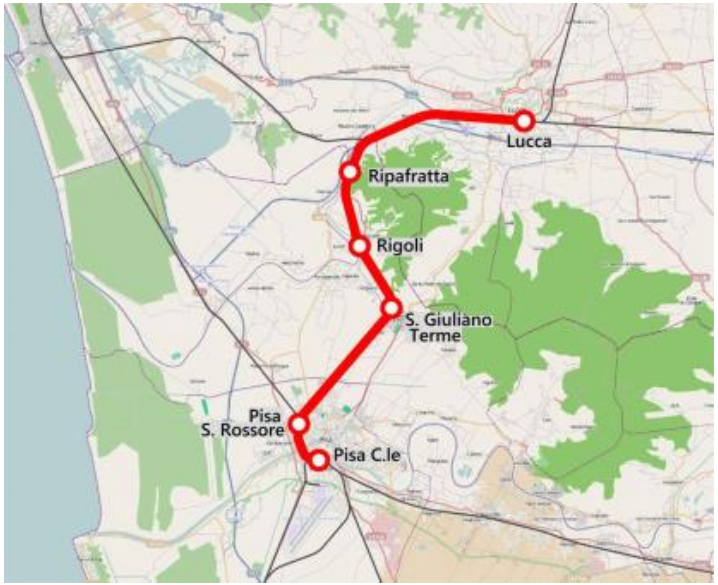
Mobile and Pervasive Systems - 2020 project

Group 1A - Federico Cappellini, Andrea Lelli, Alberto Lunghi, Lorenzo Susini

Application requirements

- **Monitor user's positions and store them locally**
  - creating a system to manage user's sessions

- **Offer the user an interface to manage his traces**
  - Visualize them.
  - Delete them completely or partially



Google’s new location system

The fused location provider is a location API in Google Play services that **intelligently combines different signals** to provide the location information needed by our app.

The fused location provider manages the underlying location technologies, such as GPS and Wi-Fi, and provides a simple API that can be used to specify the required quality of service. For example, the most accurate data available can be requested, or the best accuracy possible with no additional power consumption.

The main classes that are needed to properly use this API are

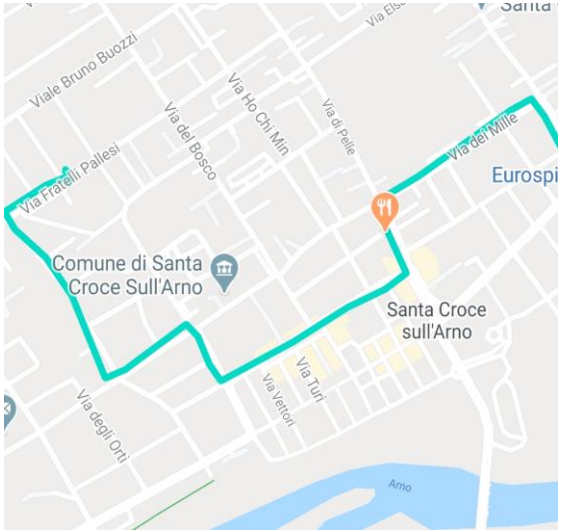
- **LocationRequest**
- **LocationCallback**
- **FusedLocationProviderClient**

Battery life vs accuracy

Best result of Balanced profile

VS

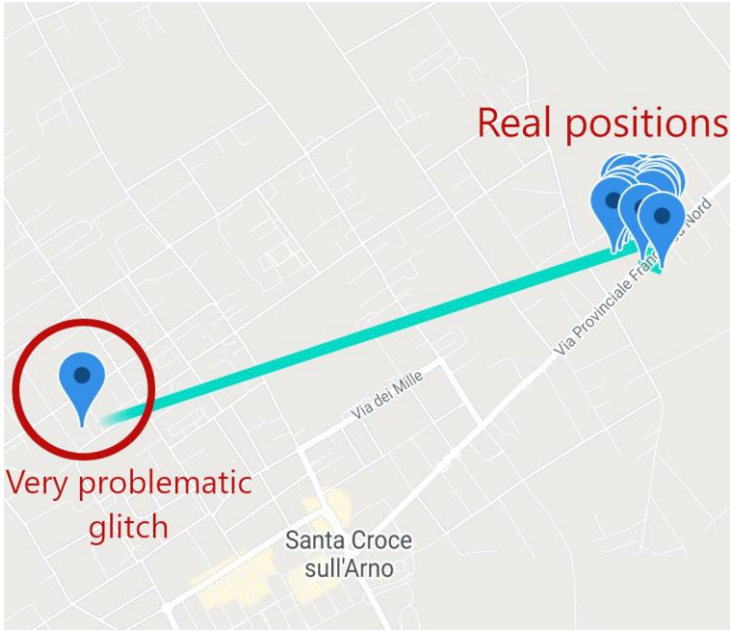
Average result of High precision profile



Battery life vs accuracy: the glitch problem

The glitch problem
Using the balanced profile, sometimes a glitch in the measures appears ruining the entire user session (he appears to have been in places never really visited or to have been tracked for kilometers more)

The proposed solution
The first proposed solution saw a post-processing filter able to spot the glitches and delete them

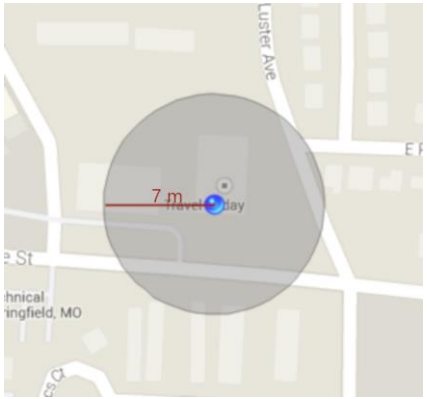


Battery life vs accuracy: final decision

Do we really need a balanced profile?
After a lot of test the result is that the saved energy using a balanced profile is negligible also considering that the filtering function could potentially take a long CPU time nullifying the advantages

Other ways to save energy

1. Ask the framework for a smallest displacement of 7 meters to avoid pointless computations for a pointless accuracy
2. Activity recognition system (that we will see in a while)
    - Automatic user's session recognition and trace recognition
3. Speed detection (that we will see in a while)



Google’s activity recognition API

User’s activity via events
Using Google’s API, it is possible to retrieve the user current activity avoiding polling requests. By registering a service as an activity recognition client it is possible to receive an event (to handle) every time the user’s activity changes.

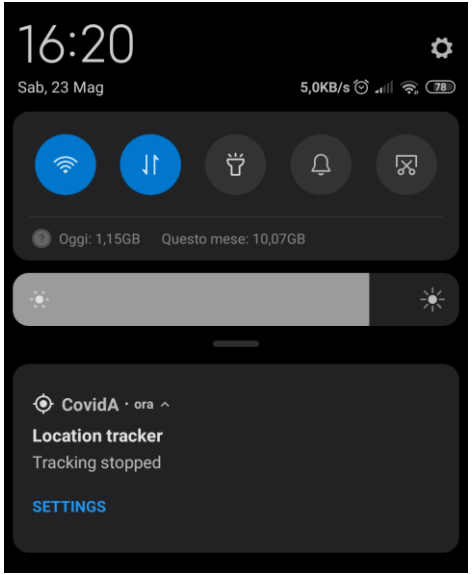
Probabilistic model
The event provides a list of possible user activities organized each with a confidence level.

What we really need
This system could potentially recognize also if the user is walking, running or using a vehicle. Anyway, after some tests, we decided to not rely on such automatic guesses. To make our application more “*user-aware*” we decided, instead, to use a speed-detection system.
At the end, **what we really need to know from this system is if the user is still or not**.

User-awareness: user session

In order to avoid to force the user to be *“application-aware”*, we built a more *“user-aware”* application. The activity detection system is used to detect when a tracking session starts (the user starts moving), when it ends (the user finally arrives at destination), when it must be paused (the user stop moving but his session is not completed) and when it must be resumed.

This lets the application be hugely more energy efficient without requiring the user intervention: **the user can potentially do not know even that the application is running**!



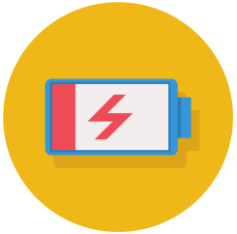
User-awareness: continuous trace

Each user session is subdivided in separated traces that are set of contiguous GPS points and for this reason take the name of ContinuousTraces.

Between the end of a ContinuousTrace and the following there are no other GPS points and this is because the GPS sampling is in a paused state to avoid pointless GPS sampling. This happens for instance when a user goes from home to the park and comes back to home: **it is pointless to take tens of sampling in the same place**.

This can also happen when the user decides to delete a part of his trace. As we saw, this has to be implemented for privacy issues. The result of this operation can be two distinct continuous traces.

User-awareness: speed detection

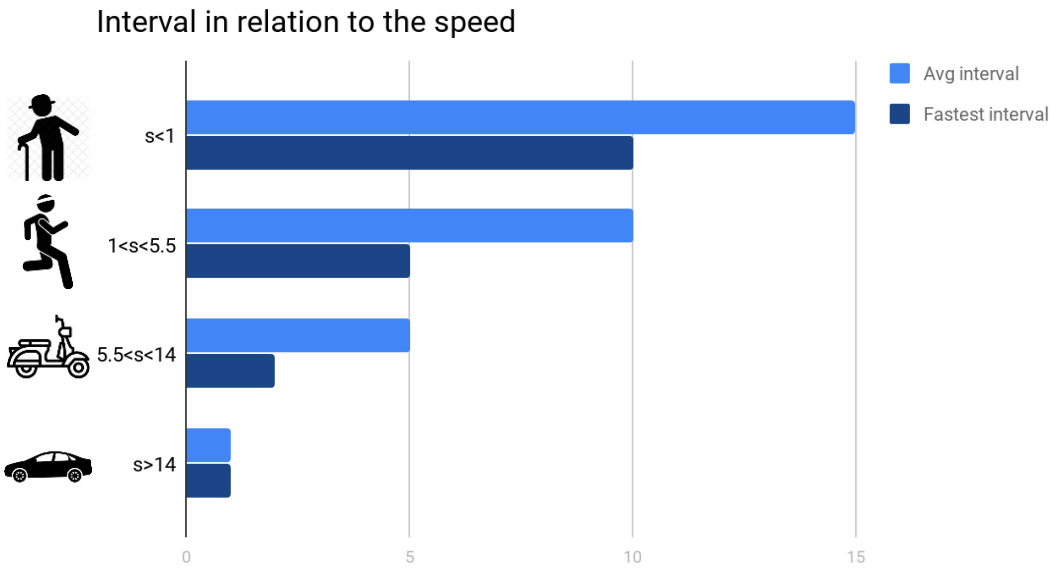


Battery life vs accuracy

When a user walks slowly there are no need to sample his position very fast. Vice versa, when a user is running a slow sample rate could lead to an inaccurate tracking.

We built a speed detection mechanism that prescinds the type of activity (that could potentially be a wrong guess): **what we really need is the user's speed**. This mechanism allows the application to adjust the sampling interval dynamically to obtain an optimal battery life and accuracy at the same time.

User-awareness: speed detection



User-awareness: home recognition

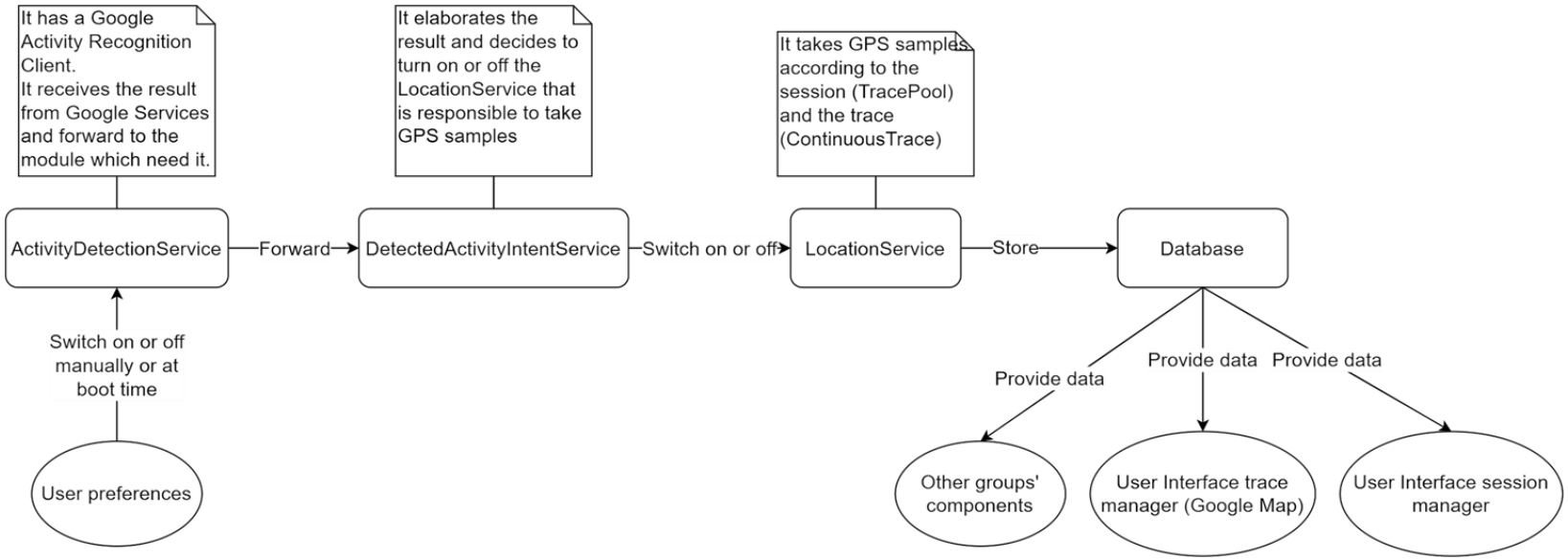
Requirement and problem

Rise an event automatically when the user comes back home. How to know where is the house?

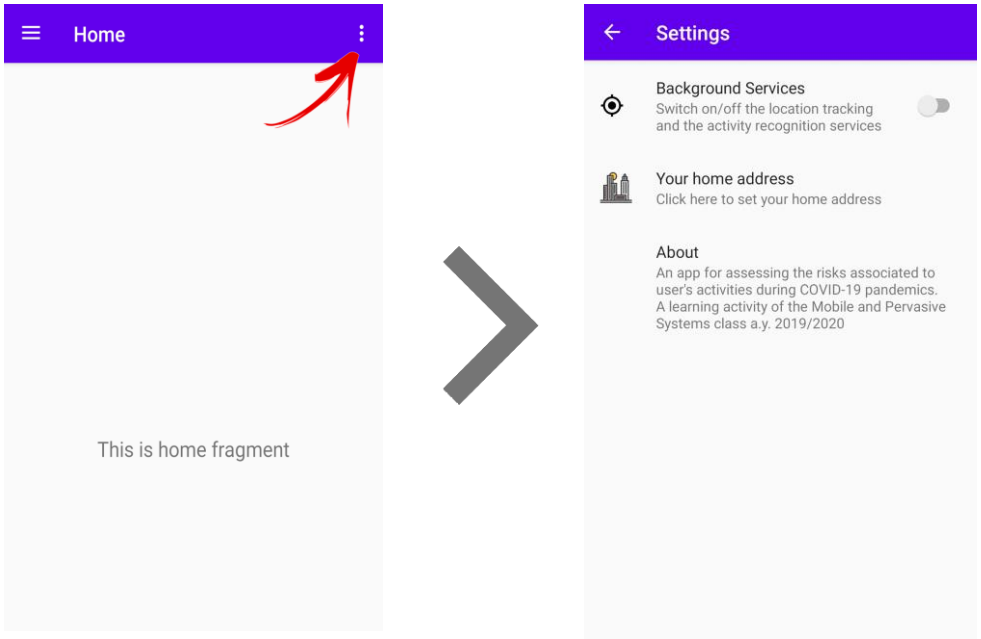
Three approaches

1. *Full "user-aware"*: the house is automatically recognized studying the user behavioural pattern
    - Error prone and unnecessarily complicated: it is a more pervasive solution, but not necessary.
2. *Geo-localized*: the user tells the application to be in his house and the application geo-localize it
    - Easy to use and to implement, but GPS accuracy may lead to non correct behaviours of the application.
    - Require the user being at home during the setting.
3. *House address*: the user inserts the address
    - Easy to use and to implement, no drawbacks.

Final architecture

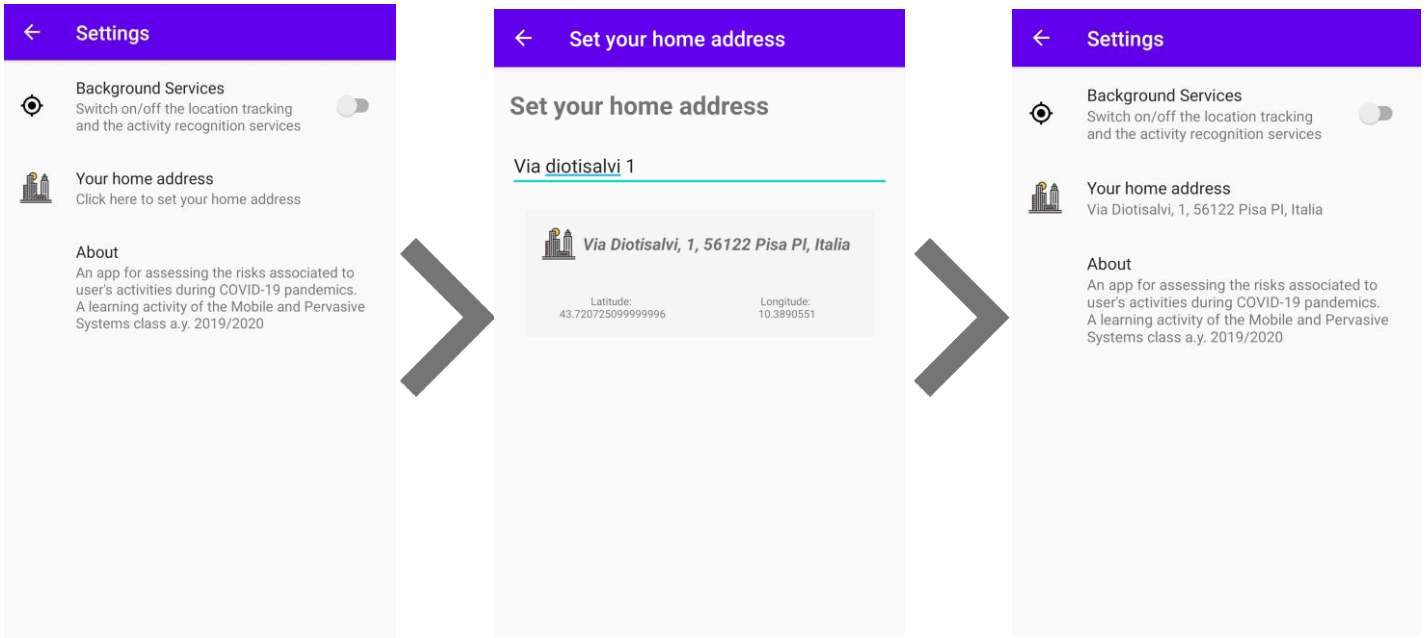


App tour: settings



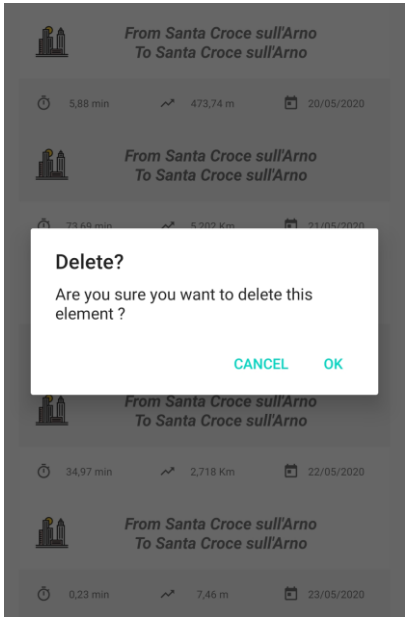
- on/off button to enable/disable activity recognition and location services
- button that allows the insertion of user's home address
- information about the application

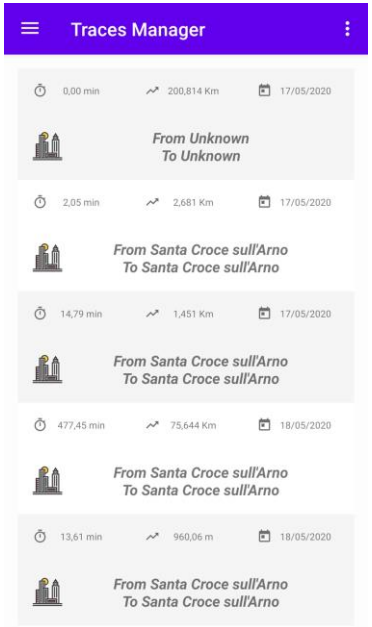
App tour: set home address

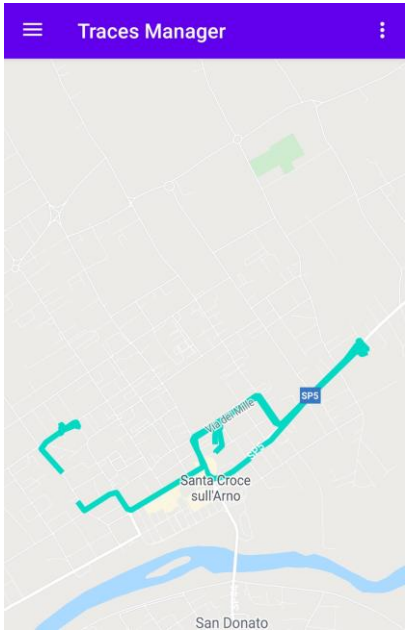


An example of home address setting.

App tour: session manager



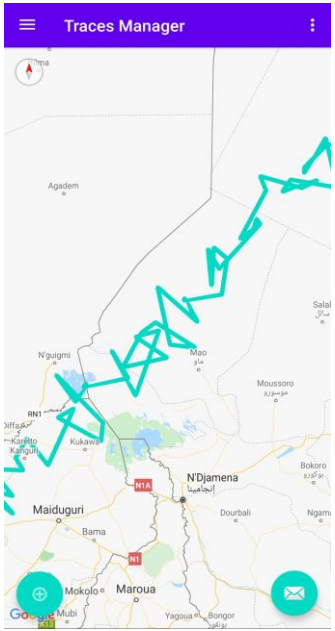




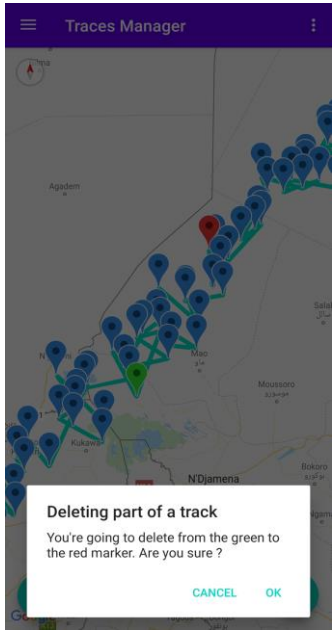
For each session:

- total time
- total travelled distance
- starting date of the sampling
- departure city
- arrival city

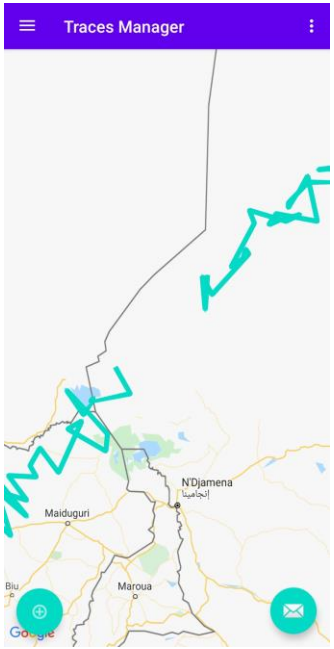
App tour: trace editing



tap on the track and the select the bounds of the trace you want to hide



press “OK” to confirm the deletion or “CANCEL” to abort it



Final considerations and conclusions

The application has the **advantage** of being almost completely independent from the user's actions:

- services always in background: activity recognition and location services are always up and running (if the user enabled them), even if the user closes the application (unless he "brutally" terminates it)
- recovery after boot: the application takes the previous settings specified by the user and restarts all the services, if he enabled them

Notifications are displayed in silent mode to improve:

- automatization
- usability

Any questions?