Abstract—we address the problem of learning vision-based, collision-avoidance and target selection controllers in 3D, specifically in underwater environments densely populated with coral reefs. Using a highly maneuverable, dynamic, six-legged vehicle to swim underwater, we can exploit real-time visual feedback to make close-range navigation decisions that would be hard to achieve with other sensors. Our approach uses computer vision as the sole mechanism for both collision avoidance and the selection of the targets to observe. In particular, we seek to swim close to the reef to make observations yet avoid collisions and barren, coral-deprived regions. To do this simultaneous navigation task of collision avoidance and path selection, we use monocular image data processed in real-time. The proposed system is based on a convolutional neural network that takes as input an image from a forward-facing camera and predicts unscaled and relative heading changes. The network is trained to encode our desired obstacle avoidance and reef exploration objectives via supervised learning from human-labeled data. The predictions from the network are transformed into absolute heading changes via a combination of a temporally-smoothed proportional controller for heading targets and a low-level motor controller. This system enables safe autonomous coral reef navigation in underwater environments.

We validate our approach using an untethered, fully autonomous robot swimming within a coral reef in the open ocean. We demonstrate that the robot successfully traversed 1000m of the ocean bottom doing collision-free navigation, while collecting close-up footage of coral reefs.

I. INTRODUCTION

In this paper, we present and validate an integrated system for vision-based navigation combined with collision avoidance and visual target detection and selection. Specifically, the system we present relies solely on computer vision to select a navigational path that maximizes the value of its visual view while simultaneously avoiding collisions in a highly cluttered environment that is also visually challenging. This visual complexity could include optical artifacts such as floating particulate matter, reflective effects, and (optical) caustics, while the control problem is complicated by exogenous forces on the vehicle such as surge currents. We develop and test our navigational approach in an underwater environment where the objective is to perform close-range coral reef monitoring, classification, and observation (see Figure 1). Our approach, however, is intended to be useful for a wider range of applications.

Since underwater environments have large variations on appearance and visibility, floating particulate matter, and additionally our objective is to not only navigate safely, but also approach and observe structures of interest to a human domain expert, we resort to a learning-based approach to vision-based navigation. We first collect open-loop image data from swim-throughs by a human domain expert, such that there are examples of both good and bad navigation scenarios. The images from these data sets are then manually annotated off-line and used to train the appearance-based controller we employ. We also employ state-of-the-art vision-based SLAM methods to post-process our image data and reconstruct the trajectories.

In our experiments, we use a highly-maneuverable hexapod underwater robot that autonomously swims over the coral reefs in the open ocean. Due to the high maneuverability of the robot we employ, the typical distance between the robot and the reef structures we are recording is maintained at roughly 10-50 cm and the turning radius of the robot is roughly 7 cm. This means that swimming is, in terms of the control demands and operational parameters, much more like typical drone flying than navigation of standard underwater vehicles. To our knowledge, this close-approach navigation in six degrees of freedom (6DoF) is the first instance of vision-based underwater navigation with these kinds of performance requirements.

Vision-based odometry in six degrees of freedom has been considered by many authors [1], [2], [3], [4], [5], [6], [7], [8], [9]. In this work our emphasis is not on odometry per se, but on safe close-range navigation combined with surveillance of desirable content, but many of the constraints from odometry apply. Note, however, that aside from the need to account for observing useful content, traditional methods for visual odometry generally do not intrinsically include the control feedback for navigation nor provide an assurance of collision avoidance since that task is distinct from odometry. In addition, we need to compute a 6DoF navigation solution
in the presence of non-rigid structures, floating particular matter, lighting variations due to caustics (refraction through the water surface) and with exogenous forces (i.e. currents) and in the absence of GPS or range data. This invalidates some of the assumptions of classical methods based on a combination of visual features and inertial input and makes them difficult to apply. Note that underwater environments, GPS is essentially impossible while other sensing mechanisms such as sonar tend to be relatively bulky and slow relative to the close-range scale of the operation in which we are interested.

II. RELATED WORK

Behavioral cloning is one of the earliest techniques used for training a vision-based controller from human-provided examples. Pomerleau [10] trained a feed-forward neural network controller to predict the angle of the steering wheel for driving a van. The network is trained via supervised learning, using data collected in simulation. A similar setup has been replicated for training deep convolutional neural network controllers for self-driving cars, using data collected while driving a real vehicle [11]. Recent related methods for aerial vehicles have a similar setup but differ in the way data is collected. Loquercio et. al [12] used a dataset collected by bicycles and cars for autonomous flying in urban environments. Giusti et. al. [13] used a dataset collected with head-mounted cameras to train a controller that predicts steering angles for an aerial vehicle. A similar data collection system was used by Smolyanskiy et. al [14] to also train a controller for the lateral offset of an aerial vehicle. Our work is closely related to [14] in that we use a similar controller architecture, but differs on the data collection setup, how the system is trained, and the deployment environment.

A related method that does not rely on training a deep neural network controller is the visual teach-and-repeat framework by Furgale et. al [15]. In this framework, an autonomous vehicle is driven through a desired path, collecting images along the way. These images are used to build a locally consistent map of the environment and a reference path to be tracked. Using the map, an autonomous rover can determine how far it is from the reference path and use it as an error signal for a path-tracking controller.

The experiments we describe in this paper also bear close resemblance to the DAGGER algorithm [16]. DAGGER proceeds by training a controller from an initial set of expert demonstrations. This controller is applied on the target system, possibly leading the system to states that were not observed in the initial demonstrations. To continue, an expert must label the new states visited by the expert, thus providing the actions that they would have taken in those novel situations. The new labeled data is added to the initial dataset to train a new controller. This procedure is repeated until the controller’s performance is satisfactory. We trained our controller in a similar fashion, repeatedly deploying the controller and labelling new data as it was generated. However, in our case, the expert demonstrations came from different experts with slightly different ideas of what the robot should do in novel situations. Furthermore, the labels we obtain do not correspond directly to the actual actions taken by the robot; they represent a relative preference for changing the robot’s heading in different directions.

Recent work by Shkurti et. al. [17] uses a neural network controller to control the same underwater vehicle used for our work. Their controller consisted of a single class object detector trained to detect divers or other underwater robots and a linear controller for keeping the detected bounding boxes centered around the optical axis of the camera. In such an application, the task is defined by the controller architecture and does not provide obstacle avoidance. Such a method may be combined with ours for safe multi-robot/multi-diver convoying.

III. SYSTEM OVERVIEW

AQUA robots are a small, human-portable, six degrees-of-freedom hexapods with six flippers as shown in the left pane of Figure 2 with a system block diagram in the right pane [18], [19]. The robot is capable of complex trajectories in confined underwater spaces including dense coral reef. Sensors on the robot include an Inertial Measurement Unit (IMU), depth sensor, and three IDS Imaging UI-3251LE global-shutter cameras with 4 mm lenses (the effective field of view underwater is 107° horizontal and 91° vertical), two forward-looking cameras at the front and one at the back (configured to look down using a mirror). On-board computation includes: 1) a 500 MHz PC104 computer running a real-time operating system dedicated to low-level motor control and autopilot systems [19], 2) an Intel NUC with an i3 processor used to capture camera images, do image processing and run high-level control algorithms, and 3) a Jetson TX2 Module used to run neural networks. All three computers communicate over TCP using gigabit Ethernet.

Our autonomous system overview is shown in Figure 3 with connections of the main components. Forward-looking camera images are captured and processed by the NUC computer using custom Robot Operating System (ROS) nodes. The images are sent to the Jetson TX2, which also has the ROS services installed, where they are processed by our
neural network model as described in section V. The model predicts relative steering angles in the form of changes to the yaw and pitch. These predictions are used by a proportional controller that weighs the predictions and performs temporal filtering (as described in section VI) to output actions in the form of target yaw and pitch angles expressed in world-frame. The actions are relayed through ROS to the autopilot, which orientates the robot to the target yaw and pitch angles.

IV. DATA COLLECTION

Our dataset consisted of 12,870 labeled images that we collected on the west-coast of Barbados. We collected video footage by treating the robot as a hand-held camera and recording several sequences of coral that we found to be of interest. We also recorded undesirable sequences such as approaches to coral closer than we would want the robot to swim or footage of the sandy sea-floor or the water surface. From these video sequences, we saved thousands of individual frames that we then hand labeled.

Our labelling tool worked by displaying a single frame to the labeler (a human) as shown in Figure 4. The labeler used the arrow keys to label each image with a desired change in the yaw and pitch angles that one would expect the robot to swim. The exact criterion for each label was subjectively defined by the users doing the labeling. Because one application of this work is to study coral reefs, we generally applied the following guidelines to direct the robot’s path:

1) Avoid obstacles and when extremely close, ensure the maximum change in yaw and/or pitch.
2) Explore interesting coral.
3) Avoid sandy and plain regions.
4) Steer away from the water surface.
5) Steer away from the sea floor.

To augment the size of the training set and add variation in our dataset, we applied image mirroring (and changed the corresponding labels) and randomly added noise and changed the luminosity to account for change in lighting conditions.

V. MODEL OVERVIEW

The vision-based controller that we used in our experiments is based on a convolutional neural network (CNN) trained to predict steering actions for the robot. These actions represent the desired degree of change in the robot’s orientation (yaw and pitch) while swimming at constant forward speed for a given input image. We based our architecture, illustrated in Figure 5, on the Resnet-18 architecture [20], [21], which has been used successfully for similar visual navigation tasks [14].
a large turn to the left. The meaning of these predictions in absolute terms was a function of the mental state of the individuals labeling the data and the scaling factor introduced in the controller that used the network predictions.

Since our datasets were relatively small for the number of parameters (~13K labelled samples for 10M parameters), we opted to regularize our model via approximate Bayesian variational inference. We introduced dropout layers [22] after every convolutional layer in the network and changed the objective function to correspond with approximate variational inference, as described in [23]. In particular, we used the Concrete Dropout approximation [24], as it allowed for fitting the dropout rates for each convolutional layer in a data dependent manner. We also added a penalty term for predictions that were overconfident and performed label smoothing, as it is suggested in [14].

More specifically, we started with a neural network \( f \) with parameters \( w \), which produced two predictive categorical probability distributions, \( f^{(θ)} \) and \( f^{(φ)} \), for the yaw and pitch steering actions. We were given a dataset \( D \) consisting of tuples of input images and one-hot encoded labels \( \{(x_1, θ_1, φ_1), ..., (x_N, θ_N, φ_N)\} \). In this scenario, the cost function for training was the following

\[
\mathcal{L}(D, w) = \mathcal{L}_{\text{pred}}(D, w) + \lambda_1 \mathcal{L}_{\text{reg}}(D, w) = \sum_{(x_i, θ_i, φ_i) \in D} l(f^{(θ)}_i, θ_i, w) + l(f^{(φ)}_i, φ_i, w)
\]

(1)

where the regularization term corresponds to the KL-divergence term in [24]. The prediction loss for yaw steering actions was given by

\[
l(f^{(θ)}_i, θ_i, w) = -\sum_j θ_{ij} \log f^{(θ)}_{ij} - \lambda_2 \sum_j f^{(θ)}_{ij} \log f^{(θ)}_{ij}
\]

(2)

where first term corresponds to the cross-entropy between the network predictions and the smoothed labels, and the second term is the penalty for overconfident predictions, which aims to maximize the entropy of the predictive distributions. The loss for the pitch steering actions was similar. The hyper-parameters \( λ_1 \) and \( λ_2 \) determined the weights of the KL divergence regularization and the entropy penalty and were selected manually.

We kept dropout enabled at test time and applied temporal smoothing to the output predictions. By keeping dropout enabled, we aimed to avoid making decisions on predictions that were too uncertain. When dropout was disabled, we obtained the expected prediction over the learned posterior distribution over the parameters \( w \). With dropout enabled, we obtained predictions from samples of the posterior distribution. When the model received images that were significantly different from the ones observed during training, we expected the distribution over predictions to have high variance (corresponding to high modeling uncertainty). By using samples from this distribution over predictions, we let modeling uncertainty affect the behavior of the robot. We did temporal smoothing to avoid sudden changes on the motor commands applied to the robot.

VI. CONTROLLER

Our controller was responsible for converting the model’s predictive distributions to target yaw and pitch angles, \( y^θ \) and \( y^φ \). This conversion was done after temporal smoothing by selecting the class with the maximum probability after adding a bias weight to the class associated with the previous action taken. Our predicted action class for yaw was then given by:

\[
\hat{y}^θ = \arg\max_{k \in 0...6} (f^{(θ)} + w_k \hat{Y}^{(θ)}_{t-1})[k] \quad (3)
\]

where \( k \) is the class index, \( w_k \) the bias weight, and \( \hat{Y}^{(θ)}_{t-1} \) is the one-hot encoded class of the previous action.

Since \( \hat{y}^θ \) is an unscaled representation of the desired change in yaw, we converted \( \hat{y}^θ \) to the scaled yaw angle action (given in the robot body-frame) by multiplying it by one-sixth of the camera’s horizontal field of view (FOV\(^H\)) and proportional gain, \( P^{(θ)} \):

\[
y^θ = P^{(θ)}\hat{y}^θ\frac{\text{FOV}^H}{6} \quad (4)
\]

The same process was done for the pitch \( y^φ \) using the vertical field of view. We sent the target pose to the autopilot in world-frame coordinates, so \( y^θ \) and \( y^φ \) were converted to world-frame by the transformation \( W^t_R^{IC} \), where \( W^t_R \) was the transformation from the robot body-frame (initialized to the IMU state when an experiment started) and \( R^{IC} \) was the transformation from the camera coordinate system to the
robot body-frame, which we estimated using the camera-IMU calibration tool, Kalibr [25].

VII. EXPERIMENTAL VALIDATION

The field experiments described in this paper were performed on the west coast of Barbados (approximately 13°11’30.00"N, 59°38’30.00"W). Our data collection was done on three separate days in different regions of a coral reef. The validation trials were performed on another two separate days. Figure 3 shows two example images where the experiments took place. As described in IV, we collected training data of similar environments where the validation experiments took place, but not in the exact locations to avoid over fitting. Since the training data was collected over a number of days, we were able to capture multiple underwater conditions in terms of lighting and turbidity, which vary with the weather.

<table>
<thead>
<tr>
<th>Learning Rate</th>
<th>Batch Size</th>
<th>λ₁</th>
<th>λ₂</th>
<th>label smoothing</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0001</td>
<td>128</td>
<td>0.00001</td>
<td>0.1</td>
<td>0.1</td>
</tr>
</tbody>
</table>

TABLE I
CNN HYPER PARAMETERS

Training our CNN took five hours for 10,000 iterations on an NVidia Titan Xp. The hyper-parameters of our network are shown in Table II. Our network is implemented using TensorFlow, and on the Jetson TX2, we processed the images in real-time at 10Hz. Although the accuracy reported during training was 41%, this is for seven classes, unlike [12] which reported the accuracy on a binomial class, and [14] which was for three classes. Further, it’s possible that there exists ambiguity in the direction to go to avoid obstacles, however this is handled by our temporal smoothing.

A. Parameter Tuning

Prior to validating the performance of our model, we manually tuned the control parameters, $w_b$, $P^{(θ)}$ in two stages. The first stage used informal heuristics on dry land, followed by refinement passes in the ocean. We tuned $P^{(θ)}$ by performing three tuning trials in the open sea where the robot was allowed to operate until an overly-close approach to an obstacle was observed by a supervising human diver (at which point we physically intervened to stop the robot). At such a point, the gain was modified. We repeated this process for approximately 20 minutes for each trial. Initially (and for the first tuning trial), $w_b$ was set to zero and the initial parameters for $P^{(θ)}$ and $P^{(φ)}$ were based on our previous work on robot tracking [17] and chosen to be 0.5. Without any bias term $w_b$, the robot oscillated, particularly in the yaw axis, and in a couple instances would have collided with coral. This result can be attributed to ambiguity when avoiding obstacles. In some instances, such as an obstacle directly in front of the robot, either a left or right turn is equally acceptable, which can lead to unstable behavior if only instantaneous decisions are used. In such situations, temporal stabilization is needed and the $w_b$ introduces hysteresis, thus biasing the decision to continue the maneuver that was chosen at the previous time window.

Other tuning actions were needed to assure that the robot would be able to pitch up sufficiently rapidly to avoid a forward obstacle by moving vertically. Likewise, ocean surge currents (back and forth currents) vary rapidly over the reef and can be particular strong between pairs of obstructions (which is the same phenomenon that leads to “rip tides” that are a risk to human swimmers). In order to account for observed difficulties due to surge (especially in rough sea conditions), some parameters were also used to increase maximum swimming speed and responsiveness. The upshot is that local variations in sea conditions imply trade-offs between responsiveness, safety, battery life and data quality.

B. Model Evaluation

In a series of evaluation trials in the open ocean, the robot was able to swim continuously, avoid collisions, and successfully collect the desired image data. However, in two instances, the combination of rough seas, very close approach to the coral, and sudden changes in current provoked an intervention by the supervising divers. Whether these interventions were actually required is impossible to determine. To completely avoid the potential of collision, the operating distance would need to be increased or sea conditions selected to reduce the changes of rough sea. We estimate that over the course of the five trials, the robot swam a total distance of more than one kilometer navigating through dense coral in the open sea. The trial times and distances are tabulated in Table III.

The robustness of our approach is validated by the time and distance that the robot swam without collision. However, to further evaluate the model’s ability to navigate dense coral and compare it to future work, we post-process the data using three different techniques to gain several performance metrics. First, we run a modified version of the Direct Sparse Odometry (Direct Sparse Odometry (DSO)) algorithm [8] to estimate the trajectory that robot executed. Second, we perform stereo matching between the left and right front cameras based on Oriented FAST and Rotated BRIEF (ORB) feature matching (our implementation is similar to that described in [9]) to evaluate the distance to the nearest

<table>
<thead>
<tr>
<th>Trial</th>
<th>Time (m:s)</th>
<th>Distance Travelled</th>
<th>Percentage of Path Containing Coral &gt;10%</th>
<th>25%</th>
<th>Average Distance to Obstacles</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6:30</td>
<td>150 m</td>
<td>95.53%</td>
<td>75.14%</td>
<td>0.55 m</td>
</tr>
<tr>
<td>2</td>
<td>6:30</td>
<td>160 m</td>
<td>97.16%</td>
<td>81.20%</td>
<td>0.48 m</td>
</tr>
<tr>
<td>3</td>
<td>7:52</td>
<td>220 m</td>
<td>100.00%</td>
<td>88.45%</td>
<td>0.34 m</td>
</tr>
<tr>
<td>4</td>
<td>11:10</td>
<td>270 m</td>
<td>92.61%</td>
<td>67.07%</td>
<td>0.46 m</td>
</tr>
<tr>
<td>5</td>
<td>7:53</td>
<td>280 m</td>
<td>99.90%</td>
<td>84.32%</td>
<td>0.37 m</td>
</tr>
</tbody>
</table>

TABLE II
TIME, DISTANCE, AND STATISTICS OF CORAL COVERAGE AND OBSTACLES FOR OUR EXPERIMENTAL TRIALS.
potential obstacle in front of the robot. Third, we measure the fraction of each observed image that is made up of coral using automated image analyses described in our previous work [26].

1) Trajectory Estimation: We ran a modified version of DSO to measure the robot’s trajectory for each trial using recorded video from the back, downward-looking camera. Due to the low-texture in sandy regions of the sea floor, DSO is unable to track the point Hessians and compute the re-projection error. To overcome this inability, we resampled and normalized the distribution of point Hessians. To account for the scale invariant nature of monocular visual odometry, we manually adjusted the scale to correspond with the observed distance to the sea floor.

The computed trajectories for each trial are shown in Figure 7 overlayed on an image taken from the sky. The placement of the paths is done by our best estimate. Figure 8 shows the path and reconstruction of the environment for the second trial and Figure 9 shows an example of the robot’s path avoiding an obstacle.

2) Depth Estimation: For each stereo pair of images that we recorded from the front and left cameras, we performed depth estimation to nearby objects by robustly matching ORB features between images and triangulating their position. From these features, we measured the approximate distance to potential obstacles in the robot’s path. Note that
this approach only measured objects directly in front of the robot due to the narrow Field of View (FOV) (107° x 91°). There are many times when the robot passed less than 10 centimeters from objects that were either beside or below the robot. Figure 10 shows the distance to the closest objects for each trial. We observed that on average, the robot maintained a distance of 43 centimeters from coral. There were also very few instances where the distance to coral was further than two meters. It is expected that in some instances, due to the nature of the environment, there may be no alternative but to head in the direction of distant coral.

C. Coral Coverage

To measure the effectiveness of navigating coral regions as opposed to dull or sandy regions of the environment, we ran an automated coral classifier [26] on every image in our trials and recorded the percentage of the image classified as coral (see Figure 12). Since much of the time we were swimming over or around coral, it is reasonable to expect that a maximum of one half of the images should be made up of coral. Figure 10 shows the percentage of the image classified as coral at each frame, for each trial. On average, the robot swam while maintaining 30% of the image as coral. Table II summarizes the path coverage of coral for two thresholds, 10% and 25%. At 10% threshold, each path can be considered as coral greater than 95% of the time (trial three, is 100%). At 25% coverage, the path coverage drops to as low as 67% for trial four, and as high as 88% for trial three.

To estimate the impact of the algorithm’s effort to retain coral in the robot’s view, we compared the amount of coral visible on the paths in Fig. 7 to trajectories that did not use visual feedback to enhance coral contact. To do this comparison, we generated several synthetic straight-line trajectories in the same regions of the reef and measured the fraction of frames in which coral was visible. This yielded an average coral visibility between 40% and 76% with a mean of 67%. These figures are substantially worse than those observed by the robot, which are between 95 and 100%.

We also calculated the mean position of nearby obstacles by averaging the position of each of the ORB features weighted by their inverse distance. Figure 11 shows a sample of the first trial with the obstacle’s average (horizontal) position along with the yaw action (where the yaw has been inverted and both the yaw and average position have been normalized to increase readability). It can be seen that there is a correlation: when an obstacle is on one side of the image, the robot steers in the other. We observed this motion in very confined spaces - however, nearby objects are not the only factors in the yaw control.
VIII. CONCLUSIONS

In this paper we have described an approach to real-time vision-based robot navigation that combines collision avoidance, proximity-maintenance, and maximization of useful visual content. Unlike traditional (and impressive) methods for SLAM or visual odometry, our technique rapidly accounts for obstacles that are positioned too close to the vehicle’s direction of motion, selects steering actions to avoid collision, operates robustly in the presence of fish, soft moving coral and non-rigid structures, and avoids oscillatory actions in the presence of ambiguous visual signals. Since formalization of all these constraints, as well as the lighting variations and subjective preferences, is difficult to achieve, we employed a learning-based approach to navigation and demonstrated extensive collision-free navigation with good data acquisition properties. Notably, the quality and nature of the images we acquire make this a suitable controller for the collection of frames that are suitable for vision-based SLAM.

In contrast, un-controlled trajectories in the same regions often include numerous sequential frames with insufficient visual content for reliable long-term position estimation. We apply vision based SLAM as a post-processing stage to illustrate the trajectories that were executed.

To act consistently and decisively, we use temporal hysteresis in the controller. At present we use a very short time window, but the manner in which this parameter could be modulated as a function of distance, speed and uncertainty is a topic for future investigation.

In future work we are interested in combining our approach, SLAM-based mapping and task-driven path planning to optimize data collection objectives. We hope that this can open up new methodologies for some of collaborators to optimize data collection objectives. We hope that this approach, SLAM-based mapping and task-driven path planning is a topic for future investigation.

REFERENCES