
sevivi Documentation

Release 0.1.0

Arne Herdick

Oct 22, 2021

CONTENTS

1	Features	3
2	Installation	5
3	Usage	7
4	Template Credits	9
5	Installation	11
5.1	Stable release	11
5.2	From sources	11
6	Usage	13
6.1	Installation	13
6.2	Command-Line Usage	13
6.3	Usage as a library	16
7	Background	19
7.1	Concept	19
7.2	Concept Examples	21
8	Reference	23
8.1	sevivi.main	23
9	Contributing	25
9.1	Types of Contributions	25
9.2	Get Started!	26
9.3	Pull Request Guidelines	27
9.4	Tips	27
9.5	Deploying	27
10	Credits	29
10.1	Contributors	29
11	History	31
11.1	X.Y.Z (YYYY-MM-DD)	31
12	sevivi readme	33
12.1	Features	33
12.2	Installation	33
12.3	Usage	33
12.4	Template Credits	34

sevivi is a python package and command line tool to generate videos of sensor data graphs synchronized to a video of the sensor movement.

- Free software: MIT license
- Documentation: <https://sevivi.readthedocs.io>.

FEATURES

- TODO

INSTALLATION

Install the package from pypi:

```
pip install sevivi
```


USAGE

Check out the usage documentation, please! If you just want to render a video to get started, keep reading. After you have downloaded the repository, you can use our test data. Run the following:

```
git clone git@github.com:your_name_here/sevivi.git
cd sevivi/
pip install sevivi
sevivi test-files/test-data-configs/kinect_sync_squatting.toml
```

If you want to use sevivi as a library, you can copy-paste the following code into your project. You should download our test files for this to run immediately.

```
import pandas as pd

from sevivi.config import RenderConfig, ManuallySynchronizedSensorConfig
from sevivi.image_provider import GraphImageProvider, VideoImuCaptureAppImageProvider
from sevivi.video_renderer import VideoRenderer

video_provider = VideoImuCaptureAppImageProvider(
    video_path="test_files/videos/imu_sync.mp4",
    imu_pb_path="test_files/sensors/video_imu_capture_app/video_meta.pb3"
)

# create a GraphImageProvider for each of your sensors
sensor_config = ManuallySynchronizedSensorConfig()
sensor_config.offset_seconds = 0.0
sensor_config.name = "Human-Readable Name"
sensor_config.path = "test_files/sensors/imu_synchronization/camera_imu.csv.gz"
data = pd.read_csv(sensor_config.path, index_col=0, parse_dates=True)
graph_image_provider = GraphImageProvider(data, sensor_config)

# render the video
renderer = VideoRenderer(RenderConfig(), video_provider, [graph_image_provider])
renderer.render_video()
```


TEMPLATE CREDITS

This package was created with [Cookiecutter](#) and the [pyOpenSci/cookiecutter-pyopensci](#) project template, based off [audreyr/cookiecutter-pypackage](#).

INSTALLATION

5.1 Stable release

To install `sevivi`, run this command in your terminal:

```
$ pip install sevivi
```

This is the preferred method to install `sevivi`, as it will always install the most recent stable release.

If you don't have `pip` installed, this [Python installation guide](#) can guide you through the process.

5.2 From sources

The sources for `sevivi` can be downloaded from the [Github repo](#).

You can either clone the public repository:

```
$ git clone git://github.com/hpi-dhc/sevivi
```

Or download the [tarball](#):

```
$ curl -OL https://github.com/hpi-dhc/sevivi/tarball/master
```

Once you have a copy of the source, you can install it with [poetry](#):

```
$ poetry install
```


USAGE

Sevivi (SEnsor VIdeo VIualizer) has two modes of operation. It can either work as a command-line tool, if you use supported input data, or you can use sevivi as a library, instantiating the input data manually and improving the results by supplementing, for example, the algorithm used to synchronize the sensors to the video.

6.1 Installation

Simply run

```
pip install sevivi
```

to install sevivi.

6.2 Command-Line Usage

Installing sevivi makes the sevivi CLI available as a python program. sevivi has the following command line syntax:

```
usage: sevivi [-h] [--output TARGET_FILE_PATH] config [config ...]

positional arguments:
  config                Configuration files. Later files overwrite earlier ones. Only
↳ the last video section is used. All given sensor configs are interpreted as a list,
↳ rather
                        than overwriting earlier configuration.

optional arguments:
  -h, --help            show this help message and exit
  --output TARGET_FILE_PATH
                        Set the output file. Overwrites config value.
```

As described above, sevivi supports multiple config files. Each later config file overwrites the configuration from previous configuration files. There is one exception for this: the configuration for sensor data sources. Every sensor data source is appended to the list of all sources instead.

If you have downloaded the repository, you can use our test data to get used to working with sevivi. Run the following:

```
git clone git@github.com:your_name_here/sevivi.git
cd sevivi/
```

(continues on next page)

(continued from previous page)

```
pip install sevivi
sevivi test-files/test-data-configs/kinect_sync_squatting.toml
```

Configuration files are comprised of general options that apply to the entire tool, a video configuration section, and a number of sensor source configuration sections.

6.2.1 Complete Example

More options and descriptions for each section can be found below. This is a single example for a full configuration file that can render one of the examples included in the repository.

```
target_file_path = "camera_sevivi.avi"

[[video]]
type = "kinect"
path = "test_files/videos/joint_synchronization_walking.mp4"
skeleton_path_3d = "test_files/skeletons/joint_synchronization_walking/positions_3d.csv.
→gz"

[[sensor]]
type = "joint-synced"
sensor_sync_column_selection = ["Acc_X", "Acc_Y", "Acc_Z"]
camera_joint_sync_column_selection = ["SPINE_CHEST (x)", "SPINE_CHEST (y)", "SPINE_CHEST_
→(z)"]
path = "test_files/sensors/joint_synchronization_walking/LF.csv.gz"

[[sensor]]
type = "joint-synced"
sensor_sync_column_selection = ["Acc_X", "Acc_Y", "Acc_Z"]
camera_joint_sync_column_selection = ["SPINE_CHEST (x)", "SPINE_CHEST (y)", "SPINE_CHEST_
→(z)"]
path = "test_files/sensors/joint_synchronization_walking/RF.csv.gz"
```

6.2.2 Common Options

These options are common to the whole tool, and can always be set. They must be in the root section, as shown in the complete example.

```
# May be 'vertical' to put all sensor graphs below the video, or 'horizontal' to put the_
→sensor graphs left and right of
# the center of the video
stacking_direction = "horizontal"
# currently, the only supported plotting method is moving_vertical_line
plotting_method = "moving_vertical_line"
# Set the four character codec name to save in the avi container
fourcc_codec = "MJPG"
# Set the target video file path
target_file_path = "./sevivi.mp4"
```

6.2.3 Video Options

We support different video input formats, each specified by its unique `type`. You can either add your video as a separate config file to the CLI call (makes it easy to switch out) or add the section into one main config file.

- Example video section for videos without associated synchronization data:

```
[[video]]
# source video file
path = "test_files/raw.mkv"
# type is "raw" as this video doesn't have any data associated with it
type = "raw"
```

- Example video section for videos from an Azure Kinect with exported skeleton data:

```
[[video]]
# path to the input video
path = "test_files/kinect.mkv"
# skeleton data. skeleton data can be created by @justamad
skeleton_path_3d = "test_files/kinect.csv.gz"
# azure kinect config type
type = "kinect"
```

- Example video section for videos created with VideoImuCapture:

```
[[video]]
# path to the input video
path = "test_files/videos/imu_sync.mp4"
# specify the path to the IMU data; this is a protobuf file from the VideoImuCapture app
imu_path = "test_files/sensors/video_imu_capture_app/video_meta.pb3"
# config type to specify this is from the VideoImuCapture app
type = "videoImuApp"
```

- Example video section for videos that have IMU data associated in some other way:

```
[[video]]
# video file path
path = "test_files/videos/imu_sync.mp4"
# specify this is a video with IMU data attached
type = "imu"
# specify the path to the IMU data
imu_path = "test_files/kinect_imu.csv.gz"
```

6.2.4 Sensor Options

Last but not least, the input sensors need to be specified. Each sensor can be added by adding another `[[sensor]]` block. Some options are common to all sensors:

```
[[sensor]]
# Only data after this time (measured in unshifted sensor time) is included
start_time = "00:00:00.000000"
# Only data before this time (measured in unshifted sensor time) is included
end_time = "00:00:01.000000"
```

Again, a number of types with specific options are available:

- Manual Synchronization – this can be useful to, e.g., synchronize a sensor that doesn't include the right modality to be synchronized against the camera

```
[[sensor]]
# how many seconds into the future the data from this sensor should be move to align its.
↪start with the video start.
# this value can be negative.
offset_seconds = 123.4
# A manually synchronized sensor
type = "manually-synced"
# path to the data of this sensor
path = "test_files/manual_imu.csv.gz"
```

- Camera IMU synchronization: This sensor configuration can be used to synchronize sensors by their data to camera data

```
[[sensor]]
# This is a sensor synchronized to the IMU of the camera.
type = "camera-imu-synced"
# Select the columns **from the sensor** that should be aligned to the columns **from.
↪the camera**
sensor_sync_column_selection = ["AccX", "Accel Y"]
# Select the columns **from the camera** that should be aligned to the columns **from.
↪the sensor**
camera_imu_sync_column_selection = ["AccX", "Accel Y"]
# Specify the path to the data
path = "test_files/sensor_imu.csv.gz"
```

6.3 Usage as a library

To use sevivi as a library, which is useful to change implementations, add some, or just because you don't feel like writing configuration files, keep in mind that the main interface to sevivi is the `VideoRenderer` class. Once you have created a `VideoRenderer` instance, you can call the `render_video` method to start writing the result.

To create the instance, you need to provide a `VideoImageProvider` subclass and a `GraphImageProvider` for each sensor you want to add to the video.

The following `VideoImageProvider` subclasses are available out of the box:

- `AzureProvider`
- `PlainVideoImageProvider`
- `ImuCameraImageProvider`
- `VideoImuCaptureAppImageProvider`

As an example, to manually create a `VideoRenderer` that renders one of the examples provided in the repository, the following code can be used:

```
import pandas as pd

from sevivi.config import RenderConfig, ManuallySynchronizedSensorConfig
from sevivi.image_provider import GraphImageProvider, VideoImuCaptureAppImageProvider
```

(continues on next page)

(continued from previous page)

```
from sevivi.video_renderer import VideoRenderer

video_provider = VideoImuCaptureAppImageProvider(
    video_path="test_files/videos/imu_sync.mp4",
    imu_pb_path="test_files/sensors/video_imu_capture_app/video_meta.pb3"
)

# create a GraphImageProvider for each of your sensors
sensor_config = ManuallySynchronizedSensorConfig()
sensor_config.offset_seconds = 0.0
sensor_config.name = "Human-Readable Name"
sensor_config.path = "test_files/sensors/imu_synchronization/camera_imu.csv.gz"
data = pd.read_csv(sensor_config.path, index_col=0, parse_dates=True)
graph_image_provider = GraphImageProvider(data, sensor_config)

# render the video
renderer = VideoRenderer(RenderConfig(), video_provider, [graph_image_provider])
renderer.render_video()
```


BACKGROUND

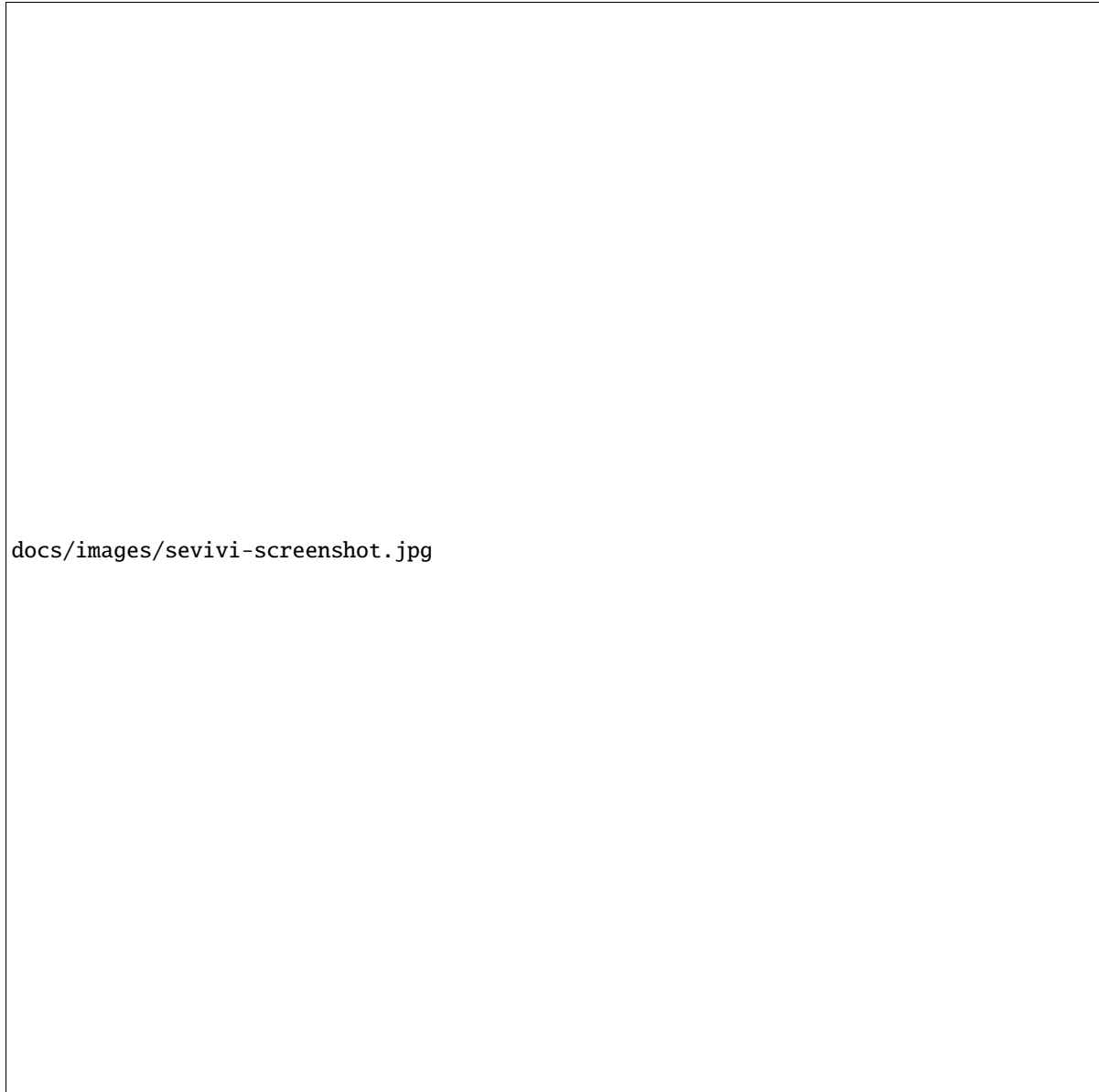
This tool has been born out of research regarding human motion analysis with acceleration- and gyroscope sensors. During our trials, we had kinect cameras running to gather movement skeletons from participants. When analyzing trials, we often found the need to see how specific movements influenced the values from the acceleration and gyroscope sensors. However, we were unable to find a tool capable of showing the sensor data stream at the same time as the video from the kinect cameras.

A first prototype showed great promise in the usefulness of such a tool, allowing us to easily create repetition detection algorithms for our squat exercise trials.

As we could not find a tool like ours before, we decided to polish our research about it and create an open-source version!

7.1 Concept

Sevivi (SEnsor VIDEO VIualizer) always uses exactly one video source. Around the video, the data from any number of sensors can be shown. An exemplary result can be seen below. Data from multiple axes of the same sensor (e.g., the 3 accelerometer axes of the ankle sensor) is grouped into the same graph.



docs/images/sevivi-screenshot.jpg

To achieve the goal of having the sensor data playback be synchronous to the video playback, some method of synchronization is required. Sevivi solves this problem by requiring data that is recorded on the same clock as the camera frames. For a Kinect, this might be the tracked skeleton, for a smartphone, it could be the integrated IMU.

As we can now assume we have data synchronous to the video, our synchronization problem suddenly becomes much simpler. We only need to align each sensor to the data from the video source, and voilà, we can simply render a graph of the data.

Let's take this very abstract information and translate it to understandable examples in the next section.

7.2 Concept Examples

While in theory sevivi could be use to combine any type of sensors, the most common use case is to use acceleration sensors. This is because many acceleration can be derived from skeleton tracking, and many wearable sensors include an acceleration sensor, leading to the necessary combination of two acceleration streams being available.

7.2.1 Kinect + IMU Sensors

Kinect cameras record an RGB image and a depth image. With both of these data streams, it is possible to achieve very good human skeleton tracking. The result of this tracking is a stream of positions for each of the 24 joints the kinect tracks. Now, let's assume we have collected data from an IMU on the wrist, and want to show its values during specific movements.

Our goal is to align the acceleration recorded by the wrist's IMU sensor with the wrist joint from the kinect. As the kinect records positions, we need to calculate the second derivation of this data to arrive at acceleration as well. Now, we can simply find the peak in the cross-correlation between the two data streams, and we know how much we need to shift the data from the sensor in time to align it with the acceleration data from the kinect.

7.2.2 Smartphone IMU + IMU Sensors

As you probably know, every modern smartphone has an IMU. There exist various apps that allow recording the IMU together with a video from the camera. Sevivi specifically has support for the [VideoIMUCapture](#) app. By shaking the camera together with the IMU sensors, distinctive spikes are recorded in the acceleration data. These spikes can be used to align the camera IMU with the IMU sensors. After shaking, the IMU sensors can be attached to whatever is to be tracked. Again, we have the two acceleration streams to synchronize on.

7.2.3 Manual Synchronization

Sevivi also allows to manually set the offset between camera and your sensor data. This is useful in case your desired sensor or camera has no data stream to synchronize on.

REFERENCE

8.1 sevivi.main

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

9.1 Types of Contributions

9.1.1 Report Bugs

Report bugs at <https://github.com/hpi-dhc/sevivi/issues>.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

9.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” and “help wanted” is open to whoever wants to implement it.

9.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with “enhancement” and “help wanted” is open to whoever wants to implement it.

9.1.4 Write Documentation

sevivi could always use more documentation, whether as part of the official sevivi docs, in docstrings, or even on the web in blog posts, articles, and such.

9.1.5 Submit Feedback

The best way to send feedback is to file an issue at <https://github.com/hpi-dhc/sevivi/issues>.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

9.2 Get Started!

Ready to contribute? Here's how to set up *sevivi* for local development.

1. Fork the *sevivi* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/sevivi.git
```

3. Install all dependencies after installing *poetry* <<https://python-poetry.org/docs/>>:

```
$ cd sevivi/  
$ poetry install  
$ pre-commit install
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

5. When you're done making changes, check that your changes pass the tests and linters and that the docs can be built:

```
$ py.test  
$ pre-commit run --all-files  
$ cd docs && make html
```

7. Commit your changes and then push your branch to GitHub:

```
$ git add .  
$ git commit -m "Your detailed description of your changes."  
$ git push origin name-of-your-bugfix-or-feature
```

8. Submit a pull request through the GitHub website.

9.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in `README.rst`.
3. The pull request should work for Python 3.7, 3.8 and 3.9, and for PyPi. This will be verified within the PR.

9.4 Tips

To run a subset of tests:

```
$ py.test tests.test_sevivi
```

9.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed, including an entry in `HISTORY.rst` and an update of the old version code in `docs/conf.py` and `pyproject.toml`.

GitHub will then deploy to PyPI if tests pass.

10.1 Contributors

- Justin Albert <justin.albert@hpi.de>
- Arne Herdick <arne.herdick@hpi.de>

11.1 X.Y.Z (YYYY-MM-DD)

- TODO

SEVIVI README

sevivi is a python package and command line tool to generate videos of sensor data graphs synchronized to a video of the sensor movement.

- Free software: MIT license
- Documentation: <https://sevivi.readthedocs.io>.

12.1 Features

- TODO

12.2 Installation

Install the package from pypi:

```
pip install sevivi
```

12.3 Usage

Check out the usage documentation, please! If you just want to render a video to get started, keep reading. After you have downloaded the repository, you can use our test data. Run the following:

```
git clone git@github.com:your_name_here/sevivi.git
cd sevivi/
pip install sevivi
sevivi test-files/test-data-configs/kinect_sync_squatting.toml
```

If you want to use sevivi as a library, you can copy-paste the following code into your project. You should download our test files for this to run immediately.

```
import pandas as pd

from sevivi.config import RenderConfig, ManuallySynchronizedSensorConfig
from sevivi.image_provider import GraphImageProvider, VideoImuCaptureAppImageProvider
from sevivi.video_renderer import VideoRenderer

video_provider = VideoImuCaptureAppImageProvider(
    video_path="test_files/videos/imu_sync.mp4",
    imu_pb_path="test_files/sensors/video_imu_capture_app/video_meta.pb3"
)

# create a GraphImageProvider for each of your sensors
sensor_config = ManuallySynchronizedSensorConfig()
sensor_config.offset_seconds = 0.0
sensor_config.name = "Human-Readable Name"
sensor_config.path = "test_files/sensors/imu_synchronization/camera_imu.csv.gz"
data = pd.read_csv(sensor_config.path, index_col=0, parse_dates=True)
graph_image_provider = GraphImageProvider(data, sensor_config)

# render the video
renderer = VideoRenderer(RenderConfig(), video_provider, [graph_image_provider])
renderer.render_video()
```

12.4 Template Credits

This package was created with [Cookiecutter](#) and the [pyOpenSci/cookiecutter-pyopensci](#) project template, based off [audreyr/cookiecutter-pypackage](#).