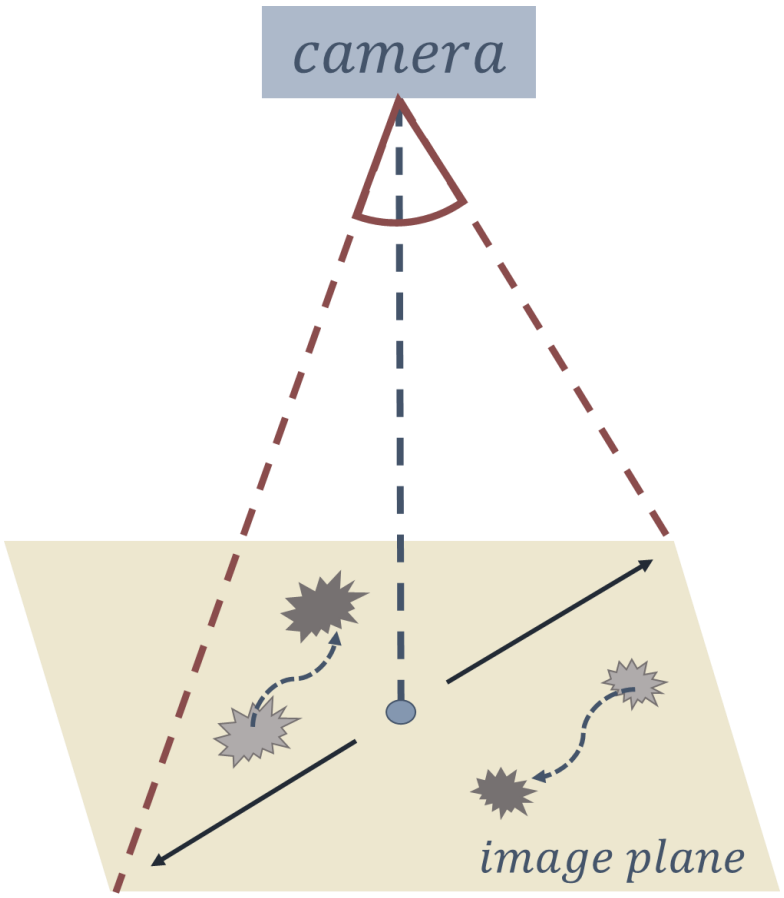
Tracking Marine Organisms and Submerged Objects with an Underwater Video System

DanaRose Brown

Faculty Advisor: Dr. Diane Foster

TECH 797

26 April 2019



Sea Grant
New Hampshire

Abstract

Observing movements of underwater organisms and objects is difficult and tedious; however, cameras can simplify this task by making it more efficient and autonomous. A system for tracking marine organisms and submerged objects was developed using GoPro HERO7 cameras and MATLAB image processing tools. To account for image distortion due to underwater refractivity, the underwater field of view of the cameras in saltwater was used to determine the image area and necessary focal length, and calibration images were used to correct distortion. Using image processing techniques for extracting prominent features in an image, an algorithm was developed to detect multiple organisms or objects within an image. Dried sand dollars were used to develop the object detection algorithm since their color and shape were easily detectable against a contrasting background. Multiple proof-of-concept tank tests were completed to optimize the system using *Strongylocentrotus droebachiensis* (green sea urchins), during which images were collected at five second intervals to track multiple sand dollars' individual movements within the tank over one to two hour time spans. A field deployment was conducted in conjunction with the ongoing deployment of a Pressure Mapped Munition (PMM) in order to test the system's ability to track a non-circular object in nearshore oceanic conditions.

Introduction

The seafloor can be littered with a range of objects, such as marine plastics, large rocks, or unexploded ordnances, and organisms such as sand dollars and sea urchins. Whether in response to large flows or self-propulsion, these objects and organisms regularly move. Tracking objects and organisms on the sea floor can be difficult and tedious; however, cameras can simplify this task by making it more efficient and autonomous.

The objective of this project was to design a camera system for tracking marine organisms and submerged objects in order to autonomously determine the motions of these objects underwater by developing a Tracking Algorithm to analyze the objects' motions. The data collected and processed using the Tracking Algorithm returns the tracks of the objects over time, and makes measurements of their dimensions, distances traveled, and velocities. The tracking system uses a GoPro HERO7 Black to collect image frames at desired intervals, which are then processed using the Tracking Algorithm.

The Tracking Algorithm uses image processing techniques in MATLAB to detect objects within an image. The images in a data series must be converted to binary, allowing the prominent features within

the image to be detected and for the diameter and centroid of these locations to be determined. Using consecutive images of an object moving, its location can be tracked over time. When detecting multiple objects in an image, the paths of the individual objects can be confused between images, resulting in an unorganized path being returned by the Tracking Algorithm. In order to account for this, a separate Sorting Algorithm was developed which is used to correct the unorganized paths, returning clear object tracks that accurately reflect the positions of the objects.

Multiple tank tests of the system were completed in the Sea Urchin Aquaculture Lab at the University of New Hampshire (UNH) using five green sea urchins (*Strongylocentrotus droebachiensis*) in a tank to show proof-of-concept, improve the system's detection abilities, and highlight the capabilities of the system such as: consistently measuring the urchins' diameters and determining their distances traveled and velocities. A field deployment was also completed at Wallis Sands, New Hampshire, in conjunction with the ongoing deployment of a Pressure Mapped Munition (PMM). The PMM is a tool developed by UNH researchers to observe how unexploded munitions being tested by the Department of Defense move and bury in sediment. This application has an important societal and environmental impacts as it will help researchers and the Department of Defense gain more insight on how unexploded ordinances bury and move, in order to find them when their location becomes unknown during testing. This field deployment was used to test the system's ability to detect and track a noncircular object in nearshore ocean conditions. This field deployment highlighted many of the challenges associated with working in the nearshore ocean environment such as water clarity, tides, and light availability; however, the noncircular shape of the PMM was successfully detected during periods of good water clarity.

The camera tracking system that was developed offers researchers the ability to make observations autonomously of important processes for both living organisms and objects in the ocean environment. By automating the process of detection and tracking using cameras and the tracking algorithm that was developed, a human does not need to be present at the experiment site and can gain information on the motions of underwater objects at the click of a button, versus sorting through images manually or physically watching the object during the testing period. Some important applications the tracking system can be used for include: observing the motions of unexploded ordnances and how they bury in sediment, such as with the testing completed using the PMM, and observing the motions of marine organisms, such as sand dollars, and their behavior when feeding or spawning. The system's capabilities can be applied to making observations of objects and organisms that move on the sea floor, and future improvements to the system look to expand its capabilities to more

applications such as observing the transport of marine plastics in the nearshore zone and observing scour around pilings and underwater manmade structures.

System Design

Materials and Physical System Assembly

GoPro HERO7 Black cameras were chosen for the system due to their wide variety of image and video capture settings and waterproofing (**Figure 1**). This camera has a "linear" operating mode, unlike older GoPro's which capture images with fisheye distortion, and therefore was selected to simplify the problem of removing distortion in images. Two GoPro HERO7 Black cameras were used for this project to simplify field testing – one camera was left in the field recording data while the other charged, and then the next day they were switched out. These cameras have a depth rating of 10 meters and a battery life of about 2 hours.

In order to extend the battery life of the cameras, two Re-Fuel 9-hour Action Pack Battery Life Extenders, which claim to be waterproof up to 10 meters, were used as well. When both the Go Pro and battery life extenders are fully charged, the battery lasts for over 10 hours capturing images in 2 second intervals. During field testing, the battery pack did not stand up to its expected waterproof rating of 10 meters, as one of the battery packs flooded during deployment. The expected reasons for this malfunction are discussed in the Field Deployment section below. Multiple lights were also purchased and used during tank testing. The lights that purchased were Suptig Diving Light High Power Dimmable Waterproof LED Video Lights. These components are shown in **Figure 1** below.

Figure 1: Left – GoPro HERO7 Black camera [2]. Middle – Re-Fuel Action Pack Extended Battery [8]. Right – Suptig Dimmable Waterproof Underwater Dive Light [13].

A testing frame was built to hold the camera steady when collecting images for tracking algorithm development and also to mount the camera and lights during tank testing. The frame was assembled using 80/20 aluminum framing, which allowed the height of the camera to be adjusted depending on the desired focal length. Attachment pieces on the frame allowed the camera and lights to be attached at an angle, or the camera could be mounted looking straight downwards by attaching it to the bottom of the cross bar on the frame. The testing frame is shown in **Figure 2** below.

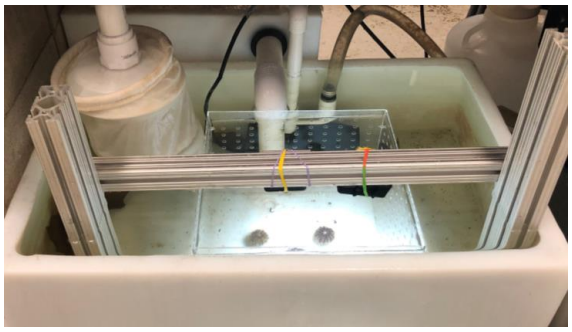


Figure 2: The testing frame during tank testing in the Sea Urchin Aquaculture Lab.

Another frame was built to mount the camera during field testing. The frame was designed to fit on a 3 inch diameter jetted pipe used to mount instruments during deployment. The frame consisted of a 6 inch long size 3 pipe stainless steel pipe cuff with a 24 inch long, 1 inch wide, 3/8 inch thick steel bar welded horizontally to the pipe cuff at the center of the cuff in the middle of the steel bar. A piece of 1 inch wide 80/20 aluminum framing was then attached to the steel bar using 80/20 fasteners and hose clamps when necessary. The length of the 80/20 framing can vary depending on the requirements of the application, which can affect the distance the camera is placed from the jetted pipe. An attachment piece with a GoPro connector secured to it was then positioned on the 80/20 framing so the camera faced downwards when attached. This design using the 80/20 framing and GoPro attachment piece allows the camera to be attached to the deployment frame at varying distances from the jetted pipe because the location of the GoPro attachment can be adjusted easily. A T-bolt is threaded through a hole on the reverse side of the pipe cuff to fasten the deployment frame to the jetted pipe. The deployment frame is shown in **Figure 3** below.

Figure 3: The field deployment frame in use at Wallis Sands Beach, New Hampshire. The cuff is secured on a jetted pipe, and the camera is extended on the mount away from the jetted pipe positioned with the lens facing straight downwards.

Field of View Calculation

Refraction is the bending of light that occurs when light enters one medium from another, such as entering water from air [9]. Underwater, cameras experience refraction, which can alter the image that is being viewed. Due to the refractivity of seawater, the camera's angle of coverage (field of view) is altered and must be calculated in order to determine the correct image area that will be captured when the camera is set at a certain focal length, or distance from the target subject [5]. The concept of refractivity and its relationship to camera angles is shown in **Figure 4** below.

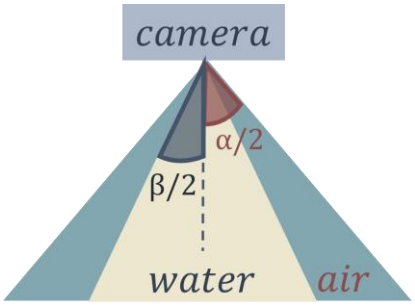


Figure 4: The relationship between the angle of coverage in air versus the angle of coverage in water as it is affected by refraction. The angle of coverage in air is represented by α, and the angle of coverage in water is represented by β.

The angle of coverage for a GoPro HERO7 operating in linear mode is 98.7° in air, which has a refractivity index of 1 [4]. Underwater, the refractivity index of saltwater must be considered (**Figure 5**). In the Gulf of Maine, salinities on average range from 32-34 ppt, so a refractivity index of 1.339 was used to calculate the underwater angle of coverage [3].

Table 2. Specific gravity and refractive index as a function of seawater's salinity of seawater. The bold rows (34-36 ppt) represent the range usually encountered in the open ocean.

Salinity (ppt)	Specific Gravity at 25° C	Refractive Index (20° C)
0	1.0000	1.33300
30	1.0226	1.33851
31	1.0233	1.33869
32	1.0241	1.33886
33	1.0249	1.33904
34	**1.0256**	**1.33922**
35	**1.0264**	**1.33940**
36	**1.0271**	**1.33958**
37	1.0279	1.33976
38	1.0286	1.33994
39	1.0294	1.34012

Figure 5: Refractivity indices for seawater at varying salinities [10].

Snell's Law states that "the ratio of the sines of the angles of incidence and refraction of a wave are constant when it passes between two different media" [5]. In this case, light is passing through air into water, and the bending of light underwater causes distortion in the image and affects the camera's angle of coverage when underwater. Snell's Law (**Equation 1**) was used to calculate the angle of coverage for the camera when underwater, where α is the angle of coverage in air, β is the angle of coverage underwater, n_α is the refractivity index of air, and n_β is the refractivity index of seawater. This equation was then rearranged to solve for the underwater angle of coverage, β (**Equation 2**).

$$\frac{\sin\left(\frac{\beta}{2}\right)}{\sin\left(\frac{\alpha}{2}\right)} = \frac{n_\alpha}{n_\beta} \tag{1}$$

$$\beta = 2 * \arcsin\left(\frac{n_a}{n_\beta} * \sin\left(\frac{\alpha}{2}\right)\right) \tag{2}$$

Using a refractivity index of 1 for air, 1.339 for seawater, and angle of coverage in air of 98.7°, an underwater angle of coverage of 69.3° was calculated.

The geometry of the camera showing the relationship between the field of view angle and the focal length and area of the image plane are shown in **Figure 6** below.

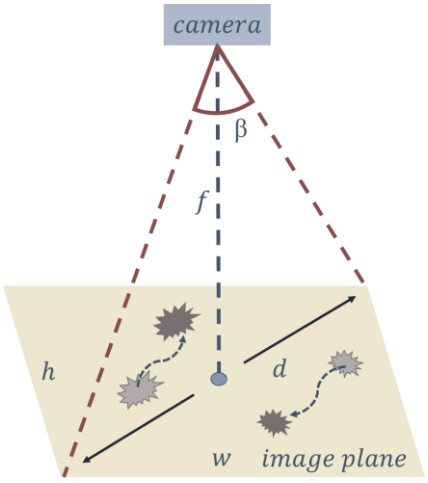


Figure 6: The camera view area for a given field of view angle and focal length is shown in yellow. The camera is downward-looking at the top of the figure, and the camera angle of coverage is expressed by β, the diagonal angle of coverage. The area of the image plane, or area that is viewed by the camera at a certain focal length, has the dimensions h and w.

The focal length is the required distance from the camera lens to the subject to view the desired image plane area. The focal length can be calculated as a function of the height (h) and width (w) of the image plane, and the angle of coverage ($\boldsymbol{\beta}$) by re-arranging the angle of coverage equation (**Equation 3**) to solve for focal length (**Equation 5**), where d is the diagonal length of the image plane (**Equation 4**).

$$\beta = 2 * \arctan\left(\frac{d}{2*f}\right) \tag{3}$$

$$d = \sqrt{h^2 + w^2} \tag{4}$$

$$f = \frac{d}{2*\tan\left(\frac{\beta}{2}\right)} \tag{5}$$

Using the calculated angle of coverage for water, 69.3°, the required minimum focal length was determined for different applications of the camera system – shown in **Table 1** below. The "minimal distance desired" was estimated for each application as the smallest side of a square image area that would be necessary for tracking the desired objects. The image plane has an aspect ratio of 3:4 (height:width), so the "minimal distance desired" is the distance that was chosen for the height of the image since this is the smaller side, and the width was 4/3 the length of the height.

Table 1: Estimated desired focal length requirement for expected applications of the tracking system.

Application	Minimal Distance Desired [m]	Diagonal Distance [m]	Focal Length [m]
Urchin Tank	0.25	0.42	0.30
Munition	0.75	1.25	0.90
Marbles	0.30	0.50	0.36
Sand Dollars	1.00	1.67	1.20

Calibration and Distortion Removal

Underwater, refraction can also cause distortion that makes objects closer to the edges of the image to appear stretched out and straight lines appear curved. A calibration was necessary to remove distortion in underwater images. A black and white checkerboard grid with 2.5 cm squares (**Figure 7**) was used to calibrate the camera during tank testing. Once the camera was attached to the testing frame and set up in the urchin tank, the grid was placed flat on the bottom surface of the urchin tank and moved into different locations and orientations within the camera's view. The camera was set to linear view in time lapse photo mode, capturing images at 5 second intervals. For each calibration, 15-20 calibration images were captured, allowing enough total images incase any were rejected by the program.

The images were loaded into the MATLAB Camera Calibrator [11], shown in **Figure 7** below. For this specific calibration, a total of 17 calibration images were captured; however, 5 were rejected by the Calibrator. The calibrator locates the intersection of the black and white boxes and determines an origin for the checkerboard.

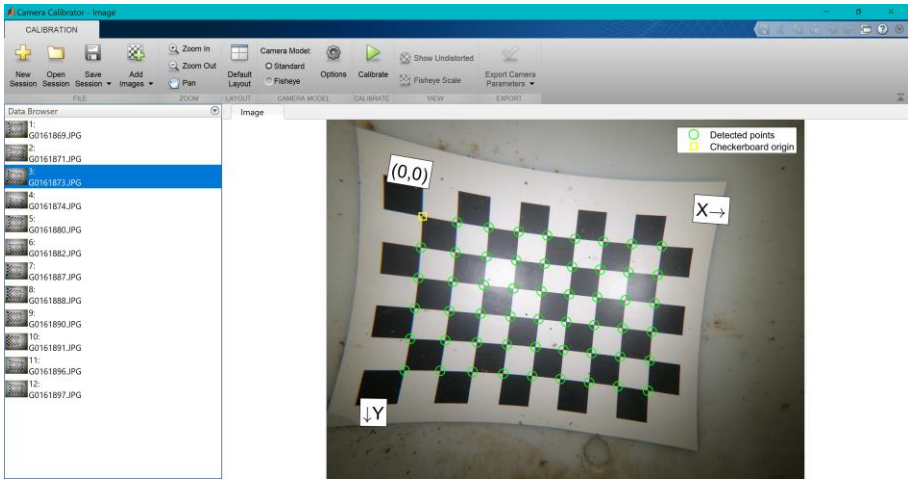


Figure 7: The camera calibrator pre-calibration. Calibration grid images are loaded in to the calibrator and the intersection points between grid squares are detected for each grid.

Once all images were loaded into the calibrator and the grid points were detected, the camera was calibrated. During calibration, the camera intrinsic parameters, extrinsic parameters, and distortion coefficients were calculated, including: world points, 3-D camera rotation, and camera translation relative to the world coordinate system. Calibration accuracy was evaluated by examining the reprojection errors, and one of the 12 images was removed due to a high reprojection error, and the camera was recalibrated to return the results shown in **Figure 8**.

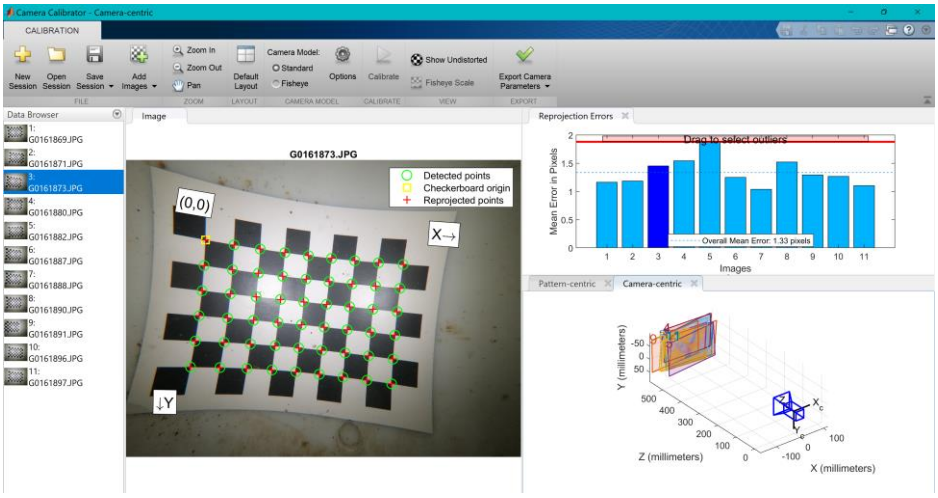


Figure 8: The camera calibrator post-calibration. The detected and re-projected points for each calibration image are shown (middle) as well as the reprojection errors in pixels for each (top right). A camera-centric view of the relationship between the camera and the calibration grids is shown in the bottom right.

The calibration was applied to the test images to remove the “pincushion” distortion that occurs when the camera is underwater. An example of the calibration grid pre-distortion removal and post-distortion removal is shown in **Figure 9** below. The edges of the grid squares and paper are visible bent in the pre-distortion removal image, while post distortion these edges appear straight.

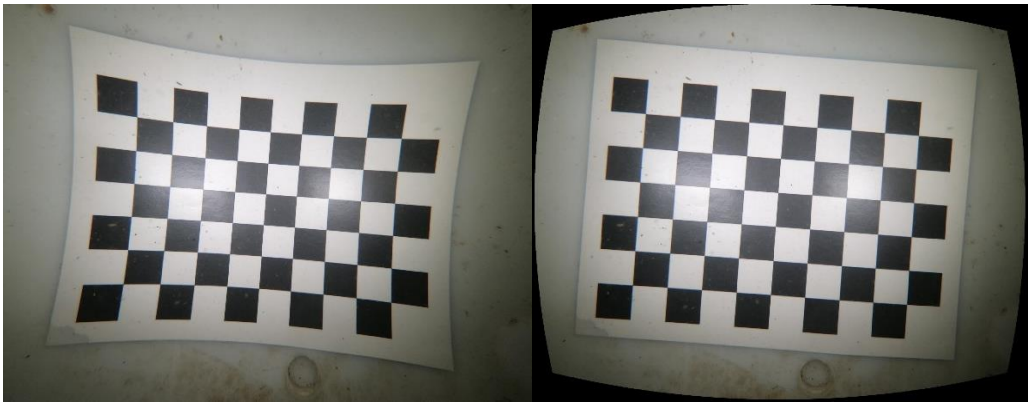


Figure 9: Distorted calibration grid pre-calibration (Left) and undistorted calibration grid post-calibration (Right).

The calibration images were also used to determine the relationship between pixels and distances in an image. The squares in the calibration grid are 2.5 cm by 2.5 cm, so the corresponding pixel length was measured to determine the pixel-metric relationship that was used to measure the size of objects in the image as well as the distance and speed at which the objects being tracked moved. Before making these measurements, the calibration was applied to the tracking images of the objects to properly measure distances in the undistorted images.

Object Tracking Algorithm Development

The Object Tracking Algorithm was coded in MATLAB and uses image processing techniques to detect objects in images by extracting prominent features within the image (example in **Appendix A**). Dried sand dollars were used as the “object” when developing the Tracking Algorithm because their color and shape could be easily detected. Multiple images of the sand dollars as they were moved to different locations were captured so the “tracks” of the sand dollars could be detected.

To detect objects in a single image, the image is first converted to binary (**Figure 10**). The necessary threshold for the image must be determined either by using the “graythresh” command on a grayscale or by trial and error (entering fraction values less than 1, usually close to the threshold, until

the objects in the image are distinguished from the background) [1]. The threshold is used to convert the image from gray to binary and specifies what pixels are converted to binary ones (white) or zeros (black). In order to be detected, the objects must be ones (white) and the background must be zeros (black). In some cases where the object is darker than the background, so the objects are shown in black and the background in white, the complement of the image must be used to represent the objects in white and the background in black.

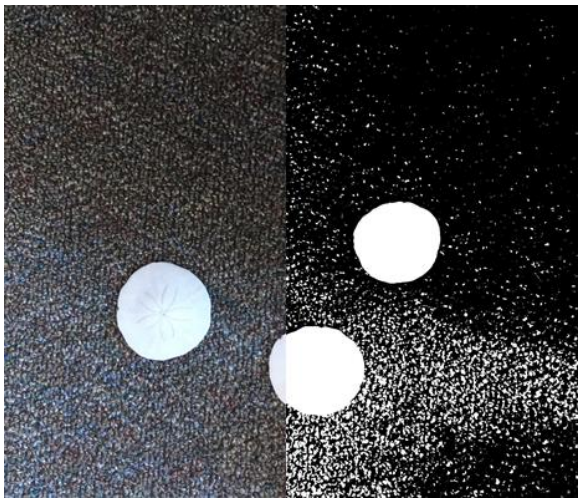


Figure 10: A test image of three sand dollars is converted to binary to detect the prominent features within the image.

Once the image is converted to binary so that the objects are represented by ones (white) and the background by zeros (black), the Hough transform is used to locate the prominent features (i.e. the objects) within the image. The Tracking Algorithm then calculates the region properties for the prominent features such as the centroid location, major axis length, and minor axis length of each feature. When detecting circular objects, the diameter of each object is then calculated using the major and minor axes lengths. The expected diameters of the objects are specified to reduce detection of undesired objects or light spots. Using the radius and centroid of each object, the detected object is then outlined using “viscircles” (**Figure 11**). When detecting non-circular or obscurely shaped objects, MATLAB function “BWboundaries” is used to outline the object, and the centroid and dimensions of the object are determined the same as if the object were circular (**Figure 11**).

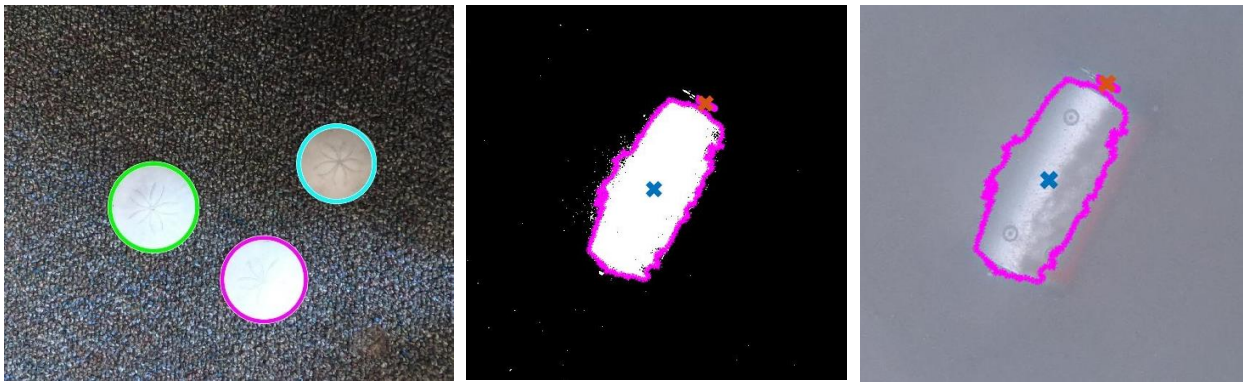


Figure 11: Left – The sand dollars in the test image are detected separately and their outlines are shown. Middle, Right – The PMM is detected and its centroid and outline are determined.

In order to track objects over time, multiple images or video frames must be captured of the object moving and loaded into the algorithm. The process described above is then completed for every frame and a matrix is populated with the locations of the centroids for each detected object. Each image has its own row in the matrix, and the centroid locations fill the columns. Another matrix contains the calculated radii for the objects in every image. Occasionally, the incorrect object centroid coordinates are populated to the wrong column in the matrix, and post-sorting of the matrix is required to properly determine and display the object's path. The Sorting Algorithm was developed to solve this issue and is described in the next section.

Without sorting the centroid matrix, the detected object tracks can appear incorrect and disorganized such as in **Figure 12** below.

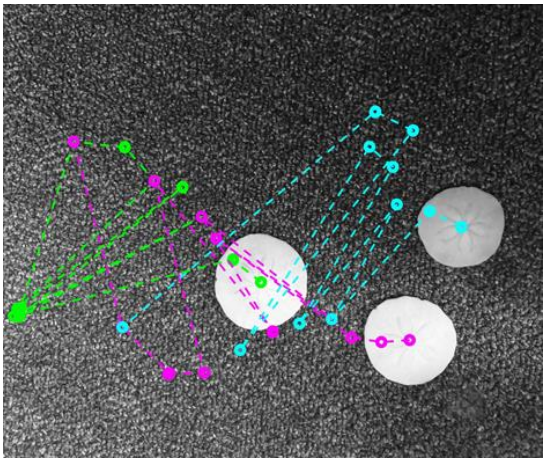


Figure 12: Unorganized object tracks that result from improper object sorting when detecting multiple objects in a single image.

Sorting Algorithm Development

Detecting multiple objects in consecutive images is much more difficult than detecting a single object because the algorithm must have a way to distinguish between each individual object in every frame in order to track each object individually. The Tracking Algorithm populates a matrix with the detected centroid locations for each object in an image (example in **Appendix B**). However, due to the way MATLAB searches through an image to find prominent features (i.e. the detected objects), the objects can be grouped incorrectly within the matrix. A separate algorithm is used to sort this matrix – it organizes the tracks of the objects after the tracking algorithm is used to detect their locations for each frame captured during the experiment. This algorithm, referred to as the Sorting Algorithm, uses the detected centroid locations of the objects to determine which object in each consecutive image corresponds with each object in the previous image. For each application, image frames are collected at a rate at which the object should not move a large distance from its location in the previous frame, so the object in each frame that has the closest centroid to the object in the previous frame is considered to be the same object. A new "sorted" matrix is created, which groups together the centroids of each object throughout every image frame of the experiment, outputting the correct track of each individual object.

In order to return the sorted tracks of the detected objects, the matrix of detected object centroids output from the Tracking Algorithm are input to the Sorting Algorithm. An empty cell array is created with the same dimensions as the unsorted matrix, to later be populated with the sorted centroid locations. For circular objects, the matrix of calculated radiuses is also input to the Sorting Algorithm and rearranged in a new matrix to match with their corresponding centroids as they are sorted.

The algorithm sorts through the matrix row by row, comparing the centroid values in the row above to the row below to find the closest centroids in the two rows. The first row of the new sorted matrix is filled with the values of the first row of the unsorted matrix. The algorithm then looks at the x and y centroid values in the first column of this row, and compares it with the x and y values of all of the centroids in the corresponding row below it in the unsorted matrix until it finds the closest match, which then is placed in the same column below this value in the new matrix. The x and y values of the matrix are compared using the following equation (**Equation 6**):

$$centroid\ difference = |x(i+1) - x(i)| + |y(i+1) - y(i)| \quad (6)$$

This process is repeated for each value within a row, and then repeated for every row until the new matrix is full of sorted centroid values.

In order to account for duplicate centroid values in the same row that can result from sorting the matrix when one of the objects leaves the view of the camera, a similar process is repeated. Instead, the centroid values of duplicates are compared to the centroid values of the corresponding object in the row above by determining the difference between the x and y values of the centroid in the row above with the x and y values of the centroid in the row below. Whichever pair has a larger difference is assumed to be the result of the object leaving the image plane area and the value of the centroid in the row below is removed and replaced with not a number (NaN). After the duplicate centroid values are removed, the matrix has been sorted so that a single column is filled with the centroid values of one object for each image frame.

The figure below (**Figure 13**) shows an example of an unsorted track that was output by the Tracking Algorithm. This result was then sorted, resulting in the clear, correct paths traveled by each of the objects in the image on the right in **Figure 13**.

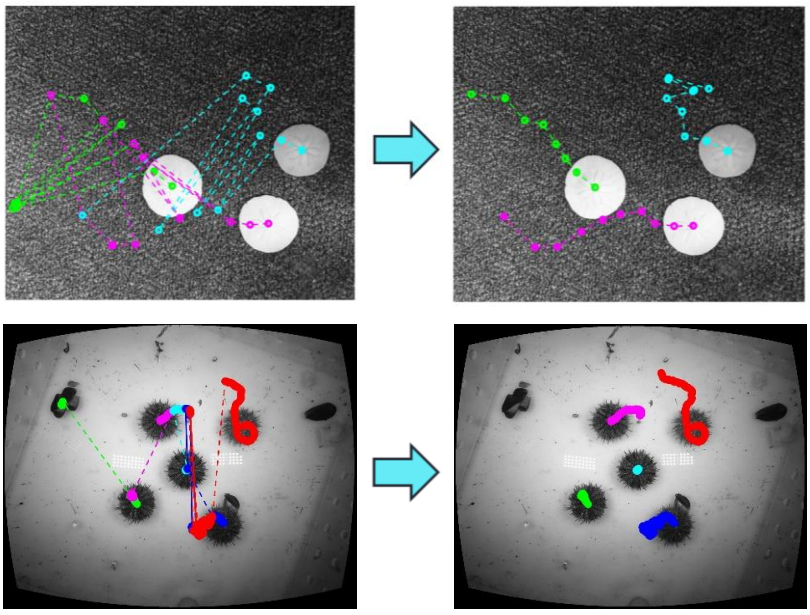


Figure 13: Left – Unorganized sand dollar (top) and sea urchin (bottom) tracks ouput by the Tracking Algorithm before sorting. Right – Organized sand dollar (top) and sea urchin (bottom) tracks output by the Sorting Algorithm.

Tank Testing and Field Deployment

Sea Urchins Tank Test

During March, multiple tank tests were conducted to observe the algorithm's ability to detect and track green sea urchins (*Strongylocentrotus droebachiensis*) in a controlled environment and collect useful data for improving the function of the tracking algorithm. The goals of the tank tests included:

- Determine ideal lighting and contrast
- Test tracking algorithm with longer time series
- Test ability of tracking algorithm to track multiple objects separately

A tank with five sea urchins was set up and kept by the Sea Urchin Aquaculture Research Team at UNH. Inside a larger white tank, the urchins were constrained to a 30 cm by 30 cm area inside a clear sided plexiglass tank. The Go Pro HERO7 camera was mounted downwards on the testing frame, with the lens of the camera submerged in the water. For each tank test, images were captured for 1-2 hours at 5 second intervals. The measured focal length of the camera was 30 cm and remained constant for each tank test. The number and position of lights within the tank was varied between tests to determine which lighting was most conducive to accurate detection of the urchins. The tank testing set up is shown in **Figure 14** below.

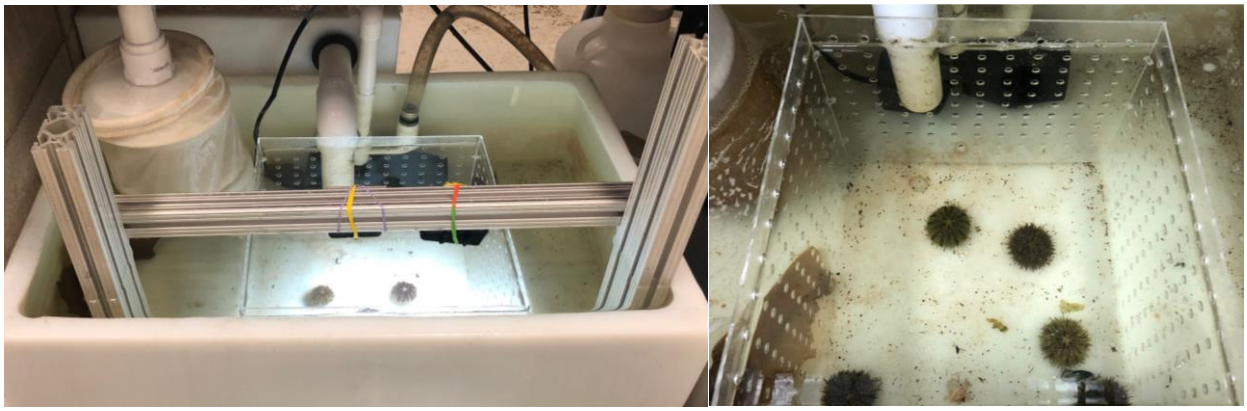


Figure 14: Left – Sea Urchin Tank set-up. The camera is attached to the testing frame facing straight down with the camera lens submerged in the water. For the tank test during which this image was captured, the use of a single light source was being tested. Right – Five *Strongylocentrotus droebachiensis* (Green Sea Urchins) were contained in a 30 cm x 30 cm plexiglass tank.

Underwater, light travels and reflects off surfaces differently than in air. Wave ripples on the surface of the water can cause shadows and glares that affect the ability of the Tracking Algorithm to accurately detect the objects within the image. Contrast between the object and the background is required for successful detection, so the lighting was adjusted to maximize contrast by varying the number and location of lights set up inside the tank.

While developing the algorithm, short time series were used to test the algorithm's ability to detect objects. During these tests, the objects were moved manually to compile a set of images of the object appearing to move that could be used to track the location of the object in the images. Tank testing with live sea urchins allowed for a longer time series to be captured because the sea urchins were self-propelled and moved on their own. Detecting multiple objects individually is difficult because it requires post-sorting of the tracks of the objects. This problem is resolved using the Sorting Algorithm to organize the tracks of detected objects for an entire time series. Since testing was done using five urchins and over 1-2 hour time spans, the urchin tank tests offered a good opportunity to test the resilience of this algorithm with multiple objects and many image frames.

Different numbers and orientations of lights were tested until the lighting provided the best contrast between the urchins and the bottom of the tank, minimizing shadows and light glares that can affect the ability to successfully detect the urchins. An example from tank tests using only one light is shown in **Figure 15** below. The ability to successfully track the urchins over the entire image plane area was hindered by shadows around the edges of the tank due to the lighting.

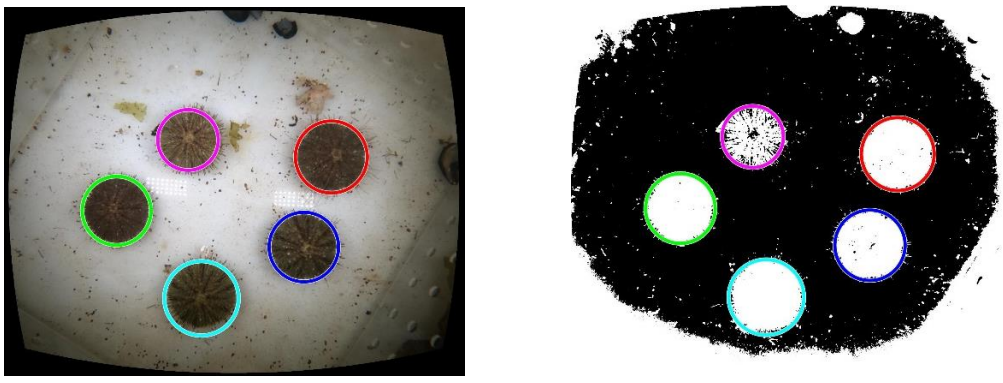


Figure 15: Left – Example of sea urchins detected using one light. The bottom of the image is darker than the top because one light did not uniformly illuminate the image area. Right – The same image converted to binary, displaying how the shadow at the bottom of the image affects the effective area of the image where the sea urchins can be detected. Sea urchins that travel into the bottom white area may not be successfully detected because they will blend into the background shading.

The lighting set-up that provided the best detection conditions in the sea urchin tanks was two lights attached to the testing frame with one on each side of the camera. After the images were collected, they were undistorted using the results of the calibration mentioned above and analyzed using the Tracking and Sorting Algorithms. The sea urchins' diameters were calculated in centimeters using the values determined during their detection with the Tracking Algorithm. Their velocities and total distances traveled were also determined in centimeters. The results of one of the tests using two lights is shown in **Figure 16** below, which shows the tracks of the sea urchins during a 1-hour time period.

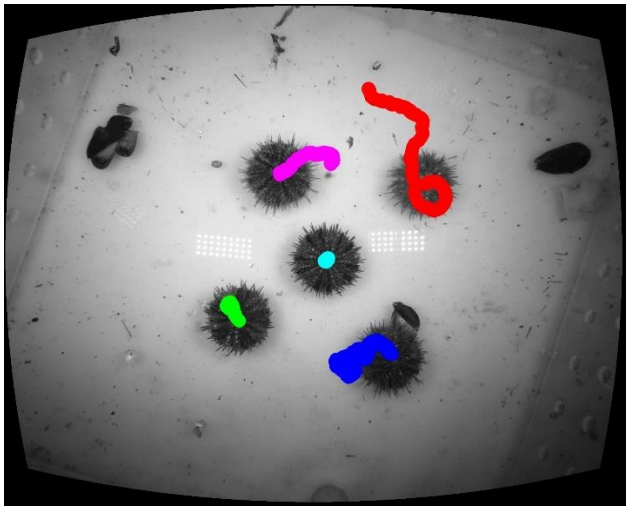


Figure 16: Sea urchin tracks during the 1-hour 2 Light testing period.

The relationship between the image pixels and their centimeter values was determined, and the motions of the sea urchins are shown in **Figure 17** in centimeters.

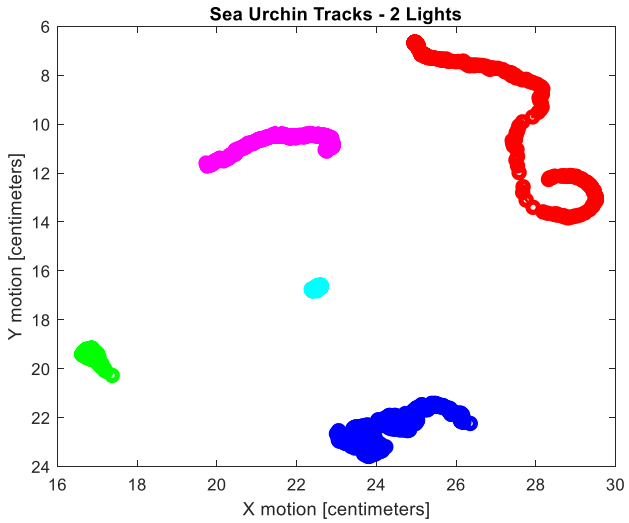


Figure 17: Sea urchin tracks with pixels converted to centimeters. The coordinates of the urchin locations correspond with their pixel locations in the collected images, converted to centimeters for easier interpretation.

The diameters of each of the urchins was also determined using the tracking algorithm. The values were first determined in pixels and then converted to centimeters using a conversion factor of 0.0109 cm/pixel determined using the known dimensions of the squares in the calibration grid. The diameter of each urchin was determined for every image frame that was captured, shown in **Figure 18.**

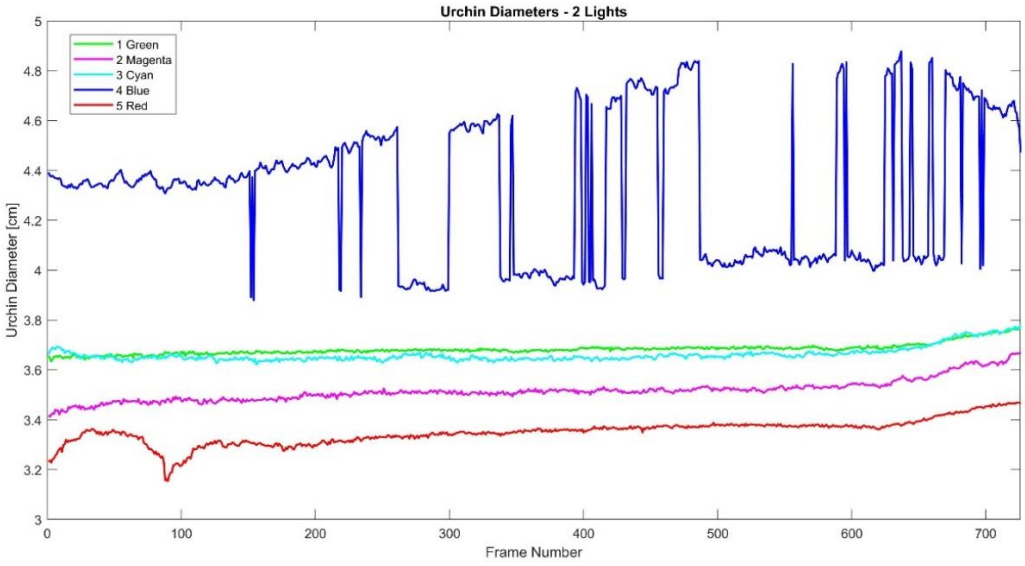


Figure 18: Sea Urchin diameters detected in each image frame captured during the 2 Lights test. For reference, frame number 700 corresponds with a time of 58.3 minutes.

As shown in **Figure 18** above, the detected sea urchin diameters remained fairly constant (the tracking algorithm consistently detected similar diameter values for each of the sea urchins) throughout the hour-long test. The blue sea urchin's detected diameter varied the most out of the five sea urchins, with a standard deviation of 0.29 cm from the mean of 4.33 cm. The detected diameters of the other urchins varied very minimally, with all standard deviations from the mean below 0.05 cm (**Table 2**).

Table 2: The means and standard deviations of the detected diameters during the 2 Lights test.

Urchin Diameters	**1 Green**	**2 Magenta**	**3 Cyan**	**4 Blue**	**5 Red**
Mean (cm)	3.68	3.52	3.66	4.33	3.35
Standard Deviation (cm)	0.02	0.04	0.03	0.29	0.05

The low standard deviations in the detected diameters of the sea urchins shows that the algorithm was able to consistently determine their diameters throughout the one-hour test period. To show the capabilities of the system, the total distances the urchins traveled, the magnitude of the distance from their start to end locations, and their velocities were determined as well (**Figure 19**).

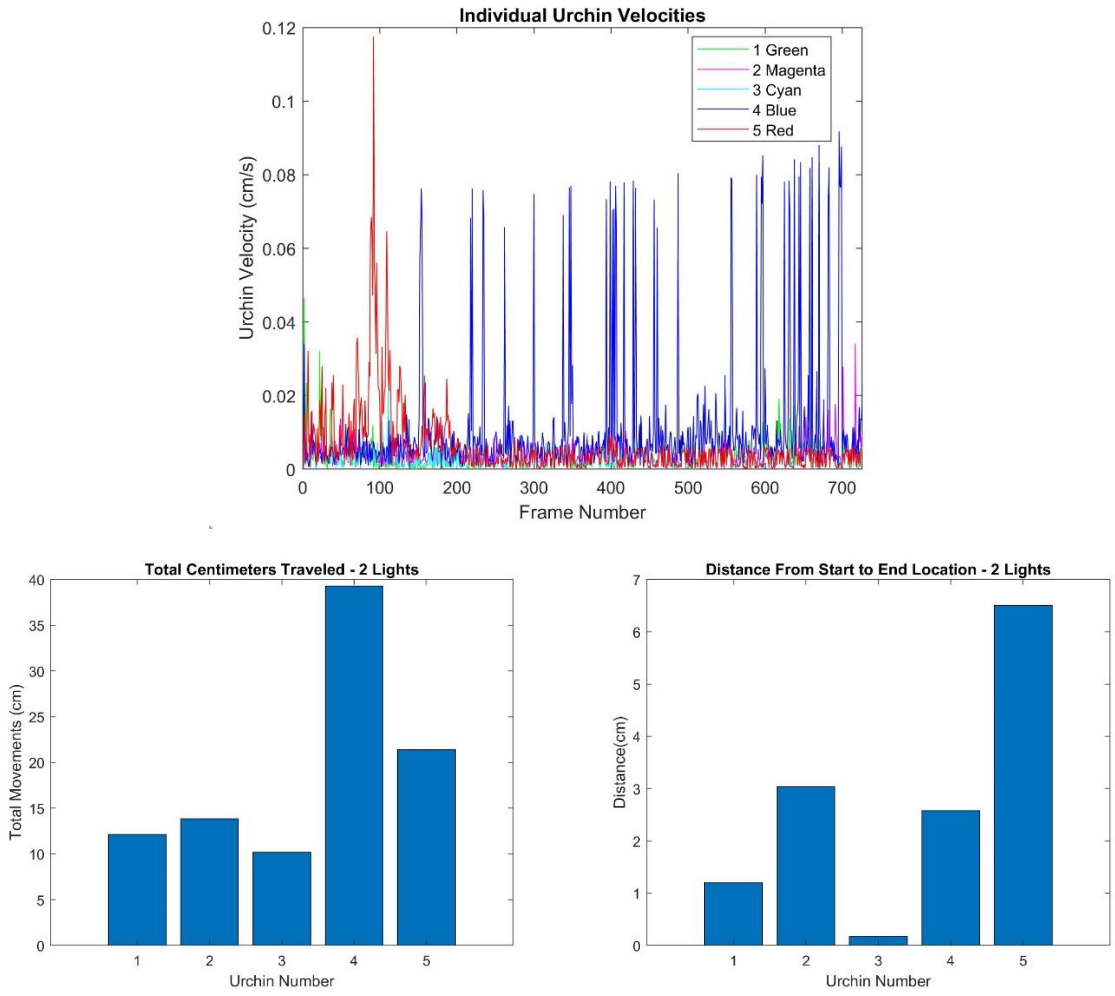


Figure 19: Top – Urchin velocities (cm/s) between frames for each individual urchin during testing. Bottom Left – Total centimeters traveled by each urchin. Bottom Right – Total distance between urchin start and end positions.

Overall, the algorithm successfully detected the paths of the sea urchins during multiple tank tests, returning consistent measurements of the urchins' diameters, as well as their paths, distances traveled, and velocities throughout each of the tests. It was determined that for tank testing, providing lighting that most uniformly illuminates objects from above will provide the best detection results. This test showed the resilience of the Tracking and Sorting Algorithms with longer data sets than previous testing and the ability to detect and track live organisms underwater. It also highlighted the capabilities of the system, such as measuring organisms and quantifying their distances traveled and velocities.

Field Testing using a Pressure Mapped Munition

During early April, two field deployments of the tracking system were completed in conjunction with the ongoing deployment of a Pressure Mapped Munition (PMM) at Wallis Sands, New Hampshire. The PMM is a research too created by researchers at the University of New Hampshire to observe how objects such as unexploded ordinances bury in sediment or roll in the nearshore zone. It is an enclosed cylinder with a 24 cm diameter lined with pressure sensors that collect data on the flows surrounding the PMM (**Figure 20**).

Figure 20: The Pressure Mapped Munition (PMM) underwater during deployment on April 14, 2019 at Wallis Sands.

The PMM was first deployed on April 10, 2019 through April 13, 2019. A pipe was jetted vertically in the nearshore zone at Wallis Sands during the morning low tide cycle on April 10 (**Figure 21**). Due to a tidal fluctuation of about 2.5 meters, the munition can be deployed in knee deep water at low tide and will be submerged in water about 2.5 meters deep at high tide [14]. An Acoustic Doppler Velocimeter (ADV) and Aquadopp attached to a deployment frame were set on the jetted pipe at a height of 54 cm. A deployment frame which was designed for the camera system was then attached to the jetted pipe above the instruments with the bottom of the deployment cuff at a height of 122 cm from the bed, with the camera positioned 52 cm from the center of the jetted pipe. The field deployment set up is shown in **Figure 22**.

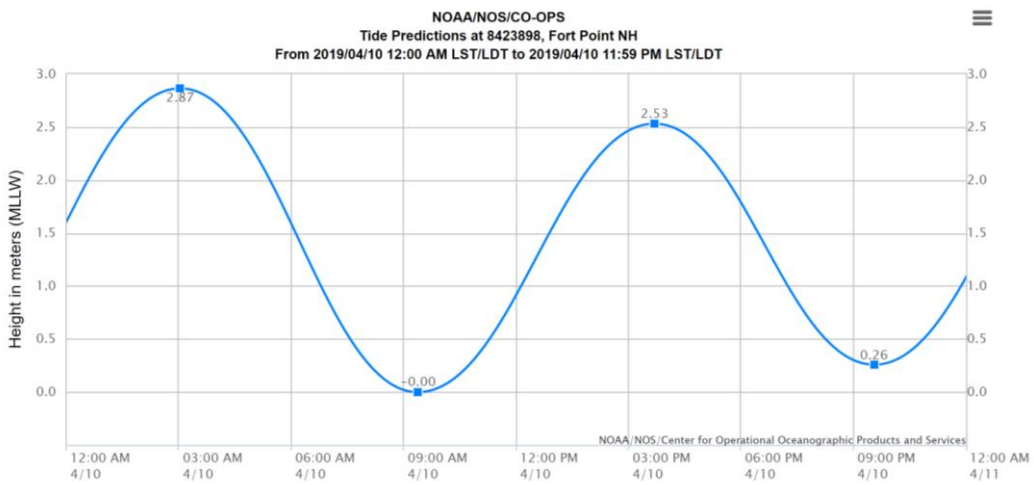


Figure 21: NOAA Tidal elevations on April 10, 2019 at Fort Point, NH [14].

Figure 22: The field deployment set up. The instrument mount with the ADV and Aquadopp attached sit on the lower part of the jetted pipe. The camera is mounted to the field deployment frame, above the instrument mount, positioned straight down.

Once all the instruments were attached to the jetted pipe, the PMM was placed below the camera on the floor of the ocean. The camera was then turned on and started capturing images at 5 second intervals. Once the camera was completely submerged as the tide got higher, the camera was able to capture images of PMM that could be analyzed using the tracking system.

Due to working in the nearshore environment, the testing area experienced varying water clarity and lighting conditions, which affected the ability to detect the PMM during this field deployment. There was an hour-long window of good water clarity that allowed for the PMM to be detected using the

tracking system. The detected munition and its track during this time period are shown in **Figure 23** below.

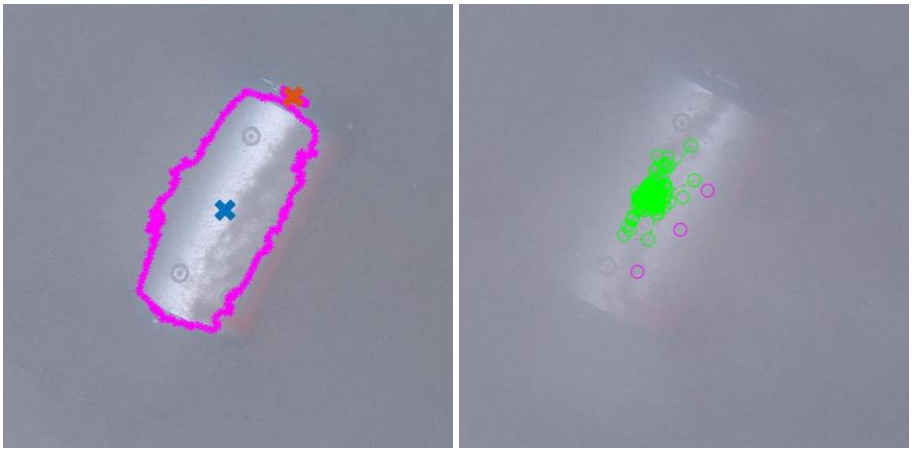


Figure 23: Left – The detected PMM during field testing on April 10, 2019. The outline of the detected area and its centroid are shown. Right – The centroid locations of the PMM over a 1 hour time period are shown in green.

During testing, the PMM showed minimal movement because it buried in the sand instead of rolling or rocking. One reason why the PMM likely buried instead of moving along the seabed at this time was because it flooded with water, which made it heavier than usual and more difficult to move. Despite capturing minimal movement, this result is still useful to help researchers understand the burial and movement of the PMM in the nearshore environment. In future deployments, the tracking system can help researchers quanitify the PMM's motions if it rolls instead of burying.

Cameras were deployed daily until April 13, 2019; however, on the second day of data collection the battery pack of the camera that was deployed flooded, likely due to manufacturing issues related to the design of these battery packs. This issue highlighted the need to create a housing for both the camera and external battery to eliminiate external connections between the camera and battery pack that can cause the waterproofing of the device to fail.

Another deployment was conducted on April 14, 2019, with the camera positioned lower in the water. Due to the closer proximity to the PMM and less wave action on this day than on April 10, clearer images were collected of the PMM; however, the camera failed to capture a long enough time series to track the munition during this deployment for unknown reasons. An example of a clearer image of the PMM from this deployment is shown in **Figure 20** above.

These initial field deployments highlighted the need for good water clarity and lighting during field deployments. In the future, improvements to the system including better lighting, a smaller focal distance to the object, and a watertight housing for the camera and external battery will result in clearer images for detecting objects and the ability to track objects and organisms over a longer period of time.

Discussion

Proof-of-concept testing for the object tracking system was successful for both tank testing with live organisms and during field testing with the PMM. During tank testing, the tracking system accurately tracked the motions of five live sea urchins over multiple one to two hour test periods, as well as determined their diameters, distances traveled, and velocities. During field testing, the ability of the tracking algorithm to detect and track a non-circular object in the nearshore environment was tested. These field tests highlighted the challenges associated with working in the dynamic nearshore environment, with water clarity being the biggest factor affecting object detection. Despite the changing conditions of the nearshore environment, the PMM was successfully detected and tracked during a period of good water clarity. Despite minimal horizontal motion of the PMM during testing, researchers observing the motions of the PPM in the future will benefit from the system's ability to return quantifiable data on the PMM's motions when it rolls and translates horizontally due to strong flows and waves.

In the future, some improvements that would increase the variety of applications for the system include extending the battery life of the camera, utilizing stronger, longer lasting lights, and applying stereophotogrammetry techniques. During the field deployments conducted with the current system setup, the duration and continuity of data collection was limited by the combined battery length of the camera and battery pack – a total of 10 hours. The current lighting for the system – which was most useful during tank testing – only lasted 6 hours on low power mode, so it was not sufficient to light the field deployment area throughout the night. This limited the duration and continuity of the time series that were collected because the cameras could only be deployed at low tide and could only capture the track of the PMM during the daylight, causing gaps in the data at night until the next low tide cycle when the camera could be replaced with a fully charged camera.

To increase the battery life and durability of the system, a waterproof container to house both the camera and a longer lasting battery pack could built. Not only would this increase the duration for

which the system could be deployed without requiring man-power to replace the camera, but it could also increase the depth to which the system is waterproof depending on the strength of the housing. The battery packs that were chosen for this project were sufficient for collecting data during the day when it is light out, which was the original goal for the project; however, they were not as resilient and waterproof as rated by the manufacturer. Therefore, employing a waterproof housing that encases both the camera and battery pack would be a more reliable approach to extending the battery life of the cameras. The battery life of the current lights could also be increased by attaching external batteries to the light and placing both the light and battery inside a housing, or by integrating the light into the housing for the camera. This would allow the system to record continuous data sequences throughout the night and day.

One useful addition that could be made to a camera housing is the use of a dome port, which would eliminate the need to correct for distortion using a calibration because light from an object underwater can travel through the dome port unaffected by refraction [12]. An example of a dome port and light traveling through a dome port from water are shown in **Figure 24**.

Figure 24: Left – The difference between a dome port and a flat port [12]. Right – Dome ports are unaffected by refraction and can remove distortion associated with capturing images underwater [12].

Stereophotogrammetry is an image processing technique that utilizes a system of two cameras to estimate the three-dimensional coordinates of an object [7]. By positioning the cameras in different locations with overlapping image planes and capturing images simultaneously, the three-dimensional coordinates of the image area can be determined. This project investigated the use of stereo imagery, but decided on a single camera for initial design to simplify object detection; however, in the future the algorithm could be updated and applied for a stereo camera system.

Because the ability of an object to be detected is dependent on the camera's ability to clearly see the object and is dependent on high levels of contrast between the object and the background, increasing the contrast of detected objects will aid their detectability. In the future when testing the PMM, considerations should be made to paint the device a color that will stand out well on the ocean floor. Currently, the metal of the PMM is highly reflective and this affects the ability of the algorithm to detect the PMM depending on the lighting. In addition, detection spots on the PMM, such as colored circles on the lateral ends of the PMM, would aid in the algorithm's ability to quantify rotational movements, as well as translational and rolling motions.

This system has a variety of applications for observing the motions of both organisms and submerged objects. One application that the system design considered was observing the motions of sand dollars. For example, this system can be used to observe the motions and positioning of sand dollars as they are affected by currents, as well as observe their reactions to food and behaviors during spawning. Stronger current velocities and tidal strength are associated with sediment that is more sorted and sand dollars are typically positioned in a more angular direction as a result [6].

This system can also be used to observe the transport of plastic marine debris affected by waves and currents. This application can use marbles and small objects in the nearshore ocean environment to observe their expected oscillatory motions in phase with the passing waves. The motions of these objects can give insight into the global impact and bigger picture of how small plastics are transported in the ocean by increasing the understanding of the affect of waves and currents on their transport.

In addition, future deployments of the tracking system aim to observe the motions of the PMM while it is rolling to gain insight on the wave and current conditions that cause unexploded ordinances in the surf zone to bury in sediment or move. This application is important to the Department of Defense in predicting the location of mines and other objects that are used for testing.

Conclusion

A video tracking system was developed to track the motions of live organisms and submerged objects in the ocean. The physical system assembly consisted of a camera, lighting, battery pack, and mounting frames, while two algorithms processed the data collected using this physical setup. The Tracking Algorithm was developed to detect and track these objects and their paths along the sea floor, while the Sorting Algorithm was used to clean up and organize the tracks determined using the Tracking

Algorithm when tracking multiple objects in an image frame. The system was successfully used to track both live objects and non-circular objects in tanks and the field to show proof-of-concept and highlight the systems applications and abilities. In the future, the system's ability to track objects can be improved with adjusted lighting and contrast. The system will be updated with longer-lasting battery packs that will aid in collecting longer time series during field deployments and will employ the use of a watertight housing containing both the camera and external battery to increase the durability of the system. Future deployments look to test more applications of the tracking system including tracking sand dollars, the PMM when rolling, and the transport of plastic marine debris to offer insight on the motions of these organisms and objects. A variety of additional applications for this system exist, and future improvements look to expand the system's capabilities to even more applications.

Acknowledgements

This project was made possible with funding from the TECH 797 Ocean Projects Course and New Hampshire Sea Grant. Thank you to Abigail Leclerc, Angela Sarni, Savannah Devoe, Dr. Larry Harris, Tiffany Winter, Jon Hunt, and Paul Lavoie for their valuable contributions and assistance.

References

[1] “Detect and Measure Circular Objects in an Image.” *MATLAB Documentation*, MathWorks, www.mathworks.com/help/images/detect-and-measure-circular-objects-in-an-image.html.

[2] “GoPro HERO7 Black - Waterproof Digital Action Camera with Touch Screen 4K HD Video 12MP Photos Live Streaming Stabilization.” *Amazon*, Amazon, www.amazon.com/GoPro-HERO7-Black-Waterproof-Streaming-Stabilization/dp/B07GDGZCCH.

[3] “Gulf of Maine (GOMOFS).” *Operational Forecast System Developmental - Station Plot Time Series*, NOAA, tidesandcurrents.noaa.gov/ofs/dev/ofs_station.shtml?stname=Jeffreys%2BLedge&ofs=gom&stnid=44098&subdomain=0.

[4] “HERO7 Field of View (FOV) Information.” *GoPro*, gopro.com/help/articles/question_answer/hero7-field-of-view-fov-information.

[5] Knight, David W. “The Angle of Coverage (or Field of View) of a Lens.” *Cameras Underwater*, 2012.

[6] Merrill, R. J., & Hobson, E. S. (1970). Field observations of Dendraster excentricus, a sand dollar of western North America. *American Midland Naturalist*, 595-624.

[7] “Photogrammetry.” *Wikipedia*, Wikimedia Foundation, 1 Apr. 2019, en.wikipedia.org/wiki/Photogrammetry.

[8] “Re-Fuel - 9 Hour ActionPack Extended Battery for GoPro HERO7, HERO6, HERO5 & HERO.” *Amazon*, Amazon, www.amazon.com/Re-Fuel-ActionPack-Extended-Battery-GoPro/.

[9] “Refraction.” *Merriam-Webster*, Merriam-Webster, www.merriam-webster.com/dictionary/refraction.

[10] “Refractometers and Salinity Measurement.” *REEFEDITION*, 21 Dec. 2015, www.reefedition.com/refractometers-salinity-measurement/.

[11] *Single Camera Calibrator App*. MATLAB.

[12] Somerville, Jeremy. “Understanding Flat Port and Dome Port Theory.” *Oceanity*, 18 Mar. 2017, oceanity.com.au/learn/understanding-flat-port-and-dome-port-theory/.

[13] “Suptig Diving Light High Power Dimmable Waterproof LED Video Light Fill Night Light Diving Underwater Light Waterproof 147ft(45m) for Gopro Hero 6/5/5S/4/4S/3+/2/SJCAM/YI Action Cameras.” *Amazon*, Amazon, www.amazon.com/Suptig-Dimmable-Waterproof-Underwater-Cameras/.

[14] “Tides/Water Levels Fort Point, NH.” *Tide Predictions - NOAA Tides & Currents*, NOAA, tidesandcurrents.noaa.gov/noaatidepredictions.html?id=8423898&units=metric&bdate=20190410&edate=20190410&timezone=LST%2FLDT&clock=12hour&datum=MLLW&interval=hilo&action=dailychart.

Appendix A – Tracking Algorithm Example

```
%% Tracking Algorithm - Sea Urchin Example
clear all; close all;
%% Load Images

imageNames = dir('*.jpg');
imageNames = {imageNames.name}';

numObjects = 5;
num_images = length(imageNames);

%% Load Calibration

load('cameraParams.mat')

%% Initialize Variables
XCENT = zeros(num_images,numObjects);
YCENT = zeros(num_images,numObjects);

radius = zeros(num_images,numObjects);


%% Determine Crop Area

img = imread(imageNames{1});
% figure()
% imshow(img)
% [imgcrop,croparea] = imcrop(img);

%% Detect Objects in each image to determine track object takes over
time
for ii = 501:num_images
    % Load & crop Images
    RGB = imread(imageNames{ii});
    % RGB = imcrop(RGB,croparea);
 % Undistort Image
    RGB = undistortImage(RGB,cameraParams);
    GRAY = rgb2gray(RGB);
 % *** Threshold may beed to changed if graythresh does not provide
 % correct contrast
    threshold = 0.3; % graythresh(GRAY);
    BW = imbinarize(GRAY,threshold);

    % figure()
    % imshow(BW)
    % str2 = sprintf('Image %d BW', ii);
    % title(str2)

 % To Blur Edges of Rough Objects:
    % windowSize = 51;
```

```
    % kernel = ones(windowSize) / windowSize ^ 2;
    % blurryImage = conv2(single(BW), kernel, 'same');
    % BW = blurryImage > 0.5; % Rethreshold
    % figure()
    % imshow(BW)

 % UNCOMMENT BELOW IF: desired object is in black instead of white
    BW = imcomplement(BW);
    imshow(BW)

 % Calculate region props, find centroids, diameters, radius
    stats = regionprops('table',BW,'Centroid',...
        'MajorAxisLength','MinorAxisLength');
    centers = stats.Centroid;
    diameters = mean([stats.MajorAxisLength stats.MinorAxisLength],2);
    radii = diameters/2;

 % Insert upper and lower bounds of expected radii in pixels
    dum = find(radii > 140 & radii < 300); % Filter out small and
large circles
    cent = centers(dum,:);
    diam = diameters(dum);
    rad = radii(dum);

 % Find all circles in given radius range
    xcenter = cent(:,1);
    ycenter = cent(:,2);

    hold on
    for jj = 1:length(dum)
     % Plot detected circular objects for each image:
        % colors = {'g','m','c','b','r','y','k','w'};
        % viscircles(cent(jj,:),rad(jj),'Color',colors{jj})

     % Create Matrices w Radius Values and X,Y Coordinates
        radius(ii,jj) = rad(jj); % radius for each sand dollar
        XCENT(ii,jj) = xcenter(jj); % x centroid location for each
sand dollar
        YCENT(ii,jj) = ycenter(jj); % y centroid location for each
sand dollar
    end

end

%% PLOT Track
figure()
image = imread(imageNames{1});
image = undistortImage(image,cameraParams);
GRAY = rgb2gray(image);
%GRAY = rgb2gray(imgcrop);
% threshold = graythresh(GRAY);
% BW = imbinarize(GRAY,threshold);
```

```
% BW_comp = imcomplement(BW);
imshow(GRAY)
hold on
for kk = 1:numObjects
    colors = {'g','m','c','b','r','y','k','w'};

plot(XCENT(:,kk),YCENT(:,kk),'o','Linewidth',3,'Color',colors{kk});
    hold on
    plot(XCENT(:,kk),YCENT(:,kk),'--
','Linewidth',1.5,'Color',colors{kk});
%     xlim([0 4000])
%     ylim([0 3000])
    set(gca, 'YDir','reverse')
    title('Sea Urchin Tracks - Unsorted')
end
```

Appendix B – Sorting Algorithm Example

```
%% SORTING ALGORITHM + PLOT URCHIN DATA
%%
dum = size(XCENT);
len = dum(1);
wid = dum(2);

XYcell = cell(len,wid);
newCoords = cell(len,wid);

RAD = cell(len,wid);
newRadius = cell(len,wid);

% Create a cell array with the coordinates from Xcent[] and Ycent[]
for ii = 1:len
    for jj = 1:wid
        % Remove Zeros - convert to NaN
        if XCENT(ii,jj) <=0  && YCENT(ii,jj) <= 0
            XCENT(ii,jj) = nan;
            YCENT(ii,jj) = nan;
        end
        XYcell(ii,jj) = {[XCENT(ii,jj), YCENT(ii,jj)]};
    end
end

% Create a cell array with the radii from Xcent, Ycent
for ii = 1:len
    for jj = 1:wid
    % Remove Zeros - convert to NaN
        if radius(ii,jj) <=0
            radius(ii,jj) = nan;
        end
        RAD(ii,jj) = {[radius(ii,jj)]};
    end
end

% Create an empty cell array with the same dimensions as XYcell
for ii = 1:len
    for jj = 1:wid
        newCoords(ii,jj) = {[0 0]};
    end
end

% Create an empty cell array with the same dimensions as RAD
for ii = 1:len
    for jj = 1:wid
        newRadius(ii,jj) = {[0 0]};
    end
end
```

```
% Copy of the first row of XYcell into newCoords
for jj = 1:wid
    newCoords(1,jj) = {[XCENT(1,jj), YCENT(1,jj)]};
end

% Copy of the first row of RAD into newRadius
for jj = 1:wid
    newRadius(1,jj) = {[radius(1,jj)]};
end


% For each row in XYcell (starting at the 2nd row)...
for ii = 2:len
    % For each column in XYcell...
    for jj = 1:wid

        % Get the current element in XYcell
        x = XYcell{ii,jj}(1);
        y = XYcell{ii,jj}(2);
        rad_cur = RAD{ii,jj};

        % If the XYcell has a value,
        if (x ~= 0 && y ~= 0)

            smallestValue = nan;
            closestY = nan;

            % Loop through each column in row above in newCoords
            for kk = 1:wid

                % Values of the element in the row above (in
newCoords)
                xAbove = newCoords{ii - 1,kk}(1);
                yAbove = newCoords{ii - 1,kk}(2);


                % Values of the current closest match
                xBelow = newCoords{ii,kk}(1);
                yBelow = newCoords{ii,kk}(2);

                % Repeat until all are used...
                if (xAbove ~= 0 && yAbove ~=0)

                    % Potentially closer difference with the new x,y
values
                    thisDif = abs(x - xAbove) + abs(y - yAbove);

                    % Current difference between the cells in
newCoords
                    currentDif = abs(xAbove - xBelow) + abs(yAbove -
yBelow);
```

```
                    % Replace radius if new match is closer
                    if (thisDif < currentDif)
                        newCoords(ii,kk) = {[x y]};
                        newRadius(ii,kk) = {[rad_cur]};
                    end
                end
            end
        end
    end

    fprintf("\n\nRow " + ii + "\n");
    % Loop through each value in that same row in XYcell

    for ll = 1:wid

        originalX = XYcell{ii,ll}(1);
        originalY = XYcell{ii,ll}(2);
        originalRad = RAD{ii,ll};

        if (originalX ~=0 && originalY ~=0)
            isUsed = false;

            for mm = 1:wid
                potentialDuplicateX = newCoords{ii,mm}(1);
                potentialDuplicateY = newCoords{ii,mm}(2);

                % If they are not used, still put it in the array.
                if (~isUsed)
                    % And it matches an entry, skip it
                    if(potentialDuplicateX == originalX &&
potentialDuplicateY == originalY)
                        isUsed = true;
                        fprintf("(" + ii + ", " + ll + ")   Potential
Duplicate " + potentialDuplicateX + "\n");

                    % And you've reached the end of the list, add it
                    elseif(potentialDuplicateX == 0 &&
potentialDuplicateY == 0)
                        isUsed = true;
                        newCoords(ii,mm) = {[originalX originalY]};
                        newRadius(ii,mm) = {[originalRad]};
                    end
                end
            end
        end
    end


% Calculate difference between each box in a row, if values
% are equal, find which is closest to value in box above
% and set farther one to NaN - if two have same centroid,
% this means there is a duplicate because one left the view
```

```
    % Loop through each value in that same row in XYcell
    for mm = 1:wid
        currentX = newCoords{ii,mm}(1);
        currentY = newCoords{ii,mm}(2);
        currentRad = newRadius{ii,mm};

        % Find values of all other coordinates in row
        for nn = 1:wid

            % Do not look at current cell for duplicate of current
cell
            if mm ~= nn
                potentialDuplicateX = newCoords{ii,nn}(1);
                potentialDuplicateY = newCoords{ii,nn}(2);

                % If same coordinate values exist in same row, find
which is
                % closest to the value above it
                if currentX == potentialDuplicateX && currentY ==
potentialDuplicateY
                    xAboveCurrent = newCoords{ii - 1,mm}(1);
                    yAboveCurrent = newCoords{ii - 1,mm}(2);

                    xAboveDuplicate = newCoords{ii - 1,nn}(1);
                    yAboveDuplicate = newCoords{ii - 1,nn}(2);

                    % Find difference between current column and
row/col above
                    % and difference between duplicate column and
row/col above
                    differenceCurrent = abs(xAboveCurrent - currentX)
+ abs(yAboveCurrent - currentY);
                    differenceDuplicate = abs(xAboveDuplicate -
potentialDuplicateX) + abs(yAboveDuplicate - potentialDuplicateY);

                    % Find which difference is smaller. The column
with the
                    % larger difference from the column/row above will
be set
                    % to nan
                    if differenceCurrent < differenceDuplicate
                        newCoords{ii,nn}(1) = nan;
                        newCoords{ii,nn}(2) = nan;
                        newRadius{ii,nn} = nan;
                    else
                        newCoords{ii,mm}(1) = nan;
                        newCoords{ii,mm}(2) = nan;
                        newRadius{ii,mm} = nan;
                    end
                end
            end
```

```
        end
    end

end


%% Convert Pixels to Distances - Tank Calibration

load('cameraParams.mat');
% Load in raw (distorted) calibration image
calImage = imread('G0161897.jpg');
% Undistort Image
calUndistort = undistortImage(calImage,cameraParams);

% figure()
% imshow(calUndistort)
% manually determine distance from bottom corner of square to diagonal
% corner of square in pixels
X1 = 1973; % bottom left
Y1 = 1669;
X2 = 2220; % top right
Y2 = 1458;
X0 = 2204; % bottom right
Y0 = 1686;
pixels_diag = sqrt((X2-X1)^2 + (Y2-Y1)^2);
centimeters_diag = sqrt((2.5^2) + (2.5^2));

centimeters_per_pixel = centimeters_diag/pixels_diag;
diameter_centimeters = 2*centimeters_per_pixel*(cell2mat(newRadius));

% Plot variation in detected urchin diameter
figure()
colors = {'g','m','c','b','r','y','k','w'};
plot(diameter_centimeters(:,1),'g','linewidth',1.5)
hold on
plot(diameter_centimeters(:,2),'m','linewidth',1.5)
plot(diameter_centimeters(:,3),'c','linewidth',1.5)
plot(diameter_centimeters(:,4),'b','linewidth',1.5)
plot(diameter_centimeters(:,5),'r','linewidth',1.5)
xlim([0 len])
legend('1 Green','2 Magenta','3 Cyan','4 Blue','5 Red')
xlabel('Frame Number')
ylabel('Urchin Diameter [cm]')
title('Urchin Diameters - 2 Lights')

% Calculate Mean and STD of Diameters
meanDiam = zeros(1,wid);
stdDiam = zeros(1,wid);
for ff = 1:numObjects
    meanDiam(1,ff) = mean(diameter_centimeters(:,ff));
    stdDiam(1,ff) = std(diameter_centimeters(:,ff));
end
```

```
%% Plot Track

for aa = 1:len
    for bb = 1:numObjects
        X(aa,bb) = newCoords{aa,bb}(1);
        Y(aa,bb) = newCoords{aa,bb}(2);
    end
end

X_dist = X*centimeters_per_pixel;
Y_dist = Y*centimeters_per_pixel;

figure()
image = imread(imageNames{1});
image = undistortImage(image,cameraParams);
GRAY = rgb2gray(image);
imshow(GRAY)

% figure()
% imshow(imgcrop)
hold on

for kk = 1:numObjects
    colors = {'g','m','c','b','r','y','k','w'};
    plot(X(:,kk),Y(:,kk),'o','Linewidth',3,'Color',colors{kk});
    hold on
    plot(X(:,kk),Y(:,kk),'--','Linewidth',1.5,'Color',colors{kk});
    set(gca, 'YDir','reverse')
end

% Plot Sea Urchin Tracks
figure()
for kk = 1:numObjects
    colors = {'g','m','c','b','r','y','k','w'};

plot(X_dist(:,kk),Y_dist(:,kk),'o','Linewidth',3,'Color',colors{kk});
    hold on
    plot(X_dist(:,kk),Y_dist(:,kk),'--
','Linewidth',1.5,'Color',colors{kk});
    xlim([335*centimeters_per_pixel 3666*centimeters_per_pixel])
    ylim([160*centimeters_per_pixel 2857*centimeters_per_pixel])
    set(gca, 'YDir','reverse')
    xlabel('X motion [centimeters]')
    ylabel('Y motion [centimeters]')
    title('Sea Urchin Tracks - 2 Lights')
end


%% Urchin Velocities
```

```
difference_distance_X = zeros(len,wid);
difference_distance_Y = zeros(len,wid);

% Calculate Distance Traveled between each image frame captured
for pp = 1:wid
    for uu = 2:len
        difference_distance_X(uu,pp) = X_dist(uu,pp)-X_dist((uu-
1),pp);
        difference_distance_Y(uu,pp) = Y_dist(uu,pp)-Y_dist((uu-
1),pp);
        distance(uu,pp) = sqrt((difference_distance_X(uu,pp).^2) +
(difference_distance_Y(uu,pp).^2));
    end
end


% Plot Velocity
time = 5; % seconds between images
velocity = distance/time; % cm/s

figure()
plot(velocity(:,1),'g')
hold on
plot(velocity(:,2),'m')
plot(velocity(:,3),'c')
plot(velocity(:,4),'b')
plot(velocity(:,5),'r')
xlim([0 len])
xlabel('Frame Number')
ylabel('Urchin Velocity (cm/s)')
title('Individual Urchin Velocities')
legend('1 Green','2 Magenta','3 Cyan','4 Blue','5 Red')

% Plot Total Distance Traveled
totalDistance = zeros(1,wid);
for rr = 1:wid
    totalDistance(1,rr) = sum(distance(:,rr));
end

figure()
bar(totalDistance(1,1:5))
xlabel('Urchin Number')
ylabel('Total Movements (cm)')
title('Total Centimeters Traveled - 2 Lights')

% Plot Distance From Start Location to End Location
distFromStart = zeros(1,wid);
for ee = 1:wid
    distFromStart_X(1,ee) = X_dist(len,ee)-X_dist(1,ee);
    distFromStart_Y(1,ee) = Y_dist(len,ee)-Y_dist(1,ee);
    distFromStart(1,ee) = sqrt((distFromStart_X(1,ee).^2) +
(distFromStart_Y(1,ee).^2));
```

```
end

figure()
bar(distFromStart(1,1:numObjects))
xlabel('Urchin Number')
ylabel('Distance(cm)')
title('Distance From Start to End Location - 2 Lights')


%% Plot Circles and Radii - Use to make time lapse

for aa = 1:len
    for bb = 1:wid
        X(aa,bb) = newCoords{aa,bb}(1);
        Y(aa,bb) = newCoords{aa,bb}(2);
    end
end

frameNum = 1;
figure()
% Start Image
GRAY1 = rgb2gray(imgcrop);
% End Image
RGB = imread(imageNames{frameNum});
RGB = undistortImage(RGB,cameraParams);
% RGB = imcrop(RGB,croparea);
GRAY2 = rgb2gray(RGB);
imshow(GRAY2)
hold on

radiusmatrix = cell2mat(newRadius);
for mm = 1:frameNum
    for jj = 1:numObjects
        centerX = X(mm,:)';
        centerY = Y(mm,:)';
        centermatrix = [centerX,centerY];
        colors = {'g','m','c','b','r','y','k','w'};

viscircles(centermatrix(jj,:),radiusmatrix(mm,jj),'Color',colors{jj})
        hold on
    end
end
```

Appendix A – Tracking Algorithm Example

```
%% Tracking Algorithm - Sea Urchin Example
clear all; close all;
%% Load Images

imageNames = dir('*.jpg');
imageNames = {imageNames.name}';

numObjects = 5;
num_images = length(imageNames);

%% Load Calibration

load('cameraParams.mat')

%% Initialize Variables
XCENT = zeros(num_images,numObjects);
YCENT = zeros(num_images,numObjects);

radius = zeros(num_images,numObjects);


%% Determine Crop Area

img = imread(imageNames{1});
% figure()
% imshow(img)
% [imgcrop,croparea] = imcrop(img);

%% Detect Objects in each image to determine track object takes over
time
for ii = 501:num_images
    % Load & crop Images
    RGB = imread(imageNames{ii});
    % RGB = imcrop(RGB,croparea);
 % Undistort Image
    RGB = undistortImage(RGB,cameraParams);
    GRAY = rgb2gray(RGB);
 % *** Threshold may beed to changed if graythresh does not provide
 % correct contrast
    threshold = 0.3; % graythresh(GRAY);
    BW = imbinarize(GRAY,threshold);

    % figure()
    % imshow(BW)
    % str2 = sprintf('Image %d BW', ii);
    % title(str2)

 % To Blur Edges of Rough Objects:
    % windowSize = 51;
```

```
    % kernel = ones(windowSize) / windowSize ^ 2;
    % blurryImage = conv2(single(BW), kernel, 'same');
    % BW = blurryImage > 0.5; % Rethreshold
    % figure()
    % imshow(BW)

 % UNCOMMENT BELOW IF: desired object is in black instead of white
    BW = imcomplement(BW);
    imshow(BW)

 % Calculate region props, find centroids, diameters, radius
    stats = regionprops('table',BW,'Centroid',...
        'MajorAxisLength','MinorAxisLength');
    centers = stats.Centroid;
    diameters = mean([stats.MajorAxisLength stats.MinorAxisLength],2);
    radii = diameters/2;

 % Insert upper and lower bounds of expected radii in pixels
    dum = find(radii > 140 & radii < 300); % Filter out small and
large circles
    cent = centers(dum,:);
    diam = diameters(dum);
    rad = radii(dum);

 % Find all circles in given radius range
    xcenter = cent(:,1);
    ycenter = cent(:,2);

    hold on
    for jj = 1:length(dum)
     % Plot detected circular objects for each image:
        % colors = {'g','m','c','b','r','y','k','w'};
        % viscircles(cent(jj,:),rad(jj),'Color',colors{jj})

     % Create Matrices w Radius Values and X,Y Coordinates
        radius(ii,jj) = rad(jj); % radius for each sand dollar
        XCENT(ii,jj) = xcenter(jj); % x centroid location for each
sand dollar
        YCENT(ii,jj) = ycenter(jj); % y centroid location for each
sand dollar
    end

end

%% PLOT Track
figure()
image = imread(imageNames{1});
image = undistortImage(image,cameraParams);
GRAY = rgb2gray(image);
%GRAY = rgb2gray(imgcrop);
% threshold = graythresh(GRAY);
% BW = imbinarize(GRAY,threshold);
```

```
% BW_comp = imcomplement(BW);
imshow(GRAY)
hold on
for kk = 1:numObjects
    colors = {'g','m','c','b','r','y','k','w'};

plot(XCENT(:,kk),YCENT(:,kk),'o','Linewidth',3,'Color',colors{kk});
    hold on
    plot(XCENT(:,kk),YCENT(:,kk),'--
','Linewidth',1.5,'Color',colors{kk});
%     xlim([0 4000])
%     ylim([0 3000])
    set(gca, 'YDir','reverse')
    title('Sea Urchin Tracks - Unsorted')
end
```

Appendix B – Sorting Algorithm Example

```
%% SORTING ALGORITHM + PLOT URCHIN DATA
%%
dum = size(XCENT);
len = dum(1);
wid = dum(2);

XYcell = cell(len,wid);
newCoords = cell(len,wid);

RAD = cell(len,wid);
newRadius = cell(len,wid);

% Create a cell array with the coordinates from Xcent[] and Ycent[]
for ii = 1:len
    for jj = 1:wid
        % Remove Zeros - convert to NaN
        if XCENT(ii,jj) <=0  && YCENT(ii,jj) <= 0
            XCENT(ii,jj) = nan;
            YCENT(ii,jj) = nan;
        end
        XYcell(ii,jj) = {[XCENT(ii,jj), YCENT(ii,jj)]};
    end
end

% Create a cell array with the radii from Xcent, Ycent
for ii = 1:len
    for jj = 1:wid
    % Remove Zeros - convert to NaN
        if radius(ii,jj) <=0
            radius(ii,jj) = nan;
        end
        RAD(ii,jj) = {[radius(ii,jj)]};
    end
end

% Create an empty cell array with the same dimensions as XYcell
for ii = 1:len
    for jj = 1:wid
        newCoords(ii,jj) = {[0 0]};
    end
end

% Create an empty cell array with the same dimensions as RAD
for ii = 1:len
    for jj = 1:wid
        newRadius(ii,jj) = {[0 0]};
    end
end
```

```
% Copy of the first row of XYcell into newCoords
for jj = 1:wid
    newCoords(1,jj) = {[XCENT(1,jj), YCENT(1,jj)]};
end

% Copy of the first row of RAD into newRadius
for jj = 1:wid
    newRadius(1,jj) = {[radius(1,jj)]};
end


% For each row in XYcell (starting at the 2nd row)...
for ii = 2:len
    % For each column in XYcell...
    for jj = 1:wid

        % Get the current element in XYcell
        x = XYcell{ii,jj}(1);
        y = XYcell{ii,jj}(2);
        rad_cur = RAD{ii,jj};

        % If the XYcell has a value,
        if (x ~= 0 && y ~= 0)

            smallestValue = nan;
            closestY = nan;

            % Loop through each column in row above in newCoords
            for kk = 1:wid

                % Values of the element in the row above (in
newCoords)
                xAbove = newCoords{ii - 1,kk}(1);
                yAbove = newCoords{ii - 1,kk}(2);


                % Values of the current closest match
                xBelow = newCoords{ii,kk}(1);
                yBelow = newCoords{ii,kk}(2);

                % Repeat until all are used...
                if (xAbove ~= 0 && yAbove ~=0)

                    % Potentially closer difference with the new x,y
values
                    thisDif = abs(x - xAbove) + abs(y - yAbove);

                    % Current difference between the cells in
newCoords
                    currentDif = abs(xAbove - xBelow) + abs(yAbove -
yBelow);
```

```
                    % Replace radius if new match is closer
                    if (thisDif < currentDif)
                        newCoords(ii,kk) = {[x y]};
                        newRadius(ii,kk) = {[rad_cur]};
                    end
                end
            end
        end
    end

    fprintf("\n\nRow " + ii + "\n");
    % Loop through each value in that same row in XYcell

    for ll = 1:wid

        originalX = XYcell{ii,ll}(1);
        originalY = XYcell{ii,ll}(2);
        originalRad = RAD{ii,ll};

        if (originalX ~=0 && originalY ~=0)
            isUsed = false;

            for mm = 1:wid
                potentialDuplicateX = newCoords{ii,mm}(1);
                potentialDuplicateY = newCoords{ii,mm}(2);

                % If they are not used, still put it in the array.
                if (~isUsed)
                    % And it matches an entry, skip it
                    if(potentialDuplicateX == originalX &&
potentialDuplicateY == originalY)
                        isUsed = true;
                        fprintf("(" + ii + ", " + ll + ")   Potential
Duplicate " + potentialDuplicateX + "\n");

                    % And you've reached the end of the list, add it
                    elseif(potentialDuplicateX == 0 &&
potentialDuplicateY == 0)
                        isUsed = true;
                        newCoords(ii,mm) = {[originalX originalY]};
                        newRadius(ii,mm) = {[originalRad]};
                    end
                end
            end
        end
    end


% Calculate difference between each box in a row, if values
% are equal, find which is closest to value in box above
% and set farther one to NaN - if two have same centroid,
% this means there is a duplicate because one left the view
```

```
    % Loop through each value in that same row in XYcell
    for mm = 1:wid
        currentX = newCoords{ii,mm}(1);
        currentY = newCoords{ii,mm}(2);
        currentRad = newRadius{ii,mm};

        % Find values of all other coordinates in row
        for nn = 1:wid

            % Do not look at current cell for duplicate of current
cell
            if mm ~= nn
                potentialDuplicateX = newCoords{ii,nn}(1);
                potentialDuplicateY = newCoords{ii,nn}(2);

                % If same coordinate values exist in same row, find
which is
                % closest to the value above it
                if currentX == potentialDuplicateX && currentY ==
potentialDuplicateY
                    xAboveCurrent = newCoords{ii - 1,mm}(1);
                    yAboveCurrent = newCoords{ii - 1,mm}(2);

                    xAboveDuplicate = newCoords{ii - 1,nn}(1);
                    yAboveDuplicate = newCoords{ii - 1,nn}(2);

                    % Find difference between current column and
row/col above
                    % and difference between duplicate column and
row/col above
                    differenceCurrent = abs(xAboveCurrent - currentX)
+ abs(yAboveCurrent - currentY);
                    differenceDuplicate = abs(xAboveDuplicate -
potentialDuplicateX) + abs(yAboveDuplicate - potentialDuplicateY);

                    % Find which difference is smaller. The column
with the
                    % larger difference from the column/row above will
be set
                    % to nan
                    if differenceCurrent < differenceDuplicate
                        newCoords{ii,nn}(1) = nan;
                        newCoords{ii,nn}(2) = nan;
                        newRadius{ii,nn} = nan;
                    else
                        newCoords{ii,mm}(1) = nan;
                        newCoords{ii,mm}(2) = nan;
                        newRadius{ii,mm} = nan;
                    end
                end
            end
```

```
        end
    end

end


%% Convert Pixels to Distances - Tank Calibration

load('cameraParams.mat');
% Load in raw (distorted) calibration image
calImage = imread('G0161897.jpg');
% Undistort Image
calUndistort = undistortImage(calImage,cameraParams);

% figure()
% imshow(calUndistort)
% manually determine distance from bottom corner of square to diagonal
% corner of square in pixels
X1 = 1973; % bottom left
Y1 = 1669;
X2 = 2220; % top right
Y2 = 1458;
X0 = 2204; % bottom right
Y0 = 1686;
pixels_diag = sqrt((X2-X1)^2 + (Y2-Y1)^2);
centimeters_diag = sqrt((2.5^2) + (2.5^2));

centimeters_per_pixel = centimeters_diag/pixels_diag;
diameter_centimeters = 2*centimeters_per_pixel*(cell2mat(newRadius));

% Plot variation in detected urchin diameter
figure()
colors = {'g','m','c','b','r','y','k','w'};
plot(diameter_centimeters(:,1),'g','linewidth',1.5)
hold on
plot(diameter_centimeters(:,2),'m','linewidth',1.5)
plot(diameter_centimeters(:,3),'c','linewidth',1.5)
plot(diameter_centimeters(:,4),'b','linewidth',1.5)
plot(diameter_centimeters(:,5),'r','linewidth',1.5)
xlim([0 len])
legend('1 Green','2 Magenta','3 Cyan','4 Blue','5 Red')
xlabel('Frame Number')
ylabel('Urchin Diameter [cm]')
title('Urchin Diameters - 2 Lights')

% Calculate Mean and STD of Diameters
meanDiam = zeros(1,wid);
stdDiam = zeros(1,wid);
for ff = 1:numObjects
    meanDiam(1,ff) = mean(diameter_centimeters(:,ff));
    stdDiam(1,ff) = std(diameter_centimeters(:,ff));
end
```

```
%% Plot Track

for aa = 1:len
    for bb = 1:numObjects
        X(aa,bb) = newCoords{aa,bb}(1);
        Y(aa,bb) = newCoords{aa,bb}(2);
    end
end

X_dist = X*centimeters_per_pixel;
Y_dist = Y*centimeters_per_pixel;

figure()
image = imread(imageNames{1});
image = undistortImage(image,cameraParams);
GRAY = rgb2gray(image);
imshow(GRAY)

% figure()
% imshow(imgcrop)
hold on

for kk = 1:numObjects
    colors = {'g','m','c','b','r','y','k','w'};
    plot(X(:,kk),Y(:,kk),'o','Linewidth',3,'Color',colors{kk});
    hold on
    plot(X(:,kk),Y(:,kk),'--','Linewidth',1.5,'Color',colors{kk});
    set(gca, 'YDir','reverse')
end

% Plot Sea Urchin Tracks
figure()
for kk = 1:numObjects
    colors = {'g','m','c','b','r','y','k','w'};

plot(X_dist(:,kk),Y_dist(:,kk),'o','Linewidth',3,'Color',colors{kk});
    hold on
    plot(X_dist(:,kk),Y_dist(:,kk),'--
','Linewidth',1.5,'Color',colors{kk});
    xlim([335*centimeters_per_pixel 3666*centimeters_per_pixel])
    ylim([160*centimeters_per_pixel 2857*centimeters_per_pixel])
    set(gca, 'YDir','reverse')
    xlabel('X motion [centimeters]')
    ylabel('Y motion [centimeters]')
    title('Sea Urchin Tracks - 2 Lights')
end


%% Urchin Velocities
```

```
difference_distance_X = zeros(len,wid);
difference_distance_Y = zeros(len,wid);

% Calculate Distance Traveled between each image frame captured
for pp = 1:wid
    for uu = 2:len
        difference_distance_X(uu,pp) = X_dist(uu,pp)-X_dist((uu-
1),pp);
        difference_distance_Y(uu,pp) = Y_dist(uu,pp)-Y_dist((uu-
1),pp);
        distance(uu,pp) = sqrt((difference_distance_X(uu,pp).^2) +
(difference_distance_Y(uu,pp).^2));
    end
end


% Plot Velocity
time = 5; % seconds between images
velocity = distance/time; % cm/s

figure()
plot(velocity(:,1),'g')
hold on
plot(velocity(:,2),'m')
plot(velocity(:,3),'c')
plot(velocity(:,4),'b')
plot(velocity(:,5),'r')
xlim([0 len])
xlabel('Frame Number')
ylabel('Urchin Velocity (cm/s)')
title('Individual Urchin Velocities')
legend('1 Green','2 Magenta','3 Cyan','4 Blue','5 Red')

% Plot Total Distance Traveled
totalDistance = zeros(1,wid);
for rr = 1:wid
    totalDistance(1,rr) = sum(distance(:,rr));
end

figure()
bar(totalDistance(1,1:5))
xlabel('Urchin Number')
ylabel('Total Movements (cm)')
title('Total Centimeters Traveled - 2 Lights')

% Plot Distance From Start Location to End Location
distFromStart = zeros(1,wid);
for ee = 1:wid
    distFromStart_X(1,ee) = X_dist(len,ee)-X_dist(1,ee);
    distFromStart_Y(1,ee) = Y_dist(len,ee)-Y_dist(1,ee);
    distFromStart(1,ee) = sqrt((distFromStart_X(1,ee).^2) +
(distFromStart_Y(1,ee).^2));
```

```
end

figure()
bar(distFromStart(1,1:numObjects))
xlabel('Urchin Number')
ylabel('Distance(cm)')
title('Distance From Start to End Location - 2 Lights')


%% Plot Circles and Radii - Use to make time lapse

for aa = 1:len
    for bb = 1:wid
        X(aa,bb) = newCoords{aa,bb}(1);
        Y(aa,bb) = newCoords{aa,bb}(2);
    end
end

frameNum = 1;
figure()
% Start Image
GRAY1 = rgb2gray(imgcrop);
% End Image
RGB = imread(imageNames{frameNum});
RGB = undistortImage(RGB,cameraParams);
% RGB = imcrop(RGB,croparea);
GRAY2 = rgb2gray(RGB);
imshow(GRAY2)
hold on

radiusmatrix = cell2mat(newRadius);
for mm = 1:frameNum
    for jj = 1:numObjects
        centerX = X(mm,:)';
        centerY = Y(mm,:)';
        centermatrix = [centerX,centerY];
        colors = {'g','m','c','b','r','y','k','w'};

viscircles(centermatrix(jj,:),radiusmatrix(mm,jj),'Color',colors{jj})
        hold on
    end
end
```