Pose-Trigger

Release v1.1.1

Keisuke Sehara, Paul Zimmer-Harwood

Jan 23, 2021

CONTENTS:

Pose-Trigger is a python application for real-time, closed-loop application of TTL trigger generation based on the pose of the subject.

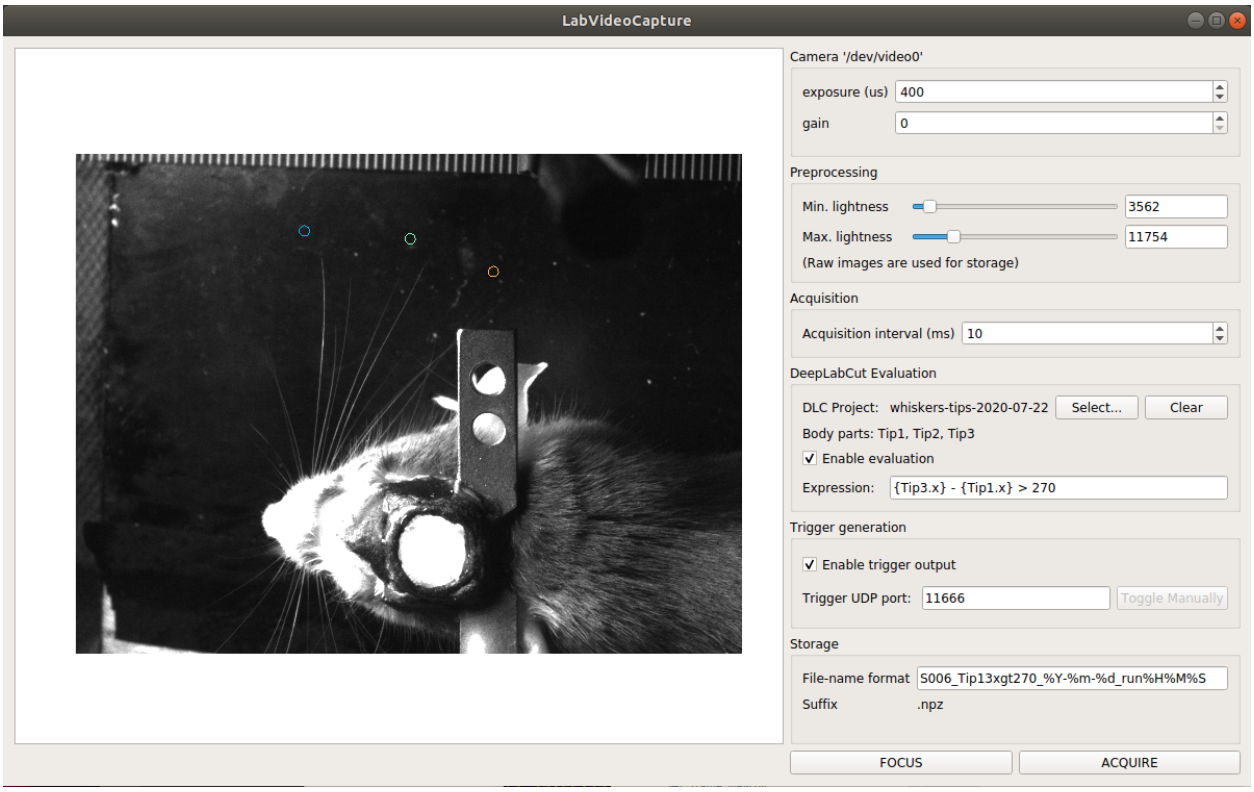

Fig. 1: A screenshot of a working Pose-Trigger app (development version)

CHAPTER

ONE

INTRODUCTION

Contents

1.1 What Pose-Trigger can do

Pose-Trigger is designed to work on a linux computer equipped with a high-speed video camera.

The current version of the software features:

- Acquisition of **high-speed videos** (up to 100-200 fps without on-line pose estimation).
- On-line exposure/gain adjustment.
- Adjustment of acquisition intervals.
- **On-line estimation of body-part positions** using DeepLabCut.
  - On-line evaluation of **arbitrary posture conditions** based on the estimated body-part positions.
  - **Fast output-trigger generation** (<1 ms) using the FastEventServer program.
- **Brightness/contrast adjustment** for on-line display.
- **Storage of frames** into the NumPy-style zip archive.

1.2 How Pose-Trigger works

Pose-Trigger is essentially a Python application. You can install Pose-Trigger on a Linux computer, and run from Terminal by typing:

```
$ pose-trigger
```

(The `$` character represents a prompt. You are not supposed to type it)

1.2.1 The model setup

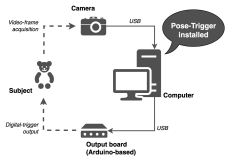

Fig. 1: The model setup

Above is the model setup that uses Pose-Trigger. Pose-Trigger is designed to work in a closed-loop experiment setup, where a single PC acquires video frames from the camera and generates trigger output based on the behavior of the subject.

For more detailed system requirements, refer to the *System requirements* section.

1.2.2 The main acquisition loop

Below is the schematics for the main acquisition loop:

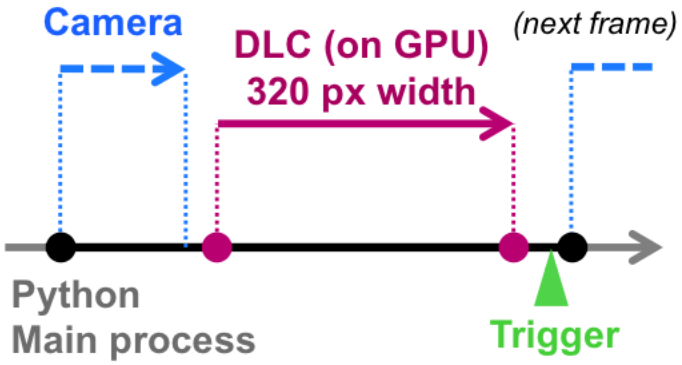

Fig. 2: The main acquisition loop

1. The **timer** generates timings for the acquisition of the next frame (black filled circles).
2. Pose-Trigger commands the camera to acquire a video frame, and receive it (blue dashed arrow).
3. Pose-Trigger delegates body-part estimation to the underlying DeepLabCut library (in case it exists; magenta arrow).
4. Pose-Trigger updates the status of trigger output by sending information to FastEventServer (in case it is serving; green arrowhead).

CHAPTER

TWO

SYSTEM REQUIREMENTS

Contents

2.1 Base requirements

Before installing Pose-Trigger, make sure you have set up the following hardware:

1. **A linux computer** (we tested on Ubuntu 18.04 LTS)
2. **a 16-bit monochrome video camera** from ImagingSource (e.g. refer to the *Reference setup specifications*).

   Note: Other Video4Linux2-compliant cameras should also work with a few adjustments in the code, but will require some efforts.

3. For a faster working of DeepLabCut, **NVIDIA graphics board with a large amount of RAM** is required.

   Note: For example, running DeepLabCut on ResNet-50 requires ~10.6 GB of RAM, so we use GeForce RTX 2080 Ti that has 11 GB on-board RAM (refer to the *Reference setup specifications*).

2.1.1 Requirements for trigger-output generation

In addition to the pose-estimation feature, the trigger-output feature requires the followings:

1. The trigger-output server ("FastEventServer").
2. An output board based on Arduino or its clone.

For Intel 64-bit CPUs, **Pose-Trigger comes with the working FastEventServer program**; you don't need to install it manually. For other architectures (e.g. AMD and ARM CPUs), refer to *Appendix: Compiling FastEventServer*.

Preparation of the Arduino-based output board may be non-trivial. Please refer to *Appendix: Preparing an Output Board*.

2.2 Reference setup specifications

We develop and test Pose-Trigger in the following environment:

2.2.1 Hardware

Table 1: Reference setup hardware specifications

Part name	Model type
CPU	3.7 GHz Core i7-9700K
RAM	64 GB DDR4-3200
GPU	NVIDIA GeForce RTX 2080 Ti (11 GB RAM)
Camera	ImagingSource DMK 37BUX287
Output board	Arduino UNO, rev. 2 (clone), with arduino-fasteventtrigger

2.2.2 Software

Table 2: Reference setup software environment

Software	Specification
Operating system	Ubuntu 18.04 LTS
Python environment	Anaconda3, Python 3.7.7
CUDA Toolkit	version 10.1 (through *conda*)
Tensorflow	version 1.13.1 (*tensorflow-gpu* package of *conda*)
DeepLabCut	version 2.1.3
NumPy	version 1.19.1 (through *conda*)

CHAPTER

THREE

INSTALLATION

3.1 Software installation

We recommend installing everything through Anaconda3. Make sure your PC satisfies the *System requirements*.

Find the environment file `posetrigger.yaml` from the repository, and run the following command in the terminal:

```
$ conda env create -f posetrigger.yaml; conda activate posetrigger
```

You can change the name of the environment by giving the alternative as `conda env create -n <name> -f posetrigger.yaml`.

Note: Please be noted that Python version must be equal or above 3.7. Otherwise, some functionality won't work properly.

3.2 Hardware installation

In addition, to make use of the trigger-output feature, you need to *prepare an output board*.

CHAPTER

FOUR

PREPARING AN OUTPUT BOARD

Contents

There are three ways to prepare a trigger-output board:

1. Write a sketch to Arduino UNO (or its clone)
2. Write a sketch to Arduino Leonardo (or its clone)
3. (Optimal) Flash the `Arduino-fasteventserver` kernel to Arduino UNO (or its clone)

Before starting, make sure about *the path to your Arduino board*.

4.1 Writing a sketch to Arduino UNO

Although this may add a 1-2 ms overhead to the output latency, this is probably the simplest way. Any UNO clone (including Nano clones) should work.

1. Find the `SimpleArduinoOutput` sketch from the `libraries` directory of the repository.
2. Using the Arduino app, compile and write the sketch to your UNO clone.

To use this type of output boards, select `uno` as the "driver type" of FastEventServer.

By default, the trigger output comes out of the pin `GPIO13` (LED).

4.2 Writing a sketch to Arduino Leonardo

Leonardo- and Micro- clones fall into this category. Boards of this type may also add a 1-2 ms overhead to the output latency.

1. Find the `SimpleArduinoOutput` sketch from the `libraries` directory of the repository.
2. Using the Arduino app, compile and write the sketch to your Leonardo clone.

To use this type of output boards, select `leonardo` as the "driver type" of FastEventServer.

By default, the trigger output comes out of the pin `GPIO13` (LED).

4.3 Flashing Arduino-fasteventoutput to a UNO clone

This method makes use of the arduino-fasteventtrigger project. Use the `leonardo` driver to use a board of this type from FastEventServer.

By using this method, the trigger-output latency will go down to the sub-millisecond order. Nevertheless, it takes some additional procedures to follow.

Caution: `arduino-fasteventoutput`, in reality, will **only make use of the serial-to-USB conversion tip on the UNO (i.e. ATmega16U2)**. This means:

- Make sure that your UNO clone has the ATmega16U2 as its converter chip.
- Other USB-based boards that uses the ATmega16U2 chip *may* work (not recommended nor supported).

To flash a kernel to ATmega16U2, we need to turn the chip into the "Device Firmware Update" (DFU) mode, by which you can send the kernel data directly through the USB cable. You can learn more about the DFU mode here on the official website.

First find the `Arduino-fasteventoutput.hex` binary from the `libraries` directory of the Pose-Trigger repository. The rest of the procedures are as follows:

1. **Install ``dfu-programmer``**: e.g. on Ubuntu, it can be simply done by running `sudo apt-get dfu-programmer`.
2. **Put the UNO board into the DFU mode, being left connected to the computer**:

   for UNO rev. 2 and later, the board can be put into the DFU mode by briefly connecting the *RESET* and *GND* pins of the 6-pin header connected to the ATmega16U2 chip. The image below shows the positions of the *RESET* and *GND* pins (and their 6-pin header):

   Note: After going into the DFU mode, **the path to the UNO device will disappear from the "/dev" directory** (so don't worry about it).

3. **Erase the previous kernel**: run `sudo dfu-programmer atmega16u2 erase`.
4. **Write out our kernel**: run `sudo dfu-programmer atmega16u2 flash Arduino-fasteventoutput.hex`.
5. **Reset (re-boot) the UNO**: run `sudo dfu-programmer atmega16u2 reset`.

   Check that the UNO board appears again under the `/dev` directory.

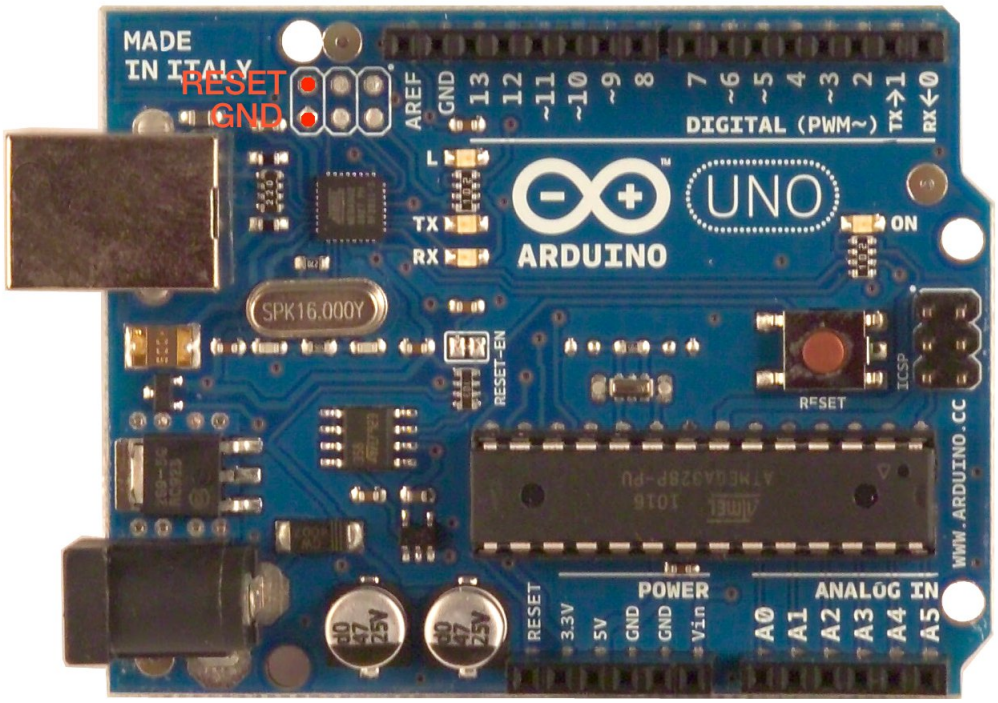

Fig. 1: The location of the `RESET` and the `GND` pins

After the procedures, you can use the board as the `leonardo`-type output board.

The trigger comes out of the `PB1` pin of the 6-pin ATmega16U2 header:

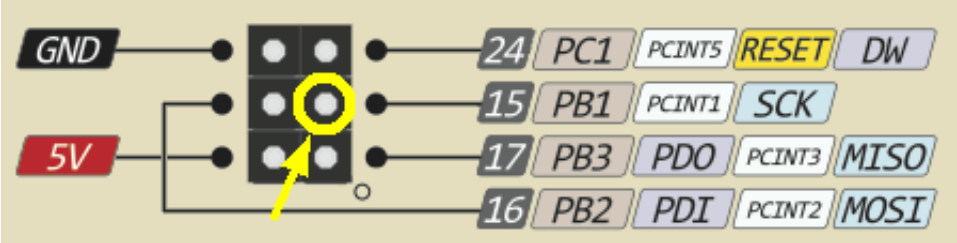

Fig. 2: The location of the `PB1` pin.

Note: By writing `fasteventoutput` to the Arduino board, **it cannot be used any more as an Arduino**.

In case you want to "resume" the Arduino functionalities, write back the official Arduino firmware using `dfu-programmer` again by following the same procedures.

4.4 Testing the board using the serial commands

You can test the boards by sending the single-character commands **without launching FastEventServer** (e.g. through the serial monitor of the Arduino app).

The setting of the console:

- **Serial port**: Your Arduino board
- **Baud rate**: 230400 baud (it matters only when you use the sketch)
- **Newline**: "No line ending" (the board ignores the newline characters anyway)

Table 1: List of serial commands

Name	Character	Description
`CLEAR`	`H`	Set the output to LOW
`EVENT`	`L`	Set the output to HIGH

Note that the command characters themselves do not really mean HIGH/LOW. On the side of the output board, in reality, it applies the bit flag *0x04* to the command to determine the output.

CHAPTER

FIVE

QUICK USAGE GUIDE

Contents

5.1 Launching Pose-Trigger

1. Open Terminal.
2. Run the following command on Terminal:

```
$ pose-trigger
```

Note: When being run without a parameter, Pose-Trigger will use the device at `/dev/video0` by default. In case you want to use e.g. `/dev/video1`, specify the device as the parameter, i.e. run `pose-trigger /dev/video1`.

To check which path corresponds to your camera, please refer to the *Appendix*.

5.2 Organization of the main window

The Pose-Trigger main window can be divided into three groups:

- The **Capture** buttons (yellow) are for starting/stopping acquisition.
- The **Preview** panel (green) provides an on-line preview of the acquired video frames. If estimation of body-part positions is activated (refer to *DeepLabCut evaluation*), estimated positions will be shown as colored circles, too.
- In the **Settings** panel (blue), you can configure how acquisition is performed (refer to the *Panel-by-panel guide*).

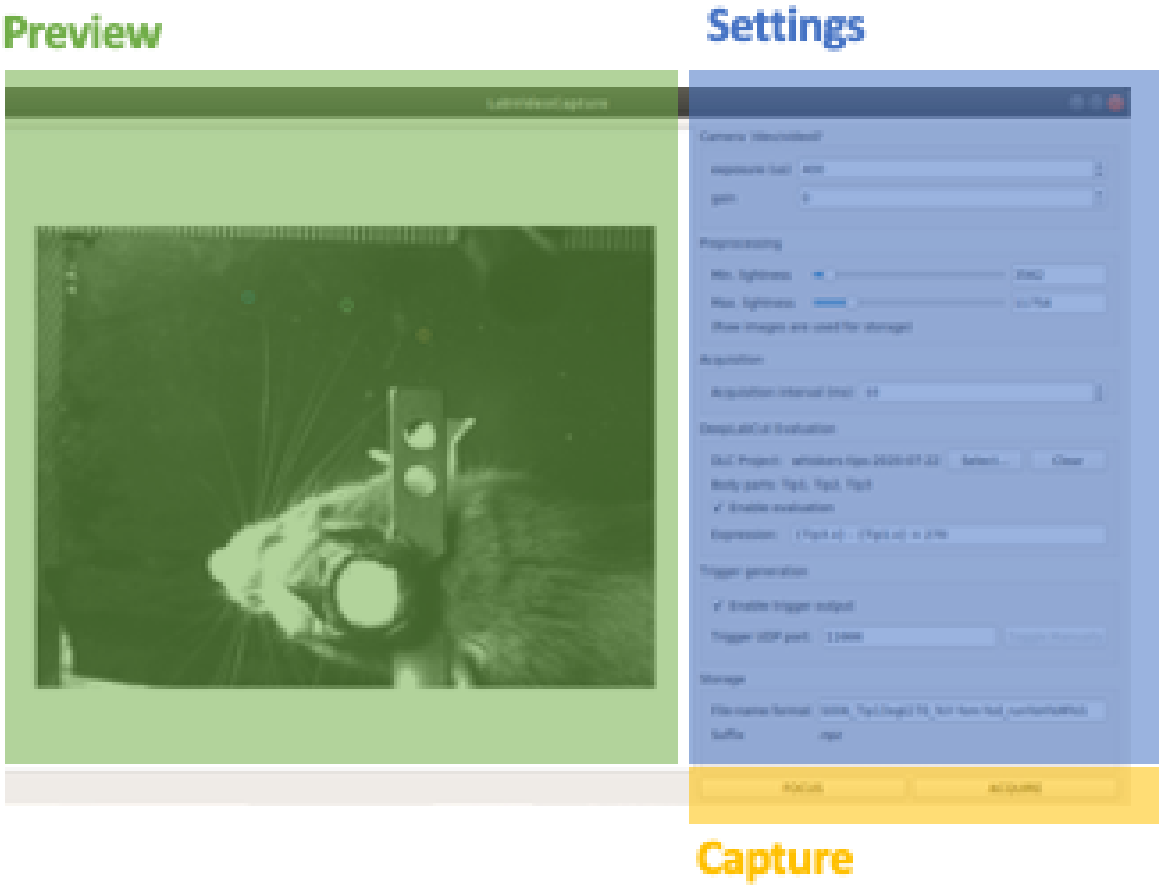

Fig. 1: Overview of the main window

5.3 Capturing videos

5.3.1 Capture modes

There are two modes of running for Pose-Trigger:

- **FOCUS mode**: capturing video frames without storing them
- **ACQUIRE mode**: captures video frames *and* stores acquired data

You can start/stop either of the capturing modes by clicking on the button at the bottom of the main window.

Caution: Pose-Trigger does !not! stream data into storage during acquisition! During acquisition, it keeps all the data in-memory. The data will be written out to a file only *after* acquisition. The duration of acquisition will be thus limited to the order of 1–2 minutes.

Note: Currently, the following parameters are "hard-coded" and used as default:

- Image format: 640x480 pixels, 16-bit grayscale
- Timing generation: a busy-wait algorithm
- Storage format: the NumPy zip-file format (.npz)

5.3.2 Format of the saved files

The data are saved in the NumPy zip-file format (i.e. ".npz" file). Each file includes the following entries:

Table 1: Entries in saved files

Name	Always there?	Description
`frames`	Yes	the 3-D frame data, (frame-index, height, width)
`timestamps`	Yes	1-D array containing unix timestamps in seconds
`metadata`	Yes	a JSON-serialized text object containing information on acquisition configuration
`estimation`	No (Optional)	when a DeepLabCut project is selected; 3-D array with the (frame-index, parameter) shape
`trigger_status`	No (Optional)	when pose-evaluation is enabled; 1-D boolean array of evaluation results

TODO

add some examples for metadata (and probably for other entries, too)

CHAPTER

SIX

PANEL-BY-PANEL GUIDE

Contents

6.1 "Camera" panel

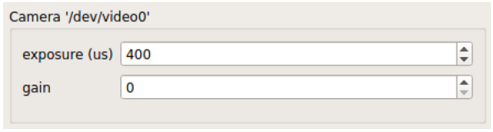

Fig. 1: "Camera" panel for capture-parameter settings

Here, you can set the exposure and the gain of each video frame acquisition.

Note: For the time being, **the image format is restricted to 16-bit grayscale, with the 640x480 frame size** (otherwise there will be an unexpected behavior).

6.2 “Preprocessing” panel

Fig. 2: “Preprocessing” settings

This controls the brightness/contrast settings for “live” video frames. It controls signal conditioning parameters for:

- Video-frame preview
- The images being fed to DeepLabCut (i.e. body-part position estimation)

On the other hand, **the raw, unconditioned images are used** for data storage.

6.3 “Acquisition” panel

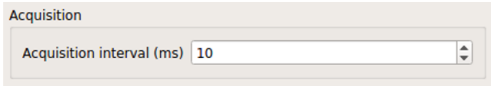

Fig. 3: “Acquisition” timing control

Here you can set the (targeted) acquisition intervals. For example, if you want to have Pose-Trigger running at 50 Hz, set this interval to 20 ms.

Note: For the time being, you can only choose to use the busy-wait timing-generation mechanism. This means that the *minimum* inter-frame interval is set to the value specified here.

6.4 “DeepLabCut evaluation” panel

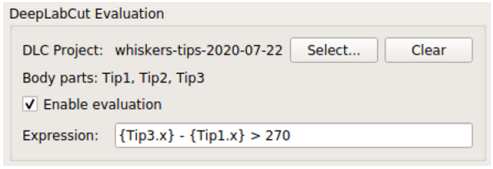

Fig. 4: “Evaluation” mode control

Here, you can configure how DeepLabCut should work in real-time.

6.4.1 Project selection

By using the "Select" button, you can select your DeepLabCut project of choice. Conversely, by clicking on the "Clear" button, you can un-set the project.

When a project is selected, the panel shows the body-part labels being registered in the project.

In addition, as long as a project is selected here, body-part position estimation occurs during video-capture processes. Estimated positions will also be stored in the data file in the case of the *ACQUIRE mode*.

6.4.2 Pose evaluation

You can enable pose evaluation by ticking the "Enable evaluation" button. Evaluation occurs using **the boolean expression entered in the "Expression" field**. The "expression" can be any Python one-line expression, but it has to be evaluated to be a boolean.

When specifying the boolean expression, you can use a **placeholder-based reference** to body part positions. For example, by entering `{Tip1.x}`, you can use the X coordinate of `Tip1` as a parameter. Other than the `x` property, you can also use the `y` and `p` properties of a body part to refer to the Y coordinate and the probability.

In computation of the expression, some major libraries can be used: use `math` for representing the `math` standard library, and use `np` to refer to the `numpy` library. For example, the expression below calculates the Euclidean distance between two body parts, `Tip1` and `Tip2`:

```
math.sqrt( ({Tip1.x} - {Tip2.x})**2 + ({Tip1.y} - {Tip2.y})**2 )
```

In addition, to enable testing of the output latency at the trigger-generation step, the custom placeholder, `{EVERY10}` is there. By using the following expression, you can toggle trigger output on and off every 10 frames:

```
{EVERY10}.get()
```

6.5 "Trigger generation" panel

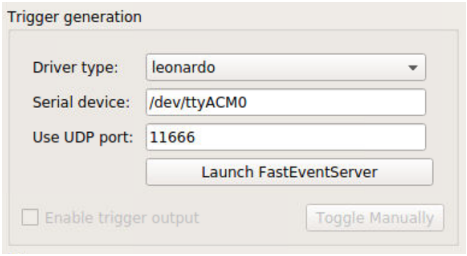

Fig. 5: "Trigger" mode control

Pose-Trigger comes with a bundled FastEventServer (currently only for Intel 64-bit systems), and manages from its startup through to its shutdown.

Here, you can manage settings on how to communicate with FastEventServer.

Before starting a behavioral session, you would typically want to:

1. Configure and launch FastEventServer.
2. Make sure about trigger output by manual trigger generation.
3. Enable trigger-output generation based on the status of pose evaluation.

6.5.1 FastEventServer settings

There are three settings you can specify:

1. **Driver type**: the type of the output board. Currently, only a certain serial devices are supported:
   - `uno`: if the output board is a UNO clone, *and* uses the `SimpleArduinoOutput` sketch (the server waits for the board to boot itself).
   - `leonardo`: if the output board is a Leonardo clone, *or* is a UNO clone flashed with the `Arduino-fasteventoutput` kernel (the board is supposed to be ready as soon as it is connected, and so the server does not wait for it).
   - `dummy`: the "dummy" output. It receives commands, but does nothing.
   - `dummy-verbose`: the "verbose" version of the dummy output. It logs the current state of the output on the console instead of generating real triggers. Note that this "logging" feature may have a certain overhead in the throughput.
2. **Serial device**: the path to your serial device. Please make sure you set a **valid path to the output board** (e.g. by checking the content of the `/dev` directory each time you plug in and out the board).
3. **UDP port**: the UDP port FastEventServer is supposed to use (defaults to 11666).

The content of the text fields will be submitted only when the return key is pressed. Until then, the color of the input turns red, indicating that your input has not been submitted.

6.5.2 Starting the FastEventServer program

By clicking on the "Launch FastEventServer" button, you can start the FastEventServer program.

The terminal console behind the Pose-Trigger app should log the output from FastEventServer. At this point, you cannot edit the server settings anymore (see the image below) until you shut down the FastEventServer process.

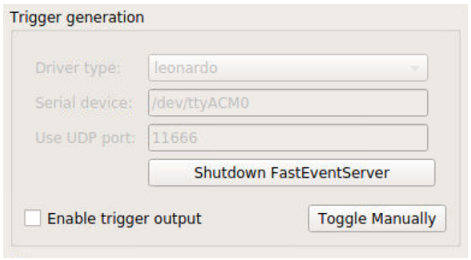

Fig. 6: What the panel will look like after launching FastEventServer

Caution: For the time being, any errors on the startup of the FastEventServer program **will not be reported until it shuts down**.

In the worst case, you think you activated the `leonardo` driver, but in reality the `dummy` driver is used and no trigger output is generated.

You have to take a close look at the terminal console, and make sure that no errors are reported. Error messages should start with asterisks e.g.:

```
***failed to initialize the output driver: ...
```

This seems to be the bug on FastEventServer (will be fixed at some point in the future).

6.5.3 Shutting down FastEventServer

By clicking on the "Shutdown FastEventServer" button, you can shut the service down at any time during your experiment. After the shutdown, you can change the FastEventServer settings, and re-launch the service again.

6.5.4 Enabling communication with FastEventServer

By ticking "Enable trigger output", the Pose-Trigger app starts sending the result of evaluation (true/false value) to FastEventServer.

Receiving the result, FastEventServer, in turn, sends command to the output board (or the dummy output) to generate the corresponding output.

6.5.5 Manually toggling the trigger

When trigger-output based on pose-evaluation status is disabled, you can manually toggle the trigger output on and off, using the "Toggle manually" button.

6.6 "Storage" panel

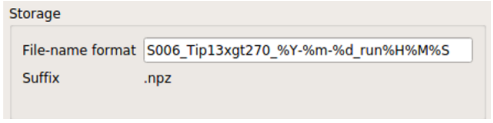

Fig. 7: "Storage" control

Here, you can control how acquired data are stored.

File names are automatically generated using the text entered in the "File-name format" field.

You can use the following **format directives**. These fields are passed on straight to the `datetime.strftime` method (refer to the python datetime module documentation on the specific format directives).

Caution: Be cautious of Pose-Trigger **automatically overwriting existing files**! Try to include (at least) the minutes/seconds directive into the file-name format, so that you do not unexpectedly delete your previous videos.

CHAPTER

SEVEN

APPENDIX: CHECKING THE PATHS TO YOUR DEVICES

Contents

When using devices like cameras or Arduino boards with applications, it is important that you know your "path" to your device in your PC. The interface to your device will typically appear as a special file in your Linux PC, using which applications can read and write data between the device.

7.1 Listing up I/O devices

Typically, files of this type are located in the directory `/dev`. You can list up the default interfaces (not only Arduino boards but also for any other I/O interfaces, including cameras) using:

```
$ ls -l /dev
```

Here, `ls` stands for the command that **l**-i-**s**-ts up the files inside a directory. By adding `-l` (a so-called "switch", to tell the command to print the files out in the **l**-ong format), you can list them up in such a way that information about each file is printed on a line.

7.2 Focusing on serial devices (like Arduino boards)

By using the `grep` command, you can pick up the files of specific names.

For example, in Ubuntu (and other Debian-based distributions), a typical serial device (like an Arduino board) should have the name starting with `ttyACM` (e.g. `ttyACM0`). Thus, by running the following command, the console will only show serial devices:

```
$ ls -l /dev | grep ttyACM
```

In the above command the `|` character is called a "pipe" that connects multiple commands, and redirecting the output of the former command to the input of the latter. Thus, the `ls` command first generates the list of files, and then the `grep` command filters the output to show only the lines that contain the characters `ttyACM`.

If you do not connect any Arduino boards, it is likely that there are not outputs there (i.e. it does not display any files info).

By plugging in and out your Arduino board, and running the above command again and again, one specific path appears and disappears in accordance. This name corresponds to **the path to your Arduino board**.

Normally, the path to a board is consistent across sessions. Thus, when specifying the "output board" for FastEventServer, you can use this path (e.g. `/dev/ttyACM0`) to select the board of interest.

7.3 Focusing on cameras

Similarly, a (video) camera typically have a path starting with `video` e.g. `/dev/video0`. You can therefore find the path to your camera by running the following command:

```
$ ls -l /dev | grep video
```

Typically, a single camera has multiple files corresponding to e.g. video and still-frame capture interfaces. You may need to check which file corresponds to the video interface of your camera (for ImagingSource cameras, it is `/dev/video0`).

CHAPTER

EIGHT

APPENDIX: COMPILING FASTEVENTSERVER

Contents

Pose-Trigger makes use of the FastEventServer project to perform low-latency, high-throughput trigger-output generation.

If you use the 64-bit Intel CPU, you probably don't have to compile FastEventServer yourself because Pose-Trigger comes with a working program. In case you use AMD, ARM etc, however, you have to have your own FastEventServer binary.

Note: In case you compile the program for any additional architecture, we appreciate it very much if you could **file a Pull Request** to the Pose-Trigger repository (or send the binary file to us)!

8.1 Cloning the repository

It contains the submodule called `libks`, so you have to populate this directory using `git submodule update`:

```
$ git clone https://github.com/gwappa/FastEventServer.git
$ cd FastEventServer
$ git submodule update
```

8.2 Compiling the program

Running `make` in the repository root should compile and update the program `FastEventServer_linux_<bitwidth>`.

8.3 Installing the program to Pose-Trigger

Pose-Trigger has its own `bin` directory inside, and looks up the appropriate program file using the `lscpu` command. **Be sure to rename your FastEventServer program accordingly!**

8.3.1 Finding the architecture of your computer

Running the `lscpu` command on your Linux computer will generate the information about the CPU:

```
$ lscpu # below is the output of this command
Architecture:        x86_64 # in case of Intel 64-bit CPU
...
```

Please note the output on the "Architecture" field, and **rename your FastEventServer program** to `FastEventServer_linux_<Architecture>` (e.g. `FastEventServer_linux_x86_64` in the above case).

8.3.2 Finding where to install

As mentioned above, Pose-Trigger looks for its own `bin` directory, i.e. `<path/to/python/posetrigger>/bin`.

You can check out the exact value of `<path/to/python/posetrigger>` by running:

```
$ python -c "import posetrigger; print(posetrigger.__file__)"
/home/mouse/anaconda3/envs/posetrigger/lib/python3.7/site-packages/posetrigger/__init_
↪_.py
```

8.3.3 Installation example

Together, you can install the FastEventServer binary e.g. by running:

```
$ make # compile
$ CPUARCH=`lscpu | grep Arch | sed -e 's/Architecture: \+/g'` # detect architecture
$ ROOTDIR=`python -c "import posetrigger; print(posetrigger.__file__)" | xargs␣
↪dirname`
$ BINDIR="$ROOTDIR/bin" # identify the directory to install
$ cp FastEventServer_linux_64bit "$BINDIR/FastEventServer_linux_$CPUARCH" # copy the␣
↪file
```