

Tracking Software

Luis Nicolás Luarte Rodríguez

Introduction

- Animal behavior quantification is at the base of multiple scientific questions
- Manual annotation is slow and cumbersome
- Drawing from open source codebases, simple and tailored solution can be achieved
- Computer vision algorithms can segment, identify and categorize in a frame by frame basis
- Allows for richer analysis

What were my objectives?

- ① Control parallel (4 or more) mice recordings
- ② Obtain where are the 'points of interest' at every frame
- ③ Synchronize 'points of interest' acquisition with other instruments
- ④ Do further analysis in 'points of interest'

Tools and resources that I used

- ① Using open source image processing libraries (python)
 - For managing the 'data' side
- ② GNU/UNIX libre tools
 - For managing the 'parallelization' part
- ③ Previously published segmentation and detection algorithms
 - To create a lightweight and efficient implementation

What were my 'restrictions'?

- ① Live data processing
- ② Must run in 'low end' software (raspberry pi 3)
- ③ Modular enough to add further data processing
- ④ Open source / libre / free / etc...

Why am I doing this?

- Intellectual challenge
 - There are many open source software that achieve similar goals
 - Coding skills acquired:
 - Reading documentation
 - Managing and reading code (using git)
 - Bug fixing and general problem solving
 - Opens the possibility of 'extending' functionality
- Thesis work
 - Decision making
 - Regret (Steiner and Redish 2014)
 - Behavioral pattern recognition

Lab objectives

- Behavioral experiments with mice/rats
 - Multiple recordings
 - Paid software handled multiple recordings, but was slow
- Develop our own equipment
 - Open source
 - 3D printing

Open source alternatives: a comparison

There are many options, however features are typically defined by the purpose

	Ethoflow	DeepLabCut	Tracktor	Current Project	
<i>Interface Type</i>	GUI	GUI	CMD-LINE	CMD-LINE	General purpose
<i>Hardware requirements</i>	High/Mid	High	Low	Low	Specific purpose
<i>Code license</i>	Open source	Open source	Open source	Open source	
<i>Background estimation</i>	Instance Segmentation	ResNets	Contour + KNN	Background Subtraction	
<i>Object detection</i>	Instance Segmentation	ResNets	Contour + KNN	Contours + Geodesic	
<i>Parallelization</i>	GPU Only	GPU Only	Not implemented	Command broadcasting	
<i>Codebase size</i>	Big	Big	Minimal	Minimal	
<i>Estimated Speed on Pi 3</i>	1 FPS	1 FPS	30-90 FPS	30 FPS	

Conclusion

- General view of features
- Too many features = slow in ‘low end’ machine
- Closed source (like Any-maze) are more featured
 - 1 processing unit per license; Can manage multiple experiments at the same time
 - Not extensible, at least, in a ‘live’ setup
- This project draws form many open source code bases
 - Contour estimation: (Sridhar, Roche, and Gingins 2019)
 - Detection algorithm: (Ben-Shaul 2017; Patel et al. 2014)
 - Segmentation: (Bradski 2000)

Results

As my project was mainly software design, results are divided in:

- Implementation
 - How I managed to achieve the objectives
- Test results
 - Preliminary results

Control parallel (4 or more) mice recordings

The project consists on two operational units

- Main computer
 - Sends instructions to Raspberrys Pi
 - Previews
 - Allows for calibrations
- Raspberrys Pi 3 (x4~)
 - Perform all the computation
 - Synchronizes with arduino 'lickometer'
 - Records behavioral test

Control parallel (4 or more) mice recordings

- Codebase is managed with Git and hosted in Github
- Code is uploaded to Pi with bash scripts
- Secure Shell network protocol relays all instruction from main computer to Pi

Must run in 'low end' software (raspberry pi 3)

- Pi cam NoIR V2 outputs 30-90 FPS
 - Whole image processing allows for stable 30 FPS

YEAR	MONTH	DAY	HOUR	MINUTE	SECOND	MICROSECOND	body_x	body_y	tail_x	tail_y	head_x	head_y
2020	11	20	22	42	27	12104	131	356	145	370	121	349
2020	11	20	22	42	27	41985	131	356	143	371	124	347
2020	11	20	22	42	27	72414	131	356	141	372	123	347
2020	11	20	22	42	27	112025	131	356	141	372	123	347
2020	11	20	22	42	27	144579	131	356	143	372	122	347
2020	11	20	22	42	27	167236	130	355	139	375	126	345
2020	11	20	22	42	27	197258	130	355	139	373	124	346
2020	11	20	22	42	27	236510	131	355	139	373	125	345
2020	11	20	22	42	27	266427	131	355	140	374	124	346
2020	11	20	22	42	27	299887	131	355	140	374	124	346
2020	11	20	22	42	27	340350	131	356	138	374	127	345
2020	11	20	22	42	27	370676	133	357	137	373	126	345
2020	11	20	22	42	27	401232	132	355	137	374	127	344
2020	11	20	22	42	27	431698	131	355	137	374	125	345
2020	11	20	22	42	27	472449	132	355	130	373	141	345
2020	11	20	22	42	27	502839	132	355	130	373	141	345
2020	11	20	22	42	27	533061	133	355	135	373	125	344
2020	11	20	22	42	27	563785	133	354	141	372	125	345
2020	11	20	22	42	27	604331	133	355	141	371	125	346
2020	11	20	22	42	27	635378	133	354	140	371	126	345
2020	11	20	22	42	27	665825	132	354	134	373	137	343
2020	11	20	22	42	27	711803	132	354	134	373	126	344
2020	11	20	22	42	27	742823	133	356	129	371	140	344
2020	11	20	22	42	27	773344	132	352	135	371	132	341
2020	11	20	22	42	27	803581	132	352	136	371	132	341
2020	11	20	22	42	27	844149	132	353	139	370	126	343
2020	11	20	22	42	27	875256	132	354	138	371	126	343
2020	11	20	22	42	27	905809	132	354	138	371	126	343
2020	11	20	22	42	27	946646	129	355	142	370	153	351
2020	11	20	22	42	27	977234	130	355	142	370	153	351

Figure 2: 30 Frames under 1 second

Obtain where are the ‘points of interest’ at every frame

- At each time step:
 - Separate background from foreground
 - Get the head point
 - Get the “body” point
 - Get the tail point
 - Do further analysis with those points

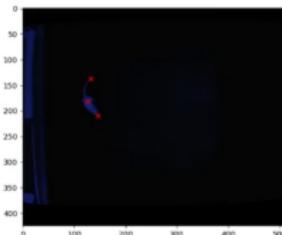
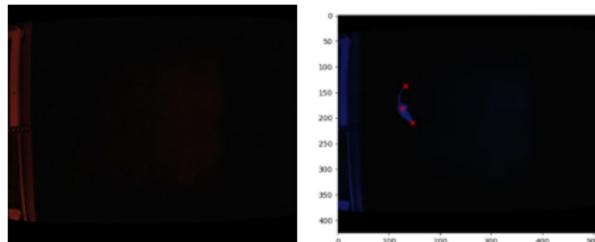


Figure 3: Finished frame

Ideal input

- First step is to have a good idea of what our background is
 - We can create a model (more, complex but better suited for environments with dynamic lighting)
 - Or we can assume that our background is never going to change (this is how I did it)
 - Main computer stores a 'background' image in every pi



However a simple image is rather complex, because each pixel has 3 channels (red, green, blue)



Figure 4: RGB channels

To simplify we turn it to gray scale (only 1 channel)

Simple averaging, allows to save time: only one input per pixel, instead of three



Figure 5: Grayscale = $(r + g + b / 3)$

However, we need to further process our background to make it 'smoother'

- We need to remove noise
 - Noise are 'details' that make an image harder to identify
 - In other words, denoising is equivalent to make an image more homogeneous, while preserving edges
 - A bilateral filter does such thing

Bilateral Filter: The kernel

- A kernel or ‘structuring element,’ in image processing, is a matrix of numbers, typically generated by some sort of function
 - It has an origin
 - Also has a ‘shape’

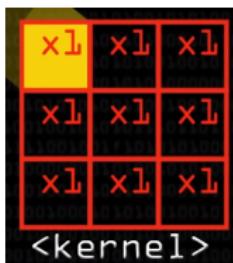


Figure 6: Example Kernel

Bilateral Filter: Convolution

In image processing a convolution is a process, where every pixel is added to local pixels, weighted by a certain kernel.

- Adding into '64,' purple frame

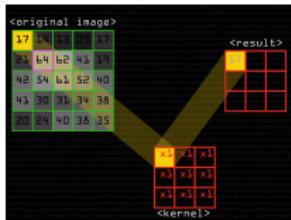


Figure 7: Convolution

Bilateral Filter: The two kernels

- Both kernels are Gaussian (typically):

$$G(\sigma, x) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2}{2\sigma^2}\right)$$

$$BF[I]_p = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_s}(|p - q|) G_{\sigma_r}(|I_p - I_q|) I_q$$

↓ ↓ ↓

Normalization Factor Space Weight Range Weight

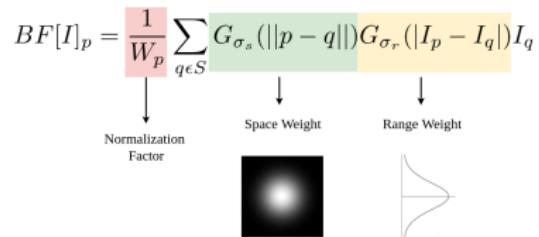


Figure 8: The two kernels

Bilateral filter: One step convolution

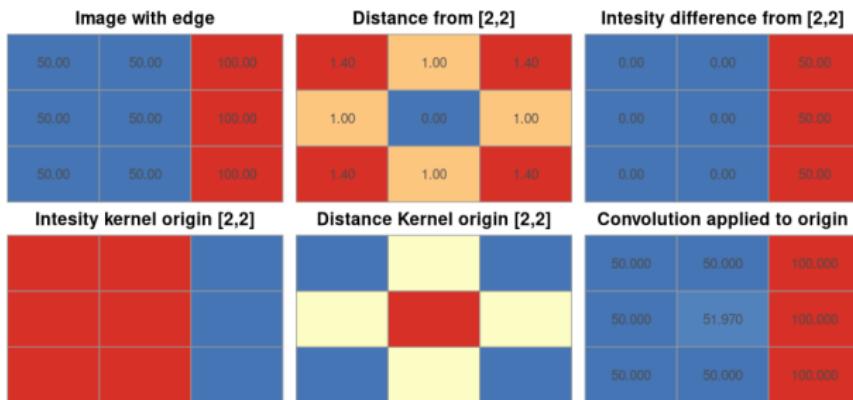


Figure 9: One step convolution

Applied to an animal

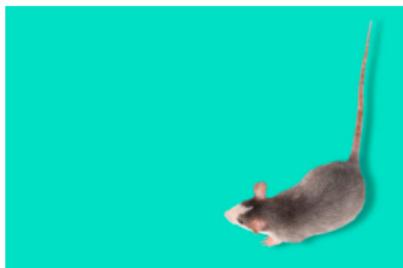


Figure 10: Original image

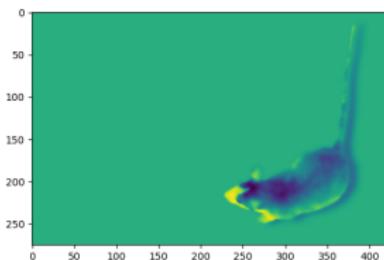


Figure 11: Smoothed image

The next step is to subtract the background from the image with the animal

- This is done by subtracting pixel intensity
- The result is not good, some areas of the animal are considered background (black color)

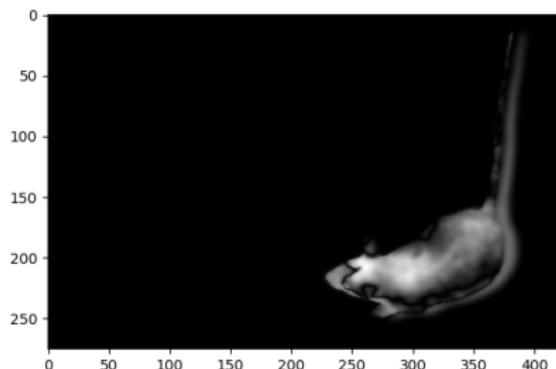


Figure 12: Difference

We use morphological transformations to fix this

- Morphological transformations are operations applied to binary images, which are based on the image shape
 - They also use a ‘kernel’ and the process is quite similar to the bilateral filter
 - Here the kernel is binary, its defined by interest and no interest regions (0 and 1, respectively)
- Our main problem is that there’s background objects INSIDE the animal (black pixels inside the white shape)
 - The closing operation allow us to fix this problem
 - The closing operation is dilation followed by erosion

Dilation

- ① We pick a kernel
- ② We apply the kernel with stride 1
- ③ If any pixel inside the structuring element is '1,' then the origin := 1, else := 0

Dilation

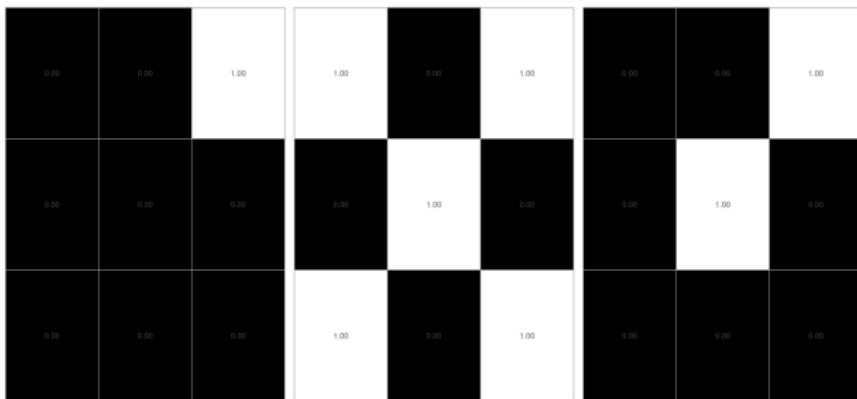


Figure 13: Upper right corner is '1,' so it dilates the image

Dilation



Erosion

- ① We pick a kernel
- ② We apply the kernel with stride 1
- ③ If **all** pixels inside the structuring element are '1,' then the origin := 1, else := 0

Erosion

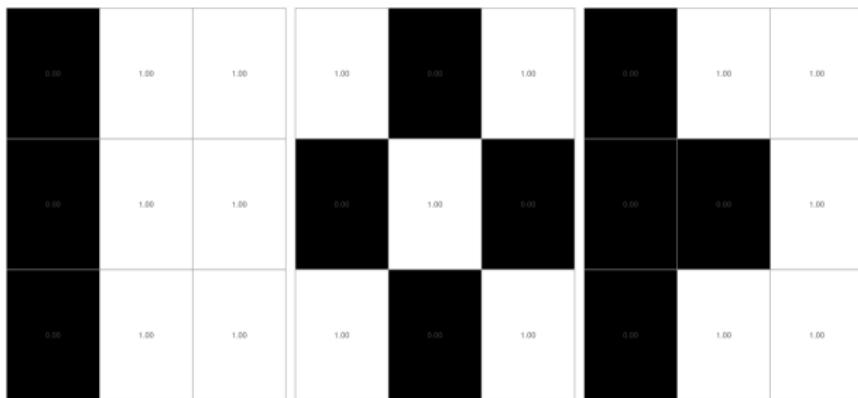


Figure 14: The kernel doesn't detect '1' in the left corners, so it erodes the image

Erosion



Closing = dilation followed by erosion



Figure 15: Closing

Result

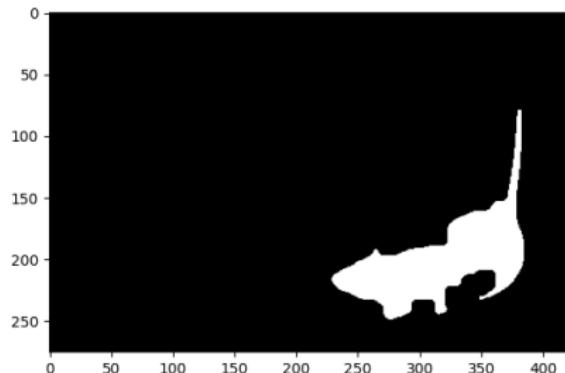


Figure 16: Not perfect, but the animal is clearly isolated

The original image is not ideal, but we can improve this and make it a more robust algorithm

- We can calculate the contours of the image
 - This makes it more robust against 'lighting artifacts'



Figure 17: Image with artifact

Artifact removal

- The algorithm simply calculates the contours of each objects, and selects the one with the bigger area
 - The image looks worse because the artifact is super big
 - In normal conditions an artifact like that can be removed manually

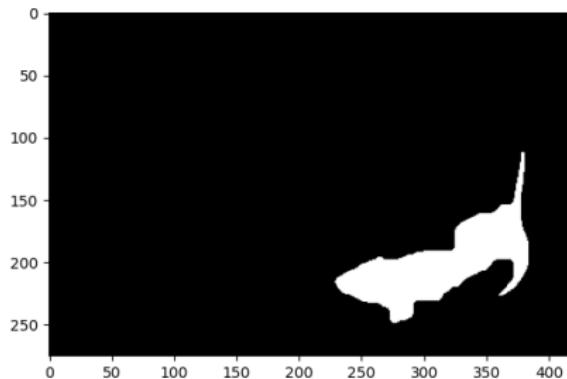
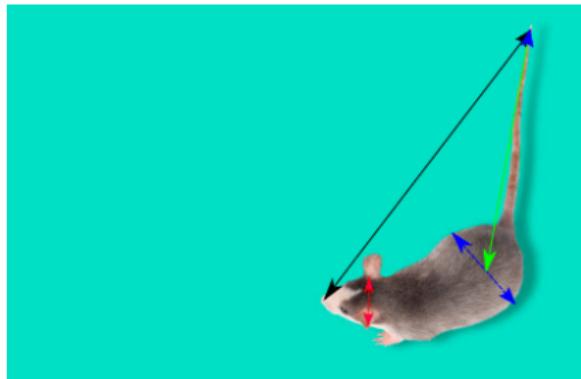


Figure 18: Image without artifact

After the segmentation problem, we need to find the head, body and tail

- The intuition is:
 - Rats/mice are 'chubby'
 - Long tails
 - 'small' head relative to the body
 - The tail is the furthest away point from the body
 - The head is the furthest away point from the tail



The geodesic distance is the proper implementation to solve this

- Approximated by the fast marching algorithm solves the distance problem within a bounded region
 - How long from a given point, a particle traveling at constant speed, will take to reach the boundaries
 - The centroid point is the slowest one
 - We can calculate this speed for every arbitrary 2 points

Considering this, the furthest away point is the 'body' because the animal is 'chubby'

- If fast marching algorithm is performed with every pixel as a seed

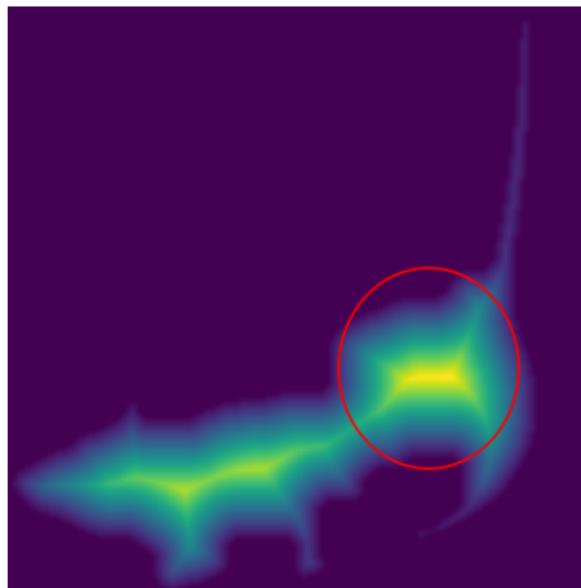


Figure 19: Warmer color is more distant

Geodesic vs euclidean distance

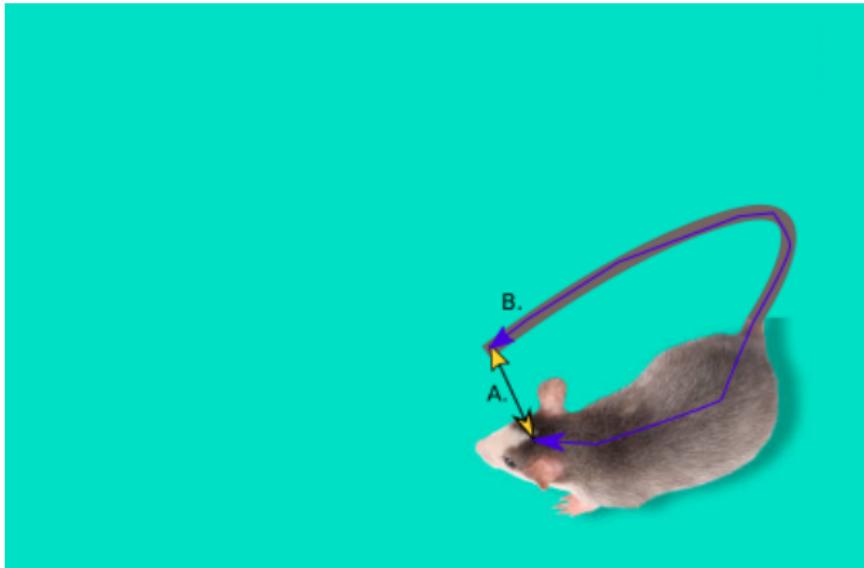


Figure 20: (A) Minimal euclidean distance, (B) minimal distance within boundaries

Detecting the head and the tail is relatively simple

- We just need 2 extra calculations
 - Tail tip := longest distance from the 'hip' or 'body'
 - Head := longest distance from tail tip

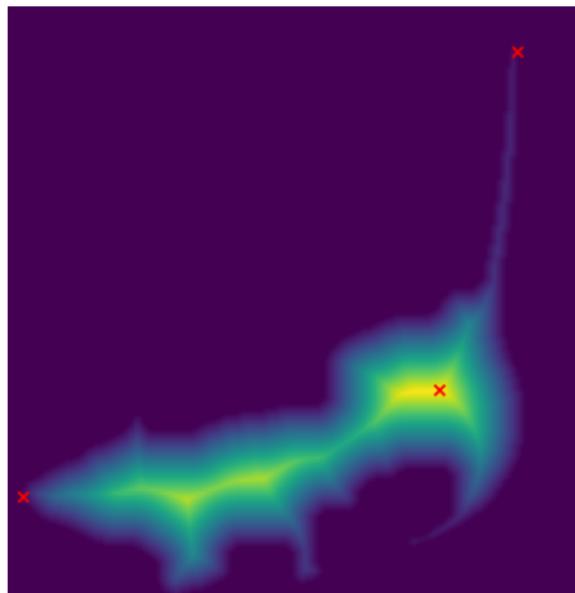


Figure 21: In read the calculated points

Demo

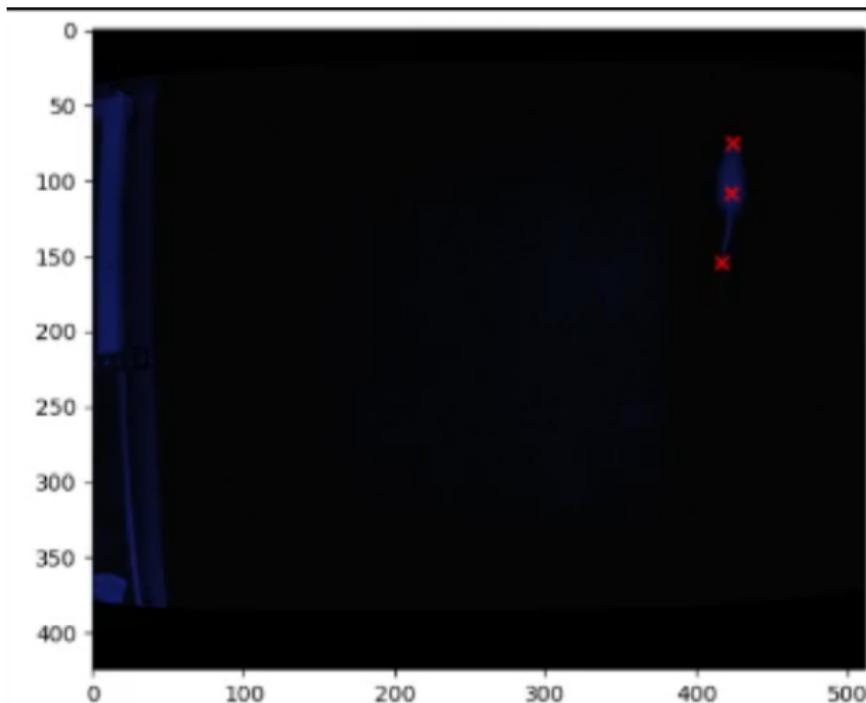


Figure 22: Graphical illustration

Synchronize 'points of interest' acquisition with other instruments

- Other instruments 'timestamp' every data points with microsecond precision
- Pi also does this
- Data is synchronized 'offline' finding nearest time point

Do further analysis in ‘points of interest’

```
ggplot(data, aes(x=body_x, y=body_y)) + geom_path()
```

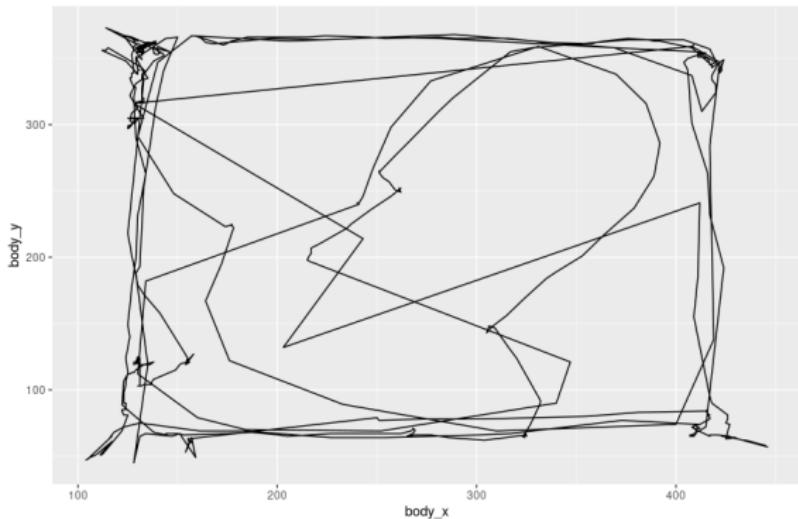


Figure 23: Animal path

How to implement the system

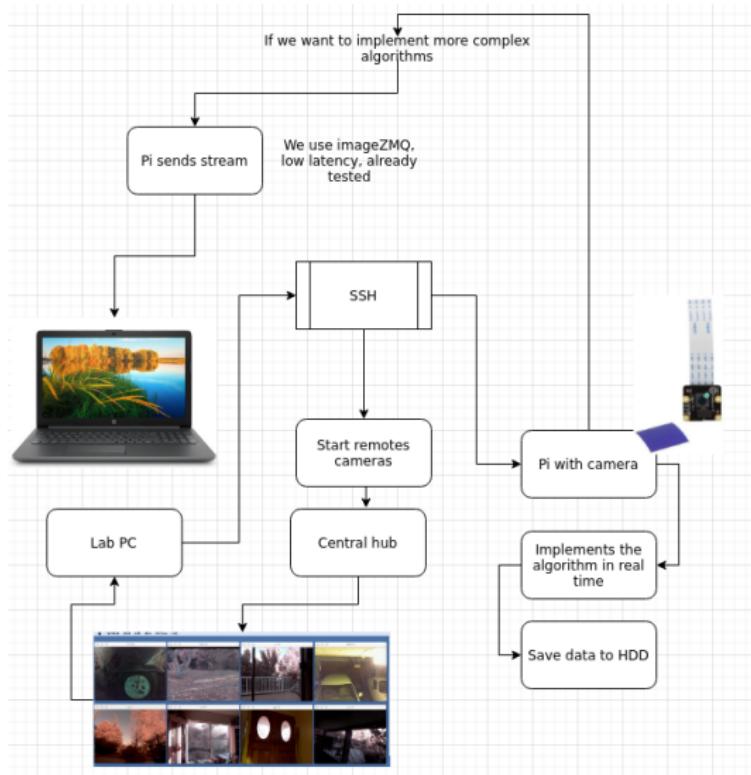


Figure 24: The big picture

Conclusions

- Creating a custom software for tracking is not as time consuming:
 - Many implementation are open source, so code can be re used
 - Python libraries are flexible enough, there's no need to extend them
- Modular design and GNU tools allows to easily extend software functionality
- Image processing can be done live
- Further data analysis can be done in programming language of choice
- If more powerful analysis is required to be done live, computing can be performed by more powerful computers or increasing the number of Pi

Codebase

https://github.com/nicolasluarte/uni/tree/master/PHD/tracking_device

References I

- Ben-Shaul, Yoram. 2017. "OptiMouse: A Comprehensive Open Source Program for Reliable Detection and Analysis of Mouse Body and Nose Positions." *BMC Biology* 15 (1): 41. <https://doi.org/10.1186/s12915-017-0377-3>.
- Bradski, G. 2000. "The OpenCV Library." *Dr. Dobb's Journal of Software Tools*.
- Patel, Tapan P., David M. Gullotti, Pepe Hernandez, W. Timothy O'Brien, Bruce P. Capehart, Barclay Morrison, Cameron Bass, James E. Eberwine, Ted Abel, and David F. Meaney. 2014. "An Open-Source Toolbox for Automated Phenotyping of Mice in Behavioral Tasks." *Frontiers in Behavioral Neuroscience* 8 (October). <https://doi.org/10.3389/fnbeh.2014.00349>.

References II

- Sridhar, Vivek Hari, Dominique G. Roche, and Simon Gingins. 2019. "Tracktor: Image-based Automated Tracking of Animal Movement and Behaviour." Edited by Luca Börger. *Methods in Ecology and Evolution* 10 (6): 815–20.
<https://doi.org/10.1111/2041-210X.13166>.
- Steiner, Adam P, and A David Redish. 2014. "Behavioral and Neurophysiological Correlates of Regret in Rat Decision-Making on a Neuroeconomic Task." *Nature Neuroscience* 17 (7): 995–1002. <https://doi.org/10.1038/nn.3740>.