Exercise Project

The exercise is designed to allow you to gain practical skills and learn to work with real-world data. The goal of the exercise is to extract data points and parameters related to human activity from the inertial signal streams captured from a wearable accelerometer sensor. Throughout the project, you will develop processing techniques for inertial signals, test them, and analyze the results. To aid development, you will record your own inertial dataset using the wearable device and phone app we provide to obtain ecologically valid samples. Together with the datasets from other groups, this will facilitate better processing and also allow you to verify your algorithms.

The schedule of the exercise is designed to fit into the semester. The schedule will also leave time at the end of the semester to prepare for the end-of-term exam.

1 Grading

Each team can reach at most 100 points for the complete exercise. Additionally, in each subtask, the top performing teams will receive bonus points.

Subtask 1: Step Counting (Deadline: March 27, 2023)

- achievable points: 20 pts (20 %)
- All points awarded based on algorithm performance, evaluated with an unpublished dataset.
- 3, 2, and 1 bonus points for the three teams with best algorithm performances

Subtask 2: Data Collection for the Path Detection in Zurich (Deadline: April 21, 2023)

- achievable points: 10 pts (10 %)
- All points awarded for submitting the minimum data quota of 15 traces.
- 3, 2, and 1 bonus points for three teams submitting the most data traces.

Subtask 3: Path Detection in Zurich (Deadline: May 15, 2023)

- achievable points: 70 pts (70 %)
- 60 pts awarded based on algorithm performance (see Section 6.7)
- 10 pts awarded for short documentation (1000 words upper limit)
- For each of the 4 algorithm outputs separately: 2 and 1 bonus points for the two teams with the best algorithm performances.

Point aggregation

The theoretical point maximum is 114 for a team. No bonus points will be required to reach the full grade (6.0). All students in each team will be awarded the same grade for the exercise project. The team must return the provided LilyGo smartwatch (plus charging and programming cable, all in the box that the team obtained at the beginning) to receive their grade.

2 Project Teams (Deadline: 06.03.2023, 12:00 CET)

The exercise will be completed in teams of three students. The teams will persist throughout the whole semester. We encourage students to openly exchange their expectations (work style, ambition, etc.) before committing to a team. If one student of a team drops the class, the team will continue as a team of 2. To form teams, please enter your names in this spreadsheet:

spreadsheet to form teams

Action required: We do not provide a matching service. To find teammates, please use the second tab in the sheet and follow the instructions. Please form teams until **06.03.2023, 12:00**. If you are not signed up by then, we cannot guarantee that you will find a team afterward.

3 System Setup (Soft Deadline: 13.03.2023, 12:00)

Task: Set up the end-to-end pipeline to record and store data using the LilyGo smartwatch, an Android or iOS smartphone, and familiarize yourself with Kaggle. Additionally, you may also set up a local environment on your computer (not a must) as outlined in Section 9. Kaggle provides you with free computation power upon creating a free account. This includes unlimited CPU access, 30h GPU access, and 30h TPU access per account per week. Exercises will be hosted as Kaggle competitions.

For this step, each team will receive one LilyGo smartwatch that must be returned in working order after the end of the exercise. All the instructions and any required files will be provided in this Gitlab repository. All links below direct you to the relevant subfolders.

1. Install the Android (instructions) or iOS application (download, instructions), try recording data using the smartwatch. *Please note: we are currently working on updates to the Android and iOS applications.*
2. Set up Kaggle (and a Kaggle-compatible python environment if you like) according to specifications provided in Section 9. The exercise-specific files and instructions on how to use them can be found (here).

This step will not be validated or graded, but it will be required for the subsequent tasks. Therefore, we highly recommend completing this by the deadline.

3.1 Tutorial and Q&A: March 6 & 13

To aid you with the process, we will release a recording on Monday, March 6 as a tutorial, which you can watch in your own time. It will outline how to use Python in Kaggle and introduce you to coding concepts relevant to the exercises. You can find the tutorials here, here and here. We will follow up with an online Q&A session on Monday, March 13 13:00 before the lecture. Please post questions on Moodle throughout the week and up-vote what you see relevant. This way, we can prepare potentially helpful material for the Q&A session on March 13.

4 Subtask 1: Step Counting (Due: 27.03.2023, 12:00 CET)

Task: Develop an algorithm to detect step count based on accelerometer data from a LilyGo smartwatch. Your algorithm should be as accurate and robust as possible.

Your algorithm will be evaluated using a dataset recorded by the TA team. Your algorithm must not rely on any signal types other than the data from the LilyGo smartwatch (i.e., accelerometer, gyroscope, magnetometer, IMU temperature sensor). Please note:

- Step counting is a problem for which several algorithms have been proposed. You are free to build on existing algorithms or Python modules, but you may have to optimize them for the best results.
- The following paper may serve as one possible starting point: Walk Detection and Step Counting on Unconstrained Smartphones by Brajdic et al. This paper will also be the first reading assignment of the course. Note that the algorithms they propose are designed for use in a smartphone rather than a smartwatch, but they still offer insight into possible approaches. Google Scholar is useful to find additional related publications on the topic. Starting with Brajdic et al.'s paper, you can find related work in the articles they cited as well as by using 'cited by' in Google Scholar.
- To achieve optimal results, recording your own training data to inspect and understand the LilyGo's signal quality and characteristics will be useful. Make sure to annotate ground truth values for the number of steps you take when recording, which will later facilitate processing. To improve the robustness of your algorithm, it may be helpful for some of your training data to be 'dirty' (e.g. include activity other than walking).
- The definition of a step is described in Subsection 8.1

4.1 Submission

To be evaluated, you'll have to take part in the step count Kaggle competition and submit a Jupyter notebook file via Polybox. The Jupyter notebook file should follow the submission template in our git repository.

- Take part in the step count Kaggle competition. We will upload a set of 10 traces to Kaggle for which you can make predictions but won't see the actual step counts. When you make predictions for these 10 traces, you will receive an immediate score as feedback that will be visible on the challenge's leaderboard. The score on this public leaderboard will not affect your grade for the step count exercise, however. It helps you see how your solution performs on an unseen test dataset and how you compare to your coursemates.
- Hand in your algorithm in a Jupyter notebook file (.ipynb) that follows the submission template in our git repository. The algorithm will be used to make predictions for 30 traces to calculate your grade. Your algorithm has to be Kaggle-compatible without further package installations. An mhealth23.yml (and mhealth23Mac.yml) file is available in our git repository and instructions on how to set up a local environment are given in Section 9 for the ones who prefer to code locally. Include e-mail addresses and legi numbers of all team members at the

top of the source code. Update the filename to 'groupXX_stepcount.py', where XX is your group number in the sign-up sheet. Use a leading zero if it is a single digit.

Upload the file to Polybox. You may resubmit as many times as you wish before the deadline. The last uploaded version of each file will be graded.

4.2 Evaluation and Grading

Your grade will be calculated based on the performance of your submitted algorithm on a dataset of 30 traces. In total, your team will be awarded up to 20 points (20% of the total exercise).

As a small aid and booster for your data collection, we will release 5 different traces recorded by the same person on March 20, 2023. While this can diversify your dataset for training and testing, we recommend not to overfit your algorithm to this data set.

The traces that will be used to evaluate your algorithm were recorded with the same setup as provided to you (LilyGo smartwatch, Android smartphone app). For all traces in the dataset, the smartwatch was located on the user's left wrist with the screen facing up.

Your algorithm must yield a result within 10 s for each trace when executing in Kaggle on CPU. If your algorithm fails to compute the necessary output in time or crashes, the result will be rated as wrong classification for the input trace.

The three best-performing algorithms will be awarded 3, 2, and 1 bonus points, respectively.

5 Subtask 2: Data Collection for the Path Detection in Zurich (Due: 21.04.2023, 12:00 CET)

For the rest of the exercise project, we will be working with data recorded on 5 paths between Zurich main station and the ETH main building. 3 paths are going up, and 2 are going down. The paths are described in detail in the appendix. **Make yourself familiar with the routes before recording data!**

Task: For this task, your team will record activity data, accurately label it, and submit it. The recorded data will help you to solve Subtask 3, in which you will reconstruct traversed paths. Afterwards, your data will be partially made available to other groups and, in return, you will receive their datasets in order to deliver Subtask 3 later on.

Each team should record at least 3 data traces on each path in the correct direction. For each path, the LilyGo smartwatch should be worn at least once at the each of the following body locations throughout the whole recording of a single data trace:: 1) left wrist, 2) fastened to your belt/waistband, 3) right ankle. The paths should be completed by walking, running, cycling, or (sometimes) standing still/waiting *and/or combinations thereof*. **Note that each activity should have at least a full minute of continuous coverage** (i.e., do not run, wait for only 20 s, then walk etc.). If possible, we recommend a 50/50 split between traces recorded using an Android or IOS smartphone to achieve a balanced and robust data set. **To increase data quality while recording, do not lock your screen and do not switch to another application.** Division of labor is up to the team and data recording may be completed by one or more team members.

For the submission, provide the labels for your data recordings as follows:F

- Smartwatch location $l \in \{0, 1, 2\}$, (0: wrist left, 1: belt, 2: right ankle)
- Path index $p \in \{0, 1, 2, 3, 4\}$
- Activities contained in the data trace and performed for more than 60 s uninterrupted. Output as a list of integers: e.g., '[0, 3]' (0: standing still, 1: walk, 2: run, 3: cycle). To avoid edge cases, please note the additional specifications in Subsection 8.2 before data recording.

5.1 Smartwatch Locations

Record the datasets with the smartwatch worn as follows.

- **Left wrist:** Wear the smartwatch similar to a regular watch around your left wrist, display facing towards you when looking at the back of your hand. Both possible rotations are allowed (touch button aligned with thumb or pinky).
- **Belt/waistband:** Strap the watch on your belt or a belt loop, so that it is centered in front of your body. The display may face any direction.
- **Right ankle:** Strap the watch to your lower right leg just above your right ankle (approximately where the leg is the leanest). If your leg is too wide to use the watch strap, you can remove the watch body from the strap and use a sock, tape or other strap to hold it in place. The watch should be on the inside of the ankle (display facing towards your left foot).

5.2 Crowd-Sourced Data Aggregation

The data you upload will be aggregated and half of the data from all groups will be released to all groups to aid in training and testing their algorithms. Groups may upload more than the minimum 5 paths $\times$ 3 watch locations $=$ 15 data traces. Additional traces will not be released to everyone but be included in the testing set of the next task, giving groups that submit more traces the slight advantage of knowing a larger share of the test set during the training phase. The three groups with the most amount of uploaded (valid) data will be awarded bonus points (see below).

Note: If you are not able to record data in Zurich, you may find it helpful to record training data in other locations to solve the next task. However, only data from the specified paths can be submitted. If you are unable to record data due to constraints, please contact the TA team.

5.3 General Rules

The data you submit must be the actual raw recordings acquired with the LilyGo smartwatch provided by us. Data cropping, augmentation, or manual modification is not allowed, as it will impact everyone's performance in training their algorithms. If a group is found to have uploaded augmented or modified data, this will result in the disqualification of all members of the group. It will also result in failing the entire class for all members of the group.

5.4 Data Submission

Upload each trace as a separate .json file, following the naming convention 'groupXX_traceYY.json', where XX is your group number in the sign-up sheet and YY is the trace number (starting at 01). Use a leading zero if it is a single digit. All traces must pass the data integrity check provided in the source code and contain the required labels. To add labels, load the .json file and provide the argument `no_labels=False` to the Recording constructor. You will then be able to provide the lables through a series of input prompts. The labels will then automatically be added to the json file. Upload all files to this Polybox. You may upload files with the same filename multiple times until the deadline. The last uploaded version of each file will be graded.

5.5 Grading

Your team will be awarded up to 10 points (10% of the total points of the exercise). Missing/incomplete traces or corrupt data will result in a proportional deduction of points (e.g., 8 traces will result in 5 points). Only the traces numbered 01–15 will be taken into consideration for this. The three teams with the most submitted traces will be awarded 3, 2, and 1 bonus points, respectively.

6 Subtask 3: Path Detection in Zurich (Due: 15.05.2023, 12:00 CET)

Task: Using the previously recorded dataset, develop algorithms to detect activity, path, step count, and smartwatch location. To aid training and testing, you will receive the collectively recorded dataset from Subtask 2.

Your algorithm should predict and output the following:

- Smartwatch location $l \in \{0, 1, 2\}$, (0: wrist left, 1: belt, 2: right ankle)
- Path index $p \in \{0, 1, 2, 3, 4\}$
- Step count total $s_1 \in \mathbb{Z}$
- Activities contained in the data trace and performed for more than 60 s uninterrupted. Output as a list of integers: e.g., '[0, 3]' (0: standing still, 1: walk, 2: run, 3: cycle). These do not need to be in the right order and they do not need to occur multiple times. For more details, see Subsection 8.2.

In addition to taking part in the path detection kaggle competition and submitting the algorithms in the form of Python code, provide brief documentation about how you tackled this problem. All specifics are provided in the following subsections.

6.1 Available Data

- GPS data are considered labels for this task. They must not be used as an input to your algorithm and will not be included in the data traces used to evaluate your algorithm. **Exception:** GPS altitude may be used in your algorithm.
- Other than GPS, all data recorded by the app on the phone and from the LilyGo smartwatch may be used. However, depending on available hardware sensors on the phone provided data traces may vary (e.g., some phones may not have a barometric pressure sensor to measure altitude). A detailed list of guaranteed and optionally contained sensor traces can be found in Section 10. Make your algorithms robust to missing optional traces and make sure it does not crash in case a trace is missing.

6.2 Smartwatch Locations

Compared to step counting, on-body position recognition is a less commonly solved problem and you may find it more difficult to find related work. When looking for related work on this topic, you can start with Where am i: Recognizing on-body positions of wearable sensors by Kunze et al. and proceed with Google Scholar as described above. Note that it may be unlikely to find an algorithm that is ready to use for the specific problem in this subtask, but related work may still serve as inspiration (e.g., features to extract).

6.3 Step Count

For this part of the task, you may reuse your previously developed algorithm or a further developed iteration. The rules remain the same as outlined in Subsection 8.1. Note that you have more data traces available now compared to the previous step counting task. However, additional data does not necessarily contain more relevant information.

6.4 Activity Recognition

Activity recognition is a core challenge in the literature and many approaches have been published. You may want to take a look at Activity recognition from accelerometer data by Ravi et al. as well as the course's reading assignments to start your research. You may also take a look at existing Python packages that may help you to solve this problem (see Subsection 8.3)

6.5 Documentation

Prepare a short documentation of your algorithm. Your writeup should be no longer than 1000 words and no more than three pages (A4, one-sided, 2 cm margin, 10 pt minimum font size) including graphics and plots. Briefly explain what you did, how you did it, and justify your decisions.

6.6 Submission

- Take part in the trace recognition Kaggle competition. This time, roughly 25% of your predictions will go towards computing a public leaderboard and approx. 75% will go towards your grade.
- Hand in your algorithm as a Jupyter notebook file (.ipynb) that follows the submission template in our Gitlab repository. Include e-mail addresses and legi numbers of all team members at the top of the source code. Update the filename to 'groupXX_pathdetection.ipynb', where XX is your group number in the sign-up sheet. Use a leading zero if it is a single digit.
- Submit the documentation as a PDF file, naming it 'groupXX_documentation.pdf'

Upload all files to this Polybox. You may upload multiple times until the deadline. The last uploaded version of each file will be graded.

6.7 Evaluation and Grading

Based on the quality of 75% of your predictions in Kaggle, your team will be awarded up to 60 points (60% of the total points of the exercise). The two best-performing algorithms for each of the 4 categories will be awarded 2 and 1 bonus points, respectively.

- Path index, smartwatch location, and activity detection will be verified using the unreleased part of the collectively recorded dataset as well as additional data recorded by the TA team.

- Step count will be verified on an unpublished reference data set of at least 3 iterations of each path (one for each smartwatch location). This dataset was recorded by the TA team using the same pipeline as the students.

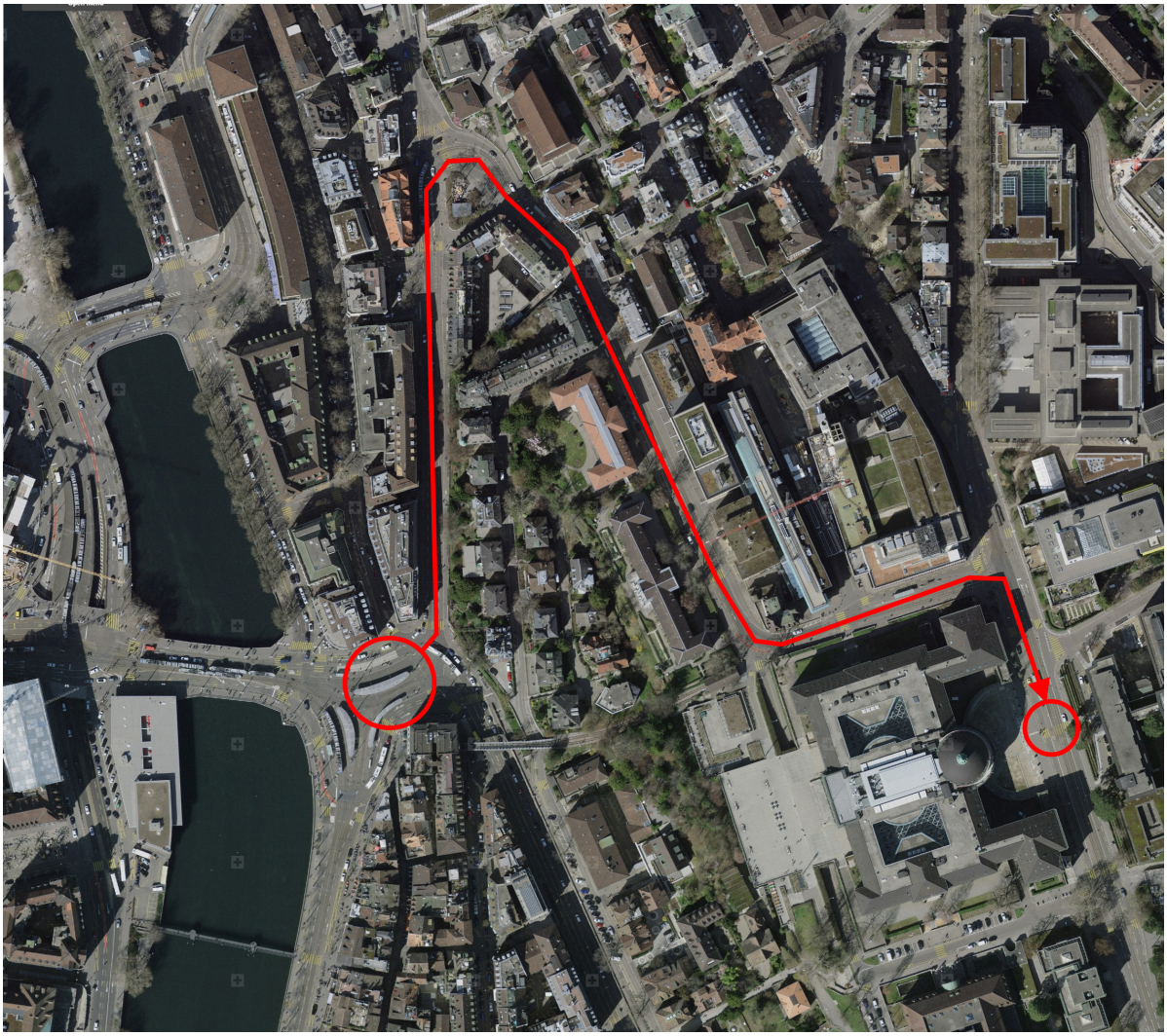
You must not implement algorithms that recognize specific data traces to improve your results. Hard coding classification results will result in the disqualification of all members of the group. It will also result in failing the entire class for all members of the group.

Your algorithm must yield a result within 10 s when executing in Kaggle on CPU. If your algorithm fails to compute the necessary output in time or crashes, the result will be rated as the wrong classification for the input trace.

The documentation will be awarded up to 10 points (10% of the total points of the exercise). Your grade for the documentation will be independent of your algorithm's performance and will focus on your methodology, approach, and justification.

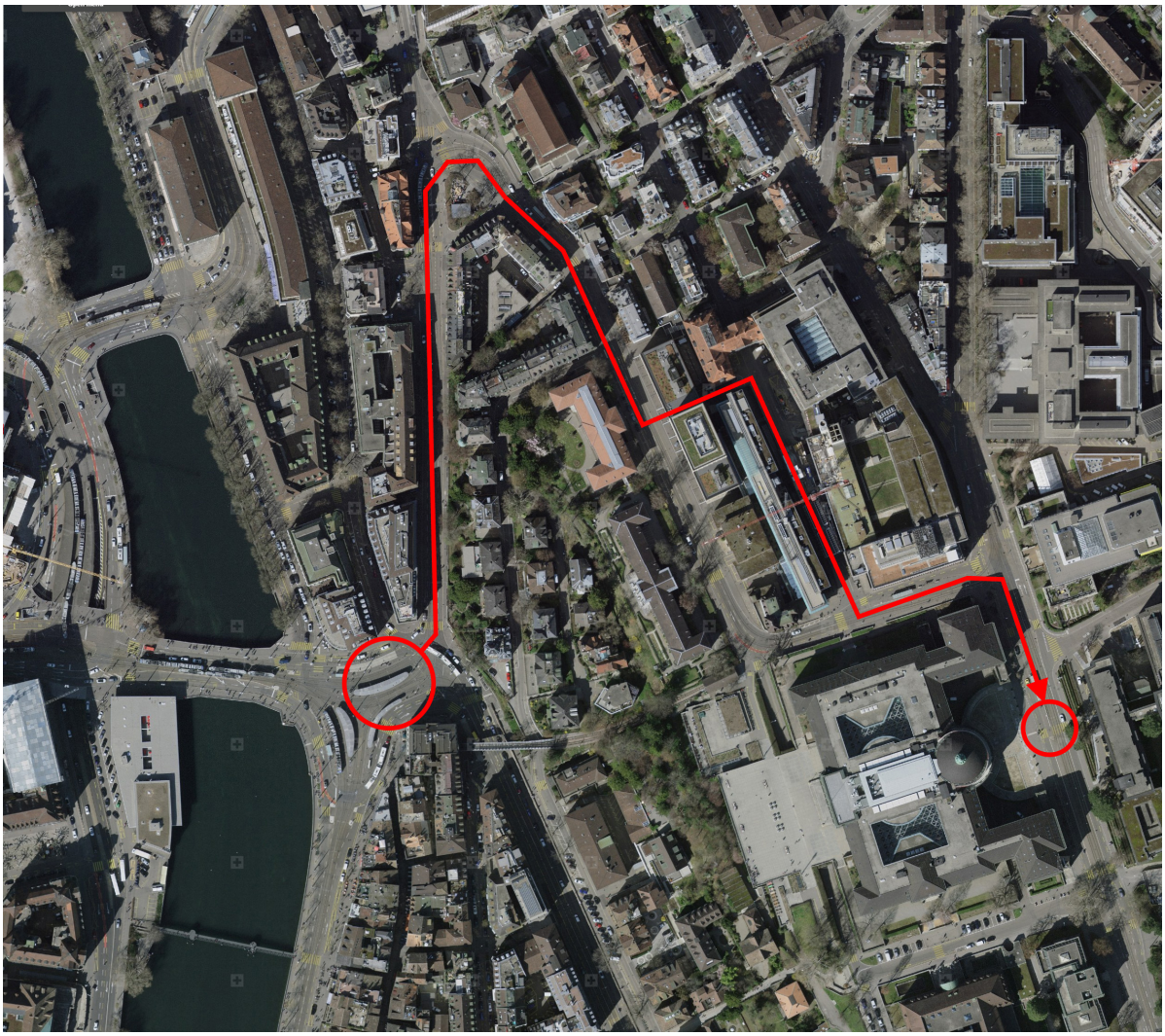
7 Zurich Paths Documentation

7.1 Path 0: Central to Main Building



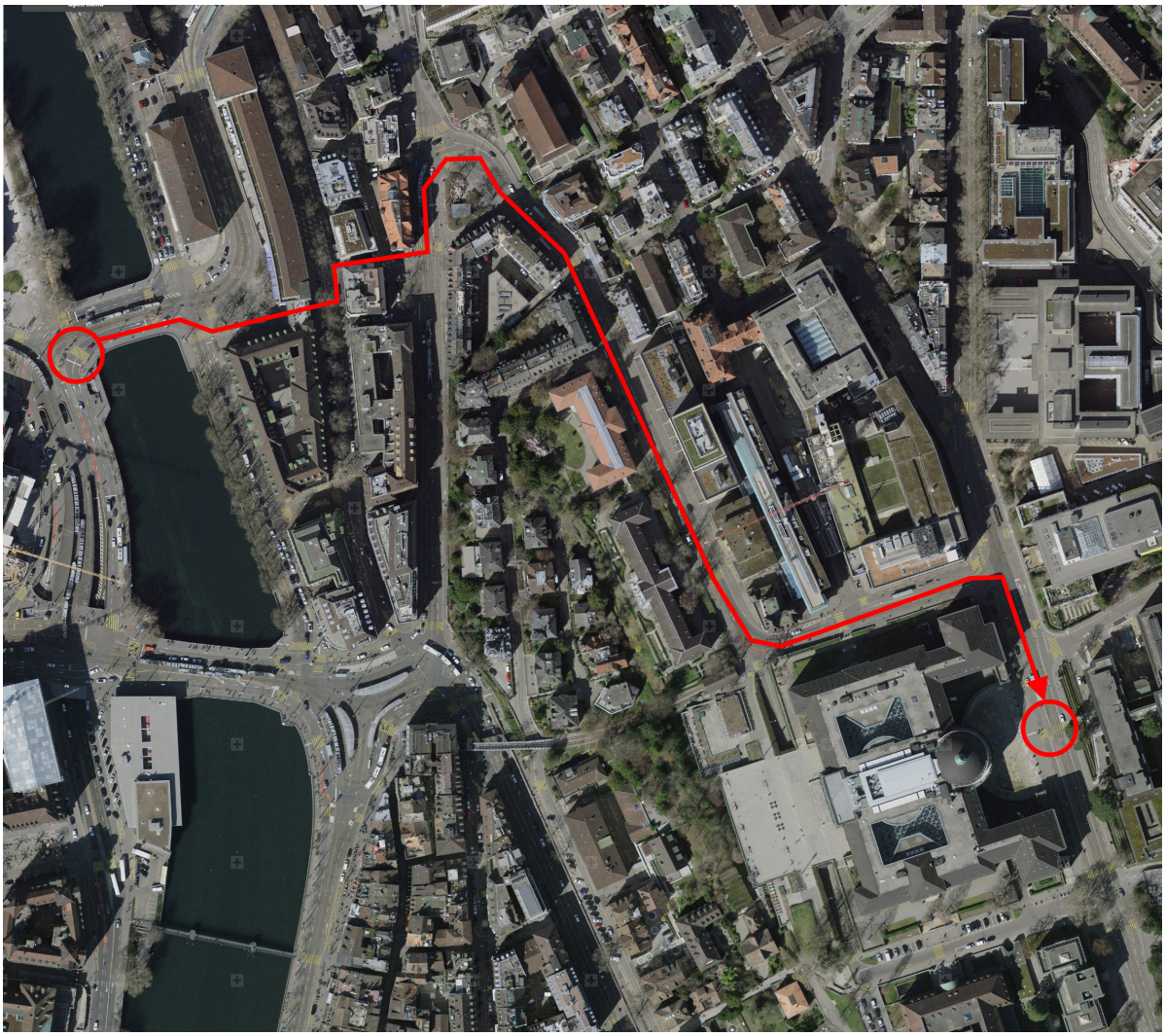
- **Start:** Central ($r_{tolerance} = 25m$)
- **End:** Main building, center in front on Rämistrasse side ($r_{tolerance} = 15m$)
- **Route:** Weinbergstrasse – Leonhardstrasse – Rämistrasse
- **Distance:** 830 m
- **Elevation gain:** 44 m
- **Special notes/constraints:**
  - Do not take the shortcut to cut the corner from Weinbergstrasse to Leonhardstrasse.

7.2 Path 1: Central to Main Building, Clausiusstrasse Variant



- **Start:** Central ($r_{tolerance} = 25m$)
- **End:** Main building, center in front on Rämistrasse side ($r_{tolerance} = 15m$)
- **Route:** Weinbergstrasse – Leonhardstrasse – Clausiusstrasse – Rämistrasse
- **Distance:** 840 m
- **Elevation gain:** 44 m
- **Special notes/constraints:**
  - Do not take the shortcut to cut the corner from Weinbergstrasse to Leonhardstrasse.

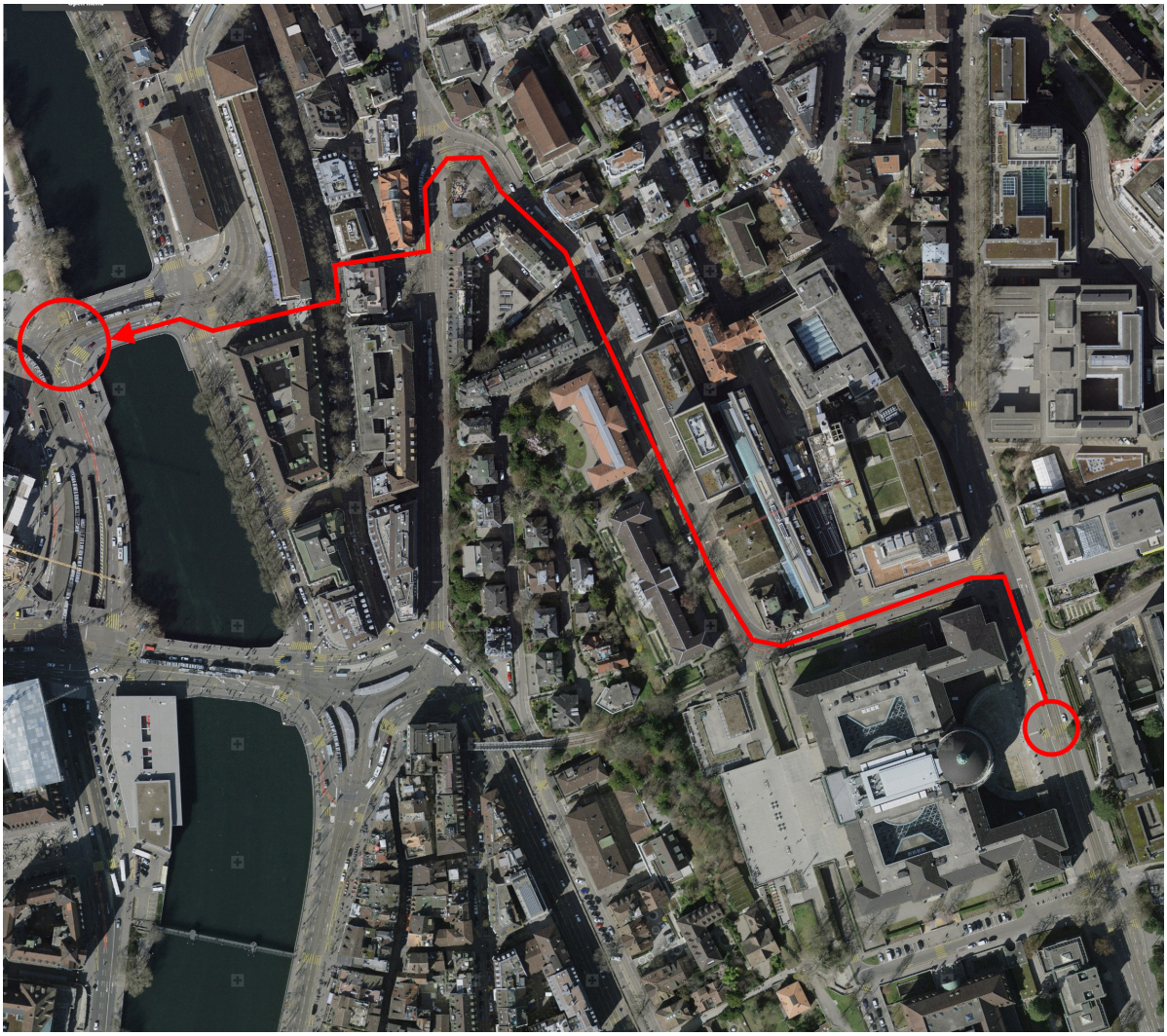
7.3 Path 2: Walchebrücke to Main Building



- **Start:** Walchenbrücke, east edge ($r_{tolerance} = 15m$)
- **End:** Main building, center in front on Rämistrasse side ($r_{tolerance} = 15m$)
- **Route:** Walchebrücke – Walchetor (pedestrian passage with a short staircase) – across Stampfenbachstrasse, up the stairs to Leonhardstrasse – Rämistrasse
- **Distance:** 830 m
- **Elevation gain:** 47 m
- **Special notes/constraints:**
  - Do not take the shortcut to cut the corner from Weinbergstrasse to Leonhardstrasse.

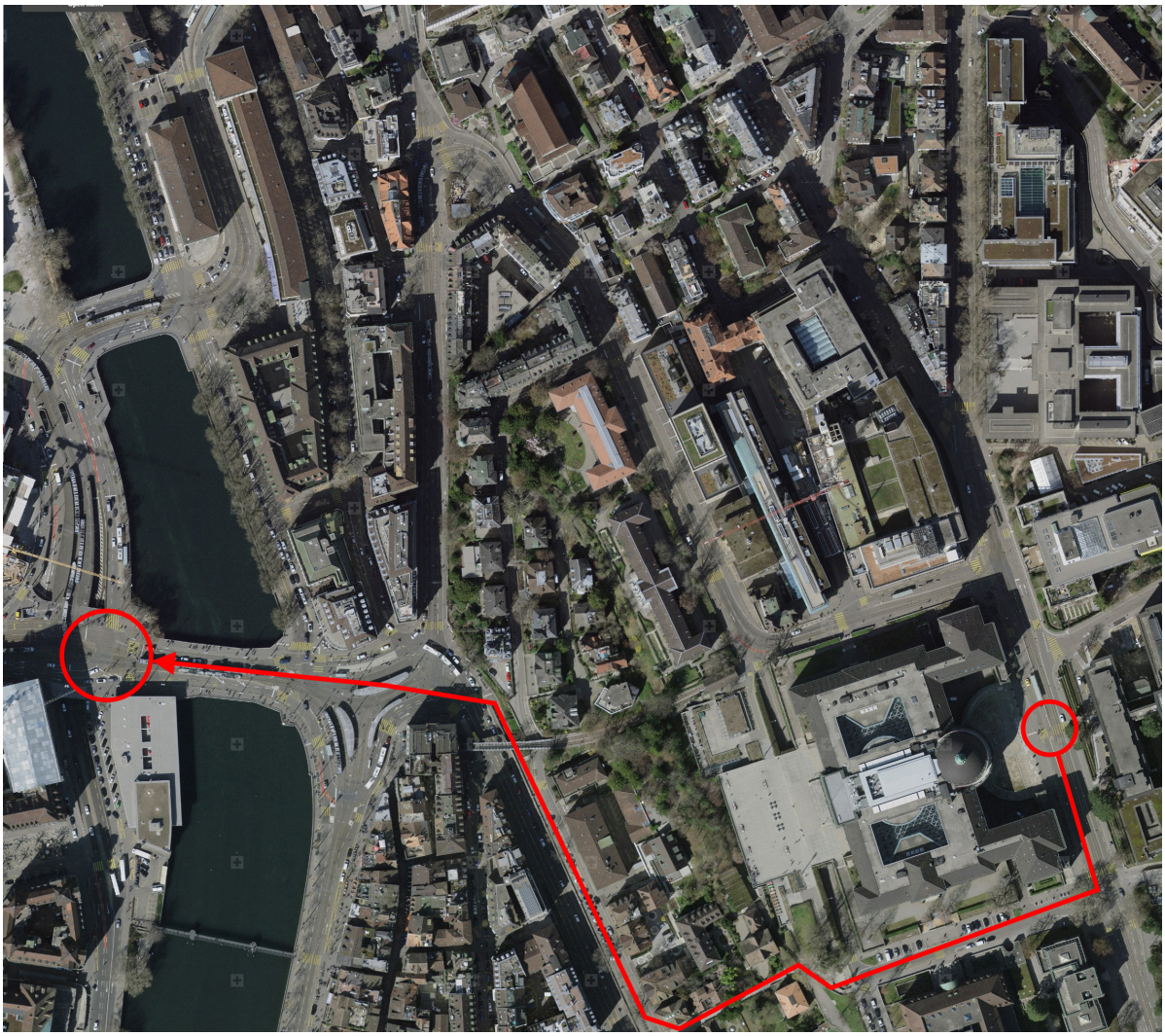
7.4 Path 3: Main Building to Walchebrücke

This path is the exact reverse of path 2.



- **Start:** Main building, center in front on Rämistrasse side ($r_{tolerance} = 15m$)
- **End:** Walchenbrücke, east edge ($r_{tolerance} = 25m$)
- **Route:** Rämistrasse – Leonhardstrasse – stairs down to Stampfenbachstrasse – Walchetor (pedestrian passage with a short staircase) – Walchebrücke
- **Distance:** 830 m
- **Elevation gain:** -47 m
- **Special notes/constraints:**
  - Do not take the shortcut to cut the corner from Leonhardstrasse to Weinbergstrasse.

7.5 Path 4: Main Building to Bahnhofquai



- **Start:** Main building, center in front on Rämistrasse side ($r_{tolerance} = 15m$)
- **End:** Junction Bahnhofquai, Bahnhofbrücke ($r_{tolerance} = 25m$)
- **Route:** Rämistrasse – Karl-Schmid-Strasse – Schienhutgasse – Hirschengraben – Central – Bahnhofbrücke
- **Distance:** 780 m
- **Elevation gain:** -45 m
- **Special notes/constraints:**
  - Make sure to use the Hirschengraben street and not Seilergraben street before Central (they run in parallel, Hirschengraben is elevated above Seilergraben)

8 Tips, Tricks, and Rules

8.1 Definition of a Step

For this exercise, we define a step as the act of lifting and setting down one's foot, contributing to a movement of the whole body. Walking stairs or running therefore contributes to your step count while cycling, sitting, standing still, ... does not. Your algorithms will be evaluated using natural body movements only (i.e., we will not record data of drawing the feet over the floor to move without producing step events or similar).

8.2 Avoiding Edge Cases in Activity Recognition

To avoid edge cases in activity recognition, please adhere to the following rules when recording and classifying data:

- Avoid performing any activity longer than 30 s but shorter than 60 s unless the activity is contained in the trace in a longer uninterrupted interval already.
- When recording data, do not walk or run in place or circles.
- Walking/Running stairs can be considered as normal walking or running.
- Differentiation between walking/running and standing still:
  - When recording data, minimize steps during a phase of standing still.
  - When classifying, up to 8 s of standing still between two steps may be still counted as uninterrupted walking/running.
  - Standing still counts as such even if the trace contains up to 7 steps within a minute of standing still.
- Differentiation between walking and running:
  - Running is characterized by an aerial phase in which all feet are above the ground. This is in contrast to walking, where one foot is always in contact with the ground.
  - When recording, minimize walking or standing still during a phase of running.
  - When classifying, up to 8 s of standing still or walking between running intervals may be still counted as uninterrupted running.

8.3 External Code and Algorithms

You may use any Python package and algorithm that is included in the Kaggle environment.yml file. If you use algorithms that are not completely your own development, you must label your source with a comment in the code including a link to the resource. Collaboration across groups is not permitted. This includes sharing code, results, data or anything that would represent an advantage to other groups.

9 Python Specifications

If you are not familiar with clean, portable Python package management, you may take this project as a chance to learn it or use Kaggle to code. Since we have to test your algorithms it is essential to the exercise that you work in an appropriate Python environment. If you choose to work locally, please follow the guidelines below.

9.1 Setup to code locally

You will need a way to create virtual Python environments. Think about them as independent Python installations you can easily create, switch between, and transfer to other machines. There are multiple tools to do this, we recommend Anaconda which is available for Windows, Linux, and macOS. We created a small sufficient environment with some basic packages that can be created using the mhealth23.yml file (or mhealth23Mac.yml for Mac users). If you want to add packages check the packagesCPU.txt or packagesGPU.txt files (depending if you're only using CPU or also GPU) to check if the package is Kaggle-compatible and what version is installed on Kaggle. Especially when using a Mac, there might be some unavoidable clashes depending on what packages you intend to use. When coding locally, the responsibility to ensure that the code is Kaggle-compatible ultimately lies with you. Please ensure that the Code runs as intended on Kaggle regularly to avoid surprises on deadline-day.

9.1.1 Anaconda

Setting things up and basic usage:

1. Install Anaconda
2. Place the mhealth23.yml file from our gitlab repository in the folder where you want to create your virtual environment
3. Open Anaconda prompt, type `conda env create -f mhealth23.yml` to create an environment named mhealth23 with Python 3.7 (this may take a while)
4. Activate the environment: `conda activate mhealth23`
5. To run e.g., jupyterlab, just type the command in the anaconda prompt while the desired environment is active: `jupyter lab`
6. If you want to use interactive plots in your notebooks, you might have to use jupyter notebook (run: `jupyter notebook`) due to issues with jupyterlab and the version of ipympl installed on Kaggle
7. Happy coding!

9.1.2 Jupyter Lab

We recommend using JupyterLab (see here). In this environment, it is easy to visualize data und test algorithms. To submit your project, copy your algorithms into the provided .ipynb template files. Please test these files before submitting.

9.1.3 Interactive Data Visualization

Data visualization is an important aspect of data analysis. We recommend that you thoroughly visualize and inspect the data before implementing any algorithms.

Especially with long data traces, interactive plots that allow responsive zooming can be very helpful. We recommend using Matplotlib in a Jupyter notebook with the interactive features provided through ipympl.

This sample code will provide you with an interactive plot with two subplots on a shared x-axis. You may use this as a starting point for data visualization.

```
import matplotlib.pyplot as plt
%matplotlib widget
fig,ax=plt.subplots(nrows=2,ncols=1,figsize=(10, 6), sharex=True)

# plotting two lines in the top subplot
ax[0].plot(xvals, yvals, label="somelabel")
ax[0].plot(xvals2, yvals2, label="somelabel2")

# plotting a line in the bottom subplot
ax[1].plot(xvals3, yvals3, label="somelabel3")

ax[0].legend(loc='lower right')

# you can plot to the other subplot using this pattern
#ax[1].plot(...)

#programmatic zoom options
#ax[0].set_xlim([123, 543])
#ax[0].set_ylim([9000000, 9500000])

plt.tight_layout()
plt.show()
```

10 Available Sensor Data

The following is a list of available sensor traces when loading a recording in Python. The name can be used as dictionary key. Bold sensor names must be contained when submitting data and will always be contained when testing algorithms. All other sensors may or may not be included depending on the phone used. You may use them in your algorithm but it is not guaranteed that the data will be included in the test set.

Meta data

- **timestamp**: UNIX timestamp in ms when packet was received
- **packetNumber**: Sent with BLE packet from wristband, should count up 1 per packet and wrap around 1 byte
- lostPackets: Number of lost packets since last received packet
- note: String of user note during recording

Smartwatch data

- **ax**: Wristband accelerometer X-axis [g]
- **ay**: Wristband accelerometer Y-axis [g]
- **az**: Wristband accelerometer Z-axis [g]
- **gx**: Wristband gyroscope X-axis [deg/s]
- **gy**: Wristband gyroscope Y-axis [deg/s]
- **gz**: Wristband gyroscope Z-axis [deg/s]
- **mx**: Wristband magnetometer X-axis [μT]
- **my**: Wristband magnetometer Y-axis [μT]
- **mz**: Wristband magnetometer Z-axis [μT]
- **temperature**: Wristband IMU die temperature [°C]

Phone data

- **longitude**[1]: Phone GPS longitude [degrees]
- **latitude**[1]: Phone GPS latitude [degrees]
- **altitude**: Phone GPS altitude [m]. *Please note that altitude measurements might be sparse. They should thus be treated with caution. Despite changes in altitude, smartphones might just duplicate a previous measurement.*
- bearing[1]: Phone GPS bearing [degrees]
- speed[1]: Phone GPS speed [m/s]
- **phone_ax**: Phone accelerometer X-axis [m/s^2]
- **phone_ay**: Phone accelerometer Y-axis [m/s^2]
- **phone_az**: Phone accelerometer Z-axis [m/s^2]
- **phone_gx**: Phone gyroscope X-axis [rad/s]
- **phone_gy**: Phone gyroscope Y-axis [rad/s]
- **phone_gz**: Phone gyroscope Z-axis [rad/s]
- **phone_mx**: Phone magnetometer X-axis [μT]
- **phone_my**: Phone magnetometer Y-axis [μT]
- **phone_mz**: Phone magnetometer Z-axis [μT]
- phone_gravx: Phone gravity X-axis [m/s^2]
- phone_gravy: Phone gravity Y-axis [m/s^2]

[1]Must not be used in classification

- phone_gravz: Phone gravity Z-axis $[m/s^2]$
- phone_lax: Phone linear acceleration X-axis $[m/s^2]$
- phone_lay: Phone linear acceleration Y-axis $[m/s^2]$
- phone_laz: Phone linear acceleration Z-axis $[m/s^2]$
- phone_rotx: Phone rotation vector X-axis
- phone_roty: Phone rotation vector Y-axis
- phone_rotz: Phone rotation vector Z-axis
- phone_rotm: Phone rotation vector $\cos(\theta/2)$
- phone_magrotx: Phone geomagnetic rotation vector X-axis
- phone_magroty: Phone geomagnetic rotation vector Y-axis
- phone_magrotz: Phone geomagnetic rotation vector Z-axis
- phone_orientationx: Phone orientation azimuth [degrees]
- phone_orientationy: Phone orientation pitch [degrees]
- phone_orientationz: Phone orientation roll [degrees]
- phone_steps: Phone step counter (probably won't work)
- phone_temp: Phone temperature sensor [°C]
- phone_light: Phone light sensor [lx]
- phone_pressure: Phone pressure sensor [hPa or mbar]
- phone_humidity: Phone humidity sensor [%]

Find more info about the Android sensors here:

- sensors overview
- position sensors
- location
- environment sensors

11 Bugs and Issues

If you encounter unexpected behavior, please file an issue in our GitLab repository. If you already have a possible fix, please create both an issue and a pull request associated with it. If you do not know how to do so, feel free to check out online tutorials or just send us the fix through e-mail see Section 12).

12 Any Other Questions?

Please ask any questions regarding this work in the Moodle forum of the lecture. Make sure to not give away parts of your solution when asking a question. If you have a question that does include parts of a solution or include specific ideas on how to solve a task, you may ask through e-mail. Send your e-mail to max.moebus@inf.ethz.ch, the subject must start with *[mhealth23 exercise]*.

13 Changelog

Here we will keep a changelog of this document. Corrections may be released during the exercise. Please check Moodle to stay up-to-date.

02.03.2023 original release

07.03.2023 added links to Kaggle competition & tutorials

07.03.2023 added guideline to improve data quality of recordings

09.03.2023 published the IOS App and added mhealth23Mac.yml file for Mac users