Bachelor Thesis in Computer Science

Biorobotics Lab, Dahlem Center of Machine Learning and Robotics
Institute of Computer Science, Freie Universität Berlin

BioTracker 3 - Implementation of a generic visualization and functional enhancement of the UI

Author
Jonas Piotrowski

Supervisor
Prof. Dr. Tim Landgraf

Berlin, 19. April 2018

Abstract

The study of animal behavior by analyzing recordings can be very labor-intensive if done by hand. To solve this problem, many automated image tracking applications were developed. All of them are either too complex to use or too specialized to be a solution fitting various tracking problems. To overcome these issues, an open source tracking framework called "BioTracker" was designed. It aims to cover many tracking scenarios by giving the possibility of creating your own tracker, while still providing high usability. The goal of this thesis is to improve the user experience of the BioTracker by implementing a generic visualization of tracking data and by adding new features to an optimized user interface.

Contents

Statement of originality

I declare that this thesis is the product of my own original work and has not been submitted in similar form to any university institution for assessment purposes. All external sources used, such as books, websites, articles, etc. have been indicated as such and have been cited in the references section.

Berlin, 19. April 2018

Jonas Piotrowski

1 Definitions

The following definitions explain many commonly used terms of this thesis:

tracking application: A software using computer vision algorithms to process media files and generate data out of detected objects in the medium.

core application: The tracking application without any plugins.

tracker / tracking plugin: A seperate part of software loadable into the core application. It provides the tracking algorithm.

trajectory: The collected data of one tracked object in the medium. Includes all entites of that object.

entity: The positional (and other) data of a tracked object in one frame of the medium.

shape object: The corresponding visualization object to a trajectory and thus to a tracked object. Always displays the current entity of the trajectory.

usability: "Extent to which a system, product or service can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use" (DIN EN ISO 9241 - 11) [25].

user experience: „A person's perceptions and responses that result from the use and/or anticipated use of a product, system or service." (DIN EN ISO 9241 - 210)[24]. It extends the usability with antizipation and reprocessing of the product.

2 Introduction

Research focussing on the study of animal behavior is often done by analyzing videos or images. In the past, extracting data, like position or orientation, required a manual image by image review of the medium. This can be very slow, inconvenient and inaccurate. To automate this process and extract consistent data, various tracking applications were developed. These applications use computer vision algorithms to process the video or image data and extract tracking data out of it automatically. Automated tracking applications were for example used in the tracking of bee dances [14] or the behavioral analysis of guppies interacting with a robotic fish [13].

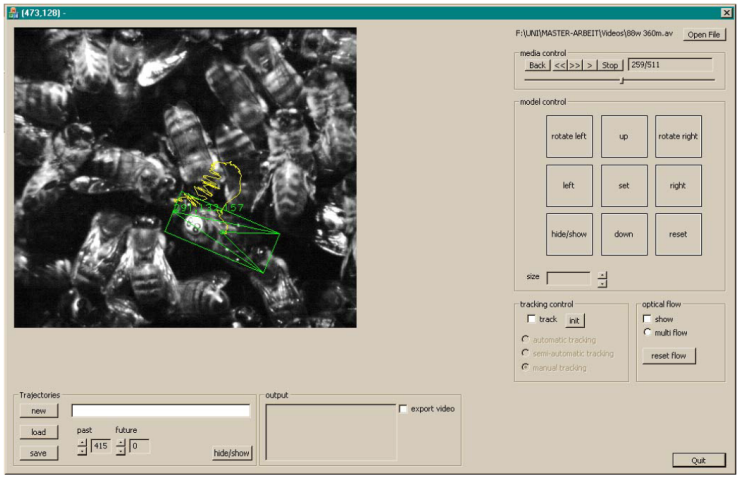


Figure 1: Screenshot of the tracking application developed for the tracking of honey bee dances. Source: [14]

3 State of the Art

There are already various (semi-) automatic image tracking applications in circulation. In a review [10] the quality of many applications are summarized. Many of these applications are either specialized on one animal or one experimental setup. An example for the specialization on an animal is ZebraZoom [17] which concentrates on automatically tracking discrete movements of zebrafish larvae. An application concentrating on the experimental setup and more precisely on the video format is Ctrax [9]. It is also an example of many applications using a background subtraction algorithm to process the images and track objects. Background subtraction is a widely used computer vision technique where the current image is compared with a stored background to find moving objects. The problem with background subtraction (and thus with all the applications using it) is that it only works if the background is stable and the object contrastful. Apart from the problem of specialization, some applications like Ethovision [21] do have a large set of features but are proprietary and have high acquisition costs (Ethovision: 5850$).

What is thus needed in a tracking application is versatility in experimental setups and open source status. A good solution to accomplish that is an application framework with the possibility to load (or even create) a tracking algorithm and adjust it to make it work with any experimental setup and tracking scenario. There are not that many tracking frameworks existing. Examples are SwisTrack [16] and ImageJ [22]. Both have the problem of being very complicated in use. SwisTrack in setting up the tracking algorithm, because much computer vision knowledge is needed. ImageJ in the overall usage of the application, because it is cluttered with unimportant functionalities as it was not designed for video

tracking but image processing.

4 Motivation

The *Biorobotics Lab*[1] of the "*Freie Universität Berlin*" developed the BioTracker reacting on the problems of existing tracking applications (see Section "Related work") and the need for a compilation of all the stand-alone tracking solutions the lab developed separately. It is a open source[2] tracking framework, designed to provide a pleasant user experience and highly customizable tracking plugins. Version two of the BioTracker was including exactly these features but had structural problems like a bad expandability[12]. That is why a complete rewrite was done by A. Jörg[12]. In his thesis, he structured the BioTracker to provide a rich, generic feature set, easy expandability, and a more simplified plugin development.

The result of his thesis was missing several features:

- focus on a high user experience
- tracking data export and import
- tracking data manipulation by the user.

Furthermore, the tracking plugin development needed to be simplified even more by implementing a generic tracking data visualization in the framework. A developer then only has to implement the tracking algorithm he wants to use. Previously he also had to implement the visualization of the tracking data himself.

The main goal of this thesis is to develop these features.

The long-term goal of the BioTracker is to provide a full ecosystem for image processing and tracking via computer vision.

5 Preparation

5.1 Strategy

J. Nielsen describes an iterative user-interface design [19] as a suitable design strategy to optimize the usability of a user interface. It adapts to the user feedback and is therefore designed to be flexible in taking new requirements, which is why it is used as a strategy for this thesis.

The base idea is that first a prototype is developed out of the initial requirements. Then per iteration, preferably non-biased, users test the software. The tests are evaluated and the software gets optimized based on that evaluation. In the case of the BioTracker, I collected the requirements, set up a prototype and did two user test iterations with effectively one optimization. The evaluation of the second user test will suggest further optimizations.

[1] http://www.berlinbiorobotics.blog

[2] Available on GitHub: https://github.com/BioroboticsLab/biotracker_core

5.2 Target groups

There are two main target groups. Users, who mainly want to use the BioTracker for its tracking functionality are the basic users. They want to use the available trackers to generate tracking data, videos and/or screenshots mostly for research purposes. We will call this group "biologists" as it mostly consists of biologists using the BioTracker for behavioral analysis. The main focus for them is high usability and broad functionality in tracking and visualization.

The second target group are developers, who want to implement tracking-algorithms. We will call them "tracker developers". They mainly want an easy-to-use framework to implement their tracking algorithm. Also needing to implement a visualization and user interactions can be disruptive and time-consuming.

Both target groups are equally important as they push forward the success of the BioTracker. The "biologists" use it for its simple interface and integrated trackers. If the given trackers are not sufficient for tracking scenario, the "biologist" can work with a "tracker developer" to develop a new tracker. Furthermore the "tracker developers" will use the BioTracker to develop their tracking plugins because of its simplicity. This all happens, of course, on the premise that the BioTracker meets these requirements and gains popularity.

5.3 Use Cases

The use cases of the BioTracker are essentially tracking objects in media, generating tracking data, visualizing it and developing a new tracking plugin.

The "biologist" (figure 2) uses the software via its user interface to track objects, visualize the tracking data and save it either in log data, video or image form. The use cases "Tracked Data visualization and Manipulation", "Media Actions" and "Set visualization options" in figure 2 are broadly defined on purpose. They consist of various functionalities which will be specified in the following sections.

The "tracker developer" (figure 3) only focuses on developing a new tracking plugin. This consists basically of implementing interfaces and a tracking algorithm. Other steps should not be required to create a functional tracking plugin.

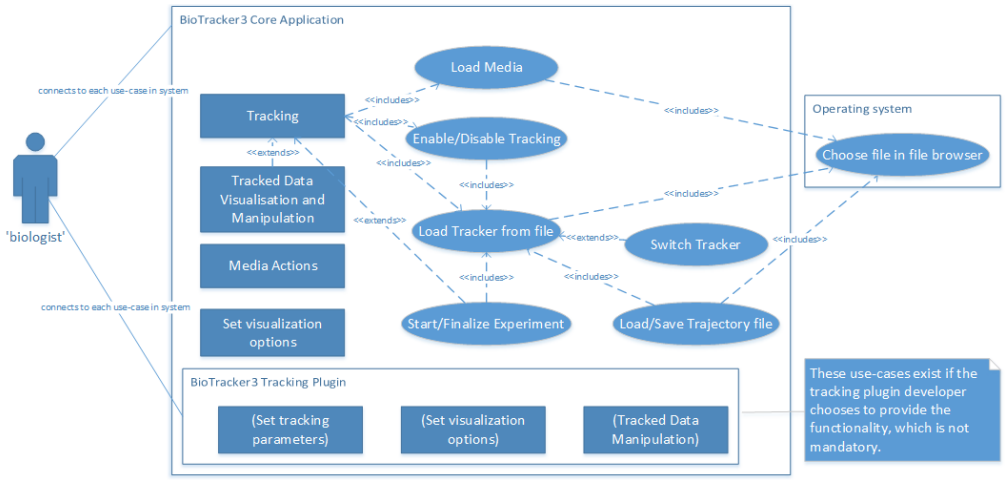


Figure 2: The basic Use Case Diagram for the actor "biologist". Rectangles represent groups of use cases. For example the "Media Actions" use case consists of video playback and recording functionalities among others. The main focus lies on the "Tracking" use case, which is why it is expanded in contrast to the other ones. The subsystem "BioTracker 3 Tracking Plugin" is part of the system "BioTracker 3 Core Application" because it is integrated into the interface when a tracking plugin is loaded. Note that the actor connects to each use case in the "BioTracker 3 Core Application" system. These associations are not visualized to simplify the diagram.

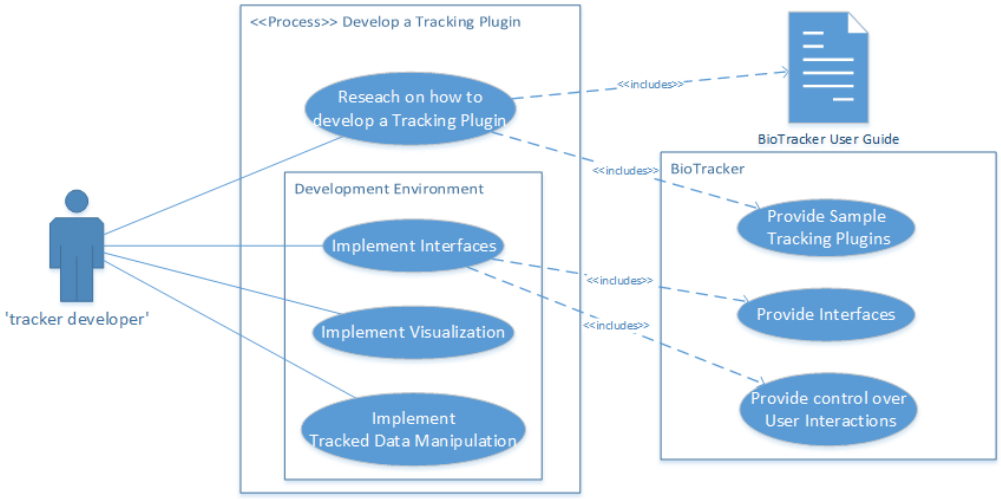


Figure 3: The basic Use Case Diagram for the actor ”tracker developer”. This diagram concentrates on the process "Develop a Tracking Plugin" which is the only use case for this target group. The developer has access to provided interfaces, sample tracking plugins and the BioTracker's user guide, to develop a tracking plugin. Furthermore, he has control over the tracked data visualization and manipulation in the core application to deactivate them if he chooses to implement his own visualization and tracked data manipulation.

5.4 Requirements

Overall requirements are:

- extending the structural definition of tracking data
- providing a generic tracking data visualization with user manipulation possibilities to simplify the development of a tracking plugin
- developing a simple but appealing Graphical User Interface (GUI) with more functionality but still high usability.

The functional requirements include the functionalities the software requires (What can be done?):

- Provide a generic, easily manageable tracking-data data structure
- Generic tracking data can be visualized by the core application
- Change appearance of visualized elements
- Modify tracking data by interacting with visualisation and interface

The non-functional requirements include basic quality-of-service conditions under which the software must be operating:

- High usability and good user experience
- Works on low performance systems
- Consistency in code and graphical interface
- Works on Windows, Linux and Mac

6 Implementation

6.1 Used libraries and frameworks

The BioTracker is conforms to C++ 14 and is based on Qt 5 [4], OpenCV [3] and boost [1]. Qt is a cross-platform framework for creating application software . It provides numerous interface classes for different design patterns such as the Model-View-Controller-Pattern or the Command-Pattern. That is why it is mostly used for developing graphical user interfaces. It supplies the structure of the BioTracker, which is why most of the codebase depends on it. Most major classes derive from QObject, which provides the signal and slot mechanism. Qt provides this functionality to simplify the communication between software components. OpenCV is a computer vision library providing functionality for image preprocessing, object recognition and motion tracking. It is mostly used in the tracking plugins. Boost is a wide-ranging utility library used only in small parts of the BioTracker, most noticeably in the data serialization and parts of the file management.

6.2 Software Architecture

The architecture of the BioTracker 3 and the used patterns and structure were extensively explained in the section “Architecture of the BioTracker 3” of A.Jörgs" thesis [12].

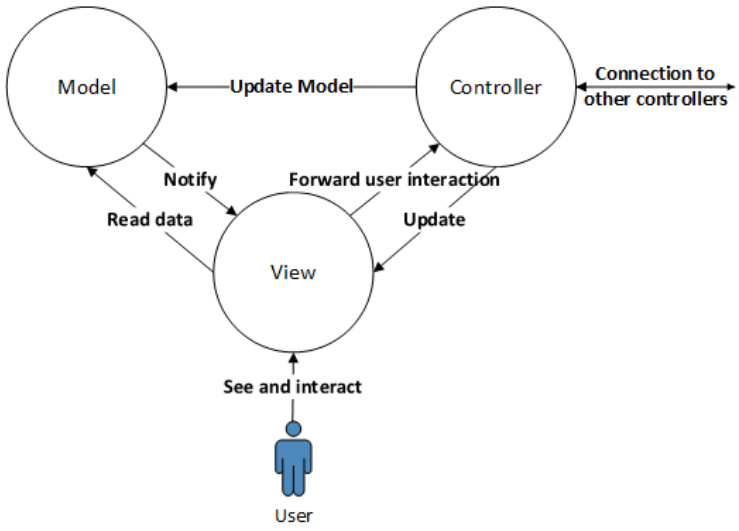


Figure 4: A simplistic overview of the Model-View-Controller pattern used as the base structural pattern of the BioTracker.

The BioTracker basically consists of multiple Model-View-Controller components (see figure 4). All these components are interconnected and each one provides a distinct functionality. Explanations on what the functionality of each previously existing component is, can be found in A. Jörg’s master thesis [12]:

- MediaPlayer Component: Is multi-threaded, takes care of the video controls embedded in the main window, controls image streams, and provides the current cv::Mat to other components.
- TextureObject Component: Receives references of cv::Mats from the MediaPlayer component and BioTracker Plugin. It converts them to QPixmap for displaying in the GraphicsScene component.
- GraphicsScene Component: Implements a QGraphicsScene. This component renders QPixmap objects provided by TextureObject, as well as tracked data, represented by TrackedComponents.

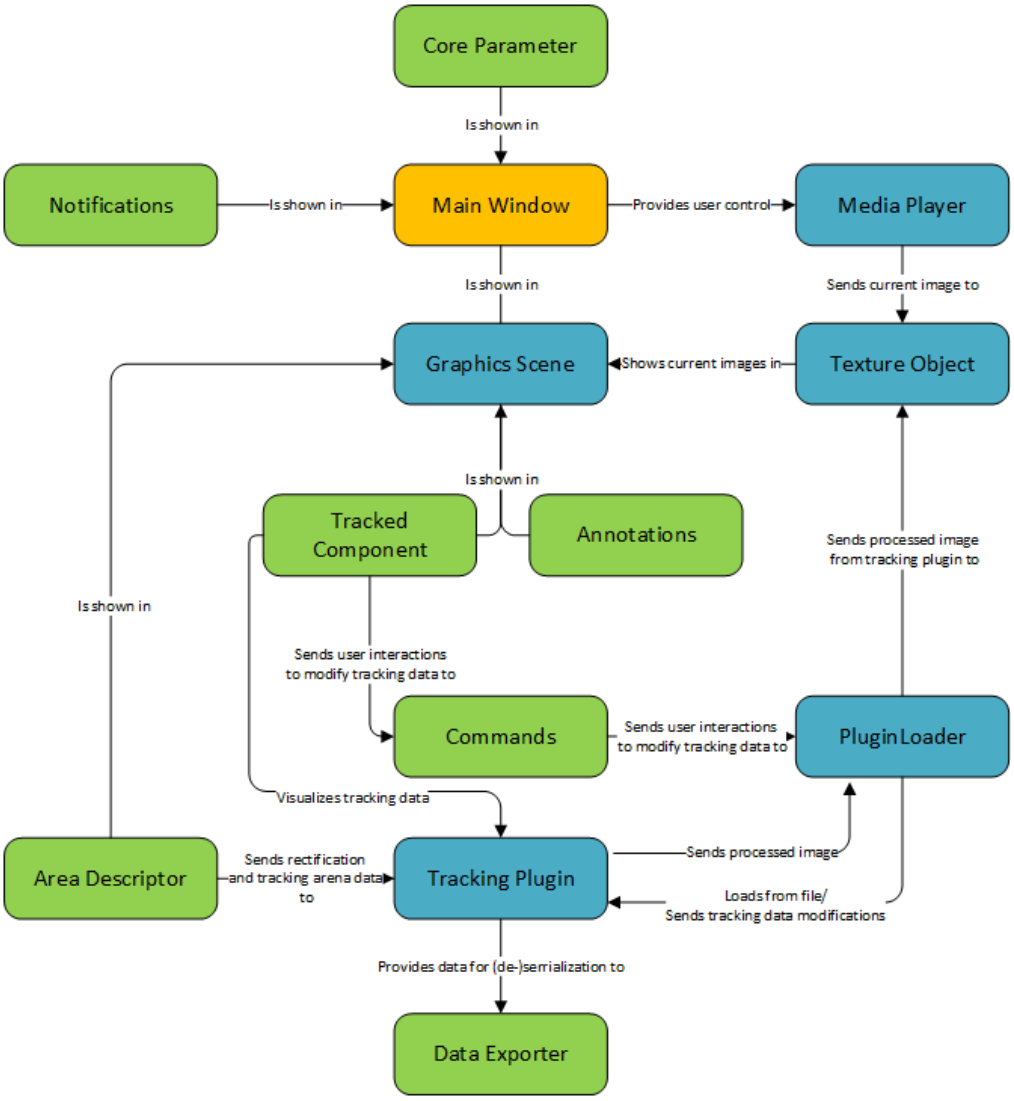


Figure 5: Simplified component structure of BioTracker 3. Blue Components were already existing (by A. Jörg). Orange components were already existing but heavily changed and green components are newly developed. Note: This is diagram is very simplified as it only shows the most basic relations between the components. Each component has the Model-View-Controller structure.

6.3 Tracking process

The main process the BioTracker is used for is tracking. Figure 6 provides a basic overview of how the tracking process proceeds. The red numbers indicate the sequence of the process. After the user started the tracking process:

1. The "Media Playback" functionality of the core application fetches the current image frame from the loaded medium
2. It sends the current frame to the "Tracking Plugin".
3. The "Tracking Plugin" processes this image frame and extracts tracking data.
4. The "Tracking Plugin" then sends a notification, that the tracking is done and the tracking data extracted, to the core application. In particular, the "Media Playback" and the "Visualization" are notified.
5. The "Visualization" functionality then visualizes the tracking data for that tracked image frame and displays it to the user.
6. Back to (1)

If the user stops the tracking, step 1 will not be executed anymore, the tracker stops tracking after the last received image frame and the "Visualisation" visualizes it.

In the previous implementation[3] of this process, the "Media Playback" sends the "Tracking Plugin" new image frames as fast as possible. The problem is that the "Tracking plugin" is normally not fast enough to process an image frame before it receives the next image frame. This causes the tracking in the plugin to continue for a little while after the tracking process has been stopped, meaning that there is always a delay. This has been overcome by serializing the process. The "Media Playback" now waits for the "Tracking Plugin" to be done with tracking the previous image frame (step 4).

[3] Version 3 of A. Jörg

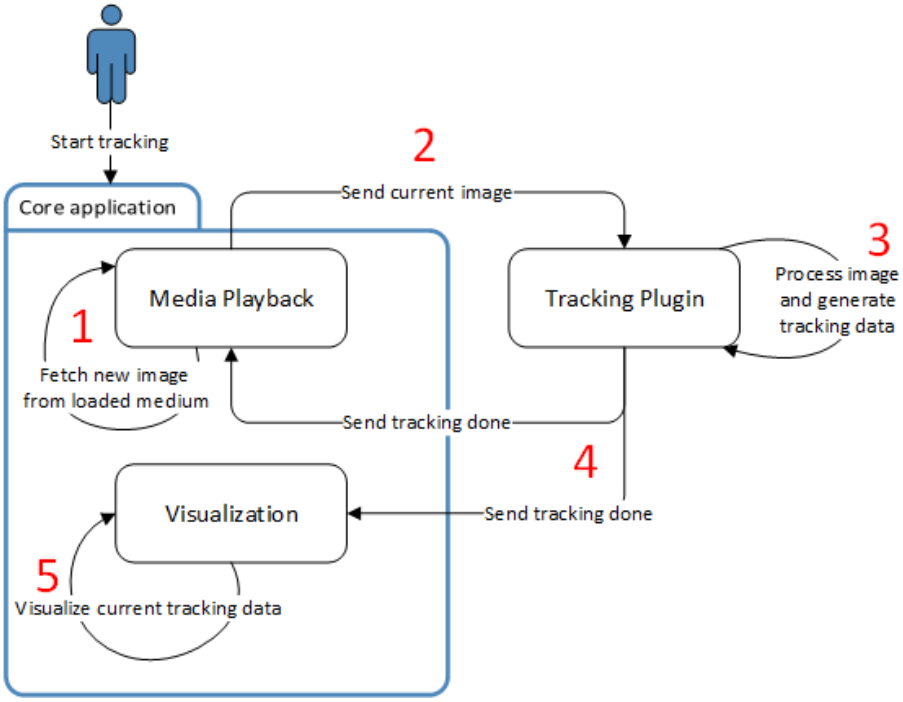


Figure 6: Overview of the basic internal tracking process. Shown are the "Core application" and the "Tracking plugin". The "Core application" contains the "Media Playback" and "Visualization" functionalities. Note: An acknowledgment from the "Visualization" is not needed for the tracking cycle. It visualizes the tracking data asynchronously and could be omitted, e.g. for a Command-line interface version. The asynchronicity is no problem as it only reads data which the tracked already generated and shall not write on.

Background Subtraction Tracker[a]**:** Uses background subtraction to process the image and track objects in it. All image processing steps can be modified to fit the video. Works best on media with a stable background.

Lucas-Kanade Tracker[b]**:** Uses optical flow calculations on the medium to keep track of marked objects. Is less sensitive to image noise but susceptible to fast movement.

[a]The Background Subtraction Tracker was developed by H. Mönck.
[b]The Lukas-Kanade-Tracker was developed by H. Mönck.

Table 1: List of currently available tracking plugins in the BioTracker 3.

6.4 Details of new Systems and Components

To provide the new functionalities listed in the requirements, new components were needed for the BioTracker (green components in figure 5). Additionally, old components were changed and extended. These new components were developed partly by me and partly by H. Mönck of the Biorobotics Lab. Additions and changes will be discussed in the following sections.

6.4.1 Tracking Data Structure

The structure of the tracking data is a three-layered data tree (see figure 7). The tree structure has the advantage of high scalability and accessibility. The highest layer is the root node (layer 0). It consists of trajectory nodes (layer one). Each trajectory node is the collection of all entities for a tracked object. The entities (layer two) represent the generated tracked data of an object for each frame.

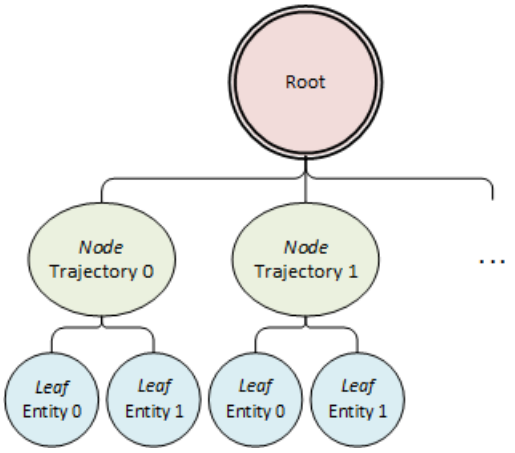


Figure 7: The BioTracker's tracking data structure. It is a tree consisting of a root (layer 0), trajectories (layer 1) and entities (layer 2)

To accomplish the tree structure the Composite Pattern was used [12]. The advantage of this pattern is that itself is based on a tree structure, that it offers a uniformity in manipulating objects and that it reduces complexity and code redundancy. The tracking plugin only needs to implement "IModelTrackedTrajectory" (node in the data structure and composite) and one of the offered leaf classes: "IModelTrackedPoint", "IModelTrackedEllipse", "IModelTrackedRectangle" or "IModelTrackedPolygon" if he wants to use the provided visualization. This provides the possibility of different forms of visualization to the tracker developer. Once generated tracking data may never be deleted afterward. To stop tracking data from being visualized it must be set invalid.

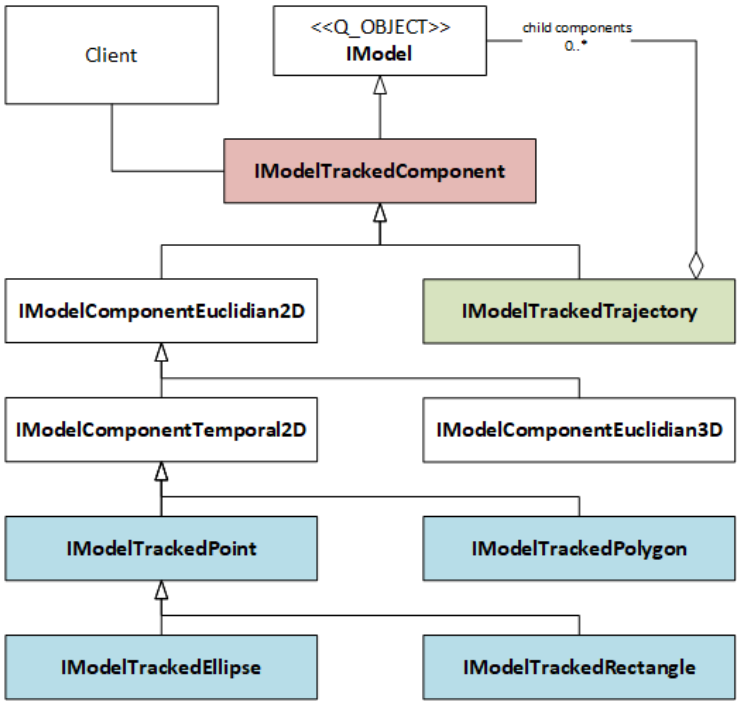


Figure 8: The class diagram showing the interfaces of the tracking data structure. The pattern used is the composite pattern [11]. Component part is marked red , the composite part is marked green and the leaf parts are marked blue. "IModelTrackedComponentsEuclidian3D" is not used yet in the BioTracker but is reserved for the possibility of 3D tracking. Note: For comparison with the previous implementation, see section "Data Entities" of A. Jörg's master thesis [12].This solution was developed by H. Mönck and myself.

6.4.2 Visualization of tracking data

The BioTracker's framework offers the possibility of visualizing the tracking data in real time during the tracking process (see figure 6) and with already generated tracked data. This is done by the visualization component which is one of the most important parts of this thesis. There is a shape object for each trajectory in the tracking data. These shape objects are created automatically in the view of the visualization component. Whenever the visualization gets updated the shape object visualizes the currently tracked entity of its associated trajectory (see figure 9). Each shape object is displayed in the view which itself is shown to the user.

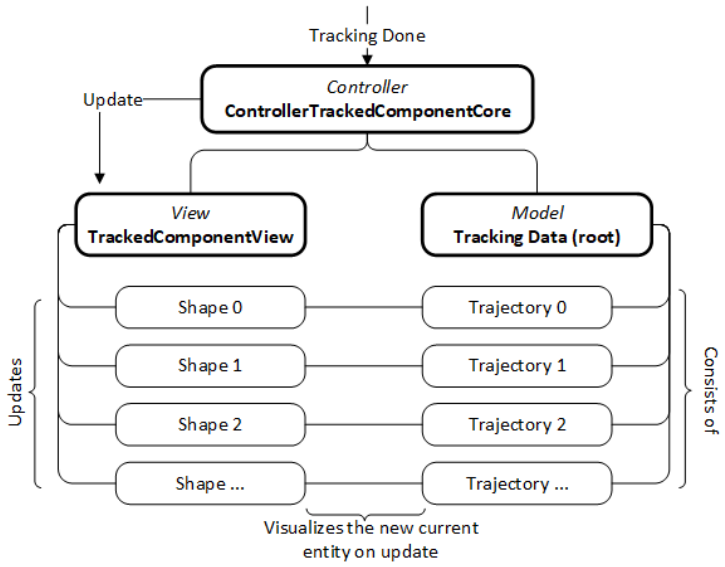


Figure 9: Simplistic class diagram of the visualization component. The view consists of one "Shape" for each "Trajectory" in the tracking data. Additionally illustrated is the process of updating these shapes while tracking (see step 4 and 5 in figure 6).

The appearance of the shape object depends on what type (figure 10) the current corresponding entity has. The entity type results what interface the entity implements. This means that the currently possible types are points, rectangles, ellipses and polygons (corresponding to leaf interfaces in figure 7; see figure 8 for examples). In every case, the position in the image frame, orientation and type[4]will be shown. If the trajectory or the current entity is invalid, the shape object is hidden. Shape objects may have the following properties:

- position: X- and Y-coordinate of the entity.

[4]e.g. an ellipse type will be shown as an ellipse with width and height

- type: Current type of entity. An ellipse type will be shown as an elliptical shape at the position.
- size: Width and height of the entity. An ellipse will be shown at its position with the current width and height
- orientation: Orientation of the entity. The shape object will be rotated to visualize the orientation
- color: Border and fill color
- transparency: Transparency of the shape object
- antialiasing: Rendering of the shape object will use antialiasing
- id: ID of the entity

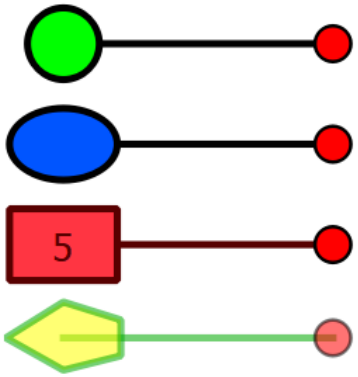


Figure 10: All types of shape objects. From top to bottom: point, ellipse, rectangle and polygon. The point has default colors, while the others have their border color and/or their fill color changed. The rectangle has its ID shown. The polygon's transparency has been lowered. The line from each shape object displays the orientation. The user can move the red circle at the end of the line to change the orientation.

The visualization component also provides user interaction functionality. On the one hand, the user can change the appearance of the shape object, not affecting the tracking data. On the other hand, the user can modify the tracking data by for example moving or rotating the shape object or even by adding new shape objects.

These interactions can be dangerous for the tracking data's integrity, which is why the tracking plugin developer has must implement the modifications himself. Additionally, he has the possibility to prohibit the user interactions by setting permissions. There is a permission for each user interaction that would alter the tracking data. There is also a permission to disable the visualization

component of the core application, which would leave the visualization to the tracking plugin. By default, all user interactions are permitted.

6.4.3 Commands

The commands component interposes and mediates between the user interactions that modify tracking data and the tracking plugin that is actually implementing these modifications. It provides an undo/redo functionality in doing so. The component is based on a Qt framework using the command pattern [7]. The command pattern helps to encapsulate actions and to decouple invoker and receiver. Each command saves the data to undo and to redo itself. For example, the command to move an entity saves the new and the old position. Qt provides a stack to save these commands and a view to visualize the stack. Currently, only commands modifying the tracking data are implemented. These are the most important ones as the user does not always have the precision or memory to undo for example a move of an entity.

6.4.4 Core parameter

The core parameter component is a small component which is used to save runtime data by providing a shared location. Stored here are for example the default values for the visualization. These values control how new shape objects are visualized. The view of the core parameter component is accessible to the user. He can, for example, modify the default visualization values.

6.4.5 Annotations

The annotation component[5] offers the functionality of creating markings in the medium. These annotations can be created, moved and deleted by the user. They are bound to a frame of a video where they are active (colored). In other frames of the medium, they are inactive (greyed out). The annotations are serialized in a file for each video when the video is closed. Upon loading the same video they are deserialized and shown to the user.

6.4.6 Notifications

The notification component offers the functionality to inform the user via messages in a text field. The messages are intercepted Qt Debug messages so the tracking plugin and the core application can easily create a notification by creating a Qt-Debug message [5]. Automatically created debug messages by Qt will also be shown. These notifications could be sent by the user to the developers when problems or bugs occur.

[5] This component was created by D. Dormagen and me

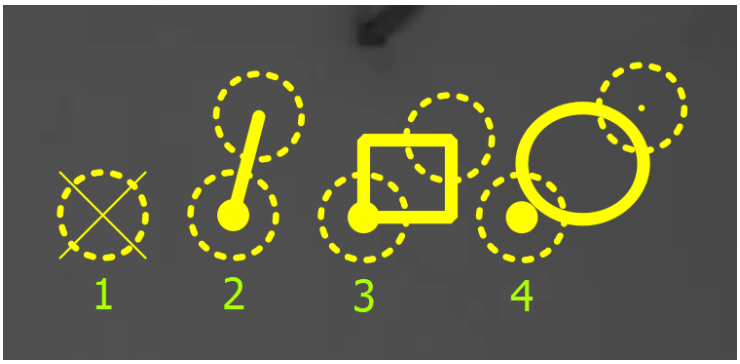


Figure 11: All currently available annotation types: (1) label annotation, (2) arrow annotation, (3) rectangular annotation and (4) elliptical annotation. The red circle is the selected end point of the arrow annotation. Each number is the current text of the annotation above it.

6.4.7 Data serialization and deserialization

The data serialization and deserialization component[6] provides the functionality of saving the tracking data to a file (exporting) and loading tracking data from a file (importing).

For serialization, the component's exporter reads the tracking data and serializes it frame by frame. This file will then be saved in the "Tracks" directory which is located inside the applications root directory. The exporting can either be user invoked or happens automatically upon loading a new tracker, a new medium or closing the application to avoid losing the data by accident. Another feature of the serialization is the possibility to create trials. These simply are tracked data files that are saved to the "Trials" directory located within the "Tracks" directory. If a trial is started by the user, the current tracking data is automatically exported. This makes it possible to have clutter-free tracking data for experiments and future analyses.

For deserialization, the component reads the tracking data from the file and forwards it to the tracking plugin. Available file formats for the tracking data are a generic file format, CSV (default) or JSON.

6.4.8 Area descriptor

The area descriptor component[7] provides a rectification and tracking arena functionality. The rectification sends the tracking plugin the necessary data to convert pixel values from the image frames into real-world centimeter values. In the interface, the user has to mark a rectangle and then enter its size in centimeters. The tracking arena functionality lets the user set the outer bounds in

[6] This component was mainly developed by H. Mönck. I added some minor changes.
[7] This component was mainly developed by H. Mönck.

which the tracking plugin can track objects. The shape of this arena can either be a rectangle or an ellipse.

6.4.9 Main window

The main window component provides the graphical interface visible to the user. It is the canvas in which most views of the other components are shown (see figure 12). Additionally, it provides links to functionalities not provided by the views of the other components. This includes, for example, the loading of media, tracking plugins and tracking data. It was initially implemented by A. Jörg in his master thesis [12] but was heavily edited in this thesis. It is the main component one can change to improve usability and perspicuity. The following section describes the structure and features of the graphical interface of the BioTracker.

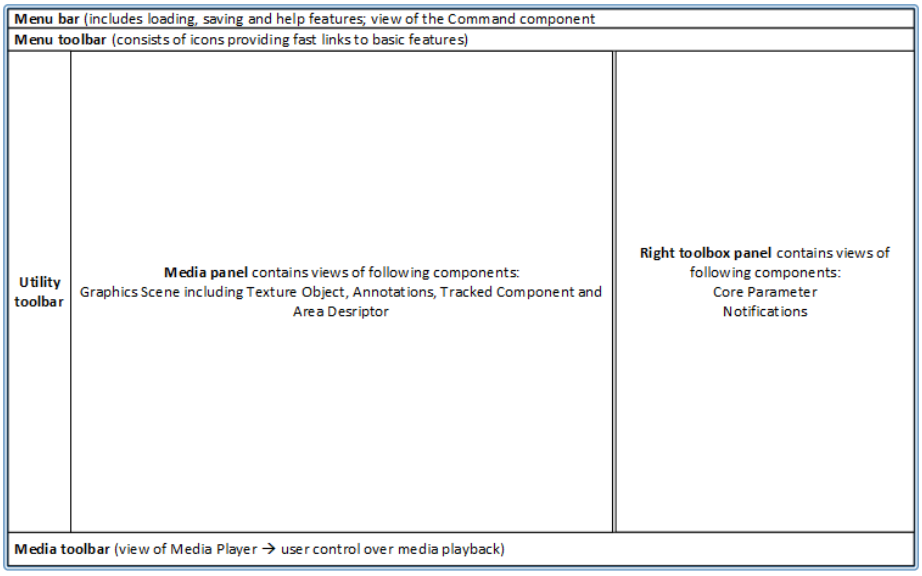


Figure 12: View of the Main Window component with included views of other components. See figure 5 for all components. This figure also illustrates how the GUI is structured.

6.5 Graphical User Interface

The graphical user interface (GUI) of the BioTracker is the part of the application constituting usability and perspicuity the most. Besides concentrating on these two aspects, Jakob Nielsen's "10 heuristics for User Interface Design" [20] (see table 2) were a guideline in the GUI development.

- Visibility of system status
- Match between system and the real world
- User control and freedom
- Consistency and standards
- Error prevention
- Recognition rather than recall
- Flexibility and efficiency of use
- Aesthetic and minimalist design
- Help users recognize, diagnose, and recover from errors
- Help and documentation

Table 2: Nielsens "10 heuristics for User Interface Design" [20]

The whole GUI is based on the "Qt Widgets Module" [8], which contains different user interface elements (such as buttons or checkboxes) to provide a native look and feel on each platform.

General features of the GUI, which are not structural are the following:

English: The language of all GUI elements is English. This ensures that most potential users can use the BioTracker.

Tooltips: Each GUI element has a tooltip appearing when hovered over. This tooltip is an explanatory text describing the functionality of the element.

Help: Apart from tooltips, there are more help features to describe how the BioTracker works. There is a document called user guide, which explains the functionalities of the BioTracker thoroughly. It is deployed with the software and linked in it. Giving a more basic overview of how to use the BioTracker, there is an illustrated introduction when starting the application for the first time. This introduction is a dialog which the user can click through to gain first information. This feature was added because it became apparent that first-time users usually do not read the user guide beforehand.

High user control: The GUI provides high control over itself and offered functionalities to the user. Most main elements can be hidden. Each toolbar can be moved.

Minimalistic color design: The color design of the GUI is based on a light gray. Some elements are pastel colored for easier recognition and better distinction. Using only muted colors shall reduce eye strain as they are not as bright as vibrant colors like pure white.

Shortcuts: Many functionalities are accessible via keyboard hotkeys. This makes the use of the BioTracker more efficient for experienced users. There also a list of all shortcuts available in the application.

Convenience links: Each time any kind of tracking data is saved to a file or a screenshot was taken, the BioTracker displays a dialog including the path of the file and buttons to open the file or the file's directory.

6.5.1 Structure

Old and new functionalities provided by the components are accessible to the user via the GUI. Besides the appearance, the structure of the GUI is very important to understand the BioTracker intuitively and to use it efficiently. Figure 13 illustrates the basic composition of the GUI. It consists of five main elements (see also figure 12). The composition of these five elements is focussed on the media panel (5) in the middle and the menu toolbar (1) on top of it, which are the most important elements used during tracking. The menu bar (1) contains the media, tracker and data I/O functionalities the user has to access to setup his tracking task. Also included are undo/redo functionalities and different help elements. The menu toolbar (1) contains mostly of icon buttons and is a compressed version of the menu bar. It is structured to clarify the steps to start tracking: "Load media", "Load tracker", optionally "Load/Save data" and in the end "Choose tracker" with a drop-down list including the currently loaded trackers and a switch to activate or deactivate tracking.

The media panel (5) (figures 13, 14) is the main user interaction element. It displays the current medium in the background and the visualization of tracking data, annotations and area descriptor functionalities on top of it. It also offers two context menus (figure 15). One context menu is reachable when the media panel, the other one opens when a shape object is right-clicked. Both are the main access point for customizing all functionalities the visualization of the shape object and manipulating the corresponding trajectory and entity (apart from moving and rotating the entity by hand).

Under the media panel, there is the media toolbar (3). It allows the user to control the media playback and gather information on the current state of the medium. Left of the media panel is the utility toolbar. As all this toolbar's functionalities are also accessible in other elements, the only purpose of this toolbar is to enhance usability by giving the user the option to speed up his interaction flow. To the right of the media panel, there is the right toolbox panel (2). It contains tabs for experiment handling, visualization options, tracker options, a short how-to and user notification. The section "Details of the right toolbar panel" describes each tab precisely. The purpose of this toolbox is to have as much structured information in the smallest space possible.

Figure 13: Screenshot of the GUI with marked main elements: (1) menu bar and menu toolbar, (2) right toolbox panel, (3) media toolbar, (4) utility toolbar, (5) media panel. In the media panel there are exemplary annotations and two tracked fish.

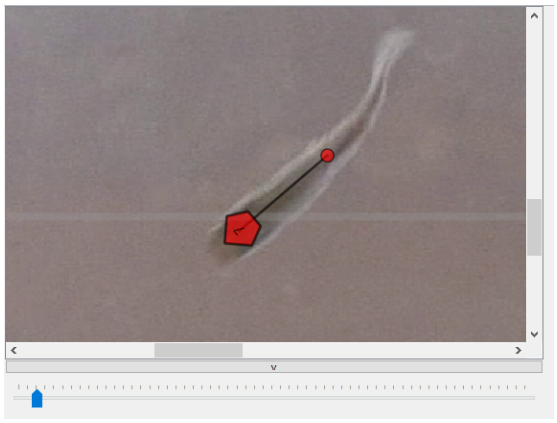
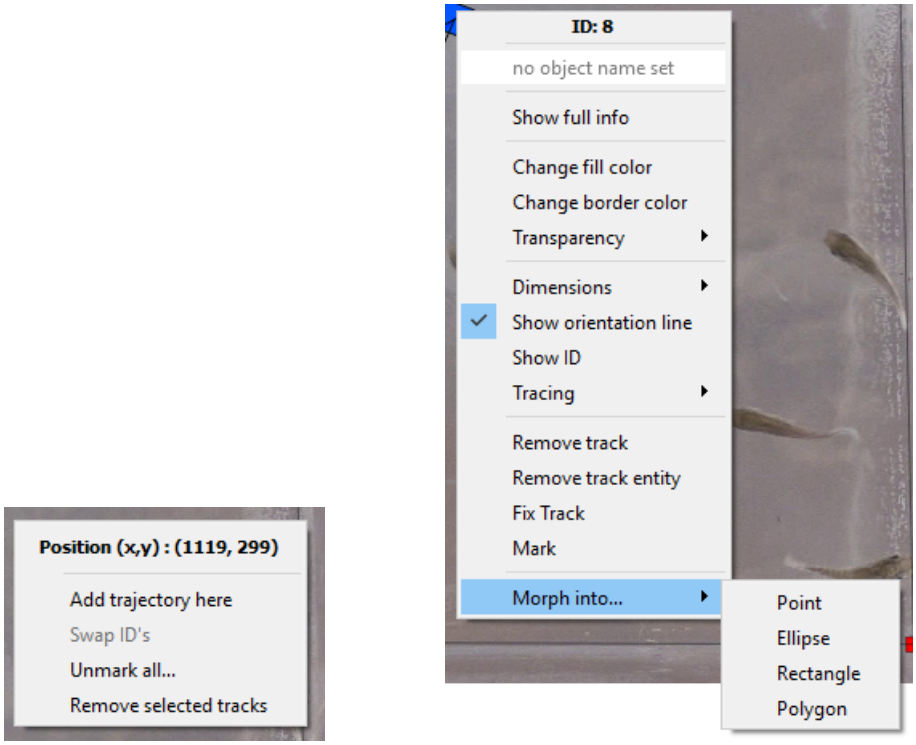


Figure 14: The media panel with a loaded video and a shape object in the form of a polygon. Underneath there is a slider displaying and controlling the current state of the loaded medium (similar to the "Current frame" text field in the media toolbar. Note: The user has zoomed into the video, recognizable by the scrollbars.



(a) (b)

Figure 15: The two context menus of the media panel. (a) opens when the media panel is right clicked. (b) opens up when a shape object is right clicked. Both contain important functionalities to manipulate the tracking data and to customize the appearance of the right clicked shape object.

6.5.2 Details of the right toolbox panel

The right toolbox panel consists of five tabs. Each tab has a distinct functionality:

(1) "Experiment": The experiment tab (figure 16) gives the user access to managing an experiment by starting, pausing and finalizing trials. When a trial is started the previous tracking data is saved in a file and then reset. This ensures that the data gathered afterward is only pure "trial data". Additionally, the tracking switch in the menu toolbar is activated and disabled, meaning that during an active trial the user cannot switch the tracking off. The trial can then either be paused or finalized. When pausing, the tracking is deactivated (but still disabled). The user can then wind through the video and visualize the current tracking data without overwriting it. When the trial is finalized the current trial's tracking data is saved into a file in the "Trials" directory. Apart from managing an experiment, the user can also save and reset the current tracking data here. This will not finalize the current trial. If no trial is active and the data is reset it will always be saved in the "Tracks" directory.

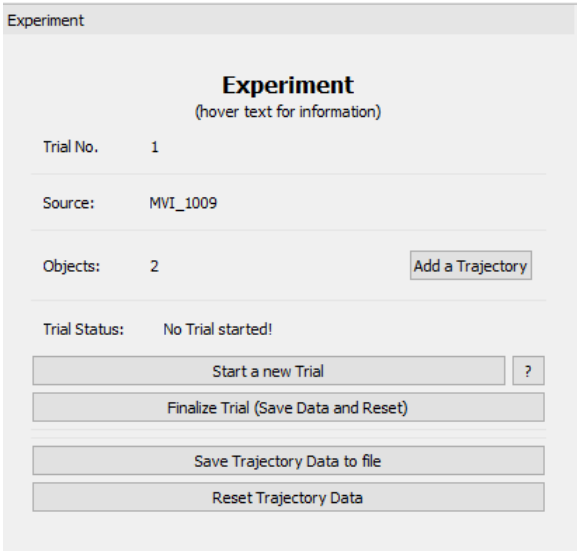


Figure 16: The "Experiment" tab. Here the user can manage trials and save and reset the current tracking data. The current trials number ("Trial No.") is always the highest number of a saved trial in the "Trials" directory plus one. The number of objects ("Objects") shows how many valid trajectories there currently are. The "?" button opens up a window with information on how the trial system works.

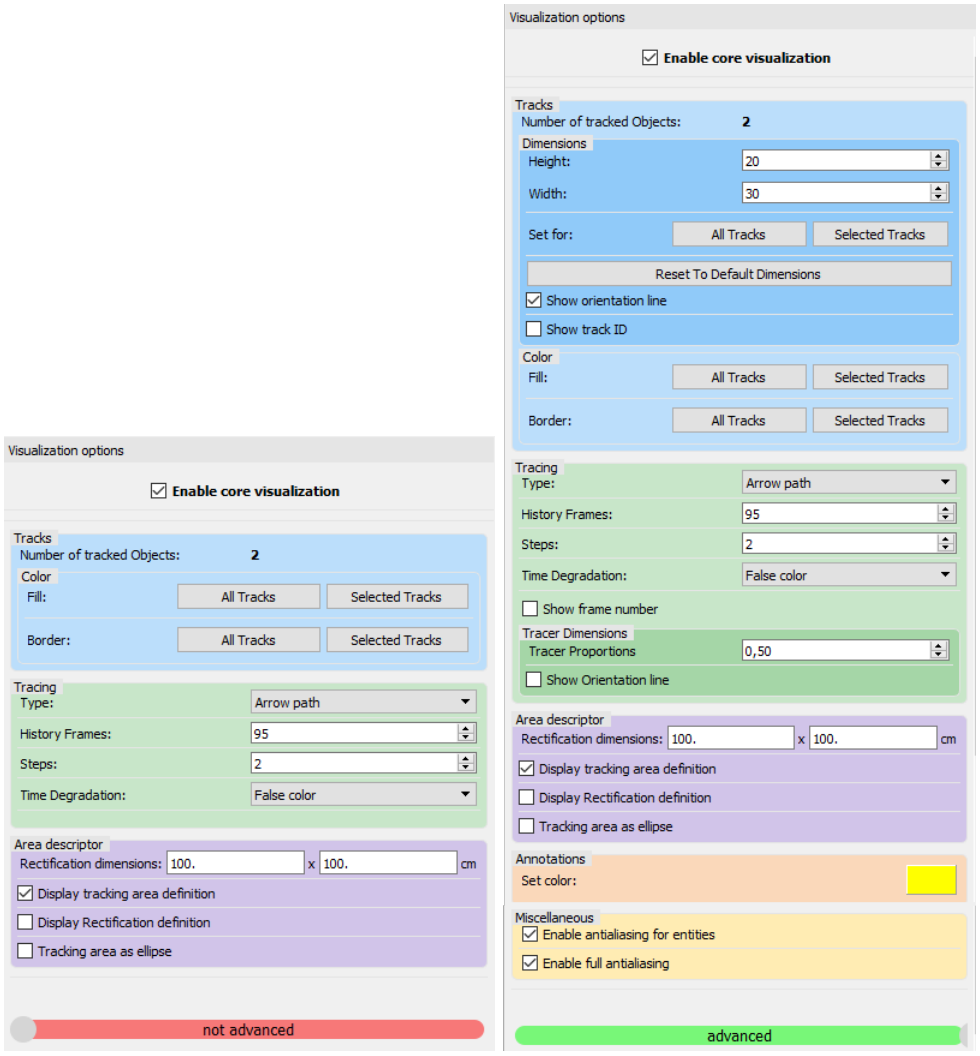
(2) "Visualization options": The "Visualization options" tab (figures 17) gives

users access to manipulating the visualization of the tracking data, changing the annotation color and managing the area descriptor settings. The settings are grouped and each group is colored differently to distinguish them from each other. The user can access more settings than illustrated in figure 17 (a) by clicking a switch in the bottom of the tab. Figure 17 (a) will then expand into (b). This was implemented to simplify the use of the BioTracker by preventing the user to be overwhelmed by too many displayed settings. The blue group box "Tracks" includes the manipulation of the shape objects. The user can for example change color and size. Changing any setting here will change the appearance of all current and future shape objects. The green group box "Tracing" includes the tracing settings. Tracers are the positions of the trajectory in the previous frames. The user can choose between different tracing types (figure 18) and set how many tracers shall be shown for each shape object. The purple group box "Area descriptor" includes all the settings needed to set up the rectification and a tracking arena. The user can display the two sets of handles (for rectification and tracking arena) in the media panel to edit them. They are set to the corners of the loaded medium by default. Then the user can enter the "Rectifications dimensions" to get valid centimeter values in the tracking data. The orange group box "Annotations" only offers the possibility to change the color of all annotations. The yellow group box "Miscellaneous" includes toggles for enabling antialiasing for either all shape objects or for the whole media panel. If the user wants to take screenshots or record the visualization these options can enhance the aesthetic quality.

(3) "Tracker options" The "Tracker options" tab (example figure 19) displays the loaded options of the current tracking plugin. These usually include settings to adjust the tracking properties so the tracker processes the image correctly.

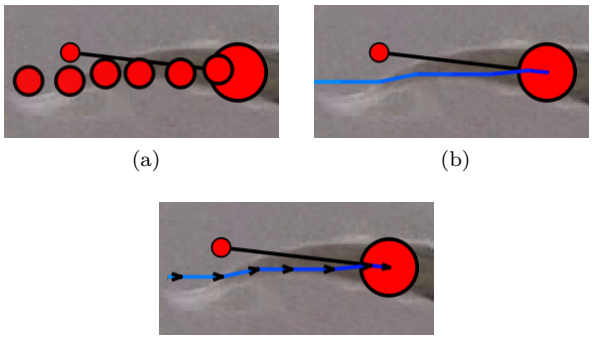
(4) "How to use the BioTracker": The "How to use the BioTracker" (figure 20) tab displays a short step-by-step guide on how to start tracking. This tab aims to give easily accessible information to the first time users. There also is a link to the user guide for more detailed guidance at the end of the test field.

(5) "Notifications": The "Notifications" tab (figure 21) displays color-coded messages from the core application, the tracking plugin and sometimes from Qt. These give the user feedback on the system status and may be important if an error occurs.



(a) (b)

Figure 17: The simplified state (a) and the expanded state of the "Visualization options" tab. The settings in the darker blue, darker green, orange and yellow group boxes are not part of the simplified state due to their unimportance.



(a) (b)

(c)

Figure 18: Types of tracers: (a) an arrow path with a false color time degradation style (ranges from blue to red the older the tracer is), (b) a path with a false color time degradation style and (c) shape tracers with no time degradation style.

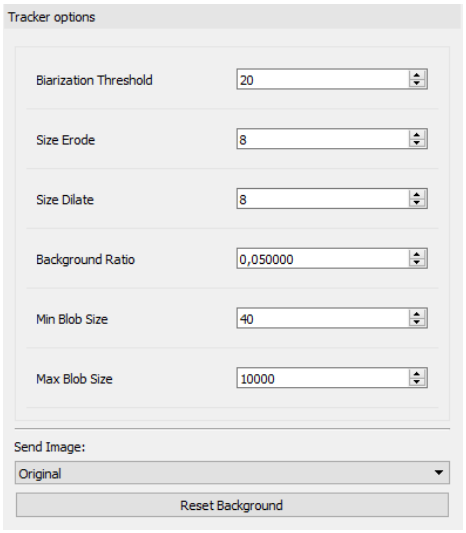


Figure 19: “Tracker options” of the “Background Subtraction Tracker”. Here the user can adjust all the settings of the tracking plugin make it work with the medium. This tracker also offers the possibility of displaying the intermediate steps of image processing in the "Send Image" drop-down list. The user can then see more clearly which settings need to be adjusted.

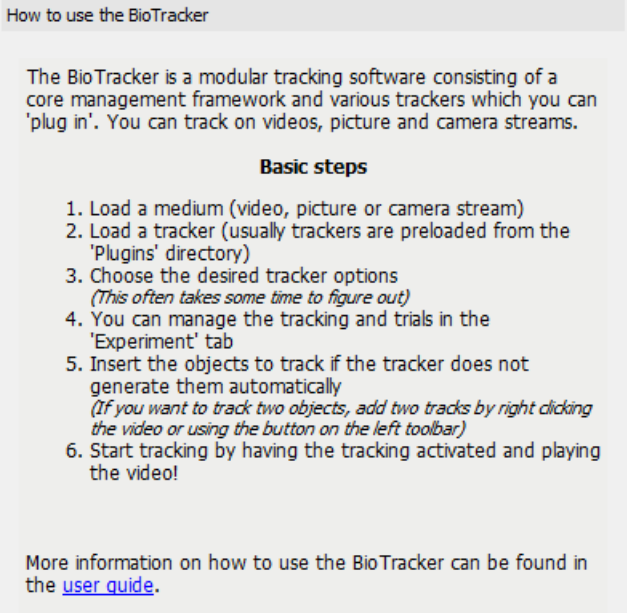


Figure 20: The "How to use the BioTracker" tab showing a step by step guide for tracking.

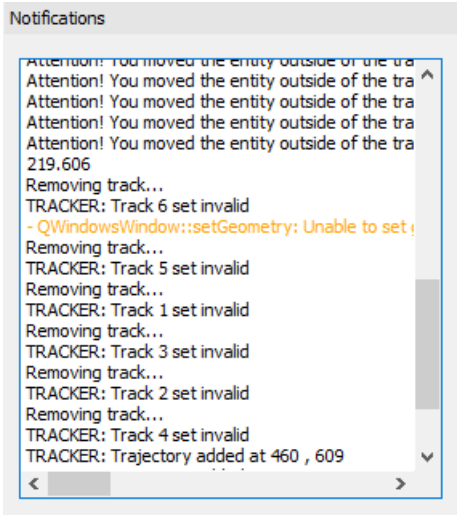


Figure 21: The "Notifications" tab. Displays messages from the core application, the tracking plugin and occasionally from Qt. It is the view of the notifications component.

6.6 Task flow diagram for tracking

The previous sections described where the user can access different functionalities. What is most important to is know how to use these functionalities to start tracking with the BioTracker. The diagram in figure 22 illustrates the typical task flow for tracking with the BioTracker. The fastest way is to load a medium and a tracker, add the trajectories, activate the tracking and start the playback of the medium. These five steps are the absolute minimum for tracking with the BioTracker.

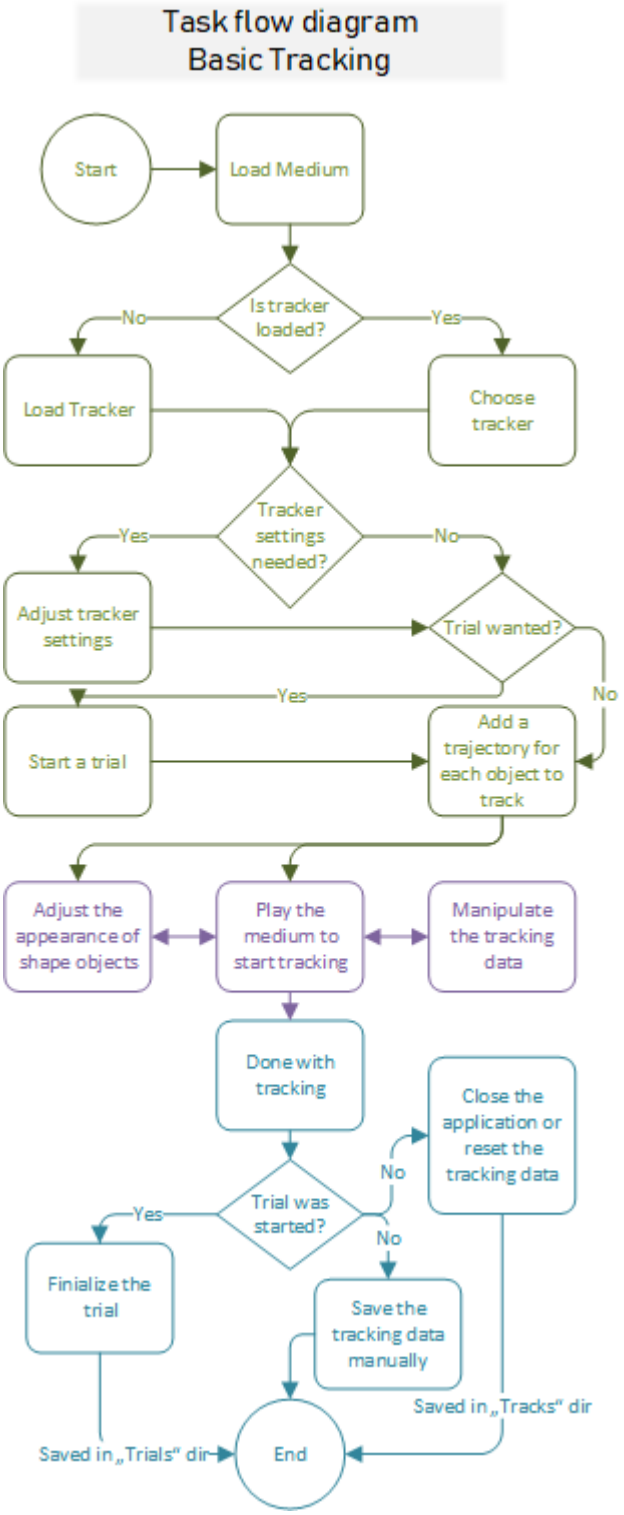


Figure 22: Task flow diagram for a basic tracking procedure. The preparation of the tracking is green, the actual tracking procedure is purple and saving the tracking data is blue.

7 Evaluation and Optimization

The evaluation is done at three levels: the performance, the user experience and an interface comparison to version two (figure 27).

7.1 Performance evaluation

The performance was tested by doing the same tracking task multiple times with different visualization settings on two different systems. After each test, the average frames per second were noted down. In the setup were 11 fish in a messy video (figure 23). Each fish was tracked. The two systems were a rather fast Windows 10 desktop PC and an old Ubuntu laptop.

The results (table 3) show that while tracking, the visualization only has an impact on the performance when moderate to high settings are set. This means that the user can enable the visualization during the tracking process but should not have too many settings enabled (tracers are the bottleneck here) to have a smooth experience. Enabling too many settings (e.g. test “Extreme Visualization settings”) can be very slow and should only be used for one frame, e.g. to take a screenshot.

The BioTracker has a fast performance for both systems tested. The tracking and visualization even run fast enough on the considerable slow Ubuntu laptop, which means that there is no real need for performance optimization. One thing to optimize could be the performance when high visualization settings are set. This would make it possible to use all functionalities during tracking.

Noticeable is that a user who wants to supervise the tracking output would most likely set an fps limit (maximum 15) to have better control. This means that the BioTracker should not be under these around 15 fps, which it only is when high visualization settings are set.

Test	Tracking deactivated	Tracking activated
No Visualization	35 fps, 71 fps (reference)	25 fps, 30 fps
Low Visualization settings	26 fps, 60 fps	23 fps, 29 fps
Moderate Visualization set.	20 fps, 45 fps	20 fps, 25 fps
High Visualization set.	9 fps, 12 fps	9 fps, 10 fps
Extreme Visualization set.	1 fps, 1 fps	1 fps, 1 fps

Table 3: Results of the performance tests. The first frames per second (fps) value of each cell is the result of the test (in average fps) on a six-year-old Ubuntu laptop with the specifications: i5 2430M, Nvidia Geforce GT 540 M, and 8GB RAM. The second value is the result of the test on a two-year-old Windows 10 desktop PC with the specifications: i5 6500, AMD Radeon R9 390 and 16 GB RAM. In the "Tracking activated" column the "Background Subtraction Tracker" was used to track the 11 fish in the medium. In the "Tracking deactivated" column the tracking data was generated beforehand to only test the visualization. "Low Visualization settings" were the default settings. "Moderate Visualization settings" included 20 styled tracers and enabled antialiasing for each shape object. "High Visualization settings" was the same test like "Moderate Visualization settings" but with 200 tracers and "Extreme Visualization settings" had 2000 tracers and all settings enabled. It is that slow due to the rendered tracer frame number.

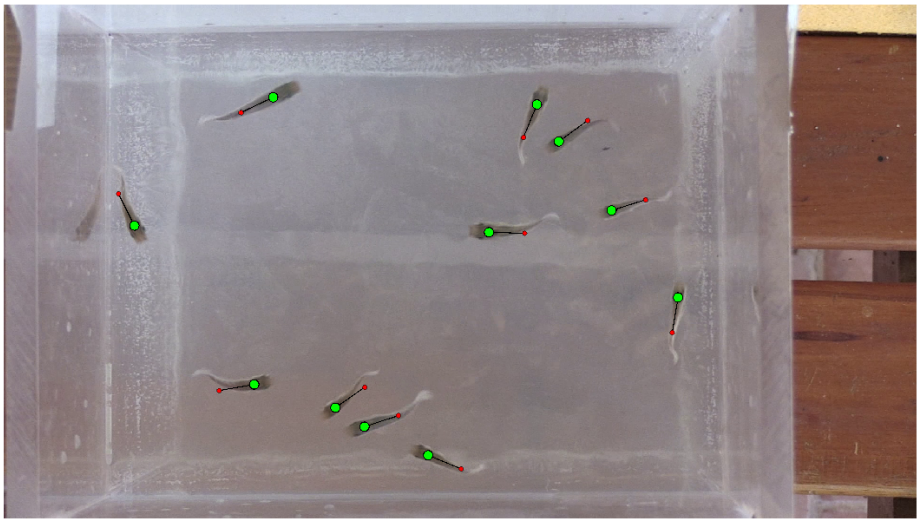


Figure 23: Screenshot of the medium during the performance tests. Currently, tracking is activated and low visualization settings are set. The video resolution is 1920x1090.

7.2 User Experience evaluation

The user experience was evaluated by conducting user tests with members of both target groups in two iterations. During these tests, the user performs given tasks with the BioTracker under supervision. He was also encouraged to think out loud. For each test, the screen and the conversation were recorded to help recognize issues in the interaction flow afterward. If for example the user does not find a feature and he talks about where he is assuming it, the feature could be moved to that position in the user interface.

Questionnaires are suitable to get a user experience quantification. It is important that the questions cover all the possible aspects of the user experience. In this thesis, the UEQ ("User Experience Questionnaire") [15] was used. It is a widely used and well-documented questionnaire with a supplementary data analysis tool for automatic evaluation.

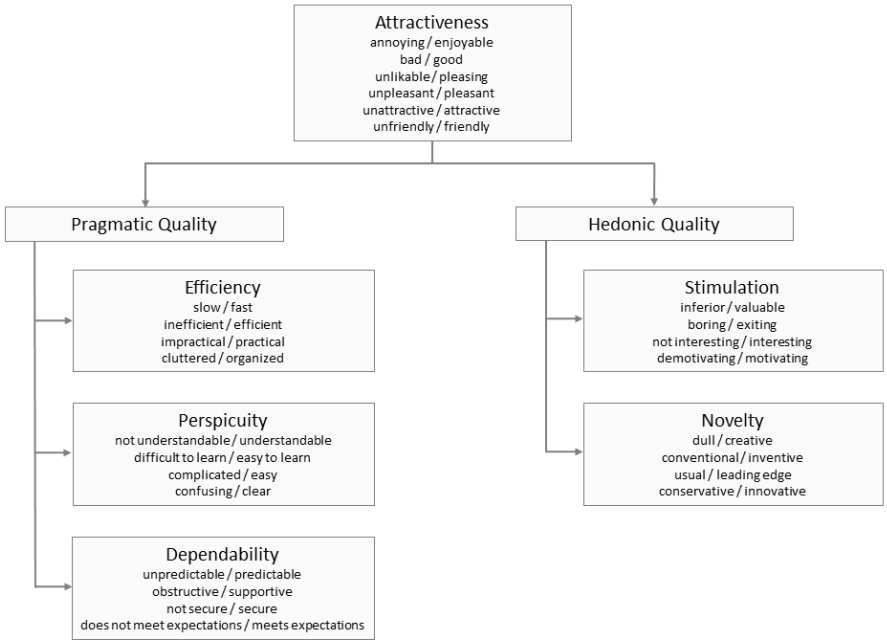


Figure 24: Structure of the UEQ. The user experience is divided into three different main aspects: the attractiveness, the group of pragmatic (goal-oriented) aspects and the group of emotional (not goal-oriented) aspects. Attractiveness includes the general impression. The group of hedonic aspects is including the aspects: perspicuity, efficiency and dependability of the BioTracker. The group of emotional aspects is including the stimulation and the novelty aspects. Each pair of opposite adjectives represents a question which the user has to answer with a value between -3 and 3. The resulting answers can be evaluated in the data analysis tool of the UEQ. Source: [23] UEQ Handbook page 3

The structure of the UEQ (figure 24) is very helpful for evaluating the user experience. For the BioTracker the concentration is on the pragmatic aspects perspicuity, efficiency and dependability because the target groups need a more usable than creative or stimulating software. Despite that, the aesthetic of the BioTracker should not be neglected.

7.2.1 Iteration 1:

During the user experience tests in iteration one, four biologists from the "IGB Leibnitz-Institut für Gewässerökologie und Binnenfischerei"[8] and one student from the Biorobotics Lab participated. All the biologists already used differ-

[8] http://www.igb-berlin.de/

ent tracking applications, like AntTrack or even self-made ones. Five test users do obviously not generate a large amount of data, which is why the UEQ results can"t be really precise. Nevertheless, they show a tendency. Additionally, the recordings of the tests and conversations during the tests provide a good understanding of current issues.

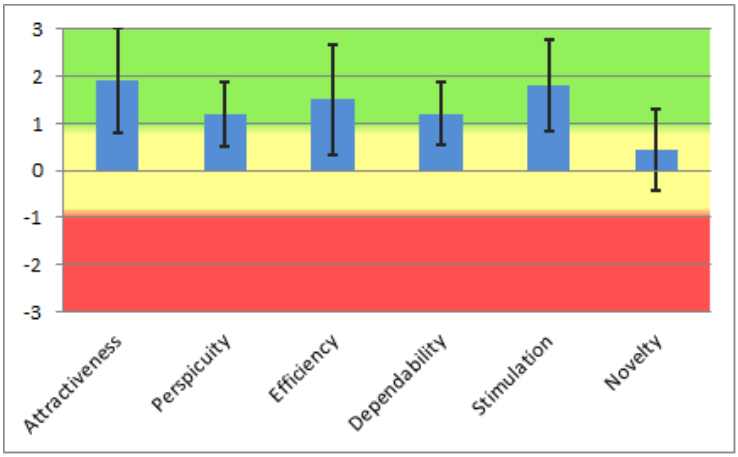


Figure 25: The means of the results from the evaluation of the user test in iteration one. The questions of the UEQ are compressed into five aspects: Attractiveness, Perspicuity, Efficiency, Dependability, Stimulation and Novelty (as seen in figure 24). The diagram goes from -3 to 3. The higher the value the better the result, e.g. "Attractiveness" has a mean of 1.9 which is very high. Each result also shows its standard deviation. This diagram was generated in the excel sheet "UEQ_Data_Analysis_Tool" supplied by the UEQ. Note: It is extremely unlikely to observe values above +2 or below -2 because of the participant's general tendency to avoidance of extreme answer categories.

The analysis tool considers results over 0.8 positive and results under -0,8 negative. The UEQ evaluation (figure 25) shows overall positive results as most of them are positive. The attractiveness (mean of 1.9) and stimulation (mean of 1.8) aspects show the highest values. This indicates that this version of the BioTracker seems to be pleasant to see and to use. The implication is that these aspects do not need much optimization in this iteration. The main focus for evaluation are the values of the perspicuity (1.2), efficiency (1.5) and dependability (1.2). These show good results as none is under 1. But one can see that there is still a need for optimization to bring these mean values to the level of attractiveness and stimulation by making the software more understandable, efficient and safe. The "Novelty" has a comparably low mean value of 0.45 showing that either the BioTracker is not creative or innovative or that the participants did not understand the aspect of novelty in

the context of the BioTracker. Either way, novelty is not a requirement or the focus of the BioTracker.

The review of the recordings[9] of the BioTracker showed several issues. Most of them correlate with the low mean values of perspicuity and dependability. If these issues are resolved during the optimization, the user test in iteration two should show better results for these UX aspects.

The main optimizations were implemented to improve the focal aspects. To optimize the perspicuity of this version more help functionality was implemented. An example is a help button for the trial functionality. To optimize the efficiency, many small functionalities, previously concentrated in one spot, were spread out to different places in the interface. For example, adding individual tracing functionality for each shape object instead of only having a general tracing for all shape objects. To optimize dependability there were added warnings before tracking data resets and an "Open file location" functionality when saving data. This aims to give the user more control over the tracking data and BioTracker in general.

7.2.2 Iteration 2:

In iteration two users tested the BioTracker in total: two biologists from the Institute of neurobiology and one student from the Biorobotics Lab. Both biologist already used other tracking applications. Despite the small number of test users, a tendency can be detected. All the aspects have improved compared to iteration one (figure 26). As 3 is the maximum, attractiveness (2.4), stimulation (2.3), perspicuity (2.3), efficiency (2.4) and dependability (2.5) show very good results. Novelty (1.0) is, as already mentioned, an aspect the BioTracker does not focus on. To improve it regardless, the BioTracker could include innovative features like 3D tracking.

The results show that the optimization goals of iteration one were accomplished and that the BioTracker's user experience overall is very good.

[9] Some of them are unfortunately corrupted.

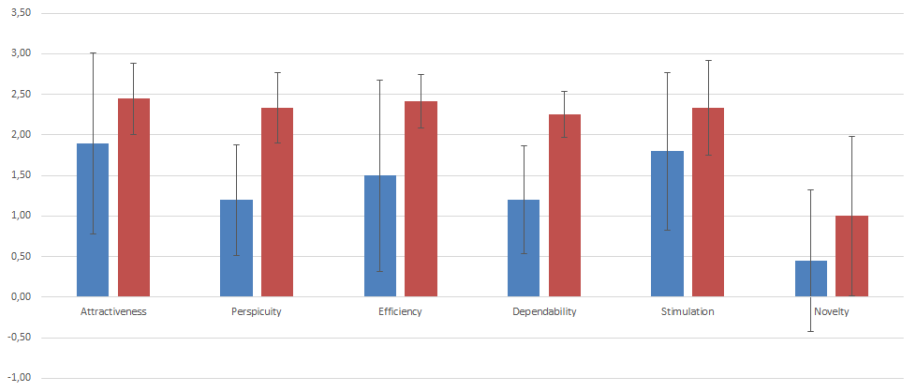
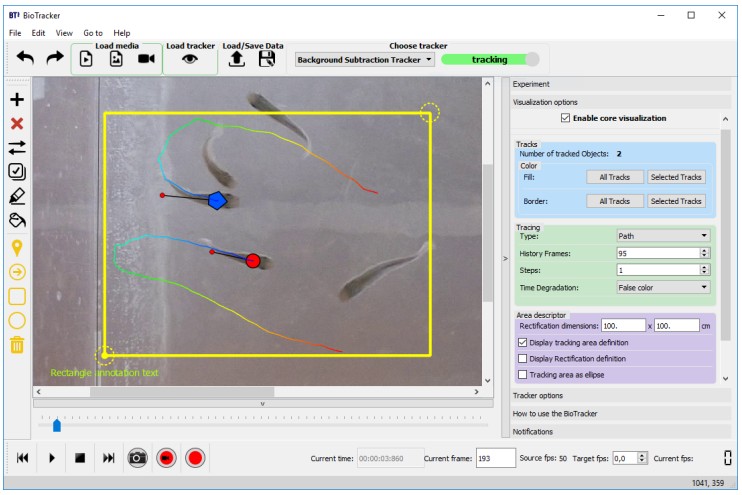
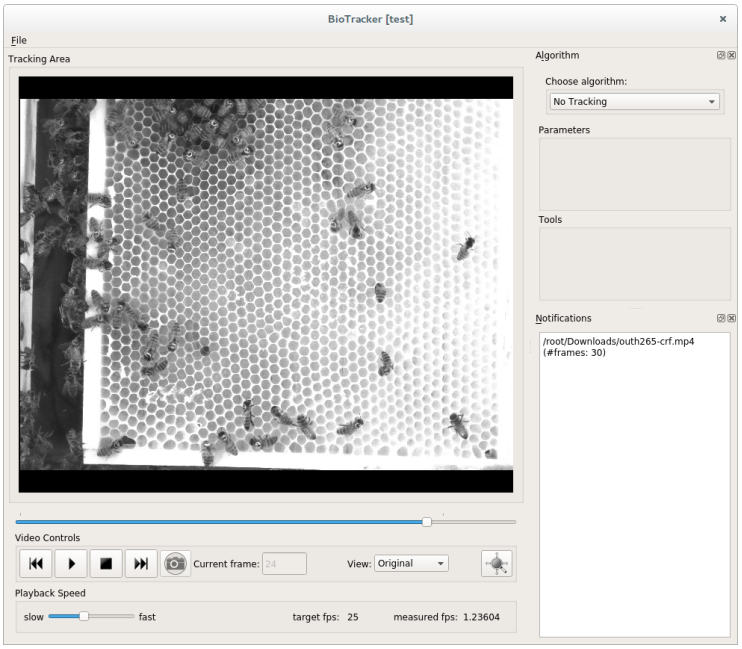


Figure 26: Comparison of the UEQ results of iteration one (blue) and iteration two (red). The diagram goes from -1 to 3. All values over 0.8 are positive. The questions of the UEQ are compressed into five aspects: Attractiveness, Perspicuity, Efficiency, Dependability, Stimulation and Novelty (as seen in figure 24).



(a)



(b)

Figure 27: Comparison between the user interface of BioTracker 2 and the current BioTracker 3. Both have a similar structure with the media panel in the middle, the media management below it and a options panel to the right of it. Version 2 is missing much of the usability functionalities (like the toolbars or the visualization options) version 3 has.

8 Discussion

8.1 Summary

For this thesis, version three of the BioTracker was expanded to have a better user experience and a more simplified way to develop a new tracking plugin. To optimize the user experience, many new functionalities, like the export and import of tracking data and various user interaction possibilities, were added. Additionally, the user interface was overhauled to include these functionalities in a structured and intuitive way. The simplification of developing a new tracking plugin was enabled by implementing a generic visualization of the tracking data into the core application framework. The result is that a developer only needs the image processing and tracking algorithms to create a new tracking plugin. Previously he also had to implement the visualization himself.

The performance and user experience evaluation show that the BioTracker is a fast tracking framework and is providing a good user experience. The attractiveness results of the UEQ also suggest that it has an appealing appearance.

8.2 Outlook

There are many features which could be included to the BioTracker in the future:

- 3D-Tracking
- Automatic error-recognition
- The automatic loading of many videos which are automatically processed using the same settings
- A "Command Line Interface" (CLI) version of the BioTracker to make it work in embedded systems
- Regions-of-interest (ROI)
- Saving and loading rectification definition files for the same kind of setup

Another consideration could be the portation of the interface from a "Qt Widgets" [8]to a "Qt Quick" [6] based system. "Qt Widgets" provides user interface elements for a traditional desktop application. It is more mature and easy than the QML-based "Qt Quick". "Qt Quick", on the other hand, is faster because it uses GPU rendering. It also provides more graphical effects and delivers features to develop a smartphone application. The portation can take a long time and thus should be considered with caution.

The BioTracker works on Windows and Linux. A Mac version could not yet be tested due to the lack of a Mac computer during this thesis. It should work, but probably needs some minor user interface adjustments. The Windows version can be downloaded as a fully functional .exe file but the Linux version

currently has to be build from source. There could also be a Linux version delivered in a package in the future.

There also is a tracking data analysis software developed for the BioTracker. It is written in Python but could nevertheless be integrated into the BioTracker's framework.

The BioTracker 3 was already published [18] and released with version 3.1. This version does not include all the functionalities created in this thesis. The rest will be included in release 3.2. If the BioTracker 3 gains popularity is the future, there could be many new tracking plugins developed from different parts of the open source community. Then, a website can be considered to collect all these plugins.

References

[1] boost. https://www.boost.org/. (visited 14.04.2018).

[2] C++ reference. http://www.open-std.org/jtc1/sc22/wg21/. (visited 14.04.2018).

[3] OpenCV. https://opencv.org/. (visited 14.04.2018).

[4] Qt 5 reference. https://doc.qt.io/qt-5.10/. (visited 14.04.2018).

[5] Qt debug message reference. http://doc.qt.io/qt-5/qdebug.html. (visited 10.04.2018).

[6] Qt quick reference. https://doc.qt.io/qt-5.10/qtquick-index.html. (visited 14.04.2018).

[7] Qt undo framework reference. http://doc.qt.io/qt-5/qundo.html. (visited 18.04.2018).

[8] Qt widgets reference. https://doc.qt.io/qt-5.10/qtwidgets-index.html. (visited 14.04.2018).

[9] K. Branson, A. A. Robie, J. Bender, P. Perona, and M. H. Dickinson. High-throughput ethomics in large groups of drosophila. *Nature Methods*, 6:451 EP –, May 2009. Article.

[10] A. I. Dell, J. A. Bender, K. Branson, I. D. Couzin, G. G. de Polavieja, L. P. Noldus, A. Pérez-Escudero, P. Perona, A. D. Straw, M. Wikelski, and U. Brose. Automated image-based tracking and its application in ecology. *Trends in Ecology & Evolution*, 29(7):417 – 428, 2014.

[11] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[12] A. Jörg. Biotracker: Design and implementation of a computer vision framework for animal tracking. Master's thesis, University of Applied Science Kempten, 2018.

[13] T. Landgraf, D. Bierbach, H. Nguyen, N. Muggelberg, P. Romanczuk, and J. Krause. Robofish: increased acceptance of interactive robotic fish with realistic eyes and natural motion patterns by live trinidadian guppies. *Bioinspiration & biomimetics*, 11 1:015001, 2016.

[14] T. Landgraf and R. Rojas. Tracking honey bee dances from sparse optical flow fields. 01 2007.

[15] B. Laugwitz, T. Held, and M. Schrepp. Construction and evaluation of a user experience questionnaire. In A. Holzinger, editor, *HCI and Usability for Education and Work*, pages 63–76, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[16] T. Lochmatter, P. Roduit, C. Cianci, N. Correll, J. Jacot, and A. Martinoli. Swistrack - a flexible open source tracking software for multi-agent systems. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4004–4010, Sept 2008.

[17] O. Mirat, J. R. Sternberg, K. E. Severi, and C. Wyart. ZebraZoom: an automated program for high-throughput behavioral analysis and categorization. *Front Neural Circuits*, 7:107, 2013.

[18] H. J. Mönck, A. Jörg, T. von Falkenhausen, J. Tanke, B. Wild, D. Dormagen, J. Piotrowski, C. Winklmayr, D. Bierbach, and T. Landgraf. Biotracker: An open-source computer vision framework for visual animal tracking, 2018.

[19] J. Nielsen. Iterative user-interface design. *Computer*, 26(11):32–41, Nov. 1993.

[20] J. Nielsen. Enhancing the explanatory power of usability heuristics. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '94, pages 152–158, New York, NY, USA, 1994. ACM.

[21] L. P. J. J. Noldus, A. J. Spink, and R. A. J. Tegelenbosch. Ethovision: A versatile video tracking system for automation of behavioral experiments. *Behavior Research Methods, Instruments, & Computers*, 33(3):398–414, Aug 2001.

[22] W. S. Rasband. ImageJ: Image processing and analysis in Java. Astrophysics Source Code Library, June 2012.

[23] M. Schrepp. User experience questionnaire handbook, 09 2015.

[24] I. O. F. Standardization. *Ergonomics of human-system interaction; Part 210 - ISO 9241-210. Human-centred design for interactive systems.* ISO, Berlin.

[25] I. O. F. Standardization. *ISO 9241-11 - Ergonomic Requirements for Office Work with Visual Display Terminals (VDTs): Part 11: Guidance on Usability.* ISO, Berlin, 1998.