Surface Detection by Robot Movements

Marian Dumitrascu

March 19, 2019

Introduction

For this project I choosed a Kaggle.com open competion project. This is *CareerCon 2019 - Help Navigate Robots*. Here is the description of the project from Kaggle:

> *In this competition, you'll help robots recognize the floor surface they're standing on using data collected from Inertial Measurement Units (IMU sensors). We've collected IMU sensor data 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.*

The task is challenging, but I believe I can obtain a decent result using techniques and tools learned in this course. Please note that many code chunks are not showing in the PDF for aestetic reasons. You can find the code behind many of the graphs and images in the Rmd version.

Report Structure

1. First I will describe data provided and the output expected
2. Then, will perform data analysis, visualization and get insights
3. pre-process and transform data
4. decide the model to use, measure its performance and tune it
5. perform the final data prediction and show the results, submit to Kaggle
6. draw some conclusions

Report and Data Location

I also keep this project on GitHub here: https://github.com/mariandumitrascu/ph125_9_HelpRobotsNavigate Data loaded by the R scripts is kept on an AWS public S3 bucket, to bee easily loaded. This will bee available for the duration of grading.

Data Description

Input data from Kaggle consists in 4 files:

- *X_trian.csv* and *X_test.csv* - the input data, covering 10 sensor channels and 128 measurements per time series plus three 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.

The orientation channels encode the current angles of how the robot is oriented as a quaternion (see Wikipedia). Angular velocity describes the angle and speed of motion, and linear acceleration components describe how the speed is changing at different times. The 10 sensor channels are:

  - *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*

- *y_train.csv* - the surfaces for training set.

  - *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*: lables or classes of the training data. this is the element that need to be predicted

- *sample_submission.csv* - a sample submission file in the correct format.

In this report I will split the training data into two partitions, will fit a model on the first one, and measure it's accuracy on the second. I will also use a small part of data to make it run faster. The R script for generating the final resuls will use all data.

Data Analysis

I will make the following assumptions about observations:

- all observations are made using the same robot
- the interval between the 128 observations for each seeries is always the same.
- the surface is a plane, no stairs, hills or valleys

From a physicyst perspective there are thre forces involved: gravitation force, robot propulsion force, and friction force. Gravitation force is constant. Friction force depends on the surface by a coefficient and propulsion is an unknown variable. We need to basically determine the friction coeficient based on a movement pattern. Moving objects will travel longer if the surface has a lower friction than on a surface with higher friction. On the other side, changing direction can be teeper on a surface with higher friction.

First Insights

Quick look at the first 5 rows of the training data:

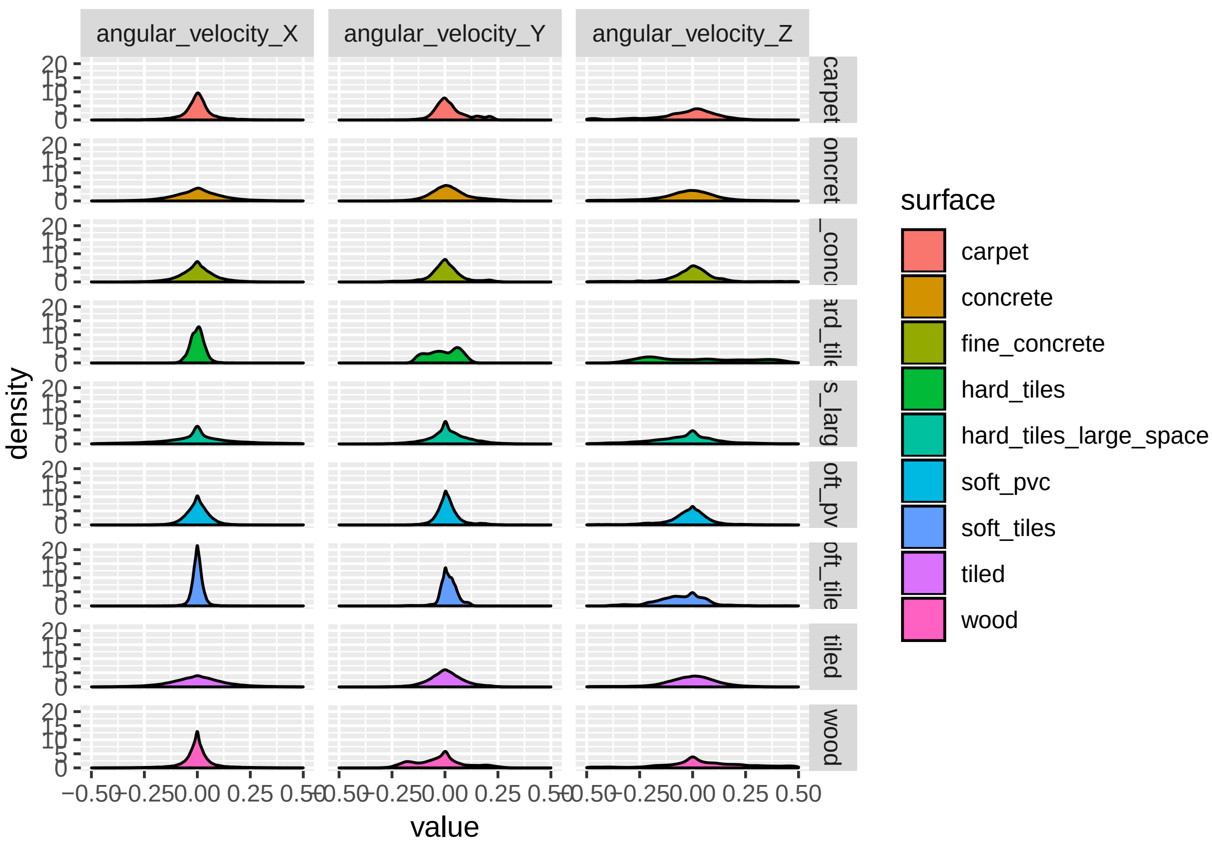
row_id	series_id	measurement_number	orientation_X	orientation_Y	orientation_Z	orientation_W	angular_
0_0	0	0	-0.75853	-0.63435	-0.10488	-0.10597	
0_1	0	1	-0.75853	-0.63434	-0.10490	-0.10600	
0_2	0	2	-0.75853	-0.63435	-0.10492	-0.10597	
0_3	0	3	-0.75852	-0.63436	-0.10495	-0.10597	
0_4	0	4	-0.75852	-0.63435	-0.10495	-0.10596	

Data Distribution

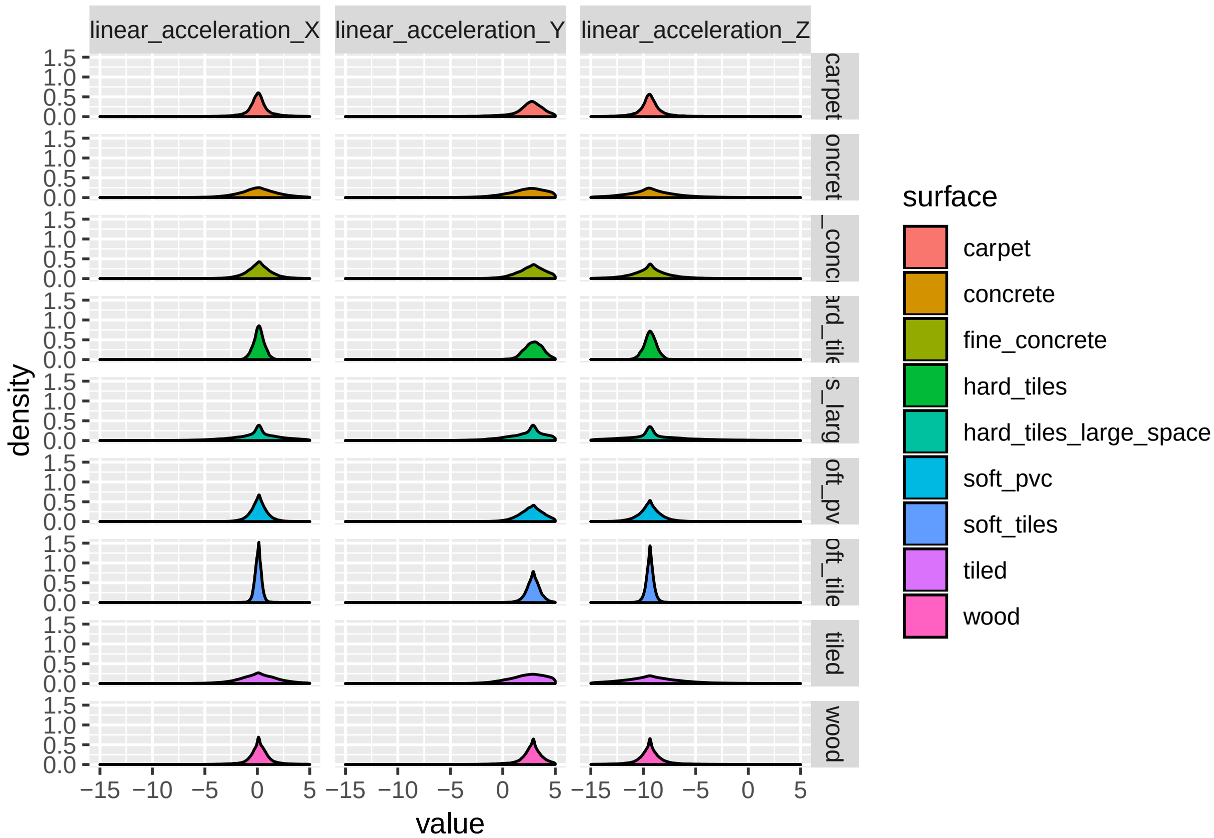
Here is a distributon of measurements by surface in the training set:

surface	measurements
carpet	189
concrete	779
fine_concrete	363
hard_tiles	21
hard_tiles_large_space	308
soft_pvc	732
soft_tiles	297
tiled	514
wood	607

Angular velocity data is produced by a magnetostatic sensor, it indicates the angular speed the robot is movig in reference with earth orientation. Here is the distribution of angular velocity by surface:



Linear acceleration data is produced by an inertial sensor. We can approximate this later with linear distacne if we consider the unit of time to be 1. Here is the distribution of linear acceleration by surface:



We can observe noticeable differences in distribution of these variables by surface.

Orientation data comes from a gyroscop sensor. To draw the distribution of orientation, we'll convert quaternion values to euler angles which are more intuitive and easier to interpret. Euler angles provide a way to represent the 3D orientation of an object using a combination of three rotations about different axes:

- roll - rotation around x, noted *phi*,
- pitch - rotation around y, noted *theta*,
- yaw - rotation around z, noted *psi*

See theimage below (from: http://www.chrobotics.com/library/understanding-euler-angles)

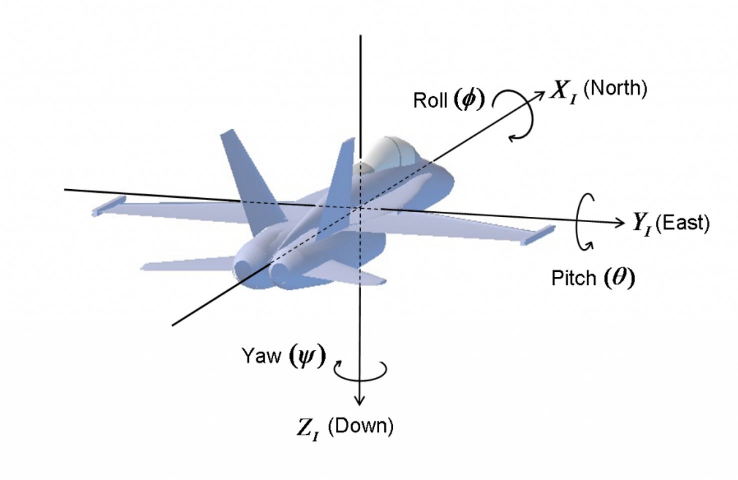


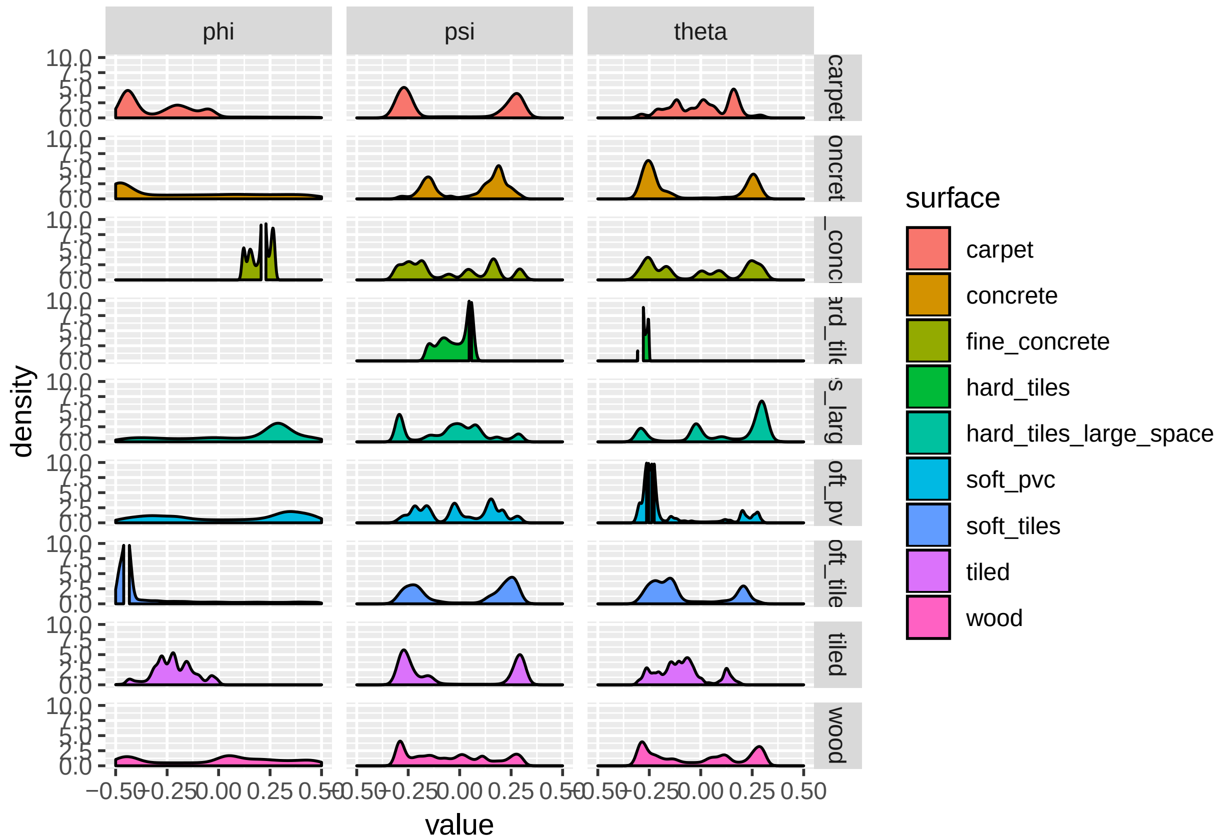
Figure 1: Euler Angles

To convert quaternions to euler angles I used *Q2EA* function in *orientlib* package.(See: https://www.rdocumentation.org/packages/orientlib/versions/0.10.3)

```
# define a function to convert quaternion values to euler angles.
convert_quaternions_to_euler <- function(a_dataset){

    # use Q2EA from RSpincalc to convert quaternions to euler angles
    Q <- a_dataset %>% select(
                    orientation_X,
                    orientation_Y,
                    orientation_Z,
                    orientation_W) %>%
                as.matrix()

    euler_matrix <- Q2EA(Q,
                 EulerOrder='xyz',
                 tol = 10 * .Machine$double.eps,
                 ichk = FALSE,
                 ignoreAllChk = FALSE)

    # add the new columns to the dataset
    a_dataset <- a_dataset %>% mutate(
                    phi = euler_matrix[,1],
                    theta = euler_matrix[,2],
                    psi = euler_matrix[,3])

    # remove quaternion columns
    a_dataset <- a_dataset %>% select(
                -orientation_X,
                -orientation_Y,
                -orientation_Z,
                -orientation_W)

    # return the new dataset
    a_dataset
```

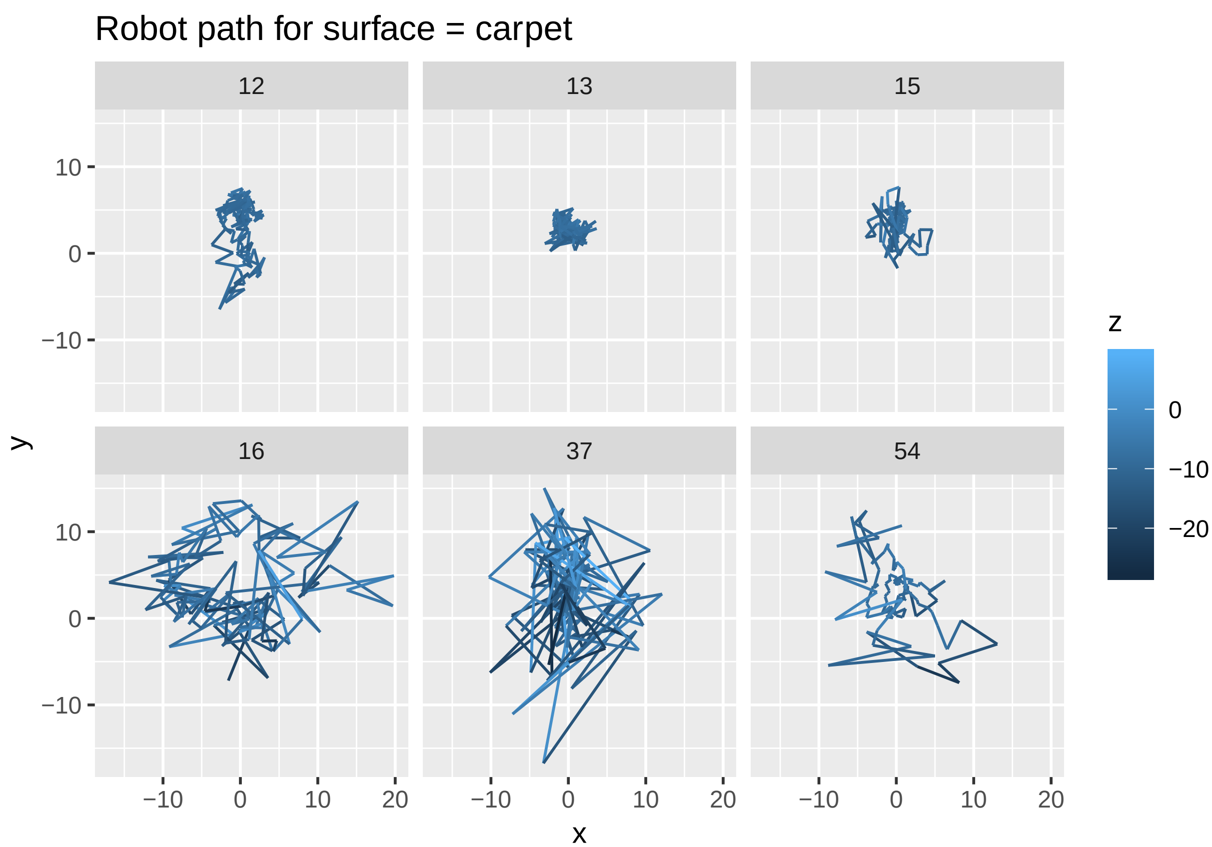
```
}

x_train <- convert_quaternions_to_euler(x_train)
x_test <- convert_quaternions_to_euler(x_test)
```

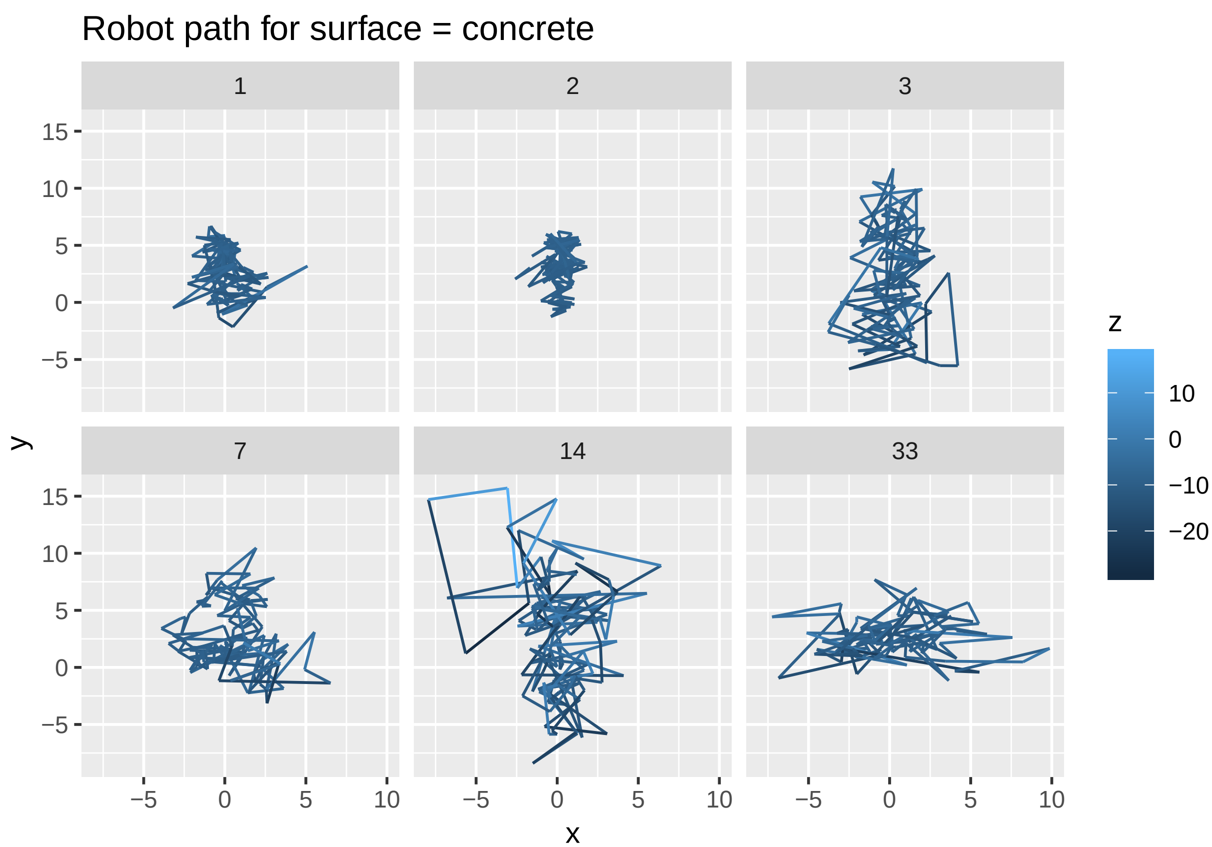
Distribution of orientation angles by surface:



It looks like the most noticeable differences between surfaces can be observed in the orientation distribution.

If we consider unit of time to be 1, we can draw the path of the robot movement. In the following figures I draw the path for several cases, faceted by surface. I expect that the surface would influence the shape of these paths.


Robot path for surface = carpet
12
13
15
16
37
54
10
0
−10
−10
0
10
20
x
y
z
0
−10
−20



Data Pre-Processing

An important step in fitting a good model is pre-processing the data. In our case we havee series of observation of 128 measurements. In this section I will create aggregations around these measurements. These are: mean, standard deviation, distances of segments from one point to the next. total distance, total angle change both from magnetostatic and gyro sensors. All of these are encapsulated in the following function:

```
# function for pre-processing
# n_of_rows defaults to total number ofseries
# is_train indicates that the data is training, thus will do an extra action
pre_process <- function(a_dataframe, n_of_rows = nrow(a_dataframe)/128 ) {

    # get data grouped by seeries_id and compute some means
    processed_data_df <- a_dataframe %>%
    group_by(series_id) %>%
    summarize(
        phi_mean_all = mean(phi),
        phi_sd_all = sd(phi),
        phi_mean_to_sd_all = mean(phi)/sd(phi),
        theta_mean_all = mean(theta),
        theta_sd_all = sd(theta),
        theta_mean_to_sd_all = mean(theta)/sd(theta),
        psi_mean_all = mean(psi),
```

```
    psi_sd_all = sd(psi),
    psi_mean_to_sd_all = mean(psi)/sd(psi),
    # this is the rectangular area that surounds the path of linear movement
    dist_area = (max(linear_acceleration_X) - min(linear_acceleration_X)) * (max(linear_acceleratio
        (max(linear_acceleration_X) - min(linear_acceleration_X)) * (max(linear_acceleration_Z) - mi
        (max(linear_acceleration_Y) - min(linear_acceleration_Y)) * (max(linear_acceleration_Z) - mi
    # this is the rectangular area that surounds the path of angular movement
    omega_area = (max(angular_velocity_X) - min(angular_velocity_X)) * (max(angular_velocity_Y) - mi
        (max(angular_velocity_X) - min(angular_velocity_X)) * (max(angular_velocity_Z) - min(angula
        (max(angular_velocity_Y) - min(angular_velocity_Y)) * (max(angular_velocity_Z) - min(angula
    euler_area = (max(phi) - min(phi)) * (max(theta) - min(theta)) +
        (max(phi) - min(phi)) * (max(psi) - min(psi)) +
        (max(theta) - min(theta)) * (max(psi) - min(psi)),
    dist_mean_x = mean(linear_acceleration_X),
    dist_mean_y = mean(linear_acceleration_Y),
    dist_mean_z = mean(linear_acceleration_Z),
    omega_mean_x = mean(angular_velocity_X),
    omega_mean_y = mean(angular_velocity_Y),
    omega_mean_Z = mean(angular_velocity_Z),
    dist_sd_x = sd(linear_acceleration_X),
    dist_sd_y = sd(linear_acceleration_Y),
    dist_sd_z = sd(linear_acceleration_Z),
    omega_sd_x = sd(angular_velocity_X),
    omega_sd_y = sd(angular_velocity_Y),
    omega_sd_Z = sd(angular_velocity_Z)

    ) %>%
    slice(1:n_of_rows)

# define an empty data frame with summary metrics that we'll use for each set of 128 observations
metrics <- c("dist_total","dist_max","dist_min","dist_max_to_min","dist_mean","dist_sd","dist_mean_
                         "omega_total","omega_max","omega_min","omega_max_to_min","omega_mean","omeg
                         "phi_total","phi_max","phi_min","phi_mean","phi_sd","phi_mean_to_sd",
                         "theta_total","theta_max","theta_min","theta_mean","theta_sd","theta_mean_
                         "psi_total","psi_max","psi_min","psi_mean","psi_sd","psi_mean_to_sd",
                         "euler_total","euler_max","euler_min","euler_mean","euler_sd","euler_mean_
tmp_df <- data.frame(matrix(ncol = length(metrics), nrow = 0) )
colnames(tmp_df) <- metrics

# loop over each series and compute aggegations
# should use apply type of function here, but I use "for" until I master the apply
for (s_id in processed_data_df$series_id)
{
    # get current measurement set
    this_chunk_df <- a_dataframe %>% filter(series_id == s_id)

    # select only columns we are interested in
    this_chunk_df <- this_chunk_df %>%
        select(
        phi, theta, psi,
        angular_velocity_X, angular_velocity_Y, angular_velocity_Z,
        linear_acceleration_X, linear_acceleration_Y, linear_acceleration_Z)
    # this is a vector with euclidian distances from one point to the next
```

```r
    dist_v <-   sqrt(diff(this_chunk_df$linear_acceleration_X)^2 +
                      diff(this_chunk_df$linear_acceleration_Y)^2 +
                      diff(this_chunk_df$linear_acceleration_Z)^2)
    omega_v <-  sqrt(diff(this_chunk_df$angular_velocity_X)^2 +
                      diff(this_chunk_df$angular_velocity_Y)^2 +
                      diff(this_chunk_df$angular_velocity_Z)^2)
    phi_v <- abs(diff(this_chunk_df$phi))
    theta_v <- abs(diff(this_chunk_df$theta))
    psi_v <- abs(diff(this_chunk_df$psi))

    # fill or temp data frame with summary computations for our 128 measurement set
    tmp_df <- bind_rows(tmp_df, data_frame(

        # all features starting with "dist_" refers to linear movement
        dist_total = sum(dist_v),
        dist_max = max(dist_v),
        dist_min = min(dist_v),
        dist_max_to_min = max(dist_v)/min(dist_v),
        dist_mean = mean(dist_v),
        dist_sd = sd(dist_v),
        dist_mean_to_sd = mean(dist_v)/sd(dist_v),  # reciprocal coef of variation

        # all features starting with "omega_" refers to angle velocity measurments
        omega_total = sum(omega_v),
        omega_max = max(omega_v),
        omega_min = min(omega_v),
        omega_max_to_min = max(omega_v)/min(omega_v),
        omega_mean = mean(omega_v),
        omega_sd = sd(omega_v),
        omega_mean_to_sd = mean(omega_v)/sd(omega_v), # reciprocal coef of variation

        # phi, theta and psi reffers to roll, pitch and yaw rotations
        phi_total = sum(phi_v),
        phi_max = max(phi_v),
        phi_min = min(phi_v),
        phi_mean = mean(phi_v),
        phi_sd = sd(phi_v),
        phi_mean_to_sd = mean(phi_v)/sd(phi_v),

        theta_total = sum(theta_v),
        theta_max = max(theta_v),
        theta_min = min(theta_v),
        theta_mean = mean(theta_v),
        theta_sd = sd(theta_v),
        theta_mean_to_sd = mean(theta_v)/sd(theta_v),

        psi_total = sum(psi_v),
        psi_max = max(psi_v),
        psi_min = min(psi_v),
        psi_mean = mean(psi_v),
        psi_sd = sd(psi_v),
        psi_mean_to_sd = mean(psi_v)/sd(psi_v),
```

```r
            # features starting with "euler_" refers to
            # agragation of all phi, theta and psi rotations
            euler_total = sum(phi_v + theta_v + psi_v),
            euler_max = max(phi_v + theta_v + psi_v),
            euler_min = min(phi_v + theta_v + psi_v),
            euler_mean = mean(phi_v + theta_v + psi_v),
            euler_sd = sd(phi_v + theta_v + psi_v),
            euler_mean_to_sd = mean(phi_v + theta_v + psi_v)/sd(phi_v + theta_v + psi_v)
            ))

    } # end of for over series

    # add the summary computations to the data set of series
    processed_data_df <- bind_cols(processed_data_df, tmp_df)

    # more features
    # I added thesee as an experimentation after observing the dependencies graphs
    # will explain later
    processed_data_df <- processed_data_df %>% mutate(
        # this is an agular momentum
        f1 = log(dist_mean_to_sd*omega_mean_to_sd),

        # this is an angular momentum
        f2 = log(dist_total*omega_total),

        # this is the angle between theta and psi vectors
        f3 = abs(atan(theta_mean_all/psi_mean_all)))

    # return the proceessed data
    processed_data_df
}
```

Pre-process the data and save it to a local file. This is to save time for further analysis by skipping the pre-processing procedure. I use *tic() toc()* functions from *tictoc* package to display processing time.

```r
# pre-process train and test data sets
tic("process train data")
x_train_processed <- pre_process(x_train)
toc()
```

```
## process train data: 46.59 sec elapsed
```

```r
tic("process test data")
x_test_processed <- pre_process(x_test)
toc()
```

```
## process test data: 45.82 sec elapsed
```

```r
# rejoin train data with the labels data set
x_train_processed <- x_train_processed %>% left_join(y_train, by = "series_id")

# creeatee a folder "data" if doesnt exist
```

```r
if (!dir.exists("data")) dir.create("data")

# save processed data, and use these files from now on
write_csv(x_train_processed, "data/x_train_processed.csv")
write_csv(x_test_processed, "data/x_test_processed.csv")

# clean some variables and the environment
rm(x_train, x_test, y_train)
rm(x_train_processed, x_test_processed)
```

Load pre-processed data from hard-disk and use them from here. In this report I will use a smaller data set from the training data, to make things work faster. Full data set will be used in the final script

```r
x_train_processed_from_file <- read_csv("data/x_train_processed.csv")
x_test_processed_from_file <- read_csv("data/x_test_processed.csv")

# if we load data from a file, convert surface to factor
x_train_processed_from_file <- x_train_processed_from_file %>% mutate(surface = as.factor(surface))

# use a smaller set of datab to save time
x_train_processed_from_file <- x_train_processed_from_file %>%  slice(1:1000)
```

Feature Analysis

Fit The Model

One-vs-One or One-vs-All

Results and Submit the Data

Conclusion

One of the most important outcome of this prroject is that I learned a few things in addition to what was presented in the course.

Reference

1. Applied Predictive Modeling - Max Kuhn, Kjell Johnson
2. Q2EA: Convert from rotation Quaternions to Euler Angles. Q2EA converts from Quaternions (Q) to Euler Angles (EA) based on D. M. Henderson (1977). Q2EA.Xiao is the algorithm by J. Xiao (2013) for the Princeton Vision Toolkit. https://rdrr.io/cran/RSpincalc/man/Q2EA.html
3. Understanding Quaternions. http://www.chrobotics.com/library/understanding-quaternions
4. Understanding Euler Angles. http://www.chrobotics.com/library/understanding-euler-angles
5. Tune Machine Learning Algorithms in R (random forest case study) by Jason Brownlee. https://machinelearningmastery.com/tune-machine-learning-algorithms-in-r/

6. Classification with more than two classes, from Introduction to Information Retrieval, Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze,, Cambridge University Press 2008 https://nlp.stanford.edu/IR-book/html/htmledition/classification-with-more-than-two-classes-1.html

7. A Guide To using IMU (Accelerometer and Gyroscope Devices) in Embedded Applications - http://www.starlino.com/imu_guide.html