Determination of instantaneous velocity, distance travelled, waiting time between journey from GPS data from a smartphone

- Submitted by

ED11B004

ED14D012

INTRODUCTION

Data smoothing is technique that developed an approximated mathematical function which permits to obtain data patterns, to isolate the noise, outliers and irregularities in a sequence of data points. It is known nonparametric regression as well. When signal speed is smoothed, the local speed variations due to small accelerations of differential trajectory changes are erased from the signal, providing more continuous speed profiles.

There exist several smoothing methods reported in literature such as, splines, kernel-based method, n-exponential methods, and regressions locally weighted methods and ARxMA methods, which permits to obtain smooth data, trends and seasonality patterns of speed. Here spline based smoothing technique has been applied to extract distance travelled,speed and waiting time from the GPS data.

APPROACH

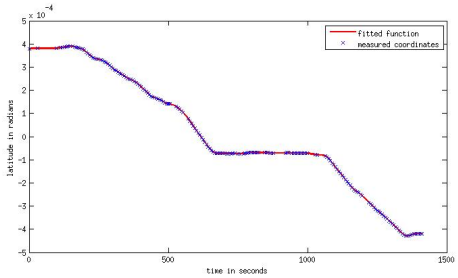
The device returned a dataset with readings of latitude, longitude and a measurement of accuracy with which the reading was made.
As a preprocessing, we removed the entries which was taken at same time instant. this was done because at a time instant, GPS cannot have different location.
From the the filtered out data, the readings has error associated with it. if we directly convert from geodetic coordinate system to cartesian, the errors will also scale up. So, we smoothed the reading directly and fitted in a cubic smoothing spline. this was done by a function in matlab called csaps. The function fits the points in a cubic spline with a weighted regression. The weights we used is inverse square of accuracy associated, because it will give more importance to values recorded with less error.
From the fitted functions, we can find latitude longitude. By differentiating the functions with respect to time. These will represent the angular speeds in geodetic system.
Having θ (we denote latitude), ϕ (we denote longitude), $d\theta/dt$, and $d\phi/dt$, we can find the velocity in cartesian coordinates by partial differentiating transformation matrix.
This will give velocity in cartesian coordinates. the magnitude of this vector is used to predict speed at particular timestamps. this was compared with independently measured speed values and error is noted.

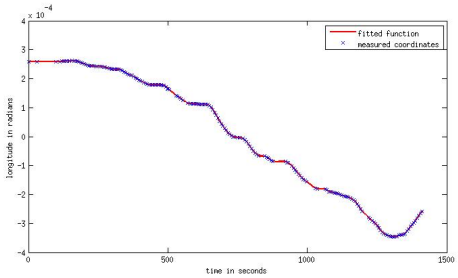
The distance travelled is found out by getting the cartesian points in journey between every 3 seconds gap. we assumed the vehicle travelled straight during this 3 second. So, we found euclidean distance between each points spaced 3 seconds and summed it to get total distance travelled.

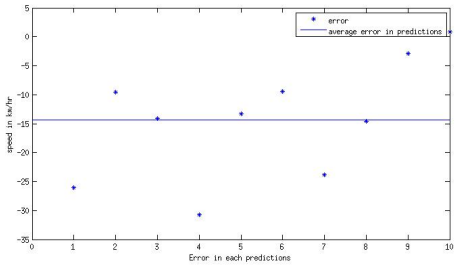
For waiting time, we calculated vector sum of velocity vectors of vehicle during a journey of three meters distance. The magnitude of equivalent velocity is calculated, if the vehicle was in stop, the magnitude should be small since velocity will be less during waiting. if the velocity is less than a particular threshold, we assume it was waiting and that time period is added to total waiting time.

RESULTS

Resulting plots for the data file recorded by us is







Waiting time measured = 121
Waiting time predicted by algorithm = 122

Predicted Speed at time instants(km/hr)	Measured speed (km/hr)	Timestamps(seconds)
3.92254252815292	30	108
12.4866692823235	22	262
22.8687596311970	37	435
19.2331786686273	50	477
28.7117873145713	42	538
30.5153406050419	40	701
19.2016353988614	43	1005
34.3854294787782	49	1095
32.0744249565954	35	1222
28.8925002642299	28	1320

Actual Distance travelled = 8 km.
Predicted distance travelled = 8.18 km.

Measured Travel time = 23 mins
calculated travel time = 23 mins

IMPLEMENTATION BY CODE
The code is written in matlab.
Instructions for running : keep all the files in zip files in same directory, move matlab working directory to that folder and run the file miniproject_main.m.
NOTE: This code is for reference, for running use the code files attached.

```
clc
clear all
load('GPSdata.mat');

%convert to radians
time = Locationdata(:,1);
lat = degtorad(Locationdata(:,2));
lon = degtorad(Locationdata(:,3));
```

```
accuracy = Locationdata(:,4);

clear('Locationdata'); % free up memory

% In the data it is evident that different GPS locations are data present for the same same timestamp
% As this is not physically possible, it is caused by error. So we assign mean of Location data to such time stamps.
[time, goodIndices, ic] = unique(time);
lat = lat(goodIndices);
lon = lon(goodIndices);
accuracy = accuracy(goodIndices);
clear('ic', 'goodIndices');

% Centralize the data
latMean = mean(lat);
lonMean = mean(lon);

lat = lat - latMean*ones(size(lat));          %centralize the data to have mean = 0
lon = lon - lonMean*ones(size(lon));
%If we convert the data to cartesian coordinates before denoising, the associated errors will also get scaled up
% So we will fit the latitude and longitude in a cubic spline
% By fitting the data in a cubic smoothing spline, we do a denoising
% Also by differentiating this, we get the angular velovity in theta and phi directions

% We use csaps function of matlab for fitting

% this does minimize sum(W*|y - f(x)|^2 + (1-p)*integral(lambda*|f''(t)|^2)dt)
% The default value for the piecewise constant weight function λ in the roughness measure is the constant function 1.
% is often near 1/(1 + h^3/6), with h the average spacing of the data sites.
% for uniformly spaced data, p= 1/(1 + h^3/0.6).

h = (time(end)-time(1))/size(time,1);
p = 1/(1+(h^3)/60);

% we use weight function w as 1/accuracy^2
weights = 1./accuracy.^2;
```

```
latFunction = csaps(time, lat, p, [], weights); %Latitude fitted in fuction
lonFunction = csaps(time, lon, p, [], weights); %Longitude fitted in fuction
% Differentiating this will give velocity in each axes
latDotFunction = fnder(latFunction);  % angular velocity function in latitude direction
lonDotFunction = fnder(lonFunction);  % angular velocity function in longitude direction

%to verify our curve fitting, we can plot the initial points and fitted curve
subplot(2,2,1);
fnplt(latFunction,'r');
xlabel('time in seconds');
ylabel('latitude in radians');
hold on;
plot(time, lat,'x');
legend('fitted function','measured coordinates');

subplot(2,2,2);
fnplt(lonFunction,'r');
xlabel('time in seconds');
ylabel('longitude in radians');
hold on;
plot(time, lon,'x');
legend('fitted function','measured coordinates');
% It is clear that all functions are fitted with a good accuracy

%================================== Instantaneous velocity
================================================%
% now we predict the velocities taken independently at timestamps
% Please refer to function getCartesianVelocity inthe parent directory
theta = fnval(latFunction,Timestamps); %temporary variable
phi = fnval(lonFunction,Timestamps);   %temporary variable
thetaDot = fnval(latDotFunction,Timestamps);  %temporary variable
phiDot = fnval(lonDotFunction, Timestamps);   %temporary variable
[xVelocityPredicted, yVelocityPredicted, zVelocityPredicted] = getCartesianVelocity(theta,
phi, thetaDot, phiDot);
clear('theta', 'phiDot', 'thetaDot', 'phi'); % free up memory

% The speed i. e. the magnitude is calculated
```

```
speedPredicted =
sqrt(xVelocityPredicted.^2+yVelocityPredicted.^2+zVelocityPredicted.^2);

speedPredicted = convvel(speedPredicted,'m/s', 'km/h')    % convert speed to km/hour

errors = speedPredicted-Speed    % display errors in each reading.
meanError = mean(errors)    % average error.

subplot(2,2,3);
plot(errors,'*');
ylabel('speed in km/hr');
xlabel('Error in each predictions');
refline([0 meanError]);
legend('error','average error in predictions');

% it is possible to have error since the reading from analog speedometer does not
accurately say the speed of travel

%====================================== Distance travelled
=========================================================%
% For calculating distance travelled,
% we assume that vehicle travelled straight in consecutive time gaps of 3 seconds.
timeInts = transpose(0:3:time(end));
theta = fnval(latFunction,timeInts); %temporary variable
phi = fnval(lonFunction,timeInts);   %temporary variable
[xTemp, yTemp, zTemp] = geodeticToCartesian(theta, phi, 0);
distance = 0;
for i=1:(size(xTemp)-1)    %calculate distance between two points considering as straight
lines
        dl = pdist([xTemp(i:i+1), yTemp(i:i+1), zTemp(i:i+1)]);
        distance= distance+dl;
end
distance = distance/1000  %convert to kilometers
clear('timeInts', 'theta', 'phi', 'xTemp', 'yTemp', 'zTemp', 'dl');    % free up memory

%====================================== Waiting time and travel time
==========================================%
% of course travel time is
travelTime = time(end);
```

```
disp ('Total time elapsed in HH:MM:SS id');
disp(datestr(travelTime/24/3600, 'HH:MM:SS'));

% for waitimg time, we assume if car is not moved by atleast 10 meters

% here is the idea, we take velocities at every second between 10 meters,
% We add it vectorically and see if is lesser than a threshold in magnitude.

timeInts = transpose(0:1:time(end));
theta = fnval(latFunction,timeInts); %temporary variable
phi = fnval(lonFunction,timeInts);   %temporary variable
thetaDot = fnval(latDotFunction,timeInts);  %temporary variable
phiDot = fnval(lonDotFunction, timeInts);   %temporary variable
[xVel, yVel, zVel] = getCartesianVelocity(theta, phi, thetaDot, phiDot);
[xTemp, yTemp, zTemp] = geodeticToCartesian(theta, phi, 0);
clear('theta', 'phiDot', 'thetaDot', 'phi'); % free up memory
for i=1:(size(xTemp)-1)    %calculate distance between two points considering as straight
lines
        tempDist(i,:) = pdist([xTemp(i:i+1), yTemp(i:i+1), zTemp(i:i+1)]);
end

j=1;
waitingTime = 0;   % initially zero
for i =1:size(timeInts)-1
        if (sum(tempDist(j:i,:))>4)    %wait till the vehicle travelled 4 meters
                % once the vehicle went forward by 4 mtrs,
    % sum all the velocity vectors and take its magnitude.
                % if the GPS gave different points in a circe, velocity vectors will add up to
small value
                % then we compare the magnitude to a threshold.
                meanVel = sqrt(sum([sum(xVel(j:i)), sum(yVel(j:i)), sum(zVel(j:i))].^2));
                if (meanVel < 4.2)
                        disp('hey')
                        waitingTime = waitingTime+timeInts(i)-timeInts(j);
                end
                j = i;
        end
end
%final waiting time
```

waitingTime