Technical Report
Deciphi-IITK, Team ID: 26285
ICASSP Signal Processing Cup 2020

Nitish Vikas Deshpande, Afzal Rao, Jaya Srivastava, Adhiraj Banerjee, Prof. Dr. Vipul Arora

***Abstract*—This is the technical report for the SP cup challenge. The report is a detailed description of our algorithms and implementation details of the code.**

***Libraries Used*— SKLearn, pmdarima, OpenCV, Keras, Matplotlib, rosbag, Pandas, NumPy**

I. Introduction

We have implemented the code using Python and its libraries. Our code is in the form of Jupyter Notebooks. In the following sections, we will explain in detail the logic behind implementation of each step and refer the reader to the particular Jupyter Notebook which contains the code. We have submitted rendered Jupyter notebooks.

The reader can execute each code block. If one does not want to execute the code blocks, we have provided .html version of the same notebooks. We have tried to provide detailed comments wherever possible in the Jupyter Notebooks. We hope that the reader will benefit from the code that we have provided.

II. Problem Statement

A ROS based dataset with data collected from various sensors like IMU, barometer, GPS and camera mounted on a drone was provided by the organisers. The data was recorded in different scenarios with separate .bag files for normal and abnormal instances.

The task was to train our models on this dataset with IMU and camera synchronised data containing normal as well as abnormal datapoints and build smart prediction algorithms to detect abnormal behaviour that can be tested on new data files.

III. Analysis of Dataset

We used *ROSBAG* library in *Python* to extract the IMU data and the *CV-bridge* library which is compatible with *Open-CV* and *ROS* to extract the image frames. We obtained the video by combining the individual frames. We used tools like *RViz (ROS Visualization)* and the online browser based toolkit *WebViz* to visualize the content of the BAG files. To ensure on the notion that the abnormal files do not purely contain abnormal data points, we implemented a 2 dimensional pca decomposition on the dataset and visualised it using *Matplotlib* as shown in fig 1. The list of all topics extracted from the BAG file in mentioned in the file topics.txt. We provide the plots for comparison of abnormal and normal datapoints in fig2,fig3, fig4,fig5,fig6,fig7.

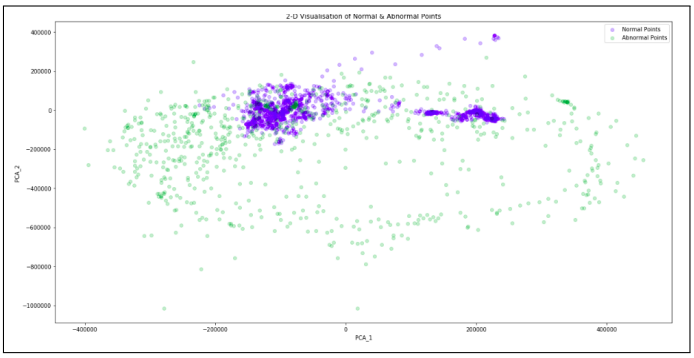

Fig. 1. 2-D Visualisation of Normal (Purple) & Abnormal (Green) Points Using PCA

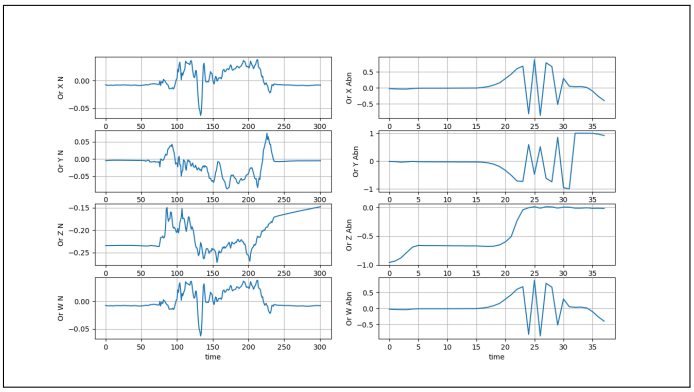

Fig. 2. Orientation

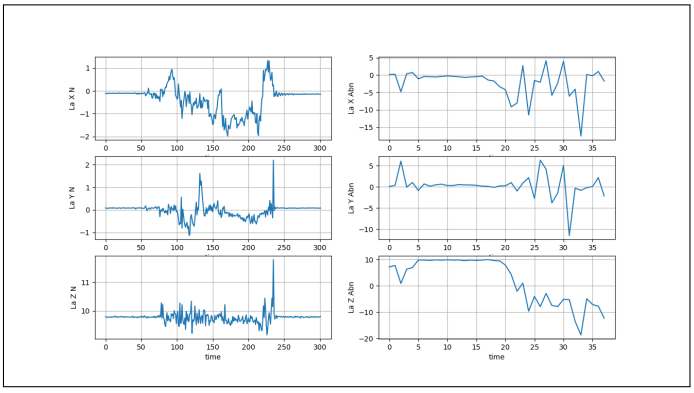

Fig. 3. Linear Acceleration

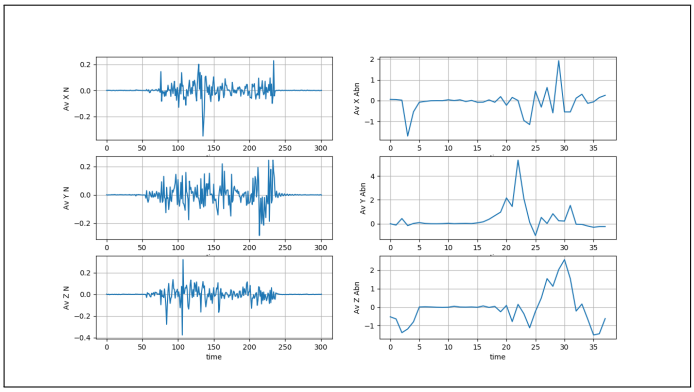

Fig. 4. Angular Velocity

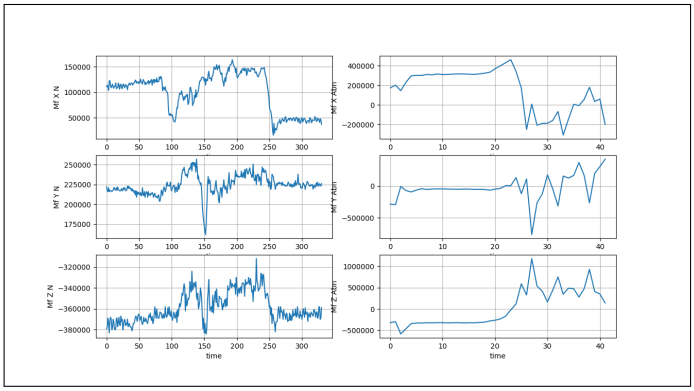

Fig. 5. Magnetic Field

IV. Pre-Processing

A. Resampling and Interpolation

Refer the notebook *"Resampling.ipynb"* for the implementation of re-sampling and interpolation. The code uses *rosbag* library in *Python*. You need to specify the directory which contains the bag files and the name of the bag file in the code block. Refer the pseudo code in algorithm 1.

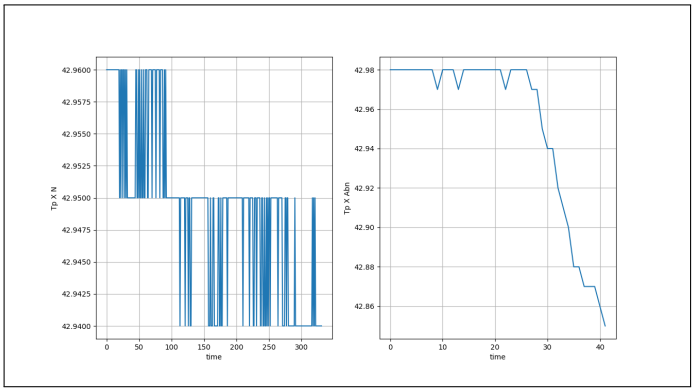

Fig. 6. Temperature

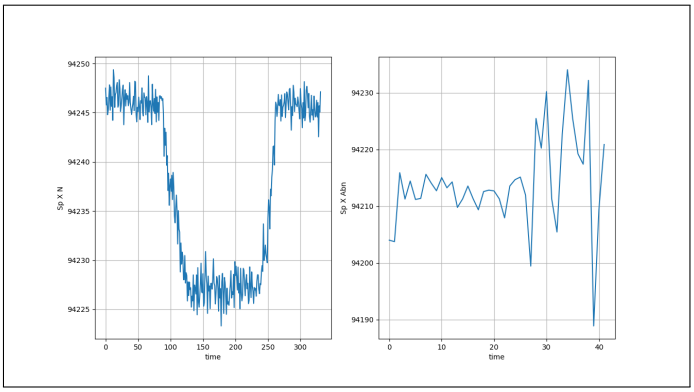

Fig. 7. Static pressure

B. Video Feature Extraction

Refer the notebook *"Video Feature Extraction.ipynb"* for the code of Video Feature Extraction. You need to specify the name of the video file for running this code. We have not attached the video files due to size constraint. This code uses the *Open-CV* library.

From the original video, we extracted frames in the form of .jpg images every 0.05 seconds. This time interval was decided considering computational costs & time complexity of our methodology.

We computed the i^{th} frame's similarity(α_i) with $(i+1)^{th}$ frame in terms of matching keypoints. The score that we defined for quantifying similarity was:

$$\alpha_i = \frac{No.of Matching KeyPoints}{Total No.of KeyPoints} \quad (1)$$

Considering computation costs and performance, we have used ORB(Oriented FAST and Rotated BRIEF) for feature detection, and through brute force matching, matched the Key Points of i^{th} frame with the Key Points of the $(i+1)^{th}$ frame by employing hamming distance as a metric for comparison.

The scores generated by this method were then linearly interpolated over the timestamps provided in the raw images file.

V. Approach 1: Timeforecasting using Convolutional Neural Network

Refer the notebook *"Approach1 Timeforecasting using CNN.ipynb"*. This notebook takes as input the resampled data. The pseudo code for data preparation is explained in algorithm 2. The neural network is built using the *Keras* library. The crux of the architecture is as follows.

```
model = Sequential()
model.add(Conv1D(filters=32,
kernel_size=4, activation='relu',
input_shape=(Hw, D)))
model.add(MaxPooling1D(pool_size=2))
model.add(Conv1D(filters=32,
kernel_size=4, activation='relu'))
model.add(MaxPooling1D(pool_size=2))
```

Algorithm 1: Resampling and interpolation

input : BAG files (both normal and abnormal)
output: All attributes for a given file have equal number of timestamps. The result for each file is stored as a matrix D*M where D is the number of attributes and M is the number of timestamps. The number of timestamps M is different for each file but D is same for each file

for *each BAG file* **do**
- *Extract every attribute as mentioned in table I using rosbag library in Python.*
- *Find the attribute i which contains maximum number of datapoints*
- *For attribute i find the mean sampling frequency F_s*
- *Create a new array of timestamps for attribute i with uniform spacing of $\frac{1}{F_s}$ between 2 consecutive timestamps*
- **for** *each attribute* **do**
  - Use the new array of timestamps with uniform spacing and find the corresponding new value of the attribute using linear interpolation
- Store the new values for every attribute in a D*M matrix with D as number of attributes and M as number of timestamps

TABLE I
DATA SET DESCRIPTION

File Name	Attribute				
	Or, Av, La	Mag Field	Temp	Press	Camera
Normal File 0	302	331	332	332	161
Normal File 1	125	136	135	135	51
Normal File 2	134	134	134	133	50
Normal File 3	150	149	149	149	55
Normal File 4	131	131	131	131	48
Normal File 5	142	141	141	141	53
Abnormal File 0	38	42	42	42	20
Abnormal File 1	129	128	129	128	44
Abnormal File 2	144	143	143	143	49
Abnormal File 3	146	145	145	145	40
Abnormal File 4	136	136	136	136	46
Abnormal File 5	157	156	156	156	51

```
model.add(Flatten())
model.add(Dense(50, activation='relu'))
model.add(Dense(D))
```

Note that the same notebook also contains the evaluation code using EM curve. We have calculated the EM score for this model and later compared with other approaches. The unified analysis will be discussed in the Results section.

VI. Approach 2: Timeforecasting using ARMA Model

Refer the notebook "Forecasting Module.ipynb".
We have designed a forecasting system which models the training data with an Auto Regressive Moving Average (ARMA) model. We have used the popular statistical library, *pmdarima (pyramid-arima, equivalent of R's auto.arima functionality)* for ARMA modelling of each time series recorded by the whole system.

For tuning the parameters of our ARMA models, we have used the auto-arima function. We then designed a rolling forecaster of the IMU data which predicts the next instant based on the train values and present values (as recorded by the system).

Algorithm 2: Data Preparation

input : Resampled and interpolated data matrix as obtained from algorithm 1
output: Input data to neural network X matrix and Target data to the neural network y matrix

*Choose a suitable history window **Hw**, a suitable prediction length **Pl** and a suitable hop size **H***
for *each attribute* **do**
- Starting from time index 0, store the first ***Hw*** datapoints in row of X and the next ***Pl*** datapoints in row of y
- Shift the index pointer by ***H*** timesteps and repeat the above step till the index pointer reaches $M - Hw - Pl$ where M is the length of the timeseries of the input sequence

Algorithm 3: Rolling Forecaster

input : IMU data as recorded by the system
output: IMU data estimate as predicted by the ARMA model

Initialize a list of name history composed of the training data.
Fit the best ARMA model for the IMU data with history.
for *each attribute* **do**
- Predict IMU estimate one step in the future.
- Add the IMU data recorded by the system to history.
- Fit the ARMA model which models the IMU data with history
- Store the predicted values in an array.

VII. Approach 3: Isolation Forest

Refer the notebook *"PCA In-Depth Analysis.ipynb"* for training code of the dimensionality reduction model & *"CLASSIFIER.ipynb"* for the training code of the Isolation Forest Classifier. Both were implemented using the *SKlearn Framework* in *Python*. The given models trained are stored through the *Pickle* Module available in *Python*. The models are saved as *"pca_transform.sav"* (for PCA Model) & *"iforest_classifier.sav"* (for Isolation Forest Model).

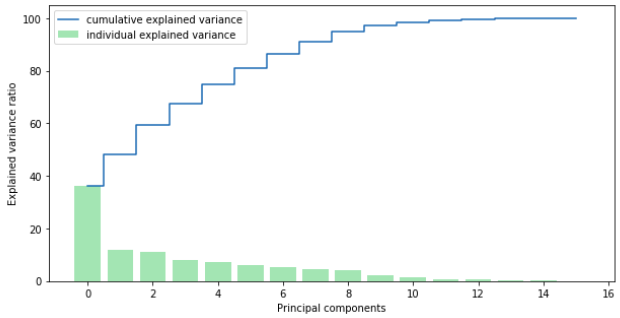

Fig. 8. Plot for Cumulative Explained Variance Ratio & No. of Principal Components

Refer the notebook *"Main_File.ipynb"* where in a test csv files are read according to the given time stamps provided in the original bag files. The data points computed according to the time stamps are transformed through the loaded pca model that we had trained earlier, & an anomaly score is computed through the isolation forest model that we had trained earlier. Out of the twelve abnormal & normal files, 5 abnormal files & 5 normal files were selected for training and the remaining 1 normal file & 1 abnormal file was used for testing the method. From the raw bag files, the following parameters were extracted:

- Orientation (x,y,z,w) [from /mavros/imu/data]
- Angular Velocity (x,y,z) [from /mavros/imu/data]
- Linear Acceleration (x,y,z) [from /mavros/imu/data]
- Magnetic Field (x,y,z) [from /mavros/imu/mag]
- Static Pressure [from /mavros/imu/static_pressure]
- Frame by Frame Similarity Score [derived from /pylon_camera_node/image_raw]

All these parameters were extracted by employing the rosbag package available in Python. The Frame by Frame Similarity Score was derived from the Video Feature Extraction method that has been described earlier.

To remove any sort of redundancy between the variables and also improve the performance of our classification task, we decided to employ PCA, which is an unsupervised dimensionality reduction technique.

One of the major challenges in implementing the PCA model was to determine what number of principal components were required. This was determined by looking at the cumulative explained variance ratio as a function of the number of principal components as shown in fig 8 . It was observed that around 91% of the explained variance was contained within 8 principal components. Henceforth, the number of principal components chosen for our PCA model was 8. The reduced data points were then fed into an Isolation Forest Model, with the parameter of n_iter was set to 300 as mentioned in 4. The contamination parameter, that considers the proportion of outliers present in the training set, was set to 'auto'. It was used to define the threshold on the scores of the samples that were trained. The PCA & Isolation Forest models trained were saved in .sav format using the Pickle library available in Python for future use.

Algorithm 4: Isolation Forest Classifier

input : d-dimensional data point & a positive integer n_iter
output: Isolation Score, a floating number between -1 & 1
Select a data point in the dataset ;
Declare a float one dimensional array A of size n_iter ;
while *n_iter>0* **do**
  for *Each feature in dataset* **do**
    Set a range to isolate between minimum & maximum
  while *Point is not isolated* **do**
    Choose a feature randomly
    Pick a value that is within the range
    if *Chosen value keeps point above* **then**
      Switch minimum of range of feature to this value ;
    else
      Switch maximum of range of feature to this value ;
  isolation number = No. of times the loop is iterated ;
  Normalise isolation number b/w -1 & 1;
  A[n_iter] = isolation number;
  n_iter = n_iter - 1;
isolation number = mean(A);
return isolation number;

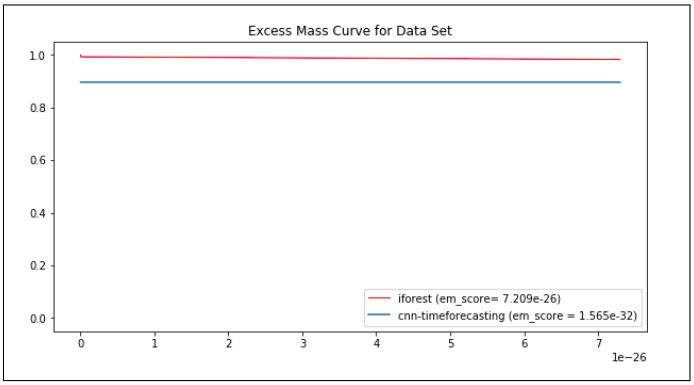

Fig. 9. EM Curve of CNN Based Method (Blue) & Isolation Forest Based Method(Red)

VIII. Results

The reader can find the computation of the EM scores in the individual notebooks of the approaches. We have made a combined plot of the EM curves in fig 9 for ease of comparison. The Iforest method performs better as compared to the timeforecasting method. We refer the reader to the future scope section in our technical paper which discusses about the limitations of our algorithms and proposes scope of future work.