1 Introduction

This tool is designed to work with Industrial Cameras from Basler via their Python API, and any common projector that can be connected to the PC via HDMI, essential serving as a second monitor.

The program consists of three phases, which are executed consecutively. The first phase is the Projector Synchronization, designed to calibrate the projector and the camera. The second phase is the Background Capture, in which you take snapeshots to assemble the background with. The last one is the actual tracking and projection of the dot, as well as video writing.
The three phases and their tunable parameters are documented in detail below. If you're only interested in what the individual parameters mean, just read the "Usage & Paramaters" section of the respective phase.
In order to understand why each phase is necessary and which steps it actually performs behind the hood, read the "Functionality" sections.
Every parameter slider can be also changed by clicking it once and then pressing the arrow keys, allowing for more precise tuning.

2 Projector-Camera Synchonization

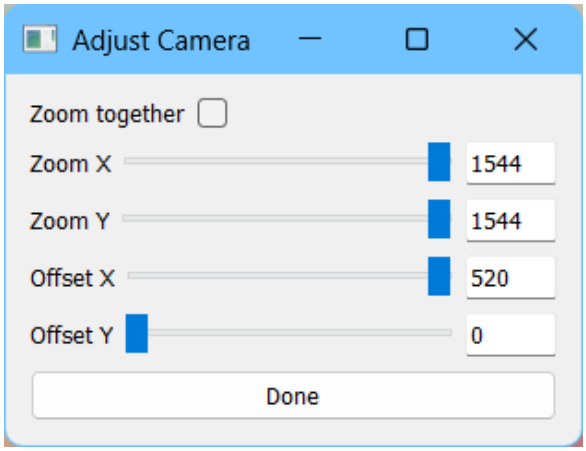
2.1 Functionality

Before anything else, it's crucial to synchronize the projector field of view with the camera field of view, such that the projection of an object is at the exact same position as the actual object is in the scene. This is critical, since we need to ensure that a dot for prey simulation, computed from the camera video input to be at some position relative to the animal in the camera image, is also properly projected onto the scene at that same position relative to the actual animal.
In order for this to be the case, this first phase of the program does nothing more than project the whole camera image live back onto the scene. Like this, the projector and camera alignment can be adjusted such that every projection overlays with the actual object it's projecting. In this phase, one can adjust the hardware, like move and tilt the camera and projector, as well as changing the digital camera parameters.

2.2 Usage & Parameters

The Basler camera has many parameters that can be changed. This tool accesses some of those parameters, just like the official Basler software does, and allows changing them. Regarding the zoom, in actuality, only the camera clip window is changed, but the tool uses this in combination with image resizing to achieve zoom: It will always resize the camera image into a square, so reducing the clip window width or height (or both) causes the program to take that smaller image and stretch it, making it a same size square again. This stretching results in a digital zoom.
In this phase, the program will show the parameter control panel on the right (depicted below), and a live camera image on the left, which shows the same image that is projected onto the scene for synchronizing with the actual objects.

- Zoom together: Checking this box causes the change of one zoom slider to also set the other to the same value, in order to maintain the same zoom across both axis.
- Zoom X: Camera clip window width. Reducing this value results in a stronger zoom along the X-Axis.
- Zoom Y: Camera clip window height. Reducing this value results in a stronger zoom along the Y-Axis.



- Offset X: When the clip window width is smaller than the maximum possible camera image width, this value allows moving the clip window along the X-axis. (Even with a maximum square image of 1544x1544, an offset of 520 is still possible, as the actual maximum camera size is rectangular and only forced to be square by the program, by limiting the clip window width by the maximum height possible.)
- Offset Y: When the clip window height is smaller than the maximum possible camera image height, this value allows moving the clip window along the Y-axis. The maximum camera image height is 1544.
- Done button: Once the parameters are tuned such that the camera and projector are synchronized, this button can be clicked to proceed to the next phase. The values are saved in the camera and will be the same upon closing and opening the program again, expect when they are changed by other programs, such as the Basler software. Therefore, this step usually has to be done only once.

3 Background Capture

3.1 Functionality

Since the tracking depends on the background of the scenery, this background has to be acquired somehow. There are two main possibilities to do this: Either, one captures the scenery before putting the animal in, or one computes the background while the animal's already present, using multiple images that differ from each other due to the animal moving around. The first approach poses the problem that the act of putting the animal into the tank usually results in displacements, shifts and other changes, i.e. water bubbles forming, after the background image was already acquired. This causes problems with the tracking. Therefore, this background capture phase implements the second approach (however, it can theoretically also be used for the first approach).

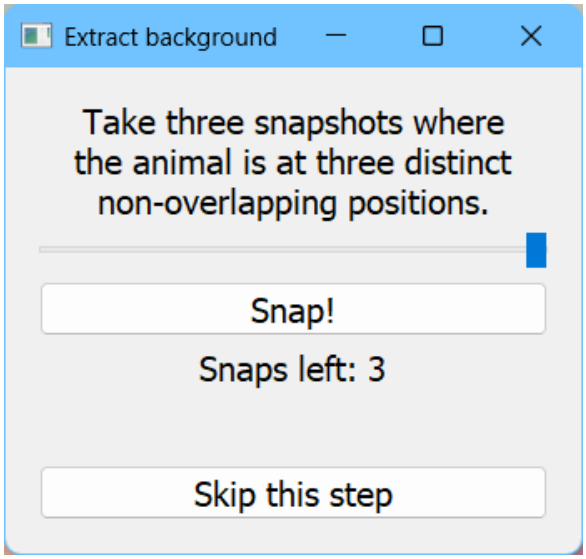
The tool needs three images only, in order to compute the background. To achieve this, for each pixel position in the output image, it looks up the corresponding three pixels in the three respective captured images and picks one of the two most similar pixels (in fact it actually takes the mean of both; theoretically they should be the same however). If a pixel happens to show parts of the animal, it can be expected to be much different than the corresponding pixels of the other two images. Therefore, the color of the two most similar pixels can be expected to show the background. This only works if there is no animal overlap in the images, otherwise the two most similar pixels could both be showing the animal.

Therefore, this phase requires the user to take three snapshots of the scene, each showing the animal at a distinct position, non-overlapping with the other two snapshots.

3.2 Usage & Parameters

Upon starting, the tool will first try to find the Basler camera and the second screen, which should be the projector. The camera is necessary for the program to continue and it will stop when there's none connected or access is not possible (e.g. because it is currently in use by another program such as Basler's own software).
A connected projector is in theory not necessary, however, this phase also allows setting the projector background brightness (projector background meaning everything that is not the dot) for later projection, showing the effect of changing it live. This is important since the actual projector background brightness might change the scenery as perceived by the camera and therefore the captured background, mainly due to the projector lamp reflection in the water.

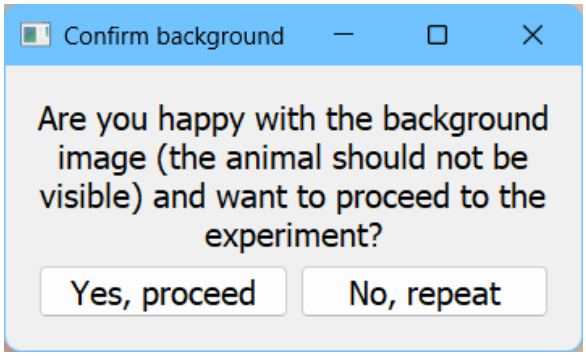
It will then go on and show the camera and a control panel with instructions next to it, as depicted on the right. Here, you can do three things:



- The Slider: Change the projector background brightness.
- The Snap! Button: Take a snapshot. Once it is taken, it will be overlayed with the camera, so you can tell whether the current animal position would overlap with the previously taken snapshots. As soon as it does not, press the button again to take the next snapshot, until there is none left.
- The Skip this step button: If you did this step before, you can skip it. This will cause the tracking program to use the previously taken background image and set projector brightness. If you skip this step but it has also never been done before, you will be warned about this by the program. The tracking phase later will then first prompt you to chose a background image file manually, and the projector background will be set to full brightness.

After all three snapshots are taken, they are assembled into the background image and the result is displayed on the window on the left. The panel on the right changes into a confirmation dialog:

Here, you can check the result and determine whether parts of the animal are still visible in the assembled background. Then click a button:



- Yes, proceed: The animal is not visible anymore on the background and you want to use it for the tracking. The program will proceed into the next phase and use the assembled image and set projector brightness for tracking and projection.
- No, repeat: For any reason, probably due to the animal overlapping in the snapshots, parts of the animal are still visible. Repeat the process, showing the previous panel again.

4 Tracker & Projection

4.1 Functionality

The tracking follows the simple approach of subtracting the background of the scene from the current camera image in each frame. Everything that differs from the background is detected as an object to be tracked. In the ideal case, this should only be the animal. In a real world scenario however, it can result in picking up small dispositions and other noise that cause differences between the background and the current frame. In order to counteract these artifacts, the tool uses tunable image erosion & dilation in order to reduce noise. An environment with minimal interferences is still necessary to achieve proper functionality.

Preprocessing

In the preprocessing, the image is merely re-scaled to a maximum allowed size of 1024 x 1024, and afterwards the difference image is computed and normalized. The maximum size is designed to maintain sufficient performance of the following algorithms.

Body Detection

The next step is the detection of the animal's body. A simple binarization is performed which is the process of throwing away all pixels below a certain threshold and only keeping the ones that are bright enough. The result will only contain black and white pixels, depending on whether they passed the threshold or not.
After that, the noise reduction is applied. Erosion will make black pixels spread out further around their position, covering potential white pixel outliers. The reverse operation is executed afterwards, restoring the strength of surviving white pixels, while not being able to restore completely removed pixels, effectively reducing noise.
Last but not least, the body centroid is computed, as it is necessary for later dot position computations and the whole reason we need to detect the body in the first place. This is achieved by using image moments (first-order moment), as shown in Tracking object orientation with image moments together with OpenCV's efficient `moments()` implementation.

Eye Detection

The eye detection is equivalent to the body detection in terms of binarization and noise reduction. The main difference is the higher binarization threshold, as the eyes are expected to appear much brigher in the difference image, since they are usually pitch black in the raw image, and the subsequent process of finding the two eye centroids.
The problem of finding two centroids requires a significantly more complex approach as opposed to just one, which is equivalent with a mean point in a binarized image. In order to retrieve two objects of which to individually compute centroids, the tool makes use of OpenCV's `findContours()` to find the contours (see Contours: Getting Started) of the binarized eye image. The two largest contours can be expected to represent the two eyes. Therefore, the eye centroids are computed as the centroids of the two largest contours, measured by contour area, making use of OpenCV's ability to compute moments of not only images (i.e. numpy arrays), but also contour objects.

Dot Position Computation

At this point, the tool has acquired the body centroid as well as the two eye centroids. The vector from one eye centroid to ther other is computed and defined as the lateral body axis. The dot base position is now computed as being the center of the two eye centroids, plus a medial distance vector, which is orthorgonal to the lateral body axis. In order to properly orient this distance vector towards the front of the animal, it always points to the opposite side of where the body centroid was computed to be. An additional lateral distance can be set, oriented towards the right side of the animal. Both distances can be tuned and set to a negative value in order to point to the respective opposite directions.
In the case of the eyes not being detected properly, the dot will simply remain at its last position, until the eyes are found again.

Note that the precise position of the dot depends on the eye centroids' position, whereas only its medial (and lateral) distance's orientation depends on the body centroid's position. The eyes are a much more robust reference, as they are easily detectable due to their clear visibility. The body centroid has the intrinstic property to always jump around slightly, which is a further reason to not use its precise position and rather to only detect on which side of the eyes it is located, by using the sign of the dot product between the medial body axis vector, starting from the center of both eye centroids, and the vector from that same center towards the body centroid, as the orientation.

After computing the dot position, a disposition is added that changes after a certain amount of frames, creating a jitter effect. This disposition strength can be tuned, selectively along the medial and lateral body axis.

Dot Projection

The last part of the main tracking loop is drawing the dot onto a blank canvas image, and displaying this image on the second screen, which is supposed to be the projector. Additionally, the dot will be drawn on the image displayed in the control window and into the resulting video output. The control window image will also contain markings for the dot vector (i.e. medial + lateral axis vectors), body centroid, eye centroids, eye centroids' center and eye centroids' vector.

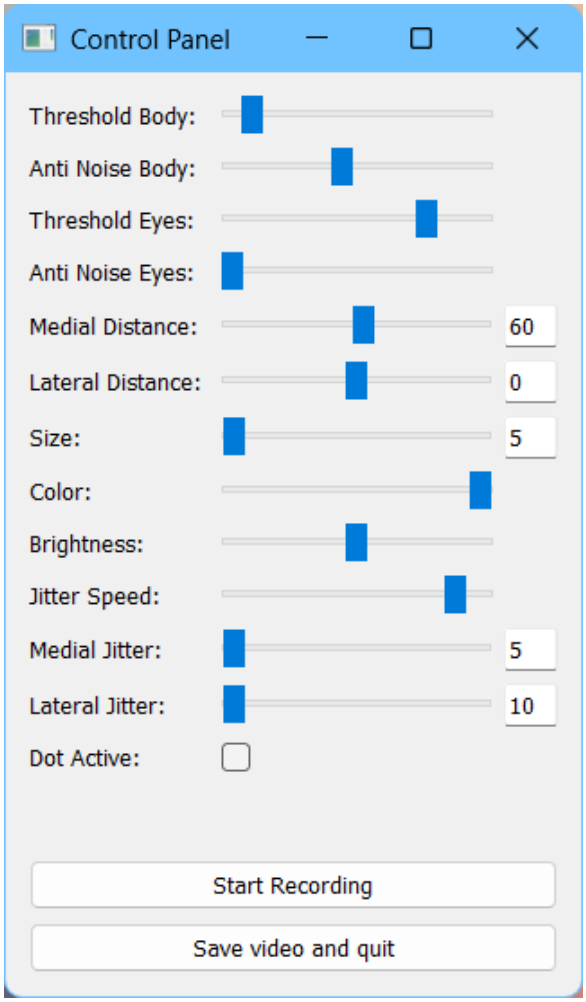
4.2 Usage, Parameters & Views

In this phase, there is, as usual, the camera image on the left and a control panel on the right. This time however, the camera image supports multiple views. One can cycle through them forwards and

backwards by left- and right-clicking the image, respectively. The current view is hereby displayed on the top left.

- View 0: raw: The raw camera image.
- View 1: diff (raw): The result of subtracting the background from the raw camera image. Here, you can see "what the program sees". Everything that is white is detected as a difference from the background and potentially something that should be tracked. Only the animal should appear in a bright white, while the rest should be black or at least dark gray.
- View 2: diff (norm): Same as view 1, but normalized: The whitest visible spots are made whiter, the darkest darker. This view is better in order to recognize differences between background and actual scene.
- View 3: binary (body): The binarized image of the body detection. Every pixel is either white or black, depending on whether the program classifies it as a pixel from the animal's body or not. Here, ONLY the animal's body should be white, other large white spots can result in problems with the tracking. In that case, one can try to tune the threshold or anti-noise parameters, or restart the program, opting for a better background image or an environment with less disturbance.
- View 4: binary (eyes): Same as view 3, but using a different (usually higher) threshold, in order to only detect the bright eyes in the difference image. Here, ONLY the animals eyes should be white. Other white spots will significantly disturb the tracking as it highly depends on the eyes. In problematic cases, one can try to tune the anti-noise and threshold parameters or restart in a better environment or with a better background image.

Independant of the view, the actual tracking process is shown, with the body centroid, eye centroids, eye vector, eye centroids center, dot vector and actual dot being overlayed onto the image.



- Threshold Body: How white pixels must be to be classified as the animal's body.
- Anti Noise Body: How many white pixels must be close together to not be classified as noise during body detection.
- Treshold Eyes: How white pixels must be to be classified as the animal's eyes. Should be significantly higher than the body threshold.
- Anti Noise Eyes: How many white pixels must be close together to not be classified as noise during eye detection. Default value is 1, meaning no reduction. It is not recommended to increase this parameter too much, as it could then easily classify the eyes as noise.
- Medial Distance: The distance from the center of the eyes to the dot, along the medial body axis of the animal (oriented forwards).
- Lateral Distance: The distance from the center of the eyes to the dot, along the lateral body axis of the animal (oriented to the right).
- Size: The radius of the dot.
- Color: The color hue of the dot.
- Brightness: The brightness of the dot.
- Jitter Speed: The speed of the jitter effect. The leftmost value results in no movement at all, the next higher value starts the effect, with one jump every three seconds. The highest value corresponds to a jump at every frame.
- Medial Jitter: The medial magnitude of the jitter effect, i.e. the maximum size of the jump along the medial body axis. Each individual jump is randomly chosen between 0 and this value, and randomly oriented.
- Lateral Jitter: The lateral magnitude of the jitter effect, i.e. maximum size of the jump along the lateral body axis. Each individual jump is randomly chosen between 0 and this value, and randomly oriented.
- Dot Active Checkbox: Checking this checkbox activates the dot. While the dot is always shown in the camera control window, it is only written onto the video output and actually projected onto the scene when it is activated.

- Start Recording Button: Once the parameters are tuned to satisfaction, the recording can be started with this button. While recording, the parameters can still be changed.
- Save video and quit button: In order to stop the recording and save the video, press this button. After saving, the program will tell you where it saved the output to, and then quit.

5 Problems & Limitations

The main problem left with the tool is how its precision and comparability holds up in a serious experiment. This comes down to three main points:

- Some variables only have sliders for changing them, making it difficult to reliably reproduce the experimental setting without a precise value that can be set again.
- The pipeline from the camera recording to the projector and closed feedback loop back to the camera introduces a natural delay in the dot projection. A delay is desireable in general, as it helps accurately simulating prey behaviour. However, the precise delay in milliseconds comes down to hardware properties and is not controllable. Additionally, the dot in the output video won't have the delay, since it is artificially drawn into it. Therefore, the actual dot the animal perceives and the dot as it is shown in the video are not precisely congruent.
- The code works with a 30 FPS video input and writes the output to a video stream with 30 frames per second. The tool therefore has $1/30$ seconds to perform all its computations. if it works faster, the remaining time is filled by `time.sleep()`, to still ensure 30 FPS. If the computations take longer than $1/30$ seconds however, the frame will be written onto the video stream with a delay, resulting in a lagged feedback loop, which can be problematic if you want to use the output video for later analysis. There is no code yet that warns about this or deals with this problem properly. Therefore it is recommended to choose parameters which don't end up in expensive computations.