Autonomous Meta-Classifier for Surface Hardness Classification from UAV Landings

Elizabeth Basha[1], Tristan Watts-Willis[1], and Carrick Detweiler[2]

Abstract— **Developing surface classification models manually requires significant time and detracts from the goal of automating systems. We create a system that automatically collects the data using an Unmanned Aerial Vehicle (UAV), extracts features, trains a large number of classifiers, selects the best classifier, and programs the UAV with that classifier. Motivating our work is a prior project [1] that manually developed a surface classifier using an accelerometer; to verify our system functionality, we replicate those results with our new automated system and improve on those results, providing a four-surface classifier with a 75% classification rate and a hard/soft classifier with a 100% classification rate. We further verify our system through a field experiment that collects and classifies new data, proving its end-to-end functionality. Overall, our system reduces the time and machine learning expertise needed by the user to develop new time-series classifiers usable by the UAV. The general form of our system provides a valuable tool for automation of classifier creation and is released as an open-source tool [2].**

I. Introduction

Previous research demonstrated that a unmanned aerial vehicle (UAV) can classify a surface using only its accelerometer sensor [1]. This allows the UAV to determine whether a surface is soft enough to install a sensor node with a spike without breaking the spike; this spike allows the node to stay fixed in place despite erosion and unstable terrain. A key challenge with that work was the need to manually create, train, and implement the classifier as well as manually collect the training data. This process takes weeks of manual tuning and work, limiting efforts to modify the classifier for different conditions. The problem of manual development of the classifier is not isolated to the surface classification problem, but any classification problem. In this paper, we automate the classifier development process, demonstrating our process and validating the results in the context of the UAV surface classification application.

We developed an open-source system (available on BitBucket [2]) consisting of a set of algorithms that collect data using the UAV, compute features, train classifiers, select the best classifier, and implement the classifier system on a UAV. The system automates the process, reducing the amount of time a person needs to spend by orders of magnitude as it only requiring starting the process.

We are grateful to USDA-NIFA 2017-67021-25924, USDA-NIFA 2013-67021-20947, and NSF CNS (CSR-1217400 and CSR-1217428), which partially supported this work.

[1]E. Basha and T. Watts-Willis are with the Department of Electrical and Computer Engineering, University of the Pacific, Stockton, CA 95211, USA `ebasha, twattswi at pacific.edu`

[2]C. Detweiler is with the Department of Computer Science and Engineering, University of Nebraska-Lincoln, Lincoln, NE 68588, USA `carrick at cse.unl.edu`

To use our system, a user defines a set of locations, labeling the locations with the categories. The system does not require the user to know anything about machine learning; it provides default settings such that the user only has to provide labeled locations. For users who want to configure the process, the system allows for optional user configuration of various parameters, including the set and number of features, the set of classifiers, and the definition of a "best" classifier. Once started, the system sends the UAV to the user provided locations to collect training data, which is fed into our tool. The tool then selects the best classifier based on the user goals (e.g. minimizing false positives, minimizing features, etc.). This selected classifier model is programmed on the UAV, which now can use the model to classify unknown locations. While we focus on the application of surface classification in the development of our system, the system uses generalized approaches that allow it to classify any time-series sensor data that the UAV can collect.

To verify our new system, we use the data collected in our prior surface classification work [1] and first compare our results under the same conditions, showing that our process can provide similar results. We then run the full automated system, allowing for different features and classifiers from the prior work, to find the best classifiers through our automated approach. To classify four different surfaces, examining only classification accuracy, we find 5499 classifiers with an accuracy higher than 63.75%, the best accuracy found in our prior work. The best of these classifiers provides a classification accuracy of 75%, a significant increase over the prior work. Classifying two surfaces, our new approach provides a 100% classification accuracy, also improving over the prior work. We also compare our system to Auto-WEKA 2.2 [3], a different form of meta-classifier that de-emphasizes model efficiency; our system provides more efficient models with better classification rates and reduced false positives than the latest Auto-WEKA system. Having proved our approach achieves our goals of efficient models and minimal user interaction, we deploy the complete system in a field experiment that collects and classifies based on new data, thereby validating its use in an end-to-end approach.

The rest of this paper is organized as follows. We start by discussing related work in Section II before describing our system in Section III. Section IV compares our results to our prior work. Section V then describes our field experiment proving the complete operation of our system on new data. Section VI summarizes our work and outlines our future work extending this tool.

II. Background and Related Work

We first provide background on surface classification work and then, given our contribution of a tool for automated classifier creation and use, discuss the related work on automated classifiers with a focus on the use in robotics.

A. Surface Classification

Surface classification is the application area of our work in which we will verify our approach. To better understand it, we discuss the prior work that explored a number of methods for surface classification, covering ground robots, online learning approaches, and aerial robots.

Many of the ground robot approaches that used vibrations for classification also used manually trained classifiers [4], [5], [6], [7], [8], [9], [10]. Some do adaptively modify parameters such as Komma *et al.* who adaptively changed the parameters for their Bayes Filter approach on their ground robot [11]. Otsu *et al.* provided a more automated approach with adaptive co-learning using a camera and a vibration sensor [12]. Dutta and Dasgupta utilized a number of smaller classifiers in an ensemble approach; these classifiers were still manually trained although the approach considered computational complexity [13].

Closer to our motivating application is research utilizing aerial robots. Most of these approaches are camera-based with manually tuned classifiers as demonstrated by Mejias [14]. Mizui *et al.* inspected infrastructure using a hammer on a UAV and measured the sound waves to determine defectiveness [15]. Although this did rely on the UAV striking the system under test, the classification approach was binary thresholding on the frequency response.

In our prior work, Anthony *et al.* used acceleration information with an aerial robot [1] to classify surfaces on landing. The authors manually piloted a UAV over various sample surfaces and collected data samples using the onboard IMU. The authors then manually identified the most useful features and trained several classification algorithms, selecting a decision tree classifier as the best. This paper directly extends and improves that work through our new automation system.

Our system automatically gathers data using a UAV with an onboard accelerometer and processes this data to extract useful features. This system automates the selection of features and classifiers, increasing the amount explored and resulting in a better classification model for the data than other approaches. It also allows for easy re-training and re-selection for new terrains.

B. Automated Classifiers

Different methods exist for selecting a machine learning model, especially for the task of minimizing error by filtering models that underfit or overfit the data. Biem used Hidden Markov Models with a discriminative information criterion to determine the best model [16]. Chapelle *et al.* utilized two methods for model selection: smallest empirical bound and direct eigenvalue estimator [17].

Thornton *et al.* developed Auto-WEKA, a meta-classification algorithm that found the best model using different classifiers with many different parameters [18]. Kotthoff *et al.* extended this work with Auto-WEKA 2.0, an improved version that provided support for regression algorithms, parallelized training of classifiers, saved multiple models, and optimized more metrics [3]. Feurer *et al.* also improved Auto-WEKA by focusing on the efficiency of the underlying optimization problem [19]. They implemented a meta-learning step at the beginning of the process to better prepare the Bayesian optimization algorithm for the various machine learning frameworks; this showed a significant improvement in the performance of the classifiers.

Reif *et al.* developed a regression-based approach to estimate the classifier accuracy based on statistical analysis of the data set [20]. The system trained only the selected classifiers and optimized the parameters for those classifiers. While this approach is fast, one drawback is the lack of feature selection, resulting in a classifier that may not provide the best accuracy possible.

These methods do not consider computational complexity of the models, reducing their utility on a robotic platform. Our system trains many different classifiers over every combination of features and available sensors. This allows us to check the relevance of each feature and each sensor in order to determine the classifier model that has the best trade-off between computational efficiency and classification rate. Additionally, we can configure our system to run on the computational system available, which allows it to run on a robot while in the field.

III. Meta-Classification System

We developed a meta-classification system to autonomously compute features, train classifiers, and select the best classifier based on user-defined criteria to program the aerial robot. This system utilizes a UAV, the sensors already on the UAV, and a computer; these autonomously work together to perform the meta-classification. For the computer, the approach was designed so that the user can configure the system to match the processing power available for anything from an embedded processor like a Raspberry Pi to a high-power desktop system. As of now, the system works with supervised learning, assuming some method of labeling exists (either automated or manual).

In this section, we describe our overall classification system and then discuss the different sub-algorithms.

A. Classification System

Algorithm 1 shows the high-level steps of the full classification system. Our system begins by collecting data from pre-labeled locations using the UAV's onboard sensors. With the collected data, the system computes features. Next, the useful features feed into our classification algorithm, which generates models based on all possible combinations of classifier algorithms and features. For each run, we collect metrics, such as the number of features, the classification rate, and the confusion matrix; after all runs, the system computes which combination worked best. The exact parameters for determining the "best" model are provided to our program

Algorithm 1 Meta-Classification Overview

```
1: {trainingData, verificationData} = CollectData(locations, numSamples)   ▷ Collect and label data

2: featureNames = $\langle \dots \rangle$
3: featureSets = $\emptyset$   ▷ Recursively generate list of all feature combinations
4: for $f \in 1 \dots$ Length(featureNames) do
5:     featureSets += GetFeatureSets(featureNames, $f$)
6: end for
   ▷ Train classifiers, gather statistics, and save information (See Algorithm 2)
7: modelList = BuildAndTestModels(classifierList, featureSets, trainingData, verificationData)
8: SaveModelInformation(modelList)
   ▷ Using supplied parameters, determine best model in modelList (See Algorithm 3)
9: model = GetBestModel(modelList, selectionParameters)

10: ProgramUAV(model)   ▷ Program the best classifier model onto the UAV
```

Algorithm 2 BuildAndTestModels

```
1: modelList = $\emptyset$
2: for $c \in$ classifierList do
3:     for $f \in$ featureSets do   ▷ Remove unused features from data set and train the classifier
4:         data = GetDataSetWithFeatures(trainingData,$f$)
5:         model = BuildClassifier($c$,data)   ▷ Run cross-validation on model and measure time taken
6:         evalTime = time()
7:         evaluation = CrossValidateModel(model,data)
8:         evalTime = time() $-$ evalTime
9:         falsePositives = GetFalsePositives(evaluation.confusionmatrix)
10:        if verificationData then   ▷ Evaluate model with optional verification data
11:            verification = TestModel(model,verificationData)
12:        end if   ▷ Store information on classifier and performance
13:        modelList += $\langle c, f,$ evaluation.accuracy, falsePositives, evalTime, verification.accuracy$\rangle$
14:    end for
15: end for
```

on launch. Finally, the best classifier is then programmed on the UAV to classify new locations.

B. Data Collection

The first step of Algorithm 1 is the data collection algorithm, shown on Line 1. Initially, the user needs to provide labeled locations that match the classification categories. With these locations, the UAV flies to each, measures the appropriate information, and repeats to collect the data set. The system divides this data set into training and validation. While the system does use cross-validation when training, the additional and separate validation phase reduces the likelihood of selecting an overfit model. These two data sets then progress to the feature computation stage.

C. Feature Computation

Our system's next algorithm extracts features and selects the appropriate data sets as Algorithm 1 describes in Lines 2-6. The data we are examining is time series data; therefore, we perform feature extraction to acquire several key indicators. Examples of possible features include maximum/minimum, second maximum/minimum, peak frequency, variance, and skew [1]. The user configures the set of features to test and our approach will prune features that are not useful. This means that, if the computational resources are available, the user can comfortably add many features to better explore the model options in cases of uncertainty or initial investigations.

Once all of the features are extracted and the tagged data is loaded, we generate a set of all possible feature combinations. By default, the program will generate all feature combinations. As the number of combinations scales rapidly, to reduce computation time, the user can specify a limit to the maximum number of features. The number of combinations generated is given by

$$\sum_{i=1}^{\min(n,m)} \frac{m!}{i!(m-i)!}$$

where n is the user specified maximum number of features and m is the total number of features in the data. While computation time scales rapidly with the number of features, in practice, using eight features takes 19 minutes on a desktop system and 321 minutes on a Raspberry Pi (see Section V-B for details). This is sufficient for most systems we are interested in.

D. Build Classifier

With the feature set computed, our system now trains all of the classifiers using all feature sets, shown in Lines 7-8 of Algorithm 1. Algorithm 2 outlines the key components of this step. From the prior algorithm, this algorithm receives the training data, the verification data, and the classification categories. The user defines the maximum number of features, the minimum number of features, and the category to use for false positives (whichever category could most damage the UAV if misclassified). Optionally, the user can define the number of cross-validation folds, the maximum allowable number of false positives, and the allowable drop in classification percentage in order to find the result with the smallest number of false positives or smallest number of features. All of these parameters have reasonable default settings so the user can optionally ignore the parameters.

Having defined the necessary parameters, we now consider the classification algorithms. We use several classification algorithms, including: (1) Discriminant Analysis, (2) Support Vector Machines, (3) Decision Trees/Forests, (4) Logistic Regression, (5) Naive Bayes, and (6) Multilayer Perceptron Neural Network. This is only a subset; the user can include many others in the system easily by modifying the configuration text file that defines our classifier list. These provide a diversity of algorithm types, allowing our meta-classifier to generate the best possible model. To implement each algorithm, we use the WEKA machine learning workbench [21]; a different machine learning library could easily be used in our system such as Python scikit-learn [22].

We loop through these classifiers and, for each classifier, we generate a model for all combinations of features using the WEKA default parameters. To utilize non-default parameters, we can modify the configuration text files as we do to define the classifier list. Adding the ability to perform parametric sweeps of the various parameters within our system is future work along with quantifying the increased runtime.

In the WEKA implementation, we use the functions *BuildClassifier*, *CrossValidateModel*, and *TestModel* for generating and evaluating classifiers. *BuildClassifier* accepts a classifier algorithm, data set, and optional parameters and outputs a usable model (see Lines 4-5 in Algorithm 2). *CrossValidateModel* (Line 7) accepts a model and data set, performs *n*-fold cross-validation, and then returns performance metrics, such as the confusion matrix. *TestModel* (Line 10) evaluates a model and returns performance metrics without validation.

The model is evaluated on the training data as well as the separate verification data, if provided by the user (Line 11). We save the evaluation results for each data set such as classification rates and confusion matrices for further processing.

E. Classifier Selection

Once all of the models have been trained and evaluated, we rank them by classification rate and filter based on user inputs. Algorithm 3 describes the selection process. First, the user can specify a false positive rate limit. If the user sets this, the program will filter out models that have equal or less than the false positive limit; otherwise, the program returns all models. Then the program selects all models within that set that have the highest accuracy. Additionally, if the user specifies a maximum allowable percentage drop, the program will add those models within that accuracy range of the best classification rate. Finally, the algorithm selects the model in the remaining set that has the least number of features and lowest computational time.

F. Program UAV

The classifier models are represented by small configuration files, typically less than 50 KB. This enables easy storing and transmission of the models between robots and optional base stations. In our current implementation, we send the model to the UAV via SCP (Secure Copy) over Wi-Fi when using a base station for computation or we can perform computations on an onboard Raspberry Pi.

IV. Algorithm Validation

To validate our system and approach, we compare our system's results to the results of the motivating paper [1]. We start by reviewing the results of the motivating work, our results replicating their work on the same data, and our improved results using the full abilities of our system.

A. Motivating Results

Anthony *et al.* previously developed surface classification algorithms for UAVs [1]. This process was highly manual and motivates our goal of automating classifier selection.

In their experiments, the authors recorded accelerometer data for landings on four types of indoor surfaces (wood, carpet, rubber tile, and foam). They then extracted features based on several common time series predictors, a sub-set of those we use. Using MATLAB, the authors manually created and trained models using Decision Trees, Linear Discriminant Analysis (LDA), Quadratic Discriminant Analysis (QDA). This required weeks of work to manually compare results, to pick parameters, and to adjust parameters.

They trained models in two different ways for each set of data. The first classification had four categories of indoor surfaces; the second classified two categories of hard or soft. Table I shows the results of these trials.

B. Our System's Replication Results

We began by replicating the original results. Using our system, we first performed classification using LDA, QDA, SVM, and Decision Trees on the set of all original features. We also performed the second experiment from the prior paper to classify surfaces as hard or soft, again training our classifiers using all 11 original features. Table I shows how our results compare to those of the original paper.

Our results are within reasonable ranges of the original paper's results. Overall, both approaches select the decision tree algorithm for both experiments (the original paper selected it for all surfaces due to its computational simplicity even though it performed slightly worse than LDA). Our system took minutes for the user to setup and all remaining steps were automated, while the prior work took the user weeks of direct work, showing the clear improvement our system provides.

Algorithm 3 GetBestModel

```
1: modelList = {$m \in$ fullModelList | $m$.falsePositives $\leq$ selectionParameters.falsePositives}    ▷ Filter list of models by false positive rate
2: maxAccuracy = $\max(m$.accuracy | $m \in$ modelList$)$    ▷ Determine highest accuracy model
3: maxAccModels = {$m \in$ modelList | $m$.accuracy = maxAccuracy}
4: maxAccuracyModel = $sort($maxAccModels, $m \rightarrow m$.numFeatures$)[0]$
5: highAccModels = {$m \in$ modelList | maxAccuracy − $m$.accuracy $\leq$ selectionParameters.percentDrop}
6: minFeatures = $\min(m$.numFeatures | $m \in$ highAccModels$)$    ▷ Find minimum feature count of available models
7: minFeatureModels = {$m \in$ highAccModels | $m$.numFeatures = $minFeatures$}
8: fastestModels = $sort($minFeatureModels, $m \rightarrow m$.classifyTime$)$
9: bestModel = $sort($fastestModels, $m \rightarrow m$.numFeatures$)[0]$    ▷ Output model with lowest time and fewest features
```

	All Surfaces		Hard/Soft	
Classifier	Classification Accuracy	False Positives	Classification Accuracy	False Positives
ICRA Decision Tree	62.5%	5%	97.5%	5%
Our Decision Tree	68.75%	4.8%	97.5%	5%
ICRA LDA	63.75%	0%	93.75%	0%
Our LDA	68.75%	4.8%	97.5%	9.1%
ICRA QDA	58.75%	5%	97.5%	5%
Our QDA	53.75%	0%	93.75%	0%
ICRA SVM	41.25%	35%	88.75%	45%
Our SVM	36.25%	0%	75%	0%

TABLE I

SUMMARY OF INDOOR TESTING RESULTS FROM ORIGINAL ICRA 2015 PAPER [1] AND OUR RESULTS

C. Our System's Improved Automated Results

Upon verifying that we were seeing similar results under the same conditions, we ran the data through our meta-classification system to generate the best model. For this test, we defined three different "best" models: (1) the model with the best classification accuracy, (2) the model with the lowest false positive rate, and (3) the model using the fewest features with the lowest false positive rate that was within 5% of our highest classification accuracy model (for example, if the most accurate model is 80%, this would be the model with least features with an accuracy within 76% to 80%).

We began by classifying all surfaces, for which we generated 22517 different classifier models. The manual method from the paper resulted in four classifier models; the best model for all surfaces had a classification accuracy of 62.5% with 5% false positives. Figure 1 compares these four models to all of our models. This histogram represents the number of classifier models providing that classification accuracy and shows that 5499, or 24.4%, of our models provide better results than the best one presented in the original paper. Table II outlines all three of our best models in comparison to the original paper's best model. Each results in a 12% or higher classification accuracy with the QDA model also avoiding misclassifications and only requiring three features, which would reduce the processing on the data to enable the UAV to compute results faster. We also compare our results to Auto-WEKA 2.2 [3], which is a commonly used system to automatically find a classifier; that system runs for a defined length of time, which we set to the same amount of time our system required to compute our results. Our system finds a classifier with a better classification accuracy, less false positives, and a reduced number of features.

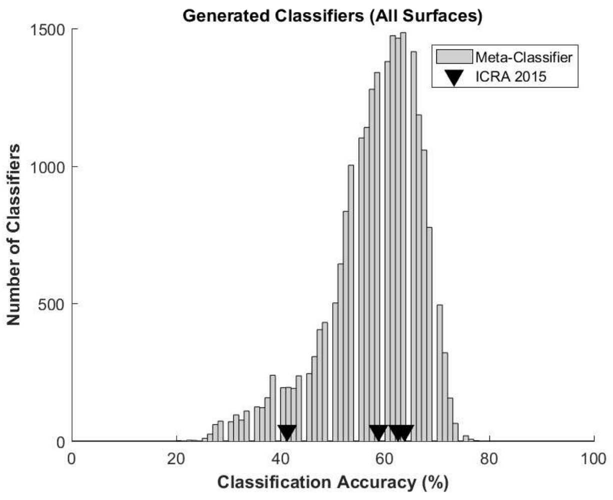

Fig. 1. Histogram of classification accuracy of all generated classifiers for all surface data, showing the classifiers from [1] with red triangles

We next examined the reduced complexity hard versus soft data set; Table II shows these results as well. The original paper concluded that a Decision Tree model worked best, resulting in a classification accuracy of 97.5% with soft surfaces being misclassified 5% of the time. Our three best models are all the same Logistic model providing a perfect classification accuracy based on only two features. Again, comparing to Auto-WEKA 2.2, our approach provides an efficient and accurate classifier. Overall, automating the process and focusing on computational complexity allows

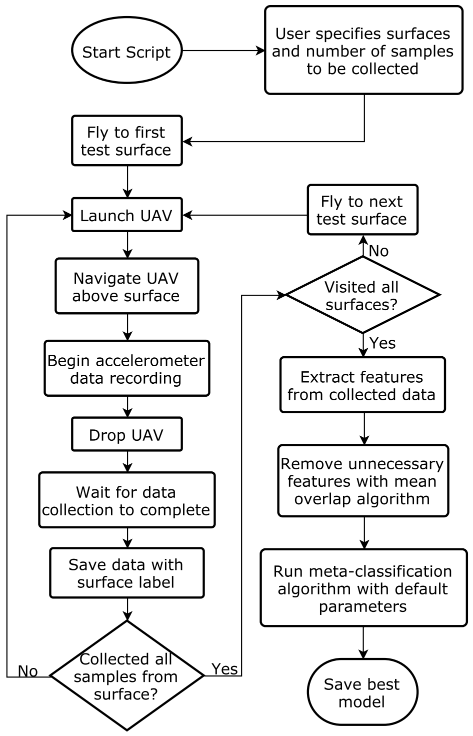
Start Script
User specifies surfaces and number of samples to be collected
Fly to first test surface
Launch UAV
Navigate UAV above surface
Begin accelerometer data recording
Drop UAV
Wait for data collection to complete
Save data with surface label
Collected all samples from surface?
No
Yes
Visited all surfaces?
No
Yes
Fly to next test surface
Extract features from collected data
Remove unnecessary features with mean overlap algorithm
Run meta-classification algorithm with default parameters
Save best model

Fig. 2. Flowchart describing data collection procedure for meta-classifier

for determination of an improved and more efficient model, which reduces the computational time and energy the UAV requires to classify new locations.

V. Field Experiment

After ensuring the classification aspect of our system functions, we implement the complete system and perform a field experiment using a AscTec Hummingbird [23] and a Raspberry Pi 3 Model B [24] (Pi).

A. Experiment Setup

We have the Raspberry Pi aboard the UAV with an accelerometer. Data samples are collected from preset locations for different surface types. Figure 2 describes the experiment procedure. Our program on the Pi instructs the UAV to fly to these locations, to free-fall from 20cm while the Pi records three seconds of data, and then fly to the next sample location until all samples are collected. Our system extracts the required features from the samples and then begins our meta-classification algorithm to generate the best model. Once the model has been generated, our system sends the model to the Pi for use on the UAV. The UAV then collects data from both the original training surfaces and two additional surfaces, classifying the surface immediately after data collection.

The training data set for the classifier is taken from 160 samples, split evenly between a *soft* pillow and *hard* tile. For verification of the model, the UAV recorded an additional 20 samples from both surfaces. The system also recorded 20 samples of an alternate *hard* surface (a lab table) and 20 samples of an alternate *soft* surface (a different pillow) to test the classifier on surfaces upon which it was not trained.

B. Results

Our meta-classification system was configured to minimize both the required number of features and the occurrences of false *soft* classifications. We limited the number of features to eight to reduce the number of generated models and decrease runtime. On our computer, a 3.9GHz i7 4770 with 8GB RAM, this process took approximately 19 minutes. On our Raspberry Pi, a 1.2GHz ARMv8 with 1GB RAM, this process took approximately 321 minutes.

In total, the algorithm trained and evaluated 23760 classifiers. Ten of the models result in a classification accuracy of 100% on the training set. Of these ten, four of the models used the LDA classifier while the remaining six used the Logistic classifier. Between these, the system chose one of the Logistic classifiers as it is the classifier with the highest classification accuracy and the fewest required input features.

The chosen Logistic model uses four features from the accelerometer z-axis (vertical axis of the UAV): (1) minimum, (2) variance, (3) kurtosis, and (4) mean. Each verification data set was separately classified using the chosen model to populate Table III. Running the classifier for one test on the Pi required 0.38ms. The verification set for the training surfaces resulted in 90% classification accuracy. Only four of the 20 samples were misclassified and, of these, only one sample was falsely classified as *soft*.

After verifying the model was accurate on our training surfaces, we tested to ensure the UAV could classify other surfaces correctly. All 20 alternate *hard* surface samples were classified correctly. For the alternate *soft* surface, eighteen samples were correctly classified with only two of the 20 samples on this surface misclassified as *hard*.

C. Analysis

This experiment verified the full system from data collection to classification of new surfaces. The UAV successfully performed automated classification and chose a model that performed well on new surfaces.

VI. Conclusion

We developed an automated system that allows a UAV to collect data, train and select the best classifier, and then use that classifier on new surfaces. The system minimizes the user's time and requires no machine learning experience while also allowing the UAV to perform the work in the field without user interaction. For those with machine learning expertise, the system parameterizes the number of features, the number of classifiers, and different best models, allowing the system to optimize performance based on computation and energy constraints. This automation also allows the system to re-select models when the current one no longer provides reasonable results, which allows for in-field automated reprogramming of the system to improve results. While our experiments focused on surface classification, the system can classify any time-series data that the UAV can measure, enabling a wide variety of applications. Our system is publicly available on BitBucket [2] in order to support the community in performing these classifications.

		Classifier	Classification Accuracy	False Positive Rate	Number of Features
All Surfaces	ICRA	LDA	63.75%	0%	11
	Best Accuracy	RandomSubSpace	77.5%	9.1%	7
	Best False Positive	QDA	75%	0%	3
	Fewest Features	QDA	75%	0%	3
	Auto-WEKA	RandomForest	70%	4.8%	11
Soft/Hard	ICRA	Decision Tree	97.5%	5%	11
	Best Accuracy	SimpleLogistic	100%	0%	2
	Best False Positive	SimpleLogistic	100%	0%	2
	Fewest Features	SimpleLogistic	100%	0%	2
	Auto-WEKA	RandomForest	98.5%	9.1%	11

TABLE II

COMPARISON OF BEST ICRA CLASSIFIER TO BEST OF SEVERAL CATEGORIES FROM META RESULTS

	Training Set		Verification Set			
Classification	Pillow (Soft)	Tile (Hard)	Pillow (Soft)	Tile (Hard)	Table (Hard)	Second Pillow (Soft)
Soft	80	0	17	1	0	18
Hard	0	80	3	19	20	2

TABLE III

CLASSIFICATION RESULTS FOR EACH SURFACE TYPE BASED ON CHOSEN LOGISTIC MODEL

Our system does still rely on labeled locations; we plan to address this in future work. We also plan to add a metric for the computational complexity of the classification model to include that in selection of the best model. Ideally, the system, while in the field, can adapt to the computational and energy constraints of the UAV and other robotic systems working to measure and understand the environment.

REFERENCES

[1] D. Anthony, E. Basha, J. Ostdiek, J.-P. Ore, and C. Detweiler, "Surface classification for sensor deployment from uav landings," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 3464–3470.

[2] "University of the Pacific's meta-classifier open tool repository," 2017, [Accessed: Feb. 27 2017]. [Online]. Available: https://bitbucket.org/pacific_ecpe/pacific_metaclassifier

[3] L. Kotthoff, C. Thornton, H. H. Hoos, F. Hutter, and K. Leyton-Brown, "Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka," *Journal of Machine Learning Research*, vol. 17, pp. 1–5, 2016.

[4] E. Coyle and E. G. Collins Jr, "A comparison of classifier performance for vibration-based terrain classification," DTIC Document, Tech. Rep., 2008.

[5] E. M. Dupont, C. A. Moore, E. G. Collins, and E. Coyle, "Frequency response method for terrain classification in autonomous ground vehicles," *Autonomous Robots*, vol. 24, no. 4, pp. 337–347, 2008.

[6] P. Giguere and G. Dudek, "Surface identification using simple contact dynamics for mobile robots," in *IEEE International Conference on Robotics and Automation*. IEEE, 2009, pp. 3301–3306.

[7] D. Sadhukhan, C. Moore, and E. Collins, "Terrain estimation using internal sensors," in *Proc. of the IASTED Int. Conf. on Robotics and Applications*, 2004.

[8] K. Sullivan, W. Lawson, and D. Sofge, "Fusing laser reflectance and image data for terrain classification for small autonomous robots," in *International Conference on Control Automation Robotics & Vision (ICARCV)*. IEEE, 2014, pp. 1656–1661.

[9] A. Vicente, J. Liu, and G.-Z. Yang, "Surface classification based on vibration on omni-wheel mobile base," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 916–921.

[10] X. A. Wu, T. M. Huh, R. Mukherjee, and M. Cutkosky, "Integrated ground reaction force sensing and terrain classification for small legged robots," *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 1125–1132, 2016.

[11] P. Komma, C. Weiss, and A. Zell, "Adaptive bayesian filtering for vibration-based terrain classification," in *IEEE International Conference on Robotics and Automation*. IEEE, 2009, pp. 3307–3313.

[12] K. Otsu, M. Ono, T. J. Fuchs, I. Baldwin, and T. Kubota, "Autonomous terrain classification with co-and self-training approach," *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 814–819, 2016.

[13] A. Dutta and P. Dasgupta, "Ensemble learning with weak classifiers for fast and reliable unknown terrain classification using mobile robots," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. PP, no. 99, pp. 1–12, 2016.

[14] L. Mejias, "Classifying natural aerial scenery for autonomous aircraft emergency landing," in *International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2014, pp. 1236–1242.

[15] M. Mizui, I. Yamamoto, S. Kimura, and M. Maeda, "Research on hammering test system by unmanned aerial vehicles for infrastructure surveillance," in *International Symposium on Experimental Robotics*. ISER, 2016.

[16] A. Biem, "A model selection criterion for classication: Application to hmm topology optimization," in *Proceedings of the Seventh International Conference on Document Analysis and Recognition (ICDAR03)*. IEEE, 2003, pp. 104–108.

[17] O. Chapelle, V. Vapnik, and Y. Bengio, "Model selection for small sample regression," *Machine Learning*, vol. 48, no. 1-3, pp. 9–23, 2002.

[18] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Auto-weka: Combined selection and hyperparameter optimization of classification algorithms," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2013, pp. 847–855.

[19] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," in *Advances in Neural Information Processing Systems*, 2015, pp. 2962–2970.

[20] M. Reif, F. Shafait, M. Goldstein, T. Breuel, and A. Dengel, "Automatic classifier selection for non-experts," *Pattern Analysis and Applications*, vol. 17, no. 1, pp. 83–96, 2014.

[21] G. Holmes, A. Donkin, and I. H. Witten, "Weka: A machine learning workbench," in *Second Australian and New Zealand Conference on Intelligent Information Systems*. IEEE, 1994, pp. 357–361.

[22] scikit-learn, "scikit-learn," 2017, [Accessed: July 20 2017]. [Online]. Available: http://scikit-learn.org/stable/

[23] "AscTec Hummingbird," 2017, [Accessed: Feb. 27 2017]. [Online]. Available: http://www.asctec.de/en/uav-uas-drones-rpas-roav/asctec-hummingbird/

[24] "Raspberry Pi 3 Model B," 2016, [Accessed: Feb. 27 2017]. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/