Bluebot Pattern Design and Detection

Jack Defay

May 2023

1 Abstract

The Blueswarm system is a group of underwater, fish inspired robots that communicate implicitly by estimating pose using vision [3]. Each bluebot has 2 HD cameras, and a Raspberry Pi Zero to handle all of the image processing, planning and control. These robots currently operate only in the dark, using 3 onboard LEDs to signal their position and heading to other robots. In order to bring this system closer to deployment in real marine settings, it is crucial to develop a system that works in dynamic daylight environments.

This paper makes several contributions towards this goal, including datasets, evaluation code, and 2 pattern and algorithm designs for daylight detection of other bluebots. The two designs presented in this paper are a red fish with a custom blob detection algorithm, and a checkerboard fish, with a Haar feature based ensemble detector.

2 Introduction

The Blueswarm system is a relatively new robotic platform for underwater swarm behavior [3] [2] [1] [4]. This paper contributes to the Blueswarm system by proposing a novel design of patterns and algorithms to aid in daylight detection of bluebots. This paper builds off of two classical computer vision detectors, the blob detector and the Viola-Jones Haar feature detector [6].

The key challenge of this work was creating a favorable signal to noise ratio between the fish and the background. Although this problem would be relatively easy on a blank white background, even the controlled underwater environments that we are currently conducting experiments in pose some challenges. The lighting comes only from the top of the tank, casting shadows, there are a few features on the bottom and sides of the pool that might distract, the water distorts and attenuates the light, there are 2 windows that shine very brightly compared to the rest of the background, and the water surface often provides a near mirror image of the view. These challenges necessitate a more sophisticated fish detector. Ultimately, we hope to use these fish in natural environments, which are even less controlled. To do this, we draw inspiration from real fish

scale patterns, which are hypothesized to help fish recognize each other, and estimate pose to facilitate schooling [5].

3 Data Collection

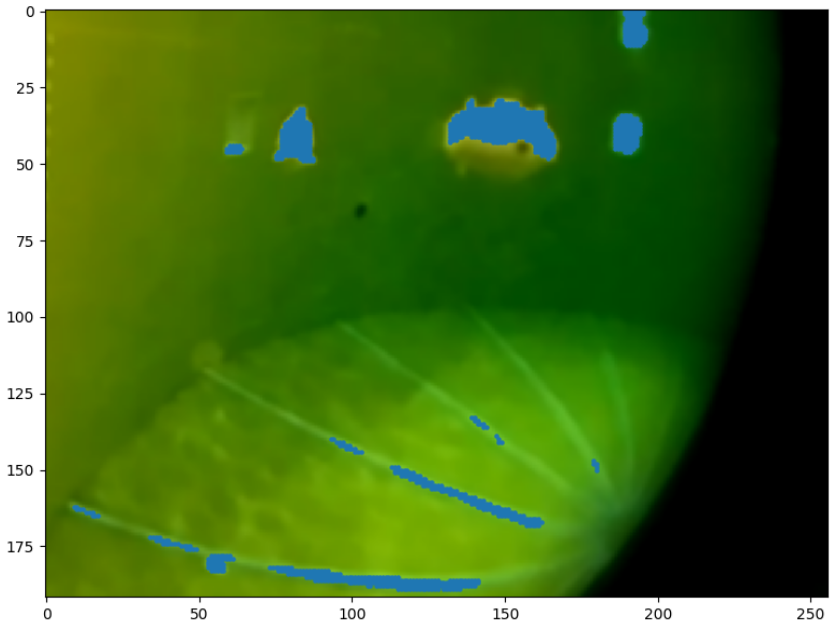

Figure 1: Photo from original underwater dataset with thresholding applied and shown in blue

I took 4 datasets during this project as I iterated on the design. First, I took a set of images during a different bluebots experiment in our large tank (12 images). One of these images with thresholding applied is shown in figure 1. Next I took a dataset of a black fish in air at different distances and backgrounds to test different OpenCV detectors (5 images). Once I decided on the pattern designs that I wanted to try, I took a third dataset in air behind the Engineering Quad, where I recorded videos in front of trees, bricks, and a gray wall (46 images). These varied backgrounds allowed me to develope specialized algorithms for each pattern and to stress test it against different backgrounds. Finally I took an underwater dataset using the patterned fish for final tuning and evaluation, and manually labeled the fish centroids (114 images). Each image was downsampled twice with the OpenCV function pyrDown and then cropped to 192x256px, the image size currently used in the bluebot's vision stage [3].

4 Design and Implementation

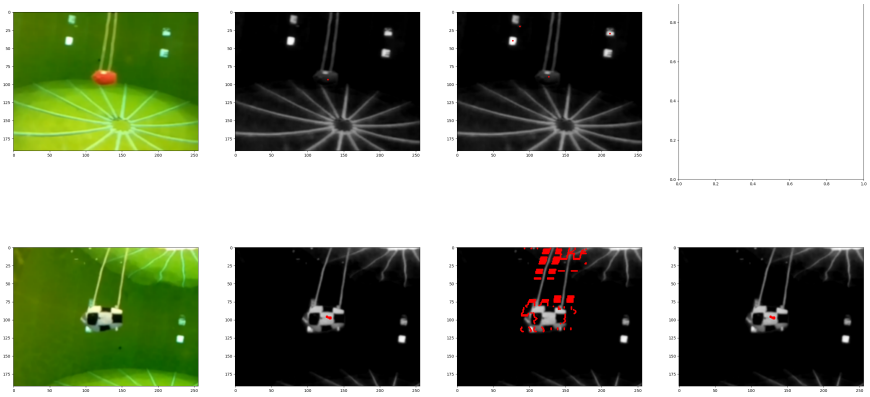

Figure 2: Top left to right: the original image of a red fish, the red_blob detector, and a stock OpenCV blob detector on the red channel. Bottom left to right: the original image of a checkerboard fish, the Haar feature checkerboard detector, the fast checkerboard detector, and the inverse-corrected checkerboard detector. Red dots are matches.

The core piece of this project was a hardware-software co-design to create robot fish that can see each other. I designed and painted fish shell patterns to be easily detectable, and then wrote algorithms to detect them.

4.1 Red Fish

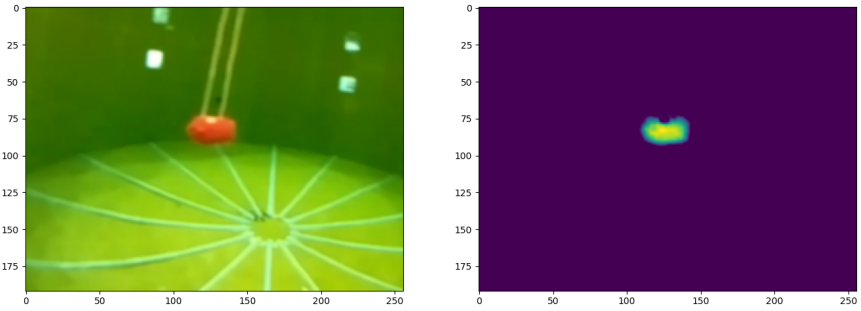

Figure 3: Left: cropped photo from underwater dataset. Right: That same photo with red filtering applied

The design of the first fish pattern is simply a red fish. Although the color red is highly attenuated by the water, it is also quite uncommon in underwater settings. In our two tanks there is no shade of red, and in the majority of

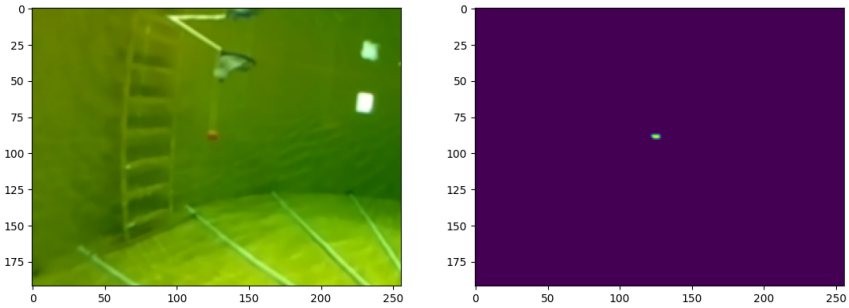

Figure 4: Left: cropped photo from underwater dataset. Right: That same photo with red filtering applied. This photo is significantly more challenging than the previous, but the filter still isolates the fish.

underwater settings there is limited to no red. Although this approach will have its limitations, it can also serve as a baseline with which to build more tailored detection algorithms on top of.

The red fish detection alogorithm builds off of previous work on the bluebots using an optimized version of pixel continuity to detect blobs and calculate centroids from a filtered and thresholded image [3] [2] [1] [4]. In previous work the pixel continuity algorithm was applied to the LED images, but in more recent (possibly unpublished) work, the author applied this same algorithm to a school of black painted fish on a white background. In this work I take a similar approach by painting the fish red, on a more complex background. The main contribution to this algorithm however, is in preprocessing the image. In order to get the filtered image in figure 3 and figure 4, I apply the following equation:

$$img\_filtered = img[:,:,0] - (img[:,:,1] + img[:,:,2])$$

Which subtracts the green and blue color channels from the red color channel. This is important because although the red fish shows up brightly in the red color channel, so does the color white. Even in these relatively controlled images, there is white from the windows, from the panels on the bottom of the pool, and from the strings and pool net I used to suspend the fish shell. This subtraction highlights the differential between red coloring and overall brightness in the image. I then clip any value less than zero to zero, and scale the image to [0,1] where the maximum brightness pixel has value 1. This step is also important because it lets the threshold parameter (currently set to 0.9) to be an absolute threshold, which lets us separate even the tiny, dull fish in figure 4 from the background due to its red color.

4.2 Checkerboard Fish

The design of the second pattern was inspired by the Viola-Jones detector and Haar features [6]. The premise is that checkerboard patterns would be uncommon in nature, kind of like a QR code. Hopefully this pattern could provide enough signal-to-noise ratio without relying on color, because color is always in danger of failure because there happens to be something else of that color in the environment. Although a HOG or SIFT detector is an obvious choice for this type of pattern, these detectors require calculating the features over the entire image, which is costly. In order to run this detector in real time, it should leverage a method designed for real time operation like the Haar feature. This detector uses an ensemble of 3x3 black and white checkerboard features at different scales (1px squares, 3px squares, etc.). By reducing the feature matching task to a binary mask, it can be very efficiently calculated with logical_and calculations over a thresholded image. And by ensembling different scale checkerboards, the detector can find fish at different distances away. After developing the original detector, I came up with two modifications: a fast checkerboard detector, and an inverse-corrected checkerboard detector.

4.2.1 Fast Checkerboard Detector

The fast checkerboard detector came from the observation that the checkerboard detector was still running significantly slower than the red blob detector. Since it was running in a nested for loop to mask the window of the image. It seemed like the best way to speed it up was to cut out one for the for loops, so I reduced the checkerboard detector to more of a barcode detector. Although this looses one dimension of the uniqueness of the pattern, it seemed reasonable that barcodes are also uncommon in nature so they should be detectable. I was also able to use the same fish images, because every 2D checkerboard embeds several 1D barcodes.

4.2.2 Inverse-Corrected Checkerboard Detector

The second modification of the original detector was a result of the poor signal to noise ratio. Although the pattern was often detectable, there were also a lot of false positives and it was difficult to separate the fish from the background. This algorithm uses the same Haar feature for matching, but also takes the inverse Haar feature (checkerboard starting with a black square in the top left corner instead of a white square). The key issue that this solves is similar to the green and blue channel subtraction in the red_blob detector. An entirely white area will fool the detector because it has a 55.5% white area, so by requiring that a window both matches with the positive mask and does not match with the negative mask, the detector can better discriminate between checkerboards and the background.

4.3 Red Blob Baseline

Finally, I wrote a detector using the simple blob detector from OpenCV on the red channel of the image as a baseline to compare our custom blob detector to.

4.4 Tuning

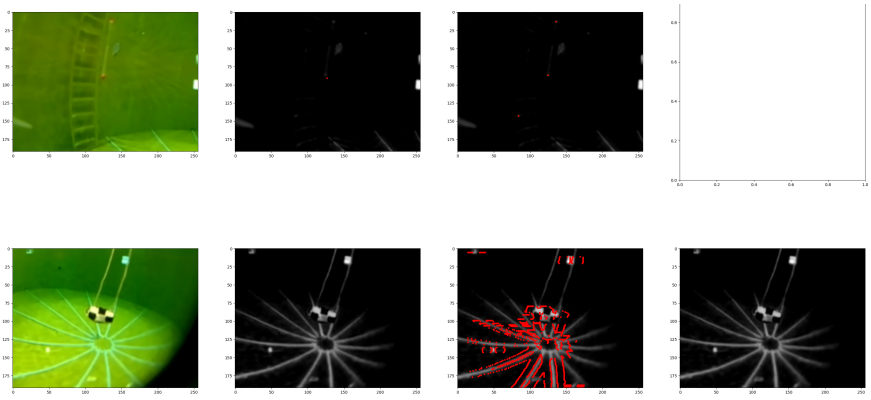

Figure 5: Top left to right: the original image of a red fish, the red_blob detector, and a stock OpenCV blob detector on the red channel. Bottom left to right: the original image of a checkerboard fish, the Haar feature checkerboard detector, the fast checkerboard detector, and the inverse-corrected checkerboard detector. Red dots are matches. This figure presents a more challenging example.

Tuning the detectors was a challenging, but very important step. Although it was fairly straightforward to get the detectors to work on the images in figure 2, there was a set of images more like those in figure 5 that were much more difficult to get. The other key challenge of tuning was to detect the fish without a high false positive rate. The first important step came from the larger modifications to the algorithms as stated above. The green-blue channel subtraction from the red image, the fast and inverse-corrected checker detectors all came from this. In order to reduce the false positive rate of the red_blob detector I also included a catch at the end similar to the LED-blob detector in [3]. If there are multiple distinct blobs detected, simply take the lowest one as the truth. This only works if the detector is well tuned to only detect the fish, but is sometimes fooled by the mirror image presented from the surface reflection. The next challenge was more in the line of parameter tuning. Although there are few parameters in these detectors, they are very important to performance. The red_blob detector had two parameters for thresholds, both of which ended up being set to 0.9 out of 1. The baseline blob detector threshold that worked the best was 0.7 out of 1. For the checkerboard detectors, I found that the smaller detectors were more sensitive than the larger detectors, and thus required higher thresholds to effectively discriminate between fish and background. I tried many ways to do

this, but found that a simple linear dependence on the side length of the detector worked best. My hope is that these algorithms will be sufficiently general for the parameter set to generalize to other situations, but further experiments will tell. Although I was able to get a good tune on the red_blob detector, I believe more work is required to properly tune the checkerboard detectors.

5 Results

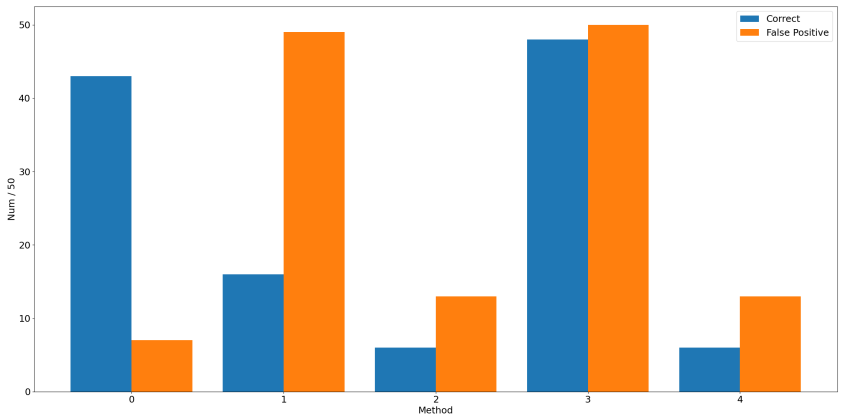

Figure 6: Evaluation of the 5 detection algorithms on a dataset of 50 underwater images taken by bluebots. Left to right the methods are: red_blob, red baseline, checkerboard, checkerboard fast, and checkerboard inverse-corrected.

The methods are evaluated on 3 metrics: accuracy, false-positive rate, and runtime. We want the highest accuracy, with the lowest false positive rate, with few pathological failures besides those induced by distance. We also require a very low runtime in order to run realtime on the computationally limited Raspberry-Pi zero. For reference, the current vision algorithm employs a very similar algorithm to the red_blob approach, and that is barely fast enough to give the fish a fast enough control loop. Note the log scale on the y-axis of the runtime plot. Also note that these evaluations were done on the same dataset as the tuning, so this is a validation set not a test set.

5.1 Accuracy

The accuracy of these methods depends a lot on the specific tune, and this test unfortunately only covers a limited amount of generalizability of the algorithms. Although there is pretty significant visual variation in these datasets, this is only in one specific pool at one specific time. The red_blob detector performs quite well, with an accuracy of 86% (fig. 6). When looking at the failure cases, there doesn't seem to be anything special about them, so I will investigate and

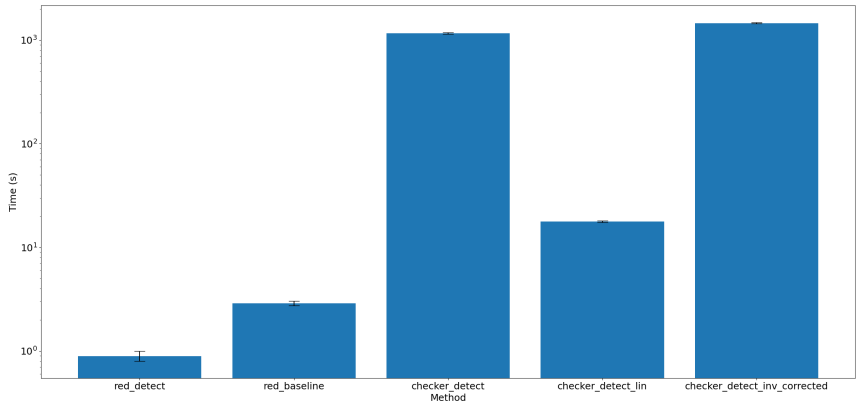

Figure 7: Runtimes of the 5 detection algorithms on a dataset of 50 underwater images taken by bluebots. Left to right the methods are: red_blob, red baseline, checkerboard, checkerboard fast, and checkerboard inverse-corrected. Note the log scale on the y-axis.

hopefully we can get this number even higher. The blob baseline performs much worse, and when inspecting the failure cases it seems that it is able to get the cases where the fish is close, but when the fish is far or turned to face the camera, it is difficult to distinguish it from the background. It also gets easily thrown off by the windows, reflections, and other distractors. The checker detect and inverse corrected perform very similarly, the main culprit was whenever the fish was not at exactly eye level, only two rows of the checkerboard could be seen, making it much harder to match the template. The fast checkerboard detector was able to identify most of the fish, but with a very high false positive rate. This detector actually allowed it to get around the previous problem, but I believe it needed to be tuned to be a bit less sensitive. Looking at failure cases showed that it almost always matched with many features in the environment.

5.2 Runtime

This plot shows the average and std. runtime of the different algorithms over the underwater dataset. The red_blob algorithm peforms very well, with an average runtime of only 0.9 seconds compared to the baseline blob detector at 2.9s. The checkerboard and inverse-corrected checkerboard took significantly longer to run at 1162s and 1460s, respectively, while the fast checkerboard was much improved at only 18s. The red_blob detector has only a single nested loop on the number of pixel matches, the checkerboard detectors have a nested for loop over the entire image repeated for currently 4 different sized masks, and the fast checkerboard detector has a single loop over the width of the image, repeated for 4 masks. In their current form, only the red_blob algorithm could be run in real time onboard the fish, but I hope that with further development,

the Haar feature approach could be used as well. Each of these algorithms run significantly faster than alternatives that use HOG or SIFT features from earlier, qualitative analysis.

6 Conclusion

The performance both in accuracy and runtime of the red_blob detector is very promising, and I hope to implement this strategy on the bluebots for future experiments. The Haar feature checkerboard detectors show promise, but are not yet able to effectively detect the fish, and are too slow to run onboard. The datasets collected will inform new pattern and algorithm designs, and the codebase is sufficiently general to plug in new algorithms to evaluate. This project makes somewhat simple, yet significant contributions to the Blueswarm project, and hopefully will be incorporated into the next iteration of the robots.

Next steps for this project include exploring new types of detectors, collecting a more comprehensive dataset across different underwater environments, and evaluating the red_blob detector onboard a bluebot. I also plan to incorporate benefits of both approaches into a red-barcode fish, with either red and black or maybe red and blue stripes. I suspect that the more distinct red coloring will improve the signal to noise ratio, while the barcode design will protect against random red objects in the environment. Perhaps an orange and black striped fish would work well too, a Princeton tiger-fish.

7 Contributions

This project was almost entirely original work. I borrowed a few code snippets from the Blueswarm project. Namely the "thresholding" and "continuity" functions from the vision pipeline. To collect most of the data, I performed experiments in our large pool with help from Dr. Di Ni using the Blueswarm system.

References

[1] Florian Berlinger, Julia T. Ebert, and Radhika Nagpal. "Impressionist Algorithms for Autonomous Multi-Robot Systems: Flocking as a Case Study". In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). ISSN: 2153-0866. Oct. 2022, pp. 11562–11569. DOI: 10.1109/IROS47612.2022.9981448.

[2] Florian Berlinger, Melvin Gauci, and Radhika Nagpal. "Implicit coordination for 3D underwater collective behaviors in a fish-inspired robot swarm". In: *Science Robotics* 6.50 (Jan. 13, 2021). Publisher: American Association for the Advancement of Science, eabd8668. DOI: 10.1126/scirobotics.abd8668. URL: https://www.science.org/doi/full/10.1126/scirobotics.abd8668 (visited on 05/02/2023).

[3] Florian Berlinger and this link will open in a new window Link to external site. "Blueswarm: 3D Self-Organization in a Fish-Inspired Robot Swarm". ISBN: 9798534680669. PhD thesis. United States – Massachusetts: Harvard University, 2021. 178 pp. URL: https://www.proquest.com/docview/2564171061/abstract/5B15CD83A10A4A97PQ/1 (visited on 05/02/2023).

[4] Florian Berlinger, Paula Wulkop, and Radhika Nagpal. "Self-Organized Evasive Fountain Maneuvers with a Bioinspired Underwater Robot Collective". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. 2021 IEEE International Conference on Robotics and Automation (ICRA). ISSN: 2577-087X. May 2021, pp. 9204–9211. DOI: 10.1109/ICRA48506.2021.9561407.

[5] Tony J. Pitcher, ed. *The Behaviour of Teleost Fishes*. Boston, MA: Springer US, 1986. ISBN: 978-1-4684-8263-8 978-1-4684-8261-4. DOI: 10.1007/978-1-4684-8261-4. URL: https://link.springer.com/10.1007/978-1-4684-8261-4 (visited on 05/09/2023).

[6] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Texts in Computer Science. Cham: Springer International Publishing, 2022. ISBN: 978-3-030-34371-2 978-3-030-34372-9. DOI: 10.1007/978-3-030-34372-9. URL: https://link.springer.com/10.1007/978-3-030-34372-9 (visited on 05/02/2023).