Fault detection for long-duration AUV missions with minimal human intervention

Ben Yair Raanan, Moss Landing Marine Laboratories, San Jose State University

Mentor: Dr. James Bellingham

Summer 2014

Keywords: AUV, Fault detection, Autonomy, Simulator

ABSTRACT

For autonomous platforms to be successful in long duration deployments, they must be reliable in the face of subsystem failure and environmental challenges. Here, efforts to increase vehicle reliability are made by laying framework for detection of anomalous vertical plane flight performance of the *Tethys* class long-range autonomous underwater vehicle (AUV) with high probability of detect and low probability of false detect. The newly developed fault detection system presented here, compares observed vehicle behavior to references of expected behavior catalogued from previous experience or generated by a six-degree of freedom dynamics simulation model of the *Tethys* AUV. Probabilistic binary classifiers trained to differentiate between normal and abnormal operation are used to determine the presence of a fault.

1. INTRODUCTION

1.1 *TETHYS* CLASS LONG-RANGE AUV

Tethys class long rang autonomies under water vehicles (LRAUVs) developed at the Monterey Bay Aquarium Research Institute (MBARI), are designed to satisfy the scientific need for studying biological processes at relevant temporal and spatial scales using elaborate in situ sensors. *Tethys*' development focused on combining the merits of propeller-driven and buoyancy driven vehicles, as well as adding some new features such as the drift capability [Bellingham et al. 2010].

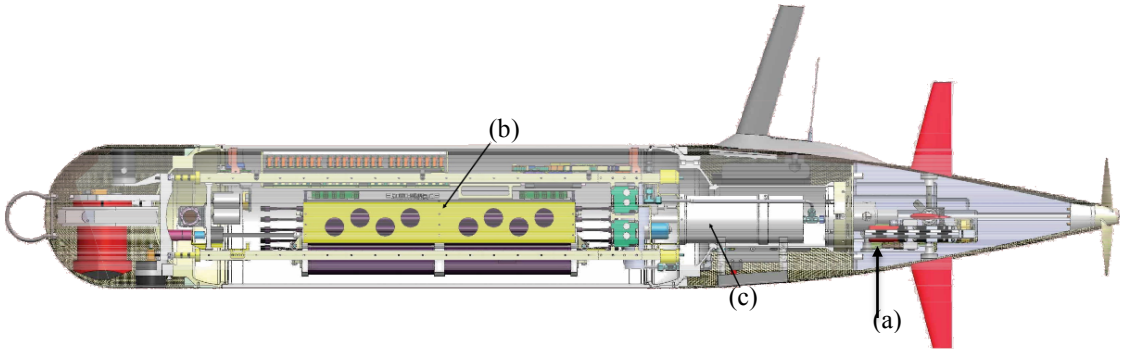

Figure 1. *Tethys* class LRAUV. Indicated here are the elevators, (a), shifting battery mass, (b), and bouncy engine (c).

LRAUV's vertical plane control is provided by elevators, shifting mass, and a buoyancy engine (Figure 1). In combination these allow the vehicle to trim to fly at zero angle of attack with no elevator angle at a range of pitch angles, and thus minimize drag [Hobson et al. 2012]. In addition to standard control surfaces (i.e. vertical rudders and horizontal elevators), the vehicle's batteries (Figure 1b) are mounted on a tray that can be moved forwards and backwards, allowing the platform's stable attitude to be adjusted through pitch angles in excess of +/- 30 degrees. *Tethys* is designed to be operated from shore, via an Iridium Satellite link. The vehicle can support an impressive 8-watt sensor payload for distances in excess of 1000 km at 1 m/s and remain continuously at sea for periods exceeding 3 weeks. For a complete vehicle profile see Bellingham et al. [2010], Hobson et al. [2012] and Kieft et al. [2011].

1.2 LRAUV RELIABILITY

For the autonomous platform to be successful in long duration deployments, it must be reliable in the face of subsystem failure and environmental challenges. The existing failure prevention system built in to the main vehicle application is already quite successful at detecting problems before the vehicle is at risk [Kieft et al., 2011]. However, in the majority of cases the system responds to failures by terminating the mission and requires operator intervention via satellite link. Over the course of thousands of hours of combined LRAUV field operations, the team of operators has encountered hundreds of critical failures resulting in unplanned operator interventions. Of those, roughly 60% were associated with vertical plane flight (vertical plane is critical for survivability) and only a handful necessitated physical recovery of the vehicle.

Almost all failures that have required operator intervention (via satellite) could be handled very simply by the vehicle. The long-term goal of this project is to give the vehicle the ability to mitigate problems autonomously by developing an onboard fault protection system that responds automatically to faults by: 1) detecting the fault, 2) diagnosing the source, 3) identifying possible responses, and 4) executing best response. Here focus is given to the development of step 1: Fault detection.

1.3 PROJECT OBJECTIVES AND FRAMEWORK

The main objective of this summer internship project is to develop methods for detecting anomalies in vertical plane flight performance of LRAUV with high probability of detect and very low probability of false detect.

Following the general framework for fault detection established by Isermann, [2005], Ernits et al., [2010] and Hwang et al., [2010], the newly developed fault detection system compares observed vehicle behavior to references of expected behavior catalogued from previous experience or generated by vehicle dynamics simulations. Failures show up as a residual, or difference, between the two output streams. To avoid false positives resulting from devotions from the norm that are within the natural variability of the vehicle's behavior, the residuals are then

passed through a probabilistic binary classifier trained to differentiate between normal and abnormal operation.

Finally, performance of the developed fault detection system is evaluated using a receiver operating characteristic (ROC) curve.

The development of the fault detection system is based on analysis of historical mission data. Presented here as a study case, is the dataset of the CANON ESP 75 mission executed by *Tethys* on September 12th 2013. This mission is highlighted since it includes a critical failure (Figure 2a) that was not identified by the existing failure prevention system and that resulted in the bottoming of the vehicle (Figure 2b). The vehicle then proceeded to push itself along the bottom until its recovery at the beach (Figure 2c)

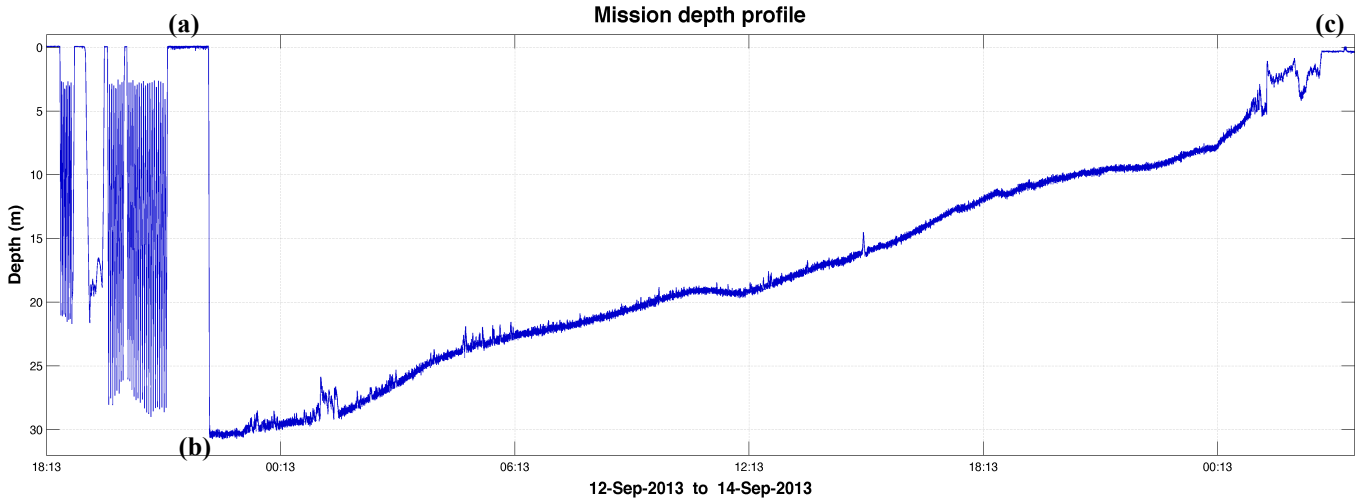

Figure 2. CANON ESP 75 mission depth profile. Shown here are the estimated times of vehicle's failure (a), bottoming (b) and recovery at the beach (c).

2. METHODS

2.1 IDENTIFYING FAULT INDICATORS

To identify properties of the vehicle's motion that are indicative of vertical plain failures, we started by reviewing datasets of previous missions (training datasets) which included vertical plain faults as well as normal vehicle operation (e.g. CANON ESP 75 mission shown above). We then performed time series analysis

of these datasets to characterize expected vertical plane flight behaviors empirically, identify outliers corresponding to failures and to inform the development of the vehicle dynamics simulator (described below).

2.2 LRAUV SIMULATION MODEL

To generate high fidelity estimates for the vehicle's state and dynamics (i.e. expected behavior) we developed a six-degree of freedom dynamics simulation model of the *Tethys* AUV. The developed simulator is in fact a hybrid of a pre-existing model developed for the LRAUV at MBARI [Kieft et al., 2011] and a model developed for the REMUS vehicle at WHOI/MIT [Prestero, 2001]. The model is based on the vehicle's equations of motion and was designed to allow complete or partial simulation of the vehicle's dynamics.

The vehicle equations of motion follow the standard submarine equations of motion framework [Abkowitz, 1969] in which external forces and moments resulting from hydrostatics, hydrodynamic lift and drag, added mass, and the control inputs ($\sum F_{ext} = F_{hydrostatic} + F_{lift} + F_{drag} + F_{control}$) are all defined in terms of vehicle coefficients.

The vehicle equations of motion consist of the following elements:

- Kinematics: the geometric aspects of motion
- Rigid-body Dynamics: the vehicle inertia matrix
- Mechanics: forces and moments causing motion

Combining the equations for the vehicle rigid-body dynamics with the equations for the forces and moments on the vehicle, we arrive at the combined nonlinear equations of motion for the LRAUV in six-degrees of freedom. Note that these equations follow the SNAME convention for the assignment of the body-fixed vehicle coordinate system.

Surge, or translation along the vehicle x-axis:

$$(m - X_{\dot{u}})\dot{u} + mz_g\dot{q} - my_g\dot{r} = X_{HS} + X_{u|u|}u|u| + \left(X_{wq} - m\right)wq + \left(X_{qq} + mx_g\right)q^2 + (X_{vr} - m)vr + \left(X_{rr} + mx_g\right)r^2 - my_gpq - mz_gpr + X_{prop} + \sum X_{fins} \quad \text{(Eq. 2.1)}$$

Sway, or translation along the vehicle y-axis:

$$(m - Y_{\dot{v}})\dot{v} + mz_g\dot{p} + (mx_g - Y_{\dot{r}})\dot{r} = Y_{HS} + Y_{v|v|}v|v| + Y_{r|r|}r|r| + my_gr^2 + (Y_{ur} - m)ur + \left(Y_{wp} + m\right)wp + \left(Y_{pq} - mx_g\right)pq + Y_{uv}uv + my_gp^2 + mz_gqr + \sum Y_{fins} \quad \text{(Eq. 2.2)}$$

Heave, or translation along the vehicle z-axis

$$(m - Z_{\dot{w}})\dot{w} + my_g\dot{p} - (mx_g - Z_{\dot{q}})\dot{q} = Z_{HS} + Z_{w|w|}w|w| + Z_{q|q|}q|q| + \left(Z_{uq} + m\right)uq + \left(Z_{vp} - m\right)vp + \left(Z_{rp} - mx_g\right)rp + Z_{uw}uw + mz_g(p^2 + q^2) - my_grq + \sum Z_{fins} \quad \text{(Eq. 2.3)}$$

Roll, or rotation about the vehicle x-axis

$$-mz_g\dot{v} + my_g\dot{w} + \left(I_{xx} - K_{\dot{p}}\right)\dot{p} = K_{HS} + K_{p|p|}p|p| - (I_{zz} - I_{zz})qr + m(uq - vp) - mz_g(wp - ur) + K_{prop} + \sum K_{fins} \quad \text{(Eq. 2.4)}$$

Pitch, or rotation about the vehicle y-axis

$$mz_g\dot{u} - \left(mx_g + M_{\dot{w}}\right)\dot{w} + \left(I_{yy} - M_{\dot{q}}\right)\dot{q} = M_{HS} + M_{w|w|}w|w| + M_{q|q|}q|q| + \left(M_{uq} - mx_g\right)uq + \left(M_{vp} + mx_g\right)vp + \left[M_{rp} - (I_{xx} - I_{zz})\right]rp + mz_g(vr - wq) + M_{uw}uw + \sum M_{fins} \quad \text{(Eq. 2.5)}$$

Yaw, or rotation about the vehicle z-axis

$$-my_g\dot{u} + \left(mx_g - N_{\dot{v}}\right)\dot{v} + (I_{zz} - N_{\dot{r}})\dot{r} = N_{HS} + N_{v|v|}v|v| + N_{r|r|}r|r| + \left(N_{ur} - mx_g\right)ur + \left(N_{wp} + mx_g\right)wp + \left[N_{pq} - \left(I_{yy} - I_{xx}\right)\right]pq - my_g(vr - wq) + N_{uv}uv + \sum N_{fins} \quad \text{(Eq. 2.6)}$$

These equations can be summarized in matrix form as follows

$$\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \\ \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} m-X_{\dot{u}} & 0 & 0 & 0 & mz_g & -my_g \\ 0 & m-Y_{\dot{v}} & 0 & -mz_g & 0 & mx_g-Y_{\dot{r}} \\ 0 & 0 & m-Z_{\dot{w}} & my_g & -mx_g-Z_{\dot{q}} & 0 \\ 0 & -mz_g & my_g & I_{xx}-K_{\dot{p}} & 0 & 0 \\ mz_g & 0 & -mx_g-M_{\dot{w}} & 0 & I_{yy}-M_{\dot{q}} & 0 \\ -my_g & mx_g-N_{\dot{v}} & 0 & 0 & 0 & I_{zz}-N_{\dot{r}} \end{bmatrix}^{-1} \times \begin{bmatrix} \sum X \\ \sum Y \\ \sum Z \\ \sum K \\ \sum M \\ \sum N \end{bmatrix} \quad \text{(Eq. 2.7)}$$

Note that the equations do not include negligible or zero-valued coefficients.

LRAUV coefficients shown in Appendix A were fitted at MBARI using tow tank experiments and empirical formulas. For the complete derivation of the combined non-linear equations of motion and model assumptions see chapters 3, 4 and 6 of Prestero [2001] and Fossen [1994].

Given the complex and highly non-linear nature of these equations, simulation of the vehicle's motion was achieved through numeric integration of the vehicle's equations of motion. The model code works by calculating for each time step the forces and moments on the vehicle as a function of vehicle speed and attitude. These forces determine the vehicle body-fixed accelerations and earth-relative rates of change. These accelerations are then used to approximate the new vehicle velocities, which become the inputs for the next modeling time step. The vehicle model requires two inputs:

- Initial conditions, or the starting vehicle state vector.
- Control inputs, or the vehicle elevator and rudder fin angles, propeller thrust and torque and battery mass position (δ_e, δ_r, X_{prop}, K_{prop} and m_{pos}, respectively). Buoyancy is assumed to be neutral at all times.

For each time step Equation 2.7 is expressed as follows:

$$\dot{x}_n = f(x_n, u_n) \quad \text{(Eq. 2.8)}$$

where x is the vehicle state vector:

$$x = [\, u \;\; v \;\; w \;\; p \;\; q \;\; r \;\; x \;\; y \;\; z \;\; \phi \;\; \theta \;\; \psi \,]^T \tag{Eq. 2.9}$$

and u_n is the input vector:

$$u_n = [\, \delta_e \;\; \delta_r \;\; X_{prop} \;\; K_{prop} \;\; m_{pos} \,]^T \tag{Eq. 2.10}$$

Partial simulation of selected variables is possible due to the structure of the state vector, which allows compartmentalization by overwriting non-relevant variables with historical data.

The Runge-Kutta method for numerical integration was implemented to further improve the accuracy of the approximation by averaging the slope at four points as follows:

$$\begin{aligned} k_1 &= x_n + f(x_n, u_n) \\ k_2 &= f\left(x + \frac{\Delta t}{2} k_1, u_{n+\frac{1}{2}}\right) \\ k_3 &= f\left(x + \frac{\Delta t}{2} k_2, u_{n+\frac{1}{2}}\right) \\ k_4 &= f\left(x + \frac{\Delta t}{2} k_3, u_{n+\frac{1}{2}}\right) \end{aligned} \tag{Eq. 2.11}$$

where the interpolated input vector

$$u_{n+\frac{1}{2}} = \frac{1}{2}(u_n + u_{n+1}) \tag{Eq. 2.12}$$

We combine the above equations to yield:

$$x_{n+1} = x_n + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{Eq. 2.13}$$

The simulator code was developed using MATLAB. Although MATLAB runs slowly compared to other compilers, the program greatly facilitates data visualization. In developing the code, we did not use any MATLAB-specific

functions, so exporting the model code to another, faster language will be easy. The model code can be seen in Appendix B.

2.3 SIMULATOR OPTIMIZATION AND ERROR MINIMIZATION

Since fin angle measurements were often found to contain offsets and inaccuracies introduced by faulty data logging, mechanical backlash and hysteresis, we developed error minimization routines to correct the control inputs and optimize the performance of the dynamic model. Simulation error was defined quite simply as

$$\varepsilon_i = \sqrt{(y_i - \hat{y}_i)^2} \qquad \text{(Eq. 2.14)}$$

where, ε is error, y is an observed vehicle parameter (e.g. pitch angle) and $\hat{y}$ is the simulated output of the same parameter. To initialize the error-minimization routine, we first selected at random a few sections of the mission known to contain normal vehicle operation. The function then created simulations of the selected sections for a range of offsets; the offset value that minimized the simulation error term was selected.

In a similar fashion, error minimization routines were implemented to adapt the vehicle coefficients to changes made to the vehicle's configuration between missions (i.e. changes to weight distribution and body shape). Code used for error-minimization can be seen in Appendix C

2.4 BINARY CLASSIFICATION AND PREDICTIVE FRAMEWORK

Probabilistic classifiers that predict the likelihood of a fault as a function of indicator value were established using logistic regression; a specific case of the generalized linear model (GLM) [Gelman and Hill, 2007]. Logistic regression is a standard way to model binary outcomes, that is, data Y that take on the values 0 or 1 (i.e. normal vehicle operation or fault, respectively). As shown in Equations 2.14, 2.15 and figure 3, logistic regression is implemented to find the equation that best predicts the probability for observing a certain value of the Y variable for

each value of the X variable (i.e. indicator). What makes logistic regression different from linear regression is that the Y variable is not directly measured; it is instead the probability of obtaining a particular value of a nominal variable (i.e. 0 or 1).

$$ln\left(\frac{p(y=1)}{1-p(y=1)}\right) = \beta_0 + \beta_1 x \quad \text{(Eq. 2.14)}$$

$$p(y=1) = \frac{e^{(\beta_0+\beta_1 x)}}{1+e^{(\beta_0+\beta_1 x)}} \quad \text{(Eq. 2.15)}$$

Equations 2.14 and 2.15 – *Logit* and *logistic* functions, respectively, where p is probability, y is the outcome variable, x is the indicator variable and $\beta_{0,1}$ are regression coefficients.

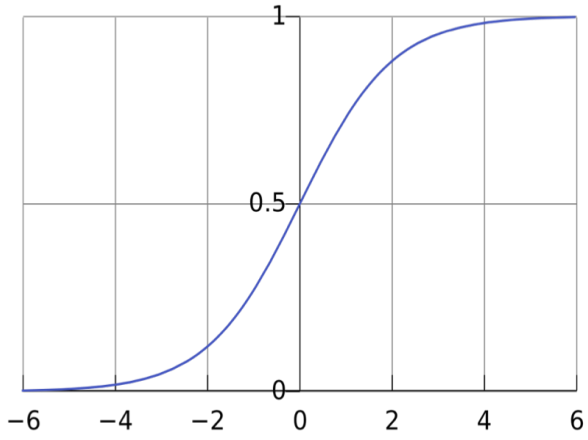

Figure 3. An example of the logistic sigmoid function of a single variable with the measurement variable on the horizontal axis and the probability for the outcome of the dependent variable (between [0,1]) on the vertical axis.

2.5 RECEIVER OPERATING CHARACTERISTIC (ROC)

The receiver operating characteristic is a metric used to check the quality of classifiers. For each output of the classifier, we applied threshold values across the interval [0,1]. For each threshold, two values were calculated, the True Positive Ratio (TPR; the number of outputs greater or equal to the threshold, divided by the number of actual positives), and the False Positive Ratio (FPR; the

number of outputs less than the threshold, divided by the number of actual negatives).

3. RESULTS

3.1 EMPIRICAL FAULT INDICATORS

Of the vehicle parameters examined, we identified negative pitch angle to be a good indicator of vertical plane flight failure. Vehicle pitch angles recorded during the CANON ESP 75 mission are shown in figure 4 (right) as a function of depth rate. Platform pitch angles logged during the malfunctioning profile that preceded the vehicle's bottoming (orange), exhibited anomalously low pitch angles in comparison to normal vertical plane flight patterns (blue). The anomalous pitch angles exceeded the vehicle's commanded pitch angle of -20 degrees by up to 15 degrees.

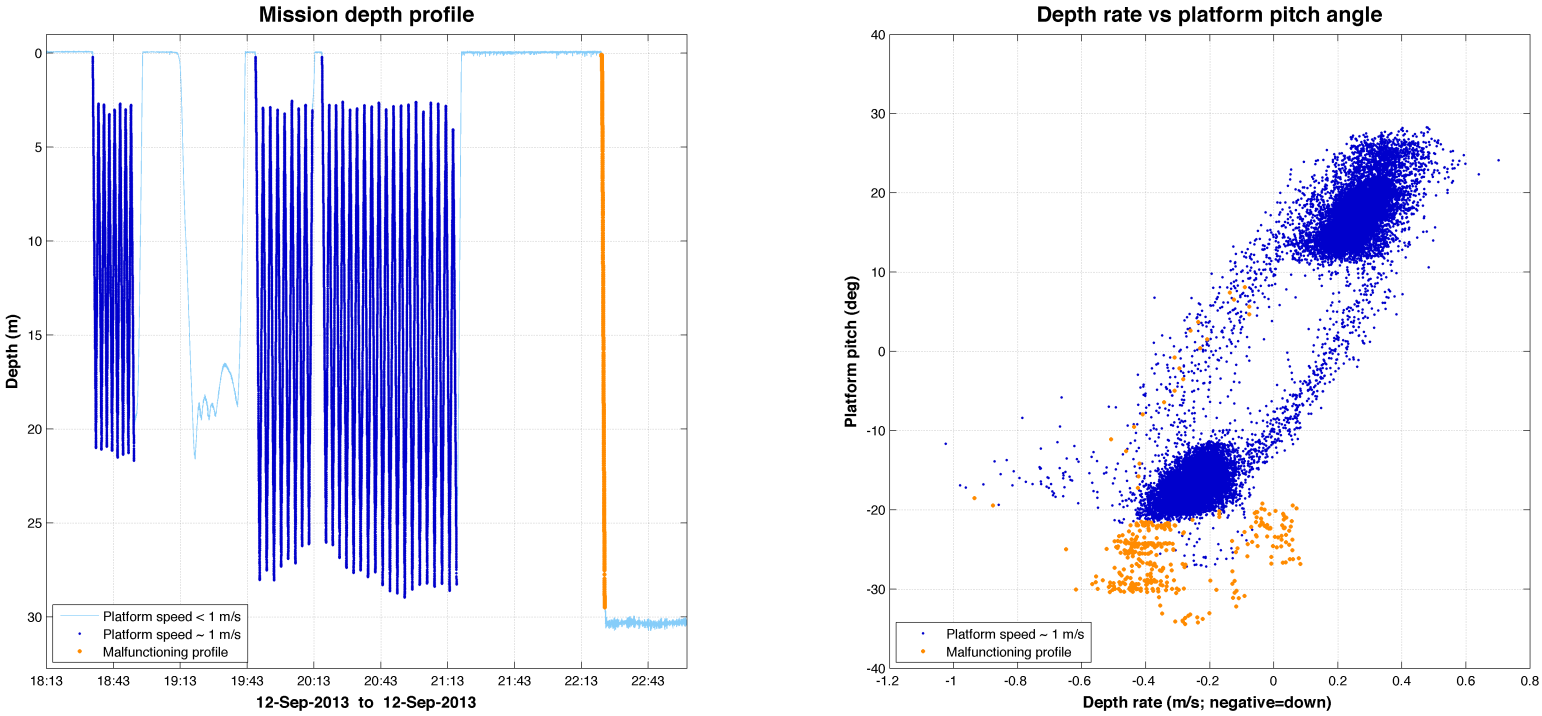

Figure 4. CANON ESP 75 mission depth profile (left) and platform pitch angle as a function of depth rate (right). Plotted in dark (light) blue are vehicle speeds exceeding (under) 0.8 m/s. Malfunctioned vehicle behavior is plotted in orange.

3.2 SIMULATED FAULT INDICATORS

The findings above provided a strong incentive to concentrate our efforts on optimizing subsets of the dynamic model to produce high quality simulations of parameters that could serve as robust indicators of vertical plain flight faults. To this end, we reduced the number of simulated variables by using historical data measurements of surge, roll and roll rate (u, p and ϕ, respectively) as model inputs. Additionally, optimization routines were focused solely on minimizing error in simulated pitch angle. Following optimization, we found it necessary to adjust a subset of the vehicle coefficients as listed in Table 3.1.

Table 3.1: Vehicle Coefficient Adjustment Factors

Coefficient	Value	Units	Adjustment Factor	Description
M_{qq}	-632.7	kg/m^2	+411	Pitch Rate Resistance Moment
dCL	4.13	n/a	+2.01	Coefficient of Lift Slope
δ_e	--	deg	-0.90	Elevator Fin Angle Offset
$\delta_e < 0$	--	deg	-0.65	Elevator Fin Angle
$\delta_e > 4$	--	deg	+1.00	Elevator Fin Angle

Adjustments made to the Pitch Rate Resistance Moment coefficient, M_{qq}, improved simulation capabilities of the vehicle's transitional phases considerably. To compensate for instabilities generated by the reduction of M_{qq}, the Coefficient of Lift Slope, dCL, was amplified. Finally, corrections were applied to Elevator Fin Angle measurements. We note that although adjustments to these coefficients were effective, they do not necessarily reflect the true cause for deviation. For reference, an example of adjusted simulation outputs is shown in Figure 5.

A summary of the residuals generated by comparing observed vehicle pitch angles with estimates generated by the dynamic model is shown in Table 3.2.

Table 3.2: Summary of Simulation Residuals

	Max	Min	Mean
Normal operation	22.80	-11.08	01.10 ± 0.041
Failure	70.76	00.22	54.21 ± 2.113

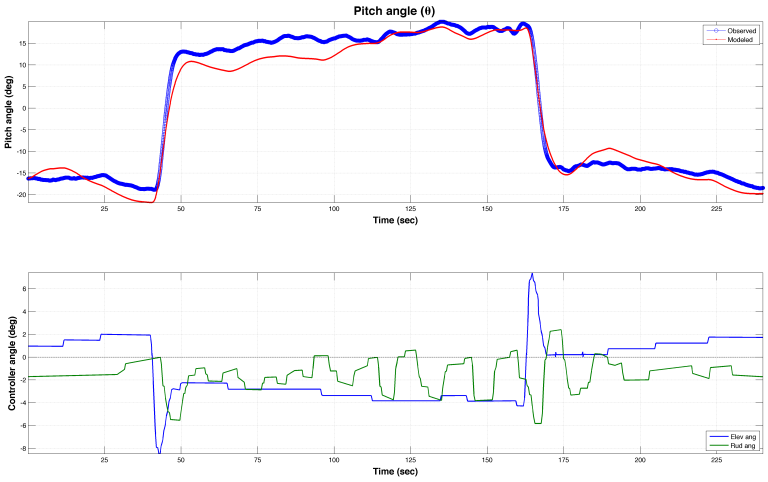

Figure 5. Top: Simulated pitch angle output (red) compared with observed data (blue). Bottom: Corresponding adjusted elevator fin angles (blue) and rudder fin angles (green).

The residuals were found to be a powerful indicator of vertical plane flight failure. As shown in Figure 6a, the absolute value of the residuals (i.e. Error) not only reflected vertical plane faults known to exist in the dataset, but also exposed a minor mid-water collision (Figures 6b and 6c) that was not known of prior to this analysis. Following its discovery the mid-water collision incident was cataloged as a fault.

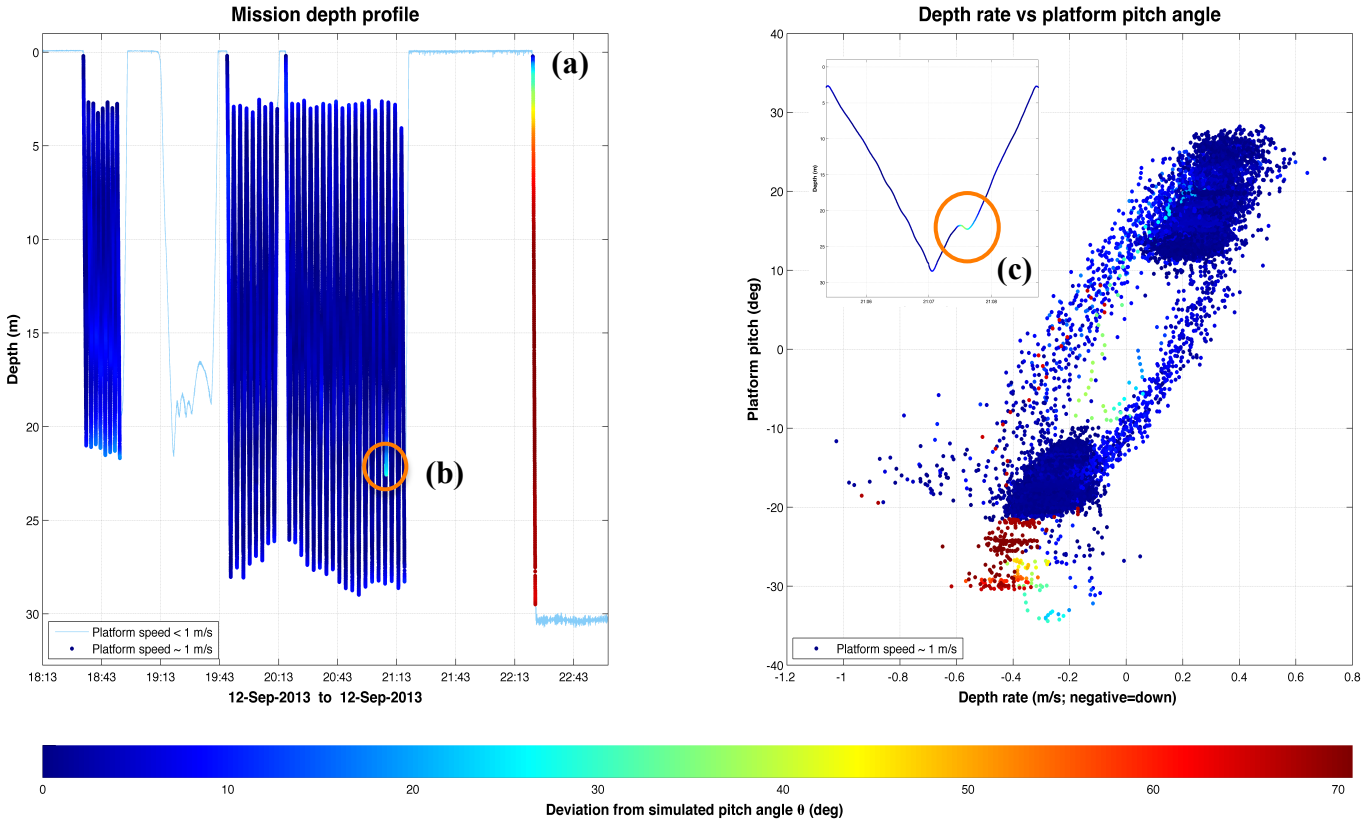

Figure 6. CANON ESP 75 mission depth profile (left) and platform pitch angle as a function of depth rate (right), overlaid by |residuals| (color-bar). Bottoming profile, (a), is clearly indicated by high residual values. Exposed mid-water collision, (b), is designated by orange ring and shown in close-up (c).

3.3 CLASSIFIER DEVELOPMENT

Both empirical and simulated indicators found during our analysis (i.e. negative platform pitch angle and simulation residuals, respectively) were fitted under the Logistic Regression framework to produce binary classifiers. The derived Logistic Regression coefficients are shown in Table 3.3.

Table 3.3: Logistic Regression Coefficients ($logit(y) \sim \beta_0 + \beta_1 x$)

Indicator (x)	Intercept (β_0)	Slope (β_1)
Empirical	-20.81 ± 0.404	-0.655 ± 0.015
Simulated	-08.63 ± 0.725	0.413 ± 0.046

The computed estimates for the probability of a fault in response to indicator value (Eq. 2.15) are illustrated in Figure 7.

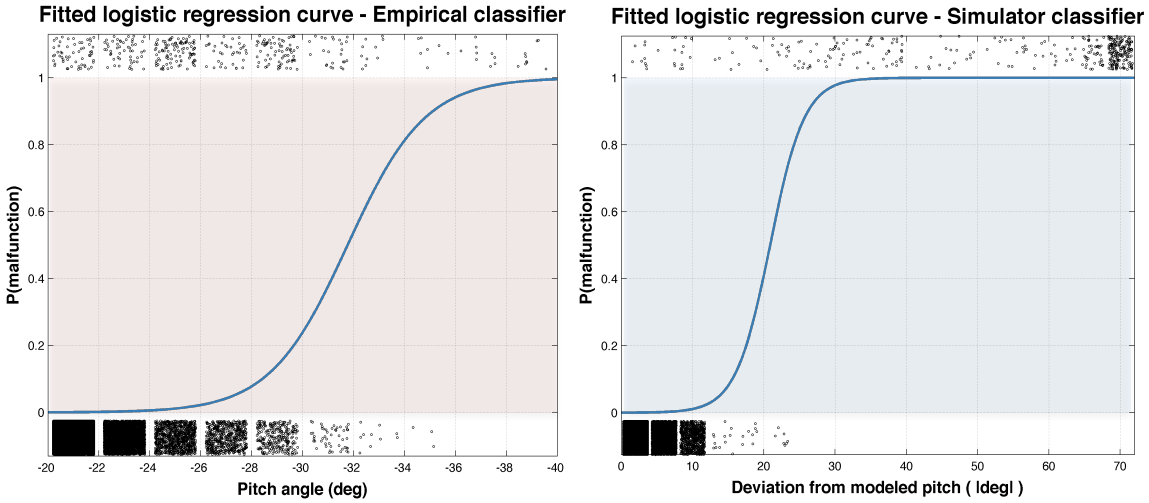

Figure 7. Logistic sigmoid curves for empirical (left) and simulated (right) indicators. Binned indicator data points (black dots) for normal operation (bottom) and faults (top) are also shown; the density of dots reflects the abundance of data in each bin.

3.4 CLASSIFIER PERFORMANCE

To quantify the success of this project (recall main objective) and for comparison of approaches, Receiver operating characteristic (ROC) curves were determined to illustrate the varied discriminatory thresholds of each classifier (Figure 8). In both

classifiers high true positive rates with negligible false positive rates were attained against increasing probability thresholds. However, when comparing between the approaches, the simulation-based classifier exhibited higher classification accuracy. Closer analysis revealed that the difference in performance was rooted in the inability of the empirical classifier to detect the mid-water collision incident (Figure 6c).

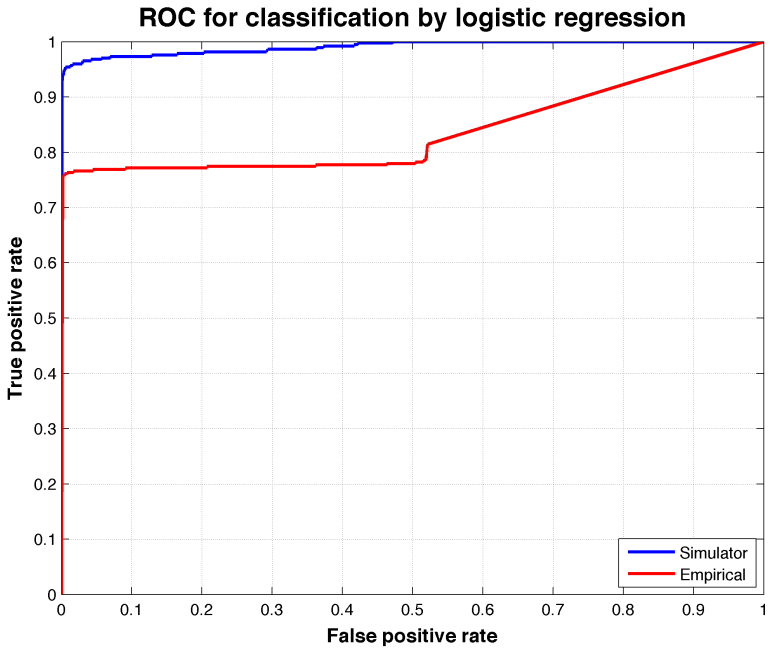

Figure 8. ROC curves representing the accuracy of classification under varying probability thresholds of empirical (red) and simulation (blue) based classifiers.

Computation of the area under the curve (AUC), a common metric used for comparison between classifiers, further stressed the superiority of the simulation-based approach. A summary of this section's results is given in Table 3.4.

Table 3.4: ROC Curve Optimal Operating Points

Approach	TPR	FPR	Threshold	AUC
Simulated	0.9005	0.0000	0.6971	0.9908
Empirical	0.7473	0.0016	0.0013	0.8371

4. DISCUSSION

The derivation of empirical indicators was relatively straightforward and yielded an effective classifier capable of identifying faults that deviate from normal vehicle behavior. In theory, once logistic regression coefficients are fitted, the classifier can easily be incorporated to the vehicle's existing software to provide continues estimates of the probability for a fault as a function of raw input values produced onboard the vehicle in real time. These computationally cheap classification methods can allow multiple classifiers based on a verity of indicators to run simultaneously and parallel to each other, and thus provide a somewhat holistic view of the vehicle's state. Moreover, based on additional analysis of historical malfunction data, classifiers may be grouped to provide a basis for determining the root cause of a malfunction.

The main compromise when implementing this approach is in its inability to identify malfunctions which occur within the natural variability of the vehicle's behavior (e.g. mid-water column collision incident Sep. 12th, 2013). Additionally, the basic definitions for behavior normality are likely to change between missions and configurations; this will necessitate retraining the classifiers on a mission-to-mission basis. Characterization of the vehicle's behavior as a function of the control parameters (e.g. elevator fin angle) stands to significantly improve the performance of empirically-based classifiers.

The results obtained from time series analysis of historical datasets greatly contributed to development of the simulation model. Once we identified the vehicle parameters linked to vertical plane flight failures, we were able to significantly enhance the accuracy of the six-degree of freedom dynamics simulation model by: 1) introducing parameters that weren't essential for modeling vertical plan flight as inputs to the model, and 2) focusing optimization routines solely on relevant subsets of the model. By implementing the above we were able to generate accurate and continues simulations of the vehicle's expected behavior throughout entire missions.

Classification based on simulation residuals has proven to be a powerful tool for identifying vertical plane flight anomalies in historical datasets. In fact, the post-mission classification process successfully exposed vertical plane flight faults that were not detected by the main vehicle application (MVA) or by operator incident investigation (e.g. mid-water collision Sep. 12th, 2013). This is greatly due to the fact that, unlike the empirical approach, the simulator responds directly to the control loop actuator commands and the vehicle "behavior patterns" are embedded in its descriptive coefficients and equations of motion.

The simulator's reliance on control system commands effectively enhances the performance of the simulation-based classifier. This is seen clearly when the AUV's flight patterns deviate from mission guidance commands. When this is the case, the control loop enforces aggressive corrections which dramatically affect the simulation stream in comparison to the observed performance of the vehicle, consequently, driving the residuals up (e.g. bottoming incident Sep. 12th, 2013). This allows the simulation-based classifier to identify deviations from expected vehicle flight patterns (i.e. faults) with high certainty and low risk of false detect in a wide rage of scenarios and regardless of mission type.

We note however, that changes to the vehicle's configuration will most likely dampen the simulator's ability to reflect the vehicle's states and dynamics with high fidelity. In some key coefficients, such as center of gravity position, minor variations may have a substantial impact on the performance of the model. Additionally, the actuator commands inputted to the simulator often contain offsets and inaccuracies that change on a mission-to-mission basis and occasionally between system reboots.

To account for these changes, we have developed and implemented the error-minimization routine. Although effective, the method employed here is computationally expensive and relies heavily on manual operator inputs. This poses limitations when examining multiple missions or when running the dynamic simulation onboard the vehicle. A solution to this problem may lie in combining a complete vehicle ontology that includes detailed descriptions of different vehicle configurations, with a fully automated and computationally efficient error-

minimizing program. Additionally, a powerful error-minimizing program may have the potential to effectively diagnose the root cause of a fault by identifying abnormalities in vehicle coefficients that are associated or caused by a specific failure.

Applying the framework presented here to a large number of missions, will be a necessary step for assessing its effectiveness and will undoubtedly raise other complications which were not challenged by currently examined datasets.

5. CONCLUSION

This summer project lays out framework for detecting anomalies in vertical plane flight performance of LRAUV with high probability of detect and low probability of false detect. The greatest accomplishment of this project is undoubtedly the advancement of the dynamics simulation model, which provides a robust reference point of expected vertical plane vehicle behavior over the length of an entire mission. The vehicle simulator was complemented by the development of software tools and a procedure for characterizing LRAUV performance throughout historical mission datasets. This project also addressed aspects of binary classification and estimation of classifier performance, which inform the transformation of raw input values into intermediate products that are expressive for discriminating between vehicle states.

While this project provides a good starting point for achieving the long-term goal of an autonomous onboard fault detection system, much more work is needed to reach full automation of the steps discussed in the chapters above.

ACKNOWLEDGEMENTS

This research project is dedicated to the memory of Drew Gashler.

This Drew Gashler Memorial Internship would not have been possible without the support of the Gashler Family, the community members of the Monterey Bay Aquarium Research Institute and Friends of Moss Landing. James Bellingham provided invaluable insight and guidance which shaped this project and much more. The contributions of Thomas Hoover, Jordan Stanway, Brian Kieft, Rob McEwen, Yanwu Zhang, Mark Abbott and Will Dillon are also noted and greatly appreciated. Special gratitude is given Tomo Eguchi and his 2-cents suggestions that went a long way.

References:

Abkowitz, M. A. (1969). *Stability and motion control of ocean vehicles: organization, development, and initial notes of a course of instruction in the subject*. MIT press.

Bellingham, J. G., Hobson, B., Godin, M. A., Kieft, B., Erikson, J., McEwen, R., ... & Mellinger, E. (2010, February). A small, long-range AUV with flexible speed and payload. In *Ocean Sciences Meeting, Abstract MT15A* (Vol. 14).

Ernits, J., Dearden, R., & Pebody, M. (2010, September). Automatic fault detection and execution monitoring for AUV missions. In *Autonomous Underwater Vehicles (AUV), 2010 IEEE/OES* (pp. 1-10). IEEE.

Fossen, T. I. (1994). *Guidance and control of ocean vehicles* (Vol. 199, No. 4). New York: Wiley.

Hwang, I., Kim, S., Kim, Y., & Seah, C. E. (2010). A survey of fault detection, isolation, and reconfiguration methods. *Control Systems Technology, IEEE Transactions on*, *18*(3), 636-653.

Hoover, T. (March, 2014) *Operating Tethys, a Long Range Autonomous Underwater Vehicle*. Friends of MLML. Lecture conducted from Moss Landing Marine Labs, Moss Landing, CA

Isermann, R. (2005). Model-based fault-detection and diagnosis–status and applications. *Annual Reviews in control*, *29*(1), 71-85.

Gelman, A., & Hill, J. (2006). *Data analysis using regression and multilevel/hierarchical models*. Cambridge University Press.

Glover, D. M., Jenkins, W. J., & Doney, S. C. (2011). *Modeling methods for marine science*. Cambridge University Press.

Hemminger, D. L. (2005). *Vertical plane obstacle avoidance and control of the REMUS autonomous underwater vehicle using forward look sonar* (Doctoral dissertation, Monterey California. Naval Postgraduate School).

Hobson, B. W., Bellingham, J. G., Kieft, B., McEwen, R., Godin, M., & Zhang, Y. (2012, September). Tethys-class long range AUVs-extending the endurance of propeller-driven cruising AUVs from days to weeks. In *Autonomous Underwater Vehicles (AUV), 2012 IEEE/OES* (pp. 1-8). IEEE.

Hoerner, S. F. (1965). *Fluid-dynamic drag: practical information on aerodynamic drag and hydrodynamic resistance* (p. 598). Midland Park, NJ: Hoerner Fluid Dynamics.

Kruschke, J. (2010). *Doing Bayesian data analysis: a tutorial introduction with R*. Academic Press.

Prestero, T. T. J. (2001). *Verification of a six-degree of freedom simulation model for the REMUS autonomous underwater vehicle* (Doctoral dissertation, Massachusetts institute of technology).

APPENDIX A: LRAUV COEFFICIENTS

```
% vehicle_coeffs.m
% July 14, 2014
%--------------------------------------------------------------------------

global zg Mqq xg


% Mass Properties:
%--------------------------------------------------------------------------
rho         = 1025;              % kg/m3
g           = 9.80665;           % m/s2
mass        = 147.8671;          % kg Flooded Vehicle mass
volume      = 0.144260585365854; % m3 (equals 1450 N buoyancy in 1025 kg/m3
water)

% Excludes buoyancy bladder at default setting
m = mass;           % kg, mass
W = m*g ;           % N, Weight
B = rho*volume*g;   % N, Buoyancy

% Geometric Parameters (Used only by the simulation):
%--------------------------------------------------------------------------
% rG - vehicle centers of gravity
xg =  0.0;          % m
yg =  0.0;          % m ***-0.000236***
zg =  0.0067940;    % m ***0.0067940***

% rB - vehicle centers of buoyancy
xb =  0.0;          % m ***0.1181***
yb =  0.0;          % m
zb =  0.0;          % m

% Fin Parameters
%--------------------------------------------------------------------------
Sfin    =   1.15e-2;        % m^2     Total area of elevator = 2 x fin.
bfin    =   18.57e-2;       % m       Fin span
zfin    =   0.152;          % m       Centerline to fin
xfin    =  -0.633;          % m       Midpoint to elevator axle (x)

% Mass Properties:
Ixx =  3.000000;    % kg-m2     Diagonal inertia tensor
Iyy =  41.980233;   % kg-m2     Diagonal inertia tensor
Izz =  41.980233;   % kg-m2     Diagonal inertia tensor

% Thruster parameters:
Kpp   = -0.191601;    % kg-m2  Rolling Resistance
Kprop =  0.23;        % N-m     Propeller Torque


% Added Mass:
%--------------------------------------------------------------------------
Yvdot  = -126.324739; % kg;     // Yvdot, kg.
Zwdot  = -126.324739; % kg;     // Zwdot, kg.
Xudot  =   -4.876161; % kg;     // Xudot, kg.
Mqdot  =  -33.463086; % kg-m2;  // Mqdot, kg-m^2.
Nrdot  =  -33.463086; % kg-m2;  // Nrdot, kg-m^2.
Kpdot  =    0.000000; % kg-m2;  // Kpdot, kg-m^2.
Kvdot  =    0.000000; % kg-m;   // Kvdot, kg-m.
Mwdot  =    7.117842; % kg-m;   // Mwdot, kg-m.
Zqdot  =    7.117842; % kg-m;   // Zqdot, kg-m.
Nvdot  =   -7.117842; % kg-m;   // Nvdot, kg-m.
Yrdot  =   -7.117842; % kg-m;   // Yrdot, kg-m.
Ypdot  =    0.000000; % kg-m;   // Ypdot, kg-m.
```

```
% Stability Derivatives:
%-------------------------------------------------------------------------
Xqq =  7.117842;    % kg-m;
Xrr =  7.117842;    % kg-m;
Xvv = -54.370919;   % kg/m;     // Xvv   , kg/m
Xww = -54.370919;   % kg/m;     // Xww   , kg/m
Yvv = -601.274653;  % kg/m      Cross-flow Drag (Yv|v|)
Yrr =  0.000000;    % n/a       Cross-flow Drag (Yr|r|)
Zww = -601.274653;  % kg/m      Cross-flow Drag
Zqq =  0.000000;    % n/a       Cross-flow Drag (Zq|q|)
Mww = -58.113144;   % kg        Cross-flow drag (-Nv|v|)
Mqq = -632.698957;  % kg-m2     Cross-flow drag (Mq|q|)
Nvv =  58.113144;   % kg        Cross-flow drag (Nv|v|)
Nrr = -632.698957;  % kg-m2     Cross-flow drag (Nr|r|)

Yuv = -23.954759;   % kg/m      Body Lift Force and Fin Lift
Zuw = -23.954759;   % kg/m      Body Lift Force and Fin Lift
Nuv = -105.660262;  % kg        Body and Fin Lift and Munk Moment
Muw =  105.660262;  % kg        Body and Fin Lift and Munk Moment

% Added Mass:
%-------------------------------------------------------------------------
Xwq = -126.324739;  % kg
Xvr =  126.324739;  % kg
Yur =  8.719853;    % kg        Added Mass Cross-term and Fin Lift
Zuq = -8.719853;    % kg        Added Mass Cross-term and Fin Lift
Nur = -61.182063;   % kg-m      Added Mass Cross-term and Fin Lift
Muq = -61.182063;   % kg-m      Added Mass Cross-term and Fin Lift

Ypq = -7.117842;    % kg-m      Added Mass Cross-term (-Zqdot)
Ywp =  126.324739;  % kg-m      Added Mass Cross-term
Zvp = -126.324739;  % kg        Added Mass Cross-term
Zrp = -7.117842;    % kg-m      Added Mass Cross-term (Yrdot)
Mpr =  33.463086;   % kg-m2     // Mpr   , kg-m^2
Mrp =  33.463086;   % kg-m2     Added Mass Cross-term
Mvp =  7.117842;    % kg-m      Added Mass Cross-term (-Yrdot)
Npq = -33.463086;   % kg-m2     Added Mass Cross-term
Nwp =  7.117842;    % kg-m      Added Mass Cross-term (Zqdot)


%-------------------------------------------------------------------------
```

APPENDIX B: SIMULATOR CODE

```
% LRAUV_SIM.M

% LRAUV_SIM: Main script for running vehicle simulation.
% Last modified Aug 1, 2014
% Ben Raanan | Inspired by REMUS_SIM.M [Prestero, 2001]
%--------------------------------------------------------------------------

clear all
close all

h = waitbar(0,'Initializing LRAUV Vehicle Simulator...');

% Load data:
fpath = '~/Documents/MATLAB/MBARI/mat/workver/';
filename = [fpath 'LRAUV_SIM_201309121813_201309140344.mat'];

%--------------------------------------------------------------------------
% STATE AND INPUT VECTORS:
% x = [u v w p q r xpos ypos zpos phi theta psi]'
% ui = [ delta_s delta_r Xprop Kprop ]'

% Arrange dataset to match model requirements
[time, time_step, xstruct, names, controls] = initialize_LRAUV_SIM( filename );

% Define time range of interest
timeIn  = datenum(2013,09,12,20,00,00);
timeOut = datenum(2013,09,12,22,00,00);

% Index start point
[~,timeIni] = min(abs(time - timeIn));

% Initiate first step and set runtime
%--------------------------------------------------------------------------
startPoint = timeIni;
timeEval    = 240; % sec, evaluation run time
n_steps     = fix(timeEval/time_step);
n = startPoint:startPoint+n_steps;
n_ind = 1:length(n);

% Define global variables
global xg zg Sfin Mqq ARe dCL CDc

zg      =   0.0067940;          % m       Center of gravity
Sfin    =   1.15e-2;            % m^2     Fin area
Mqq     =   0.35*-632.698957;   % kg-m2   Scaled Cross-flow drag (Mq|q|)
ARe     =   6.500000;           % n/a     Fin aspect ratio
dCL     =   1.5*4.130000;       % n/a     Scaled Coef. of Lift Slope
CDc     =   0.030000;           % n/a     Crossflow Coef. of Drag

% Define variables associated w/ x center of gravity
mass        =   147.8671;         % kg Flooded Vehicle total mass
movableMass =   26;               % kg Battery movable mass
dropWtMass  =   1.0;              % kg Mass of the drop weight #1, kg
dropWtX     =  -0.1330;           % m  X location of the drop weight #1, m

% Account for movable mass shift (x center of gravity)
Xmass = (movableMass.*xstruct.mass_p + dropWtMass*dropWtX)./mass;

% Get control data
ui = controls;
```

```
% Correct for offsets in data:
%--------------------------------------------------------------------------
ele_offset =  -0.9*pi/180;            % found in ele_offsetLRAUV_SIM.m
rud_offset =   0*pi/180;              % found in ele_offsetLRAUV_SIM.m

ui(:,1) = ui(:,1) + ele_offset; % zeros(size(ui(:,1)));
ui(:,2) = ui(:,2) + rud_offset;

% Correct for hysteresis and backlash offsets
%--------------------------------------------------------------------------
for k = 1:length(ui)

    if ui(k,1)<0
        ui(k,1) = ui(k,1) -0.65*pi/180;
    elseif ui(k,1)>4*pi/180
        ui(k,1) = ui(k,1) + 1*pi/180;
    end

end

% Unpack state vector
%--------------------------------------------------------------------------
x = zeros(1,12);

for c=[1:6,9,10:12];
    x(c) = xstruct.(names{c})(startPoint);
end; clear c



% RUN MODEL
%--------------------------------------------------------------------------
waitbar(0.01,h,'Running Vehicle Simulation...');
for i = startPoint:startPoint+n_steps

% Account for movable mass shift
    xg = Xmass(i) ;

% Set some variables constant
    x(1)  = xstruct.u(i);
    x(4)  = xstruct.p(i);
    x(10) = xstruct.phi(i);


    ui_in = ui(i,:); % extract control data for iteration

% Compute and log fin forces and moments:
    [ F1, F2, F3, F4, M1, M2, M3, M4 ] = robsFins(ui_in , x );
    fin.X(:,i) = F1(1)+F2(1)+F3(1)+F4(1);
    fin.Y(:,i) = F1(2)+F2(2)+F3(2)+F4(2);
    fin.Z(:,i) = F1(3)+F2(3)+F3(3)+F4(3);
    fin.K(:,i) = M1(1)+M2(1)+M3(1)+M4(1);
    fin.M(:,i) = M1(2)+M2(2)+M3(2)+M4(2);
    fin.N(:,i) = M1(3)+M2(3)+M3(3)+M4(3);

% Log timestep data:
    simlog(i,:) = [x ui_in];

% Calc next step
    [xdot,forces] = lrauv(x,ui_in); % main simulation function

% Log outputted forces
    f(:,i) = forces;
```

```
% NOTE: overwriting old states with new states, saving back at the top
%       of the loop

    % RUNGE-KUTTA APPROXIMATION to calculate new states
    % NOTE: ideally, should be approximating ui values for k2,k3
    k1_vec = xdot;
    k2_vec = lrauv(x+(0.5.*time_step.*k1_vec)', ((ui(i,:)+ui(i+1,:))./2)) ;
    k3_vec = lrauv(x+(0.5.*time_step.*k2_vec)', ((ui(i,:)+ui(i+1,:))./2)) ;
    k4_vec = lrauv(x+(time_step.*k3_vec)', ui(i,:)) ;
    x = x + time_step/6.*(k1_vec +2.*k2_vec +2.*k3_vec +k4_vec)';

    waitbar((find(n==i)/length(n)),h,['Running Vehicle Simulation... ['...
        num2str(100*(find(n==i)/length(n)),2) '%]'] );
    pause(0.01)
end
%-------------------------------------------------------------------------
waitbar(1,h,['Vehicle Simulation Complete [' num2str(100) '%]'] );


% Calc Error of simulation run
%-------------------------------------------------------------------------
error = [ sum((xstruct.theta(n+lag(minlagi))-simlog(n,11)').^2),...
    max(abs(xstruct.theta(n+lag(minlagi))-simlog(n,11)')*180/pi) ]

% Close waitbar
      close(h)
%-------------------------------------------------------------------------
```

```
% lrauv.m    Vehicle Simulator Function

% Returns the time derivative of the state vector
% Last modified July 17, 2014

function [ACCELERATIONS,FORCES] = lrauv(x,ui)

% TERMS
% ------------------------------------------------------------------------
% STATE VECTOR:
% x = [u v w p q r xpos ypos zpos phi theta psi]'
%  Body-referenced Coordinates
%  u            = Surge velocity            [m/sec]
%  v            = Sway velocity             [m/sec]
%  w            = Heave velocity            [m/sec]
%  p            = Roll rate                 [rad/sec]
%  q            = Pitch rate                [rad/sec]
%  r            = Yaw rate                  [rad/sec]
%  Earth-fixed coordinates
%  xpos         = Position in x-direction   [m]
%  ypos         = Position in y-direction   [m]
%  zpos         = Position in z-direction   [m]
%  phi          = Roll angle                [rad]
%  theta        = Pitch angle               [rad]
%  psi          = Yaw angle                 [rad]
%
% INPUT VECTOR
% ui = [delta_s delta_r]'
%  Control Fin Angles
%  delta_s  = angle of stern planes         [rad]
%  delta_r  = angle of rudder planes        [rad]

% Initialize global variables
%-------------------------------------------------------------------------
% load vdata          ;  % W and B, CG and CB coords
[ ~, Minv ] = inv_massmatrix( 'vehicle_coeffs' );   % Minv matrix
vehicle_coeffs ;             % non-zero vehicle coefficients only

% Get and check state variables and control inputs
%-------------------------------------------------------------------------
% Get state variables
u   = x(1) ; v  = x(2) ; w  = x(3) ; p  = x(4) ; q  = x(5) ; r  = x(6);
phi = x(10) ; theta  = x(11) ; psi  = x(12) ;

% Get control inputs
% delta_s = ui(1) ; delta_r = ui(2) ;
Xprop = ui(3) ; Xuu = ui(4) ;

% Initialize elements of coordinate system transform matrix
%-------------------------------------------------------------------------
c1 = cos(phi); c2 = cos(theta); c3 = cos(psi);
s1 = sin(phi); s2 = sin(theta); s3 = sin(psi);
t2 = tan(theta);

% Get fin forces and moments
%-------------------------------------------------------------------------
[ F1, F2, F3, F4, M1, M2, M3, M4 ] = robsFins( ui, x );




% Set total forces from equations of motion
%-------------------------------------------------------------------------
X = Xprop - (Wp-Bp)*s2 + Xuu*u*abs(u)...
```

```
    + m*(v*r - w*q + Xgp*(q*q + r*r) - Ygp*p*q - Zgp*p*r) ...
    + F1(1) + F2(1) + F3(1) + F4(1)...
    + Xvv*v*v + Xww*w*w + Xvr*v*r + Xwq*w*q + Xrr*r*r + Xqq*q*q;


Y = (Wp-Bp)*c2*s3 + Yvv*v*abs(v)...
    + Yuv*u*v + Yur*u*r + Yrr*r*abs(r) + Ywp*w*p...
    + m*(w*p - u*r + Ygp*(r*r+p*p) -Zgp*q*r - Xgp*p*q)...
    + F1(2) + F2(2) + F3(2) + F4(2);


Z = (Wp-Bp)*c2*c3 + Zww*w*abs(w) + Zqq*q*abs(q)...
    + Zuq*u*q + Zuw*u*w + Zvp*v*p...
    + m*(u*q - v*p + Zgp*(p*p + q*q) - Xgp*p*r - yg*q*r)...
    + F1(3) + F2(3) + F3(3) + F4(3) ;


K = -(yg*W-yb*B)*cos(theta)*cos(phi) - (zg*W-zb*B)*cos(theta)*sin(phi) ...
    + Kpp*p*abs(p) - (Izz-Iyy)*q*r - (m*zg)*w*p + (m*zg)*u*r...
    + Kprop + M1(1) + M2(1) + M3(1) + M4(1);


M = -(Zgp*Wp - Zbp*Bp)*s2 - (Xgp*Wp - Xbp*Bp)*c2*c1...      % PITCH MOMENTS
    + Mww*w*abs(w) + Mqq*q*abs(q) ...
    + Muw*u*w + Muq*u*q + Mpr*p*r...
    + (Izz - Ixx)*p*r...
    - m*(Zgp*(w*q - v*r) + Xgp*(u*q - v*p))...
    + M1(2) + M2(2) + M3(2) + M4(2) ;

N = (Ygp*Wp - Ybp*Bp)*s2 + (Xgp*Wp - Xbp*Bp)*c2*s1...         % YAW MOMENTS
    + Nvv*v*abs(v) + Nrr*r*abs(r) + Nuv*u*v ...
    + Nur*u*r + Npq*p*q...
    + (Ixx - Iyy)*p*q...
    - m*Xgp*(u*r - w*p) + m*Ygp*(w*q - v*r)...
    + M1(3) + M2(3) + M3(3) + M4(3);

FORCES = [X Y Z K M N]' ;


ACCELERATIONS = ...
  [Minv(1,1)*X+Minv(1,2)*Y+Minv(1,3)*Z+Minv(1,4)*K+Minv(1,5)*M+Minv(1,6)*N
   Minv(2,1)*X+Minv(2,2)*Y+Minv(2,3)*Z+Minv(2,4)*K+Minv(2,5)*M+Minv(2,6)*N
   Minv(3,1)*X+Minv(3,2)*Y+Minv(3,3)*Z+Minv(3,4)*K+Minv(3,5)*M+Minv(3,6)*N
   Minv(4,1)*X+Minv(4,2)*Y+Minv(4,3)*Z+Minv(4,4)*K+Minv(4,5)*M+Minv(4,6)*N
   Minv(5,1)*X+Minv(5,2)*Y+Minv(5,3)*Z+Minv(5,4)*K+Minv(5,5)*M+Minv(5,6)*N
   Minv(6,1)*X+Minv(6,2)*Y+Minv(6,3)*Z+Minv(6,4)*K+Minv(6,5)*M+Minv(6,6)*N
   c3*c2*u + (c3*s2*s1-s3*c1)*v + (s3*s1+c3*c1*s2)*w
   s3*c2*u + (c1*c3+s1*s2*s3)*v + (c1*s2*s3-c3*s1)*w
     -s2*u +            c2*s1*v +            c1*c2*w
         p +            s1*t2*q +            c1*t2*r
                           c1*q -               s1*r
                        s1/c2*q +            c1/c2*r] ;

end
%-------------------------------------------------------------------------
```

APPENDIX C: CODE USED FOR ERROR-MINIMIZATION

```
% ele_offsetLRAUV_SIM.M    -   For LRAUV Vehicle Simulator

% ele_offsetLRAUV_SIM : Minimizes error introduced by off-set in elevator
%                       angle data.
%
% Last modified July 21, 2014
% Ben Raanan


clear all
close all

startTime = datestr(clock)
tic
h = waitbar(0,'Initializing error minimization...');

% Load data
fpath = '~/Documents/MATLAB/MBARI/mat/shark/workver/';
filename = [fpath 'LRAUV_SIM_201309301141_201310070703.mat'];

%--------------------------------------------------------------------------
% STATE AND INPUT VECTORS:
% x = [u v w p q r xpos ypos zpos phi theta psi]'
% ui = [ delta_s delta_r Xprop Kprop ]'

[ time, time_step, xstruct, names, controls ] = initialize_LRAUV_SIM( filename
);

timeIn  = datenum(2013,09,30,13,04,10);
timeOut = datenum(2013,09,30,14,28,24);

[~,timeIni] = min(abs(time - timeIn));


% Initiate first step and set runtime
%--------------------------------------------------------------------------
startPoint = timeIni+240*20;
timeEval    = 240; % sec, evaluation run time
n_steps     = fix(timeEval/time_step); %size(ui,1);
n = startPoint:startPoint+n_steps;


% Define global vars
global xg zg Sfin Mqq ARe dCL CDc

zg      =   0.0067940;          % m         Center of gravity
Sfin    =   1.15e-2;            % m^2       Fin area
Mqq     =   0.35*-632.698957;   % kg-m2     Cross-flow drag (Mq|q|)
ARe     =   6.500000;           % n/a     Fin aspect ratio
dCL     =   1.5*4.130000;       % n/a     Coef. of Lift Slope
CDc     =   0.030000;           % n/a     Crossflow Coef. of Drag !!try 0.6!!

mass        =   147.8671;         % kg Flooded Vehicle total mass
movableMass =   26;               % kg Battery movable mass
dropWtMass  =   1.0;              % kg Mass of the drop weight #1, kg
dropWtX     =  -0.1330;           % m  X location of the drop weight #1, m

Xmass = (movableMass.*xstruct.mass_p + dropWtMass*dropWtX)./mass;

% Define range of offsets
%--------------------------------------------------------------------------

tryVal = (-5:0.025:-4.75);

%--------------------------------------------------------------------------
```

```
% hold space
theta_m   = zeros(length(tryVal),n_steps+1);
psi_m     = zeros(length(tryVal),n_steps+1);


% RUN MINIMIZATION
%--------------------------------------------------------------------------
waitbar(0.01,h,'Running error minimization loops...');
for k = 1:length(tryVal)

    % Define error range and resolution
    ele_offset  = tryVal(k)*pi/180;
    rud_offset  = 0*pi/180;

    % Reset control vars
    ui = controls;

    % Correct for offsets in data:
    ui(n,1) = ui(n,1) + ele_offset; % zeros(size(ui(n,1)));
    ui(n,2) = ui(n,2) + rud_offset;

    % Correct for Hysteresis offset
    %{
    for q = 1:length(ui)
        if ui(q,1)<0*180/pi
            ui(q,1) = ui(q,1) - db(k)*pi/180;
        elseif ui(q,1)>0*180/pi
            ui(q,1) = ui(q,1) + 0*pi/180;
        end
    end; clear q
    %}

    % Unpack state vector
    x = zeros(1,12);
    %
    for c=[1:6,9,10:12];
        x(c) = xstruct.(names{c})(startPoint);
    end; clear c
    %}

    % Run simulation
    for i = startPoint:startPoint+n_steps

        % Account for movable mass shift
        xg = Xmass(i);

        % Set some vars constant
             x(1)  = xstruct.u(i);
             x(4)  = xstruct.p(i);
             x(10) = xstruct.phi(i);


        ui_in = ui(i,:);

        [xdot,forces] = lrauv(x,ui_in); % LRAUV


% Log outputted forces
        simlog(i,:) = [x ui_in];
        f(:,i) = forces;

        % RUNGE-KUTTA APPROXIMATION to calculate new states
        % NOTE: ideally, should be approximating ui values for k2,k3
        k1_vec = xdot;
        k2_vec = lrauv(x+(0.5.*time_step.*k1_vec)',((ui(i,:)+ui(i+1,:))./2)) ;
```

```
        k3_vec = lrauv(x+(0.5.*time_step.*k2_vec)', ((ui(i,:)+ui(i+1,:))./2)) ;
        k4_vec = lrauv(x+(time_step.*k3_vec)', ui(i,:)) ;
        x = x + time_step/6.*(k1_vec +2.*k2_vec +2.*k3_vec +k4_vec)';

    end

    theta_m(k,:) = simlog(n,11)';

    waitbar(k/length(tryVal)-0.01,h,['Running error minimization loops... ['...
        num2str(100*k/length(tryVal),2) '%]'] );
    pause(0.1);
end
%-------------------------------------------------------------------------
toc
waitbar(1,h,['Error minimization complete [' num2str(100) '%]'] );

% Compute squared sum of residuals and find minimizing offset value
%-------------------------------------------------------------------------
res = (theta_m - repmat(xstruct.theta(n+lag(median(minlagi))),k,1)).^2;
err = [ sum(res,2), tryVal' ];
[minError, minErrori] = min(err(:,1))

close(h)
%-------------------------------------------------------------------------
```

Published with MATLAB® R2014a