

PALMS

USER MANUAL

Date: 7-Oct-20

Current version of the manual might not reflect the latest state of PALMS software

Contents

Intro.....	3
Setup the environment.....	4
Open project in PyCharm.....	4
Create virtual environment (venv).....	4
Install dependencies from requirements.txt	4
Annotating workflow	5
Create Annotation Config	5
Create a custom database class.....	6
Run the tool, select database and file	6
Annotate the data.....	7
Annotation mode.....	7
Partitioning mode	9
Browse mode	9
Epoch mode	10
Settings.....	10
Controls.....	10
Save data and close the tool	12
Appendix	13
Database creating walk-through.....	13
Annotation Config file	14
Final remarks.....	15
Shortcuts	15
Configuration	16
Portable (executable) version	17
Future work.....	Error! Bookmark not defined.

Intro

PALMS software tool serves the purpose of annotation of time-series data. Its main features are:

- Multi-type events\fiducials annotation, e.g. it is possible to annotate the whole ECG complex, not only R-peaks (annotation mode)
- Continuous-time regions partitioning, e.g. mark artifacts within the signal (partition mode)

It is made:

- Generic: file format and signal type independent, a user himself defines how the data is fetched
- Fast and user-friendly: PyQt based GUI ensures fast browsing through data and convenient use of mouse and keyboard

Setup the environment

In this section we explain how to setup and run PALMS on **Windows**, using clean **Python 3.6** interpreter and **PyCharm IDE**. Other configurations, including various python versions (>3.5) might also work, but it is not guaranteed.

Python 3.6.8. executables can be downloaded from [here](#). We recommend using clean python, rather than preconfigured software packages as Anaconda, as they might make some subtle changes to your PC configuration (environmental paths) and then it is hard to track the issue.

PyCharm community edition is available [here](#). PyCharm is free and powerful IDE for Python, but one can use another IDE or run the tool from the command line.

Once python is installed, we need to set a python environment for the project and install required packages.

[Open project in PyCharm](#)

[Create virtual environment \(venv\)](#)

Virtual environment is preferable (in comparison with the base python interpreter), because:

- Separation of environments per project allows only install packages which are actually used
- No need to write to the C:\ drive, where python is usually installed
- One can copy the venv to another PC and it will work instantly (Python still has to be installed).

NB: highly not recommended to keep venv in git repository, as it is heavy and git is not made for this purpose.

[Install dependencies from requirements.txt](#)

Annotating workflow

The way to use the tool is very straightforward:

1. [Create the Annotation Config csv file](#)
2. [Create a database class](#)
3. [Run the tool and select database/file](#)
4. [Annotate](#)
5. [Save and close the tool](#)

Only the second step requires python code modification. It is done once per database.

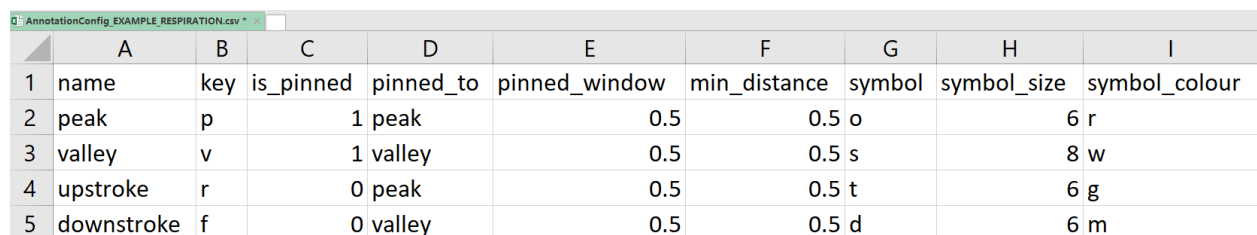
[Create Annotation Config](#)

We provide two example databases and their corresponding Annotation configurations in `AnnotationConfig_EXAMPLE_PPG.csv` and `AnnotationConfig_EXAMPLE_RESPIRATION.csv`. In case you just want to check how the tool works or someone else already created all necessary files, you can reuse that and go directly to the [next](#) step. In addition, `config/videos/` contain screen recordings with explanations.

In the annotation configuration file, one defines:

- which events\fiducials he wants to annotate (name)
- how should annotations be presented in the plot (marker, size, colour)
- annotation controls (keyboard keys and pinning options)

Some of the options can be reconfigured on the fly. More detailed explanation of each column is given in [Appendix](#). Path to this file has to be mentioned in database class `annotation_config_file` field.



	A	B	C	D	E	F	G	H	I
1	name	key	is_pinned	pinned_to	pinned_window	min_distance	symbol	symbol_size	symbol_colour
2	peak	p	1	peak	0.5	0.5	o	6	r
3	valley	v	1	valley	0.5	0.5	s	8	w
4	upstroke	r	0	peak	0.5	0.5	t	6	g
5	downstroke	f	0	valley	0.5	0.5	d	6	m

Figure 1 Screenshot of the `AnnotationConfig_EXAMPLE_RESPIRATION.csv`

Create a custom database class

We provide two example databases and their descriptions in `EXAMPLE_PPG.py` and `EXAMPLE_RESPIRATION.py`. In case you just want to check how the tool works or someone else already created the class for the data you intend to use, you can reuse that and go directly to the [next](#) step. In addition, `config/videos/` contain screen recordings with explanations.

To ensure flexibility in use-cases, data formats and signals, one needs to provide minimal description of the database. This is done by creating a python class, which inherits from `Database` class (`DatabaseHandler.py`). Newly created class name also becomes the database name. `DatabaseHandler.py` and `EXAMPLE_PPG.py` contain detailed inline comments explaining the class fields. Walk-through instructions also available in [Appendix](#).

Run the tool, select database and file

Run the `__main__.py` located in `PALMS/logic/`

After the tool starts a popup window appears. The combobox contains a list of databases. It is generated based on classes found in `PALMS/logic/databases`. After selecting a database, you will see the files found according to selected database description. *NB: that database description does not ensure presence of the data, so that the file list can be empty.*

Here we select the `EXAMPLE_PPG` database and the only available file, then press Select or Enter.

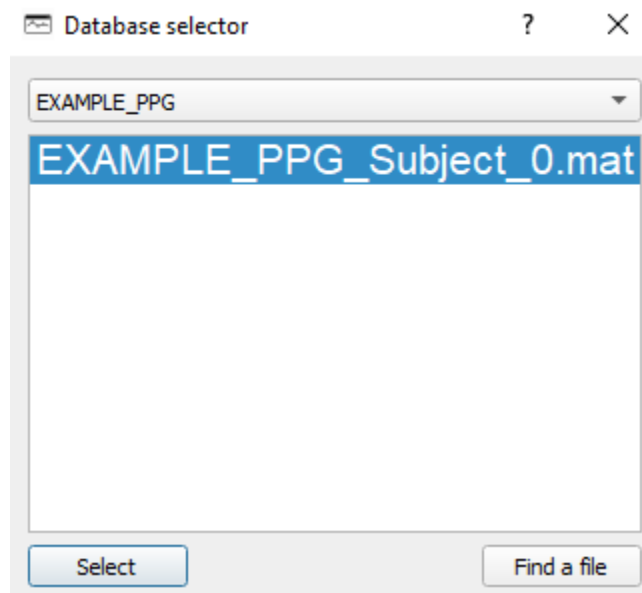


Figure 2 Database selection dialog

Annotate the data

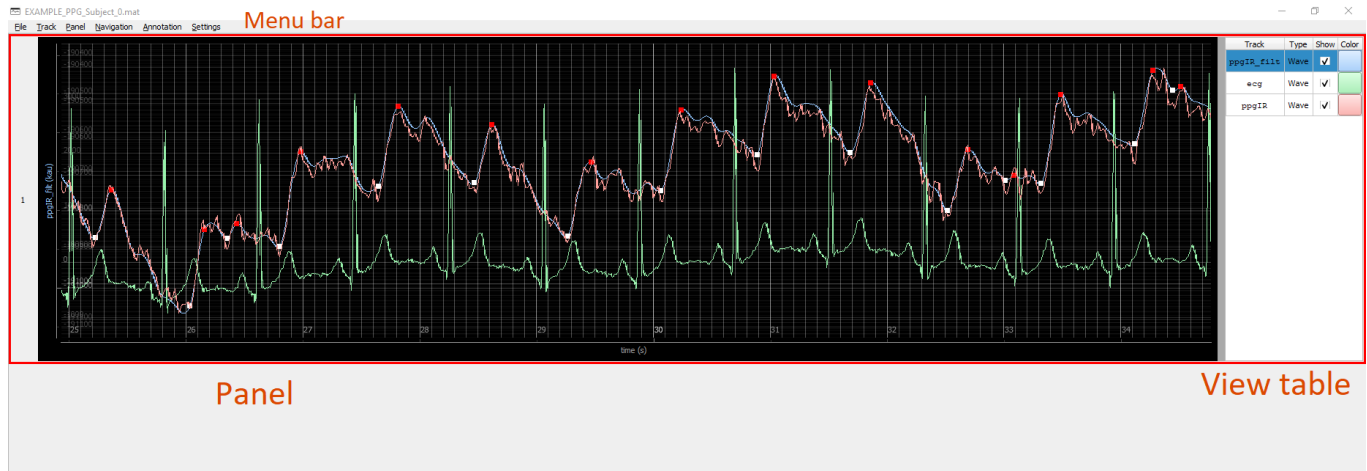


Figure 3 PALMS main window

Once the main GUI containing signals appeared (Figure 3), one can start the annotation process. Current version of PALMS has three modes:

- Annotation mode for events annotation (partitions are not visible)
- Partitioning mode for custom region creation (annotations are hidden)
- Browse mode for observing annotations and partitions (neither can be modified)
- Epoch mode for quality labeling.

Annotation mode

[Annotation configuration dialog](#)

Some of the settings defined in the annotation configuration file is possible to modify from within the tool. If current mode is *Annotation mode*, then *Annotation->Config* menu (Figure 4) is enabled in the top menu (default shortcut is *Alt+Shift+C*). Changes take place immediately.

Annotation Configuration

	name	key	is_pinned	pinned_to	pinned_window	min_distance	symbol	symbol_size	symbol_colour
1	peak	p	<input checked="" type="checkbox"/>	peak volume_filt	0.50 s	0.50 s	o	6	r
2	valley	v	<input checked="" type="checkbox"/>	valley volume_filt	0.50 s	0.50 s	s	8	w
3	upstroke	r	<input type="checkbox"/>	peak volume_filt	0.50 s	0.50 s	t	6	g
4	downstroke	f	<input type="checkbox"/>	valley volume_filt	0.50 s	0.50 s	d	6	m

Load Csv File!

Write Csv File!

Figure 4 Annotation configuration dialog

Place annotations

To annotate an event, one should first press a keyboard key corresponding to that event\fiducial (as set in the configuration) and then *LeftMouseClicked* at a position of the event. Once the annotation is done, corresponding timestamp, x-axis point index and signal amplitude are saved to the database, and a new marker appears on the plot.

Remark 1: If one wants to annotate the first event\fiducial in the list of all events (Figure 4, first row), he\she can omit the keyboard button and use *LeftMouseClicked* only.

Remark 2: If the “[sticky fiducial](#)” options enabled, one also does not have to press the button, only the *LeftMouseClicked*.

Remark 3: It is only possible to annotate the main signal (defined in the database class). If another signal is selected in the table view and one tries to do the annotation, a warning dialog window will popup (Figure 5). The user has to select the main signal in the table view and repeat the annotation action.

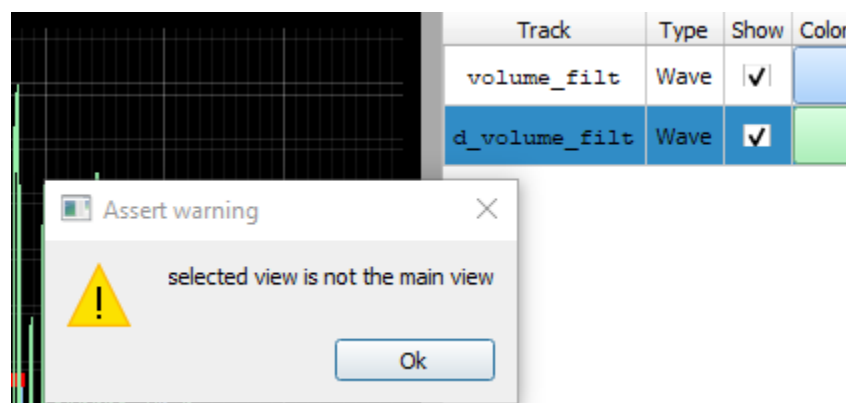


Figure 5 Warning dialog

[Remove annotation](#)

Removing an annotation is done by *RightMouseClicked*. Then the closest (in terms of time) annotated event is deleted from the database and from the plot. The tool doesn't consider which type annotation it is (except if the "[sticky fiducial](#)" option enabled).

Partitioning mode

This mode is created to mark continuous chunks of data and allocate custom names to them. It is useful to label and later chop artifacts from the data or mark beginning\end of useful data in the whole signal. On Figure 4, regions for normal, deep and fast breathing are labeled, as well as movement artifact and end of the signal.

New partition is created by *CTRL+LeftMouseClicked* and deleted by *CTRL+RightMouseClicked*. Existing partition or its borders can be shifted\dragged by *SHIFT+LeftMouseDown*. Batch annotation delete under a partition is done by *SHIFT+RightMouseClicked*.

Partitions can not overlap (all partition limits are adjusted every time a new instance is created or existing instance position change is detected).

Remark: Partition mode is not suitable for complete signal labeling epoch-by-epoch (although it is possible to fully cover it with a number of non-overlapping partitions).

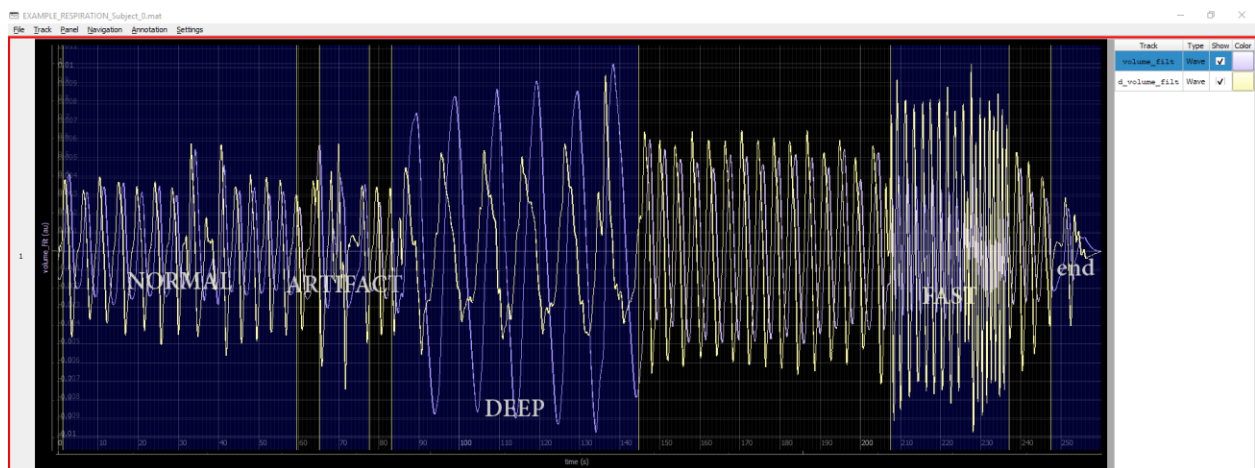


Figure 6 Partitioning mode

Browse mode

Browse mode activates automatically, when neither Annotation nor Partitioning mode are enabled. In this mode user can observe signals, annotations and partitions. It is not possible to modify

any of them, thus mouse clicks and keyboard actions will not make any effect, except the default moving\zooming.

Epoch mode

Documentation incomplete

Epoch mode adds the possibility to annotate chunks of data (in non-overlapping windows) with categorical labels (e.g. 'good', 'low'). More info on configuration is available in *config/EpochConfig/*.

Settings

There is a number of settings available under *Settings* menu:

- *Show cursor* enables real-time cursor position output. Timestamp, index and amplitude correspond to currently selected track in the View table.
- *Autoscale Y-axis* controls whether the signals y-range can will change during the zooming to fit only currently visible part of the signal. This option is essential, when the track has large peak-to-peak amplitude changes or baseline shifts.
- *Save tracks with data* defines whether original signals will be saved together with the annotations\partitions. File size can increase a lot, but it is possible to keep all data in one place.
- *Overwrite .h5 files if any* enables saved files being overwritten without warning. In the opposite case, another file with modified name is created.

Controls

Following control options are available

- *Track->Delete*: any track except the main can be deleted, but it is also possible to disable its plot (*Show* checkbox in the View table)
- *Panel->various operations*: it is possible to make the panel larger or smaller, also create other panels and move tracks around (via Table view). **Although, GUI part of the tool supports operations with multiple panels (and it may be convenient to move some signals to another panel), annotation logic can still be buggy when using multiple panels. Be careful, when using this, as signal might not zoom in synchronously or annotations might not work correctly.**
- *Navigation* menu contains self-explanatory controls which duplicate mouse operations.
- *Table view context menu*: by *RightMouseClicked* on a track in the Table view user can add other plots to the panel (Figure 7):
 - o from the database tracks as set in the corresponding Database class;

- 1st and 2nd derivatives;
- RR-interval (1st derivative) plot of the annotation. This is useful when one expects fiducials to be approximately periodical (e.g. ECG) and wants to check whether annotations show this behavior. Note, that given plot is a snapshot of the annotation and is not updated with new information. To see updated RR-intervals plot, one has to create a new RR-interval plot.

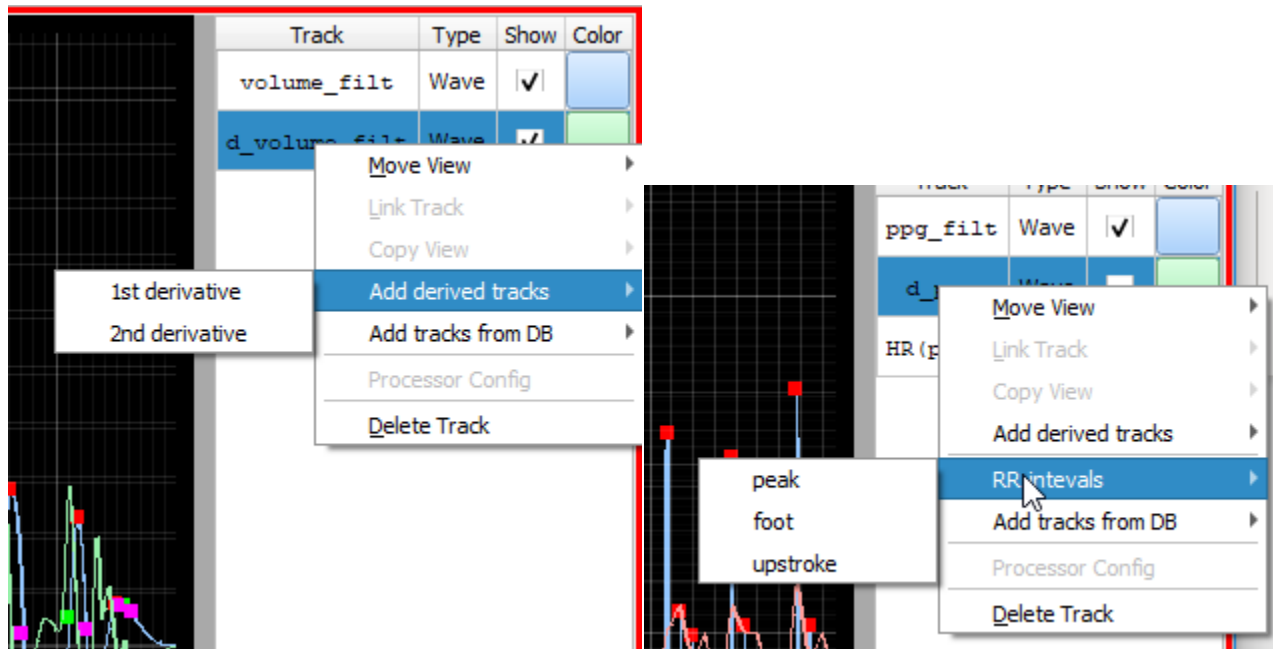


Figure 7 Table view context menu options

- *Sticky fiducial mode*: as stated in Place annotations, in order to set label for a secondary fiducial, user has to first press a key on the keyboard. It might be inconvenient if your way of work is to firstly annotate all peaks, then all valleys, then all upstrokes and downstrokes. This option allows temporarily “stick” the key and annotate any fiducial as if it is the main one. The option (Figure 8) is available in *Annotation->“Sticky” fiducial* menu (default shortcut *Alt+Shift+V*). Note that if the option is activated, then also *RightMouseClicked* will delete the **nearest selected fiducial**, not the nearest any fiducial.

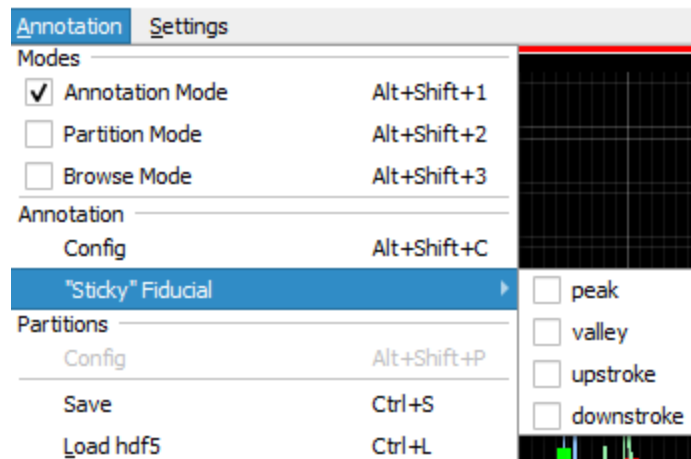


Figure 8 Sticky fiducial menu

Save data and close the tool

By *Annotation->Save* (default *Ctrl+S*) annotations, partitions and (*Settings*) all tracks are saved as .h5 files (HDF5 data format). Save location is as defined in the database class (*self.output_folder*, default: PALMS top-level folder).

After that the tool can be closed. If another file has to be annotated user can use *File->Restart* action. The database selection dialog will pop up.

Appendix

Database creating walk-through

1. Create a new *.py* file in the *logic/databases/* folder. The name of the file is not affecting anything, but for consistency call it same way your database is called. In this section we refer to this file as to *new_database.py*
2. Copy the contents of *EXAMPLE_PPG.py* to *new_database.py*
3. Rename the class name to *new_database*
4. In the *__init__()* method: make changes according to inline comments
5. Implement *get_data()* method
 - a. Run base class *get_data()* to initialize new fields
 - b. Describe how to get all necessary information from the files in your database
 - c. Create new signals or modify existing data
 - d. Create Wave class instances
- e. Run parent's *test_database_setup()* method to check database setup correctness
6. (Optionally) Implement *set_annotation_data()* method
 - a. One can just add *return* command to implement this method, if you want to start annotating from scratch: clean signals, no initially guesses for annotations
 - b. One can search for existing annotations in *self.existing_annotations_folder* as it is done in examples
7. (Optionally) Implement *save()* and *load()* methods if changes to base methods required

Annotation Config file

Name

fiducial name (1 word): peak, valley, foot, blink, step, etc.

Key

keyboard key (one letter) used to distinguish which fiducial (if > 1) user wants to annotate; Corresponding keyboard button has to be pressed before *LeftMouseClicked* in Annotation mode;

If nothing is pressed, the first fiducial in the list is assumed;

is_pinned

1 or 0;

sets whether the actual click position is used for annotation or adjustments are desired;

If 1: the annotation is set to the nearest %pinned_to% in the signal of choice;

can be changed during runtime

pinned_to

peak' or 'valley';

if %is_pinned% is set to 1, annotation will be adjusted to the nearest 'peak' or 'valley' in the signal;

after starting PALMS tool, one can use also other signals from the database, derivatives, etc.;

can be changed during runtime

pinned_window

window in [sec] within which the %pinned_to% is searched for;

can be changed during runtime

min_distance

minimum allowed distance between two annotations of this fiducial in [s];

helps to avoid accidental double clicks when annotating;

if another annotation already exists within %min_distance% the new one is skipped;

can be changed during runtime

symbol

Annotation marker to be seen in the plot; One of: 'o','t','t1','t2','t3','s','p','h','+','star','d';

google 'pyqtgraph symbols'

symbol_size

Annotation marker size

symbol_colour

Annotation marker colour; One of: r, g, b, c, m, y, k, w

Final remarks

Shortcuts

Many GUI actions are assigned a shortcut. If the chosen shortcut is not optimal, one can modify it in *config/shortcuts.json* (Figure 9). Shortcuts info pop-up window is available from the GUI by pressing F1.

```
"restart": "Ctrl+Alt+N",
"remove_track": "Ctrl+Backspace",
"-----": "-----",
"new_panel": "Ctrl+N",
"close_panel": "Alt+F4",
"move_up": "Ctrl+PgUp",
"move_down": "Ctrl+PgDown",
"increase_height": "Ctrl+=",
"decrease_height": "Ctrl+-",
"toggle_all_views": "Ctrl+Alt+H",
"-----": "-----",
"play_pause": "Space",
"move_left": "Left",
"move_right": "Right",
"goto_start": "Ctrl+Left",
"goto_end": "Ctrl+Right",
"zoom_in": "Up",
"zoom_out": "Down",
"zoom_to_match": "Ctrl+Home",
"zoom_to_fit": "Ctrl+End",
"-----": "-----",
"annotation_mode": "Alt+Shift+1",
"partition_mode": "Alt+Shift+2",
"browse_mode": "Alt+Shift+3",
"annotation_config": "Alt+Shift+C",
"partition_config": "Alt+Shift+P",
"sticky_fiducials_popup": "Alt+Shift+V",
"save": "Ctrl+S",
"load": "Ctrl+L"
```

Figure 9 shortcuts.json

Configuration

Figure 10 gives clarification on which settings can be configured by user.

Settings which are auto updated (orange) have to be changed from the GUI and they will be auto saved once the process is finished and the tool closed.

Not auto updated settings (green) have to be set in *config/config.json* before starting the tool.

Fixed settings (red) are not for the user to change, only for developers.

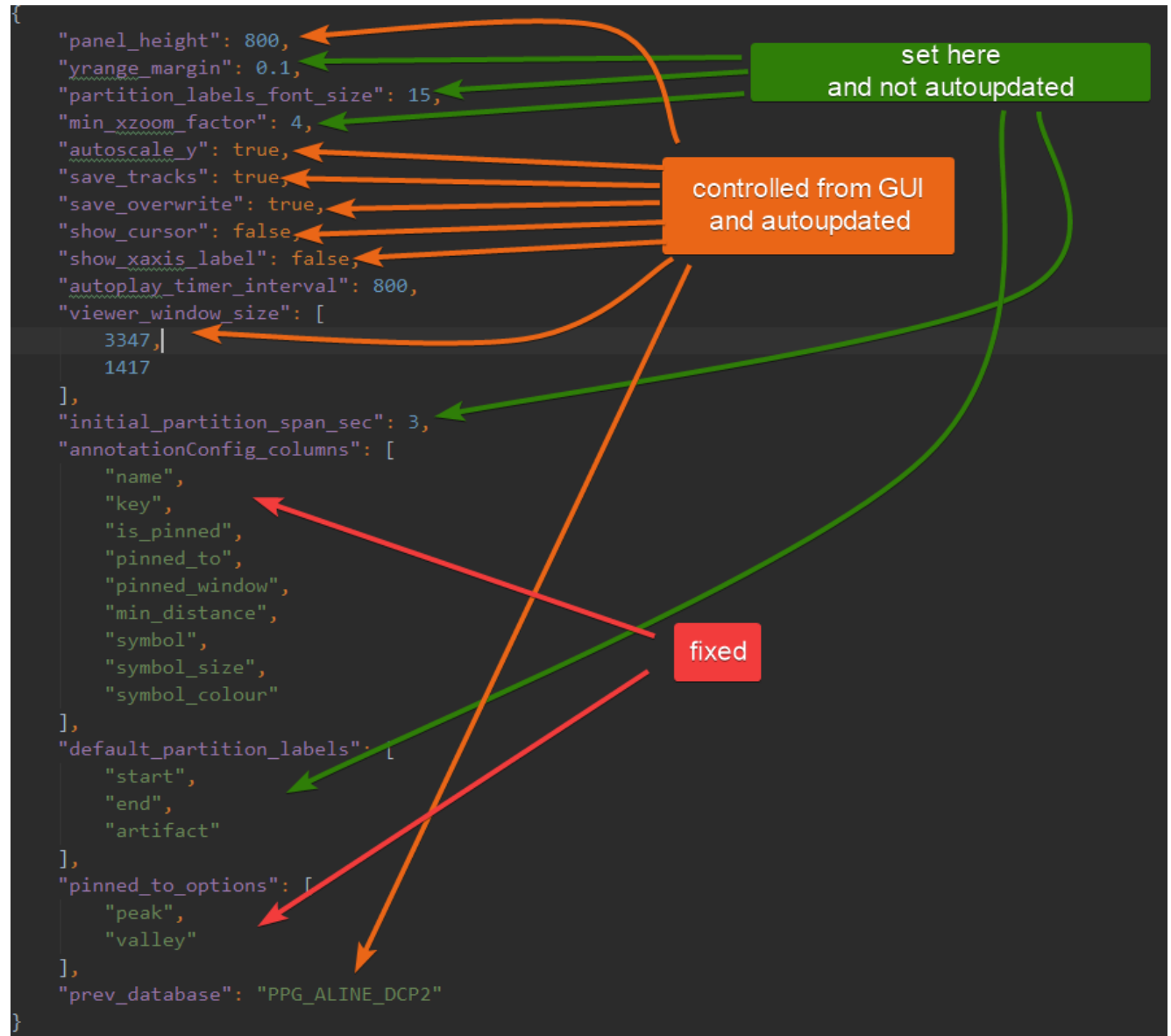


Figure 10 *config.json*

Portable (executable) version

It is recommended to run the tool from conventional Python environment for developing\debugging purposes, but it is possible to use PALMS functionality without python installation or programming experience.

Executable will only work on Windows PC. *PALMS.exe* contains all necessary dependencies to show several examples, but in order to work with an external database, one should create a corresponding *%DATABASE_NAME%.py* file containing *DATABASE_NAME* class inheriting from the *Database* class, that is: PALMS should be given to an annotator-expert in the following folder:

```
annotation_task\  
    |-- data_folder\  
        |-- file_to_annotate  
    |-- PALMS.exe  
    |-- newDatabase.py (contains newDatabase class)  
    |-- config.json
```

The executable itself may be created as follows:

1. Execute in the console: `*pyinstaller --onefile --name PALMS __main__.py*`
2. Modify `*PALMS.spec*` created in the root to add non-python files necessary to run the app:

2.1 Add the following block of code:

```
...  
added_files = [  
    ("docs\\examples", "docs\\examples" ),  
    ("docs\\user_manual.pdf", "docs"),  
    ("gui\\LICENSE.txt", "gui"),  
    ("config\\shortcuts.json", "config"),  
    ("config\\icons\\PALMS.png", "config\\icons"),  
    ("config\\AnnotationConfig", "config\\AnnotationConfig"),  
    ("config\\EpochConfig", "config\\EpochConfig"),  
    ("logic\\databases\\EXAMPLE_PPG.py", "logic\\databases"),  
    ("logic\\databases\\EXAMPLE_RESPIRATION.py", "logic\\databases")  
]
```

2.2 Below find a line `'datas=[]'` and assign:

```
datas = added_files  
...
```

2.3 including ("config\\shortcuts.json","config") will make shortcuts fixed, won't be possible to change in portable version. Omitting this will require a config\shortcuts.json to be near executable when running (or default is used)

3. Execute in the console: `*pyinstaller --onefile PALMS.spec*`

4. PALMS.exe is ready in `*dist*`