]. The setup uses an Ethernet interface for communication and can obtain measurement data in a wide field up to 10 meters with millimetre resolution [1].

A supervised machine learning model has been developed to classify the detected objects to be human or non-human with the help of clustering algorithms at a success rate of 78.25 percent. It further investigates more methods and practices to improve the accuracy in-person tracking with respect to LIDAR data. The method has been implemented using the Numpy, pyKalman, pyqtgraph, scikit-learn, and other publicly available software packages.

Table of Contents

LIST OF FIGURES

1. INTRODUCTION:

1.1 OVERVIEW

The advantage of knowing the perpetual position of people in an indoor environment leads to a wide range of applications. The ability to detect, track and count people will play a crucial part in many areas including the recognition of daily living activities. The indoor localization techniques to count and track people were traditionally limited to Wi-Fi, wireless access point (WAP), and Bluetooth and were based on the principle of detecting RSSI (Received Signal Strength Indicator) at the receiver's end. The mentioned technologies suffer from non-line of site issues and radio interference [2]. The signal strength of Bluetooth and Wi-Fi estimates from 1 to 2 m of accurate position which is insufficient for several applications in a confined region. Another major drawback of these technologies is that it requires users to carry sensors in the form of an on-body device. The project focuses on an alternative solution of using LIDAR that is based on time and does not suffer from communication interferences which is similar to ultrawideband (UWB) devices that can transmit at high speeds over small distances with minimum power requirements [3]. In addition to that, it also does not require the user to wear any on-body sensors as the LIDAR device emits rapid pulses of laser light and measures the amount of time takes for each pulse to bounce back. A complex map of the surface in the two-dimensional (2D) form is built by repeating the process. Among the range of Lidars, 2D Lidar is preferred over the 1D and 3D Lidar to strike a balance between computational complexity and low-cost.

1.2 MOTIVATION

The objective of the project is to count the number of humans present in the kitchen environment using LIDAR as a sensor of measurement. The project aims to display the count of humans and track the humans in the kitchen environment from the data obtained through LIDAR data. The machine learning model will be created and trained by collecting positive and negative samples respectively. The application will then use the trained machine learning model to classify the objects in the LIDAR data to be human and non-human along with the count. Though the kitchen is the main area of experimentation, the Library is used to collect only the positive samples due to the large presence of human beings. The application running in the kitchen setup should be able to detect and display the count of humans in real-time as and when the human walks in or walks out of the kitchen.

1.3 BACKGROUND WORK:

Over the years, the primary application of Lidar has been mobile unmanned vehicles [4],[5] and robots [6] for collision detection and avoidance. The method of human detection in a closed environment is separated into two categories. One category involves users to wear on-body sensors such as GPS, Wi-Fi receiver, GPS accelerometer, Bluetooth. The other type does not require users to wear on-body due to remote sensing. This category involves cameras, Lidar, etc. Wi-Fi is increasingly becoming popular among indoor localization

techniques due to its widespread availability of infrastructure. The paper [7] focuses on the detection of humans using channel state information (CSI) in Wi-Fi networks. However, usage of LIDAR is has drawn increasing interest for its accuracy in tracking systems in fixed platforms, as indicated in papers [8] and [9]. The LIDAR tracking approach faces several challenges in distinguishing objects in the environment from human beings. For example, keeping the LIDAR stationary eliminates the background or static objects. In the case of mobile LIDAR, the removal of static objects from the environment becomes complex. The idea for this project is inspired by project [10] that focuses on the concept of person tracking implementation using robotic operating system (ROS) like many of the other existing LIDAR projects. The thesis [11] introduces the idea of joint leg tracking and evaluates it using many benchmarks. The current report aims to recreate the same design from scratch but implement the application for people counting and detection by placing the LIDAR at a hip level using python packages and the best available algorithms.

1.4 OBJECTIVE AND CONTRIBUTION

This project aims to reproduce the technique of detecting humans from LIDAR data [11] from previous work using alternative tools to propose an accurate 2D Lidar indoor location determination.

1. Use real-world data to train the supervised machine learning model.
2. To successfully track multiple people, present in the environment.
3. To successfully count the number of people, present without occlusion.

1.5 REPORT STRUCTURE:

Chapter 2 explains the requirements and design of the project. Chapter 3 introduces the concepts and describes hardware and software tools required for the understanding of this project. Chapter 4 explains the steps involved in the implementation and Chapter 5 examines the results of the system and compares with the objectives mentioned.

2. LIDAR SETUP REQUIREMENTS AND DESIGN

This section covers the requirements and design of the LIDAR setup. In general, it specifies the considerations and assumptions for the experiment of human detection and counting from LIDAR data.

2.1 PROBLEM STATEMENT AND HYPOTHESIS

The project proposes a novel method to detect and count humans using a planar laser-equipped at the hip level in real-time within the kitchen area. The machine learning model will be trained using the samples collected in the Library and the kitchen.

It focuses on the problem of accurate human counting and detection which is critical in social and assistive robotic applications and supports other applications outside the field such as security in hotel premises, rehabilitation in medical care, location-based public advertisements and beyond. The problem of interest is divided into three sub-phases: (1) Training, (2) Detection, (3) Display. For this project, HOKUYO UST-30LX is chosen, which is a popular planar laser used in several autonomous robots, especially in public environments for their accuracy and reliability. In addition to that, Laser sensing of this model is computationally feasible to process and functional under several diverse operating conditions.

2.2 USE CASE

USE CASE	STAKEHOLDERS	CHALLENGES	PROCESSES
1. HOTEL ROOM SECURITY	Customers, Hotel Authorities, Police department.	Customers use the premises of the room and services available. Authorities monitor the pattern of usage and identify abnormal patterns. Police use the data for investigation in case of an accident or crime.	Collection of data of customers inside the room. Store the collected data. Perform analysis and pattern recognition. Notify authorities and pattern results in case of crime or accident.
2. PATIENT REAHBILATION	Patients, Doctors.	Patients movement is recorded in their rehabilitation ward and stored. Medial practitioner monitors the pattern of usage and identify abnormal patterns. Doctors use the results to identify the progress of the rehabilitation program.	Collection of data of patients inside the room. Store the collected data. Perform analysis and pattern recognition over the physical movement. Doctor analyse the results and alter the rehabilitation program.

2.3 SYSTEM OVERVIEW

In the previous work [10], the user hosts the human tracking algorithm on a robot and follows a particular human subject. In this project, the application hosted on a laptop next to the LIDAR in the kitchen area where the human's movements are continuously scanned by the sensor and processed. During the detection and display phase, it highlights the detected human samples and displays the count in the application. A machine learning model developed at the end of the training phase is used for the detection and display phases later. A machine learning model is a mathematical representation of a real-world process generated from feeding the training data. The training data consists of positive and negative samples from which the model learns the patterns such that the input parameters correspond to the target. The created model can later be used to classify objects or make predictions of objects to be human or non-human during the detection phase. In this project, the positive samples are the humans and negatives samples refer to the non-human objects present in the environment. The model is referred as a classification model because the output data is categorized into predefined classes. The target is the output of the input variables, which is human or non-human in the case of this project. Features are independent variables derived from the original data that acts as an input to training or prediction from the model. The process of deriving features is known as feature extraction, and it is done to reduce the number of dimensions. Finally, in the display phase, we indicate the detected humans as well as the count in real-time

over the 2D scrolling map of the LIDAR data. This conceptual view of the system is shown in the below figure1.

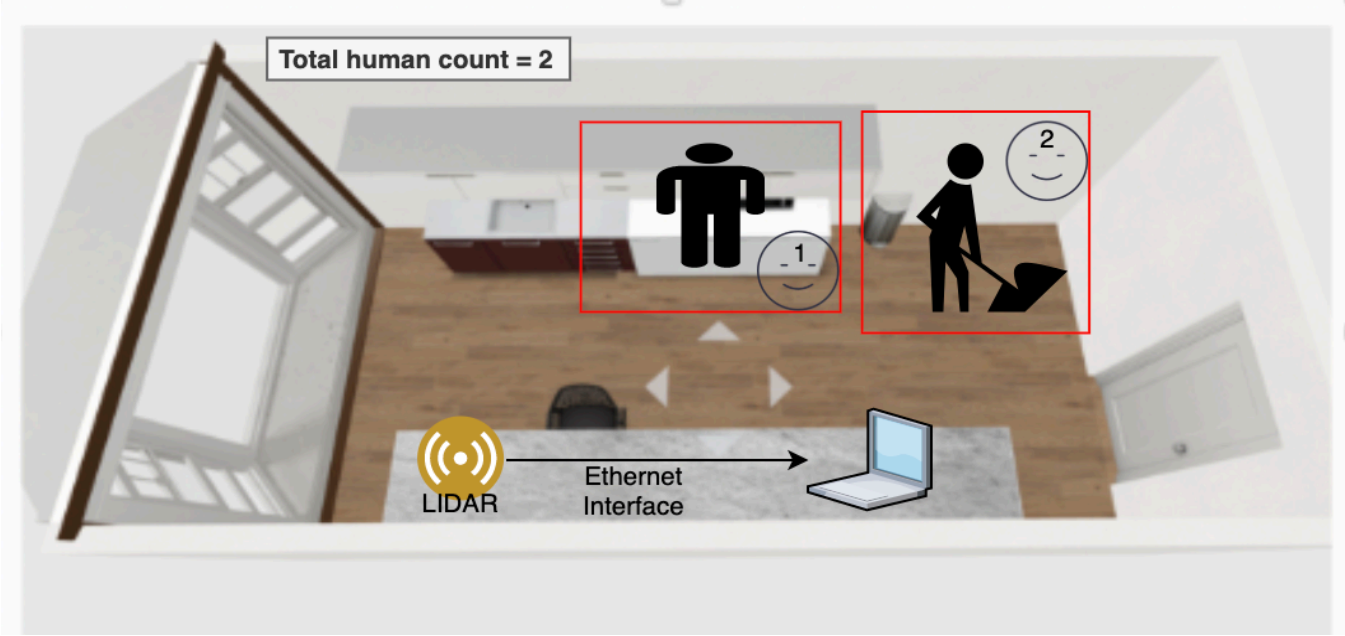

Figure 1- Conceptual View

2.4 SYSTEM REQUIREMENTS

The system requirements are further divided into functional and non-functional requirements of the setup. Non-functional requirements represent the minimum settings required for the experiment. Functional requirements define the features of the configuration needed to achieve the described design.

2.4.1 FUNCTIONAL REQUIREMENTS

PHASE	PARAMETER	REMARKS
1.TRAINING&2. DETECTION	LIDAR height	The Lidar should be placed at the height of 75 cms to detect humans at a hip level.
1.TRAINING&2. DETECTION	Field of view	Lidar should be placed in the corner of the room to allow maximum viewing angle (~90 deg). Technology supports up to 270 deg.
1.TRAINING&2. DETECTION	Planar assumption	LIDAR should be mounted on the same level as the plane of the surface. If the LIDAR is mounted slightly upwards or downwards, the reflections in the scan could degrade the performance of the application.
1.TRAINING&2. DETECTION	Measurement Distance Range	Target of 40…120 cms from the sensor for the kitchen/Library setup. Technology typically allows for max. distances in the range of 60 m.
1.TRAINING&2. DETECTION	Data processing of each sweep	1081 points of (X, Y) coordinates should be collected for each sweep/scan.
1.TRAINING&2. DETECTION	Data transmission	The LIDAR should communicate with the computer through Ethernet interface and remain connected at all times.
1. TRAINING	Positive Samples	The collected data must contain only samples of human body.
1. TRAINING	Negative Samples	The collected data must contain only samples of non-human body.
2. DETECTION	Feature Extraction	A set of 14 geometric features should be extracted for all positive and negative sample clusters with no null values.
2. DISPLAY	Human Identification	The system should identify all the detected human clusters and display a bounded box around the body.
2. DISPLAY	Human Count	The system should display the identification number and total count next to each human body.

2.4.2 NON-FUNCTIONAL REQUIREMENTS

The detection of humans from the LIDAR data application is expected to operate independently in the specified zone, which is the kitchen in this project. The LIDAR is subjected to draw power from the nearest outlet and hence, no power restrictions apply to the setup. The interaction with the LIDAR is done using the laptop and process the data in the same. The device is expected to be active throughout the experiment.

As a summary, the setup is required to be active during the time of experiment where prediction and counting of humans are done in real-time.

2.4.3 USER INTERFACE

The LIDAR setup design is in a way that there are no user controls exposed and is considered to be a passive device mounted in the kitchen to monitor humans detection throughout the period. The device itself is so quiet and does not have any sound notification or working noise. The blue LED indicator is present on the LIDAR device to indicate the normal functioning of the sensor.

2.5 HARDWARE SPECIFICATION

Specifications

Model	UTM-30LX-EW
Light Source	Laser Semiconductor λ = 905nm Laser Class 1
Power Source	12VDC ± 10%
Supply Current	MAX : 1A, Normal : 0.7A
Power Consumption	Less than 8W
Detection Range and Detection Object	Guaranteed Range: 0.1 to 30m(White Kent Sheet)*2 Maximum Range: 0.1 to 60m Minimum detectable width at 10m: 130mm(Vary with distance)
Accuracy	0.1 to 10m : ±30mm,10 to 30m : ±50mm(White Kent Sheet)*2 Under 3000lx : White Kent Sheet : ±30mm*1(0.1 to 10m) Under 100000lx : White Kent Sheet : ±50mm*1(0.1 to 10m)
Measurement Resolution and Repeated Accuracy	1mm 0.1 to 10m : σ<10mm,10 to 30m : σ<30mm(White Kent Sheet)*2 Under 3000lx : σ=10mm*1(White Kent Sheet up to 10m) Under 100000lx : σ=30*1(White Kent Sheet up to 10m)
Scan Angle	270°
Angular Resolution	0.25°(360°/1440)
Scan Speed	25ms(Motor speed : 2400rpm)
Interface	Ethernet 100BASE-TX(Auto-negotiation)
Output	Synchronous Output 1-Point
LED Display	Green : Power supply. Red : Normal Operation(Continuous),Malfunction(Blink)
Ambient Temperature	-10 to +50 degrees C
Ambient Humidity	Less than 85%RH(Without Dew,Frost)
Storage Temperature	-25 to +75 degrees C

Figure 2 - Hardware Specification [1]

2.6 DEVELOPMENT REQUIREMENTS AND DESIGN SUMMARY

The LIDAR setup is designed to meet the required functional and non-functional requirements. The installation should be active and continuously functioning during the monitoring period. It is expected to work under ideal kitchen environment or any other similar indoor environment to produce consistent results. The key aspect for achieving accurate results is by collecting the positive and negative samples in the same confined region of the experiment. This practice adds to the accuracy of the 2D map generation.

In this implementation, the setup does not require any additional configuration or alteration to the environment. It integrates into any existing environment without any further commotion. The setup is adapted to perform and produce similar results in real-time as much as it does on passively processed data. The application code is designed to log and display error messages for easy debugging purposes continually and is well documented on GitHub to track every feature and changes added to the code during the development process. The project is also divided into independent components to handle data wrangling, supervised machine learning, and human detection functions for easy implementation and troubleshooting. Each functionality is meant to be replaced or updated independently.

3. TOOLS REQUIRED FOR HUMAN DETECTION AND COUNTING

3.1 WORKING OF LIDAR DEVICE

LIDAR (Light Detection and Ranging) sensors are very prevalent and familiar in the field of navigation and mapping. The LIDAR investigated in the project is the Hokuyo UST-30LX Scanning Laser Range Finder, as shown in Figure 3. The figure also represents the summary of the sensor's field of regard where output is plotted as a 2D point cloud.

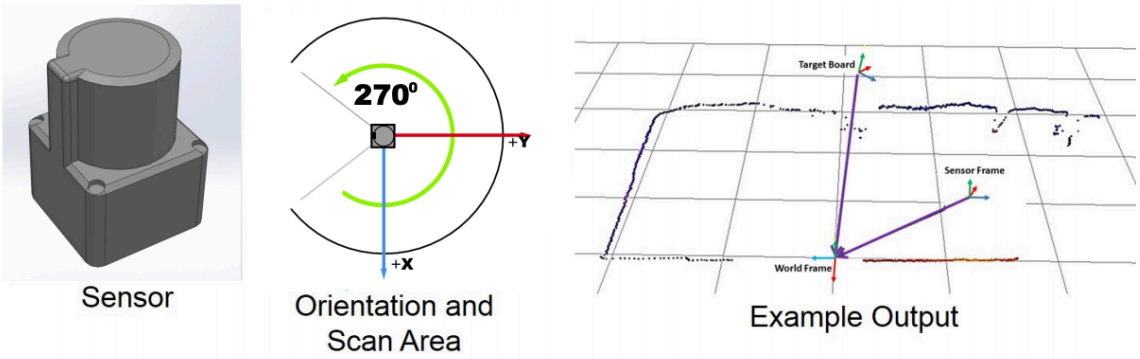

Figure 3 Hokuyo UST-30LX sensor summary, adapted from [12]

The basic principle the LIDAR is that it emits light at the surface and measures the time taken for it to reflect and then return to its source. The instrument fires up to 150,000 pulses per second. The sensor present on the LIDAR measures the time taken by the light signal to

bounce back. With the known speed of light, which is constant, the LIDAR is capable of measuring a very accurate distance between the object and itself. The device houses smart electronics with paired optics and covered with a protective housing. In 2D Lidar, the laser diode is pointed down with a rotating motor at the bottom. There is also a mirror on the rotating motor to deflect the laser beam outwards and another mirror below to deflect the incoming laser beam down to the sensing electronics. The mounted motor has a scanning window of 270 ° (90° is the blind spot) [1]. The objects should be within a maximum detectable distance of 20 m at a high angular resolution of 0. 25° [1]. The term SCAN is defined as the data collected in a single sweep of the rotating sensor between 0° and 270°. The area swept is segmented into 1081 segments, to get the 1081 potential measurements. The sensor readings are modified to only output a section of 1081 data points, with a minimum sweep size of two measurements. The HOKUYO sensor used in the experiment is classified as Class 1 Laser Device, and the output is less than one mW and hence can be considered to operate without demanding safety gears or equipment. Simultaneous localization and mapping (SLAM) are a critical component in several robotic applications. SLAM is described as the problem of estimating the robot pose and at the same time, reconstructing the movement in the environment incrementally [13].

3.2 PYTHON LIBRARIES

Numpy is a scientific library for use in python. The Library provides excellent functionality in providing high-performance when dealing with multidimensional array objects of the same type. It is observed that the use of Numpy over python lists offers the following benefits [14]. Hence, it was chosen to deal with data during the project.

- Functionality – SciPy, and NumPy have optimized functions such as linear algebra operations built-in.
- Memory – Numpy consumes less memory as lists in python requires an additional 8 bytes to reference a new object.
- Speed – With the above characteristics, it is common that Numpy performs better than core python components.

Pyqtgraph is a user interface library intended for python applications with strong graphic capabilities which is dependent on the QT bindings. This is more suitable for science and engineering applications for its fast, interactive ability in displaying data (plots, videos) [14]. This library also supports rapid prototyping and development such as Qt designer. Following are the core features of the Pyqtgraph library [15].

- Basic data visualization.
- Realtime data/video plotting.
- Interactive Scaling and plot selection.
- Widgets for marking/selection of image or region of interest.
- Extend the GUI to web and other applications.

In Comparison with other popular libraries, Matplotlib is considered to be a more mature library intended for publication-quality graphics, which is comparatively slower than pyqtgraph [15]. The pyqtgraph serves more towards the field of data acquisition. Though matplotlib popular among MATLAB developers, pyqtgraph is generally aligned towards python and qt

programmers. Pyqtgraph also provides additional features such as flowcharts, trees, and volumetric rendering [15].

3.3 SUPERVISED MACHINE LEARNING

In supervised machine learning, a group of labelled samples is the training data with known category, and the model learns from the correctly identified samples. The individual observations converted into a quantifiable property is known as the features. The algorithm that performs the classification is called the classifier.

Clustering Analysis is the technique of grouping objects based on their similar characteristics. It is one of the core concepts behind machine learning, pattern recognition, and image analysis. As the real-world data is more dynamic and unpredictable, the supervised machine learning models developed can classify and correct the data based on commonness among the data. Some of the famous and standard clustering algorithms are discussed and compared in the following sections. The following chapters examine and analyses the two types of clustering algorithms and their usability. The target variable of a dataset is the feature of a dataset about which the application wants to predict. A feature is an individual measurable property or characteristic of a phenomenon observed among the data.

3.3.1 K-Means:

K-Means is an efficient way of clustering in terms of computing complexity and is the main reason for its high popularity. However, they do not perform at their best to identify classes

when data is not of spherical distribution. The K-Means algorithms aim to find and group the data points that have high similarity between them. In terms of the algorithm, the similarity is based on the distance between data points. The closer the data points are, the more similar and more likely to belong to the same cluster [16]. The other challenges involved with the K-Means is that the number of clusters must be known for the algorithm beforehand. The output of the algorithm is not always the same as the centroids are set randomly every time. As stated before, it expects the data to be of a spherical result, which is not possible for LIDAR data. The number of clusters is challenging to identify as it depends on the number of objects in the environment.

3.3.2 Density-based Spatial clustering of Applications with Noise (DBSCAN):

This clustering algorithm is highly famous for its ability to identify the noise, which is more prevalent in real-world data. It is based on several points with a specified radius ε, and there is a unique label assigned to each data point. The steps involved in the process of assigning a label as follows [16]:

1. Consider a minimum specified number (MinPts) of neighbour points to form a cluster.
2. A core point is assigned if there is this MinPts number of points that fall within ε radius.
3. Every other point is considered as noise points.

DBSCAN fits the task of clustering points based on their space locations [17]. In this project, the maximum radius (E) of the neighbourhood is set to 100 cms, and the minimum points (MinPts) in the neighbourhood is set to 120 based on the size of the object and review of the sample data. One comparison with K-Means, DBSCAN has the following advantages and suits to process LIDAR data more due to the following benefits [17].

- In k-means clustering, each cluster is identified by a centroid. The points are assigned to the nearest centroid available. In the case of DBSCAN, there are no concept of centroids, and clusters are formed by grouping nearby points to one another.
- Unlike K-means, DBSCAN does not require number of clusters as parameter. It requires only two parameters which influences the decision of whether two nearby points should be linked into the same cluster. These two parameters are - distance threshold, ε (epsilon), and “MinPts” (minimum number of points).
- k-means loops over several iterations to coincide on a good set of clusters, and cluster assignments varies on different configuration. DBSCAN makes only a single pass through the data, and once a point has been assigned to a particular cluster, it never changes.

Point clouds are a collection of points that represent a 3D/2D shape or feature. Each point has its own set of (X, Y) coordinates and in some cases, additional attributes. The point cloud is a collection of multiple points. In other words, when many points are brought together, they start to show some exciting qualities of the feature that they represent.

3.3.3 RANDOM FOREST CLASSIFIER

This is a tree-based ensemble classifier which is a meta estimator that fits several decision trees on various subcategories of data. This algorithm is used for both regression and classification. In general, the number of trees determines the robustness of the forest. Similarly, the greater the number of trees higher the accuracy of the random forest algorithm. However, this comes with the cost of computational power. The advantages of using a random forest classifier are:

- A Random forest classifier is robust to null values.
- More number of trees would not necessarily overfit the model.
- The Random forest classifier can also be used for categorical data.

The basic concept of a decision tree is a rule-based method. When the algorithm is provided with targets and features, the decision tree returns the set of rules which is later used for prediction. The algorithm calculates the nodes and forms the rules using information gain and Gini index calculations. Each tree constructed shall emit a particular set of attributes for the input data to be predicted. The average attribute is utilized to attain better predictive accuracy and to avoid over-fitting. Since this technique does not use feature space transformation, the computation time has decreased to a large extent [18]. A decision tree is the most fundamental block of the random forest. In the CART (Classification and Regression) algorithm, the decision tree is created by identifying the questions when answered, results in a more

significant reduction known in Gini Impurity. This indicates that the decision trees form nodes of a higher proportion of sensor data from a single category or a class by finding values in the features that divide the class. It should also be noted that the drawback of a decision tree is learning the data ultimately may result in overfitting. The random forest is made up of several decision trees. The concept of merely averaging the prediction of trees refers to the forest model. Random forest model does the following.

1. Random sampling of training data points when building trees

2. Random subsets of features considered when splitting nodes

During training, each tree present in the forest learns from a random sample of data points. The sample drawn by replacement is known as bootstrapping. This could have the same values being utilized again in the same tree. The concept is that by training each tree on different samples, although each tree might have high variance concerning a particular set of the training data. Overall, the entire forest results in lower variance but not at the cost of increasing the bias. The random forest combines hundreds of decision trees, trains each one with a slightly different set of the observations. It then splits nodes in each tree considering a limited number of the features. The ultimate predictions of the random forest are achieved by averaging all the predictions from each tree.

The random forest classifier performs better when evaluated against AdaBoost and Support Vector machines [19]. The algorithm uses the geometric features extracted from the data, as

explained in the previous section as a criterion to sample the cluster. The 14 features extracted from each cluster are taken into account to classify the incoming samples. In the process of training, the model develops into a multitude of decision trees. The output of the algorithm is a scored label which is either 1 or 0 with 1 representing the cluster as human and 0 as non-human.

3.4 CONCLUSION

This background section described the core concepts used behind this project and reasons for why the particular tools and algorithms are chosen for the same. All the concepts explained are written using the python programming language.

4. IMPLEMENTATION

4.1 OVERVIEW

The main objective of the project is to track and count the number of people present in the environment, such as a kitchen floor using a LIDAR. The application consists of LIDAR hardware to collect the data which is processed using clustering algorithms as discussed in previous section on the computer and then displays the result. Suppose a person or more walks into the kitchen where the LIDAR setup is installed, the LIDAR which is constantly scanning the environment records the data and sends it to the computer for processing. The computer shows a 2D map of objects detected in the kitchen environment and highlights the humans present. It also indicates the total count of humans present in the current frame of detection. This section covers the steps to collect and process the data as seen in Figure 4.

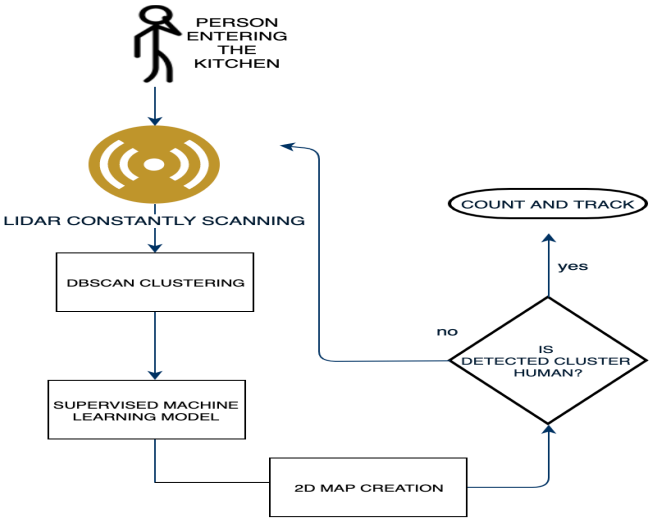

Figure 4- Process flow human detection using LIDAR

4.2 EQUIPMENT AND TOOLS

- LIDAR -The UST-30LX scanning laser rangefinder, a low power, a small, accurate, high-speed device for obstacle detection device is used in the project.
- SCIKIT-LEARN - Scikit learn libraries are used for clustering and supervised machine learning model.
- PYQTGRAPH - Pyqtgraph library is used for 2D mapping, labelling, and visualization.
- LAPTOP - A laptop is used in this project to process the LIDAR data and display results.

4.3 DEVELOPMENT

4.3.1 Setting up the LIDAR

The data collected from the LIDAR is processed passively for the ease of development and debugging purposes. Data is collected from HOKUYO 30-LX through Ethernet interface using the tool URGBenriplus software as displayed in figure 6. The data collection procedure is done in a kitchen setup similar to the graphical representation, as shown in Figure 5.

Figure 5-Kitchen Setup [20]

The UrgBenriPlus tool is provided by hardware manufacturer of the LIDAR itself, and the dataset is stored locally on the laptop. Setting up of the LIDAR is an easy step. The first step is to connect the LIDAR to a power outlet and Ethernet cable from the LIDAR to the Ethernet port of the laptop. The blue light is on the LIDAR indicates that the device is properly functioning. Click on the connect button the UrgBenri tool, which usually takes a few minutes to connect. The IP address of the LIDAR is 192.168.0.10 by default to connect the device unless changed otherwise. The LIDAR data is instantly shown on the viewer once the connection is successful. The data in the viewer can be saved offline in the (X, Y) format to be later utilized for pre-processing.

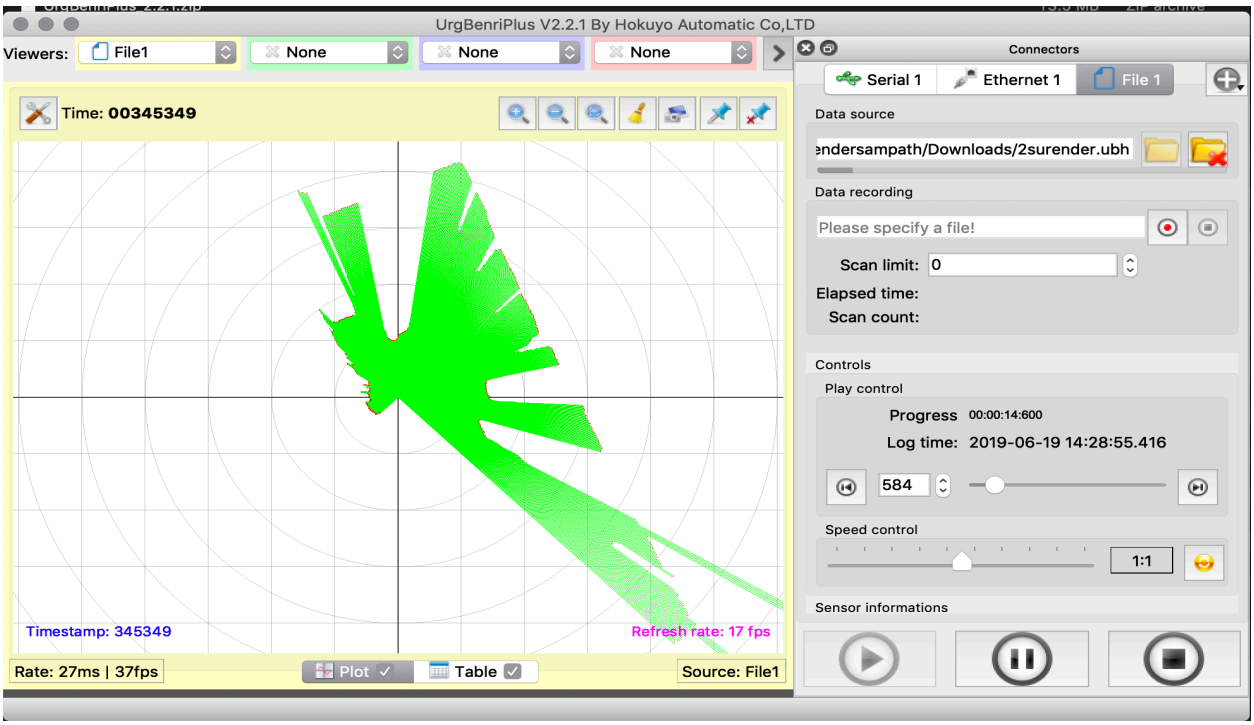

Figure 6- URBgener tool.

4.3.2 LIDAR DATA WRANGLING

Data wrangling is the process of cleaning complex datasets for easy access and analysis. So, it is important to transform the available raw data into another format to make it more appropriate and valuable. The data set contains a set of 1081 points of (X, Y) coordinates for each scan. The number of scans is proportional to the duration of the data collected. The first column represents the scan ID, and the subsequent columns represent the pair (X, Y) of co-ordinate, as seen in figure 7.

	A	B	C	D	E	F	G	H	I	J
1	332288	(-1516.74;-1516.74;0)	(-1508;-1521.22;0)	(-1497.14;-1523.5;0)	(-1483.5;-1522.86;0)	(-1472.68;-1525;0)	(-1459.09;-1524.19;0)	(-1418.7;-1495;0)	(-1399.83;-1488.06;0)	(-1376.95;-1476.6;0)
2	332313	(-1534.42;-1534.42;0)	(-1534.05;-1547.49;0)	(-1527.28;-1554.17;0)	(-1516.3;-1556.52;0)	(-1490.04;-1542.98;0)	(-1475.69;-1541.52;0)	(-1439.35;-1516.76;0)	(-1429.29;-1519.38;0)	(-1412.42;-1514.63;0)
3	332338	(-1719.68;-1719.68;0)	(-1694.56;-1709.42;0)	(-1668.16;-1697.54;0)	(-1607.01;-1649.64;0)	(-1540.06;-1594.78;0)	(-1508.88;-1576.2;0)	(-1488.91;-1568.98;0)	(-1466.98;-1559.44;0)	(-1436.29;-1540.23;0)
4	332363	(-1826.46;-1826.46;0)	(-1832.55;-1848.61;0)	(-1788.02;-1819.5;0)	(-1739.59;-1785.74;0)	(-1626.2;-1683.97;0)	(-1551.76;-1620.98;0)	(-1504.05;-1584.94;0)	(-1488.9;-1582.75;0)	(-1479.25;-1586.31;0)
5	332388	(-1832.82;-1832.82;0)	(-1828.33;-1844.35;0)	(-1821.66;-1853.74;0)	(-1773.78;-1820.84;0)	(-1717.2;-1778.21;0)	(-1578.03;-1648.43;0)	(-1547.42;-1630.64;0)	(-1518.37;-1614.07;0)	(-1494.94;-1603.13;0)
6	332413	(-1837.77;-1837.77;0)	(-1830.44;-1846.48;0)	(-1828.67;-1860.87;0)	(-1824.02;-1872.41;0)	(-1736.65;-1798.35;0)	(-1672.77;-1747.4;0)	(-1563.25;-1647.33;0)	(-1535.5;-1632.28;0)	(-1525.63;-1636.04;0)
7	332438	(-1841.31;-1841.31;0)	(-1834.66;-1850.74;0)	(-1826.57;-1858.73;0)	(-1806.58;-1854.51;0)	(-1764.43;-1827.12;0)	(-1696.97;-1772.68;0)	(-1588.03;-1673.44;0)	(-1543.72;-1641.02;0)	(-1540.63;-1652.13;0)
8	332463	(-1837.77;-1837.77;0)	(-1826.92;-1842.93;0)	(-1815.35;-1847.32;0)	(-1807.28;-1855.22;0)	(-1782.49;-1845.83;0)	(-1714.95;-1791.46;0)	(-1601.8;-1687.95;0)	(-1541.66;-1638.83;0)	(-1529.72;-1640.43;0)
9	332488	(-1846.26;-1846.26;0)	(-1850.85;-1867.08;0)	(-1843.39;-1875.85;0)	(-1805.18;-1853.07;0)	(-1777.63;-1840.79;0)	(-1724.63;-1801.58;0)	(-1625.21;-1712.61;0)	(-1549.88;-1647.58;0)	(-1544.04;-1655.78;0)
10	332513	(-1843.43;-1843.43;0)	(-1838.18;-1854.29;0)	(-1829.37;-1861.58;0)	(-1816.35;-1864.53;0)	(-1805.42;-1869.56;0)	(-1742.61;-1820.36;0)	(-1683.03;-1773.54;0)	(-1562.22;-1660.69;0)	(-1526.31;-1636.77;0)
11	332538	(-1846.26;-1846.26;0)	(-1836.77;-1852.87;0)	(-1827.97;-1860.16;0)	(-1812.86;-1860.95;0)	(-1804.72;-1868.84;0)	(-1782.72;-1862.25;0)	(-1779.4;-1875.09;0)	(-1595.79;-1696.38;0)	(-1531.09;-1641.89;0)
12	332563	(-1839.18;-1839.18;0)	(-1830.44;-1846.48;0)	(-1823.77;-1855.88;0)	(-1817.05;-1865.25;0)	(-1803.33;-1867.41;0)	(-1784.8;-1864.42;0)	(-1744.29;-1838.1;0)	(-1638.96;-1742.26;0)	(-1542.68;-1654.32;0)
13	332588	(-1853.33;-1853.33;0)	(-1842.41;-1858.56;0)	(-1834.98;-1867.29;0)	(-1820.54;-1868.83;0)	(-1806.81;-1871;0)	(-1788.25;-1868.03;0)	(-1775.27;-1870.74;0)	(-1663.62;-1768.48;0)	(-1550.18;-1662.37;0)
14	332613	(-1846.26;-1846.26;0)	(-1840.29;-1856.42;0)	(-1818.86;-1850.88;0)	(-1810.77;-1858.8;0)	(-1807.5;-1871.72;0)	(-1799.32;-1879.59;0)	(-1764.94;-1859.86;0)	(-1658.14;-1762.66;0)	(-1554.96;-1667.49;0)
15	332638	(-1834.94;-1834.94;0)	(-1830.44;-1846.48;0)	(-1825.87;-1858.02;0)	(-1827.51;-1875.99;0)	(-1812.36;-1876.76;0)	(-1794.48;-1874.53;0)	(-1762.88;-1857.68;0)	(-1717.75;-1826.03;0)	(-1558.37;-1671.14;0)
16	332663	(-1832.82;-1832.82;0)	(-1831.14;-1847.19;0)	(-1820.26;-1852.31;0)	(-1811.46;-1859.52;0)	(-1800.55;-1864.53;0)	(-1793.78;-1873.81;0)	(-1780.08;-1875.82;0)	(-1679.38;-1785.24;0)	(-1583.6;-1698.2;0)
17	332688	(-1843.43;-1843.43;0)	(-1830.44;-1846.48;0)	(-1825.17;-1857.3;0)	(-1802.39;-1850.21;0)	(-1804.03;-1868.13;0)	(-1795.17;-1875.26;0)	(-1776.64;-1872.19;0)	(-1673.22;-1778.68;0)	(-1593.83;-1709.17;0)
18	332713	(-1834.94;-1834.94;0)	(-1837.48;-1853.58;0)	(-1837.78;-1870.14;0)	(-1832.4;-1881.01;0)	(-1795;-1858.77;0)	(-1793.78;-1873.81;0)	(-1793.16;-1889.6;0)	(-1746.53;-1856.62;0)	(-1559.73;-1672.61;0)
19	332738	(-1857.57;-1857.57;0)	(-1835.37;-1851.45;0)	(-1827.97;-1860.16;0)	(-1827.51;-1875.99;0)	(-1811.67;-1876.04;0)	(-1784.8;-1864.42;0)	(-1761.5;-1856.23;0)	(-1682.12;-1788.15;0)	(-1542.68;-1654.32;0)
20	332763	(-1849.79;-1849.79;0)	(-1841;-1857.13;0)	(-1827.27;-1859.44;0)	(-1823.33;-1871.7;0)	(-1808.89;-1873.16;0)	(-1790.33;-1870.2;0)	(-1743.6;-1837.37;0)	(-1617.72;-1719.68;0)	(-1553.59;-1666.02;0)
21	332788	(-1842.01:-1842.01:0)	(-1841.7:-1857.84:0)	(-1833.58:-1865.86:0)	(-1822.63:-1870.98:0)	(-1812.36:-1876.76:0)	(-1805.54:-1886.09:0)	(-1737.41:-1830.84:0)	(-1579.35:-1678.9:0)	(-1546.77:-1658.71:0)

Figure 7 - Desired dataset in X, Y format.

The dataset is then converted into a 2-D array using numpy library with dimensions = no of scans x 1081 x 2 using the code snippet as shown in Figure 8.

```
#-Data forming - start
filepath="d1_sorted.csv"
def parsecoord(coortxt):
    coortxt=coortxt[1:-2]
    xy=coortxt.split(';')
    x=float(xy[0])
    y=float(xy[1])
    return x,y
with open(filepath,"r") as file:
    line=file.readline()
    scanlist=np.array([]).reshape(0,1080,2)
    while line:
        ll=line.split(',')
        ll=ll[1:-1]
        nump=len(ll)
        scan=np.array([]).reshape(0,2)
        for i in range(nump):
            coortex=ll[i]
            x,y=parsecoord(coortex)
            xy=[[x,y]]
            scan=np.append(scan,xy,axis=0)
        line=file.readline()
        scan=np.reshape(scan,(1,1080,2))
        scanlist=np.append(scanlist,scan,axis=0)
```

Figure 8- Code snippet for data wrangling

Figure 9 shows the terminal results showing dimensions of the resulting 2D array created from the LIDAR dataset. The 2D element created in the program is used later for clustering and plotting purposes.

```
scan number :  3540
scan number :  3541
scan number :  3542
scan number :  3543
End of scans
(35208, 14)
--- 558.4434361457825 Time taken ---
completed......

Process finished with exit code 0
```

Figure 9 - Data wrangling results

4.3.3 LIDAR DATA PLOTTING AND CLUSTERING

On using pyqtgraph library, the LIDAR points are visualized as scatter plots on a 2D moving map, and scrolling display is generated for all scans recorded by looping through the formatted 2D array of LIDAR points and plotting the (X, Y) coordinates. This is done by updating the *update* plot function in the library.

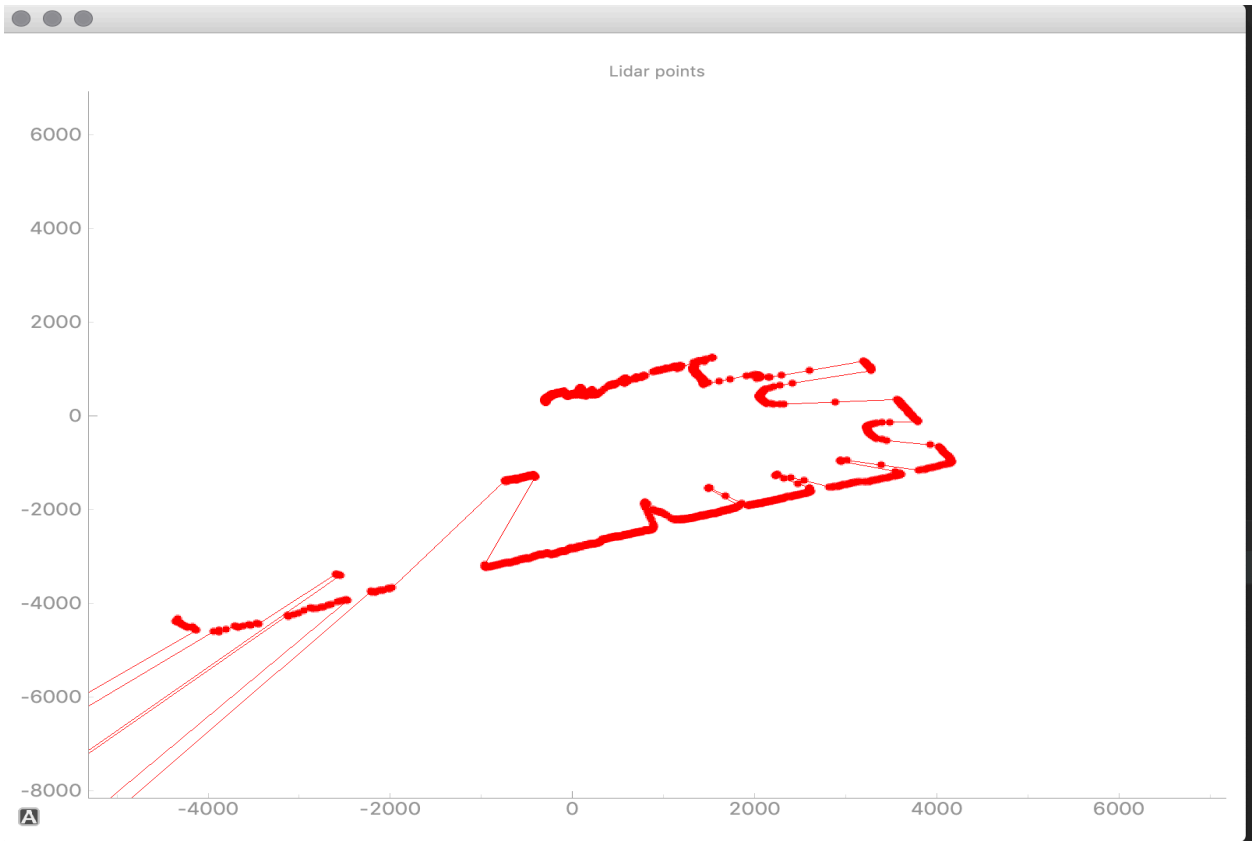

Figure 10 - Plotting of scans

In Figure 10, the plot shows the 2D map is created from the coordinates of each scan. DBSCAN clustering method is used to group the points based on the distance using the MinPts and eps value. The values are decided based on the size of the objects and review of samples present in the test environment. Figure 11 represents the code snippet used for implementing the DBSCAN algorithm.

```
# Compute DBSCAN
db = DBSCAN(eps=100, min_samples=15).fit(centers)
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
labels = db.labels_

# Number of clusters in labels, ignoring noise if present.
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
n_noise_ = list(labels).count(-1)
label_list = set(labels)
```

Figure 11 - Code snippet for clustering

To distinguish the clusters generated using the DBSCAN method, a list of random colour codes is generated at the beginning of the program. Each of the colours represent a single cluster or a cloud of points in the scan. Now as opposed the plotting in Figure 10 where it plots all co-ordinates of a scan consecutively, it now plots each of the coordinates of clusters discovered in each scan of 1081 points using unique colours. The number of clusters discovered in each scan may vary as per the movement of objects entering and exiting the environment. Figure 12 shows the clustered objects discovered in the scan each represented by an individual colour.

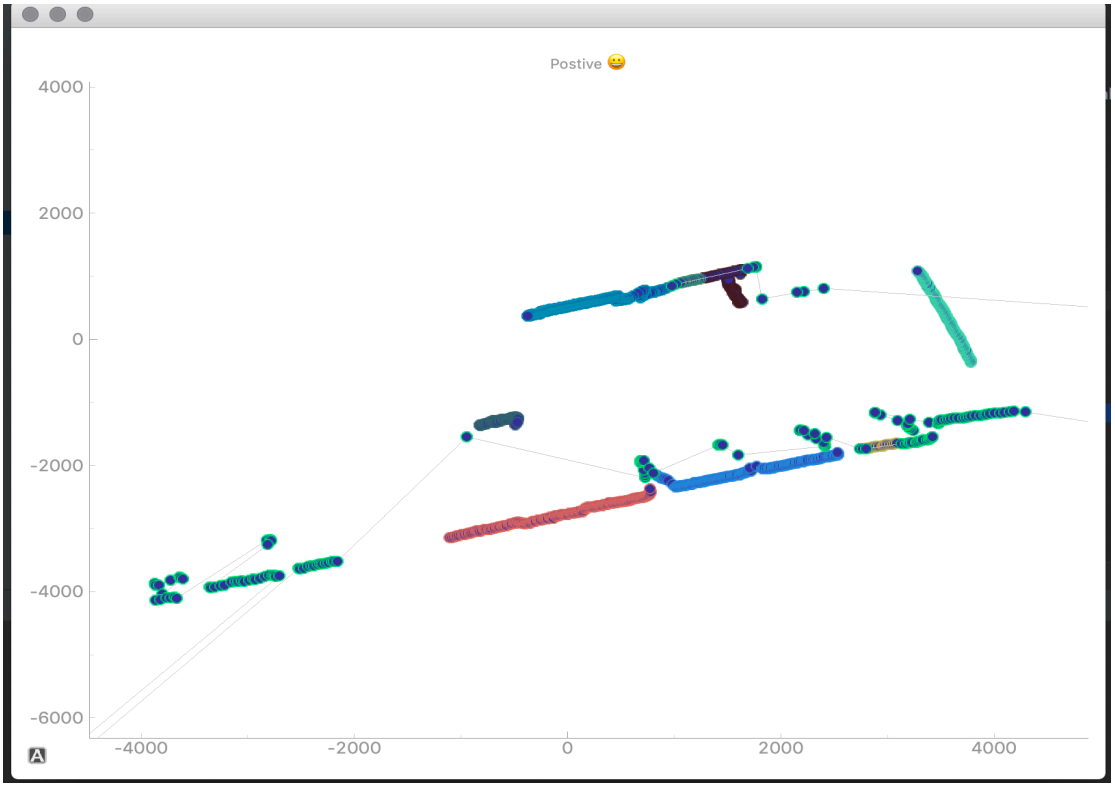

Figure 12- Clustering objects using unique colours

4.3.4 TRAINING THE PREDICTION MODEL

Once the collection of data, clustering of objects is done, the next step is to train the model to predict if the detected cluster is human or not. The Random forest classification algorithm is chosen for this particular project as it has been receiving significant attention in the field of image classification, pattern recognition for its computational efficiency, and robustness [21]. As with any classifier, the random forest machine learning model also learns the features from the supervised data and can predict the category from the series of binary decisions from the

features given [22]. The predicted output categories in this project will either be 0 or 1, indicating non-human and human, respectively. Therefore, it is very important to train the model with accurate samples.

When the input to an algorithm is too large, it can reduce the effectiveness of the model itself. This is because in many cases, additional data attributes may contribute meaningless information for the model to make a prediction. With regards to the project, it is redundant to provide all the information of a cluster to the model. Also, the feature extraction technique helps in reducing the number of resources required to represent a large dataset. Hence, the program extracts a set of 14 features from each cluster, which is then provided as an input to the model. The selected features represent the cluster geometry to differentiate among the clusters. The formulas to calculate features and techniques are used from the [23] where a similar tracking algorithm is implemented in C++ and open CV. Figure 13 represents a snippet of a function where a set of 14 features are returned for each cluster of data. Cluster size, a standard difference from the mean, the average mean difference, width, linearity, circularity, radius, boundary regularity, mean curvature, angular difference, inscribed angular variance, standard inscribed angular variance, distance, distance/cluster size are the 14 features extracted from the cluster.

```
features=[clustersize, std, avg_med_dev, width, linearity, circularity,
                    radius, boundary_regularity, mean_curvature, ang_diff, iav, std_iav,
                    distance, distance/clustersize]
return features
```

Figure 13 - Code snippet for feature extraction

4.3.4.1 COLLECTION OF POSITIVE AND NEGATIVE SAMPLES

The model has to be provided with accurate samples for classification of humans and non-humans. University Library is chosen as an ideal place to collect samples for humans as the places involve the large movement of humans. Figure 14 shows the setup is where the samples of humans which are positive samples collected during the experiment.

Figure 14 - Positive sample collection setup - Library.

As the library includes several other objects like pillars, tables, reception desk of negative samples, it is important to filter out the human samples to record the features.

```
for label in label_list:

    index = labels == label
    cluster = scan[index]
    # print(cluster.shape)

    # x_displayed = xy_dat[((xy_dat[:, 0] > min) & (xy_dat[:, 0] < max))]

    x1 = [2350, 2350, 6000, 6000, 2350]
    y1 = [-10000, 4500, 4500, -10000, -10000]

    self.plotItem.plot(x1,y1,pen='r')
    # c1 = self.plotItem.plot(cluster[:, 0], cluster[:, 1], symbol='o', symbolPen=colors[label], name='red', symbolSize=5)


    print(cluster[cluster[:,0] > ])
    c1 = self.plotItem.plot(cluster[:, 0], cluster[:, 1], symbol='o', symbolPen=colors[label], name='red', symbolSize=5)
    if np.all(cluster[:, 0] > 2350) & np.all(cluster[:, 0] < 6000):
        if np.all(cluster[:, 1] > -10000) & np.all(cluster[:, 1] < 4500):
            c1 = self.plotItem.plot(cluster[:, 0], cluster[:, 1], symbol='o', symbolPen=colors[label], symbolSize=5)
            features = getFeatures(cluster[:, 0], cluster[:, 1], cluster.shape[0])
            featureslist = np.append(featureslist, np.array(features).reshape(1, 14), axis=0)
```

Figure 15- Code snippet for virtual perimeter setup.

Figure 15 shows the code snippet for a virtual perimeter with fixed coordinates to be created. Only the clusters that fall within the defined range are considered for feature extraction. The coordinates of the perimeter are chosen in such a way to collect a maximum number of positive samples without any interference from negative or non-human objects. The perimeter does not necessarily have to be rectangular as it can also be made polygonal based on the location of the site. Figure 16 shows the virtual perimeter set up through software, and only the humans passing through the perimeter is considered to extract positive samples. The rectangle in red is the virtual perimeter, and the human clusters can be seen entering and leaving.

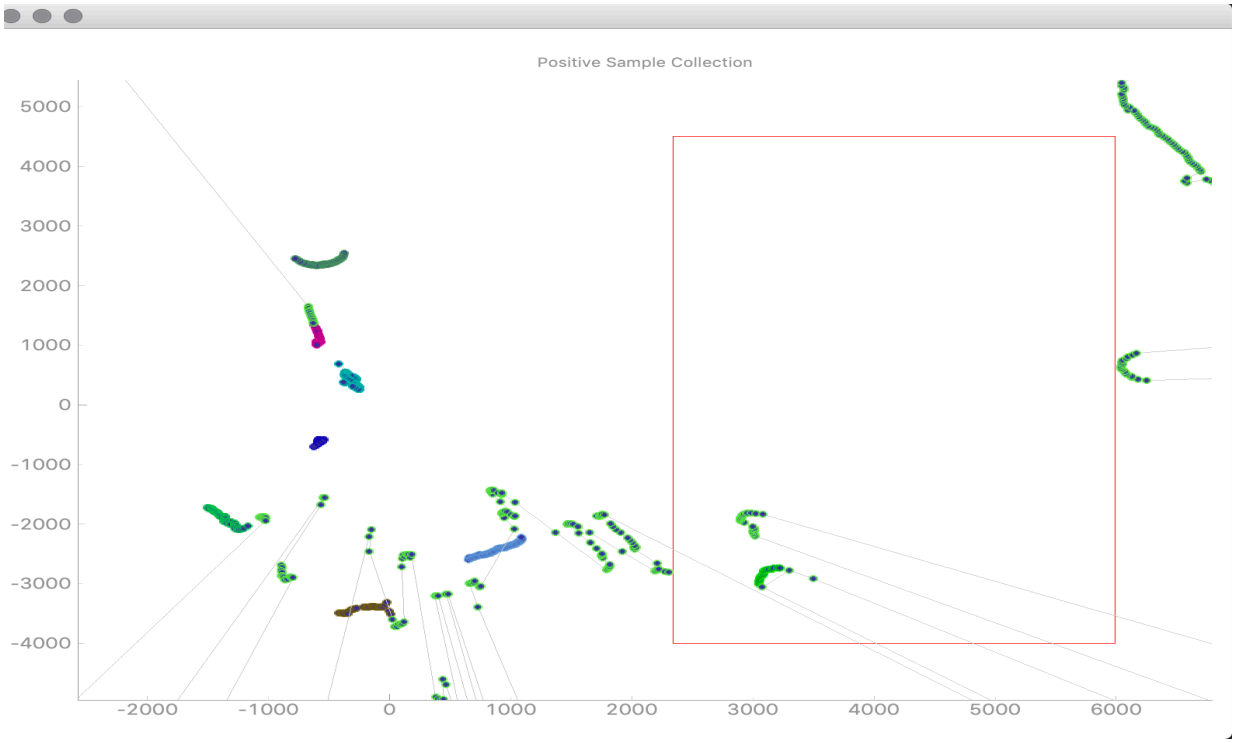

Figure 16 - Positive sample collection setup using virtual perimeter.

Any samples which are outside the perimeter is ignored. For the collection of negative samples, the same procedure is repeated without a virtual perimeter by placing the LIDAR setup on the surface of the floor.

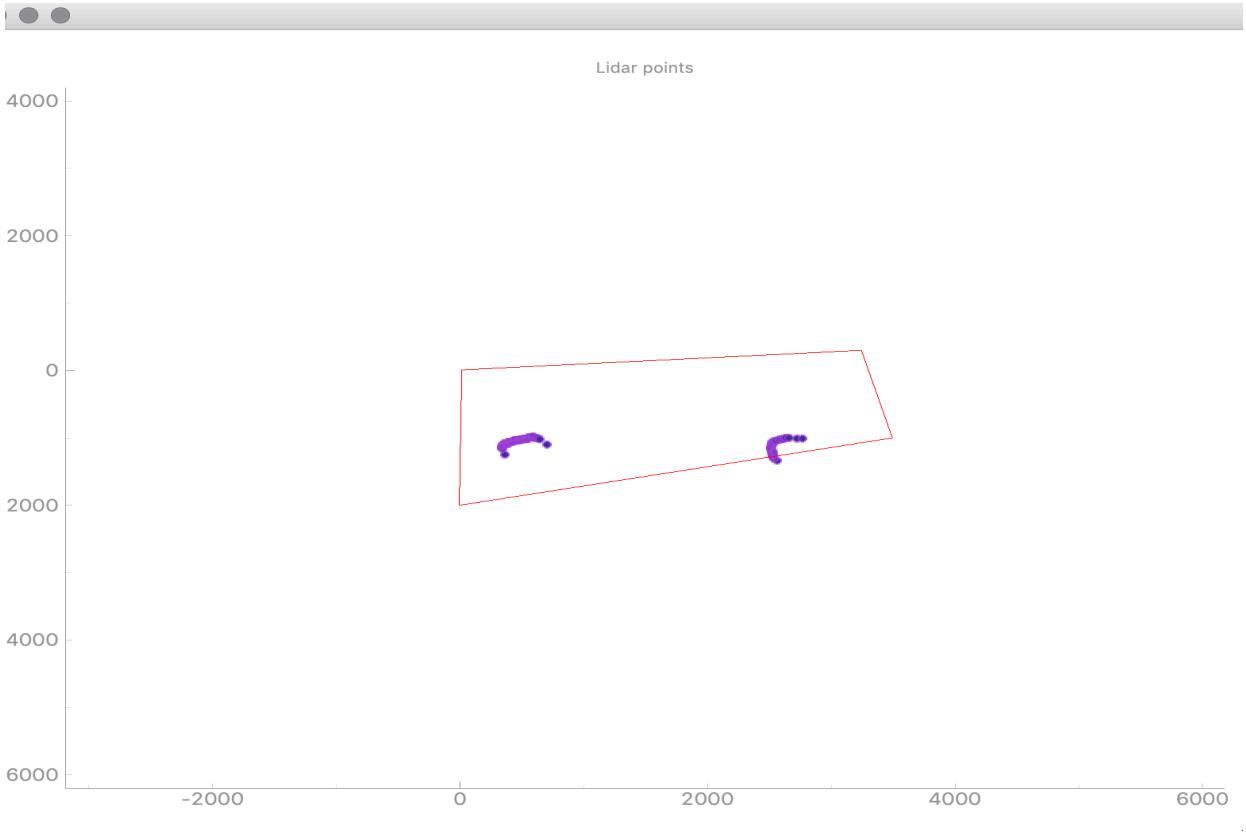

Figure 17 - Positive sample collection - Kitchen

Even though there is a presence of humans or positive samples within the region, since it is placed on the surface of the floor, the features extracted would differ and can be considered as negative samples. After the collection of positive and negative samples, the next step is to feed into the model. The entire collection procedure is repeated in the kitchen location as showing Figure 17 in order to improve the accuracy with respect to the location.

4.3.4.2 IMPLEMENTING RANDOM FOREST CLASSIFIER

The implementation of random forest classifier is done using the scikit-learn library. As seen in figure 18, the positive and negative sample datasets are serialized and de-serializing as python objects using the pickle library. Pickling" is the process whereby a Python object hierarchy is converted into a byte stream, and "unpickling" is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy [24]. There are multiple fractions of positive datasets as the human samples are collected over a different period in multiple locations.

Creating the machine learning model is executed as a separate program where the training and fitting are done using the extracted feature datasets of positive and negative samples. The output model is pickled so that it can be used for prediction in the main application program. This method saves computation time in loading and unloading large datasets.

```
import pickle
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import numpy as np
filename = 'rfcmodel'
outfile = open(filename, 'wb')



#Importing featurelist p1,p2,p3,p4,n1
filename = 'models/modelp1'
infile = open(filename,'rb')_# Postive dataset1
p1 = pickle.load(infile)
filename = 'models/modelp2'
infile = open(filename,'rb')_# Postive dataset2
p2 = pickle.load(infile)
filename = 'models/modelp3'
infile = open(filename,'rb')_# Postive dataset3
p3 = pickle.load(infile)
filename = 'models/modelp4'
infile = open(filename,'rb')_# Postive dataset4
p4 = pickle.load(infile)
filename = 'models/modeln2'
infile = open(filename,'rb')_# negative dataset1
n1 = pickle.load(infile)
```

Figure 18 - Code snippet for loading datasets

The entire dataset is converted into pandas data frame to perform input-output operations seamlessly. In addition to that, the pandas library also offers a variety of data manipulation utilities. As seen in Figure 19, an additional column ‘human’ is added to the datasets along with the previous 14 extracted features. The positive sample dataset is manipulated manually to contain ‘1’ in the human column, whereas the negative sample dataset contains ‘0’ for the same column.

```
dfp5['human'] = 1 #setting the category '1' as human
dfn1['human'] = 0 #setting the category '0' as human



data = pd.concat([dfn1, dfp5])

#Setting the featueres. We choose use all the 14 columns as features
X=data[['feature 1', 'feature 2', 'feature 3', 'feature 4', 'feature 5', 'feature 6','feature 7', 'feature 8', 'feature 9',
        'feature 10','feature 11','feature 12','feature 13','feature 14']]  # Features



y=data['human']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=66)

rfc = RandomForestClassifier(n_estimators=20, random_state=0)
rfc.fit(X_train,y_train)
pickle.dump(rfc, outfile)
rfc_predict = rfc.predict(X_test)

print(confusion_matrix(y_test,rfc_predict))
print(classification_report(y_test,rfc_predict))
print(accuracy_score(y_test, rfc_predict))

print(pd.DataFrame({'feature': list(X),'importance': rfc.feature_importances_}).\
                    sort_values('importance', ascending = False))
```

Figure 19- Code snippet for pandas data manipulation.

The technique where the model learns the parameters of the prediction function and tries to predict it on the same test samples is very common. This will result in a perfect score and might perform poorly on real-world data, and this situation is known as over-fitting. To avoid this, a popular supervised machine learning practice is carried out where a part of available sample data is held and tested. Figure 20 shows the flowchart where the available sample

data is split into test and train partitions randomly using the scikit functionality. The value set in the random_state is the seed for a random split. The model learns the parameters from the train data and tries to predict the test data. In this project, we set the test size to be 0.3, meaning 30 percent of the available samples is test data and the remaining 70 percent in training data.

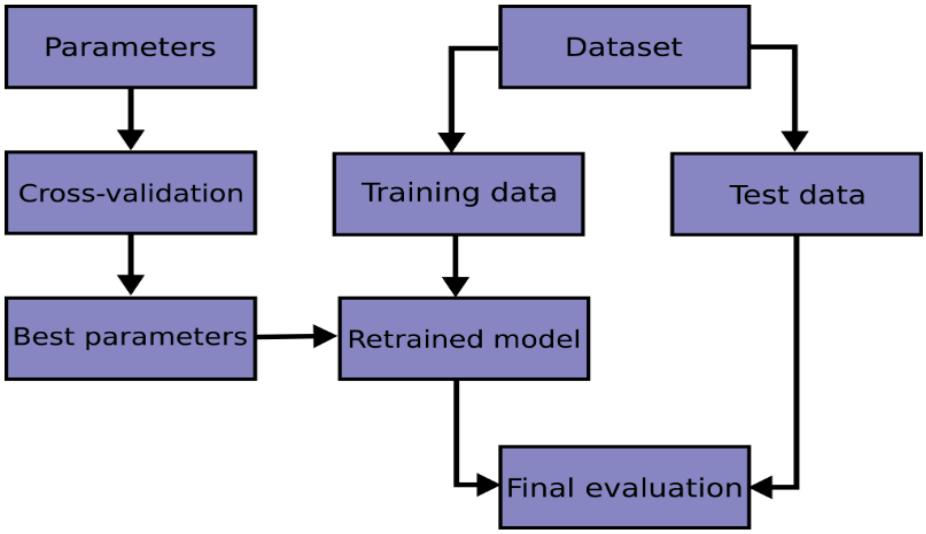

Figure 20 - Evaluator flowchart

The following are the basic steps involved in the random forest prediction algorithm.

1. Choose N random records from the pandas data frame.
2. Construct a decision tree based on N samples.
3. Decide the number of trees in the prediction algorithm and repeat the above steps.

4. In case of a prediction for new input, each tree in the forest predicts a category. The new input record will be assigned with the category that is voted the most by the trees.

As the random forest classifier is an ensemble model comprising a number of decision trees, the number of estimators is set to 20 which represents the number of trees created in the classifier model in the code. The max_features is auto by default which decides the number of features that are considered on a per-level split, rather than on the entire decision tree construction. Hence, this is not used in the code. The trained model is also pickled as an output in order to be used in the main application code.

4.3.4.3 PREDICTING THE CLUSTER

Figure 21 shows the code snippet from the main application program, where each scan from the sensor is clustered using the DBSCAN algorithm. The clustering algorithm returns all the clusters present in the scan as a list. Each cluster is then fed into the supervised, trained model to predict if the tested cluster is human or not. As defined earlier, the model should return a value 1 for a human cluster and a value 0 for a non-human cluster.

```
    # Compute DBSCAN
    db = DBSCAN(eps=100, min_samples=15).fit(centers)
    core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
    core_samples_mask[db.core_sample_indices_] = True
    labels = db.labels_

    # Number of clusters in labels, ignoring noise if present.
    n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
    n_noise_ = list(labels).count(-1)
    label_list = set(labels)

    def getFeatures(clust_x, clust_y, clustersize):...

    for label in label_list:
        index = labels == label
        cluster = scan[index]

        clus_max_x = max(cluster[:, 0]) + 200
        clus_min_x = min(cluster[:, 0]) - 200
        clus_max_y = max(cluster[:, 1]) + 200
        clus_min_y = min(cluster[:, 1]) - 200
```

```
    features = getFeatures(cluster[:, 0], cluster[:, 1], cluster.shape[0])

    c1 = self.plotItem.plot(cluster[:, 0], cluster[:, 1], symbol='o', symbolPen=colors[label], symbolSize=8)
    txt = '\U0001f600' + '-' + str(humans)
    tx1 = 'total humans detected =' + str(humans)
    # text = pg.TextItem(html=txt, anchor=(0, 0), border='w', fill=(0, 0, 255, 100))

    if (rfc.predict([features])==1) and features[0]>25 and features[4]< 700000:



        humans += 1
        txt = '\U0001f600' + '=' + str(humans)

        text = pg.TextItem(html=txt, anchor=(0, 0), border='w', fill=None)
        print(features)


        clus_max_x = max(cluster[:, 0])
        clus_min_x = min(cluster[:, 0])
        clus_max_y = max(cluster[:, 1])
        clus_min_y = min(cluster[:, 1])
        x1 = [clus_min_x ,clus_min_x, clus_max_x, clus_max_x, clus_min_x]
        y1 = [clus_min_y ,clus_max_y, clus_max_y, clus_min_y, clus_min_y]
        self.plotItem.plot(x1, y1, pen='r')
        self.plotItem.addItem(text)
        text.setPos(clus_max_x, clus_max_y)
```

Figure 21- Code snippet for cluster prediction.

Initially, the Kalman filter tracking algorithm was considered for object tracking in this project as it focuses on the prediction of object's future location, reduction of noise, and association of multiple objects to their tracks. Kalman filter is a dynamic measurement model defined for the

tracking systems, assuming both velocity and position measurements [25]. One of the fundamental problems of conventional Kalman tracking is that the algorithm considers only the position-based measurements and is, therefore, incapable of making use of sensors that measures velocity like the ultrawideband sensors [25]. However, the primary objective of the project, which is to count and track the humans present in the environment is implemented without the Kalman filter because the measurements or the positions are already known. Since the exact coordinates of the cluster points are known, the minimum and maximum co-ordinates are identified in each cluster to draw a bounding box around the contour only when the detected cluster is a human. This is represented in figure 22. In order to eliminate the noise and improve the observations, we manually fine-tune the conditional parameter for feature 0 and feature 4 as showing in the code snippet of figure 23.

```
    for label in label_list:
        index = labels == label
        cluster = scan[index]

        clus_max_x = max(cluster[:, 0]) + 200
        clus_min_x = min(cluster[:, 0]) - 200
        clus_max_y = max(cluster[:, 1]) + 200
        clus_min_y = min(cluster[:, 1]) - 200
```

Figure 22- Code snippet for the bounded box.

“\U0001f600” in the code snippet shown in figure 23 is the Unicode block for emoticons. This emoticon is added as a text to the bounded box around the detected human clusters along with the count. Each human detected in the frame is given a unique identifier number starting from the range of 1. The first detected human has a tag of 1 along with an emoticon symbol, and the tag id increases with the number of humans detected. There is no upper limit defined

on the number of humans detected. Also, the total count balances accordingly as and when the human enters or exits the environment. This is implemented in the snippet shown in Figure 23.

```
if (rfc.predict([features])==1) and features[0]>25 and features[4]< 700000:
    humans += 1
    txt = '\U0001f600' + '=' + str(humans)

    text = pg.TextItem(html=txt, anchor=(0, 0), border='w', fill=None)
```

Figure 23 - Code snippet to count humans.

4.4 LIMITATIONS AND MILESTONES

Although, the project achieves the desired objectives it still suffers from the following limitations.

4.4.1 GROUND PLANE ASSUMPTION

The application requires the sensor to be functioning on a planar area, or on the same level as the ground plane. Hence, a downwards angle or mounted on a slope would degrade the performance of the application.

4.4.2 LASER SCANNING HEIGHT

The application requires the sensor to be mounted at the hip level. If the sensor is mounted above or below, the system may not able differentiate human and non-human samples and

therefore be unable to detect and display the accurate count. This could be improved by collecting samples with different mounted heights of the sensor.

4.4.3 LACK OF POSITIVE SAMPLES

Another major challenge in collecting training data is the number of positive samples. While collection of negative samples is relatively easy, it requires more time to collect positive samples as human clusters per scan is very minimal in number when compare to the non-human clusters. This could be improved by repeating the collection procedure in multiple public crowded places like the library.

4.4.4 OCCLUSION OF MULTIPLE BODIES

The drawback of the current application is that it can fail to detect or count people due to occlusions when human bodies come in line of sight with one behind the another. This could be avoided by using a complementary sensor and fusion the data which may increase the cost of the setup. Another possible approach is to introduce object tracks through software and keep track of the detected objects.

5. RESULTS AND ANALYSIS

This project has developed a conceptual model to detect and count humans with respect to the kitchen scenario. The following tests were carried out for the same.

5.1 QUANTIFYING THE QUALITY OF THE PREDICTIONS

The figure 24 visualises the importance of features derived from the training samples. The Table 1 identifies the features with its name and importance score. It is observed from the scores that all features contribute to the model construction. However, cluster size, standard difference from mean and average mean difference have more significant importance when compared to rest of the attributes.

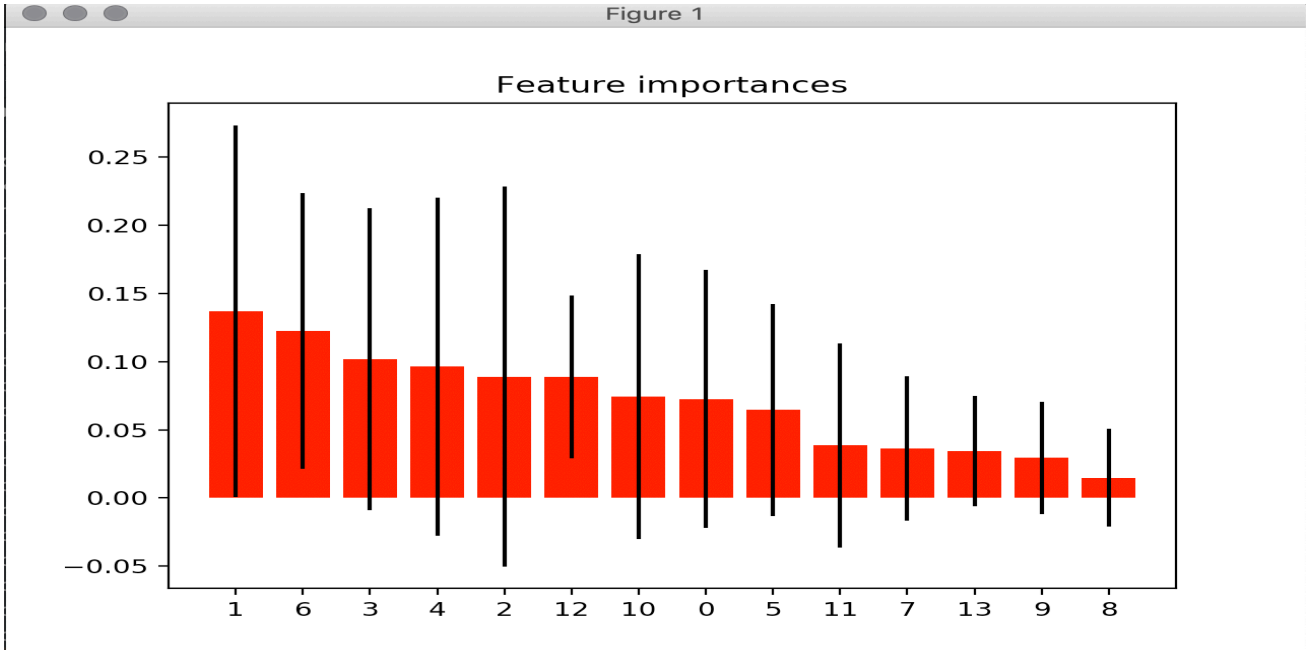

Figure 24 - Feature Importance

FEATURE NUMBER	FEATURE	IMPORTANCE
1	CLUSTER SIZE	0.136895
6	STANDARD DIFFERENCE FROM MEAN	0.122566
3	AVERAGE MEAN DIFFERENCE	0.101642
4	WIDTH	0.096307
2	LINEARITY	0.088965
12	CIRCULARITY	0.088703
10	RADIUS	0.074311
0	BOUNDARY REGULARITY	0.072566
5	MEAN CURVATURE	0.064613
11	ANGUALAR DIFFERENCE	0.038548
7	INSRIBED ANGLULAR VARIANCE	0.036325
13	STANDARD INSRIBLED ANGULAR VAIRANCE	0.034369
9	DISTANCE	0.029407
8	DISTANCE/CLUSTER SIZE	0.014784

Table 1 – Cluster Importance

5.2 PERFORMANCE OF THE MODEL

In the case of prediction of binary class problem to classify the object as human or non-human, there are two common types of errors that might occur.

1. False Positive - Predict an object as human when it is non-human.
2. False Negative. Predict an object as non-human when it is human.

In this project, it is more important to have low false positives as classifying non-human objects as human may produce inaccurate human count with non-human objects being stationary in the environment throughout the experiment. The classification metrics of the model is shown in Table 2.

CATEGORY	PRECISION	RECALL	F1-SCORE	SUPPORT
NON-HUMAN 0	0.81	0.86	0.84	8542
HUMAN 1	0.64	0.54	0.58	9227

Table 2 - Classification report

Precision: It is the ratio of correctly predicted observations to the total predicted positive observations.

Recall: It is the ration of correctly predicted observations to all the observations in the actual class.

F1 Score: It is the weighted average of precision and recall. It takes both the false negative and false positives into account.

```
=== Mean AUC Score ===
Mean AUC Score - Random Forest:  0.7808062678062678
```

Figure 25 - AUC score

AUC provides an aggregate measure of performance across all possible classification thresholds. One way of interpreting AUC is as the probability that the model ranks a random positive example more highly than a random negative example. The model developed in this project has a score of 0.78 as shown in Figure 25.

5.3 EXPERIMENT RESULTS

The below figure 26 shows the output of the experiment results in Kitchen area with two human bodies walking around for a span of ten minutes and the application processes the dataset collected during the experiment. Figure 27 shows the terminal results of the same dataset processed by the application. The experiment consists of 11097 scans of two human bodies resulting in 22194 clusters of human objects. The application is able to detect 17367 clusters out of 22194 total human objects.

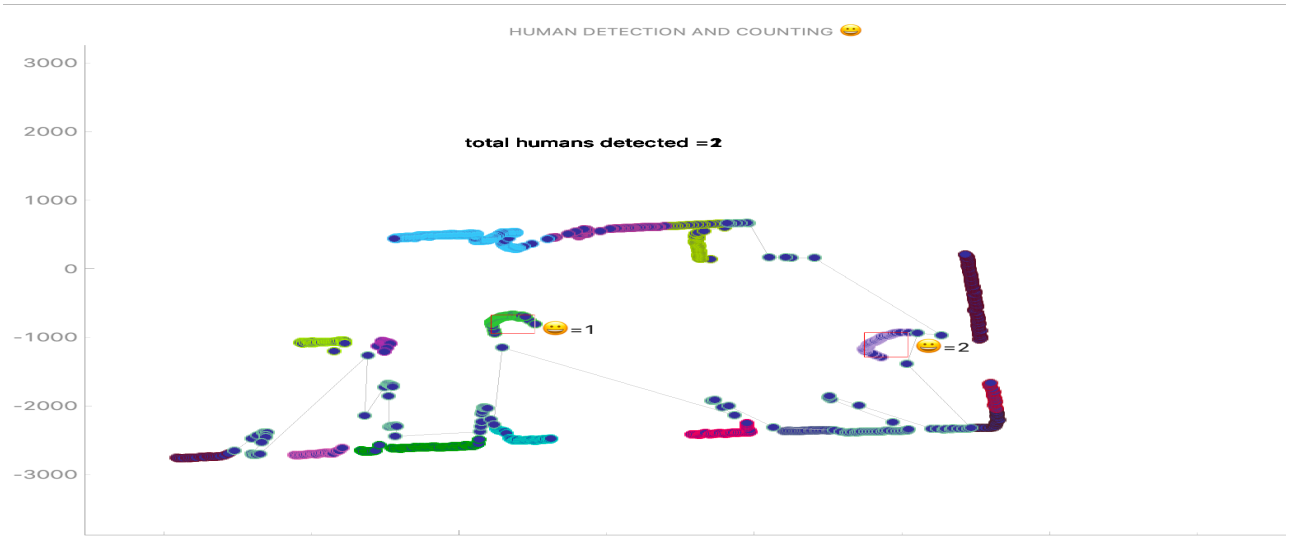

Figure 26-Screenshot of application

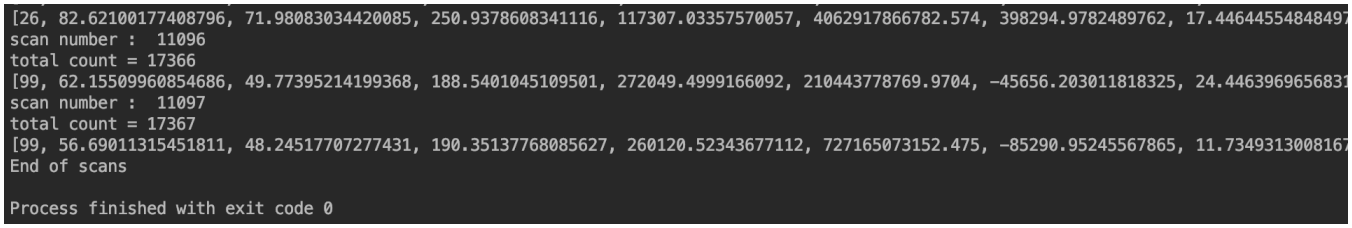

Figure 27- Experiment results

$$Application\ Accuracy = \frac{Total\ Human\ Clusters\ detected\ by\ the\ model}{Total\ human\ clusters\ present\ in\ the\ observation}$$

Equation 2

The classifier model scored a 72.5 percent precision in detection and calculation of humans for the all the samples collected. Also, as per the equation 2, the application is able to achieve a functional accuracy of 78.25 percent in detecting humans during the total time of the experiment.

6. CONCLUSION AND FUTURE SCOPE

The previous section showed the metrics and results of the project, and thus indicate a proof of concept that can be extended to other applications. The project has succeeded in achieving the objectives of creating a supervised machine learning model and an application to detect and count human beings. However, the major drawback was the problem of occlusion when multiple objects come in line of sight. This could be addressed in future by using special tracking algorithms such as Kalman filters or complementary sensors. The project implementation was done completely in python using the passive data collected from the sensor. This implementation can further be extended to process data and train the model in real-time. It is observed that the accuracy of the model improved by collecting more positive samples at different mounted heights and locations. False negatives and false positives were also reduced significantly by training the model in a well-known predefined area such as the kitchen setup and by tuning the feature parameters.

This project suggests a great potential for applications such as clinical locomotion analysis, activity classification, motion pattern analysis of customers and smart kitchen design solutions. The future work can also investigate low powered LIDARS combined with microcontrollers that takes advantage of machine learning models hosted in the cloud.

7. REFERENCES

[1] Hokuyo. (2015). UST-30LX: Specification. Retrieved from https://www.hokuyo-aut.jp/dl/UST-30LX_Specification.pdf

[2] Ma, Z., Bigham, J., Poslad, S., Wu, B., Zhang, X. and Bodanese, E. (2018). Device-Free, Activity During Daily Life, Recognition Using a Low-Cost Lidar. 2018 IEEE Global Communications Conference (GLOBECOM).

[3] Krishnan, S., Sharma, P., Guoping, Z. and Woon, O. (2007). A UWB based Localization System for Indoor Robot Navigation. 2007 IEEE International Conference on Ultra-Wideband.

[4] Han, J., Minju Kang, Yonghoon Cho and Kim, J. (2015). Planar SLAM under a semi-submersible offshore platform with an unmanned surface vehicle. 2015 12th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI).

[5] Ji, K., Chen, H., Di, H., Gong, J., Xiong, G., Qi, J. and Yi, T. (2018). CPFG-SLAM:a Robust Simultaneous Localization and Mapping based on LIDAR in Off-Road Environment. 2018 IEEE Intelligent Vehicles Symposium (IV).

[6] Chen, X., Zhang, H., Lu, H., Xiao, J., Qiu, Q. and Li, Y. (2017). Robust SLAM system based on monocular vision and LiDAR for robotic urban search and rescue. 2017 IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR).

[7] Li, C. and Fang, S. (2016). Device-free human detection using WiFi signals. 2016 IEEE 5th Global Conference on Consumer Electronics.

[8] David V Lu and William D smart. Towards more efficient navigation for robots and humans. In International Conference on Intelligent Robots and Systems (IROS), 2013

[9] A. Fod, A. Howard and M. J. Mataric, "Laser-based people tracker" IEEE International Conference on Robotics and Automation, Washington DC, USA, May 2002.

[10] Angus Leigh. Person tracking and following using 2D Laser Scanners. [Thesis]. Montreal: McGill University of London; 2015. 46p.

[11] J. Cui, H. Zha, H. Zhao and R. Shibasaki, "Laser-based detection and tracking of multiple people in crowds", Computer Vision and Image Understanding, Vol 106, May 2007, pp 300-312.

[12] Hokuyo UST-30LX Scanning Laser Rangefinder. Hokuyo: Osaka, Japan, 2016. Available online: https: //www.hokuyo-aut.jp/search/single.php?serial=167 (accessed on 23 June 2019).

[13] X. Shao, H. Zhao, K. Nakamura, K. Katabira, R. Shibasaki and Y. Nakagawa, "Detection and tracking of multiple pedestrians by using laser range scanners", IEEE/RSJ International Conference on Intelligent Robots and Systems, San Diego, CA, USA, 2007.

[14] Webcourses.ucf.edu. (2019). Python Lists vs. Numpy Arrays: IST Advanced Topics Primer. [online] Available at: https://webcourses.ucf.edu/courses/1249560/pages/python-lists-vs-numpy-arrays-what-is-the-difference [Accessed 13 Jul. 2019].

[15] Pyqtgraph.org. (2019). Introduction — pyqtgraph 0.10.0 documentation. [online] Available at: http://www.pyqtgraph.org/documentation/introduction.html#what-is-pyqtgraph [Accessed 11 Jul. 2019].

[16] Medium. (2019). Unsupervised Machine Learning: Clustering Analysis. [online] Available at: https://towardsdatascience.com/unsupervised-machine-learning-clustering-analysis-d40f2b34ae7e [Accessed 21 Jul. 2019].

[17] El-Bendary, N., Kim, T., Hassanien, A.E. et al. Computing (2014) 96: 381. https://doi.org/10.1007/s00607-013-0342-0

[18] L. Breiman, "Random forests", Machine Learning, vol. 45, no. 1, pp. 5-32.

[19] Wellhausen, L., Dube, R., Gawel, A., Siegwart, R. and Cadena, C. (2017). Reliable real-time change detection and mapping for 3D LiDARs. 2017 IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR).

[20] Luo, F., Poslad, S. and Bodanese, E. (2019). Kitchen Activity Detection for Healthcare using a Low-Power Radar-Enabled Sensor Network. ICC 2019 - 2019 IEEE International Conference on Communications (ICC).

[21] Zheng Gan, Zhong, L., Li, Y. and Guan, H. (2015). A random forest-based method for urban object classification using lidar data and aerial imagery. 2015 23rd International Conference on Geoinformatics.

[22] Breiman, L., Friedman, J., Olshen, R. and Stone, C. (2017). Classification and regression trees. Boca Raton, Fla.: CRC Press.

[23] GitHub. (2019). angusleigh/leg_tracker. [online] Available at: https://github.com/angusleigh/leg_tracker [Accessed 19 Jun. 2019].

[24] Docs.python.org. (2019). pickle — Python object serialization — Python 3.7.4 documentation. [online] Available at: https://docs.python.org/3/library/pickle.html [Accessed 21 Aug. 2019].

[25] Saho, K. (2018). Kalman Filter for Moving Object Tracking: Performance Analysis and Filter Design. Kalman Filters - Theory for Advanced Applications.

8. APPENDIX

All project related code and files is present in the supporting materials along with the report. The entire project is created in the virtual python environment which can be activated and quickly run the application files without requiring any installation of libraries. There are three python files created to execute the three problem phases separately. The DATAPREPARE.py processes the dataset in the CSV format collected from the sensor where it converts the dataset into 2D array and serializes the object. This program is used to convert all collected training and test datasets. The feature_export.py reduces the dimensionality of the previous files and extracts features required to be fed to the model. The model_creator.py program is used to build the supervised machine learning model from the pickle files created in the previous program. The final program detection.py uses the model file and test data object from the previous programs. The pickled machine learning model file and the test data file is required for the detection.py to detect and display the human count. The serialised files of test and training data are provided in the folder ‘pickled files’. All important parts of the code are well documented with the comments and showcased on Github (https://github.com/ec18473/lidarproj) for future development and understanding.