GMSU, task APP2 - GPS Track

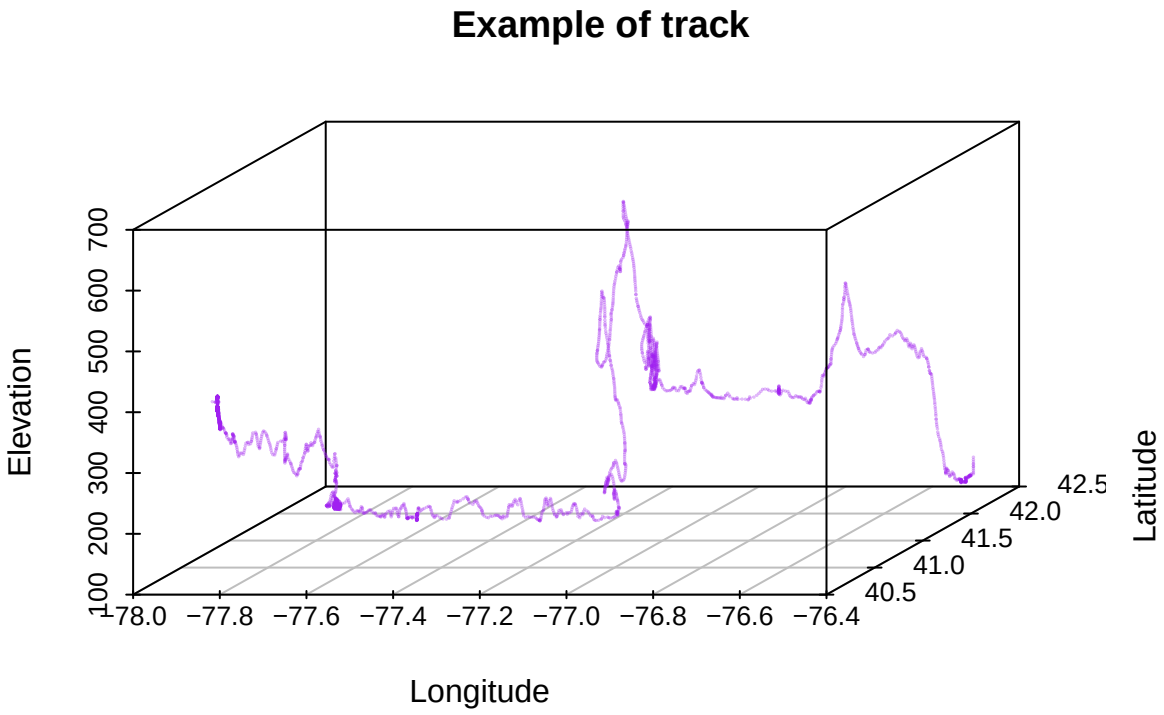
Askar Gafurov

June 23, 2017

Input data

Each data file covers one day of tracking (varying from 3 to 20 hours). Each file consists of several tracks (typically from 2 to 13), each track has several datapoints (total count of datapoints in one file varies from 1000 to 5000). Each datapoint has 4 properties: longitude, latitude, elevation and timestamp.

We decided to drop data about tracks, as for usually most of datapoints are stored in one track.



Task

Our task is to decide, for each datapoint, whether the user has been *driving a car*, *walking*, or *standing still*, using hidden Markov models.

Preprocessing of data

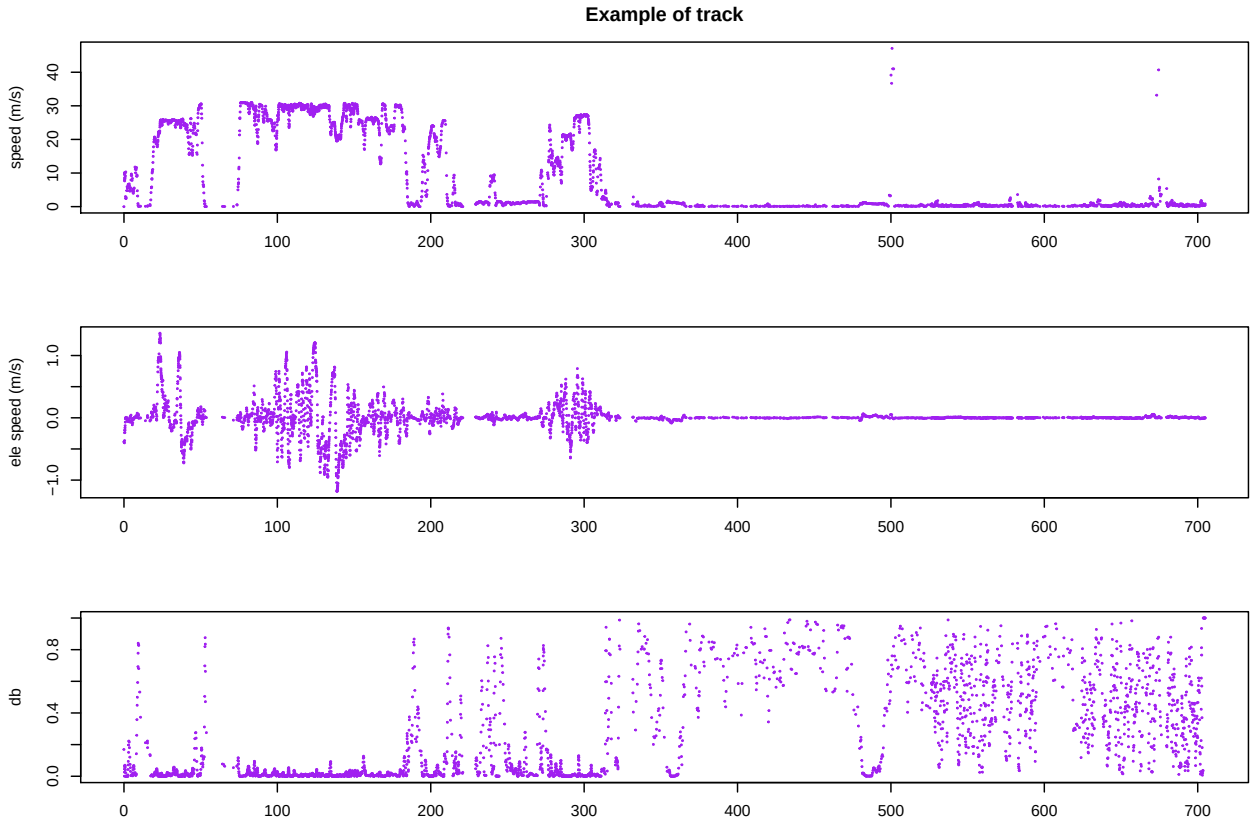
Main idea to detect the type of activity is by analysing the speed, difference in elevation, etc. So we have to evaluate the difference between two points. In order to decrease noise in the data, we have evaluated, for each datapoint, the difference with a 60 seconds lag. We have extracted several factors from the given data (we will call them "diffpoints" or "diffdata" for the rest of the report):

- `dist`: distance (in meters) between compared datapoints

- `bearing`: azimuth (in degrees, from 0 to 360), the orientation of a speed vector
- `dt`: time difference (in seconds) between compared datapoints
- `dele`: difference in elevation (in meters) between compared datapoints
- `db`: characterisation of irregularity of bearing (real number from 0 to 1; 1 - uniformly random changings in bearing, 0 - constant bearing)

Variables `dist` and `bearing` are evaluated using R package *geosphere* functions `distVincentyEllipsoid()` and `bearingRhumb`, respectively.

Variable `db` is evaluated in the following way: we pick bearings of k consequent diffpoints (in our work, we have used $k = 10$). The resulting variable is evaluated as a length of a sum of the picked bearings as vectors on a unit circle. For convenience, we will use $(1 - length)$ instead. It is easy to see that if the resulting value is close to 0, then the bearings must be close each other. The main purpose of this variable is to detect, whether the user had stopped.



HMM model

We have decided to use HMM with continuous observations (so called *continuous density HMM*).

After several models, we have settled with 5-state HMM:

1. walking [color: green]
2. car (*driving*) [color: darkred]
3. walk_stopped [color: blue]
4. car_stopped [color: purple]
5. walking up/down the hill [color: orange]

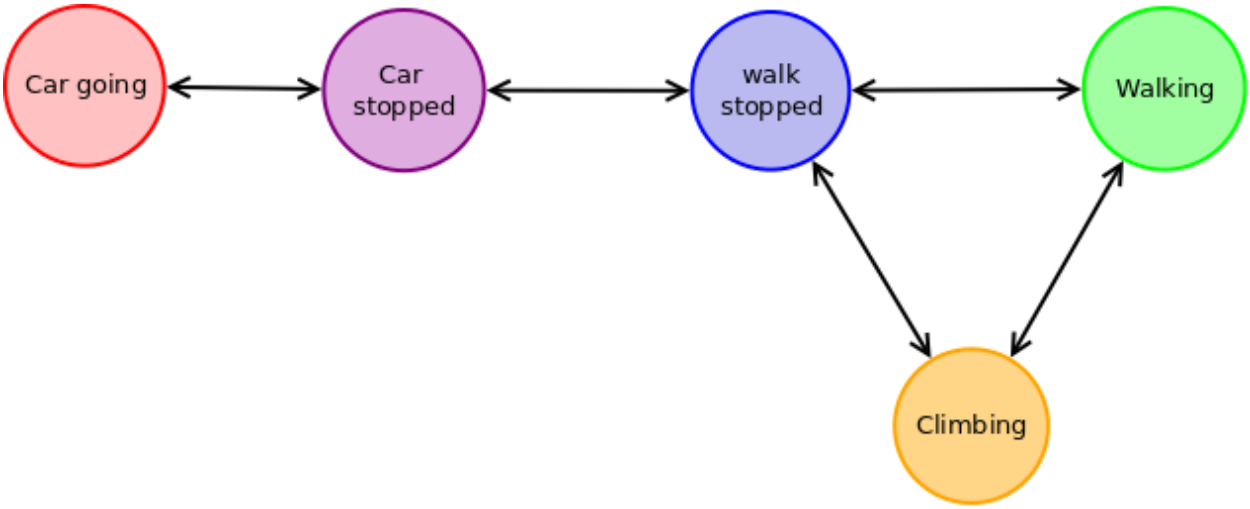


Figure 1: HMM

Table of transition probabilities

```
##           walk car walk_stop car_stop walk_ele
## walk      0.60 0.0      0.10     0.00      0.3
## car       0.00 0.7      0.00     0.30      0.0
## walk_stop 0.09 0.0      0.80     0.01      0.1
## car_stop  0.00 0.3      0.01     0.69      0.0
## walk_ele  0.30 0.0      0.10     0.00      0.6
```

The reason to have 2 separate states for stopping is that car can often stop for a few minutes and then continue to go.

Emission probabilities

Walk: low speed and moderate db

```
walk_e <- function(dato) {
    speed <- dato["dist"]/dato["dt"]
    db <- as.numeric(dato["db"] * 10)
    res <- as.numeric(dnorm(as.numeric(speed), mean = 1, sd = 1)/(1 - pnorm(0,
        mean = 1, sd = 1)) * dexp(db, rate = 0.5))
    return(res)
}
```

Car: high speed and low db

```
car_e <- function(dato) {
    speed <- as.numeric(dato["dist"]/dato["dt"])
    db <- as.numeric(dato["db"] * 10)
    res <- as.numeric(dunif(speed, min = 1, max = 60) * dexp(db, rate = 1))
    return(res)
}
```

Stop_walk and Stop_car are equal: low speed and high db

```
walk_stop_e <- function(dato) {
    speed <- as.numeric(dato["dist"]/dato["dt"])
    db <- as.numeric(dato["db"])
    ele_speed <- abs(as.numeric(dato["dele"]/dato["dt"]))
    res <- as.numeric(dexp(speed * 2, rate = 2) * dunif(db, min = 0, max = 2) *
        dexp(ele_speed * 10, rate = 3))
    return(res)
}
```

```
car_stop_e <- function(dato) {
    speed <- as.numeric(dato["dist"]/dato["dt"])
    db <- as.numeric(dato["db"])
    ele_speed <- abs(as.numeric(dato["dele"]/dato["dt"]))
    res <- as.numeric(dexp(speed * 2, rate = 2) * dunif(db, min = 0, max = 2) *
        dexp(ele_speed * 10, rate = 3))
    return(res)
}
```

Walk ele: is similar to stop, but allows greater elevation speed (to distinguish between stop and climbing)

```
walk_ele_e <- function(dato) {
    speed <- as.numeric(dato["dist"]/dato["dt"])
    db <- as.numeric(dato["db"])
    ele_speed <- abs(as.numeric(dato["dele"]/dato["dt"]))
    res <- as.numeric(dexp(speed * 2, rate = 2) * dunif(db, min = 0, max = 2) *
        dunif(ele_speed, min = 0.2, max = 2))
    return(res)
}
```

Inferring of hidden states

We have used traditional Viterbi algorithm (no additional library was used). The only modification was that we had to use functions to determine the emission probabilities instead of tables.

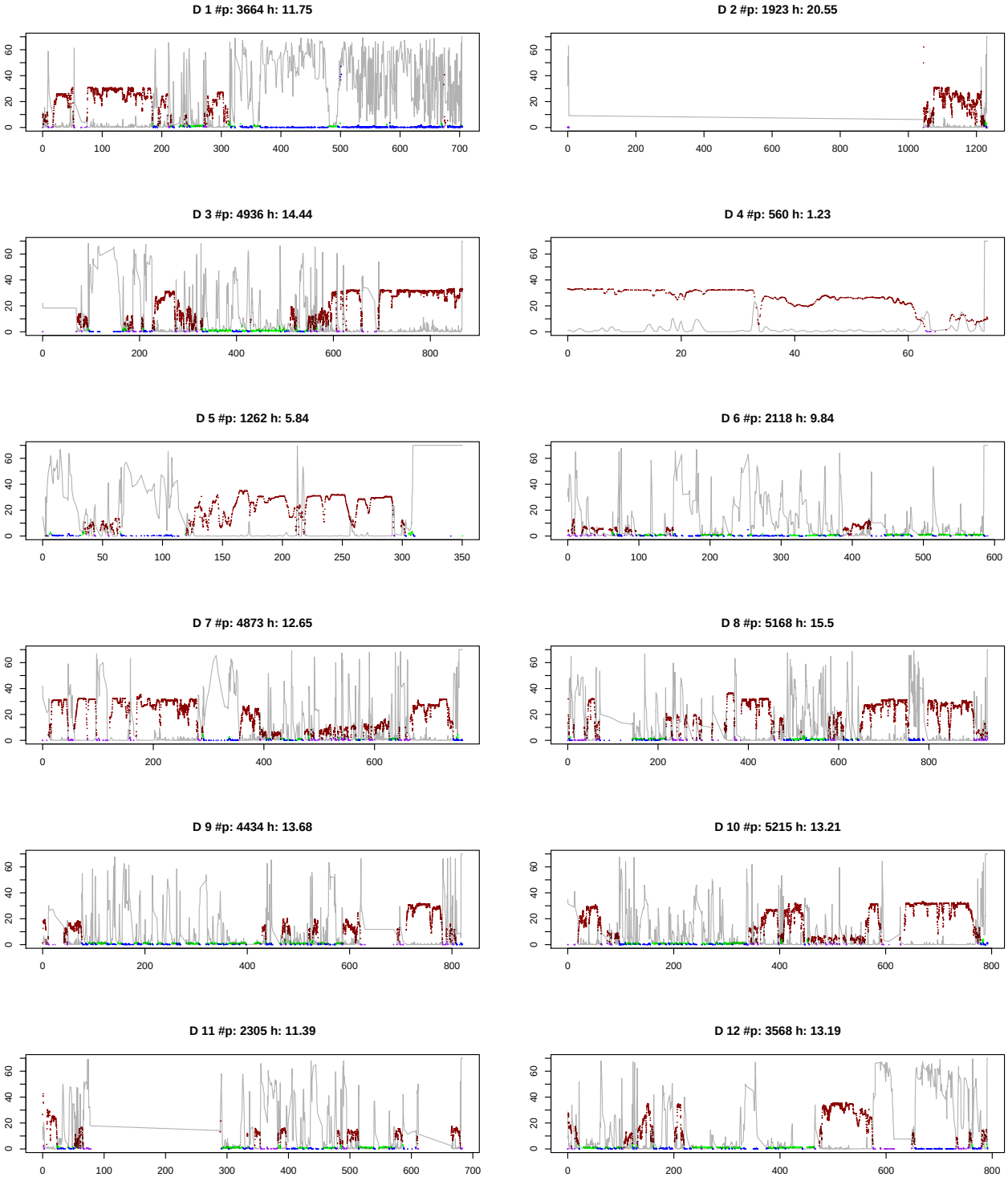
```
get_viterbi <- function(transition_pp, emission_pp_f, init_prob, data, EPS = 10^(-6)) {
    if (is.na(data)[1])
        return(NA)
    states <- rownames(transition_pp)
    S <- length(states)
    N <- dim(data)[1]
    # prob of the best path so far
    T1 <- matrix(0, nrow = S, ncol = N)
    # tail of the best path so far
    T2 <- matrix(0, nrow = S, ncol = N)

    for (i in 1:S) {
        T1[i, 1] <- log(init_prob[[states[i]]] + EPS)
        +log(emission_pp_f[[states[i]]](data[1, ]) + EPS)
        T2[i, 1] <- 0
    }

    for (i in 2:N) {
        for (j in 1:S) {
            poss <- T1[, i - 1] + log(transition_pp[, j] + EPS)
```

```
            best <- which.max(poss)
            T1[j, i] <- log(emission_pp_f[[states[j]]](data[i, ]) + EPS) + poss[best]
            T2[j, i] <- best
        }
    }

    path <- vector(length = N)
    path[N] <- which.max(T1[, N])
    for (i in seq(N - 1, 1, by = -1)) {
        path[i] <- T2[path[i + 1], i + 1]
    }

    return(list(path = path, T1 = T1, T2 = T2))
}
```

Results

First visualisation has two variables: speed (=`dist`/`dt`) (dots with color, indicating determined state) and `db` (grey line). Expected behaviour is:

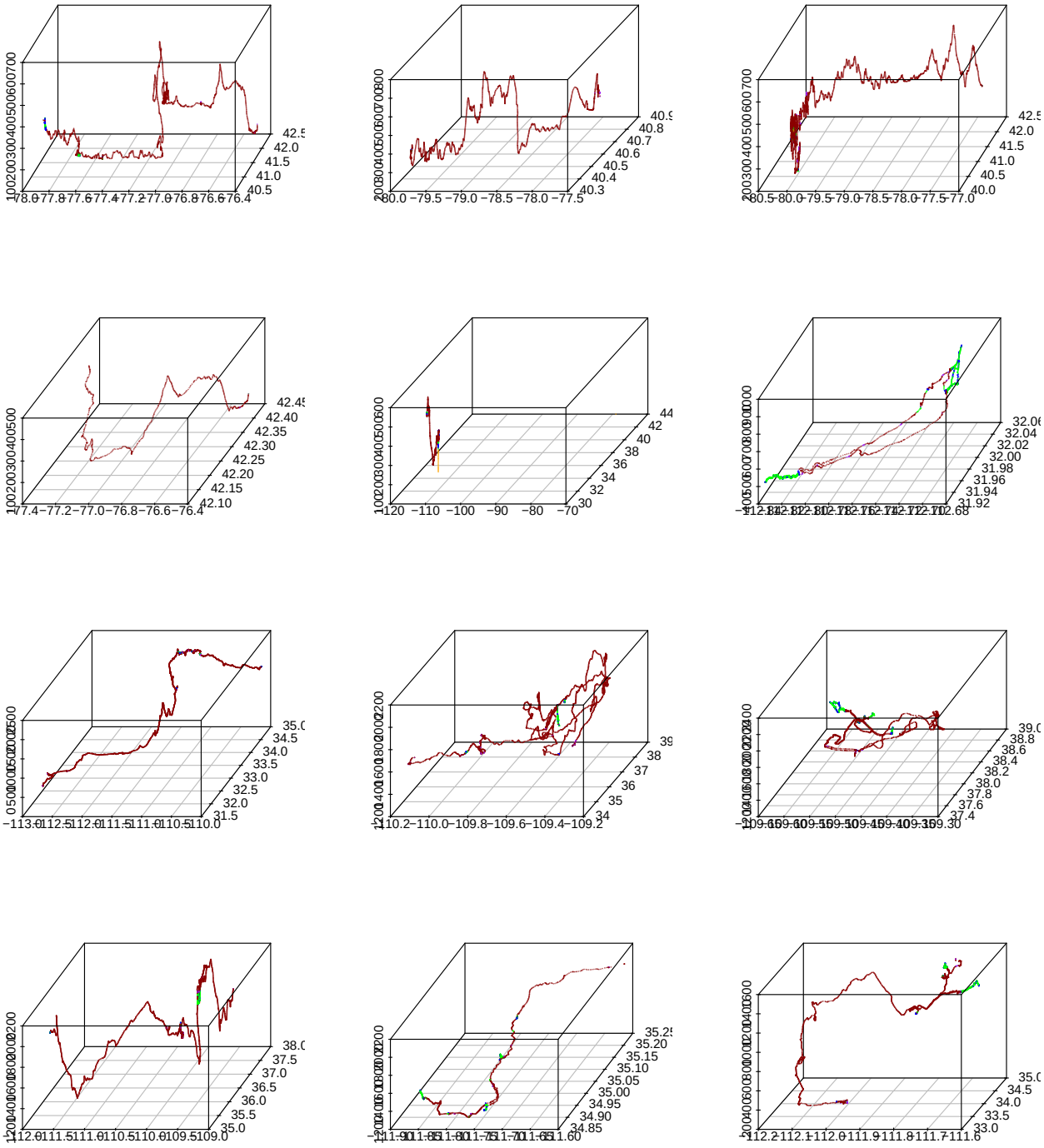
- high speed + low `db` -> car [darkred]
- low speed + moderate `db` -> walking [green]
- low speed + high `db` -> stopped [blue]



3D mapa (X - longitude, Y - latitude, Z - elevation)

Expected behaviour is:

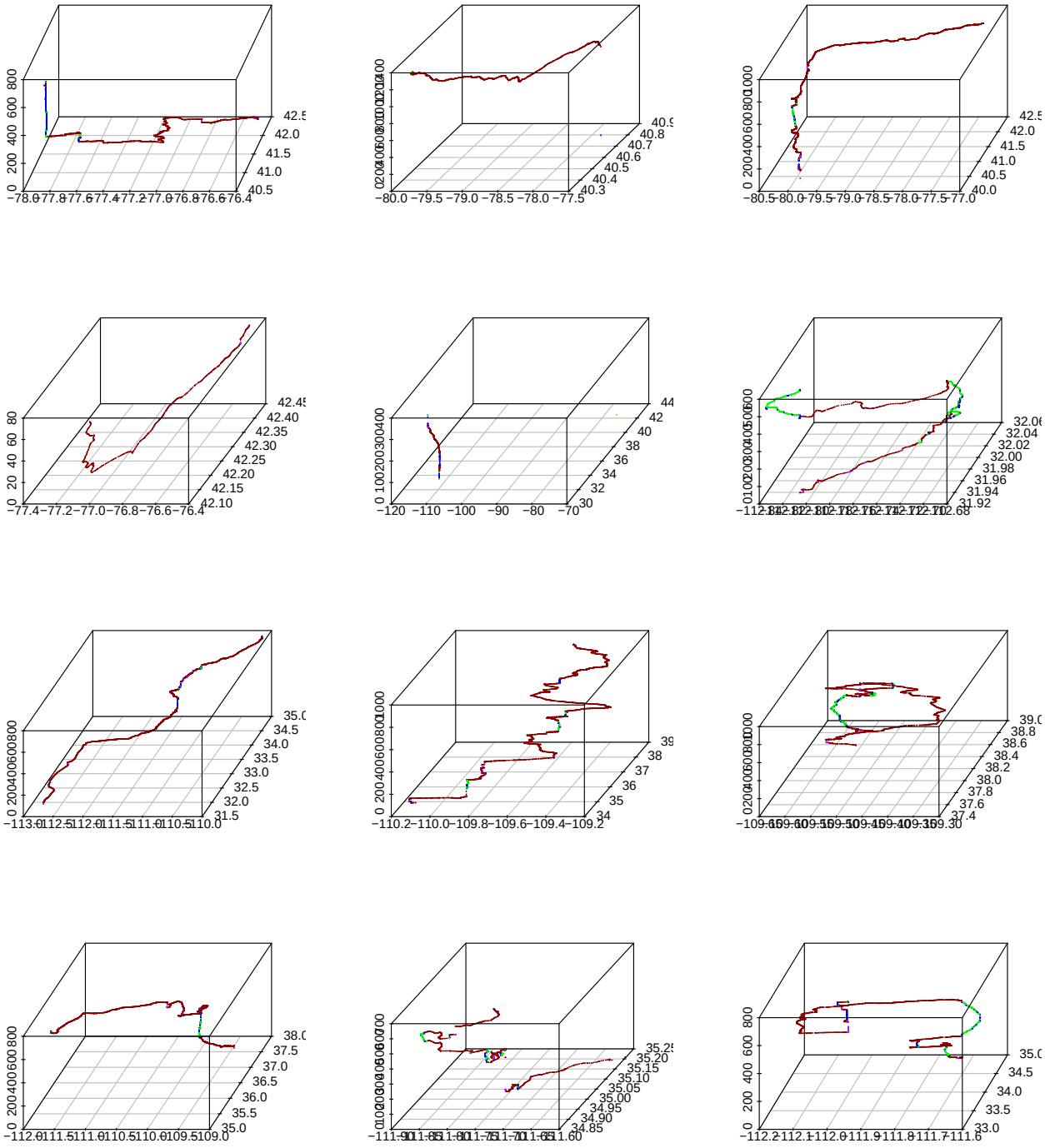
- no blue or purple lines (as for these colors stands for “stopped”) and not so much of green (because it is hard to cover larger distancies on foot)



2D mapa + Z=time

Expected behaviour is:

- vertical blue or purple lines (stopped for a while)
- darkred lines are mostly horisontal (because of high speed of a car)
- green lines are not so horisontal, and indivudial lines are arcs (both ends share X and Y coordinates) = walk from a car and then back



Conclusions

All visualisations are consistent with our expectations, so we can claim we that our model is successful at the given task :)

Code

We have used a small Python script to convert `.gpx` to `.csv` and the rest of computations were done in R. This report has all the code included (in respective `.Rmd` file). For earlier versions of model, one can check files `hmm_v*.{Rmd|html}`. For initial analysis of the data one can check `first_analysis.{Rmd|html}`. It is recommended to view these files by RStudio (install packages `scales`, `scatterplot3d`, and `geosphere`).

All materials are available at https://github.com/japdlsd/gps-track-analysis.