CAPSTONE REPORT

ROBOT NAVIGATION

Gbemileke Onilude

May 10th, 2019.

1. Definition

Project Overview

Navigation for robots as always been on the front line of research, Vision-based navigation is on the top front of research and implementation when it comes to robotic navigation but there are some instances where its implementation can be limited. That is where an inertial measurement unit comes into play, though they are been around for a while, their usefulness cannot be ignored. An inertial measurement unit (IMU) is an electronic device that measures and reports a body's specific force, angular rate, and sometimes the magnetic field surrounding the body, using a combination of accelerometers and gyroscopes, sometimes also magnetometers. IMUs are typically used to manoeuvre aircraft, including unmanned aerial vehicles (UAVs), among many others, and spacecraft, including satellites and landers. Recent developments allow for the production of IMU-enabled GPS devices. An IMU allows a GPS receiver to work when GPS signals are unavailable, such as in tunnels, inside buildings, or when electronic interference is present.

This research is to help robots recognize the floor surface they're standing on using data collected from Inertial Measurement Units (IMU sensors). IMU sensor data has been collected while driving a small mobile robot over different floor surfaces on the university premises. The task is to predict which one of the nine-floor types (carpet, tiles, concrete) the robot is on using sensor data such as acceleration and velocity. Succeed and you'll help improve the navigation of robots without assistance across many different surfaces so they won't fall down on the job.

Problem Statement

A blind man wearing a shoe can still tell if the floor he is working on is slippery, sandy or concrete based on his speed and orientation, but for a robot this is one major source of problem, if a robot cannot identify the nature of the ground in which it navigated, it might have problems when it comes to the necessary speed and forces need to move through its environment. It might use so much force where little is require and little where much is needed. While some robots are required to change the shape of their tyres just to be able to navigate this terrain effectively, but without knowledge of their terrain, this could be a major problem.

Metrics

The aim of the project is to build a model with high accuracy since it is dealing with a multi-class classification model. Therefore accuracy will be a major metric in which the model is optimized with. Accuracy is the number of correct prediction over the total number of prediction.

2. Analysis

Data Exploration

The data to be used is gotten from Kaggle. It is a measurement gotten from a robot in which an IMU is attached and made to walk over different surface areas, It dataset is divided into 3 parts, The training feature set (X_train), the testing feature set (X_test) and the training target set (y_train). The feature set has input data, covering 10 sensor channels and 128 measurements per time series plus three ID columns.

ID Columns

- row_id: The ID for this row.
- series_id: ID number for the measurement series. Foreign key to y_train/sample_submission.
- measurement_number: Measurement number within the series.

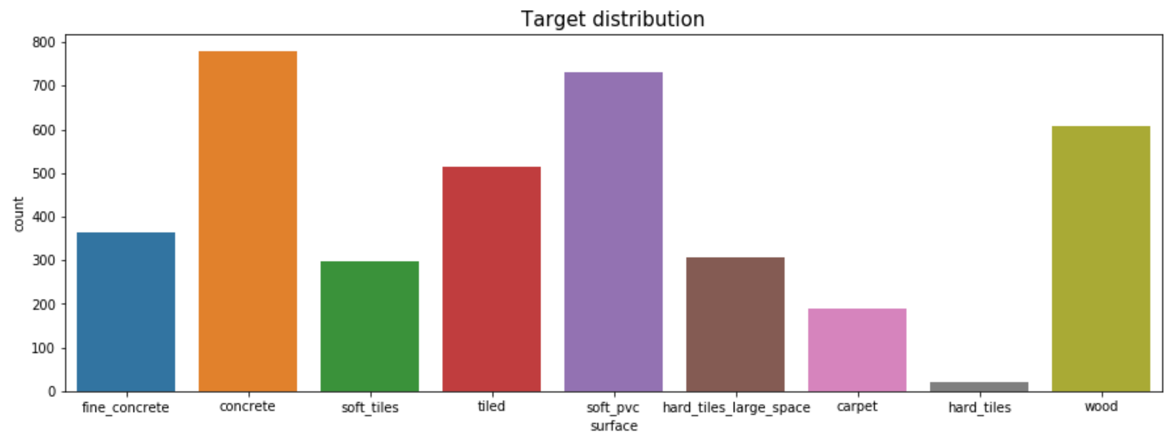
Sensor Channels

- orientation_X
- orientation_Y
- orientation_Z
- orientation_W
- angular_velocity_X
- angular_velocity_Y
- angular_velocity_Z
- linear_acceleration_X
- linear_acceleration_Y
- linear_acceleration_Z

The orientation sensor channel for X, Y, Z and W are all in quaternion angles and will need to be converted to Euler angle (row, yaw and pitch), this is because Euler angle will better represent different movement from different terrain as the robot move across. Linear acceleration sensor channel is for collecting the linear acceleration of the robot while the angular velocity sensor channel is for angular velocity in both X, Y and Z axes. The dataset has no null value. The training dataset has 3810 series measurement, with each series having 128 data points.

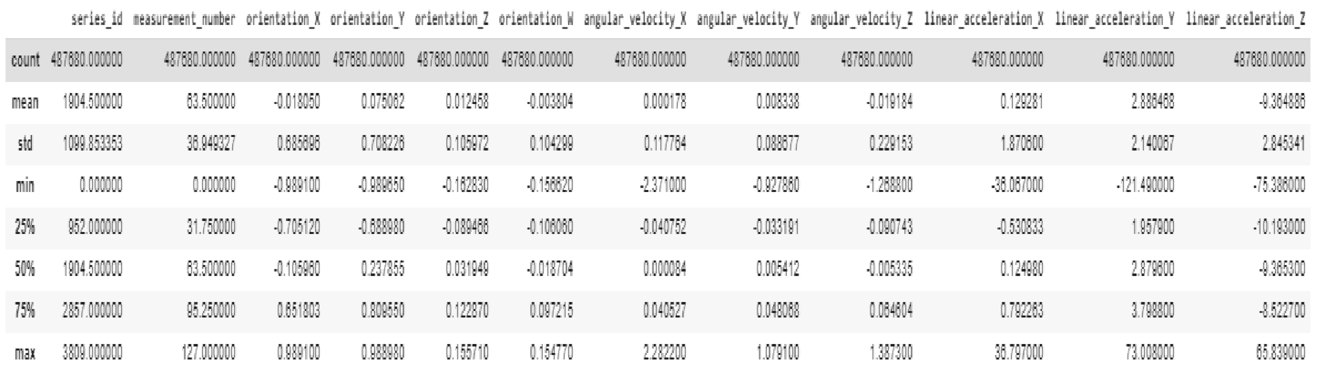
The target training set (y_train), has the following information.

- series_id: ID number for the measurement series.
- group_id: ID number for all of the measurements taken in a recording session. Provided for the training set only, to enable more cross-validation strategies.
- surface: The different surface the robot moved upon (the target). It contains 9 distinct surface terrain ()



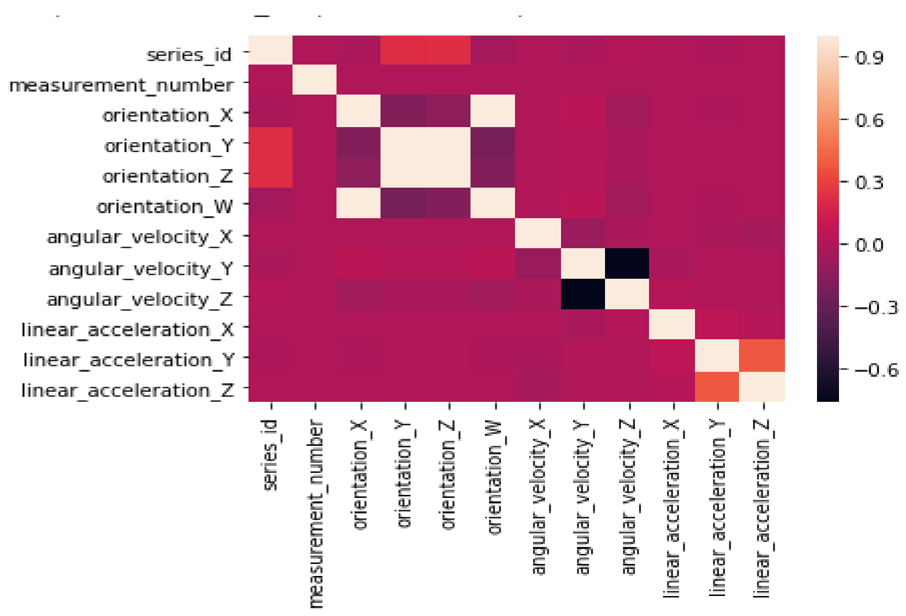
Exploratory Visualization

The diagram below describes the mean, median and stander derivation of each column. There are no abnormal behaviour in this section only that linear acceleration for both X, Y and Z has the widest range of values,



	series_id	measurement_number	orientation_X	orientation_Y	orientation_Z	orientation_W	angular_velocity_X	angular_velocity_Y	angular_velocity_Z	linear_acceleration_X	linear_acceleration_Y	linear_acceleration_Z
count	487680.000000	487680.000000	487680.000000	487680.000000	487680.000000	487680.000000	487680.000000	487680.000000	487680.000000	487680.000000	487680.000000	487680.000000
mean	1904.500000	63.500000	-0.018050	0.075062	0.012458	-0.003804	0.000178	0.008338	-0.019184	0.129281	2.886468	-9.364886
std	1099.853353	36.949327	0.685696	0.708226	0.105972	0.104299	0.117764	0.088677	0.229153	1.870600	2.140067	2.845341
min	0.000000	0.000000	-0.989100	-0.989650	-0.162830	-0.156620	-2.371000	-0.927860	-1.268800	-36.067000	-121.490000	-75.386000
25%	952.000000	31.750000	-0.705120	-0.688980	-0.089466	-0.106060	-0.040752	-0.033191	-0.090743	-0.530833	1.957900	-10.193000
50%	1904.500000	63.500000	-0.105960	0.237855	0.031949	-0.018704	0.000084	0.005412	-0.005335	0.124980	2.879600	-9.365300
75%	2857.000000	95.250000	0.651803	0.809550	0.122870	0.097215	0.040527	0.048068	0.064604	0.792263	3.798800	-8.522700
max	3809.000000	127.000000	0.989100	0.988980	0.155710	0.154770	2.282200	1.079100	1.387300	36.797000	73.008000	65.839000

Looking at the correlation plot, some interesting relation exists between some various columns. Column “orientation_X” has a positive correlation with “orientation_W”, Column “orientation_Y” has a positive correlation with “orientation_Z” and lastly, Column “angular_velocity_Y” has a negative correlation with “angular_velocity_Z”



From the target distribution figure, concrete has the highest values while hard tiles have the lowest.

Algorithms and Techniques

The algorithms to be used is LSTM precisely CUDNNLSTM. This is due to the fact that the dataset is a measurement series. It will have a shape of (128, 10), it will then be accompanied by some Dense neural network. Also, Fourier Transformation will be applied to parts of the dataset and concatenate with the rest of the neural network. The optimizer to be used is Adam because it works well will LSTM. K-fold technique will be used to supplement for the uneven distribution of the target values and also because of a few datasets

Benchmark Model

The benchmark used was gotten from this Kaggle user who has an accuracy of 0.6381 in the testing dataset. He makes use of a Recurrent Neural Network. Firstly he divided the state set into train and testing, drop the 'row_id' "series_id" and "measurement_number" column, encode the output values and created an input neural network of shape (128, 10), two hidden layers and an output layer of size nine. The optimizer used was ‘adam’

3. Methodology

Data Pre-processing

The first pre-processing was to check if there were any missing values within the data set, from the process carried out, there were no missing values within the dataset. The next pre-processing action taken was to convert the quaternion angle to Euler angle. The following formula was used to calculate the row, yaw and pitch angle of the robot.

```
def  quat_to_euler(X):
 #roll
 sinr_cosp = 2.0 * (X['orientation_W'] * X['orientation_X'] +
           X['orientation_Y'] * X['orientation_Z'])
 cosr_cosp = 1.0 - 2.0 * (X['orientation_Y'] * X['orientation_Y'] +
           X['orientation_Z'] * X['orientation_Z'])
 roll = np.arctan(np.array(sinr_cosp), np.array(cosr_cosp))
 #Pitch
 pitch = np.arcsin(np.array(2*(X['orientation_W']*X['orientation_Y'] -
                X['orientation_Z']*X['orientation_X'])))
 #Yaw
 siny_cosp = 2.0 * (X['orientation_W'] * X['orientation_Z'] +
           X['orientation_X'] * X['orientation_Y'])
 cosy_cosp = 1.0 - 2.0 * (X['orientation_Y'] * X['orientation_Y'] +
           X['orientation_Z'] * X['orientation_Z'])
 yaw = np.arctan(np.array(siny_cosp), np.array(cosy_cosp))


 return np.array([roll, pitch, yaw])
```

The following columns was then drop: orientation_X, orientation_Y, orientation_Z, orientation_W.

The next preprocessing was to create another subset of the training data which included all the column expect row, yaw and pitch columns, a Fourier transformation was then performed on the rest of the data using this code

```
```

```
def absfft(x):
    return np.abs(np.fft.rfft(x))
feat_fft_array = np.copy(feature_arr[:,:,:6])
feat_fft_array = np.apply_along_axis(absfft,1,feat_fft_array)
```

The last pre-processing done was to normalise the data set, both the main and the subset (Fourier Transformation).

Implementation

The LSTM architecture is as follows. Since the data set has now been divided into two part, X_train(128, 9), and X_train_fft(65, 6), therefore two architecture was used and then they were later concatenated together.

First Architecture
It has 6 CuDNNLSTM layers with the following unit (200, 200, 100, 100, 50, 50), a dropout was inserted between each layer. Then three Dense layers with the unit (80, 40, 20) also a dropout was inserted between each layer and a BatchNormalization() also before each dropout. Activation used was "relu"

Second Architecture
It has 6 CuDNNLSTM layers with the following unit (200, 200, 100, 100, 60, 60), a dropout was inserted between each layer. Then three Dense layers with the unit (70, 50, 20) also a dropout was inserted between each layer and a BatchNormalization() also before each dropout. Activation used was "relu"

The two architecture was then concatenated and a hidden layer and an output layer of unit nine, for the nine different surface texture. The activation function for the output layer was softmax.

```
def LSTM(drop):
   input_data = Input(shape=(128,9))
   input_data_fft = Input(shape=(65,6))


   # First Architecture

   x = CuDNNLSTM(units=200,
return_sequences=True,
return_state=False)(input_data)
   x = Dropout(drop)(x)

   x = CuDNNLSTM(units=100,
return_sequences=True,
return_state=False)(x)
   x = Dropout(drop)(x)

   x = CuDNNLSTM(units=50,
return_sequences=True,
return_state=False)(x)
   x = Dropout(drop)(x)
   x = CuDNNLSTM(units=50,
return_sequences=False,
return_state=False)(x)
   x = Dropout(drop)(x)
   x = Dense(80)(x)
   x = BatchNormalization()(x)
   x = Activation("relu")(x)
   x = Dropout(drop)(x)

   x = Dense(40)(x)
   x = BatchNormalization()(x)
   x = Activation("relu")(x)
   x = Dropout(drop)(x)

   x = Dense(20)(x)
   x = BatchNormalization()(x)
   x = Activation("relu")(x)
   x = Dropout(drop)(x)


   # Second Architecture

   x1 = CuDNNLSTM(units=200,
return_sequences=True,
return_state=False)(input_data_fft)
   x1 = Dropout(drop)(x1)
   x1 = CuDNNLSTM(units=200,
return_sequences=True,
return_state=False)(x1)
   x1 = Dropout(drop)(x1)
   x1 = CuDNNLSTM(units=60,
return_sequences=True,
return_state=False)(input_data_fft)
   x1 = Dropout(drop)(x1)
   x1 = CuDNNLSTM(units=60,
return_sequences=False,
return_state=False)(x1)
   x1 = Dropout(drop)(x1)
   x1 = Dense(70)(x1)
   x1 = BatchNormalization()(x1)
   x1 = Activation("relu")(x1)
   x1 = Dropout(drop)(x1)

   x1 = Dense(50)(x1)
   x1 = BatchNormalization()(x1)
   x1 = Activation("relu")(x1)
   x1 = Dropout(drop)(x1)

   x1 = Dense(20)(x1)
   x1 = BatchNormalization()(x1)
   x1 = Activation("relu")(x1)
   x1 = Dropout(drop)(x1)


   x = concatenate([x,x1])

   x = Dense(50)(x)
   x = BatchNormalization()(x)
   x = Activation("relu")(x)
   x = Dropout(drop)(x)

   output = Dense(9,
activation="softmax")(x)

   model = Model(inputs= [input_data,
input_data_fft], outputs=output)

model.compile(loss='categorical_crosse
ntropy', optimizer=Adam(),
metrics=['accuracy'])

   return model
```

Refinement

Dropout: the dropout, 0.2 was first used, but the model begins to overfit, therefore I moved to 0.5.

Epoch and Batch Size: Base on intuition, I went with 200 for epoch and 60 for batch size, but any increase in epoch did not produce any improvement in the accuracy, so I stay with this.

K fold: At first I used 5 but the improvement starts to reduce by the third round so I want with 3

4. Result

Model Evaluation and Validation

The Model did not perform up to expectation despite all the optimization perform on it. The best accuracy I saw was 0.5. The model overfit very easily and this is due to the fact that the model I decided to use, requires a lot of data to function optimally.

Justification

My final result was not up to the benchmark discuss earlier, LSTM is a model that require a lot of data set to junction properly and using it for this few data leads to over-fitting easily.

5. Conclusion

Reflection

So the model works by taking the measurement series from a robot over a period of 128 steps, it then passes this data over some pre-processing: conversion from quaternion to Euler angle and also perform Fourier transformation on it. It is then pass through a series of LSTM layers and Dense Layer to determine an output which is the type of surface texture the robot is working on.

The most interesting aspect of the project was the pre-processing stage, I had to do a lot of reading on time series and also Fourier transformation for frequency recognition, but the most changeling will be the turning of the hyper-parameters because the model does not perform up to expectation at all.

Improvement

A different model should be used because LSTM requires a lot of data to function properly but base on the available data given, this is not the case. Base on this, I propose the used of Random forest or 1D Convolution Network. Also if the coefficient of friction can be included in the dataset, it will go a long way in helping the performance of the model