B. Developer documentation

In this chapter, we provide an overview of the whole application code base. The motivation is to help to get an idea how the program is working and to get oriented in the code.

B.1 Application parts

The program consists of several components, namely Application Process, Graphical User Interface, Cameras Provider, Calibration Provider, Trackers Provider and Localization. Each of this component has own file. In the following list, we explain the purpose in each of them.

ApplicationProcess.py

This part of the program is responsible for the running all the other parts. It runs them step by step, starting with the initialization of the cameras and ending with the localization process. For some of these tasks, it creates a new separate thread to run.

Running other parts is provided in this order:

- initialization of the cameras – CamerasProvider.py,
- initilaization of the GUI – GUI.py,
- performing mono and stereocalibration – CalibrationsProvider.py,
- initilization trackers – TrackersProvider.py,
- computing projection matrices and localization – Localization.py.

CamerasProvider.py

Cameras provider runs in a separate thread. Based on the options it will capture the images from the cameras or the video files. Capturing is stopped, when the application closes, as a final step it releases the cameras or the video files.

In case that the input is provided by video files, waiting loops are added, to not to play video accelerated. In case that computation of the other parts slows down the application, the provider will capture an image from the video, which is behind (that means, videos with different FPS may be used).

GUI.py

GUI is responsible for everything that user can see. GUI takes care of printing messages to console, updating images in the camera view, displaying results of tracking and also localization.

If the window is closed, the GUI set an event, for all other threads to signalize to end their work. Then the application exits.

CalibrationsProvider.py

We divide calibration process into two separate parts. Mono calibration of each camera and stereo calibration. Both calibrations find pictures with the chessboard and then run OpenCV functions to calibrate. To set a number of images to calibrate on, change the value in the Config.py

TrackersProvider.py

Trackers provider takes care of all trackers – their initialization and also tracking. If the initialization of the tracker is requested, then it initializes the tracker with the specified bounding box (provided by mouse clicks) on the most recent image.

If the tracker was already initialized, the tracker's provider asks the tracker to return the position of the object in given image.

Because the trackers are usually not able to reinitialize, we encapsulate tracking algorithm into Tracker class. On this encapsulated class initialization may be called multiple times. It that case, it creates a new tracker in the background and replaces the current one by the new.

Since we call trackers by name, the class TrackersFactory provide a mapping between the names and the classes to create the trackers. Each tracker has to be added to this class.

Localization.py

Localization class is a static class. It can compute projection matrices on given calibration results, which the Localization class use later when computing the object position in the 3D space.

Then for receiving a position of the object in 3D point we need to provide coordinates from both images. Firstly it corrects the distortion from the cameras and then uses triangulation method to compute the position.

B.2 Threads

The application runs in several threads. The *Main* thread is responsible for exiting the whole application. It creates a thread for the *Application*. Also the *GUI* and the *TrackersProvider* runs in own thread. All other tasks do *Application* thread.

In Main.py the stop_event is created. This event is passed to every thread. After setting a stop event, every thread should finish its work. The stop event is set by closing application windows – so it is set inside the GUI thread.

B.3 Queues

To share information between different parts and threads, we provide several queues for the data. In this section, we provide a short description to each queue, what information it contains and which components of the program use it. All queues are created in `QueuesProvider.py`.

Image entries Queue

Images entries queue contains captured images with some additional information. We store only the last 500 pictures for each camera. The oldest pictures are

```
class TrackerSample(object):
    def init(self, image, bbox):
        # Should return True or False
        pass

    def update(self, image):
        # Should return a tuple (True/False, bbox)
        pass
```

Figure B.1: Template for the new tracker class

automatically thrown away by using `dequeue`. Using common web camera with 30 FPS, it stores the last 16 seconds per camera. The image entries are then used by the calibration process, trackers and finally displayed by GUI.

Image entry

Each image entry in Images queue contains this information:

- timestamp – the time when the image was captured. The time in seconds since the epoch,
- image – contains three channel two dimensional arrays containing the captured image,
- chessboard – information about the chessboard in the image. Access to chessboard information is provided by functions.

Tracked points

TrackedPoints2D is a queue for the results of the trackers. It is a list of the queues, separate one for each tracker's results. These results are then used by GUI to display tracker information.

Localized points

Localized points contain a list of estimated coordinates for each object. The lists are together stored in the list LocalizedPoints3D. At the end of the program, all data are automatically saved.

The Localization class adds data to this queue and GUI is adding the estimated positions to the 3D live view.

Point

The class Point is used to store tracking results and also localization results. Point consists of the time stamp (similar to the Image entry) and coordinates.

Console output

To provide a console with the information in the application window we use a list for logging information. Everything appended to this queue will be displayed in the text box. Only strings can be appended.

To the console output can add messages any part of the program. The GUI will then display them.

Mouse clicks

Each mouse click with the left mouse button in the camera view is recorded and saved to this queue. For each camera view, we create a separate list, into which we append coordinates of the mouse click.

Mouse clicks are used only for trackers initialization. The GUI records them by calling a callback.

B.4 Adding a new tracker

In this application you can test different trackers and their ability to track an object in this task. It is possible to add a new tracker and use it.

We recommend to include a tracker into a directory with all other trackers (`program/trackers`). We keep a naming convention, so we start the name of the tracker with a prefix "Tracker".

It is important to remember, that for each object in each camera view a new instance of the tracker will be initialized.

Tracker is expected to be a class which implements two methods. The skeleton for the new tracker is displayed in the Figure B.1.

The image will be provided as a three channel two dimensional NumPy array. The bounding box is represented by four numbers, in the following order: `(x, y, width, height)`. Coordinates `(x, y)` represents top left corner of the bounding box and `width`, `height` its dimension.

The function `init` should return `True` or `False`, depending if the initialization was finished successfully.

The function `update` should return tuple `(state, bouding\_box)`. If the tracker can locate the object, it should return state `True` and corresponding bounding box in the same format as used during initialization. If the object was not found, a tuple `(False, None)` should be returned.

As the last step, we add a new tracker to `program/TrackersFactory.py`. We associate its name as a string to the tracker's class.

B.5 Configuration

Additional configuration setting are available in `program/Config.py`.