02637 Advanced Matlab Programming

Project 1: GPS Data Processing

Katarzyna Chyzynska (s091825)
Andrei Cibu (s111445)

Software: MATLAB R2012b

The professional and amateur runners often use the GPS receivers to monitor their running progress. The data from such receivers are stored as numerical descriptors of the route passed, namely the latitude, longitude and time. The following report constitutes description of the GPS Data Analyzer software, developed in Matlab GUI environment. The program is capable of manipulating such GPS data and computing key performance statistics for the user-specified route. Thus the user has possibility to load his data to the program, choose subset of data to analyze, simulate the covered route or extract key performance information.

Modules and functions

This section contains a description of the modules and functions that build up the GPS Data Processing tool. We will group the functions in such a way that their combined effect covers the tasks mentioned in the project description.

Task 1. Read data from a file.

readInputData

```
function [x, y, z, time, lat, lon] = readInputData()
% READINPUTDATA Reads data from a file
%   readInputData() reads the data from a file. The file is to be selected
%   via a standard dialog box. The accepted file types are *.txt, *.mat,
%   *.csv. The input files must have the following predefined structures:
%       *.csv & *.txt: 3 values per row, representing latitude, longitude
%       and time values.
%       *.mat files: containing the (x, y, z, time) or (lat, lon, time)
%       variables.
%
%   If the input data is in the latitude, longitude this function converts
%   it into x, y, z coordinates.
```

The function reads the GPS data from *.txt, *.mat or *.csv file formats. It displays a standard dialog box, where the user can select the input file. In case of *.csv or *.txt files, the data must be stored as three values per line representing latitude, longitude and time values. As for the *.mat files, they can contain either of the two sets of column shaped variables: (x, y, z, time) or (lat, lon, time). This function is also responsible to trigger the conversion from polar coordinates (i.e. latitude, longitude) to Cartesian coordinates if necessary.

Note: if the Cartesian coordinates (i.e. x, y, z) are read from a *.mat file it is required that they are expressed in [m]. Likewise, the time should be expressed in [s].

Task 2. Reformat data

reformatData

```
function [x,y,z] = reformatData(lat, lon)
% REFORMATDATA Reformat data to Cartesian coordinates
%   reformatData(lat, lon) transforms lat, lon coordinates into cartesian
%   coordinates by considering the altitude value equal to 0.
```

The function converts the data from polar to spatial coordinates.

Task 3. Save data.

saveData

```
function [ ] = saveData(varargin)
% SAVEDATA Save data to file.
%   saveData(varargin) saves data to a *.mat file. The data to be saved
%   defines a trajectory in (latitude, longitude, time) or (x, y, z, time)
%   formats.
```

This function saves data to a *.mat file. The accepted number of input arguments is either 3 or 4. In case the number of input arguments is 3, it is assumed that they are latitude, longitude and time, otherwise they should be x, y, z and time. The order of the input arguments should comply with the previous enumerations.

Task 4. Interpolate the data points.

interpolateData

```
function [xi, yi, zi, ti] = interpolateData(x, y, z, time, n, method)
% INTERPOLATEDATA Interpolates the route points.
%      interpolateData(x, y, z, time, n, method) returns the interpolated
%      trajectory given an initial trajectory. x, y, z and time define the
%      initial trajectory. n is the number of interpolation points. method
%      defines the method for interpolation (accepted values 'spline' or
%      'pchip').
```

This function returns a set of interpolated points. It supports ‘spline’ or ‘pchip’ interpolation methods, which are selectable via the method input argument. The number of interpolation points is given by n.

Task 5. Compute key descriptive values.

getInstantKeyValues

```
function [instSpeed, instPace, cumDist] = getInstantKeyValues(x, y, z, time)
```

```
%GETINSTANTKEYVALUES Get instant key timeseries.
%   getInstantKeyValues(x, y, z, time) returns the time series containing
%   instant values for speed, pace and cumulated distance. x, y, z and time
%   define the trajectory of the movement.
%   NOTE: expected units for x, y, z: [m]
%         expected units for time: [s]
```

The function returns time series for instantaneous speed, instantaneous pace and covered distance for the GPS route. It is meant to optimize the computation of the aggregated values such as total time, total distance of the route, average speed and average pace. It is enough to compute the instantaneous values once after the GPS data are loaded, and then use this values to calculate the aggregated key values of a selection of the route.

getAggKeyValues

```
function [tdist, ttime, avgspeed, avgpace] = getAggKeyValues(instSpeed, instPace,
cumDist, time)
%GETAGGKEYVALUES Get aggregated key values.
%   getAggKeyValues(instSpeed, instPace, cumDist, time) returns the
%   aggregated values for distance (total), time (total), speed
%   (average), pace (average) given the instantaneous values.
%
%   The length of the instantaneous vectors should be greater than 1,
%   otherwise the function will return empty sets for the aggregated
%   values.
```

The function calculates total distance, total time, average speed and average pace for the specified route. It takes as input the instantaneous speed, pace and distance time series. If we want to calculate the aggregated values for a partial route, the time series should be tailored to cover only that selection. It is required that the time series contain more than one value.

Task 6. Plot the data points and the interpolated curves.

plotPath

```
function [hroute, hPartRoute, hinterp, hp1, hp2] = plotPath(h, x, y, xi, yi)
%PLOTPATH Plot GPS path/route.
%   plotPath(h, x, y, xi, yi) plots, on the axes with handle h, the
%   trajectories defined by x, y (original) and xi, yi (interpolated).
%   It also creates two markers (impoints) placed at the beginning and at the
%   end of the interpolated trajectory.
%   Output arguments:
%       hroute: handle to the lineseries defined by the original trajectory
%       hPartRoute: handle to the lineseries defined by the points between
%       the start and end markers
%       hinterp: handle to the lineserios defined by the points on the
%       interpolated trajectory
%       hp1: handle to the start marker
```

```
%       hp2: handle to the end marker
```

The function plots the specified Cartesian coordinates of the route, the interpolated trajectory and creates two draggable points used to specify the partial route.
The output arguments are the handles to the original route, partial route, interpolated route and start and end points of the partial route. The cursors are placed at the beginning and at the end of the interpolated trajectory. The start cursor is colored red and is attached an 'S' string, whereas the end cursor is colored blue and is attached an 'F' string.

refreshGPSRoute

```
function refreshGPSRoute(hObject, handles)
%   This function recalculates all the internal data, assuming that the GPS
%   data changed. It also fills the GRP Route plot with the appropriate
%   data.
```

This function is called every time the GPS data is changed. It coordinates the interpolation of the new data, the calculation of the instantaneous key values and the plotting of the GPS route.

Task 7. Plot key values as a function of time.

plotKeyValues

```
function [hVelPlot, hPacePlot, hDistancePlot] = plotKeyValues(h, instSpeed, instPace,
cumDist, time)
%PLOTKEYVLUES Plots velocity, pace and distance as a function of time
%   plotKeyValues(h, instSpeed, instPace, cumDist, time) plots the
%   instantaneous values for speed, pace and cumulated distance in the
%   uipanel/figure with handle h. It returns handles to the plots
%   corresponding to each of the instantaneous series.
```

The function plots the instantaneous velocity, instantaneous pace and the current total covered distance as functions of time in the three subplots. The output constitutes the handles to the respective subplots.

refreshKeyValues

```
function refreshKeyValues()
%   This function triggers the calculation of the aggregated key values. It
%   fills then the appropriate text fields with the aggregated values. It
%   also refreshes the key values plots.
```

Task 8. Partial data set chooser.

As mentioned previously, ***plotPath*** creates two cursors that the user can manipulate. They are

impoint objects which we constrain to move only on the interpolated trajectory.

constrainStartCursorPos

```
function newpos = constrainStartCursorPos(pos)
%   This function imposes the start cursor to move only on the interpolated
%   trajectory. Additionally, if a simulation is running, the user cannot
%   move the cursor.
```

This function is called whenever the start cursor moves. It is in its responsibility to make sure that the new position of the cursor is on the GPS interpolated trajectory. Additionally, it does not allow for the start cursor to be moved while in simulation mode.

constrainEndCursorPos

```
function newpos = constrainEndCursorPos(pos)
%   This function imposes the start cursor to move only on the interpolated
%   trajectory.
```

This function is called whenever the start cursor moves. It is in its responsibility to make sure that the new position of the cursor is on the GPS interpolated trajectory.

startCursorMoved_Callback

```
function startCursorMoved_Callback(pos)
%   This function triggers the refresh of the key values after the start
%   cursor has moved.
```

This function is called after the start cursor has moved. It triggers the refresh of the key values (both instantaneous series and aggregated values).

endCursorMoved_Callback

```
function endCursorMoved_Callback(pos)
%   This function triggers the refresh of the key values after the end
%   cursor has moved.
```

This function is called after the end cursor has moved. It triggers the refresh of the key values (both instantaneous series and aggregated values).

getClosestPoint

```
function [xo, yo] = getClosestPoint(x, y, xi, yi)
%GETCLOSESTPOINT Get closest point.
%   getClosestPoint(x, y, xi, yi) returns the point from vector [xi, yi],
%   defined by [xo, yo],  which is closest to point to [x, y]
```

The function is triggered by the constraining functions after any of the cursors has moved. It returns the point on the interpolated route which is closest to the cursor. The return value of this function will be the new position of the cursor.

Task 9. Export plots to .pdf or .jpg images.

exportButton_Callback

```
function exportButton_Callback(hObject, eventdata, handles)
% hObject    handle to exportButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

This function saves the figures in either *.pdf or *.jpg formats. We have chosen to implement this function inline as it is specific to this gui. Its input arguments would have made no sense in a generic case.

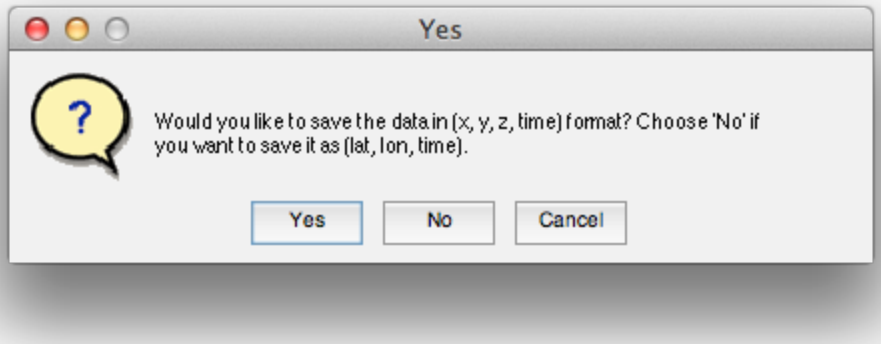
GUI elements

Load data …

Push button: It triggers the reading of the input data. After the data is read, it is stored in the gui's built in user data structures. We use this data structure to pass data between function calls. If the input file complies with the required formats and valid data is loaded, the loadButton's callback enables all the user accessible uicontrols. It triggers then refreshGPSRoute to populate the gui display elements with data.

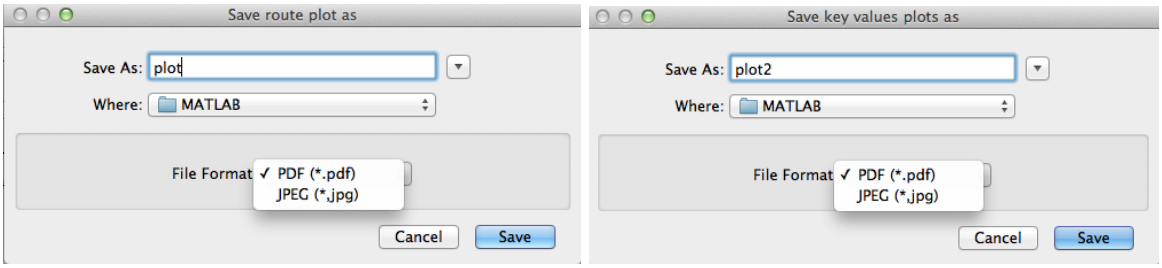
Save data …

Push button: It displays a dialog box in which the user is asked with regard to the desired format in which the *.mat file should be saved. According to the user's choice the appropriate format is selected and the output data is saved.



Export plots …

Push button: It exports the plots to *.pdf or *.jpg file. It generates two files, one for each plot type: route plot and key values. The user is requested to select output file type for each plot type via standard dialog boxes.



Interpolation *panel*

Pop-up menu: A selection change triggers refreshGPSRoute to refresh data and repopulate the gui with data.

Edit text: Constitutes input for the number of interpolation points. Right after new data is loaded in the gui, it is filled out with the number of points in the GPS trajectory. This number will be the minimum threshold for the subsequent values. It makes no sense to allow a number less than the original number of points. Therefore, the callback function for this field validates that its content is a number and that it is greater than the minimum, otherwise error messages will be displayed.

Replay *panel*

Simulate selection.Toggle button: It is used to start/stop the route simulation. The end cursor will start to cover the entire partial route. For an x1 speed-up selection it should move with the instantaneous speed calculated for the dataset. Speed-up selections of x2 or x3 will make the end cursor move faster. While the 'Simulate selection' button is pressed, therefore the simulation is running, all key values are being refreshed in their corresponding display elements. When the user toggles the button to its unpressed state, the simulation ends and the cursors will be set back to the positions they had before the simulation started.

Speed up. Panel: Contains three radio buttons designed to control the speed of the simulation (x1, x2 and x3).

Zoom/Pan

Toggle buttons: Enable the zoom and pan modes for the GPS Route plot. They are mutually exclusive. While in either of the two modes, the user cannot move the cursors.

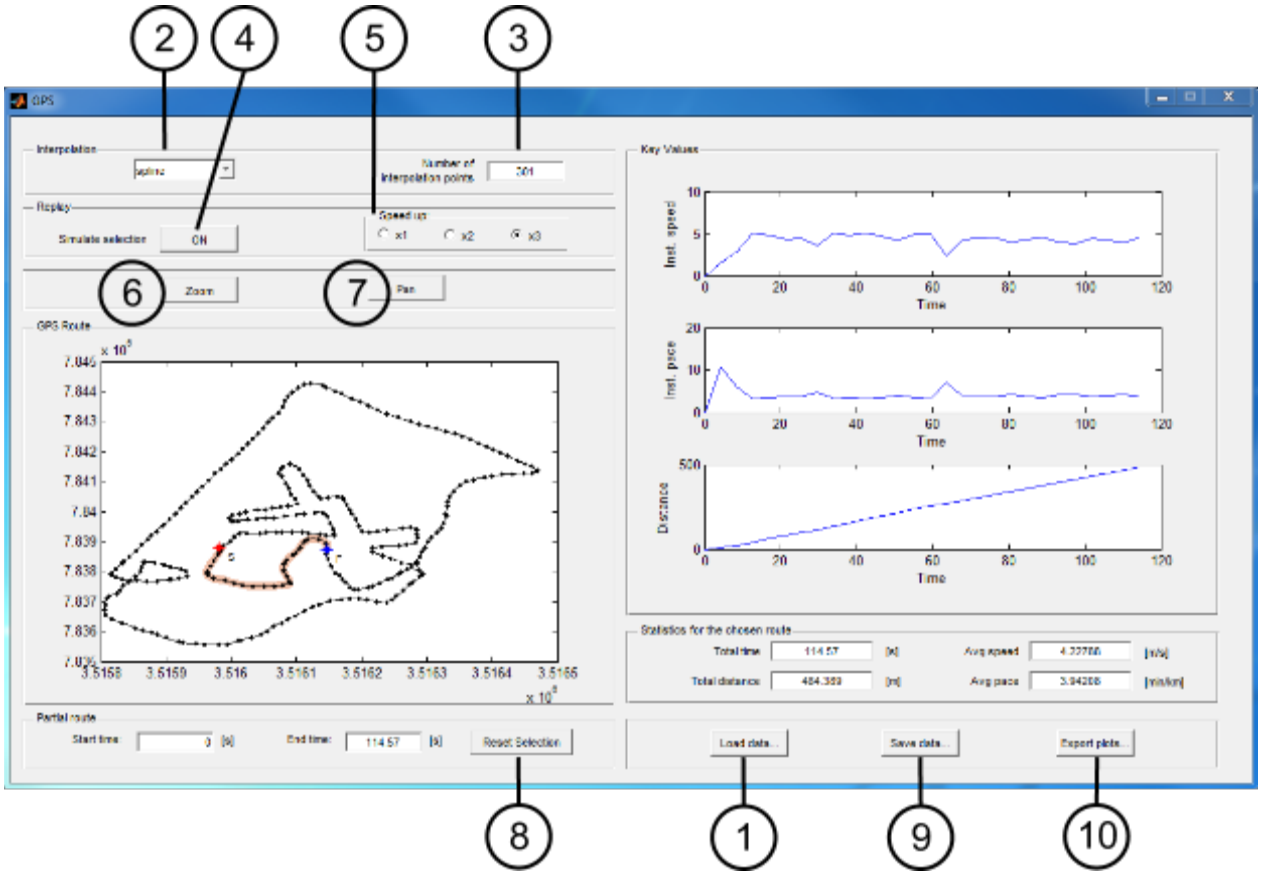
GPS Route *panel*

Panel: Accommodates the axes on which the trajectory is going to the plotted.

Reset Selection
Push button: The selection cursors move at the beginning and the end of the interpolated trajectory.

Key Values *panel*
Panel: Accommodates three subplots with time series of instantaneous key values.

GPS Data Analyzer Manual



1. Load the GPS data into the program (allowed formats: *.txt, *.csv, *.mat). The format of the data has to be either latitude, longitude, time in three columns or x, y, z (Cartesian coordinates), time in four columns (the latter can be saved as in point 9).

2. Choose the route interpolation type: *spline* or *pchip* (default *spline*).

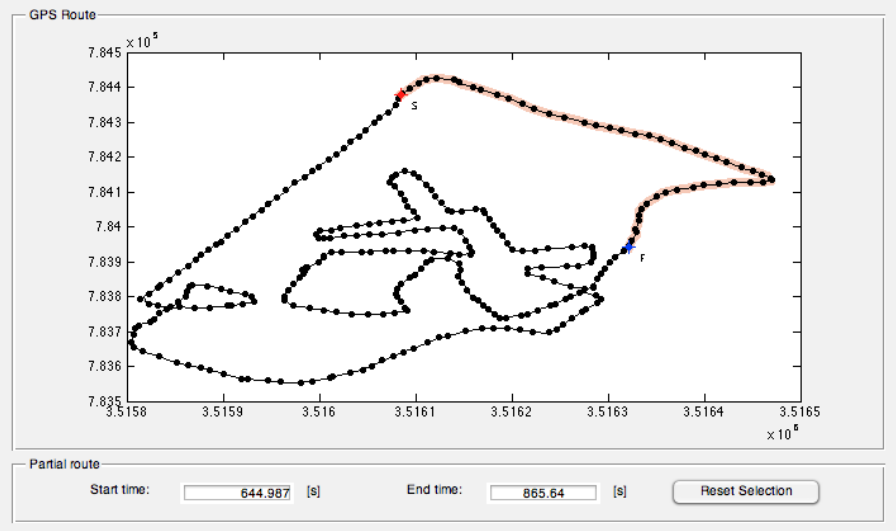
3. Choose the number of interpolation points. Optional. (by default the number of interpolation points is set to be equal to the number of points in the GPS data). The value of this field can only be modified to a value greater or equal than the default value.

4. Start/stop the route simulation. It is recommended to check that the cursors delimit the partial route of interest (highlighted in light brown).

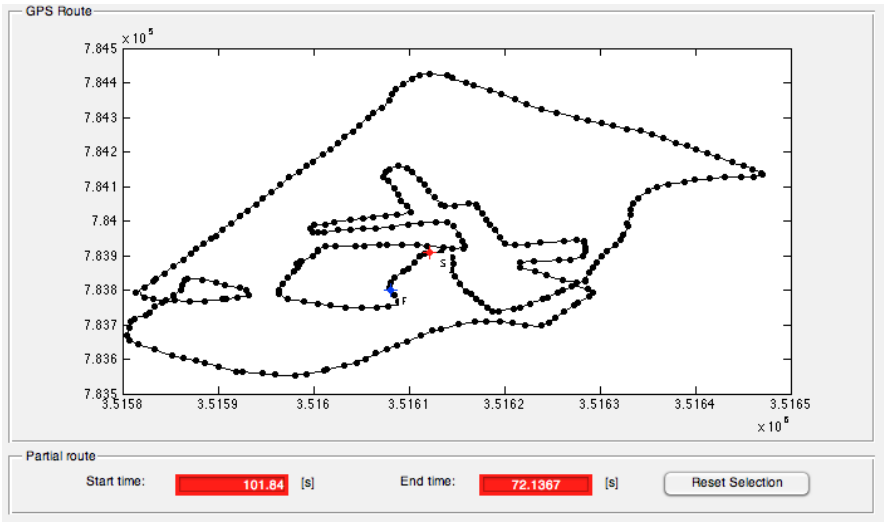
5. Choose the speed of simulation.

6. Zoom button - enable zoom mode.

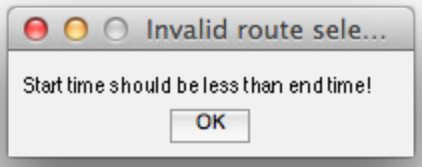
7. Pan button - enable pan mode.
The user may choose the partial route by dragging the *start* (red) and *stop* (blue) cursors on the route plot.



An invalid partial route, defined by the start cursor placed after, or on the same position with the end cursor is notified by red colored Start time and End time text fields.



Also, if the user wants to simulate an invalid partial route, they will get an error message and the simulation will not start.



The simulation may be stopped at any time by unpressing the *ON* button, or it will stop automatically when the end cursor covers the entire partial route.

The statistics plots and values on the right side of the window are updated in online as the simulation proceeds.

8. Reset partial route selection - Resets the partial route to the entire GPS route.

9. Save the GPS data. When clicked, the user will be asked if he/she wants to save the data as original format (latitude, longitude, time), if not, the data will be saved in x, y, z, time - format as the .mat file.

10. Exports the plots. The user will be prompted to choose the format and name for the exported plots (two figures are saved, one with route plot and the other with the performance plots).

Conclusions

This project offered us a deep insight into the process of GUI application development in Matlab. The application could be further improved by inserting additional functionality such as 3D plotting or analyzing other kind of running-related collected data, i.e. heart rate or pulse.