TankTrack: A multi-scale tracking system for biological targets in the laboratory

Aaron Ray
Under the supervision of Kakani Katija

August 2019

1 Introduction

There are a wide range of propulsion and feeding mechanisms in ocean invertebrates that are not well understood. We are interested in learning more about what life is like for these animals. The propulsion and feeding dynamics can be very difficult to study, as the animals are often small and the features of interest even smaller. They can also accelerate very rapidly relative to their size. For example, studying the swimming dynamics of squid paralarvae requires focusing a microscope on a larvae several millimeters in length, and then keeping it in the frame as it accelerates at several mm/s^2. These characteristics complicate capturing high-quality long-term observational data of the animals, as they will dart out of the camera field of view. In order to capture the small details of the animals, the animal must appear large in the camera image. As a result, the animal's motion is very fast in terms of pixels per second, so the system must be able to adjust very rapidly. This is the same difficulty that arises when using a pair of binoculars to track a nimble sparrow compared to a large whale.

In some cases, we can make the problem easier by restricting the motion of the target. A dab of glue, for example, may trap the animal in place and greatly simplify the task of keeping it in the camera frame. While for some studies this solution may be sufficient, it prevents us from studying the natural, long-term swimming behaviors of the animal. In order to study multi-scale behaviors–how the swimming behavior, caused my small features on the animal, relates to large-scale motion–we must allow the animals to swim freely.

The fundamental problem is that with a single camera, increasing animal resolution commensurately increases the speed at which our system must be able to respond to animal motion. In order to enable multi-scale tracking of animals, we propose a system of two cameras mounted on motorized stages. The first camera is a "spotter" camera, similar to the spotter scope used to align telescopes. The second camera is the science camera, in this case outfitted with a large macro lens to observe small-scale features on the animal of interest. The moving stages allow a computer vision algorithm automatically move the cameras to keep the animal of interest in view. This system combines the advantages of easier tracking in the spotter camera's field of view with high resolution in the science camera. Even if the animal temporarily swims out of the science camera frame, the system can catch up by maintaining the track in the spotter frame.

2 Related Work

Our system is similar to a recent plankton tracking system presented in [1]. In contrast to their purpose-built system, we hope that the system architecture used for our tracker will be easily extensible to a variety of camera, stage, and tank combinations.

3 Project Requirements and Goals

We hope that our multiscale tracking software platform will be extensible to a wide variety of use cases where it is important to relate small-scale behavior on short time scales with large scale behavior on longer time scales. The system must be able to record video from both the high-resolution science camera and the lower-resolution spotter camera. It must also be able to reconstruct the animal's position over time, meaning that we must be able to estimate the animal's depth.

We consider several constraints on system design to facilitate a wide variety of potential applications. The tracking platform cannot depend on expensive, purpose-built equipment. This eliminates tools like telecentric illuminators and lenses from consideration. The system also cannot use tracking techniques that rely on large amounts of data. Collecting and annotating data, then training a neural network based tracker such as DeepLabCut ([2]) is too costly and time consuming in many cases, such as on a ship during a research cruise. The setup time would radically dwarf the amount of time that we actually have access to the animal. The system must also work with any cameras, lenses, and stages that are appropriate for observing the target of interest.

However, to guide development, our initial goal involves tracking Ctenophores in a tank. Specifically, we aim to understand how ctene row beating frequency corresponds to animal motion and behavior. To capture this behavior we need a macro lens with a field of view on the order of 1 cm, capable of recording frames at 60 fps. The motion stages must be capable of moving the camera's field of view around a 10x10x10cm tank.

An E-con Systems liquid lens was used as the spotter camera. The liquid lens allows us to estimate object depth from a single camera. A Point Grey Grasshopper camera with a large macro lens was used for the science camera. Three Thorlabs LTS300 stages provided the actuation for moving these cameras around the tank. With this hardware in mind, we can design the software architecture.

4 System Development

We begin the development by considering how a scientist will interact with the system. A graphical interface will present the user with the tools needed to refine and tune the tracking setup. Figure 1 shows an example of this interface in development. A toolbar offers many options for tuning and control, live feeds from the spotter and science cameras allow the user to monitor the performance of the system. The user initializes the tracking by clicking on a target in the spotter camera. From this point forward, the stages move the cameras to keep the animal in the field of view. Both video feeds and a log of animal position estimate are saved automatically.

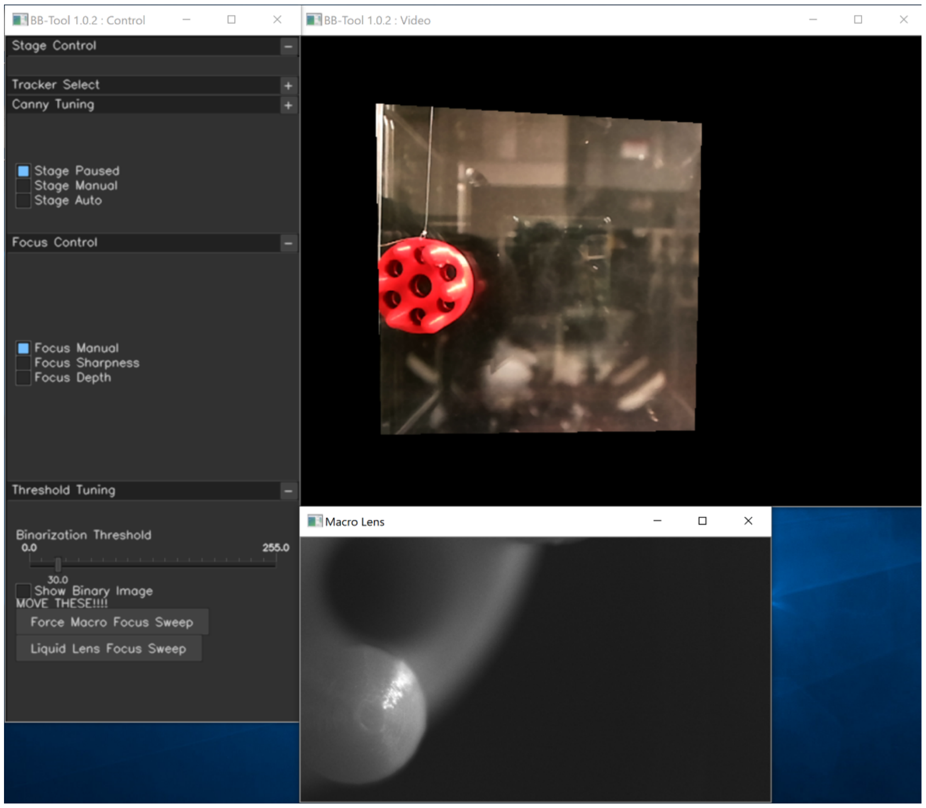

Figure 1: User interface to the tracking system. The left side of the interface provides a variety of toolbar options for tuning and interacting with the system. On the top right, the spotter camera's field of view is shown. On the bottom right the science camera view is shown. These are displayed to the user for reference and also recorded.

4.1 Animal Tracking

There are basically only two hard things about the multiscale tracking problem. The first is tracking the animal in the image. The second is reconstructing where the animal is based on these observations[1].

We begin by investigating how to track the animal in the image. In the interest of generalizablility, we prefer methods with fewer tuning parameters that require less effort to transfer to new targets.

The first tracking technique we tried identified edges in the image with a canny edge detector, then found closed contours in the edges with a routine built in to OpenCV. The center of the closed contour closest to the estimated animal position at the previous time step is considered to be the current animal position. However, as seen in Figure 2, the edges Ctenophores are often not very robust. As a result, this tracking method is not sufficiently reliable and stable.

The second tracking technique considered was a simple thresholding operation as demonstrated Figure 2. The animal position is estimated as the center of mass of above-threshold pixels with some distance of the estimated position in the previous timestep. This method is robust to poorly defined edges and makes effectively no assumptions about the object being tracked. The biggest downside to this method is that it requires sufficient contrast between the target and background. If the spotter camera were stationary, then background subtraction could easily be applied to make the thresholding much more robust. However, our initial setup assumed a moving spotter camera. This means we cannot easily use background subtraction. As the edges of the tank appear much brighter than the animal it is impossible to detect the animal in these areas.

To prevent the bright sides of the tank from being mistaken for the target, a masking operation can be applied to the sides of the tank. We made a tool that allows the user to calibrate the position of the corners of the tank in order to automatically mask out the sides of the tank(see Figure 3).

4.2 Depth From Focus

In order to reconstruct the animal's position over time, we must estimate its depth. We hoped to accomplish this with a single camera by using depth-from-focus. The spotter camera had a built-in liquid lens that allows software control of focus. When the target of interest is in focus, it should achieve its maximum sharpness. Sharpness can be calculated by applying a high-pass filter to the image (such as Laplacian kernel). We determine the sharpest liquid lens setting by sweeping through all of them and calculating sharpness from each. We can use an empirical calibration curve for the liquid lens camera (Figure 4) to map from liquid lens setting to distance to focal plane (and therefore to animal).

[1]Of course, these two concepts are not entirely distinct, and we can use each to inform the other. However in this project we treat them as more or less distinct. If there were a more specific dynamics/behavior model of the targets being tracked, it would make more sense two couple these two stages more strongly

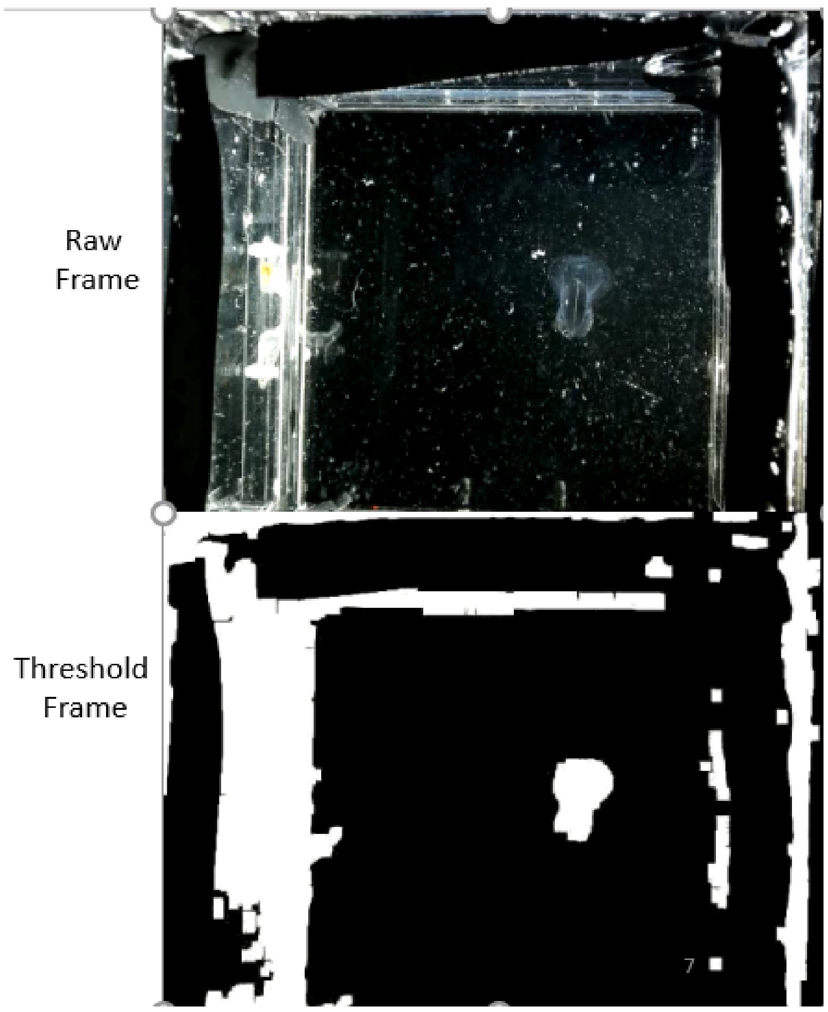

Figure 2: Comparison between raw frame from the spotter camera and the frame after thresholding and morphological operations (dilate/erode) have been applied.

Figure 5 shows examples of sharpness curves of a test object and real animal. Unfortunately it also shows the difficult of estimating depth from sharpness. Practically, it is difficult to estimate depth from sharpness because the animal may be spread out over a range of depths (in a discontinuous way) and accurate measurements require an image at every (or at least most) liquid lens setting that corresponds to a valid focal plane that is within the tank. Sweeping through all of these settings limits the frequency with which the animal state can be updated and the amount of time the animal is in focus in the spotter camera. The delay in liquid lens setting must also be accounted for, which adds another

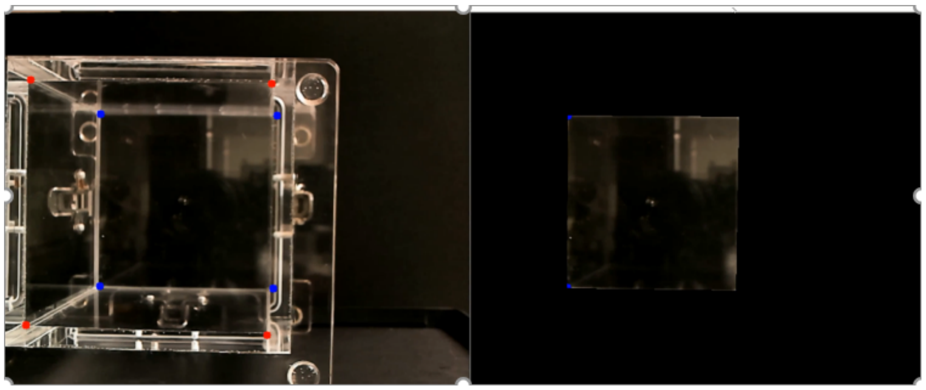

Figure 3: Utility for calibrating the 3D location of tank corners from pairs of images and using the estimated corner locations to mask the sides of the tank in the image as the stage moves.

layer of difficulty (see Figure 6).

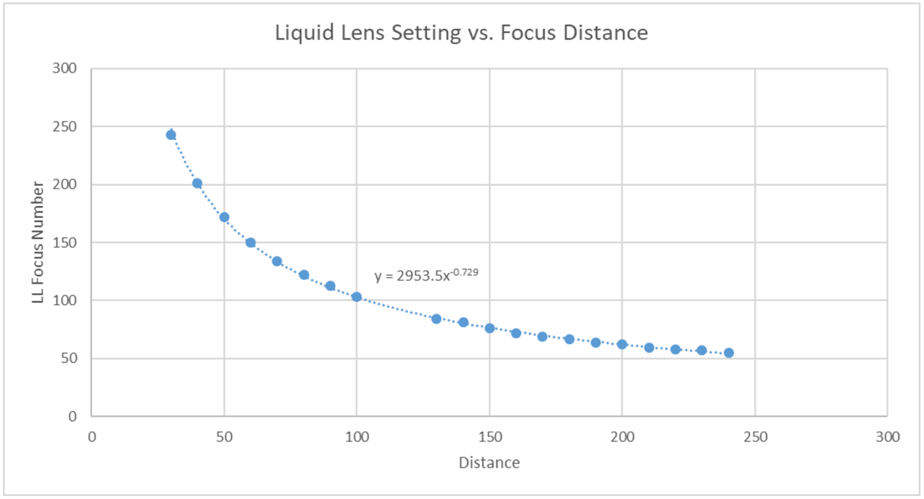

Figure 4: Calibration curve that allows us to map from liquid lens setting to distance to focal plane.

4.3 Software Architecture

A primary goal for our tracking system is extensibility and ease of changing individual components. To facilitate this flexibility, the software architecture is structured in a very modular way. Each piece of hardware interfaces with a single piece of code running in its own process. We can think of each software

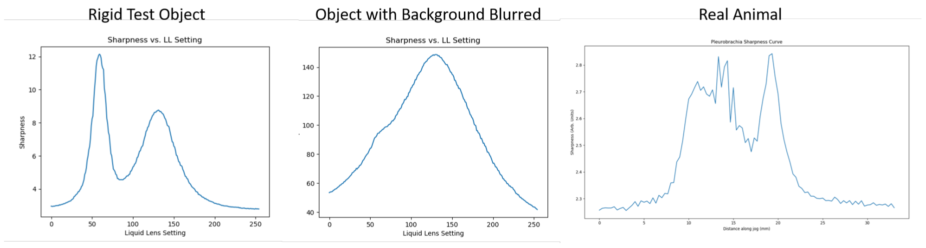

Figure 5: Example sharpness vs. liquid lens setting curves for a test object and a real animal. The left panel shows that there may be multiple sharpness peaks corresponding to different objects in the scene. In this case, the left peak was caused by very high frequency background patter. In the middle panel, the same data from the left panel was passed through a low pass filter before the high-pass Laplacian to filter out the background pattern. On the right, we see the sharpness curve for a real animal. It demonstrates that with a narrow focal plane, it is very difficult to decide solely from sharpness data where to focus, as there are two peaks that both correspond to the animal (in this case, the main body and the tentacles were at different depths)

process in the system as a node in an "interaction graph". Changing hardware only requires writing a small amount of new interface code for that specific hardware (a new "node"). While this level of abstraction could be obtained with well-defined interfaces between pieces of code within a single monolithic process, explicitly separating the pieces has several advantages.

First, the composability of the system is improved. Different pieces of software can be run independently. For example, the node that saves video can be started, stopped, and restarted independently of the rest of the system. The camera node and video saving node can be run without the rest of the system to save calibration data without the rest of the system's hardware hooked up. This type of functionality *could* be achieved by adding more fine-grained control to a user interface of a single monolithic application, but it would require extra effort compared to getting the ability "for free" in this system.

Second, separate processes are actually required to get proper parallelism in Python. Python's Global Interpreter Lock (GIL) prevents different Python threads from running at the same time. Thus splitting the program into processes directly increases performance (which is incredibly important when reading cameras a high rate and saving video).

Finally, the communication interface between processes occurs across TCP sockets. As a result, the system can trivially be spread across many computers. If hard drive bandwidth limits video saving rate, camera images can be broadcast to several computers that each save a different camera's video. Similarly, a long-term tracking experiment could be monitored remotely by running the user interface from a second computer. Composing the system with these discrete pieces makes these different combinations trivial, instead of having to explicitly

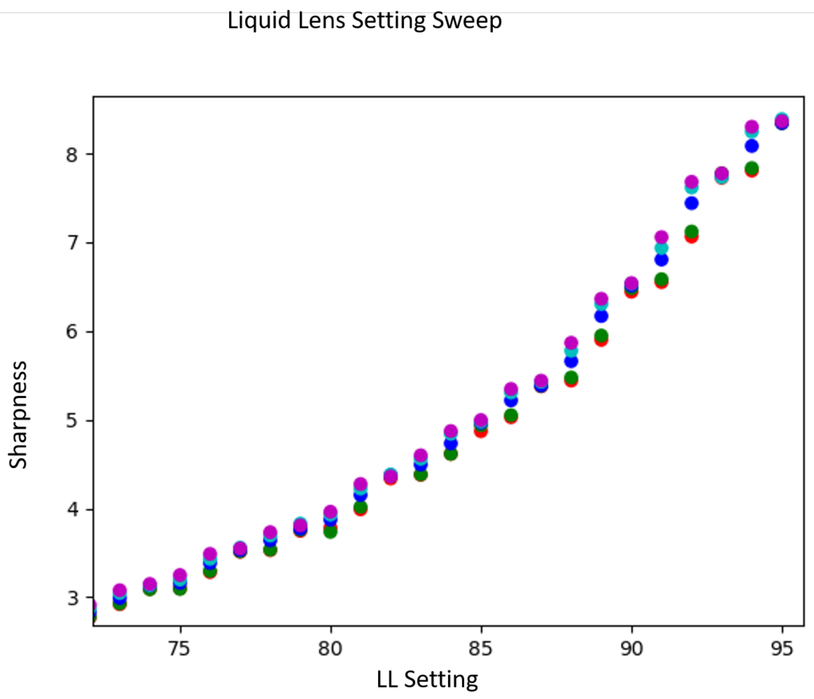

Figure 6: For a static scene, we sweep through liquid lens settings and take 5 images at each setting, computing the sharpness for each. The dot coloring corresponds to the order of images at each setting. The order of data points is Red, Green, Blue, Cyan, Magenta, and the data points are taking approximately 30 ms apart (30 Hz). Clearly the liquid lens adjust delay is on the order of tens of ms. In monotonic sweeps like this, the shape of the data would end up being more or less correct anyway. But it limits the speed at which we can change the setting back and forth. For example, trying to set the liquid lens to 80 - 85 - 70 may result in the third image being captured closer to a setting of 80 than 70.

design for them ahead of time in a monolithic application.

Our message passing interface of choice, ZeroMQ, is a general purpose high performance message marshalling library. It uses TCP or IPC as its transport protocol, although IPC is only supported on Linux. Apparently the TCP stack on Windows is smart enough to optimize sending data over TCP to the loopback address and doesn't actually incur the usual TCP overhead, but I have not confirmed this. UDP would be a more appropriate protocol for a lot of the image data we are sending, but unfortunately is not supported. On a single computer the difference does not seem to be too great, but problems may arise over a more complicated network topology (and almost certainly video problems would arise trying to stream the full-rate video feed over the internet). ZeroMQ is similar to LCM, but simpler and less fully-featured, which was an appropriate tradeoff for this project.

Figure 7 outlines the structure of the system. One set of processes interfaces directly with the hardware. This abstracts away device-specific problems like camera drivers and other interfaces specific to different hardware. A second set of processes uses the interfaces provided by the hardware nodes to display information to the user and automatically track the target. A final set of processes saves the science products generated from the first two sets, such as logging video from the cameras and state estimate of the target.

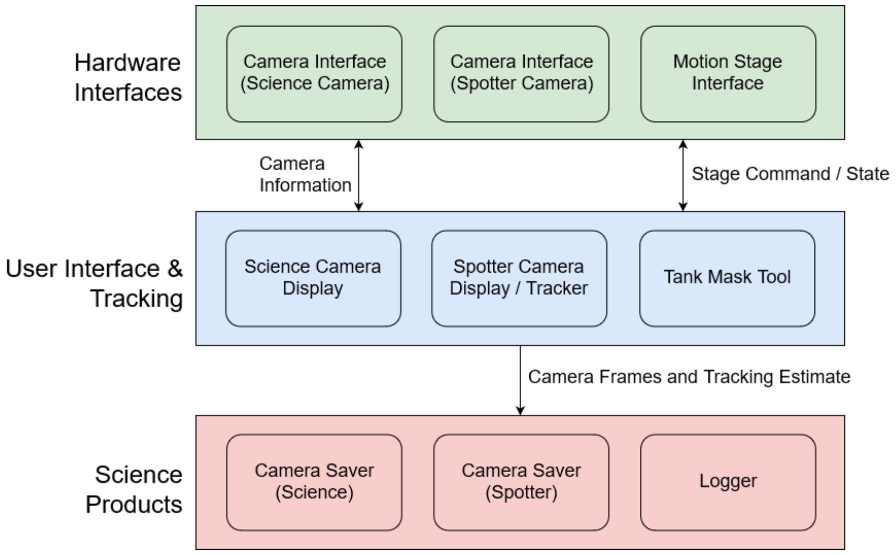

Figure 7: Flow diagram describing the software architecture. Some nodes interface with the hardware, passing images or stage position to the user interface and tracking layer. The target position estimation from this second stage and images from the hardware interfaces are saved by a final set of nodes.

5 Results

Figure 8 shows the finished tracking system and a tank used for testing. It was tested first with a set of rigid test objects (like the red plastic target in Figure 9 and later with several Ctenophores. Figure 10 shows that the multi-scale tracking system successfully follows a Ctenophore as it swims from the bottom to top of the test tank. The tracking process in Figure 10 was only tracking two dimensions (it assumed a fixed depth). Estimating depth from focus proved to be to unreliable for automatic depth estimation. The soft features of the animal and presence of other texture in the tank (e.g. bubbles on the walls, edges of the tank) made the animal's sharpness signal weaker and difficult to reliably detect. This challenge, combined with the speed limitations of the liquid lens adjustments made automatically-controlled depth tracking infeasible.

After preliminary validation of the system in the lab, we tested its performance during a research cruise on the Western Flyer. Unfortunately, it did not perform as well as hoped at sea. Relying on intensity thresholding for animal detection proved to be too brittle. It was not possible to maintain illumination and background coloring consistently enough for robust thresholding. The large amount of light transmission through the tank's walls significantly contributed to this problem. Even in the best-case scenario where we *can* ensure consistent illumination and background, Figure 10 demonstrates that the process of masking out the bright sides of the tank limits tracking in a large proportion of the tank's volume. The flexibility of the system's modular design aided the tuning and troubleshooting process as we attempted to improve performance, so although the animal detection portion of the system does not work well enough we were able to validate the overall architecture.

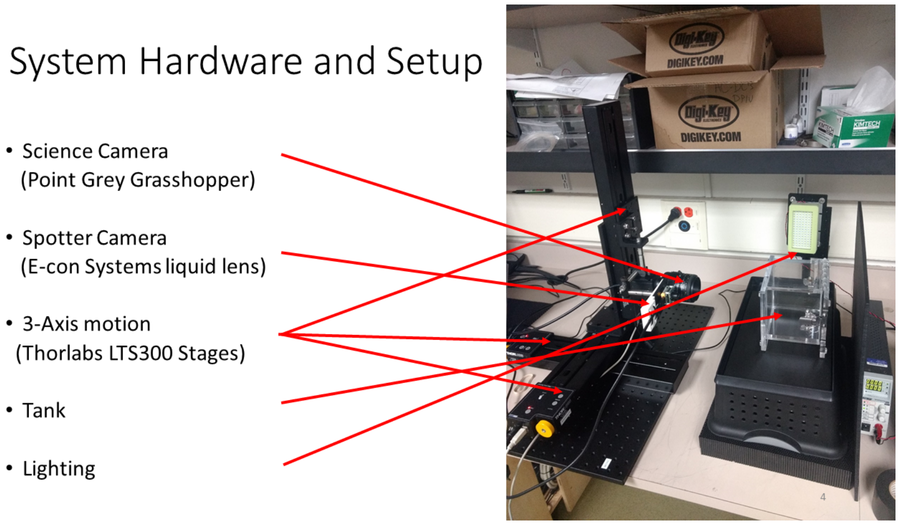
System Hardware and Setup
• Science Camera (Point Grey Grasshopper)
• Spotter Camera (E-con Systems liquid lens)
• 3-Axis motion (Thorlabs LTS300 Stages)
• Tank
• Lighting

Figure 8: Finished system construction.

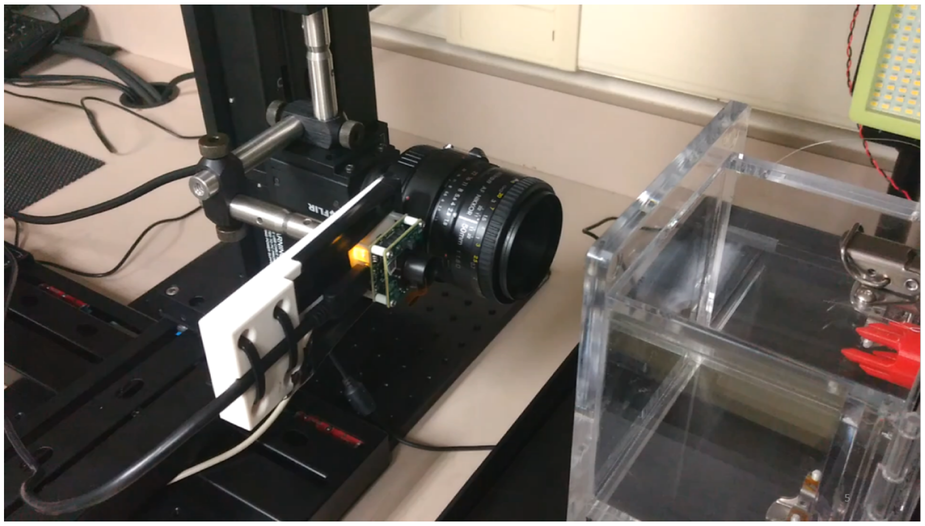

Figure 9: Closer view of the two cameras and their moving stages. The liquid lens spotter camera is on the left and the macro lens science camera is on the right.

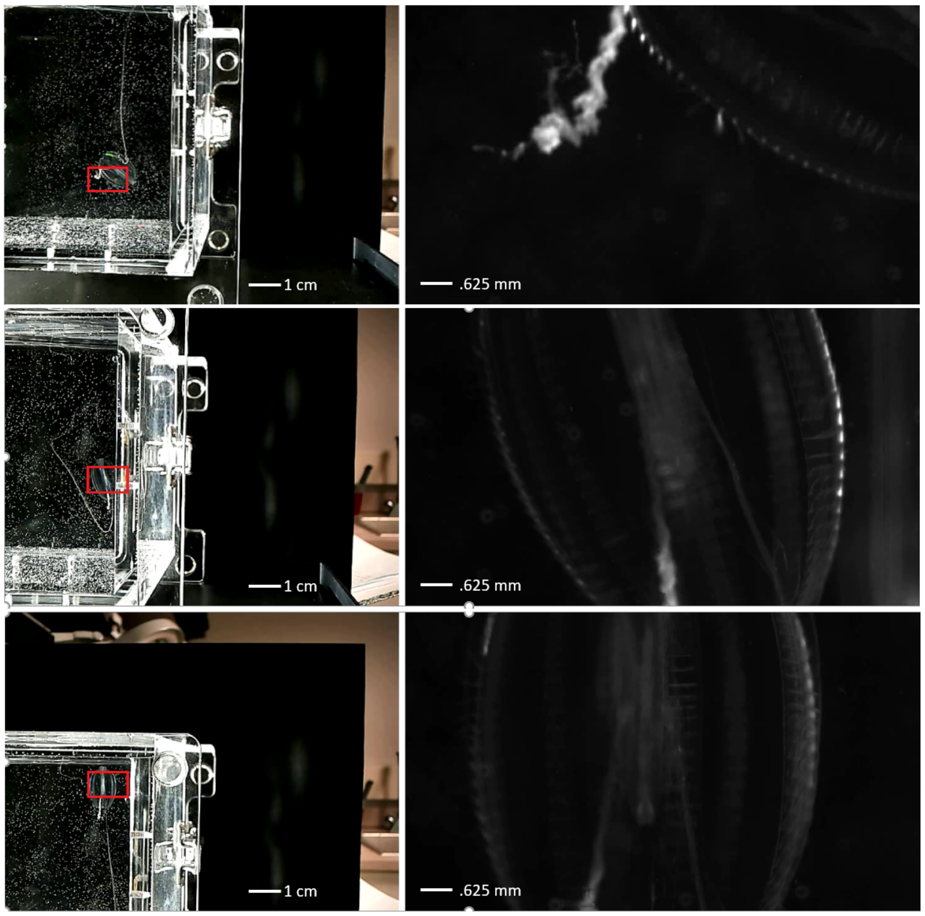

Figure 10: Successful multiscale tracking of a Ctenophore.

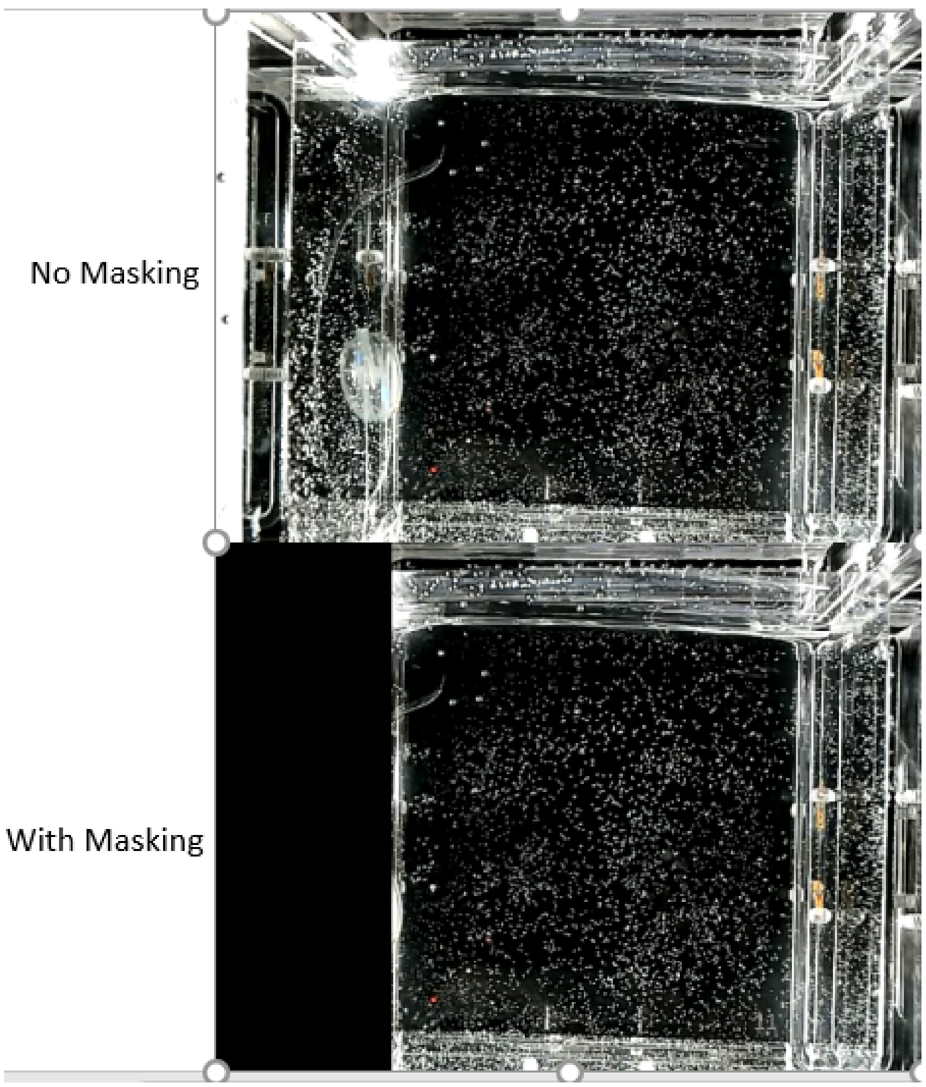

Figure 11: The problem with masking off the sides of the tank to enable thresholding-based detection.

6 Future Improvements

The tracking system's current issues stem from animal detection reliability and difficulty in depth estimation. The simplest solution to animal detection is to fix the spotter camera. A stationary spotter camera allows us to implement (dynamic) background subtraction before thresholding, greatly improving robustness to varying lighting and background conditions. Fixing the camera slightly limits the extensibility of the system, as it imposes the restriction that the tank must be small enough to be entirely seen by the camera from a single point of view. This could be mitigated in the future by adding multiple fixed spotter cameras if necessary.

Depth estimation can be solved by adding a second spotter camera and directly estimating 3D target position from camera geometry instead of sharpness. This process is much easier and more robust, at the expense of requiring an additional camera. In most cases, this tradeoff is worth the trouble.

Changing the system design to have a multiple fixed cameras and a moving science camera requires an extrinsic calibration step. When the spotter camera moves with the science camera, determining their relative offset is quite easy. With several cameras mounted in different directions, the calibration process becomes more cumbersome. During the system's testing on the Flyer cruise, the multiple, fixed spotter solution was tested, and we were able to validate that thresholding with dynamic background subtraction proved to be an extremely robust detection mechanism. We were also able to show that the target position reconstruction from multiple camera views was very straightforward to implement. Unfortunately, the camera calibration process proved to be too difficult with the time and resources available on the cruise. Despite these setbacks, we are optimistic for the science output that the system will enable in the future with just a few minor adjustments.

7 Acknowledgements

I would like to thank Kakani Katija for her mentorship this summer. I learned a lot in the Bioinspiration Lab and her guidance on this project and the time she took (out of her *very* busy schedule) was invaluable to my growth as a scientist and engineering. I would also like to thank Paul Roberts for always being there to bounce a crazy idea off of, and teaching me a great deal about image processing. Joost Daniels provided a great deal of guidance on how to design the system for ease of use in the field, and it never would have made it onto the Flyer without his guidance in the development process. Denis Klimov helped me with the lens selection process and provided valuable feedback early in the development process. Finally, I would like to thank George Matsumoto for organizing a very enriching intern program.

References

[1] Deepak Krishnamurthy, Hongquan Li, François Benoit du Rey, Pierre Cambournac, Adam Larson, and Manu Prakash. Scale-free vertical tracking microscopy: Towards bridging scales in biological oceanography. *bioRxiv*, 2019.

[2] Alexander Mathis, Pranav Mamidanna, Kevin M. Cury, Taiga Abe, Venkatesh N. Murthy, Mackenzie Weygandt Mathis, and Matthias Bethge. DeepLabCut: markerless pose estimation of user-defined body parts with deep learning. *Nature Neuroscience*, 21(9):1281–1289, September 2018.