ast_monitor

Release 0.3.2

Iztok Fister Jr., Luka Lukač

Jun 17, 2023

USER DOCUMENTATION

1 Objective **3**

2 Outline of this repository **5**

3 Hardware outline **7**

3.1 Getting Started . . . 8
3.1.1 Installation . . . 8
3.1.2 Examples . . . 8
3.2 Installation . . . 8
3.2.1 Setup development environment . . . 8
3.3 Testing . . . 9
3.4 Documentation . . . 9
3.5 API . . . 9
3.5.1 Basic data module . . . 9
3.5.2 Digital Twin module . . . 10
3.5.3 GPS Sensor module . . . 11
3.5.4 HR Sensor module . . . 11
3.5.5 Interval Training module . . . 12
3.5.6 Main Window module . . . 12
3.5.7 Model module . . . 12
3.5.8 Simulation module . . . 12
3.5.9 Training Session module . . . 12
3.5.10 Write log module . . . 13
3.6 Contributing to AST-Monitor . . . 13
3.6.1 Code of Conduct . . . 13
3.6.2 How Can I Contribute? . . . 14
3.7 Contributor Covenant Code of Conduct . . . 14
3.7.1 Our Pledge . . . 14
3.7.2 Our Standards . . . 14
3.7.3 Enforcement Responsibilities . . . 15
3.7.4 Scope . . . 15
3.7.5 Enforcement . . . 15
3.7.6 Enforcement Guidelines . . . 15
3.7.7 Attribution . . . 16

4 Indices and tables **17**

AST-Monitor is a wearable Raspberry Pi computer for cyclists.

- **Free software:** MIT license
- **Github repository:** https://github.com/firefly-cpp/AST-Monitor
- **Python versions:** 3.6.x, 3.7.x, 3.8.x, 3.9.x, 3.10.x

CHAPTER

ONE

OBJECTIVE

This repository is devoted to the AST-monitor, i.e. a low-cost and efficient embedded device for monitoring the realization of sport training sessions that is dedicated to monitor cycling training sessions. AST-Monitor is a part of Artificial Sport Trainer (AST) system. First bits of AST-monitor were presented in the following [paper](https://arxiv.org/abs/2109.13334).

CHAPTER

TWO

OUTLINE OF THIS REPOSITORY

This repository presents basic code regarded to GUI. It was ported from the initial prototype to poetry.

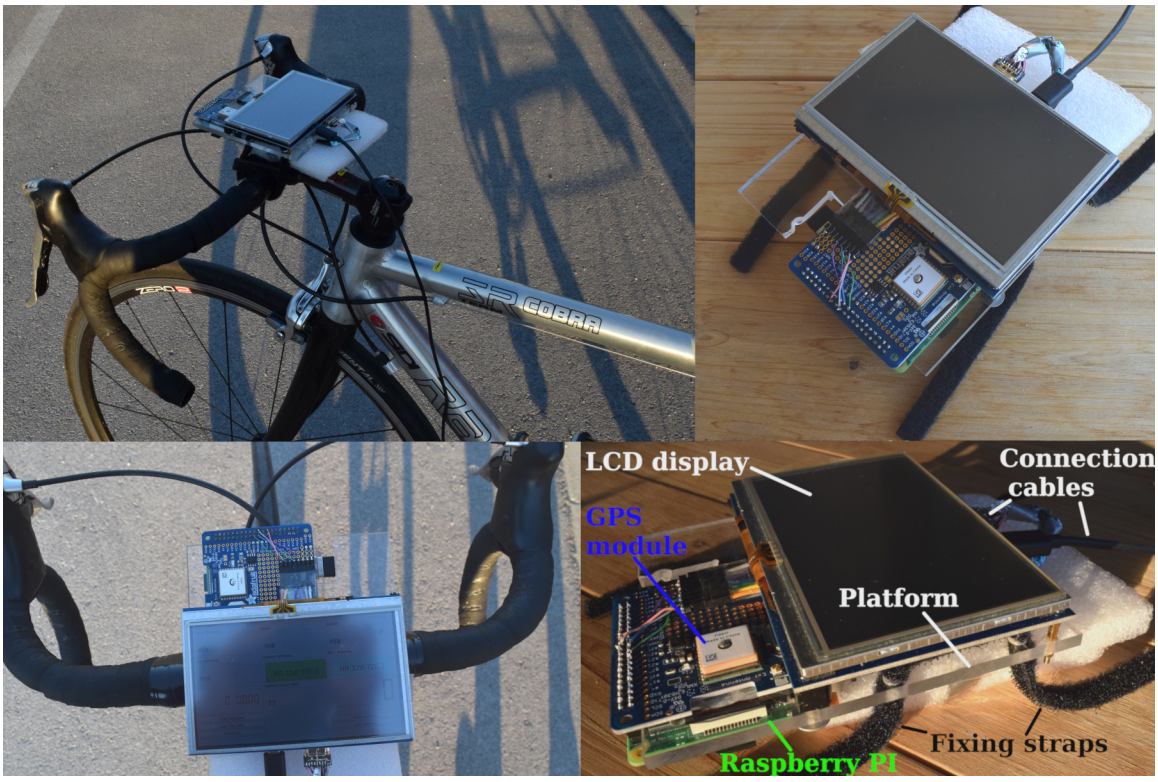
CHAPTER

THREE

HARDWARE OUTLINE

The complete hardware part is shown in Fig from which it can be seen that the AST-computer is split into the following pieces:

- a platform with fixing straps that attach to a bicycle,
- the Raspberry Pi 4 Model B micro-controller with Raspbian OS installed,
- a five-inch LCD touch screen display,
- a USB ANT stick,
- Adafruit's Ultimate GPS HAT module.



A Serial Peripheral Interface (SPI) protocol was dedicated for communication between the Raspberry Pi and the GPS peripheral. A specialized USB ANT stick was used to capture the HR signal. The screen display was connected using a modified (physically shortened) HDMI cable, while the touch feedback was implemented using physical wires. The computer was powered during the testing phase using the Trust's (5 VDC) power-bank. The AST-Monitor prototype is still a little bulky, but a more discrete solution is being searched for, including the sweat drainer of the AST.

The main documentation is organized into a couple of sections:

- *User Documentation*
- *Developer Documentation*
- *About*

3.1 Getting Started

This section is going to show you how to use the AST-Monitor toolbox.

3.1.1 Installation

First install AST-Monitor package using the following command:

```
pip install ast-monitor
```

After the successful installation you are ready to run your first example.

3.1.2 Examples

You can find usage examples here.

3.2 Installation

3.2.1 Setup development environment

Requirements

- Poetry: https://python-poetry.org/docs/

After installing Poetry and cloning the project from GitHub, you should run the following command from the root of the cloned project:

```
$ poetry install
```

All of the project's dependencies should be installed and the project ready for further development. **Note that Poetry creates a separate virtual environment for your project.**

Development dependencies

List of AST-Monitor dependencies:

Package | Version | Platform |

+==============+============+==========+ | PyQt5 | ^5.15.6 | All | | matplotlib | ^3.5.1 | All | | geopy | ^2.2.0 | All | | openant | v0.4 | All | | pyqt-feedback-flow | ^0.1.0 | All | | tcxreader | ^0.3.8 | All | | sport-activities-features | ^0.2.9 | All |

List of development dependencies:

Package	Version	Platform
Sphinx	^3.5.1	Any
sphinx-rtd-theme	^0.5.1	Any

3.3 Testing

Before making a pull request, if possible provide tests for added features or bug fixes.

In case any of the test cases fails, those should be fixed before we merge your pull request to master branch.

For the purpose of checking if all test are passing localy you can run following command:

```
$ poetry run python -m unittest discover
```

3.4 Documentation

To locally generate and preview documentation run the following command in the project root folder:

```
$ poetry run sphinx-build ./docs ./docs/_build
```

If the build of the documentation is successful, you can preview the documentation in the docs/_build folder by clicking the `index.html` file.

3.5 API

This is the AST-Monitor API documentation, auto generated from the source code.

AST_Monitor package - API

AST-Monitor — A wearable Raspberry Pi computer for cyclists

3.5.1 Basic data module

class basic_data.**BasicData**(*hr_data_path: str*, *gps_data_path: str*)

Bases: `object`

Class for storing and tracking the basic training data in real time.

Parameters

- **hr_data_path** (*str*) – path to the file that contains HR data
- **gps_data_path** (*str*) – path to the file that contains GPS data

calculate_speed() → None

Calculating the speed between the previous and the current trackpoint.

read_current_gps() → None

Reading the current GPS position from the file.

read_current_hr() → None

Reading the current HR from the file.

3.5.2 Digital Twin module

class digital_twin.**DigitalTwin**(*proposed_heart_rate: int*, *duration: int*, *tick_time: float*, *basic_data*)

Bases: object

Class that represents a digital twin that uses a mathematical prediction model to analyze an exercise in the real time.

Parameters

- **predicted_heart_rate** (*int*) – a predicted heart rate of an interval (speed or rest segment) in beats per minute
- **duration** (*int*) – duration of an interval in minutes

calculate_prediction(*trimp_delta: float*, *average_heart_rate: float*, *proposed_heart_rate: int*, *time_delta: float*, *expected_trackpoints: int*) → float

Calculating the predicted heart rate.

Parameters

- **trimp_delta** (*float*) – the calculated TRIMP difference
- **average_heart_rate** (*float*) – the average heart rate of the interval in beats per minute
- **proposed_heart_rate** (*int*) – the proposed heart rate of the interval in beats per minute
- **time_delta** (*float*) – the time difference in seconds
- **expected_trackpoints** (*int*) – the number of expected trackpoints

Returns
the proposed heart rate

Return type
float

predict_heart_rate() → None

Digital twin algorithm that monitors the athlete performance in the real time and calculates the predicted heart rate.

read_control_data() → tuple

Reading the current heart rate and the interval duration.

Returns
current heart rate, current duration

Return type
tuple[int, float]

3.5.3 GPS Sensor module

class gps_sensor.**GpsSensor**(*gps_path: str = 'sensor_data/gps.txt'*)

Bases: object

Class for working with GPS sensor.

Parameters
gps_path (*str*) – path to file for storing GPS data

get_gps_data() → None

Method for listening the channel for obtaining GPS data from sensor.

Note: Example is based on source code from
https://github.com/adafruit/Adafruit_CircuitPython_GPS

write_gps_data_to_file(*longitude: float*, *latitude: float*, *altitude: float*) → None

Method for writing GPS data to text file.

Parameters

- **longitude** (*float*) – longitude on Earth
- **latitude** (*float*) – latitude on Earth
- **altitude** (*float*) – current altitude

3.5.4 HR Sensor module

class hr_sensor.**HrSensor**(*hr_path='sensor_data/hr.txt'*)

Bases: object

Class for working with HR sensor.

Parameters
hr_path (*str*) – path to file for storing HR data

get_hr_data() → None

Method for listening the channel for obtaining HR data from sensor.

Note: Example is based on source code from
https://github.com/Tigge/openant/blob/master/examples/heart_rate_monitor.py

on_data(*data*) → None

Extracting and writing heart rate data to file.

Parameters
() (*data*) – list that contains heart rate

write_hr_data_to_file(*hr: int*) → None

Method for writing hr data to text file.

Parameters
hr (*int*) – heart rate

3.5.5 Interval Training module

3.5.6 Main Window module

3.5.7 Model module

3.5.8 Simulation module

class simulation.**Simulation**(*hr_path='../sensor_data/hr.txt'*, *gps_path='../sensor_data/gps.txt'*)

Bases: object

Implementation of methods for simulating heart rate and GPS data.

Parameters

- **hr_path** (*str*) – path to the file with heart rates
- **gps_path** (*str*) – path to the file with GPS data

Date:
2021

License:
MIT

simulate_gps() → None

GPS simulation.

simulate_hr() → None

Randomly generate heart rate between 150 and 160 beats per minute.

write_gps_to_file(*lon: float*, *lat: float*, *alt: float*) → None

Write GPS simulation data to the text file.

Parameters

- **lon** (*float*) – longitude on Earth
- **lat** (*float*) – latitude on Earth
- **alt** (*float*) – current altitude

write_hr_to_file(*hr: int*) → None

Write HR simulation data to the text file.

Parameters
hr (*int*) – heart rate

3.5.9 Training Session module

class training_session.**TrainingSession**

Bases: object

Class for tracking training sessions.

add_ascent(*current_elevation: float*) → None

Adding to the total ascent.

Parameters
current_elevation (*float*) – current elevation in meters

add_distance(*distance: float*) → None

Adding to the total distance.

Parameters
distance (*float*) – distance in meters

calculate_time() → None

Method for calculating the current time of the session.

3.5.10 Write log module

class write_log.**WriteLog**

Bases: object

Class for writing log.

static write_interval_training_header(*file: str*)

Method for writing interval training header.

Parameters
file – str path to the log file

static write_interval_training_trackpoint(*file: str*, *digital_twin*)

Method for writing interval training trackpoint.

Parameters

- **file** – str path to the log file
- **digital_twin** – DigitalTwin the Digital Twin

3.6 Contributing to AST-Monitor

First off, thanks for taking the time to contribute!

3.6.1 Code of Conduct

This project and everyone participating in it is governed by the *Contributor Covenant Code of Conduct*. By participating, you are expected to uphold this code. Please report unacceptable behavior to iztok.fister1@um.si.

3.6.2 How Can I Contribute?

Reporting Bugs

Before creating bug reports, please check existing issues list as you might find out that you don't need to create one. When you are creating a bug report, please include as many details as possible in the issue template.

Suggesting Enhancements

Open new issue using the feature request template.

Pull requests

Fill in the pull request template and make sure your code is documented.

3.7 Contributor Covenant Code of Conduct

3.7.1 Our Pledge

We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

We pledge to act and interact in ways that contribute to an open, welcoming, diverse, inclusive, and healthy community.

3.7.2 Our Standards

Examples of behavior that contributes to a positive environment for our community include:

- Demonstrating empathy and kindness toward other people
- Being respectful of differing opinions, viewpoints, and experiences
- Giving and gracefully accepting constructive feedback
- Accepting responsibility and apologizing to those affected by our mistakes, and learning from the experience
- Focusing on what is best not just for us as individuals, but for the overall community

Examples of unacceptable behavior include:

- The use of sexualized language or imagery, and sexual attention or advances of any kind
- Trolling, insulting or derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or email address, without their explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

3.7.3 Enforcement Responsibilities

Community leaders are responsible for clarifying and enforcing our standards of acceptable behavior and will take appropriate and fair corrective action in response to any behavior that they deem inappropriate, threatening, offensive, or harmful.

Community leaders have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, and will communicate reasons for moderation decisions when appropriate.

3.7.4 Scope

This Code of Conduct applies within all community spaces, and also applies when an individual is officially representing the community in public spaces. Examples of representing our community include using an official e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event.

3.7.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported to the community leaders responsible for enforcement at iztok.fister1@um.si. All complaints will be reviewed and investigated promptly and fairly.

All community leaders are obligated to respect the privacy and security of the reporter of any incident.

3.7.6 Enforcement Guidelines

Community leaders will follow these Community Impact Guidelines in determining the consequences for any action they deem in violation of this Code of Conduct:

1. Correction

Community Impact: Use of inappropriate language or other behavior deemed unprofessional or unwelcome in the community.

Consequence: A private, written warning from community leaders, providing clarity around the nature of the violation and an explanation of why the behavior was inappropriate. A public apology may be requested.

2. Warning

Community Impact: A violation through a single incident or series of actions.

Consequence: A warning with consequences for continued behavior. No interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, for a specified period of time. This includes avoiding interactions in community spaces as well as external channels like social media. Violating these terms may lead to a temporary or permanent ban.

3. Temporary Ban

Community Impact: A serious violation of community standards, including sustained inappropriate behavior.

Consequence: A temporary ban from any sort of interaction or public communication with the community for a specified period of time. No public or private interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, is allowed during this period. Violating these terms may lead to a permanent ban.

4. Permanent Ban

Community Impact: Demonstrating a pattern of violation of community standards, including sustained inappropriate behavior, harassment of an individual, or aggression toward or disparagement of classes of individuals.

Consequence: A permanent ban from any sort of public interaction within the community.

3.7.7 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 2.0, available at https://www.contributor-covenant.org/version/2/0/code_of_conduct.html.

Community Impact Guidelines were inspired by Mozilla's code of conduct enforcement ladder.

For answers to common questions about this code of conduct, see the FAQ at https://www.contributor-covenant.org/faq. Translations are available at https://www.contributor-covenant.org/translations.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search