Tracking Software

Luis Nicolás Luarte Rodríguez

Introduction

- Animal behavior quantification is at the base of multiple scientific questions
- Manual annotation is slow a 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 do I want to achieve?

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’

How do I plan to do it?

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 are 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
- It offers some slight advantages as is tailored to the lab objectives

Comparison: General purpose tracking

- Ethoflow (Bernardes et al. 2020)
  - Designed for heterogenous backgrounds
  - GUI based
    - More user friendly
    - Not that compatible with GNU tools
    - Less extensible (harder to extend)
  - Instance segementation
    - Way better results, segments every object
    - More computationally intensive
    - Live segementation is around $>$1 fps in pi 3
  - Better object estimation
    - Detection is general, not intended for mice/rat
    - Background estimation is a every frame

Comparison: Specific purpose tracking

- DeepLabCut (Mathis et al. 2018)
  - Extremely powerful
    - Requires tensor flow
    - GUI-based
    - Requires neural network training
- Tracktor (Sridhar, Roche, and Gingins 2019)
  - Uses a similar idea
  - Lightweight and fast
  - Extensible due to minimalist codebase
  - Background estimation
    - Unnecessary in our case
  - Not designed for parallel or remote control

Conclusion

- General view of features
- Too many features = slow in ‘low end’ machine
- Closed source (like Any-maze) are more featured
  - No parallel design
  - 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 relies 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 1: 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

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 out background is never going to change (this is how I did it)
    - Main computer stores a ‘background’ image in every pi

We can get an image of our background

Figure 2: Example background

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

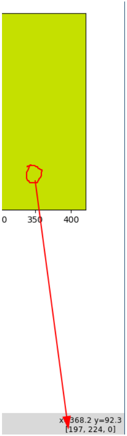


Figure 3: RGB channels

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

Figure 4: 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

Noise image

Figure 5: Notice the details

Noise image after bilateral filter

Figure 6: Details are gone, edges preserved

Similar steps are applied to the image with the animal

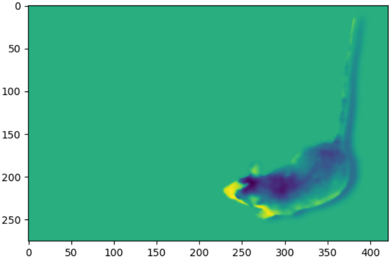


Figure 7: Smoothed image

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

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

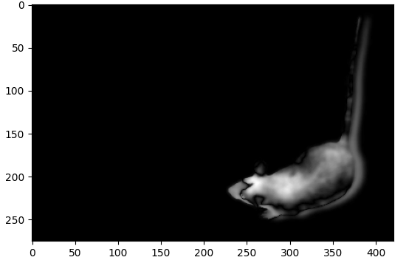
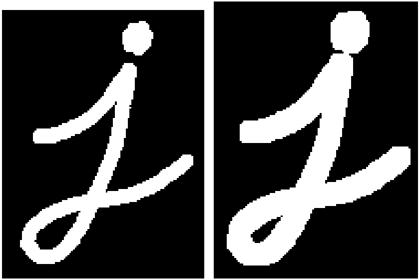


Figure 8: Difference

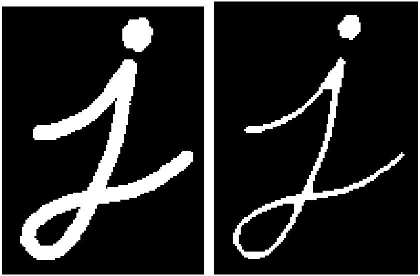
We use morphological transformations to fix this

- Morphological transformations are operations applied to binary images, which are based on the image shape
  - They use a 'kernel,' which a windows where a certain operation is performed
- Our main problem is that there's background objects INSIDE the animal
  - The closing operation is applied to the kernel
    - A pixel element is defined as '1' if: inside the kernel there's atleast a '1' pixel (dilation)
    - Then we 'erode' the boundaries: a pixel is considered '1' is all pixels under the kernel are 1's, otherwise is 0

Dilation



Erosion



Closing = dilation followed by erosion

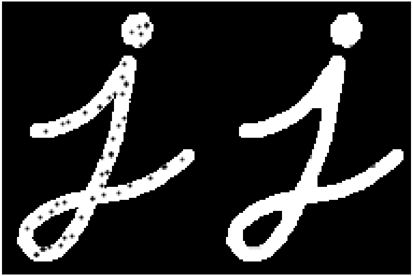


Figure 9: Closing

Result

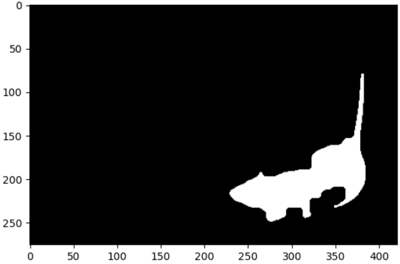


Figure 10: Not perfect, but the animal is clearly isolated

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 11: 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

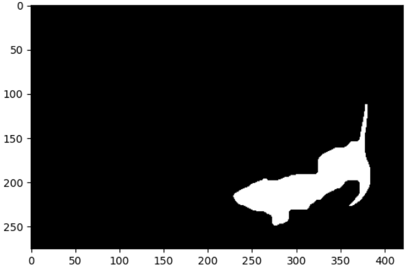


Figure 12: Image without artifact

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

- The geodesic distance is a shortest path between 2 points in a certain space
  - We define our space as the animal surrounded by boundaries (background)
  - The distance, considering the boundaries, is the geodesic distance
  - Is approximated by the fast marching algorithm

Geodesic vs euclidean distance

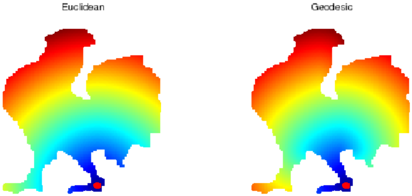


Figure 13: Notice how the boundaries inform the distance at the feet

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

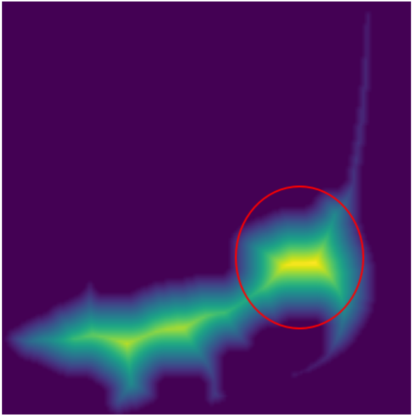


Figure 14: Warmer color are more distant

Detecting the head and the tail is relatively simple

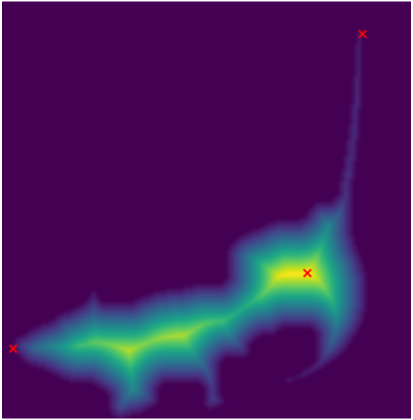


Figure 15: In read the calculated points

Demo

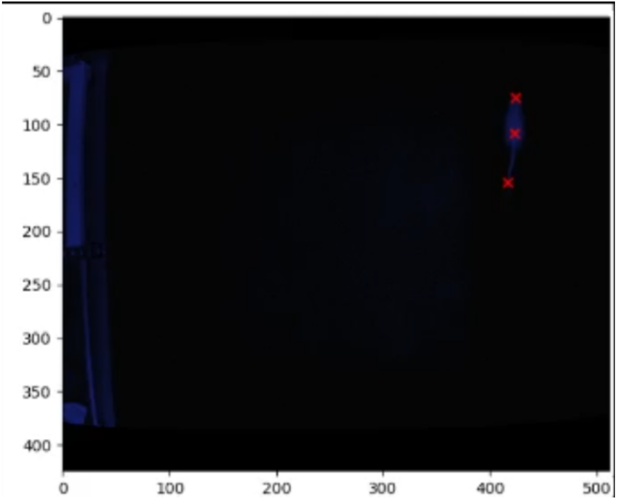


Figure 16: Graphical illustration

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()
```

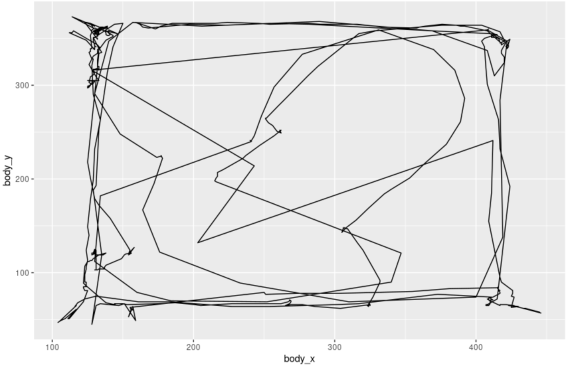


Figure 17: Animal path

How to implement the system

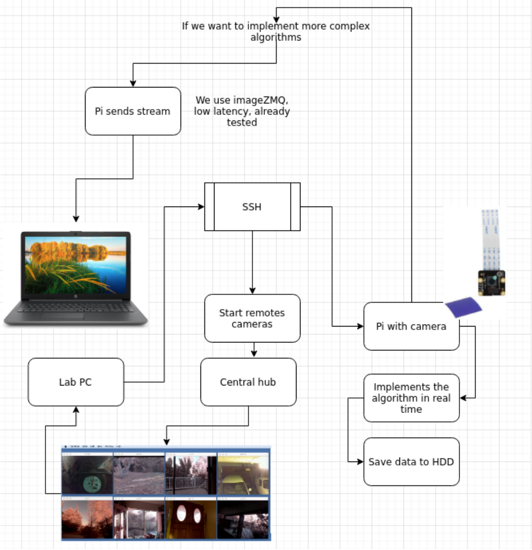


Figure 18: The big picture

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.

Bernardes, Rodrigo Cupertino, Maria Augusta Pereira Lima, Raul Narciso C. Guedes, and Gustavo Ferreira Martins. 2020. "Ethoflow: Computer Vision and Artificial Intelligence-Based Software for Automatic Behavior Analysis." Animal Behavior and Cognition. http://biorxiv.org/lookup/doi/10.1101/2020.07.23.218255.

Bradski, G. 2000. "The OpenCV Library." *Dr. Dobb's Journal of Software Tools*.

References II

Mathis, Alexander, Pranav Mamidanna, Kevin M. Cury, Taiga Abe, Venkatesh N. Murthy, Mackenzie Weygandt Mathis, and Matthias Bethge. 2018. "DeepLabCut: Markerless Pose Estimation of User-Defined Body Parts with Deep Learning." *Nature Neuroscience* 21 (9): 1281–89. https://doi.org/10.1038/s41593-018-0209-y.

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 III

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.