Spotter v1.1: system description and quantification

Nora Gaspar
November 14, 2017
Last update: November 14, 2017

Contents

Introduction

Compared to manual administration of experiments, automated experimental design has significant advantages in terms of speed, accuracy, reproducibility and the reduction of human errors. In behavioral experiments, especially those targeting spatial memory, the most important variable is the location and head direction of the animal. The most robust, flexible and simple way to obtain these variables is a camera-based tracking of the animal. Furthermore, automated systems enable closed-loop experiments, allowing to probe animal behavior and neuronal activity at specific points in space-time and neuronal network state.

While there are several available camera-based tracking systems, each of them target different use cases, and none of them fits to our specifications. In the following, we describe and quantify an open source software and hardware system for a closed-loop, real-time, LED-based movement tracking with analog and digital feedback signals. The system takes the camera's video as an input, and outputs real-time analog and digital signals that describe the position, speed, and head direction of the animal, among other parameters.

The first version of this system was designed by Ronny Eichler, and the documentation of that version is available here.

Specifications

Spatial resolution and jitter

Spatial resolution depends on the location of the camera relative to the recorded area. In our setup it is one pixel is **3.57mm**. This is calculated from the distance between the middle point of the two LEDs (**25mm**), which is represented as 7 pixels. Depending on the light conditions, and on the light intensity of the LEDs, there is sometimes a small spatial jitter of 1 pixel.

Temporal resolution, delay and jitter

In order to be able to record and stimulate in experiments targeting spike timing dependent plasticity, we need the temporal resolution and the signal emission latency to be in the 10-millisecond scale (ideally the temporal resolution should be over 100 frames per second), with as low jitter as possible. The current setup in fast mode is capable of a temporal delay of only **24.3124 ms,** with a jitter of **5ms**.

Capacity (maximum number of objects to be tracked, number of outputs)

One of the main advantages of a camera-based system is that it is flexible in terms of the number and types of tracked animals, and in terms of size and shape of the behavioral arena, however we state a few criteria in our design. Due to limitations on the Movement Controller, only four digital, and four analog outputs are available.

Calibration specifications (e.g. light conditions, LED's etc.)

The experiments are carried out in dark (or in very low light conditions), and in each case there are two LEDs (green and red) attached to the head of the animal. Depending on the settings, the system might also work in normal light conditions.

Since the tracking is based on the LED's, it is extremely important not have reflective surfaces in the field of view, so all reflecting surfaces, for example the walls of the behavioral box, or anything made from metal, or glass should be covered with a matte material (e.g. paint or tape).

By definition, the green LED is on the right side of the head and the red LED is on the left. The software tracks the locations of the two LEDs (Figure 1) and calculates 4 time-dependent movement parameters:

- **Position** defined as the 2D (X,Y) coordinates of the middle point between the two LEDs.
- **Head orientation** is defined as the angle of the normal vector of the line between the two LEDs. Since the distance between the two LED's is only 7 pixels, the resolution of the head orientation variable is limited. (0, arctg(1/7), arctg(2/7), arctg(3/7)….
- **Speed** is the difference between positions in two sequential frames divided by the duration of the later frame. Speed is measured in pixel/milliseconds.

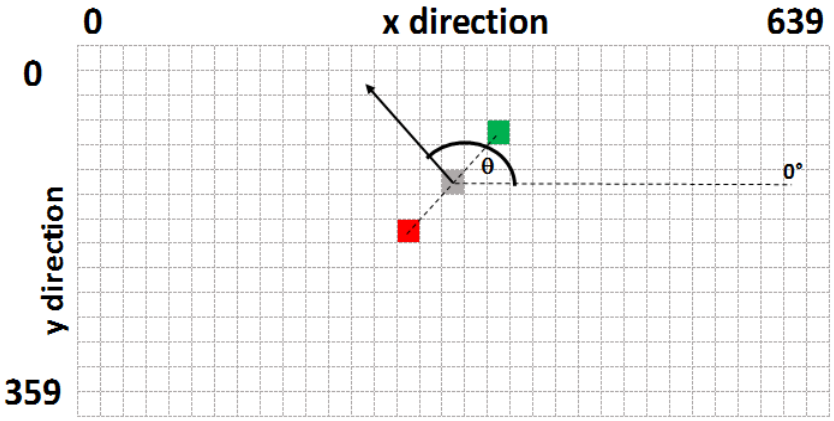

Figure 1: Field of view

Hardware setup

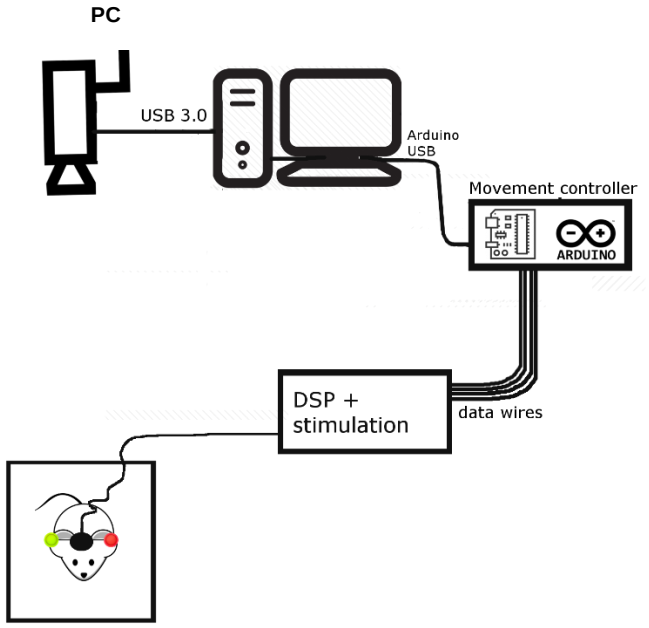

Figure 2: Hardware block diagram

The hardware consists of a camera, a PC, a microcontroller, and a custom-built PCB.

Our choice of camera is the Basler acA1300-200uc, due to its high frame-rate (up to 200 frames per second) and USB 3.0 communication, however the system works with any USB camera (e.g. Microsoft Live Studio). For the Basler acA1300-200uc camera, the maximum resolution is 1280*1024 pixels, however in our setup we record 1280*720, and resize it to 640*360 on the software.

For the PC we used an Intel i7/8 core/32 GB RAM machine running Windows 7 Enterprise. The PC communicates with the microcontroller through serial communication with the baud rate of 115200.

For a microcontroller, we chose an Arduino Mega due to the simple programming interface and the large number of output ports. The Arduino controls four 12-bit Digital to Analog converters (DAC) through SPI communication (on the custom PCB) and directly controls four digital outputs. There are four additional LEDs indicating the state of the DACs, and four more can be added to indicate the state of the digital outputs. More information about the Arduino and the custom circuit can be found in the movement controller documentation.

The 12-bit DAC's have a theoretical maximum value of 5V, and a resolution of 1/4096*5V (1.22 mV). In order to take advantage of the full range of the converter, each pixel value is multiplied by a scale factor (4096/640). This way, the measured maximum value corresponds to the number 639 (the maximum pixel value in x direction), while 0V corresponds to pixel number 0. The maximum and minimum values, averaged over 10 minutes, are shown in Table 1.

	DAC 0	DAC 1	DAC 2
Maximum	4.8818 V (σ=1.4 mV)	4.8703 V (σ=2.4 mV)	4.8706 V (σ=2.7 mV)
Minimum	-12.5 mV (σ=2.1 mV)	13.8 mV (σ=2.5 mV)	-15.6 mV (σ=2.9 mV)

Table 1: DAC extrema and SD

Software structure

The software setup is based on Python 2.7. It uses OpenCV for image processing, and is built around the Qt framework. It is based on a modular structure that consists of two main parts: Spotter, as a command line desktop application, and the Qt GUI (graphical user interface) around it, that can be turned on and off. The software structure is shown in Figure 3.

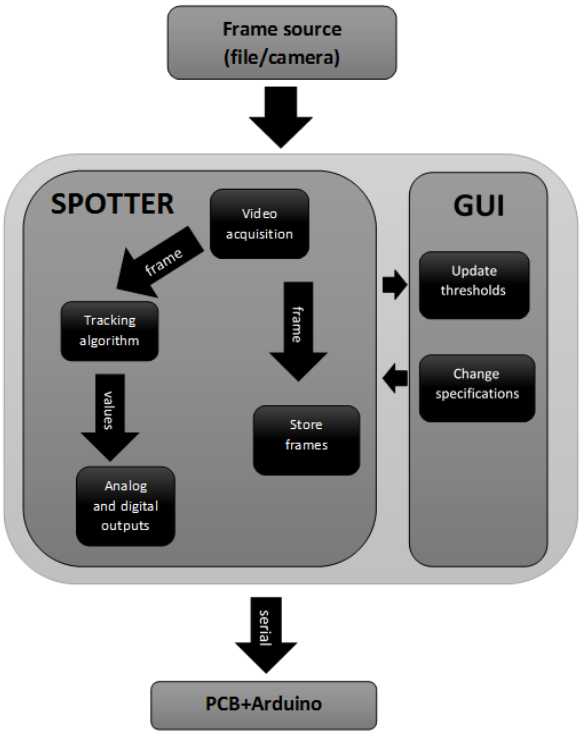

Figure 3A: Software modules by their functions

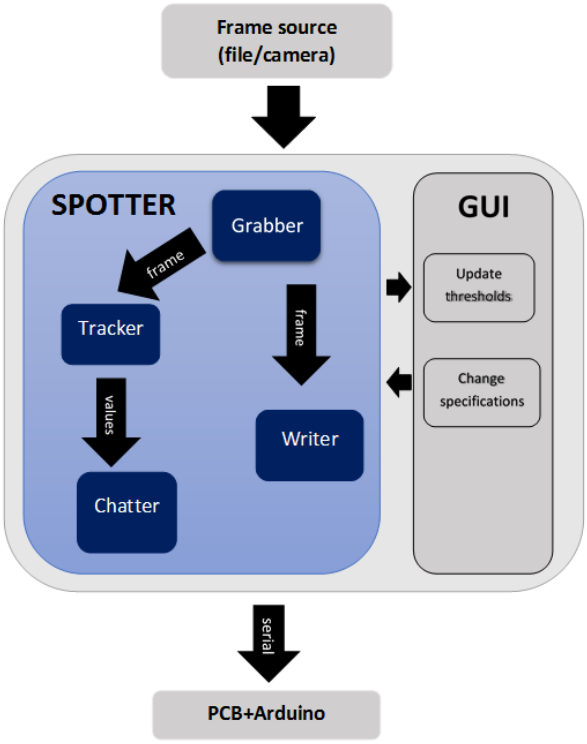

Figure 3B: Software modules by their names

Grabber

A module built with OpenCV is responsible for video frame acquisition and pre-processing. It handles the connection to the camera (or, for offline processing, the video file), grabs the next available frame, and resizes it to the required size. OpenCV handles these frames as a three dimensional NumPy array (x, y, color), where each pixel is represented as a 24 bit RGB value (red, green, blue).

Tracker

This module does the main processing of the frames. It takes an RGB frame from Grabber as an input, and outputs the requested (X,Y) coordinates, and speed or direction values to Chatter. The algorithm consist of several steps. First, the RGB frame is converted to HSV space (hue, saturation, value; Figure 5). This is important, because even though LEDs may have different colors, their color values are not immediately accessible. For example, a bright red pixel can have a higher blue component than a dark blue pixel. The HSV color scheme is a 3 dimensional cylindrical color space. Hue is encoded circularly, while saturation and value are encoded linearly. Usually the hue ranges between 0 to 359 degrees, but since OpenCV stores each channel as 8 bit data, in OpenCV hue is represented as half values ranging from 0 to 179.

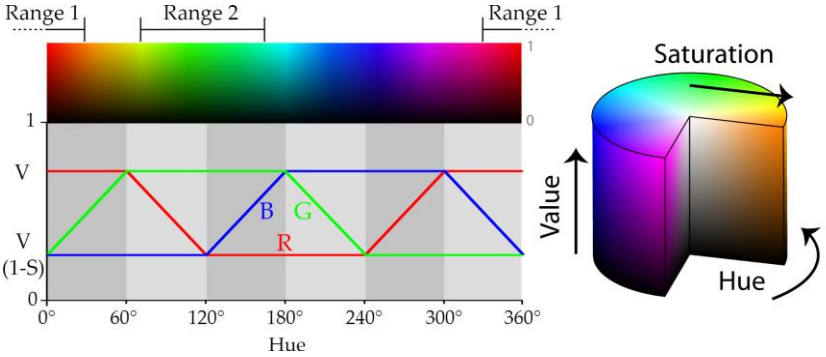

Figure 5: HSV color space

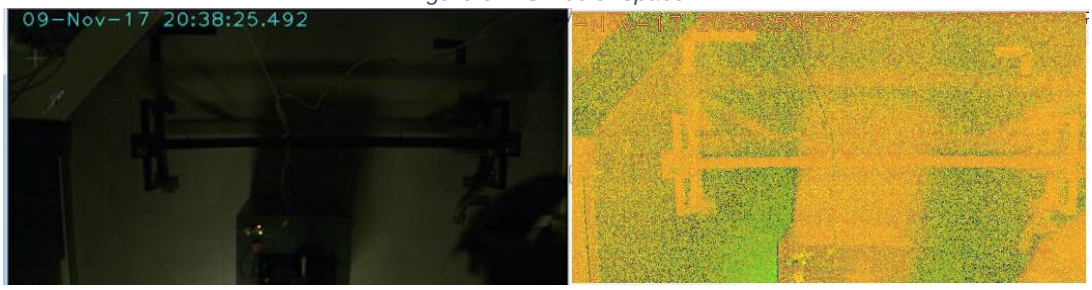

Figure 6: Converting a common RGB image to HSV

In order to optimize the process and minimize the number of pixels to be converted from RGB to HSV, a simple adaptive window is applied, where the algorithm is applied to a small rectangle centered initialized around the previous valid location (for each tracked object). If no valid location data is retrieved within this window, the window size is gradually increased.

Second, after HSV conversion, the HSV values in the window are thresholded based on the user-defined thresholds.

Third, to find objects consisting of contiguous pixels, a border-following algorithm extracts contours. The first raw moments are used to calculate the area (m_{00}) and the centroid for each contour within the window. The centroid of the contour with the largest area represents the coordinates of the feature. To avoid white pixels caused by bright LEDs saturating the RGB sensors, white pixels are simply ignored. This often results in ring-shaped contours, and may cause stability issues. For instance, if an LED does not move but its brightness is varied, the white part may change in size, and if the white part is not in the center of the LED, object movement

will be perceived. However, the noise induced by this problem is bounded from above by LED size; when 3 mm LEDs are used with 3 mm pixels, the effect is minor.

Units of tracking

1. **Features**: Features are the elementary unit of the tracking algorithm, representing a color blob (e.g. an LED). They consist of a descriptive set of parameters (hue, saturation, value, size), that are being used by the tracking algorithm.
2. **Objects of Interest**: An object consists of one or several features (LEDs). In case of an object consisting of more than one LEDs, the coordinates, the speed and the direction are calculated as the middle point between the LEDs and the angle of the normal vector between two LEDs. Currently, directionality can only be calculated for two-LED. Any of the real-time values of an object's movement parameters (x and y position coordinates, directionality, and speed) can be directly linked to one or more DACs.

3. **Regions of Interest:** Regions of interest (ROIs) are parts of the image that consist of one or more geometric shape descriptors (currently only squares). An ROI can be linked to an Object, and will then conditionally signal '1' if the Object is within the ROI, and '0' otherwise. The real-time relation of an ROI and an Object (colliding or not) can be linked to one of the digital outputs.

Chatter

This module handles all communication and connectivity with the Arduino. The module connects to the microcontroller automatically and validates the connection with a handshake message. In case there are several Arduinos attached to the system, this handshake message helps to differentiate between them, and find the one associated with the Movement Controller. The Arduino mirrors this message back, along with a report about its specifications (e.g. number of DAC's, and number of digital ports). Both chatter and the Arduino communicate with the baud rate of 115200 baud and the Arduino updates its outputs asynchronously (only the selected outputs are being updated at each frame). For each update, a four-byte packet is sent for each channel: an instruction byte, two data bytes, and a new line symbol. The instruction byte contains 3 address bits and 3 type bits, indicating the signal type (analog/digital).

Writer

To store experimental data for offline analysis, the video stream can be stored as a compressed video file in .avi format. along with a log file of timestamped position and event data. Writer is running as a separate process, and in order to maintain thread-safety, inter-process communication is handled by two queues, allowing the exchange of control messages and data.

GUI

The GUI was designed with Qt Designer and it was translated into Python code. The interface consists of the camera's image (an OpenGL context), a toolbar with control buttons, and a side panel. The task of each button can be seen in Figure 7.

Workflow

The typical workflow starts by establishing a connection of the program with the microcontroller, which can happen automatically, or require manual adjustment of the connection parameters. The user can use a live camera, or replay a previously recorded video file. Next, features are added and adjusted to yield robust tracking performance. Once the LEDs are recognized, objects can be added and linked with one or multiple features. As long as free DACs are available, object properties can be tied to analog output slots. Next, ROIs may be added. Each ROI can consist of multiple geometric shapes (presently, only rectangles). ROIs can be linked with objects, and the signal triggered by their collision can be linked to a digital output channel. Once the tracking and triggering are set up, the settings can be stored as a template for easy recovery and offline analysis (not shown). Keep in mind, that the allocation of Objects and ROI's to DAC's and digital outputs are not stored in the templates, they need to be adjusted each time the program is restarted. During the experimental run, video data, position and event logs can be written to the hard drive.

In order to further refine the data accuracy, it is possible to output a square wave on D3 digital output. This signal represents the frame to frame interval (it changes state in each frame, i.e. high on frame(i), low on frame(i+1), then high again on frame(i+2)), which may be useful in speed calculations made by external (i.e. non-spotter) devices.

In "normal mode", with the Basler camera, Spotter runs with the temporal resolution of approximately 61-62 FPS. (Depending on the camera's resolution and settings, different speeds can be achieved, however under 60 FPS, the processing won't be slowed down by Spotter.) It is fast enough for most applications, and in this mode, the sync signal has approximately 39 ms latency, which is also fast enough for real time applications. However in certain use cases a higher temporal resolution and a smaller jitter might be beneficial. This can be achieved by the "speed up" button, or by turning off the GUI. When the speed up button is turned on, the GUI only updates asynchronously every 20 ms. meaning that the GUI is skipping certain frames. It is important to note that the tracking algorithm and the analog/digital outputs are not modified by the speed up mode. Also the user experience does not change from this mode, since 20ms update rate equals to 50 FPS, which is still above the rate for human perception (an average movie is 24FPS). This mode significantly speeds up the system however it makes the framerate very unstable. It is also possible to completely turn off the GUI which results in a very high and stable framerate, however it complicates the user experience (see later).

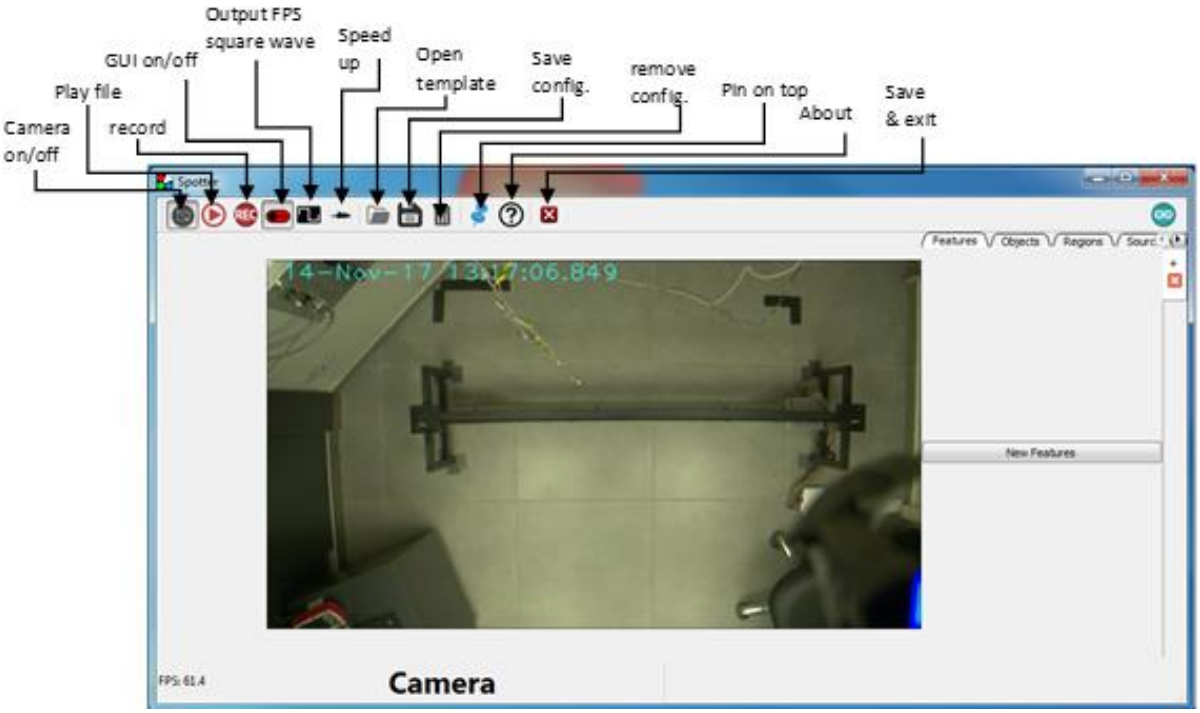

Figure 7: Spotter GUI

Sync signal

The goal of the sync signal is to synchronize between multiple systems on a second-by-second basis. In the context of the Spotter, it is used to measure overall the delay of the system, defined as the time elapsed between an event occurred (e.g. an LED was turned on) to the time Spotter identified the occurrence (e.g. a digital pin changed from low to high). The MC (Movement Controller) hardware outputs a 1 Hz square wave, which is routed to a blue LED and to the DAQ (data acquisition system). The LED is placed at the corner of the camera's field of view. To measure the delay, the blinking of the Sync LED is processed in Spotter, and the result is outputted to the DAQ. The delay between the two square waves is the delay of the system, and the jitter is the standard deviation of the delay. Depending on the processing mode we choose and on the load the system has (e.g. number of objects to process, number of outputs, etc.) the delay and the jitter can vary.

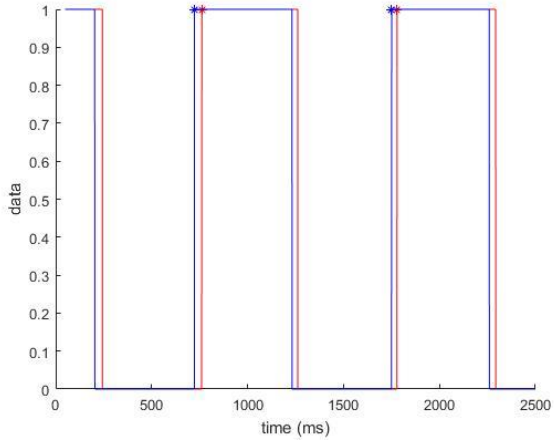

Figure 8: Example of several Sync cycles. Blue trace, sync signal; red trace, Spotter digital output.

Analog and digital delay

Since the sync signal only measures temporal delay for the digital output, it is important to compare the results for analog and digital delays too. Analog delay was measured through outputting the X coordinate of the Sync object on one of the analog outputs. The scaled results of the measurement can be seen on the figure below. As it is visible on the right figure, the temporal delay between the digital and analog sync signal is insignificant (1-2ms). This is due to the sequential operation of the Arduino board and of the Spotter software, where the more time-consuming tasks are carried out first, and the digital outputs are only being set at the last step. Therefore, it is safe to assume that the digital Sync signal represents the temporal latency for the entire Movement Controller system.

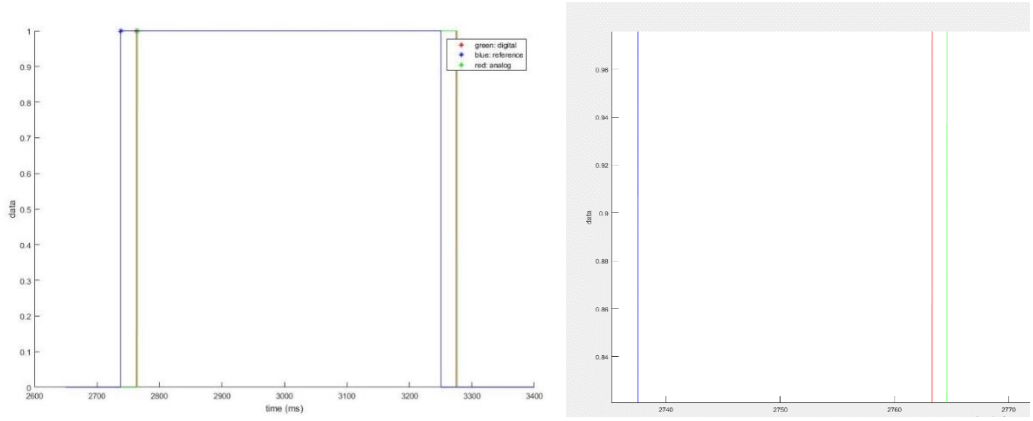

Figure 9 A&B: delay between the reference Sync signal, and the analog and the digital sync signals

Temporal delay and resolution statistics

The default frame acquisition rate on the camera is 150. The acquired frames are buffered in the computer RAM, so the perceived frame rate might be both higher and lower temporarily.

Each of the following measurements was carried out over 8-10 minutes. “Full load” measurements mean that two objects are being tracked (the sync blue LED and the head-stage green and red LEDs), and three DACs plus all four digital output channels are being used. “No load” measurements mean that only one object (the sync blue LED) is being tracked, and only two digital outputs are being used (one for FPS measurement, and one for the Sync signal). The data was recorded with the Intan board and analyzed with Matlab.

FPS measurements were done by finding each rising and falling edges of the signal, and then taking the distance between these edges and calculate their mean and standard deviation. The mean frame rate is calculated as $1000/t_{mean}$. Standard deviation values were calculated as $1000/(t_{mean} \pm t_{std})$. As can be seen in the histograms and the results below, during the “Normal” mode (GUI on, no speed-up), the SD is very small, meaning that temporal jitter is very small. However, the frame rate is lower than in other cases and is strongly affected by the load on the system. In the “Fast” mode, the average frame rate is higher and the temporal delay is very slow, but the standard deviation is still very high. This is due to the asynchronous repainting of the GUI done every 20 ms, so approximately half of the frames are slowed down, as seen by the bimodal distribution of frame durations. In case of the third, “No GUI” mode, jitter is low and the framerate is high. However, this setting makes usage and monitoring of the software more complicated.

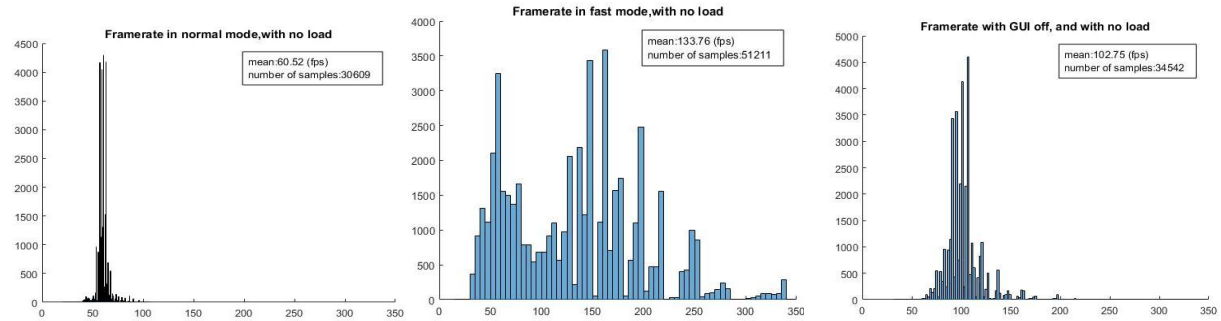

Figure 10 A,B,C: Histograms of framerate with no load in normal mode, fast mode and with the GUI turned off

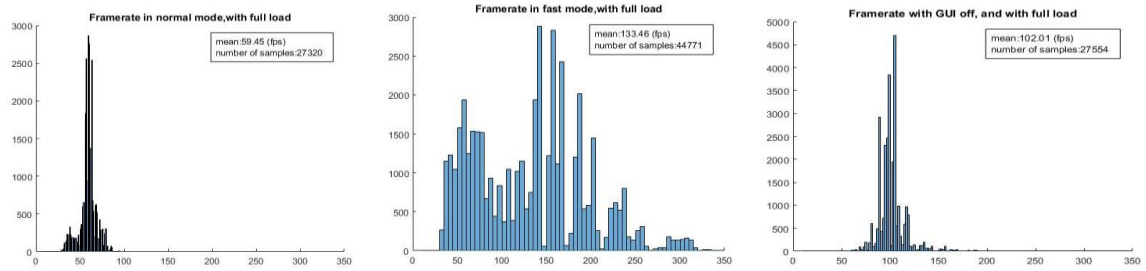

Figure 11 A,B,C: Histograms of framerate with full load in normal mode, fast mode and with the GUI turned off

Temporal resolution (FPS) - no load

	Normal	Fast	No GUI

Mean	59.9535	99.8371	99.9857
Standard deviation	54.4961 / 66.6256	62.4793 / 248.3030	86.3153 / 118.8011
resources:	16% CPU	17-20% CPU	13-15% CPU

Temporal resolution (FPS) -full load

	Normal	Fast	No GUI
Mean	57.6908	99.95	99.9786
Standard Deviation	48.1106 / 72.0352	62.393 / 251.094	86.4795 / 118.4715
resources	19-20% CPU	19-20% CPU	14-16% CPU

Temporal delay: (Sync signal difference) - no load

	Normal	Fast	No GUI
Mean (ms)	35.8416	26.9007	26.8157
Std(ms)	5.8480	8.3524	4.2224

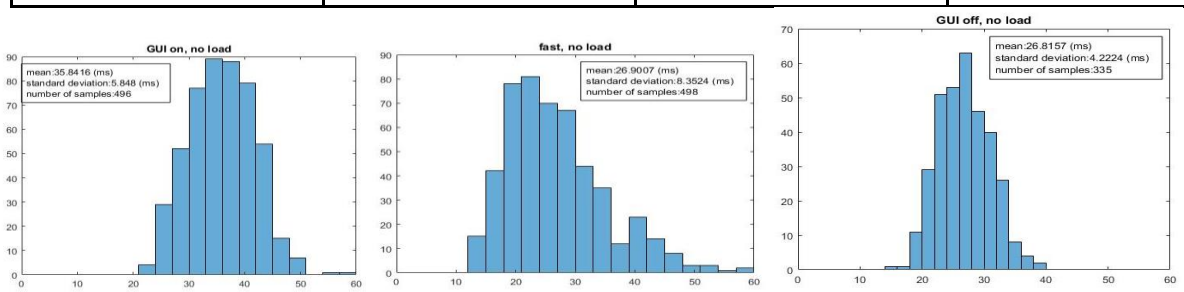

Figure 12 A,B,C: Histograms of temporal delay with no load in normal mode, fast mode and with the GUI turned off

Temporal delay: (Sync signal difference) - full load

	normal	Fast	No GUI
Mean (ms)	39.4666	29.717	27.1849
Std (ms)	7.5390	9.228	4.1299

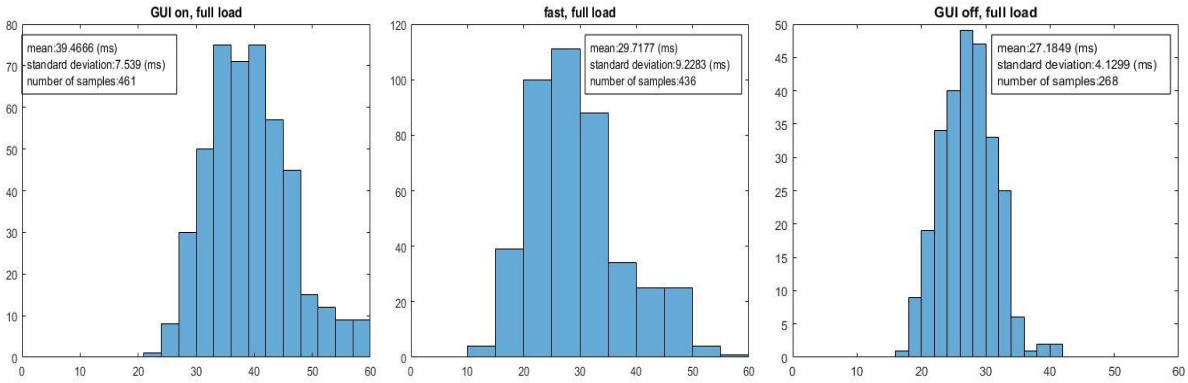

Figure 13 A,B,C: Histograms of temporal delay with full load in normal mode, fast mode and with the GUI turned off

Discussion

As a summary, we can conclude that different settings are more suitable for different use cases. All three modes of operations satisfy some of the important criteria (high temporal resolution, low temporal latency, small jitter, comfortable user interface), but may not be ideal for others.

The “Fast” mode offers the smallest temporal latency, however at the expense of a much larger jitter, and more variable frame rate. This solution might still be very useful if jitter and frame rate are not important in the application (e.g. no external speed calculations are being done)

The optimal frame rate, minimal temporal latency and smallest jitter can be reached with the GUI off. However, this is sometimes uncomfortable, because we cannot verify visually if Spotter works properly, and if something goes wrong, it’s harder to fix it… It is recommended to only turn off the GUI after all the settings are fixed, tested, and illumination conditions in the behavioral arena are kept constant. Every change in illumination conditions (entry through the curtain, change in ceiling lighting, etc.) require continuous and immediate monitoring.

The “normal” state offers a middle-ground solution between the previous two solutions, as it provides a reliable and relatively stable frame rate with smaller jitter, and an optimized user interface. However, it has a higher temporal latency and lower temporal resolution. It is recommended to use it if there is no need for the highest accuracy.

Further plans

Stand -alone system

In the near future, the PC+Arduino solution is going to be replaced by the Odroid-XU4 single-board computer. The camera will directly be connected to this device, it will run the main processes, and also directly control the outputs. The PC will only be used for calibration, and monitoring.

Improved recording

Apart from tracking and feedback, the software should work well also as a simple lightweight recording system. In order to achieve that the recording function needs to be improved.

More calibration options

More calibration options, for example different shapes of regions of interest, or manually setting the tracking area will be added later.