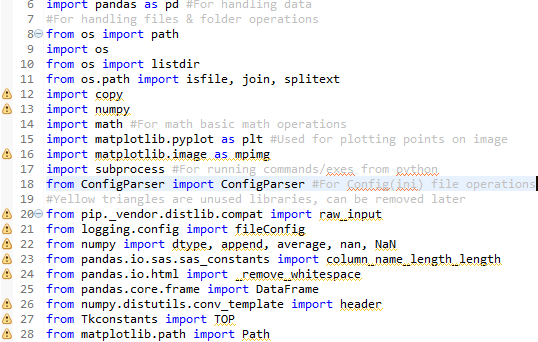
# Imports

There are several libraries used in this algorithm, the once that have a yellow triangle are not used and can be removed later.

The main libraries used are,

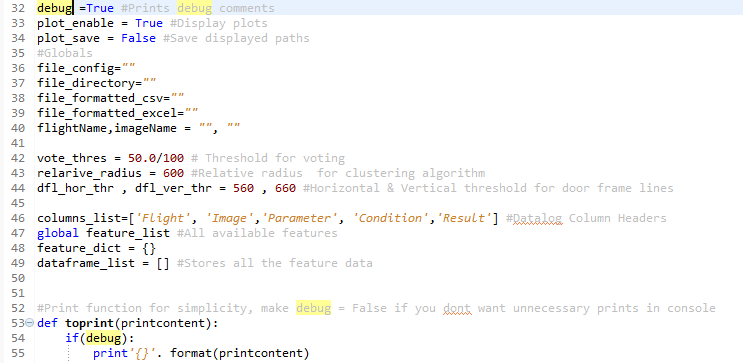
* pandas -  **for handling huge data**
* matplotlib – **for plotting**
* os -  **for file & folder operations**
* subprocess -  **for running commands/exes from python**
* math – **for basic math operations**



# Initializing Variables

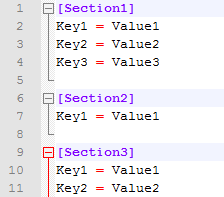
Initialize all the necessary global variables here.

**Dfl\_hor\_thr** & **dfl\_ver\_thr** will be updated in run time based on the data from the image. All the other parameters remain same throughout the program.



# INI File Operations (Config.ini File)

This is the generic format of the .ini file,



## Section

A single .ini file can have multiple sections like above. In this case this sample has 3 sections. **The section names must be unique**

## Key

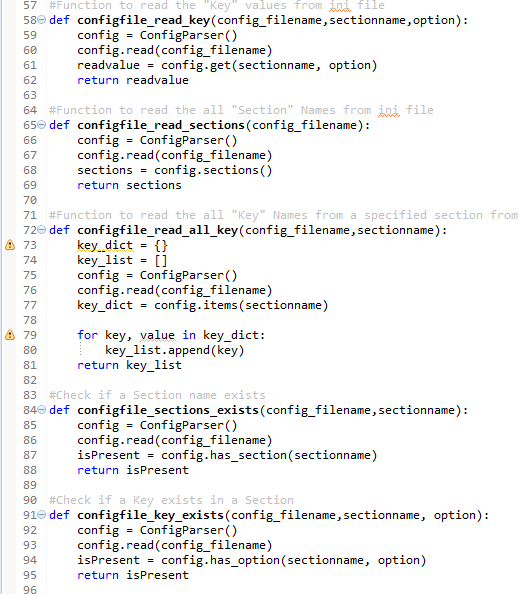
Every section can have multiple keys and the **key value of each section must be unique**

## Values

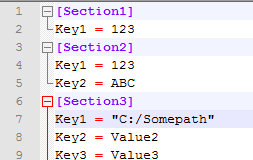
Each key is assigned with a value this can be any string or numbers or special characters

## configfile\_read\_key

This function reads the “Key’s Value” from a specific “Section” in the Config.ini file.

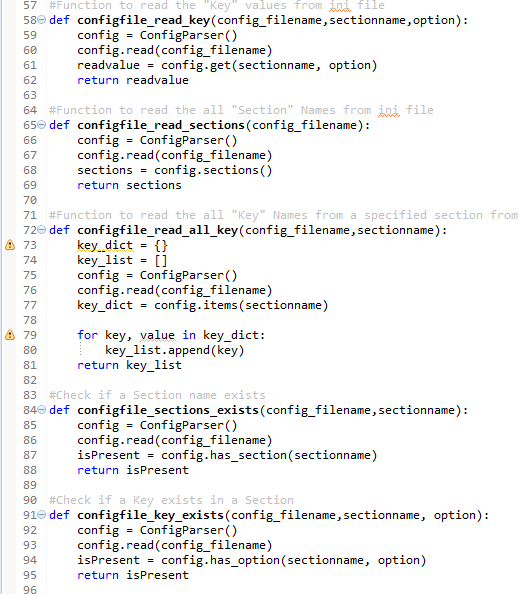


For example **configfile\_read\_key( “c:/Config/Config.ini” , “Section3” , “Key2”)** gives **“Value2”**

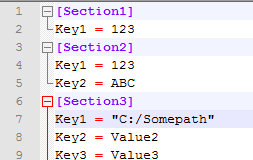


## configfile\_read\_sections

This function reads all the available “Sections” from the Config.ini file.

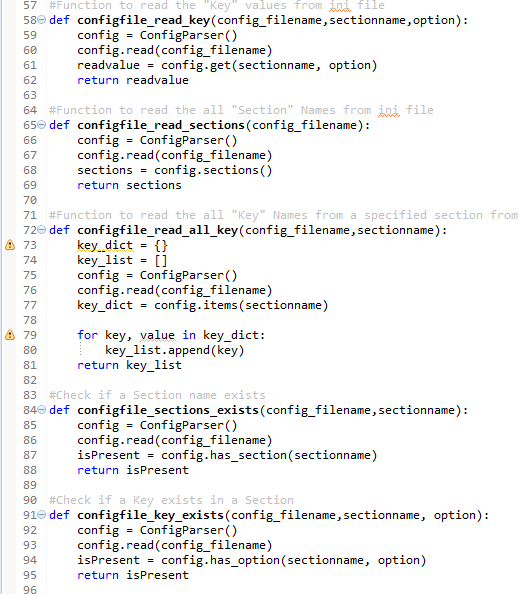


For example **configfile\_read\_sections( “c:/Config/Config.ini”)** gives **[“Section1”, “Section2”, “Section3”]**

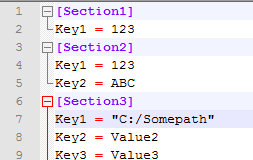


## configfile\_read\_all\_key

This function reads all the available “Keys” from a “Section” in the Config.ini file.

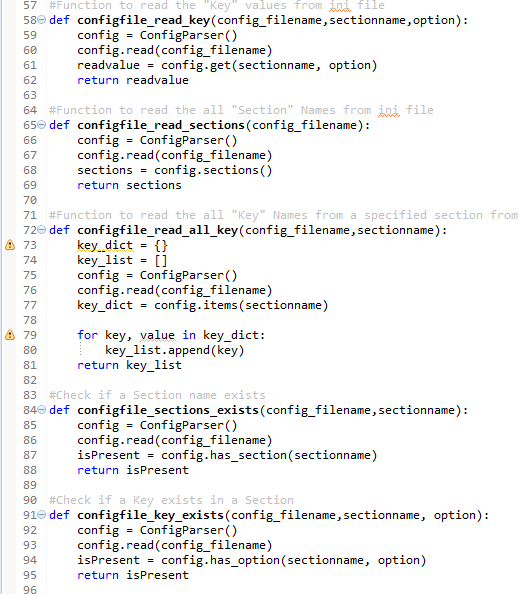


For example **configfile\_read\_all\_key( “c:/Config/Config.ini”,”Section2”)** gives **[“Key1”, “Key2”]**



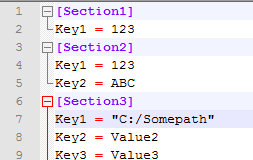
## configfile\_sections\_exists

This function checks if a specified “Section” in the Config.ini file.



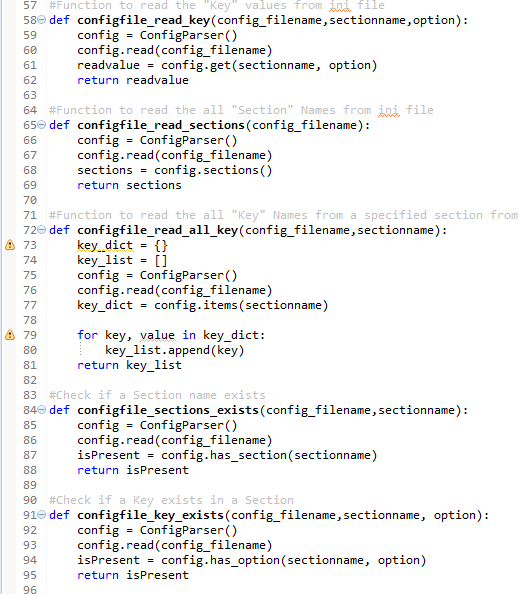
For example

**configfile\_sections\_exists( “c:/Config/Config.ini”,”Section2”)** gives **TRUE configfile\_sections\_exists( “c:/Config/Config.ini”,”Section6”)** gives **FALSE**



## configfile\_key\_exists

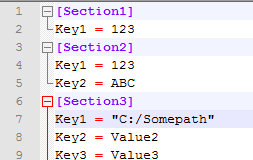
This function checks if a “Key” is available in a specified “Section” in the Config.ini file.



For example

**configfile\_key\_exists( “c:/Config/Config.ini”,”Section2”,”Key1”)** gives **TRUE**

**configfile\_key\_exists( “c:/Config/Config.ini”,”Section2”,”Key5”)** gives **FALSE**



# Run EXE

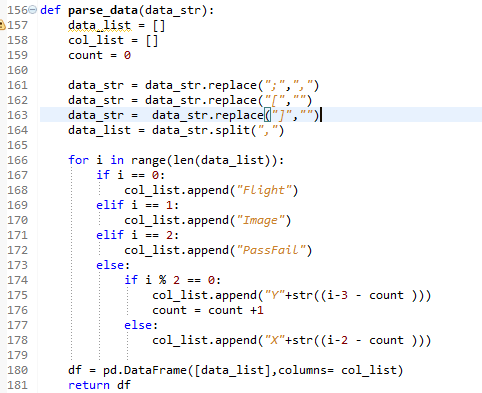
## Run\_command



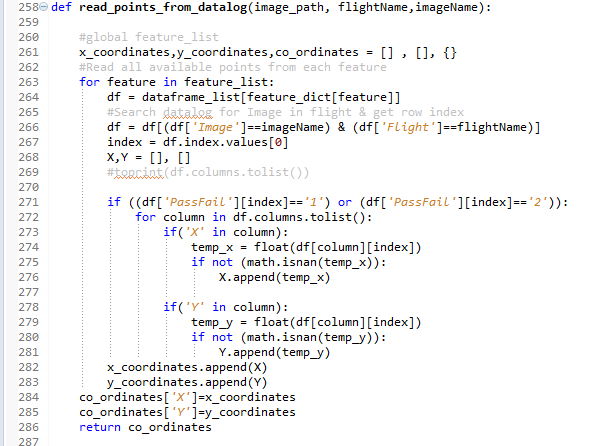
## Run\_exe



## parse\_data



## Read\_points\_from\_datalog

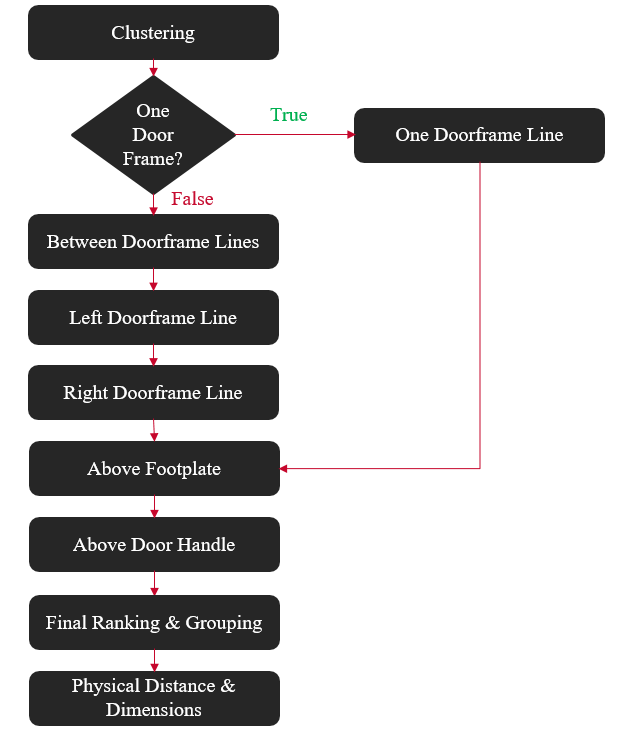


## Calculate\_doorframe\_parameters



## Plot Datapoints

# Feature Verification Algorithm



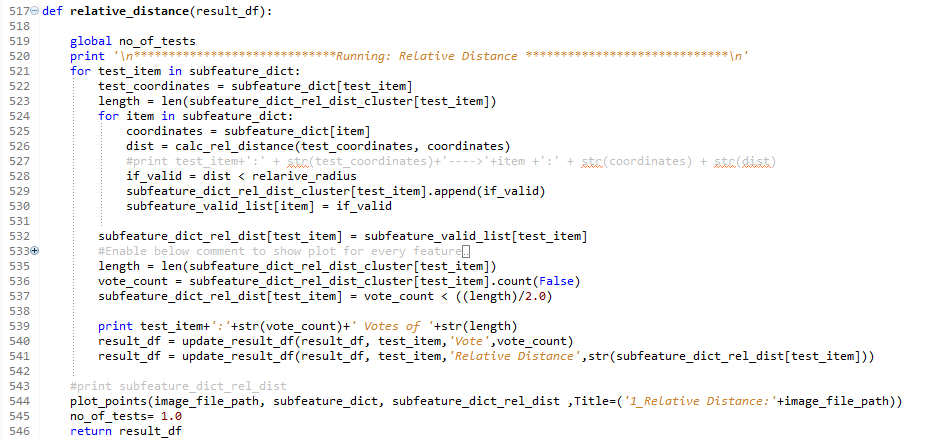
## 3.4.1 Relative Distance (Clustering)

Clustering is the process of grouping two or more objects based on the relative distance between them. The main objective of this step is to identify features in a similar group and in this process eliminating the outliers. The outliers are given an elimination vote if they fail the clustering check. Not that the actual elimination happens in the final stage, this step does not eliminate any outlier it just increases the vote for elimination.

Clustering/Relative Distance check calculates the relative distance between the center of every feature to the center every other feature detected for a specific image. If P1 & P2 are center points of two features located at coordinates (X1, Y1) & (X2, Y2) respectively the relative distance between them is given by the equation below,

Then, the distances are compared against a predetermined threshold distance value and the features that are beyond this threshold are flagged for elimination vote. The feature which receives the maximum flagging after the iterative check with all other features will be given an elimination vote.

### **Code**



### **Results:**

The input & output of Clustering check are show below,

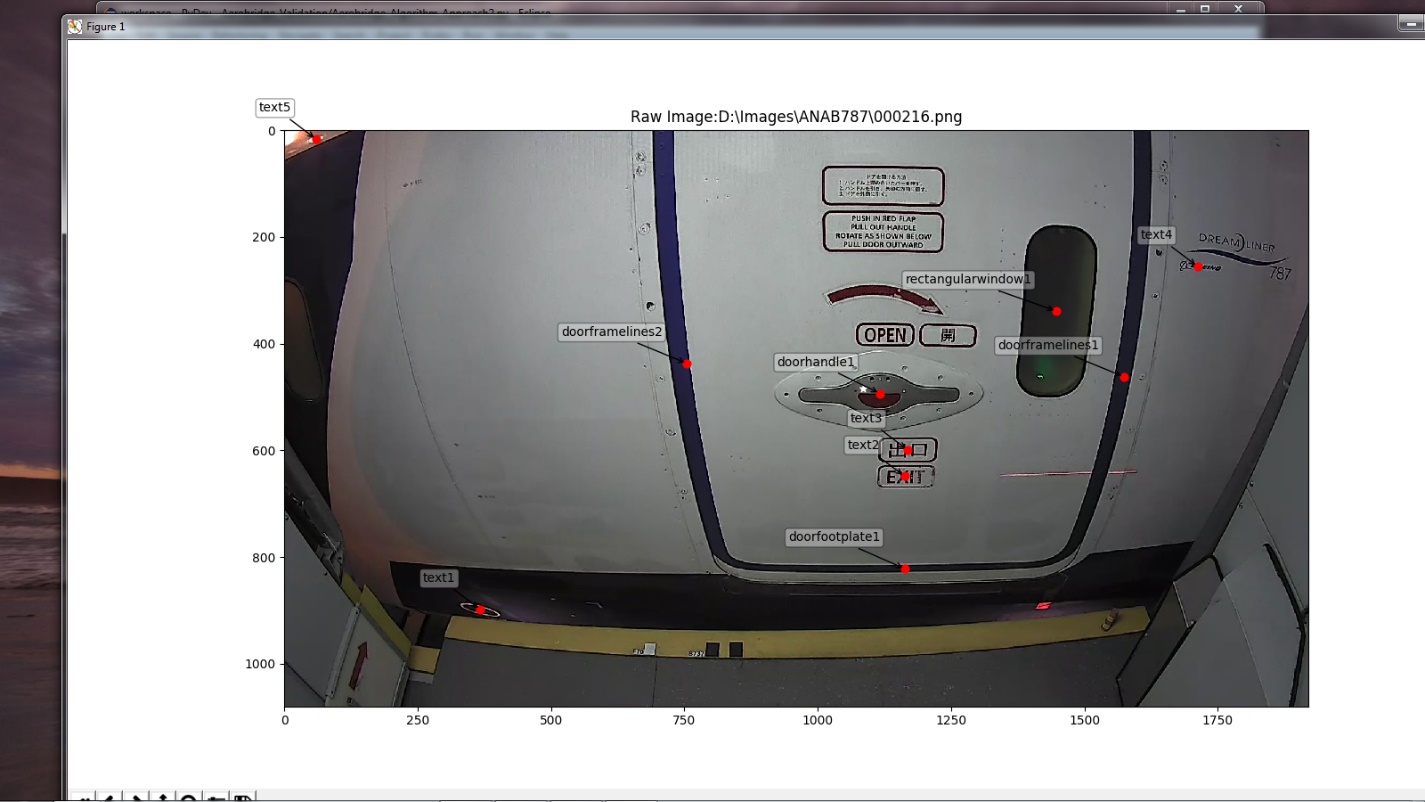


Figure 3.16 Raw Image

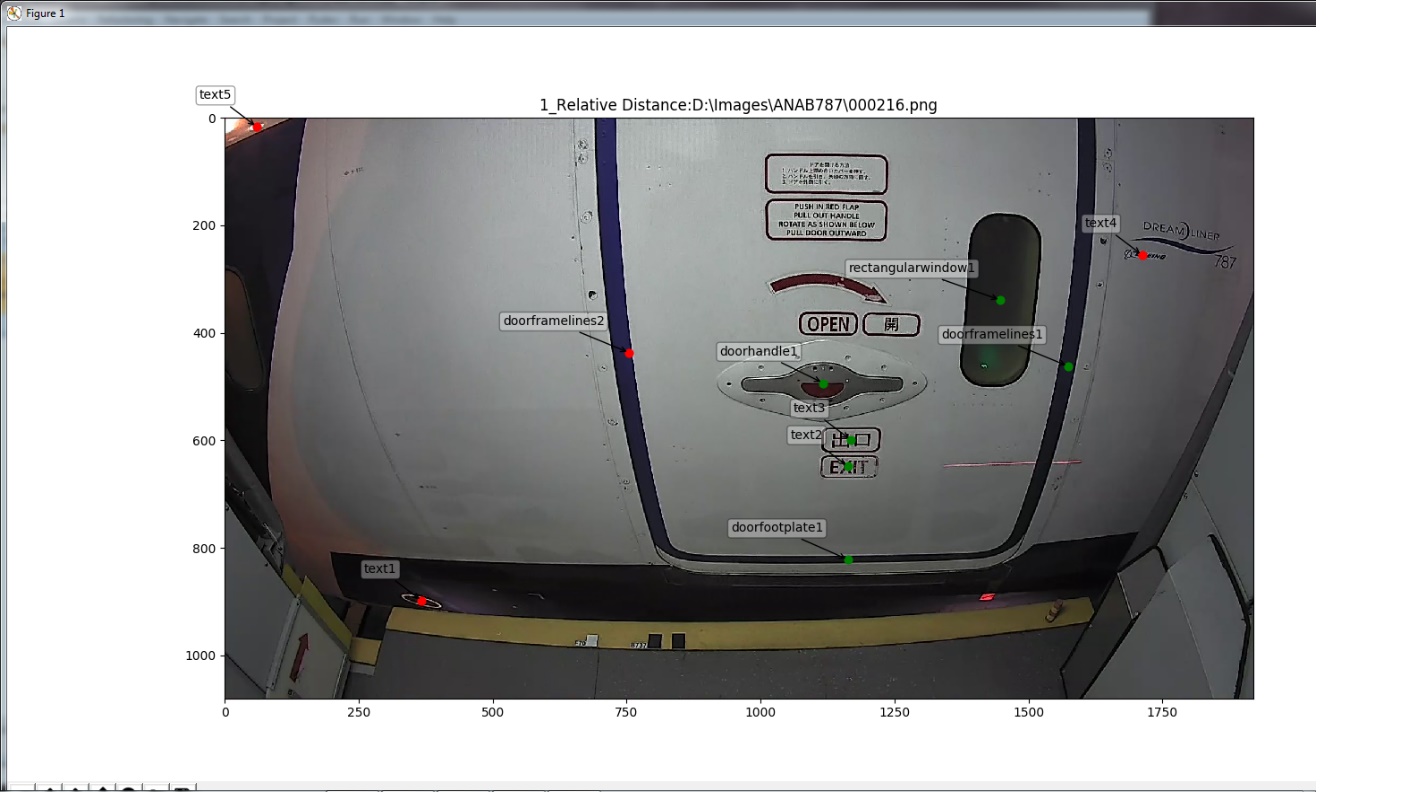


Figure 3.17 Clustering Check

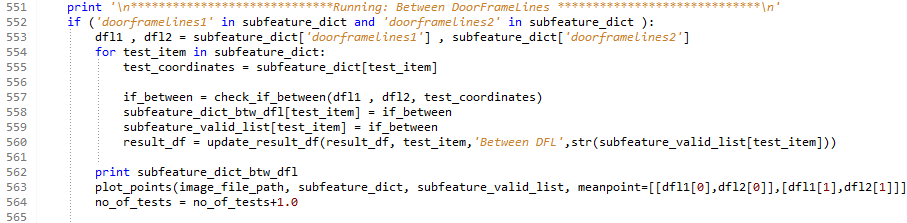
Note that the features text1, text4, text5 and doorframelines2 are very far from the cluster of features in the center and hence are flagged for voting by the Clustering check.

## 3.4.2 Between Doorframe Lines

This step as the heading describes, checks whether a feature is between door frame lines or not. It also checks if it is vertically within the height of the doorframe. i.e. If a feature is between door frame lines but far below or above the height of the frame, it must be voted for elimination. Those features that fail to satisfy the above criteria are given an elimination vote. This vote gets accumulated with the votes from the previous test(Clustering).

This check is done only if both the left and right door-frame lines are detected by the doorframe line Vision Algorithm.

### **Code:**



### **Results:**

The between door frame line check looks like the following,

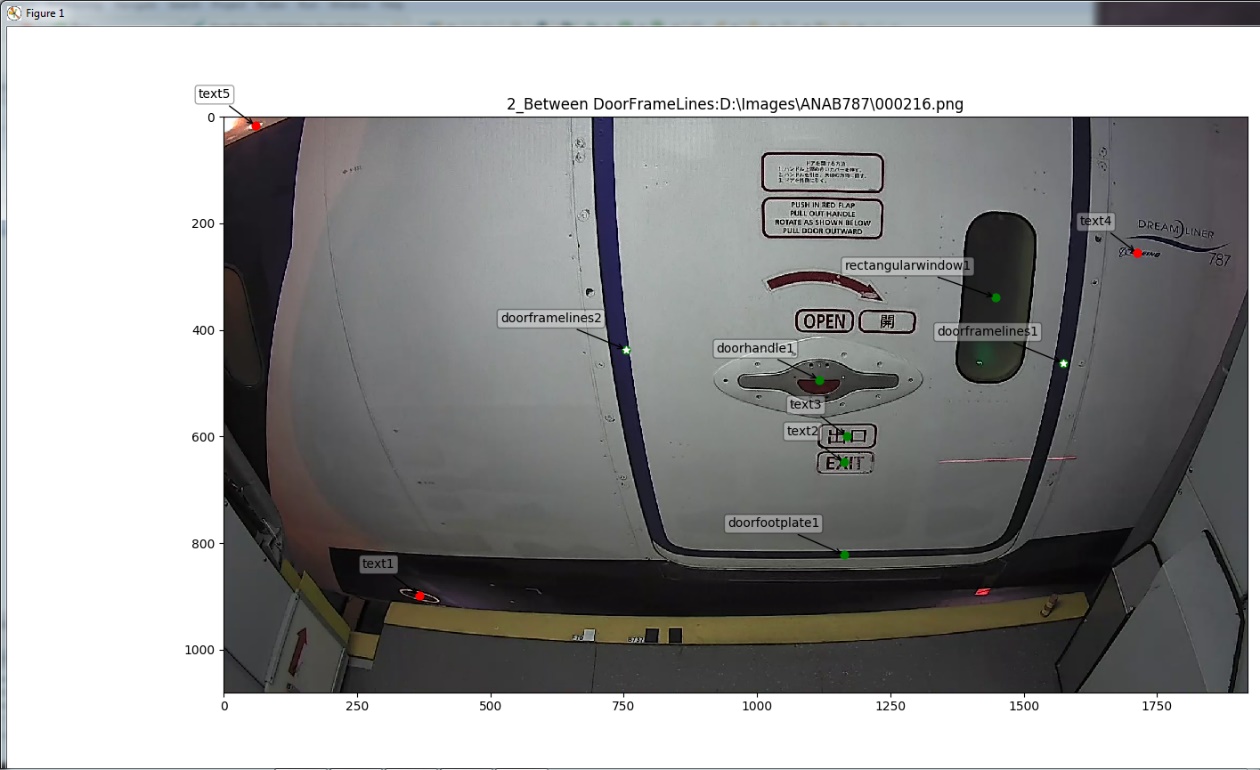


Figure 3.18 Between Doorframe Lines Check

From the image, it is evident that the features that are not between the door frame lines were voted for elimination.

## 3.4.3 Left & Right Doorframe Line

This check is done immediately after between door frame lines step. The left/right door frame step might look redundant because there is no necessity for this check if the features are between the frame lines. But if one of the door frame lines detected are wrong, a degree of error might be introduced in the verification process thus reducing the accuracy of the algorithm. By adding this check such anomalies are avoided.

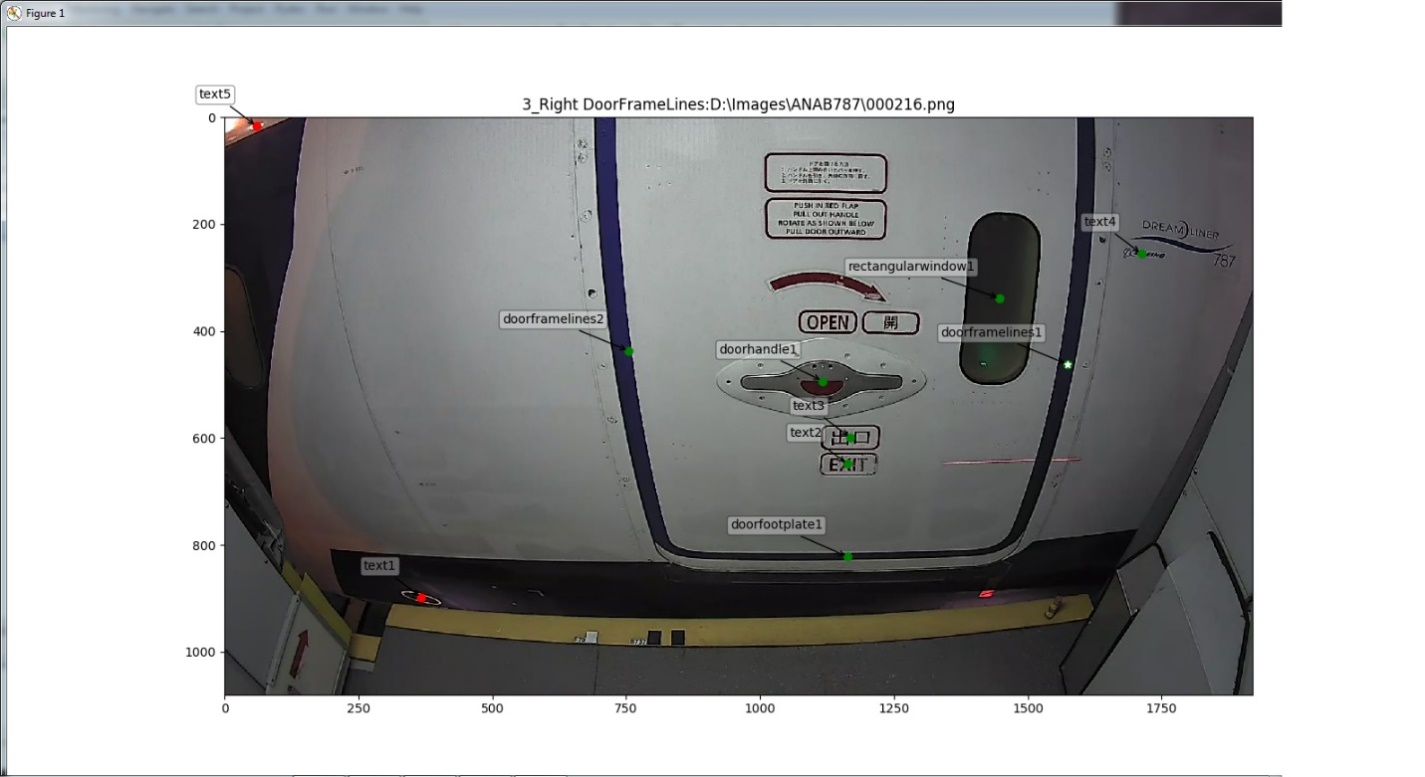
This step checks if the feature is around the left and right doorframe lines. Like previous step this also has a vertical and horizontal threshold.

### **Code:**

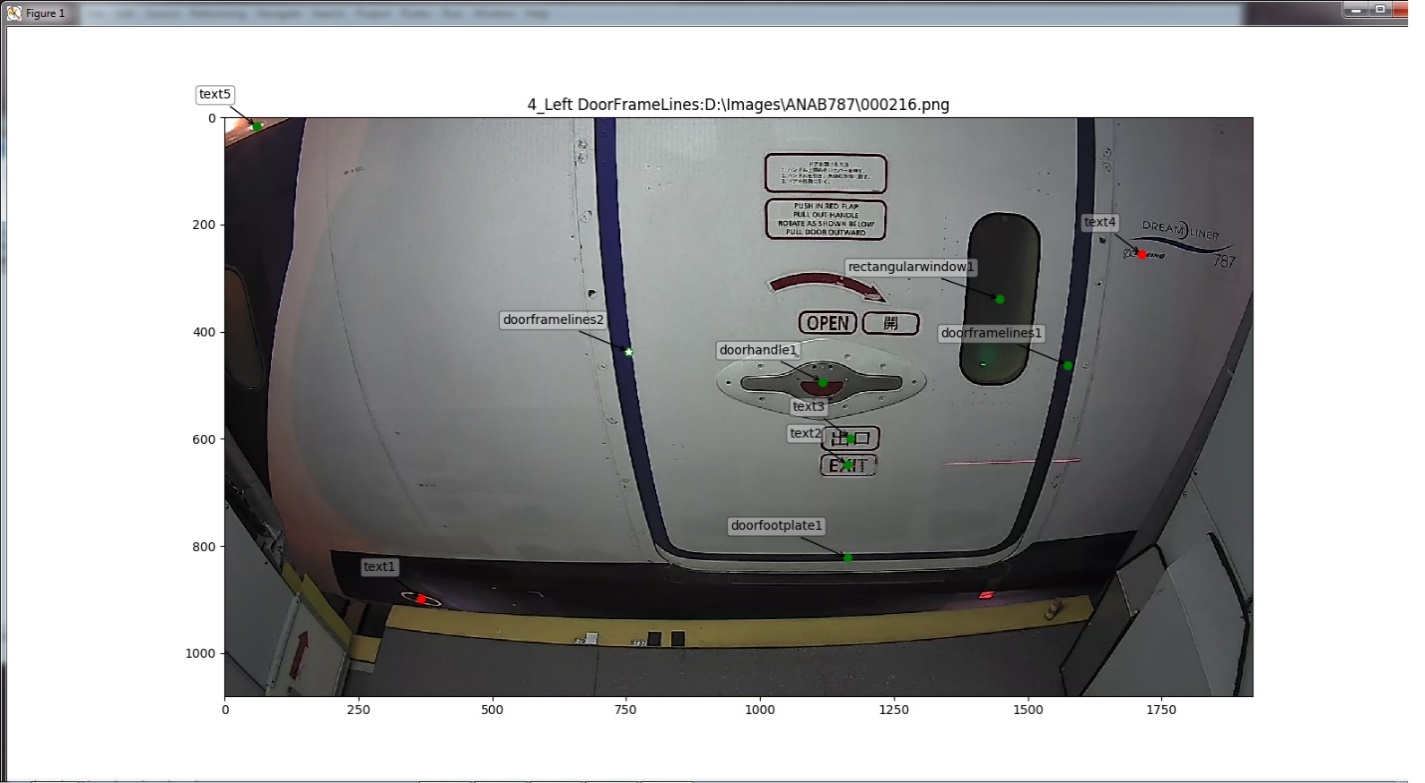


### **Results:**

The results obtained from these checks are shown below,



3.19 Right Door Frame Line Check



3.20 Left Door Frame Line Check

## 3.4.4 One Doorframe Line

This check is employed by the verification algorithm only if one door frame line is detected by the doorframe line vision algorithm. This is like the left/right doorframe line check; the only difference is its done when only one frame line is detected.

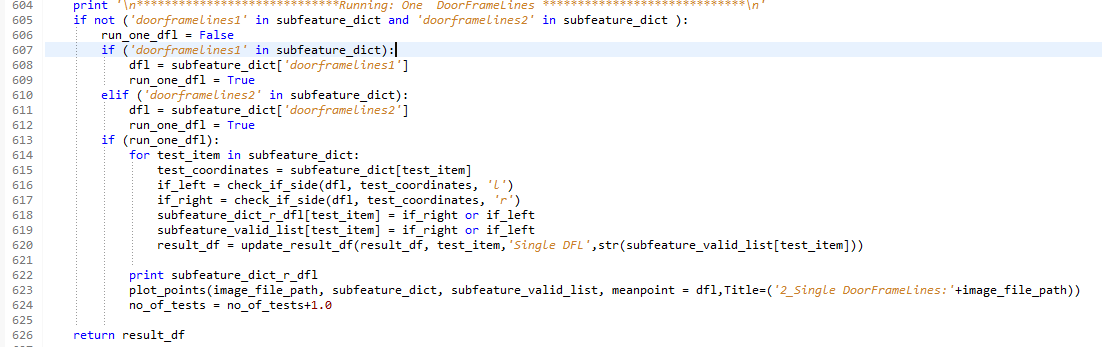
This stage checks if other features are within the height of the single door frame and at certain horizontal threshold away from the door frame line on both sides. The horizontal threshold is calculated by the following formula,

To calculate the frame width two door frame lines are needed. In this case the width is obtained from the previous time frame where both the frames are detected or if that data is not available then it is taken from a statistically computed average of door frame width. The statistical calculation involves obtaining the mean all the door frame width values stored in the SQLite database. The mean plus one standard deviation value is taken as the average of door frame width. The statistics and the histogram are shown below

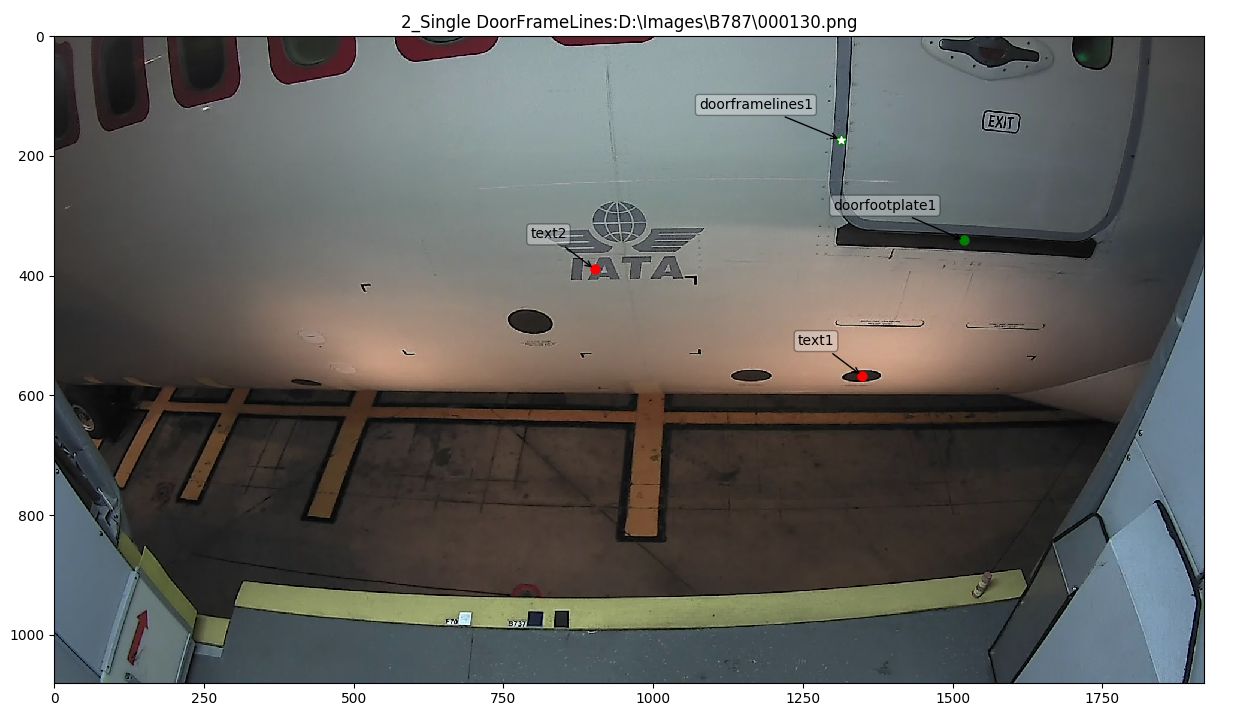
|  |  |  |  |
| --- | --- | --- | --- |
| **Mean** | **SD** | **Min** | **Max** |
| 717.4698 | 158.5301 | 400.4096 | 1034.53 |

Figure 3.21 Statistical Calculation of Average Door Frame width

### **Code:**



### **Results:**



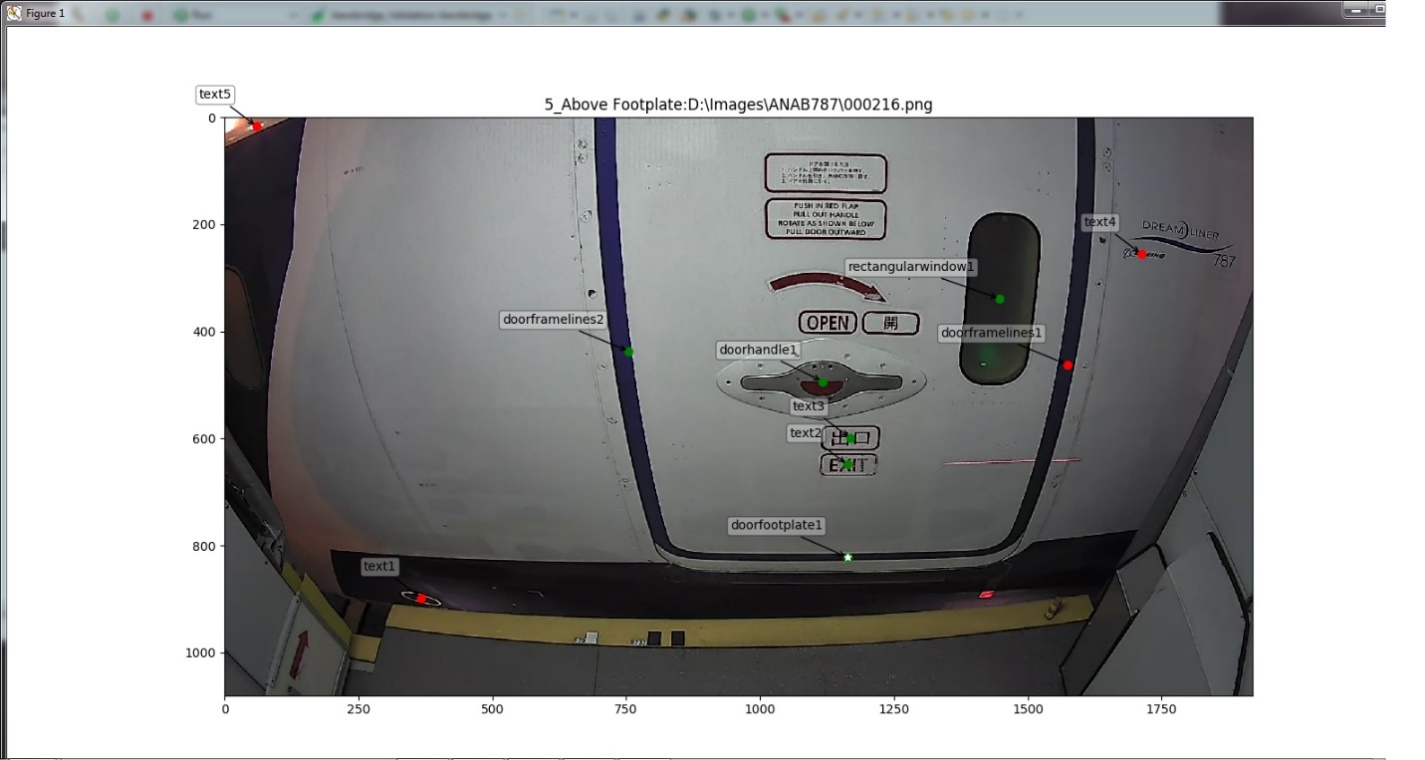
3.22 One Door Frame Line Check

## 3.4.5 Above Footplate

This section of the algorithm checks if a feature detected is above or below the footplate. This in turn also has a horizontal threshold. Any feature that fails this check adds an elimination vote to its name. At the end of the day, all the passed features will lie between door frame lines and above the foot plate. Unless the foot plate/door frame itself was detected wrong. The picture below illustrates this check,

### **Code:**

### **Results:**



3.23 Above Footplate Check

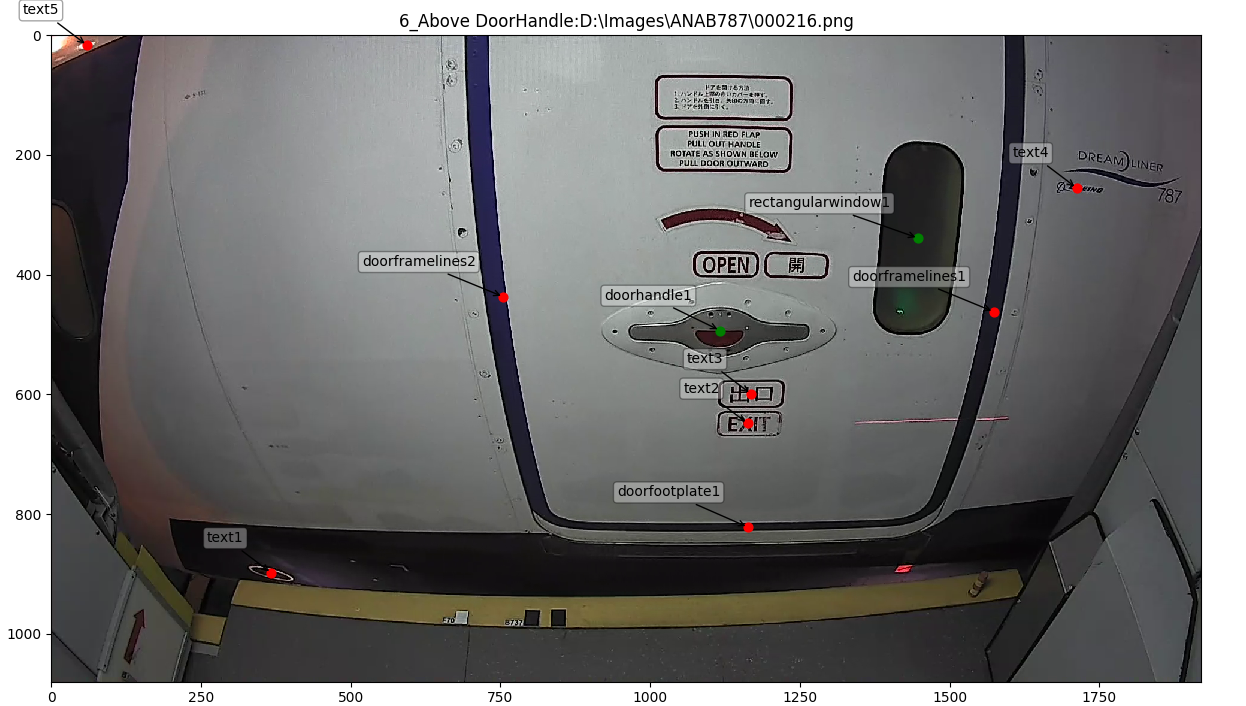
## 3.4.6 Above Door Handle

As the name suggests this step of the algorithm checks if a feature is above the door handle or not. But this test is only applied for door window and arrow, because these are the only features that are above the handle. Even though above foot plate step checks for something similar, this is just a surety check which adds extra layer of confidence to the verification algorithm

Show below is the output of this check,

### **Code:**

### **Results:**



3.24 Above Door Handle Check

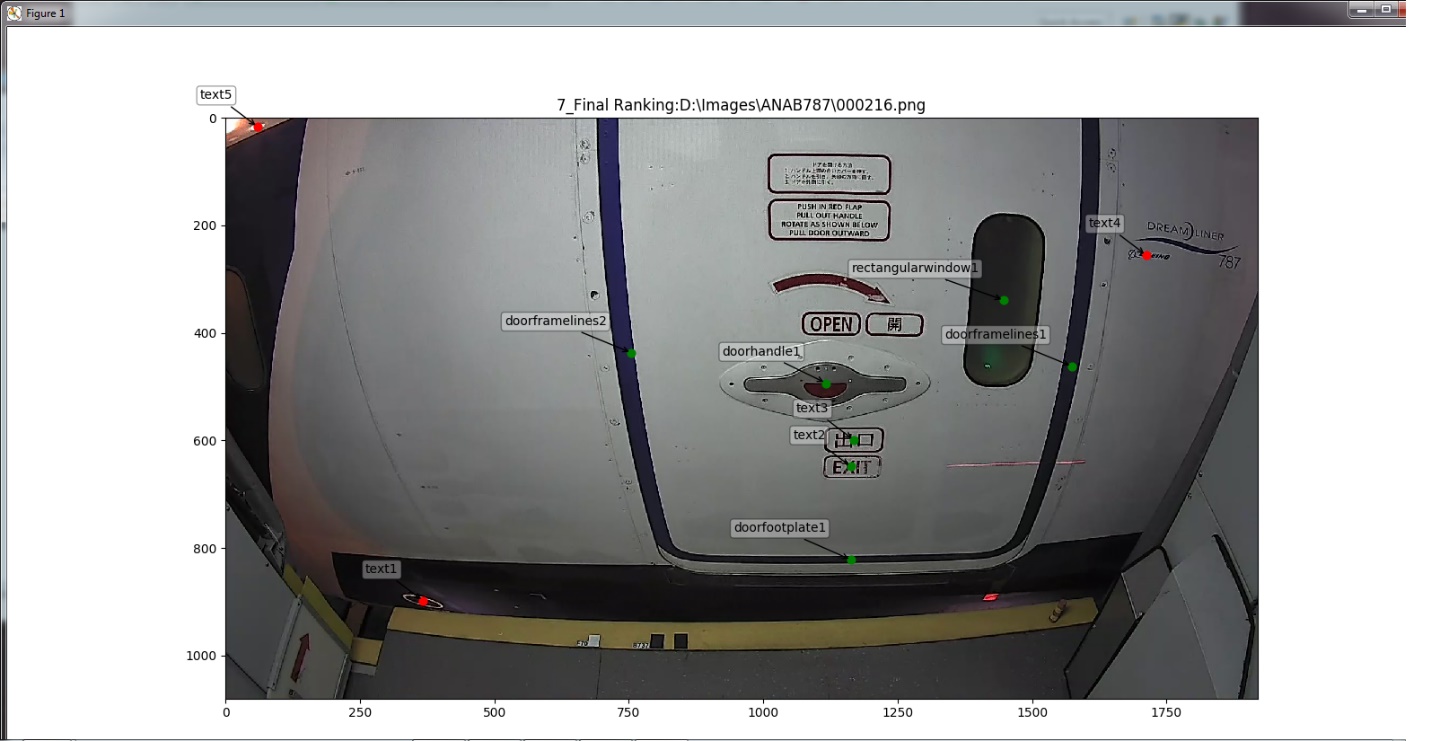
## 3.4.7 Final Voting & Grouping

This stage counts the votes for every feature and does the final elimination. Features whose votes are beyond a threshold are omitted. Currently threshold for elimination is 50%. i.e. The features that received 50% or more elimination votes are marked as outliers. This means they have failed for 3 or more of the above tests. The threshold value can later be tuned to optimize the algorithm further.

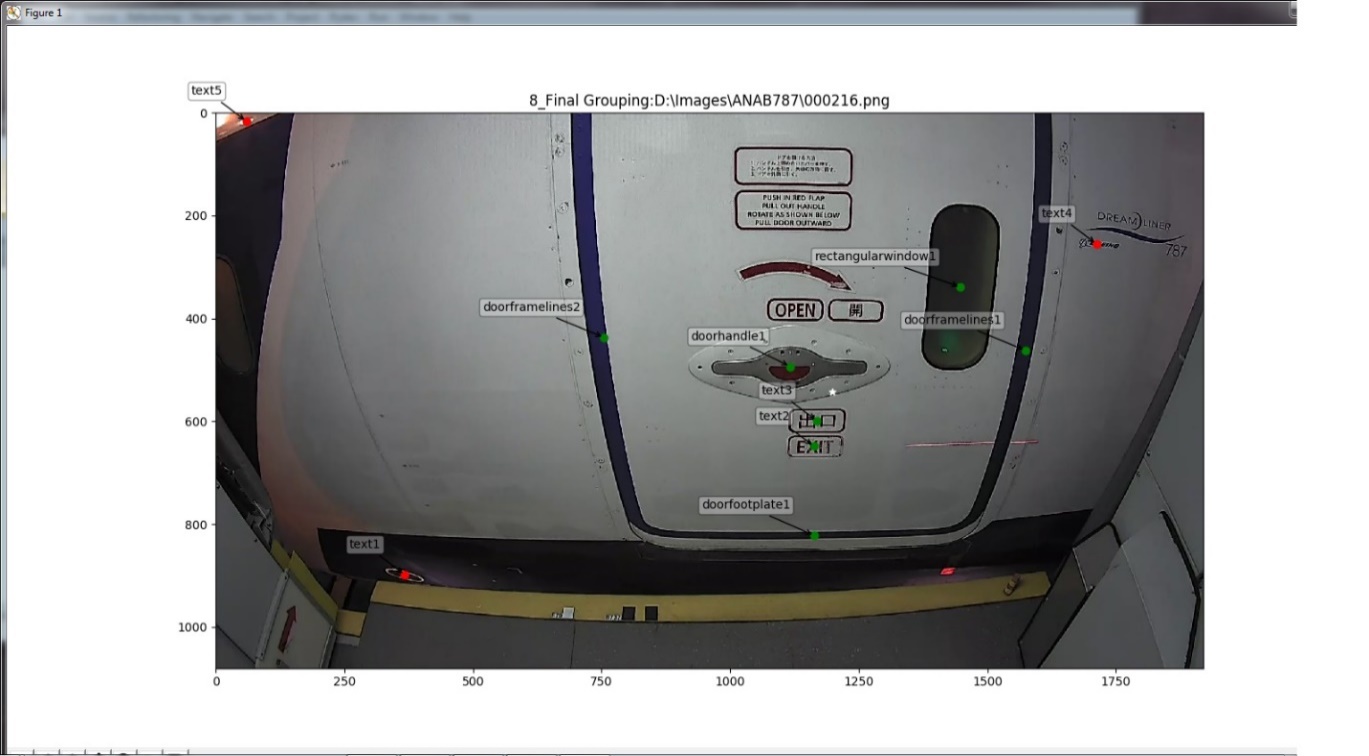
After the elimination process is complete, the midpoint of all the remaining features are calculated. And then the distance from the midpoint to all these features are calculated. If there exists any feature that is still far away from midpoint by the threshold, it is also eliminated. This step can be considered like a refined clustering. The output of the stage looks something like below,

### **Code:**

### **Results:**



3.25 Final Voting



3.26 Refined Clustering

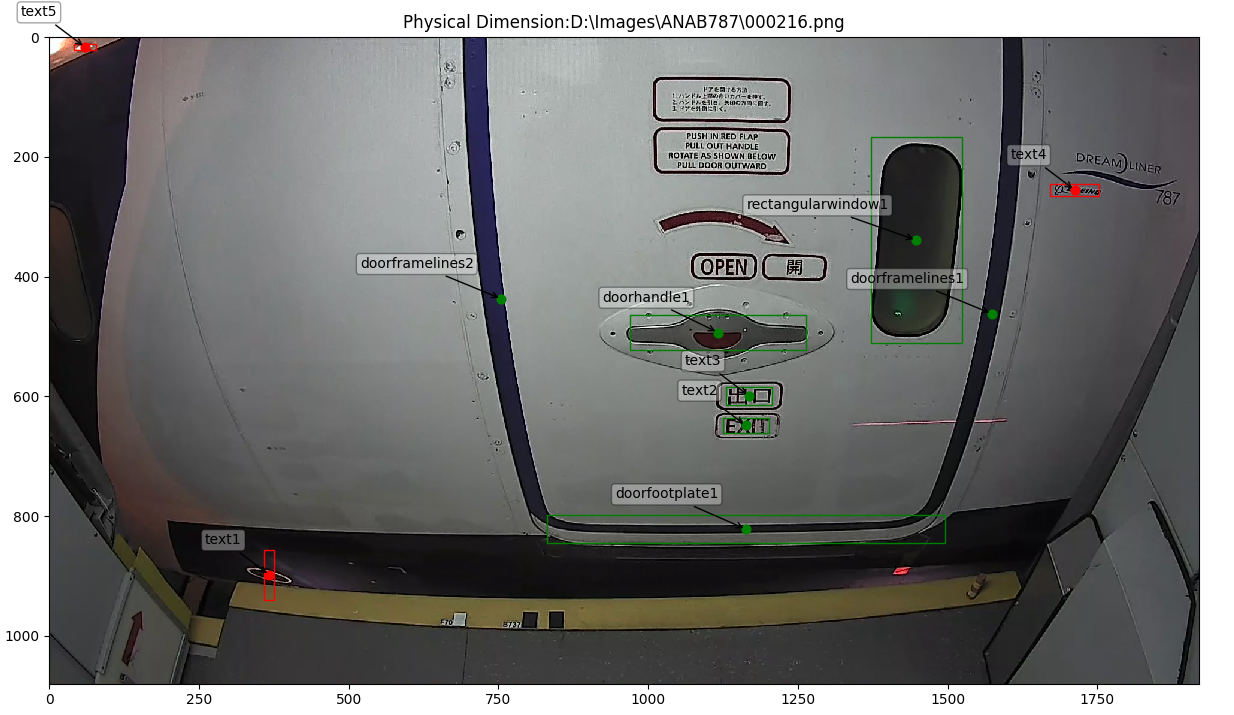
## 3.4.8 Physical Distance & Dimensions

Even though these steps are not counted for voting in the current phase of the project, will prove to be effective in further refining the features. Because this basically checks for two things, one the actual distance and angle of inclination between two features. Two the physical dimensions of every features that passes the algorithm.

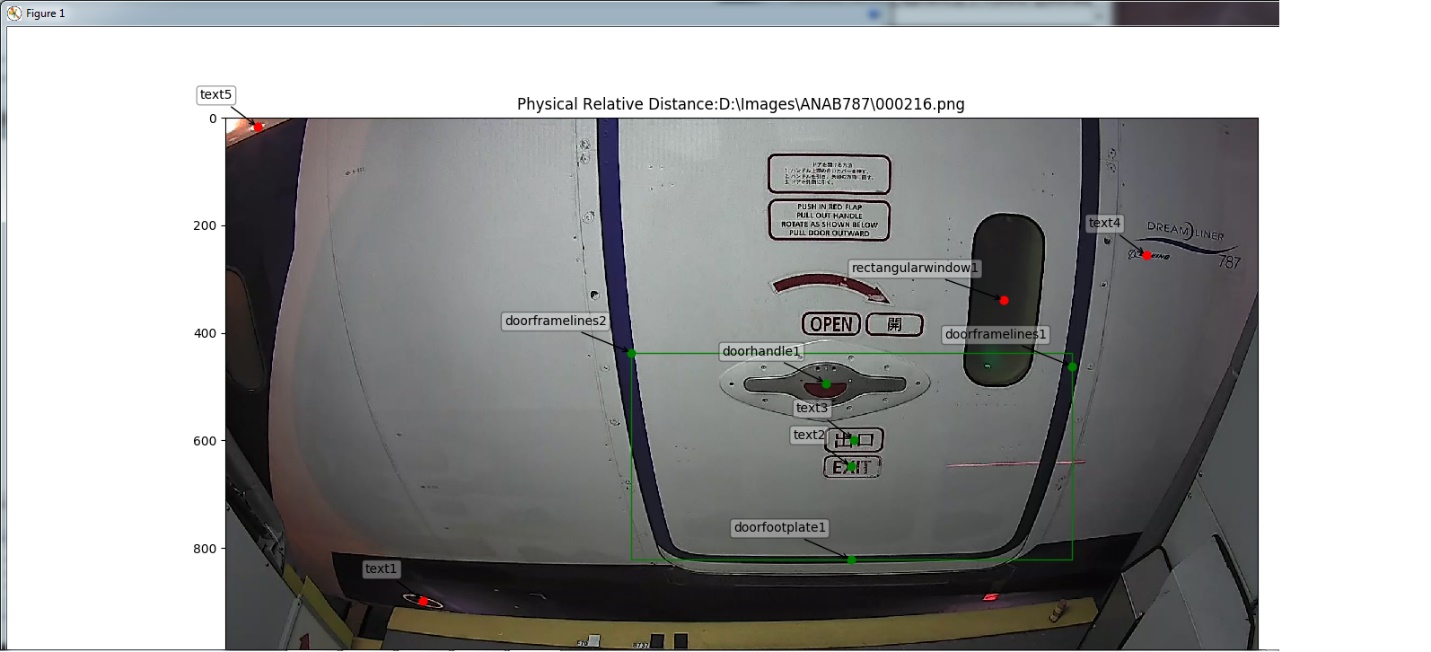
These results are then compared against the data from a database. The database containing the dimensional details of each feature is also manually filled by referring to the aircraft door’s design document. This should be done for all the aircraft models for this test to be effectively utilized.

### **Code:**

### **Results:**



3.27 Physical Dimension Check



3.28 Physical Distance & Angle Check

# Point Selection and Storage

## 3.2.1 Aerobridge Database GUI

The Aerobridge Database GUI is coded in C++ and developed using an SDK called QTCreator. QTCreator is an Open Source SDK from Nokia which is available in both Windows and Linux. Like the vision algorithm, this is also compiled using GCC MinGW Compiler.

This segment of the project involves collection of coordinate information for every feature by manually identifying them and clicking on their corresponding coordinate points. This coordinate information of features is collected for every image in the dataset. The different features and their characteristics will be discussed in detail in the upcoming subsection.

The collection of coordinate information achieved with the help of Aerobridge Database GUI and SQLite Database. The Aerobridge Database GUI is for selecting points on image and the database is for storing the selected points. The database file is portable and can be copied to any System with ease for future re-use. The Aerobridge Database GUI is shown below,

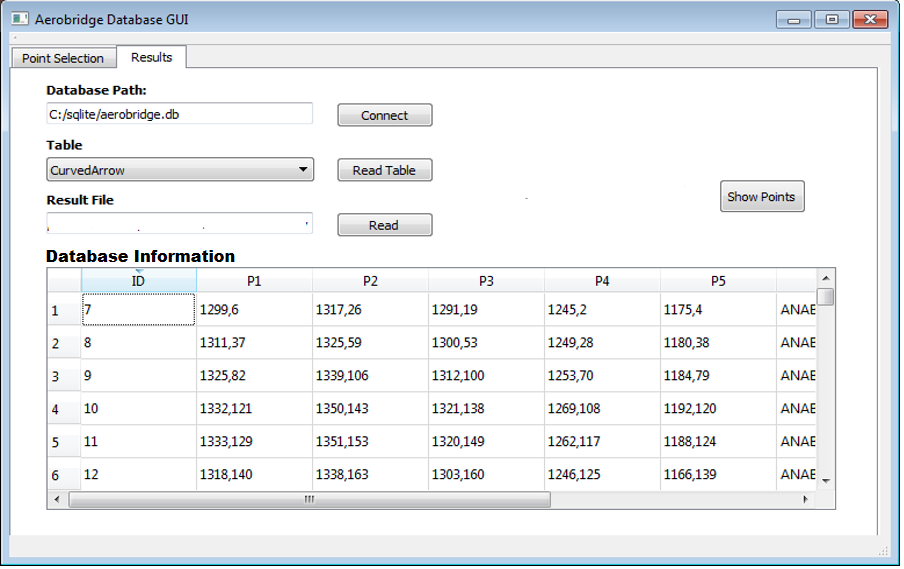


Figure 3.1 “Results” Tab shows points stored in database

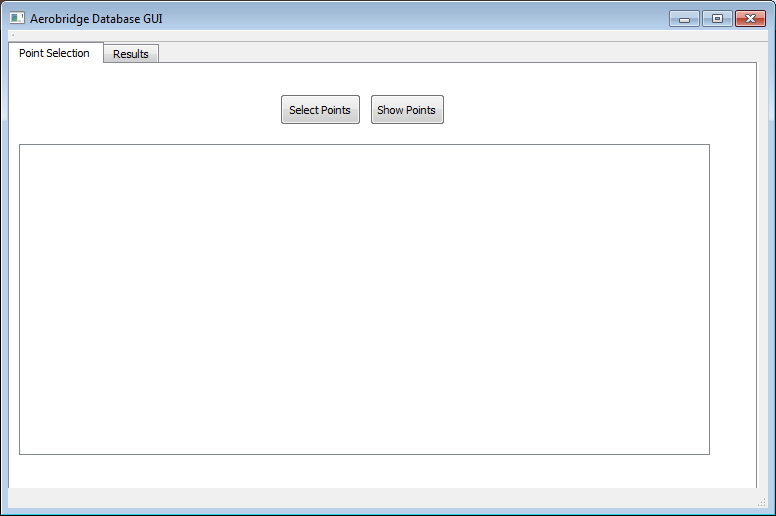


Figure 3.2 “Point Selection” Tab with “Select Points” option

The Aerobridge Database GUI allows the user to manually identify each feature in an image and log their coordinate points. By clicking on “Select Points” button, the user loads the image for which the feature coordinate point information needs to be logged. This opens a dialog showing loaded the image, the user then selects the features that needs to be stored from a drop-down list. Then, selects the corresponding number of points along the x-axis and y-axis for that feature and clicks next “>>”. This automatically saves the selected coordinate points of the respective features to a database as shown in Figure 3.1. The image below illustrates the point selection process for an image,



Figure 3.3 Manually selected points by using Aerobridge Database GUI

This image shows the points manually selected for all the features loaded from the database using the Aerobridge Database GUI,

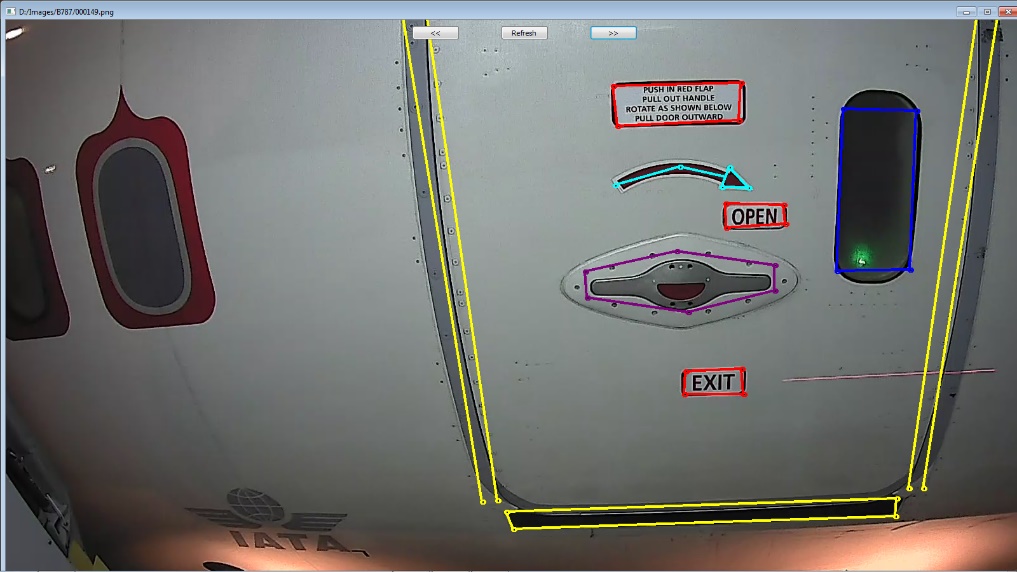


Figure 3.4 Points read and plotted back from the Database

## 3.2.3 Point Storage in SQLite

There are around 8 feature that needs to be stored manually for every image. This is done so that the accuracy of the vision algorithms can be calculated later. The features are listed below,

1. Straight Arrow
2. Curved Arrow
3. Rectangular Window
4. Circular Window
5. Door footplate
6. Door Frame lines
7. Door Handle
8. Text

Note that not all the 8 features are available on every image. Each image contains approximately 6 or less features mentioned above. Because every door frame contains only one window and one / no arrow. The window is either a circular or rectangular window and the arrow is ether straight or curved arrow.

Since each feature differ geometrically from each other, they are stored in specific ways in the database. There is a tradeoff in selecting the number of points for every feature. i.e. storing only few points leads to loss of geometric information and storing too many points increases time when the user selects points manually using Aerobridge Database GUI. To counter this issue the number of points for each feature is statistically determined such that they can be stored quickly with minimal or no loss of dimensional information. The number of points per feature and how they are stored in the database is shown below for every feature,

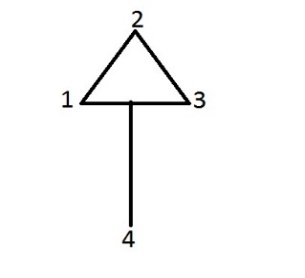
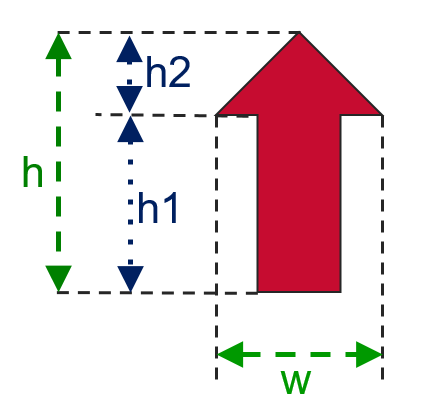


Figure 3.5 Straight Arrow

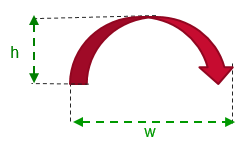


Figure 3.6 Curved Arrow

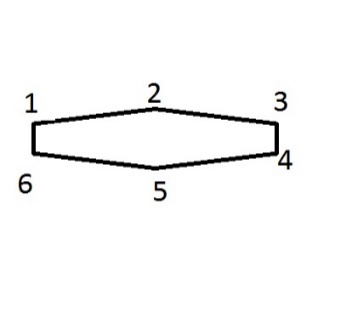
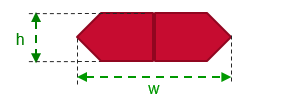


Figure 3.7 Door Handle

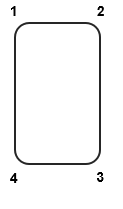
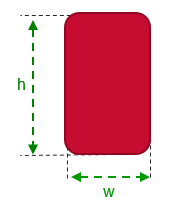
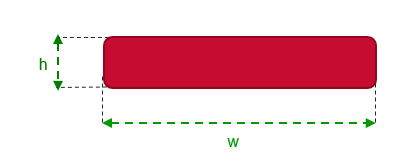


Figure 3.8 Rectangular Window



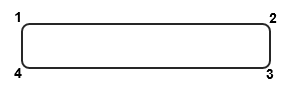
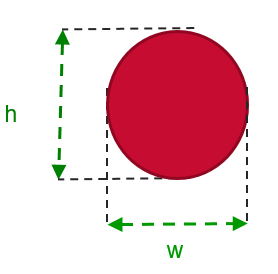


Figure 3.9 Door Footplate



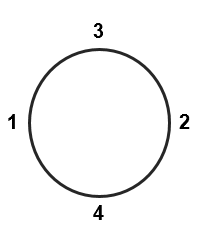


Figure 3.10 Circular Window

These feature co-ordinate points for all the features can be later retrieved from the SQLite Database file and used for the validating the accuracy of Vision Algorithm.