Unmonitored Fault Detection in an AUV Using Recurrent Neural Networks

Marc Ho

Submitted under the supervision of Junaed Sattar to the University Honors Program at the University of Minnesota-Twin Cities in partial fulfillment of the requirements for the degree of Bachelor of Science, *cum laude* in Computer Science.

May 13, 2019

Abstract

Faults are an inevitability in any robotic system but are of a particularly devastating nature in underwater robotics. If unrealized, a minor fault in an autonomous underwater vehicle (AUV) could easily lead to the AUV unintentionally damaging itself, an accompanying diver, or the marine environment in which it is operating. While some types of faults can easily be detected with sensors on the AUV, it is costly or impractical to have sensors installed to monitor every potential point of failure on an AUV. In this paper, we present a software system that enables an AUV to detect unmonitored faults via the fault's indirect effects on the AUV's normal motion. This system is composed of two primary components: a recurrent neural network(RNN)-based model that predicts the future motion of an AUV and an anomaly detection algorithm that is able to detect anomalies in the motion prediction error and decide whether the anomalies indicate a fault occurred. To evaluate the performance of this system, we demonstrate its effectiveness on data collected on an AUV in both pool and field trials.

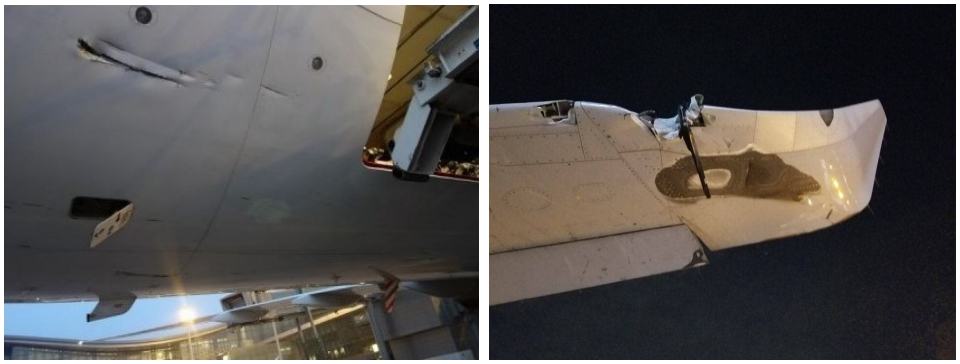
Keywords: Autonomous Underwater Vehicle — Deep Learning — Recurrent Neural Network — Fault Detection — Collective Anomaly

1 Introduction

Around the world, underwater robotics is an expanding field with a range of commercial and industrial applications such as hull inspection, surveillance, mapping, and data collection, among others. While underwater robots do make it both easier and safer for humans to perform these tasks, oftentimes such robots necessitate a human operator to manually teleoperate the underwater robot's every move. Furthermore, requiring a human operator for an underwater robot then requires the underwater robot to be physically connected by cable to some control unit since wireless signals cannot transmit through water any significant distance. Enter autonomous underwater vehicles (AUVs), underwater robots that are able to operate independently or in concert with humans without directly needing a human operator and thus can operate without a cable. Although first built during the 1960s [13], AUVs have only started to gain commercial and academic traction recently due to advances in computers' processing capabilities, allowing AUVs to better operate independently. While such AUVs can perform tasks without human operators, they need to have extremely robust internal systems to ensure they are able to understand and complete their tasks effectively.

Regardless of how sophisticated a system is, with or without humans, unforeseen problems can cause interference with the system's operation. A boat propeller could break an AUV's flipper, seaweed could tangle an AUV's propeller blades, or an AUV's motor could just cease to function due to wear

and tear. What makes these types of problems especially dangerous for an AUV compared to analogous problems for human divers (*e.g.*, cut arm by propeller blade, leg stuck in seaweed) is that the AUV will not necessarily realize that anything is wrong. Thus assuming that nothing has gone wrong, an AUV would continue to perform its task as if everything was normal, potentially exacerbating the damage to itself or causing new damage to human divers, nearby objects, and the environment. While adding sensors into the AUV to directly monitor the aforementioned problems could allow the AUV to detect said problems, it is not always feasible to do so. AUVs are already heavily constrained in terms of hardware by operating on an underwater platform of limited size, and accommodating more sensors would also increase the AUV's price significantly. Moreover, not every possible fault can be anticipated and thus be accounted for by a sensor.



(a) Qatar Airways plane with damage to fuselage. Source: [8]

(b) American Airlines plane with runway marker embedded in wing. Source: [5]

Figure 1: Two examples of damage incurred by airplanes that went unrealized by the plane's systems

This problem is not unique to AUVs and applies to many domains, most notably in airplanes, where small unnoticed faults can escalate and have disastrous consequences. While many airplanes and AUVs are able to account for unforeseen faults like these by utilizing an autopilot system that automatically adjusts for any changes affecting motion, the purpose of this paper is slightly different. Such autopilot systems do correct for unexpected faults, but they do not actively realize that said faults are happening. For example, just in the past few years, such problems have occurred in airplanes where the airplanes were damaged but proceeded to fly for hours without realizing anything was wrong. On September 15, 2015 a Qatar airways flight incorrectly took off from a runway in Miami, clipping the approach lights at the end of the runway, tearing a 46cm gash in the plane's fuselage, and leaving 90 dents and scratches across the bottom of the plane as can be seen in Figure 1a. Despite occurring during takeoff, this damage remained unnoticed until the plane landed in Qatar 14 hours later [8]. Similarly, an American Airlines flight on April 10, 2019 struck a runway distance marker while it was taking off [5]. In this case, the pilots realized something was not quite right after climbing to 20,000 feet, describing to ground control that "the plane is fine right now" despite their contradictory report of an "uncontrolled bank 45 degrees to the left". Regardless, the plane quickly returned to the airport from which it took off at which point ground crews could directly see that the marker had wrapped itself around and had become embedded in the wing as shown in Figure 1. In both these cases, it should be noted that the planes

were still able to fly due to their human pilots' and autopilot systems' corrections, but in both cases the plane itself was unable to tell that any damage had occurred. While both these planes luckily did not suffer further complications, they stress the need for such systems, especially as more and more autonomous vehicles are developed that do not have any humans directly monitoring all actions.

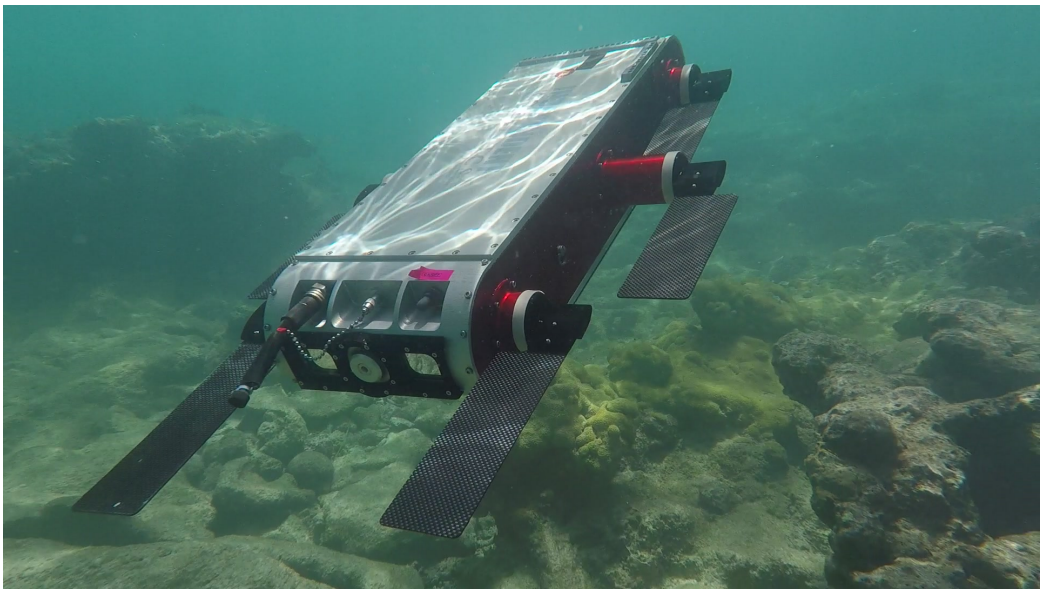


Figure 2: Aqua8/Minnebot swimming over a reef

Without sacrificing too much generality of the system, this paper will focus on detecting motion-related faults in a 6-flipper AUV, pictured in Figure 2 that will be described in greater detail in Section 4. This AUV has one of the aforementioned autopilot systems that is able to correct for the faults we were testing such that its motion is roughly similar to normal. However, like previously mentioned, this autopilot system is unable to detect that any

faults occur despite its ability to correct for faults.

To detect these unmonitored faults, we theorized that when these faults occurred in our AUV, the swimming motion of the AUV would be affected. As a result, if the swimming motion of the AUV deviated significantly from its "normal" swimming motion, that would indicate that some fault had occurred. As an analogy, if a car gets a flat tire, the car will veer to the side even if its wheels are pointed straight. The driver is thus able to infer that a tire is flat from how the car is not moving as it should. Similarly, we aimed to design a system in our AUV that would simulate this human intuition to infer whether or not a fault occurred.

To accomplish this goal, there were two primary components to the system: a prediction module to predict how the AUV should move under normal conditions and an anomaly detection module to compare the predicted motion with the actual motion of the AUV and decide if a fault has occurred. Our prediction module utilizes a recurrent neural network with Long Short-Term Memory cells to sequentially read in the variables relevant to the AUV's motion and, without needing much pre-processing, predict subsequent motion. The anomaly detection module utilizes a collective anomaly detection algorithm that uses multiple thresholds to evaluate a series of prediction errors to detect anomalies over series of data points rather than point anomalies. After performing the collective anomaly detection algorithm, we then implement a second "stacked" sliding window over the sliding windows previously generated to make the final determination of whether or not a

fault occurred.

The main purpose of this thesis was to propose a method for accurate prediction of an AUV's motion and develop a method to detect anomalies in the prediction error that indicate an unmonitored fault occurred. We evaluate the proposed system's effectiveness in a variety of situations to give insight into the conditions under which it works effectively, the conditions under which it is less effective, and why we see these results. This should provide a basic proof-of-concept for this system from which further work could be based to develop a more comprehensive anomaly detection system for our AUV.

2 Background

2.1 Non-Deep Learning-based Approaches for Robotic Anomaly Detection

While not much work has specifically been conducted on fault detection in AUVs, there has been significant work done on detecting anomalies in robotic platforms in general. One of the more common (and older) approaches for online robotic anomaly detection is to use probabilistic models or graph analysis to analyze robotic sensory data to detect anomalies. In [4], the authors split the problem of robot anomaly detection explicitly into a spatial and temporal component to detect anomalies of each type separately. They use

self-organizing feature maps to perform spatial anomaly detection and utilize a combination of common graph analysis tools and concepts based on probabilistic graphical models to perform temporal anomaly detection. [6] also attempts to develop an anomaly detection system with a similar purpose to ours: to serve as an additional tool for the robot's self-cognition rather than a complete replacement. Their system implements this by building maps of valid and invalid robot states and using a support vector machine to classify between them.

2.2 Neural Networks

A relatively recent emerging tool for anomaly detection is the usage of deep artificial neural networks (ANNs). ANNs are computational models that are loosely based off theories for how biological neurons function. They estimate complex non-linear functions to perform specific tasks by applying aweights to the input data to create an estimate for some output parameter. This model then "trains" itself on data to iteratively update its weights to steadily increase the accuracy of its estimations. If multiple layers of weights are applied in sequence between the input and output of an ANN, it is then called a "deep" neural network. Artificial neural networks first started growing in popularity in 1986 when the authors of [12] proposed the usage of back-propagation to train ANNs to learn representations of features. However, deep ANNs were still severely limited by the amount of time they took to train until work such as that of [9] enabled the usage of GPUs to

train deep neural networks at rates orders of magnitude faster than before.

2.3 Usage of Simple ANNs for Prediction

Simple ANNs have been used directly for anomaly detection, but of more particular import to our system is how accurately they are able to predict data. In [7], the authors used a simple ANN to predict a ship's motion up to 10 seconds into the future for use in the deployment of "remote piloted vehicles" (read: missiles). This setup is very similar to our own since this work attempts to predict motion patterns in a water-borne vehicle based on previous motion; however, it differs in form since it performs a single prediction for a range of time whereas we want a system that actively performs predictions as it receives more input data. While this work is able to predict within a reasonable degree of accuracy, simple ANNs lack a temporal component since all previous motion is fed into the model together. While the ANN model can learn to approximate the temporal relations of the data, it is not as explicit within the structure of the network as it is in recurrent neural networks.

2.4 Recurrent Neural Networks

Recurrent neural networks are a special class of neural networks that feed data into the network as a sequence rather than all at once into the first layer of the network. This better enables the network to understand a temporal

sequence if the ordering of the data plays a significant role in the accomplishment of the task that the network is trying to learn. Like the layers of simple ANNs, each layer in an RNN receives the output of the previous layer, and its output is passed on as input to the next layer. However, unlike the layers of simple ANNs, each layer in an RNN can also take the next entry in a sequence as input and give a separate output from the output that is passed to the next layer.

A basic RNN possesses an internal state that is modified in each layer by three sets of weights. The first and second sets of weights are used to combine the output of the previous layer with the input at that timestep to generate the new internal state. The third set of weights is then applied to the internal state to generate the output of the RNN at that timestep. The formal equations are defined in Equations (1) and (2) where the variables are defined as follows:

- h_t: internal state at timestep t
- f_h: activation function for internal state transition
- W_h: weights for internal state transition
- W_x: weights for input calculations
- x_t: input at timestep t
- y_t: output at timestep t

- f_o: activation function for output calculation
- W_o: weights for output calculation

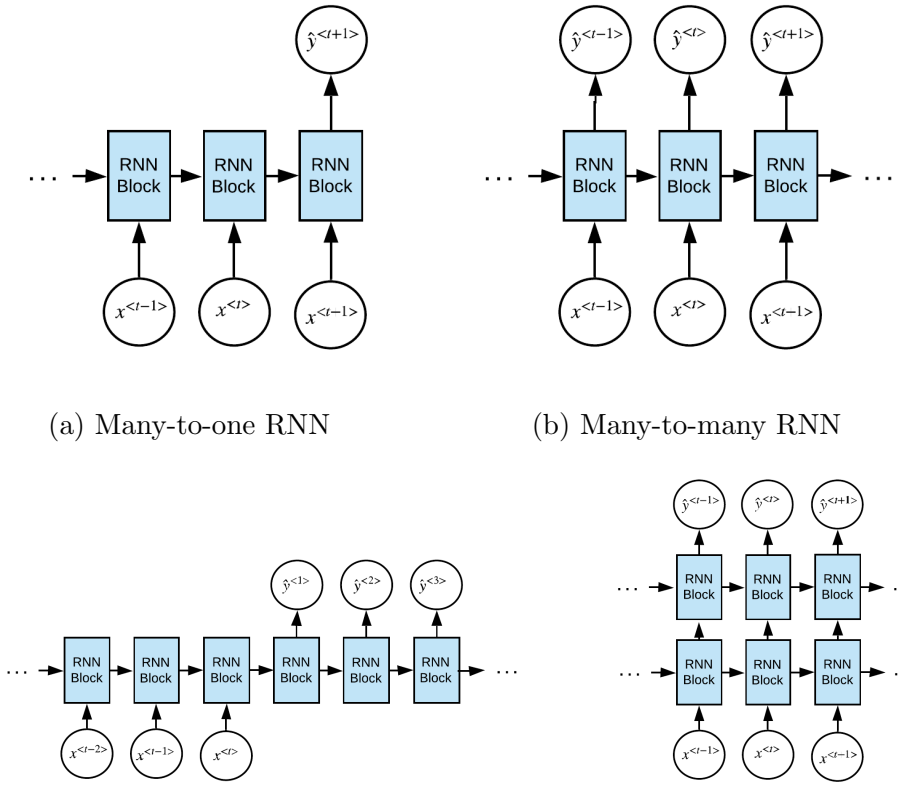
$$h_t = f_h(W_h * h_{t-1} + W_x * x_t) \tag{1}$$

$$y_t = f_o(W_o * h_t) \tag{2}$$

RNNs can be further differentiated from standard neural networks because each layer in an RNN has the same sets of weights, allowing RNNs to process sequences of various lengths without issue. Moreover, RNNs can have their layers stacked "vertically" where the vertical layers take the output of the layers "under" them as an input to learn more complicated functions. Due to these variable factors, RNNs have a wide variety of architectures that can be chosen based on their intended task as shown in Figure 3 where x^t is the input at timestep t, the arrows indicate where weights are applied to the internal state and input, and y^t is the output at timestep t.

2.5 Long Short-Term Memory

RNNs can be further modified by changing the calculations performed for the state transition within the "RNN blocks" in Figure 3. One of the most commonly used variants for the calculations is the Long Short-Term Memory (LSTM) block. LSTMs implement a "memory cell" alongside the internal state that retains long-term information whereas the internal state contains



(a) Many-to-one RNN

(b) Many-to-many RNN

(c) Offset input/output many-to-many RNN

(d) Multi-layered many-to-many RNN

Figure 3: Examples of various RNN architectures

short-term information. While LSTM blocks have greater computational complexity than standard RNN blocks, they are generally more effective than RNNs for performing tasks where important events are separated by gaps of unknown duration since the memory cell is more invariant over multiple layers. Since their initial proposal in 1997, LSTMs have undergone many changes to how their gates, memory cells, and activation functions are

implemented.

The most common variant of an LSTM differentiates itself from a standard RNN by including the aforementioned memory cell, an input gate, a forget gate, and an output gate. These "gates" control how the information in the internal state and the memory cell interact and are changed with each subsequent layer in the network. The implementation for an LSTM is shown below in Equations 4, 5, 6, 7, 8, and 9 where the variables are defined as the following:

- $\tilde{c}_t$: candidate cell state at timestep t
- i_t: input gate at timestep t
- f_t: forget gate at timestep t
- o_t: output gate at timestep t
- h_t: hidden "short-term memory" state at timestep t
- c_t: "long-term memory" cell state at timestep t
- x_t: input at timestep t
- σ: an activation function
- W: weights to apply to a distinct vector or matrix

$$\tilde{c_t} = tanh(W_{cx} * x_t + W_{ch} * h_{t-1}) \tag{3}$$

$$i_t = \sigma_i(W_{ix} * x_t + W_{ih} * h_{t-1}) \tag{4}$$

$$f_t = \sigma_f(W_{fx} * x_t + W_{fh} * h_{t-1}) \tag{5}$$

$$c_t = f_t * c_{t-1} + i_t * \tilde{c}_t \tag{6}$$

$$o_t = \sigma_o(W_{ox} * x_t + W_{oh} * h_{t-1}) \tag{7}$$

$$h_t = \sigma_t * tanh(c_t) \tag{8}$$

$$y_t = \sigma_y(W_y * h_t) \tag{9}$$

Equation 3 creates a new candidate cell state using the new input and previous hidden state. Equations 4 and 5 then compute the input and forget gates which respectively determine how to combine the candidate state and the previous cell state to create the new cell state. Equation 6 then calculates the new cell state by applying the input and forget gates to the previous cell state and candidate cell state. Lastly, Equations 7 and 8 compute the output gate and apply said output gate to the cell state to calculate the new hidden state. Oftentimes, a last set of weights is applied to the hidden state to then generate the output at each timestep like in Equation 9.

2.6 Usage of RNNs and LSTMs in Anomaly Detection

RNNs and specifically RNNs with LSTM blocks have been used extensively for anomaly detection. The general procedure to use RNNs to perform anomaly detection is to use them to predict future data in a sequence based

on previous data, then compare the predicted data with the actual data. If the predicted data is shown to be significantly different from the actual data by some metric, it can be deduced that an anomaly has occurred. This method has proven successful for a variety of applications such as for ECGs, space shuttles, power demand in electricity grids, and multi-sensor engines [10]. In their work examining these applications, [10] used a multi-layered RNN with LSTM blocks to predict future data in each dataset and then evaluate the prediction accuracy with a Gaussian distribution to determine if they were point anomalies. However, point anomaly detection will not work for our application since anomalies indicating a fault on our AUV are not point anomalies; they are *collective anomalies*. In [11], the authors use an RNN with LSTM blocks to predict anomalies by directly thresholding the prediction error to decide whether or not the sequence as a whole is considered anomalous. While this method works in post, it is not a run-time method like our system is intended to be, and even if a sliding window were implemented, this method is often more prone to false positives since individual points with large errors can skew the average.

Thus while both the above methods performed their intended goals, they do not quite fit the goal we are trying to achieve. Like [10] and [11], [1] uses an RNN with LSTM blocks to predict a sequence and then uses a collective anomaly detection algorithm to detect anomalies that occur over time in the prediction error. This work focused on network activity data for the purposes of detecting cybersecurity intrusions, but this model should likely work well

for detecting motion-related faults in our AUV.

3 The Aqua AUV

The AUV used in our experiments is an amphibious AUV of the Aqua[2] family formally named "Aqua8" and informally named "Minnebot", pictured in Figure 4. Unlike most AUVs that move using propellers or thrusters of some sort, Minnebot uses six paddles, three on each side, to move, giving it five degrees-of-freedom: X (surge), Z (heave), ϕ (roll), θ (pitch), and ψ (yaw). Minnebot is a semi-autonomous underwater vehicle outfitted with three cameras, two CPUs to separately handle motion and vision, a depth sensor, a subcon for attaching external sensor payloads, and is planned to have an onboard mobile GPU, the Jetson TX2, installed. The Jetson TX2 will not be as powerful as an actual workstation GPU, but it will allow Minnebot to run more computationally expensive tasks than most AUVs such as image processing and the deep learning models we implemented in the prediction module. Furthermore, Minnebot is outfitted with an inertial measurement unit (IMU) that measures its angular velocity and linear acceleration at a rate of 50hz.

For the purposes of this study, despite being capable of swimming autonomously, Minnebot was manually teleoperated by a human while data was recorded. While the connection of the fiber optic cable to Minnebot to enable teleoperation could hypothetically affect swimming motion, such

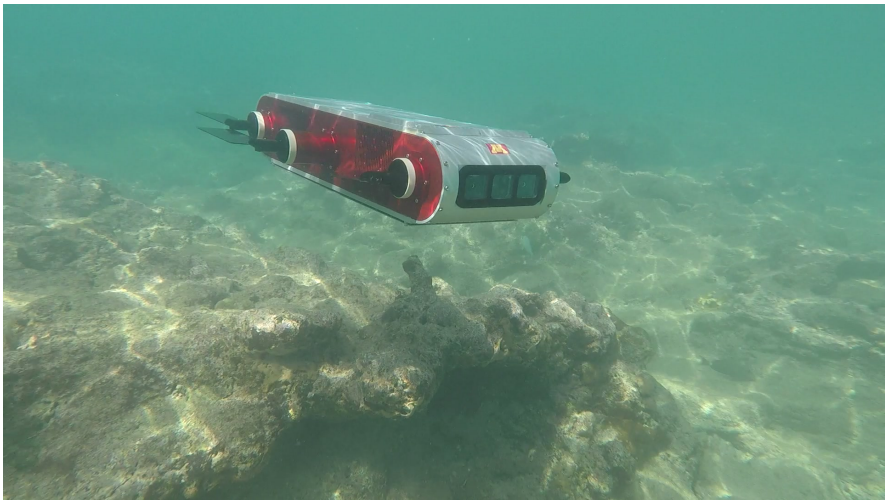


Figure 4: Minnebot as viewed from the front

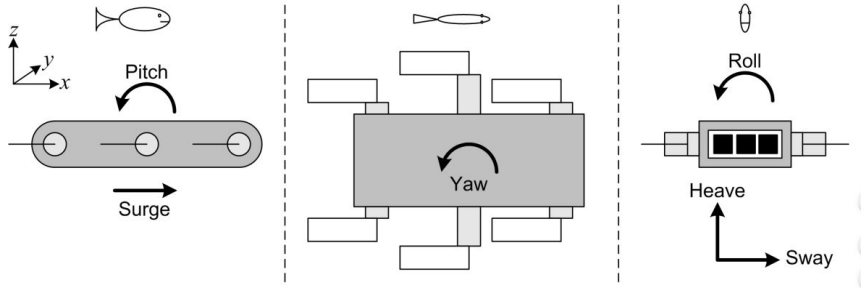


Figure 5: The Aqua Robot's Directions of Motion. Source: [3]

effects are expected to be negligible since minimal strain was placed on the cable. A full subsystems breakdown of the Aqua8 AUV is shown in Figure 6.

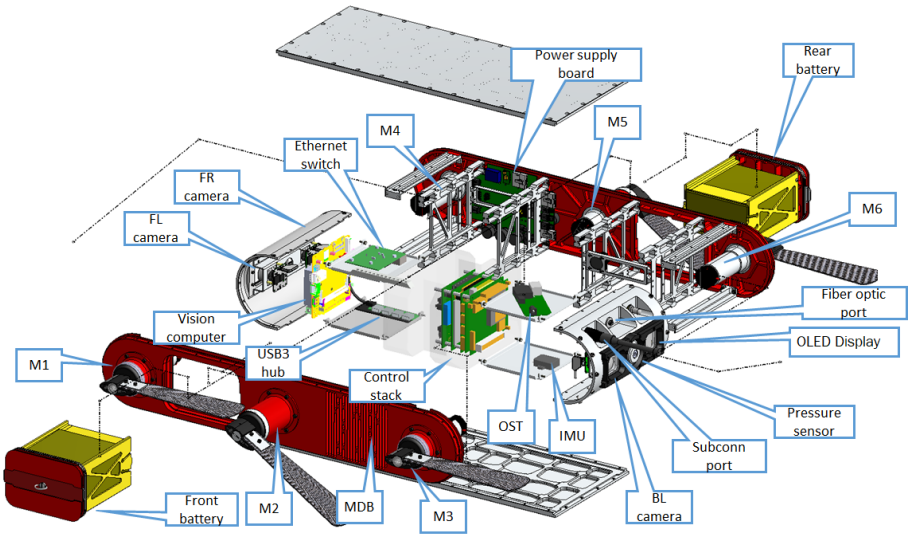


Figure 6: Breakdown of the Aqua8 subsystems. Source: McGill Mobile Robotics Laboratory

3.1 Motion-related Faults and the IMU

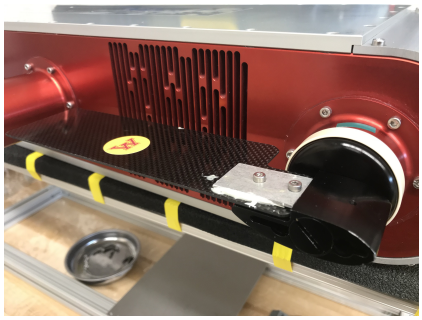
In this study, we focused mainly on detecting motion-related faults that could occur on Minnebot. The data used to define Minnebot's motion consisted of the angular velocity and linear acceleration values from the IMU and the commands for the directions of motion. It should be noted that while the IMU is fairly accurate for measurements of roll and pitch, yaw is harder to measure. Since there is no "ground truth" zero-position for yaw, the IMU's yaw measurements are prone to drift over time, or various software internal to the IMU adjusts yaw estimates in attempts to correct it.

3.2 Flippers

An example of a relatively common fault in Minnebot that would affect its motion is the breakage of one of its flippers. The flippers it currently uses to swim are made of carbon fiber and are secured to the actuators of the AUV with screws, as shown in Figure 7a. These flippers are able to tolerate repetitive motions in the water to propel the AUV in various directions, but they are brittle and prone to snapping if a solid object interrupts their motion. An example of a broken flipper and a normal flipper is shown in Figure 7b. As mentioned before, there is an autopilot system for Minnebot. The autopilot interprets the commands to move along one of the directions of motion and controls the six flippers' motions accordingly to move in that direction. This autopilot system enables the AUV to correct for flipper breakages fairly well, as will later be shown empirically in Section 6. However, as previously mentioned, the goal of this system was to detect whether or not an error occurs since the aforementioned autopilot system is unable to do so.

3.3 AUV Data Streams and Formatting

For this project, data was recorded from Minnebot in a closed environment (a swimming pool) and an open environment (the ocean). The data recorded from Minnebot to use as input for our prediction module consisted of Minnebot's five intended motions and the six values representing the linear acceleration and angular velocity of Minnebot while doing so. These data streams



(a) A closeup image showing how the flippers are secured to Minnebot

(b) A side-by-side comparison of a broken and normal flipper

Figure 7: Images of Minnebot's flippers

were recorded using the Robot Operating System (ROS) middleware at 50Hz and then reduced to a rate of 10Hz for our network to simplify the computational complexity (*i.e.,* the RNN computes 10 layers per second vs. 50 layers per second).

4 State Prediction

The goal of this component was to develop a model that would predict the future motion of Minnebot using past and present motion and intended motion data. This was accomplished by implementing a many-to-many recurrent neural network with Long Short-Term (LSTM) memory blocks using Tensorflow, as depicted in Figure 3b.

As previously stated, we used a single-layered many-to-many RNN with LSTM blocks containing 64 hidden units to perform our prediction of the

AUV's motion. While it was initially considered that a many-to-one architecture could be used to encapsulate the entire problem without including a collective anomaly detection component at all by outputting a likelihood that a fault occurred, this did not quite fit the intended constraints of this system. This system is intended to be implemented as a background system running online on an AUV and constantly running an entire network for each subsequent timestep would put significant computational load on the system. A multi-layered RNN was also considered for this task, but similarly, it significantly increases the computational load of this system to a degree that would be prohibitive for the capabilities of an autonomous platform.

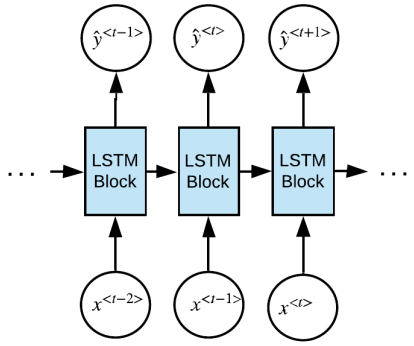


Figure 8: Our specific LSTM network for our prediction module

The input to our model at each timestep was an 11-dimensional vector containing the 5 raw "commands" that Minnebot tries to perform and the 6-dimensional IMU data. While normalization or some other form of preprocessing would likely have yielded better predictions and training time for

our network, it was chosen to use the raw data from the AUV to keep the computational overhead of the system as low as possible. The LSTM blocks were then implemented with 64 hidden units, and lastly the output at each timestep was a 6-dimensional vector representing the predicted IMU data at the subsequent timestep, which at a rate of 10Hz was 0.1 seconds later (i.e., at each moment the model predicts the IMU data 0.1 seconds in the future). A slightly modified LSTM graph depicting this is shown in Figure 8 where x and y are the aforementioned input and output data, respectively. The loss at each timestep was then determined by comparing this predicted IMU data with the actual IMU data for the next timestep and calculating the mean squared error as shown in Equation 10 where $\hat{y}$ is the predicted IMU data and y is the actual IMU data. Initially, a loss function using the total relative error was attempted, but this proved unsuccessful as the IMU and command data ranges from -1 to +1 and would often oscillate around values close to zero, resulting in huge relative errors which prevented the network from converging.

$$L^{<t>} = \frac{1}{6} \sum_{i=1}^{6} (\hat{y}_i^{<t+1>} - y_i^{<t+1>})^2 \tag{10}$$

5 Prediction Error Anomaly Detection

After predicting the future motion of the AUV, the predicted IMU values need to be analyzed to determine if the predictions deviate significantly

enough from the actual observed IMU values to be indicative of a fault. This was accomplished via the below anomaly detection algorithm based off the one described in [1].

The first component of the anomaly detection module implemented a sliding window-based algorithm like the one described in [1]. While the previous work used relative error for its error calculations, we used absolute error due to the problems of relative error described in Section 4. Irrespective of this difference, the initial component of our anomaly detection algorithm mirrored this algorithm in all other ways. This algorithm monitors three key values in each sliding window examined: the absolute error, the danger coefficient, and the average absolute error.

$$\text{DangerCoefficient} = \frac{N}{W} \tag{11}$$

$$\text{AverageAbsoluteError} = \frac{1}{W}\sum_{i=1}^{W} |\hat{y} - y| \tag{12}$$

The absolute error is calculated for each individual point within the window and is thresholded to determine if a point is considered anomalous. The danger coefficient, as shown in Equation 11, is the proportion of points within a sliding window that are considered anomalous by their absolute error, where N is the number of points considered anomalous in the window and W is the size of the window. The average absolute error is shown in Equation 12 and, as it sounds, is the average absolute error for all points within the sliding window, where W is the size of the window, $\hat{y}$ is the predicted data, and y is

the actual data. After calculating the danger coefficient and average absolute error for the window, these are also both then thresholded, and if both of these values exceed their thresholds, the sliding window is classified as being a collective anomaly as shown in Algorithm 1 where α and β are the chosen thresholds for the danger coefficient and average absolute error, respectively. Since [1] was explicitly focused on cybersecurity collective anomaly detection where there is a more explicitly determined "minimum attack time" that determines the size of the sliding window, we treated the size of the sliding window as another tunable parameter. The optimal window size and these thresholds change depending on the task and their tuning is described in more detail in Section 6.

Algorithm 1 Collective Anomaly Detection Algorithm

```
1: for all windows do
2:     if DangerCoefficient > α && AverageAbsoluteError > β then
3:         CollectiveAnomalyDetected
4:     end if
5: end for
```

To further refine our anomaly detection system since the faults we were examining are more of a state of being for Minnebot rather than a discrete temporary event like in the cybersecurity application in [1], we implemented a stacked sliding window that would slide over and examine the sliding windows created by Algorithm 1 in the previous step to make the final verdict of whether or not a fault occurred. This second stacked sliding window functioned in a simpler way than the previously derived sliding windows and just

classified the window based on the proportion of sliding windows that were classified as collective anomalies.

6 Experimental Evaluation

6.1 Description of Datasets

There were 11 datasets collected over a period of 8 months of Minnebot swimming in both a pool and in the ocean in Barbados with various other parameters changed to observe the effectiveness of the prediction and anomaly detection modules. Descriptions of the datasets are given in Table 1.

For a more detailed description of Datasets 9, 10, and 11, these datasets were collected to more directly simulate a fault that occurred while a robot was actively swimming. Dataset 9 proved fairly ineffective at simulating this, as the tape created a "floppy" connection between the flipper and the AUV's actuators rather than a rigid connection to simulate normal motion. Datasets 10 and 11 attempted to remedy Dataset 9's issue by repairing previously broken flippers with marine epoxy, but only moderately repaired the flipper so that repeated motion of the flipper in the water would eventually cause the repaired flipper to snap. This would allow a more robust observation of our system's effectiveness in actively switching from not detecting a fault to detecting a fault. Unfortunately, the intentionally badly repaired flippers were a little too fragile, leading the half-repaired flipper in Dataset 10 to immediately break again upon the first motion of the actuators. Having seen

how the flipper broke so easily when recording Dataset 10, for Dataset 11 we attempted to reinforce the badly-repaired flipper using duct tape to the best of our ability. While the flipper's marine epoxy repair snapped immediately, the tape held the flipper in place better compared to Dataset 9 but still had the same "floppy" flipper motion.

	Missing Flippers	Location	Special Notes
Dataset 1	None	Pool	N/A
Dataset 2	None	Pool	N/A
Dataset 3	None	Pool	Swimming in front of a pool waterjet
Dataset 4	Front-Left	Pool	N/A
Dataset 5	None	Ocean	N/A
Dataset 6	Front-Left	Ocean	Autopilot system enabled
Dataset 7	Middle-Left	Pool	N/A
Dataset 8	Front & Middle-Left	Pool	N/A
Dataset 9	See special notes	Pool	Front-left flipper taped on to eventually fall off
Dataset 10	See special notes	Pool	Fragile front-right flipper intended to snap
Dataset 11	See special notes	Pool	Fragile front-right flipper intended to snap but taped on

Table 1: Description of the datasets recorded and used in this project

6.2 The Training of the RNN

For the training of our RNN with LSTM blocks, we trained on data from Datasets 1 and 2 as they represented the normal swimming conditions of Minnebot, isolated from extraneous factors that could affect the motion of the

AUV. Due to constraints on the implementation of LSTMs in Tensorflow, the model was not trained on the two datasets simultaneously, but was trained sequentially for 2000 epochs on each dataset. Figure 9 demonstrates the convergence of the model during training where the test set is from an isolated subset of Dataset 1 (*i.e.*, no overlap with the training set). Since the datasets were not trained concurrently, they were concatenated together in the order in which they were trained (1 then 2). As can be seen in the figure, there is a slight kink in the training data error at epoch 2000 when the model switches from training on Dataset 1 to training on Dataset 2 but as can also be seen, the training and test set errors converged again. At the end of training, the training set error was 0.00084 whereas the test set error was 0.00552, indicating that some degree of overfitting did occur.

	Average Absolute Error
Dataset 1	0.04375
Dataset 2	0.01968
Dataset 4	0.09009
Dataset 5	0.07893
Dataset 6	0.08010
Dataset 7	0.08572
Dataset 8	0.11010

Table 2: Average absolute error for the datasets in which the AUV attempts to perform normal swimming motion

Then, using this now trained model, we ran it over Datasets 1-11 to perform predictions for each of the datasets to observe a time series of the absolute error of the predicted IMU data when compared to the actual IMU

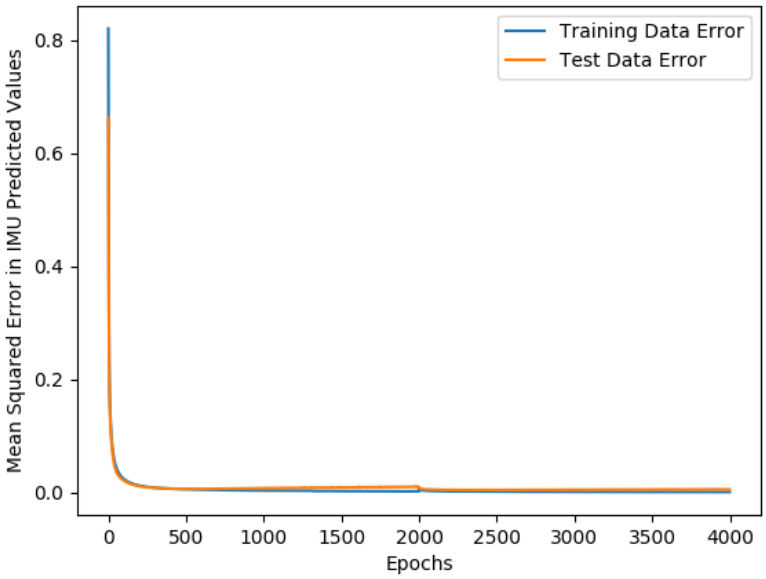


Figure 9: Convergence of the RNN on the training data

data. These predictions are depicted in Figures 10 and 11 in time series graphs representing the prediction error for each dataset. In these figures, the x-axis is the timesteps in each dataset, and the y-axis is the average absolute error in the predicted IMU values at each timestep. It should be noted that the predictions for Datasets 1 and 2 were performed on separate sequences within the dataset than the sequence on which the RNN was trained.

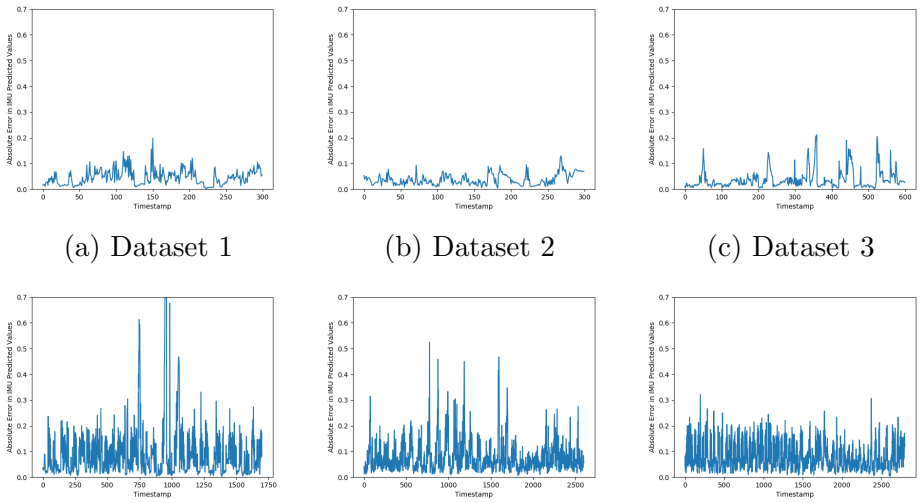
Before performing any collective anomaly detection algorithms, we can perform a visual inspection of the time-series graphs and make some observations about the quality of the predictions. It can be seen that Datasets 4, 7, and 8, the datasets in which flippers were removed and the AUV was swum under normal conditions, exhibit greater degrees of error than their

counterparts in Datasets 1, 2, and 5 where no flippers were removed. This difference can be seen more empirically in Table 2. As can be viewed in the table, Dataset 5's predictions are not as accurate as Datasets 1 and 2, which is likely due to the fact that it was recorded in the ocean under different environmental conditions than the data the model was trained on.

While it is not directly relevant to the goal of this work, it is of particular interest that the error of predictions in Datasets 5 and 6 is very similar, despite the fact that Dataset 6 was collected with a flipper removed. This is likely due to Minnebot's autopilot system that was mentioned before. The closeness of the errors in the predictions between Datasets 5 and 6 is a testament to the robustness of the autopilot system but like previously mentioned, the autopilot system does not realize that any fault had occurred, despite its ability to correct for it.

6.3 Tuning the Collective Anomaly Detection Algorithm

To tune the collective anomaly detection algorithm, we needed to choose values for the size of the sliding window, the absolute error threshold, the danger coefficient threshold, and the average absolute error threshold. To determine the best values for these parameters we created an algorithm that tested randomized permutations of these parameters that we steadily fine-tuned to maintain a collective anomaly classification rate of less than 5% for



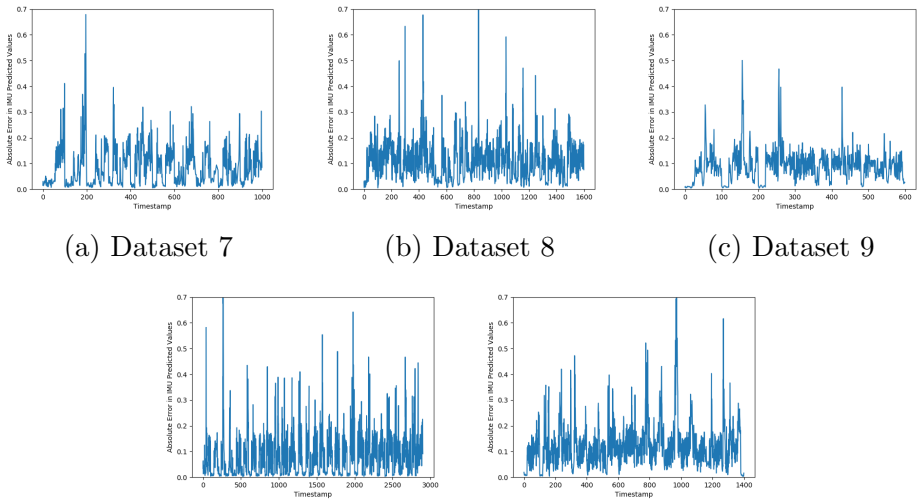
(a) Dataset 1 (b) Dataset 2 (c) Dataset 3

(d) Dataset 4 (e) Dataset 5 (f) Dataset 6

Figure 10: The absolute error of predictions on Datasets 1-6

the sliding windows for the normal swimming data (Datasets 1 and 2) and a collective anomaly classification rate of greater than 50% for the sliding windows in abnormal swimming data (Datasets 4, 7, 8). Through this process, we found that a window size of 45 timesteps (4.5 seconds), an absolute error threshold of 0.03, a danger coefficient of 0.5, and an average absolute error threshold of 0.075 consistently produced the greatest difference in collective anomaly classification rates between the normal and abnormal swimming data.

Finally, having predicted if there were collective anomalies in each sliding window, we performed the second stacked layer of sliding windows (of size 95 and thus capturing data from windows spanning 9.5 seconds) on top of the previously derived sliding windows determining if the stacked window is



(a) Dataset 7 (b) Dataset 8 (c) Dataset 9

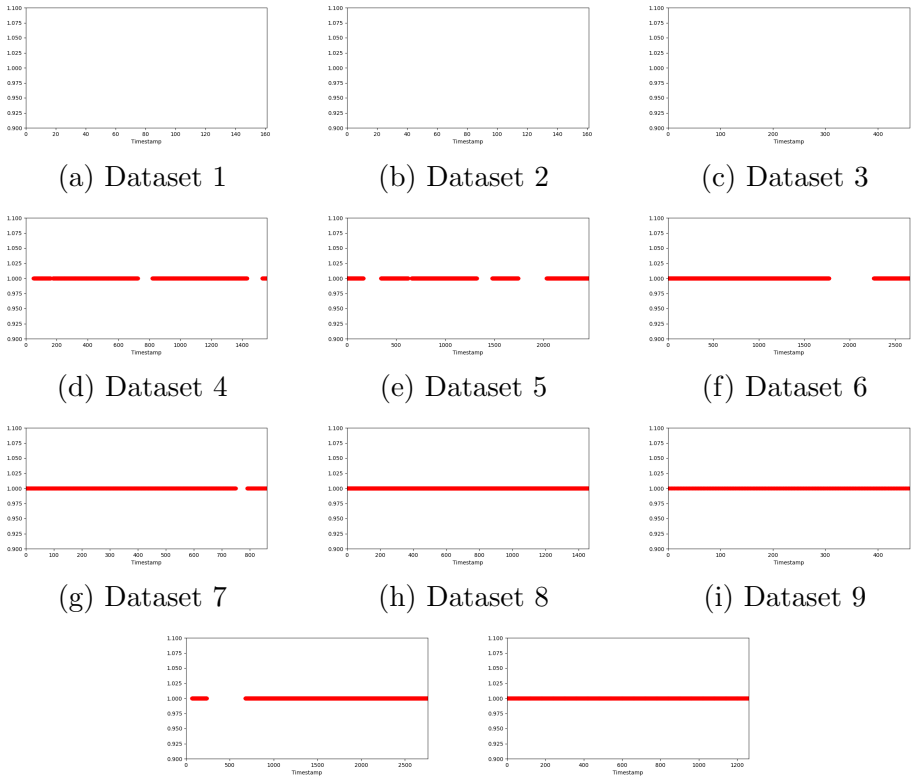
(d) Dataset 10 (e) Dataset 11

Figure 11: The absolute error of predictions on Datasets 7-12

anomalous by classifying based on a threshold of 30% of its windows being collective anomalies, giving us Figure 12. In this figure, each dataset is represented as a time series where a red dot indicates that the stacked sliding window centered at that timestep was considered anomalous.

6.4 Results

Examining the final results of the collective anomaly detection step, it is difficult to present a metric that evaluates the effectiveness of our fault detection system. As it should, it does not detect any anomalous data in Datasets 1 and 2, shown in Figures 12a and 12b, but the other datasets technically do not present anything conclusive. Ideally Datasets 4, 7, and 8 would be consistently reporting an anomaly for 100% of their duration since they were



(a) Dataset 1 (b) Dataset 2 (c) Dataset 3

(d) Dataset 4 (e) Dataset 5 (f) Dataset 6

(g) Dataset 7 (h) Dataset 8 (i) Dataset 9

(j) Dataset 10 (k) Dataset 11

Figure 12: The timesteps in Datasets 1-11 which were classified as anomalous

missing flippers for the entirety of the time they were recorded, but as can be seen in Figures 12d, 12g, and 12h, they detect anomalies for a clear majority of their duration but not all (excepting Dataset 8). It can be noted in Table 3 that Dataset 8 was determined to have an anomaly for a full 100% of its duration, a better result than Datasets 4 and 7's results. This is likely due to the greater degree of prediction error caused by missing multiple flippers at once.

Unfortunately, our model's malleability with respect to differing environ-

mental conditions is not as clearly supported as is evidenced by Dataset 5 in Figure 12e reporting an average anomaly detection rate of 73.8%. Like Datasets 1 and 2, Dataset 5 was recorded with all flippers attached as the AUV performed normal swimming motions but in the open ocean rather than in a swimming pool. While this anomaly detection rate is lower than that of the actually anomalous datasets (Datasets 4, 7, and 8), it is reporting a false positive much more than it should to be used in any capacity. Given that the RNN was trained exclusively on Datasets 1 and 2 which were recorded in a pool, it is likely that differences in environmental conditions like currents and waves and differences in ballasting on Minnebot could have affected the quality of the predictions. Surprisingly, despite our initial observations about the relatively low (compared to Datasets 4 and 7) errors in the predicted motion of Dataset 6, Dataset 6 had an average anomaly detection rate of 81.7%, a rate on an order similar to that of Datasets 4 and 7, its closed-water counterparts.

	Percent of Stacked Sliding Windows Classified as Anomalies
Dataset 1	0%
Dataset 2	0%
Dataset 4	84.2%
Dataset 5	73.8%
Dataset 6	81.6%
Dataset 7	95.5%
Dataset 8	100.0%

Table 3: Percent of stacked sliding windows in select datasets classified as anomalies

Interestingly, Dataset 3, shown in Figure 12c, seems to show that despite the model's poor ability to adjust to significant environmental differences, it possesses a resistance to minor, uncontrolled environmental events. As can be seen in Figure 10c, the prediction error certainly increases in the time series as the AUV swims in front of the waterjet and gets pushed around briefly, but this temporarily erroneous predicted motion is not reported as an anomaly even once by our prediction error anomaly detection algorithm.

Lastly, Datasets 9, 10, and 11, shown in Figures 12i, 12j, and 12k are unable to present us with much useful information. If they had worked as intended and Minnebot had begun by swimming normally until a flipper broke, we would see ideally see no anomalies reported at all until the simulated breakage occurred at which point it would be classified as anomalous for the remainder of the time series. At first glance, Dataset 10 seems to possibly be depicting this situation, but as mentioned at the beginning of this Section, the flipper broke immediately after the data started being recorded. In accordance with this, Dataset 10 should likely be similar to Datasets 4 and 7 since it effectively is a dataset in which one flipper was missing for the majority of its duration. Datasets 9 and 11, on the other hand, both presented anomaly detection rates of 100%, as shown in Figures 12i and 12i. We theorize that this is a result of how we simulated the breakage of the flipper using tape. While the broken flipper was technically still attached to the actuators of the AUV, the "floppy" motion of the flippers due to the taped connection caused the motion of the AUV to differ from normal motion even

more than if the flipper had broken entirely.

Even though the issues recording Datasets 9, 10, and 11 presented problems, our anomaly detection system does accomplish its goal of being a tool to assist in the AUV's self-awareness. While the system does not continually and invariably report that a fault occurred when a fault does occur, it is consistently reporting more than 80% of the time that a fault was detected. While the system certainly needs work to allow it more environmental flexibility, if it has experience in the environment in which its swimming, it very importantly does not report false positive fault detections.

7 Conclusion and Future Work

In this thesis, we present a system for robotic anomaly detection in our AUV that detects faults that are not directly monitored in any way. This system would ideally be run in real-time on an AUV to assist it in realizing when these unmonitored faults occur to aid in decision-making. This was accomplished by implementing a two-step system: a prediction module to predict future motion of the AUV and compare it with the actual motion and a simpler anomaly detection module to analyze the errors of the predictions.

The results indicate that this system is able to accurately predict the AUV's future motion and determine if a fault has occurred if the AUV is operating in a known environment. Equally important is that it does not show any false positives in our datasets, thus avoiding the classic "boy who

cried wolf" problem. While these results are limited in scope, they present a good proof-of-concept for this system upon which further work can be conducted.

Future work on this system could work on expanding the training set to enable our prediction system to be more generalized to varying environmental conditions rather than "perfect" conditions in a pool. Other avenues of development are varying architectures of the RNN such as the number of hidden units, usage of Gated Recurrent Units in place of LSTM blocks, and dropout to reduce overfitting. Furthermore, it would be important to evaluate the efficiency of our system when run directly on Minnebot's hardware rather than a grounded workstation. Additional features that could be added to this system could include diagnoses of the detected flippers (*i.e.*, which flipper broke) and communication methods for Minnebot to indicate to a human that it detects a fault. We hope that starting with this work, this system will eventually be robust enough to aid in the safe deployment of AUVs to perform the necessary work they do.

8 Acknowledgments

We are thankful for the assistance of Michael Fulton, Jungseok Hong, Md Jahidul Islam, and Junaed Sattar for their assistance in collecting the datasets in both pool and ocean trials used in this project.

References

[1] L. Bontemps, V. Cao, J. McDermott, and N. Le-Khac. Collective Anomaly Detection Based on Long Short-Term Memory Recurrent Neural Networks. In *Future Data and Security Engineering: Third International Conference*, pages 141–152, 2016.

[2] G. Dudek, P. Giguere, C. Prahacs, S. Saunderson, J. Sattar, L. Torres-Mendez, M. Jenkin, A. German, A. Hogue, A. Ripsman, J. Zacher, E. Milios, H. Liu, P. Zhang, M. Buehler, and C. Georgiades. Aqua: An amphibious autonomous robot. *Computer*, 40(1):46–53, Jan 2007.

[3] Philippe Giguére. *Unsupervised Learning for Mobile Robot Terrain Classification*. PhD thesis, McGill University, Dec 2009.

[4] Kai Häussermann, Oliver Zweigle, and Paul Levi. A novel framework for anomaly detection of robot behaviors. *Journal of Intelligent & Robotic Systems*, 77(2):361–375, Feb 2015.

[5] Madeline Holcombe. An American Airlines flight returned to JFK after hitting a sign during takeoff. *CNN*, 2019.

[6] R. Hornung, H. Urbanek, J. Klodmann, C. Osendorfer, and P. van der Smagt. Model-free robot anomaly detection. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3676–3683, Sep. 2014.

[7] Ameer Khan, Cees Bil, Kaye E. Marion, and Mitchell Crozier. Real-time prediction of ship motion and attitude using advanced prediction techniques. In *Congress of the International Council of the Aeronautical Sciences*, 2004.

[8] Peter Kovessi. Report: Qatar Airways fires pilots involved in Miami takeoff incident. *Doha News*, 2016.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[10] Pankaj Malhotra, Lovekesh Vig, Gautam Shroff, and Puneet Agarwal. Long short term memory networks for anomaly detection in time series. In *ESANN*, 2015.

[11] A. Nanduri and L. Sherry. Anomaly detection in aircraft data using Recurrent Neural Networks (RNN). In *2016 Integrated Communications Navigation and Surveillance (ICNS)*, pages 5C2–1–5C2–8, April 2016.

[12] D. Rumelhart, G. Hinton, and R. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, Oct 1986.

[13] H.R. Widditsch. SPURV - The First Decade. Technical report, University of Washington Applied Physics Laboratory, 1973.