# Visualizing Mobile Phone Sensors Data in an R Environment

Riccardo Miccini

Technical University of Denmark - DTU

January 17, 2017

**Abstract**

This document will cover the development of an infrastructure for collecting and visualizing geolocalization data from mobile devices.

The project has been carried out under the supervision of profs. John Aasted Sørensen and Ian Bridgwood, as part of a multidisciplinary project.

The content hereby presented follows this scheme: an introductory section containing the problem formulation and an overview of the resulting design, an analysis of the necessary requirements and tools, and comprehensive description of the implementation details. Conclusions will be drawn, assessing the overall results and the newly gained experience and skills.

*All brands, product names, logos, or other trademarks featured or referred to in this document are the property of their respective holders, and their use does not imply endorsement.*

# Contents

# Chapter 1

# Introduction

This introductory chapter will provide a description of the project in the form of assignment formulation, and a brief overview of the implementation, which will be elaborated further in the following chapters.

## 1.1   Problem formulation

The aim of the project is the application of methods for integrating mobile phone (Android, iPhone) sensor measurements and an R data visualization environment using the Google Cloud as buffer.

## 1.2   System description

The implemented system is composed of three main elements: a series of mobile devices, a remote server, and a user station.

The former are equipped with a custom-made application capable of transmitting geolocalization data to a remote destination.

The remote destination is represented by a *Google Sheet* document where such data is stored. This spreadsheet acts as a database for the collected data, and is accessible through the cloud.

The user can then query the server for the necessary data using the provided R scripts and an R environment such as *RStudio*.

# Chapter 2

# Analysis

The following sections aim to elaborate on the Problem formulation and create a more solid base to use as a guideline and template during the actual development. This analysis does not aspire to be exhaustive, but rather satisfactory in the level of detail required to establish the project.

## 2.1 Milestone plan

Here is the project milestone plan as formulated during the initial phases of the project. This version of the plan lists a series of achievements deemed necessary to accomplish the major functionalities. It is arranged in a progressive way, with a time axis roughly flowing vertically. Each task depends from its sub-tasks and — more indirectly — on the previous ones within its hierarchal level.

- Extract requirements and use cases

- Evaluate and choose remote hosting service

- Familiarize with necessary tools and languages

    - R and RStudio

    - Java and Android Studio

        - Acquire device location
        - Read and append content to the remote platform

- Software development

    - Configure remote service authorizations and keys

- Implement Android application
- Implement R script for visualizing data

- Integration and validation tests

- Compiling necessary documentation

## 2.2 Requirements

The *software requirements specification* is the formal description of the system to be developed, thoroughly pointing out what the product is — and is not — expected to do. It also helps assessing the extent of the endeavor in terms of workload, costs, and other resources. The requirements are commonly laid out in agreement with the client, and provide the basis upon which the product or project is evaluated. Following the *Unified Software Development Process* — upon which this analysis is loosely based — the requirements can be classified in *functional* and *non-functional*.

As far as this project is concerned, the supervisors have not set any particular requirements apart from those implied in the project name and problem formulation. The list of requirements is therefore mostly based on former knowledge, common sense and best practices within the industry.

### 2.2.1 Functional requirements

This section defines specific behaviors or functionalities, and is the basis for the tests. Requirements introduced by the modal verb *shall* indicate mandatory conditions, whilst those introduced by the modal verb *may* indicate optional features.

| ID | Description |
|---|---|
| SRS.F.1 | The mobile apps shall submit their ID, timestamp, and current location at a given interval |
| SRS.F.2 | The location provided by the mobile apps shall be the most precise available |
| SRS.F.3 | The submitted mobile data shall be stored in a *Google Sheets* document |
| SRS.F.4 | The location shall be expressed in geographical coordinates (latitude and longitude) |
| SRS.F.5 | The device IDs shall be unique |
| SRS.F.6 | The submitted timestamps shall be relative to UTC-0 |
| SRS.F.7 | The front-end application shall visualize the current position of the tracked devices |
| SRS.F.8 | The front-end application shall update with new data at a given interval |

Table 2.1: Software requirements specification: functional requirements

### 2.2.2 Non functional requirements

This section specifies desired characteristics and qualities of the system.

| ID | Description |
| --- | --- |
| SRS.NF.1 | The Android app code shall be written in Java |
| SRS.NF.2 | The Android app code shall employ *Android Studio* build system |
| SRS.NF.3 | The iOS app code shall be written in Swift |
| SRS.NF.4 | The front-end application shall be written in R |
| SRS.NF.5 | The front-end application should limit its data requests to the bare necessary |
| SRS.NF.6 | All source code shall follow a consistent format style |
| SRS.NF.7 | All source code shall be properly documented |
| SRS.NF.8 | All source code should be trivially buildable and executable on other platforms |
| SRS.NF.9 | The released documentation shall be written in English |

Table 2.2: Software requirements specification: non-functional requirements

## 2.3 Tools

This section will cover various software tools that have been employed during the course of the project. These include development tool and software components or libraries used by the implemented code.

### 2.3.1 Android Studio

Android Studio is the official IDE (integrated development environment) for native Android development. It is distributed freely by Google for Windows, Mac OS X and Linux platforms, and it is based on emphJetBrains' IntelliJ IDEA, a proprietary IDE for Java. It replaced emphEclipse Android Development Tools as Google's primary IDE for Android application development.

The IDE provides a series of Android-specific tools and features. The most notable ones are:

- Code editor with intelligent completion, linter, and refactoring system

- Integrated debugger and emulator based on virtual machines

- Graphical layout editor and wizards for code generation of UIs and other common software components
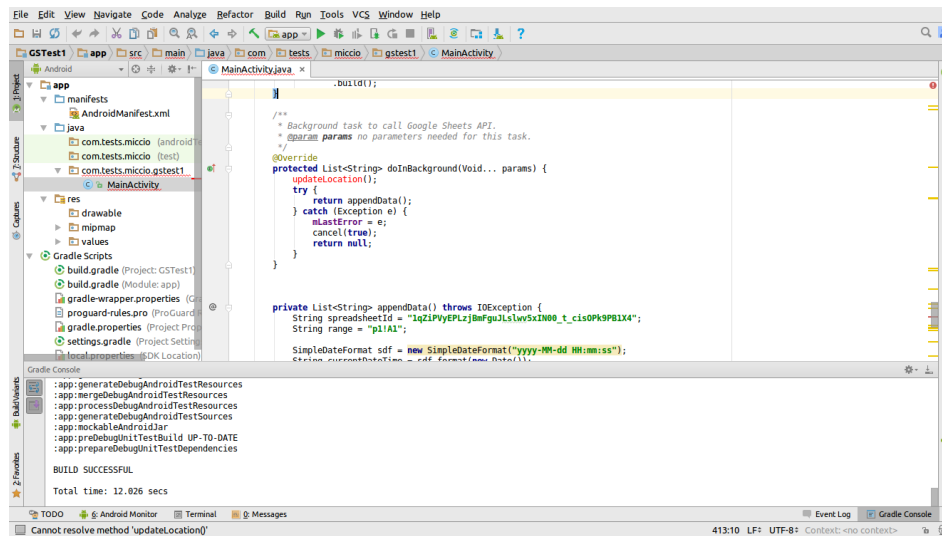
Figure 2.1: Android Studio: example view with text editor, project navigator, and build console

### 2.3.2 RStudio

RStudio is a free and open-source IDE for R, a programming language for statistical computing. Several editions of the software exist: a emphDesktop one, available on Windows, OS X, and Linux, and a emphServer and emphServerPro one, with allows access through web browser from several terminals.

The software comprises a text editor with code completion and syntax highlighting, an interactive command interpreter with built-in debug, command history, and data viewer, as well as a package manager and a documentation browser.
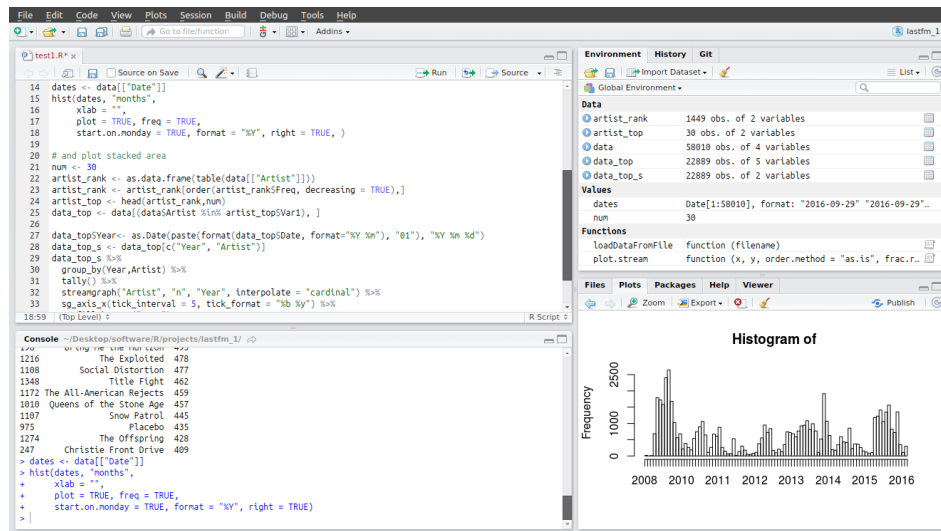
Figure 2.2: RStudio: example view with text editor, console, list of currently loaded data, and plots

### 2.3.3 R packages

The capabilities of R can be extended through a system of packages. Such packages may provide alternative graphics tools, additional statistical and data handling functions, or APIs and bindings for other services and software. R packages can be developed in R, C, C++, and Fortran, and are normally distributed through the *CRAN* (Comprehensive R Archive Network).

Packages that are not part of the core implementation have to be downloaded and loaded into the environment prior to their use. This can be done through the R console with the `install.packages()` and `library()` commands, or in RStudio's own package manager. Unofficial resources can also be obtained through GitHub or other hosting platforms.

Here follows a list and description of the R packages that will be employed for the fulfillment of the project:

**shiny** A framework for building live web applications in R. It comprises control widgets and graphic outputs, and uses a reactive model for determining which parts of the pages needs updating.

**leaflet** Integration with *leaflet*, an open-source javascript library for interactive maps. Maps can be enhanced with custom graphics such as polygons, lines, and markers.

**googlesheets** A wrapper for the *Google Sheets* APIs. It can be used to perform common file managing operations, as well as for accessing and editing data in a worksheet.

**dplyr** This package provides a set of useful and functions for manipulating data, such as `select`, `filter`, and `arrange`, with a particular focus on performances. It also provides the pipe operator `%>%`, which can be used to concatenate operations on datasets of various nature.

### 2.3.4 Other tools

In order to be able to contribute from any given location (work PC, personal laptop...), the project source code has been hosted on a web-based `git` repository hosting service called *GitHub*.

 `git` is a version control system for collaborative development of projects. It tracks changes in local files and allows them to be synchronized across multiple machines and remote hosts. Within the purpose of this project, `git` has been used to provide a means of accessing and safely storing the code base. Ordinary maintenance of the repository is performed from command-line, with the aid of graphical `diff`-tool *Meld* for code revision.

 Whilst the powerful code completion of the aforementioned IDEs is useful for general development tasks, external text editors such as *Atom* have been used as well, especially when large amount of code had to be refactored or restructured. In particular, the project documentation has been authored using *Atom* and several plugins for LaTeX integration.

# Chapter 3

# Design

## 3.1 Infrastructure

## 3.2 iOS

## 3.3 Android

## 3.4 R

# Chapter 4

# Testing

# Chapter 5

# Conclusions

## 5.1 Product assessment

## 5.2 Process assessment

## 5.3 Future improvements