Aircraft Weight Estimation During Take-off Using Declarative Machine Learning

Sinclair Gurny, Jason Falvo, and Carlos Varela
Department of Computer Science, *Rensselaer Polytechnic Institute*
{gurnys@rpi.edu, falvoj@rpi.edu, cvarela@cs.rpi.edu}

***Abstract*—Aircraft sensors measure physical quantities to help pilots and flight automation systems with situational awareness and decision making. Unfortunately, some important quantities of interest (QoI), *e.g.*, aircraft weight, cannot be directly measured by sensors. This may lead to accidents, exemplified by Tuninter 1153 and Cessna 172R N4207P, where the airplanes were underweight (not enough fuel) and overweight (6% over maximum gross weight) respectively. Learning models to infer QoI from other aircraft sensor data is thus critical to safety through analytical redundancy. In this paper, we extend PILOTS, our declarative programming language for stream analytics, to learn models from data. We illustrate the supervised machine learning extensions to PILOTS with an example where we use take-off speed profiles under different density altitudes and runway conditions to estimate aircraft weight. Using data collected from the X-Plane flight simulator for a Cessna 172SP, we compare the results of several models on accuracy and timeliness. We also consider ensemble learning to improve the accuracy of weight estimation during takeoff from 94.3% (single model) to 97% (multiple models). Given that the average length of a take-off is 26.75s, this model was able to converge within 10% of the correct weight after 10.7s and converge within 5% after 17.7s. On August 25th, 2014, a Cessna 172R, N4207P, crashed killing the pilot and three passengers. The National Transportation Safety Board (NTSB) report calculated the aircraft to be 1.06 times the maximum gross weight. We simulated the take-off in X-Plane using information from the report. We were able to estimate within 5% error after 8s, which is less than 200ft down the runway, and at the point of take-off, 27s, had an error of 3%. This implies that our model could have alerted the pilot of an overweight condition well before the aircraft became airborne, leaving more than 2000ft of runway to come to a stop. If this system were to be implemented in any fixed wing aircraft, it would create a larger safety net. Pilots would have a greater chance of catching errors thus increasing the probability of survival for crew and passengers.**

I. Introduction

More mobile devices than ever are producing data, from cellphones to cars, drones, and aircraft. Data from aircraft flight recorders can provide clues into how an accident occurred and how it could have been prevented. Aircraft sensors measure physical quantities to help pilots and flight automation systems with situational awareness and decision making. Unfortunately, some important quantities of interest (QoI), *e.g.*, aircraft weight, cannot be directly measured from sensors. As a consequence, accidents can happen, exemplified by Tuninter Flight 1153 and the crash of Cessna 172R N4207P[1], where the airplanes were underweight (2000kg less fuel than expected) and overweight (6% over maximum gross weight) respectively.

Take-off and landing are the most dangerous phases of flight because the most incidents occur during these times [1]. However, early into take-off is also the safest time to abort an overweight flight. In large aircraft, V_1 is the take-off decision speed: the speed above which take-off should not be aborted. If anything occurs before the aircraft reaches V_1, the aircraft can safely abort and come safely to a stop on the runway. However, if anything occurs after V_1, the aircraft must try to continue the take-off. During take-off, there is increased lift due to ground effect. This means that an aircraft that is able to take-off may not necessarily be able to operate properly after the ground effect. Therefore, being able to detect dangerous conditions, such as being overweight, during take-off is critical to maximize safety.

The physics-based approach to estimating aircraft weight during take-off is based on modeling the forces acting on the aircraft. The difficulty in this approach is that the rolling resistance depends on the coefficient of friction between the landing gear and the runway surface as well as the force being applied to the runway. The characteristic of different runways, such as presence of water, oil, or characteristics of the aircraft such as tire wear and tire pressure can impact the coefficient of friction. Most importantly, calculating the force on the runway requires knowing the weight, which causes a circular dependency. However, using our data-driven approach does not require explicitly modelling the physics, including aerodynamics and friction. Machine learning allows for learning models directly from data.

While general-purpose programming languages (*e.g.*, Java or Python) can be used to implement machine learning algorithms, we advocate a declarative approach. Declarative programming is a style which focuses on what a computation should do while abstracting away details of how it should be done. Declarative programming is more concise and direct which is helpful for data scientists who might not have a strong programming background. Removing unnecessary details of control flow makes declarative programs faster to write and allows for higher levels of reasoning on programs. Declarative programming more resembles mathematical logic than the low-level operations of a language like C. Creating a declarative machine learning interface allows for significantly faster prototyping of systems created by users who understand the domain but may not understand the intricacies of differ-

[1] see NTSB report CEN14FA453

978-1-7281-9825-5/20/$31.00 ©2020 IEEE

ent machine learning algorithms or their implementation in general-purpose programming languages.

This paper extends the declarative PILOTS programming language [2]. We advance upon the work of Imai *et al.* [3] by implementing and extending the declarative supervised training grammar to maximize extensibility and ease of use. We use this system to build a model which can estimate fixed-wing aircraft weight in real-time during take-off.

The contributions are as follows:

- Implemented a declarative grammar to train and test machine-learning models in a simple and model-agnostic method.
- Developed system which allows for the creation and integration of user-written machine learning algorithms into data-driven applications.
- Created a data-driven aircraft weight estimation model that can be used during take-off to maximize safety. We also illustrate how this model could have prevented the crash of Cessna 172R N4207P.

The rest of this paper is organized as follows. Section 2 details a brief background on the PILOTS programming language. In Section 3 we discuss the declarative machine learning system within PILOTS. Section 4 illustrates our data-driven method for aircraft weight estimation during take-off as well as experimental results. Related work is discussed in Section 5. Lastly, section 6 concludes the paper.

II. Background

PILOTS[2] is a ProgrammIng Language for spatiO-Temporal data Streaming especially designed for building data-driven applications on systems such as airplanes where different sensor measurements can be viewed as correlated (in space and time) streams [4]. PILOTS is a declarative language with a syntax similar to Pascal. Users of PILOTS can specify how to use spatio-temporal data streams to produce other data streams. PILOTS programs define the high-level operations showing how the output data streams are based on the input data streams. As a data stream processing language, PILOTS is unique because of its first-class support for data selection, data interpolation, and error recovery. Previous work focuses on the ability of PILOTS to detect and recover from sensor failures, such as the work by Imai, Galli, and Varela [5] to detect weight discrepancy in level-flight from data of Tuninter 1153. Imai, Hole, and Varela use multiple models of analytical redundancy (i.e. relationships between data streams) to detect multiple simultaneous sensor type failures [6].

Figure 1 shows `Twice`, a simple PILOTS program[3]. The program has two input data streams $a(t)$ and $b(t)$ and one output data stream, e, which outputs every second. The input data stream b is supposed to be twice the value of a, which are both functions of time. The output data stream e outputs the error of how far a and b are from the correct values, defined as $b - 2 * a$.

[2]PILOTS is open source and available on http://wcl.cs.rpi.edu/pilots/

[3]PILOTS version 0.6: https://github.com/RPI-WCL/pilots/releases/tag/v0.6

```
program Twice;
  inputs
    a(t) using closest(t);
    b(t) using closest(t);
  outputs
    e: b - 2 * a at every 1 sec;
end;
```

Fig. 1: Simple PILOTS program Twice.

III. Declarative Supervised Learning

Oftentimes, finding relationships between data streams can be extremely difficult without advanced domain knowledge. However, these relationships can also be inferred directly from data using machine learning. This approach is useful for its flexibility and rapid development.

This section will detail the grammar for training machine learning models as well as architecture the machine learning component of PILOTS. We will also discuss briefly the way in which new machine learning algorithms can easily be added into PILOTS.

A. Trainer Language Abstraction

We have implemented the grammar, see Figure 2, for training machine learning models, originally proposed by Imai *et al.* [3], making the system more modular and extensible for all users so that more machine learning algorithms can be easily incorporated. The PILOTS grammar is extended with a new non-terminal *Trainer* which abstracts over a declarative and extremely simple interface for supervised and potentially other training of machine learning algorithms. The syntax of trainers is simple and consistent for every algorithm, allowing for the programmer and data scientist to focus on the application easily trying different machine learning algorithms.

Machine learning models, which are mathematical representations of a real-world process, can either be trained offline or online. Offline training is completed before the use of the model in the application. Whereas online training is done in real-time while the model is being run. A model can be both trained in the offline phase while being updated during the online phase. The difference is in the implementation of the model and not in the syntax of its use. PILOTS trainers are used to train offline models as well as to create online models.

There are three main sections of a PILOTS trainer file: `constants`, `data`, and `model`. `constants` is an optional section that specifies constants. The `data` section specifies where data is collected to be sent to the model for training and validation. There are two ways to collect data: using previously created models and files. Comma-separated values (csv) files are the most common input method; the grammar also supports the selection of specific columns. Lastly, the model section specifies the data to be used to train, and possibly validate the model as well as specify any settings of the models. The `features` and `labels` subsections specify the inputs and the correct output values for the corresponding input. The model uses this information to train internal parameters. There

```
Trainer          ::=  trainer Var;
                        [constants Constants]
                        data
                            Data
                        model
                            Features
                            Labels
                            [Test_Features]
                            [Test_Labels]
                            Algorithms
                      end
Constants        ::=  [(Constant; )* Constant];
Constant         ::=  Var = Exp
Data             ::=  [(DataItem; )* DataItem];
DataItem         ::=  Vars using (Exp|ModelUser)
File             ::=  file'('String')'
ModelUser        ::=  model'('Vars')'
Features         ::=  features: Exps;
Labels           ::=  labels: Exps;
Test_Features    ::=  test_features: Exps;
Test_Labels      ::=  test_labels: Exps;
Algorithms       ::=  algorithm: [(Algorithm; )* Algorithm];
Algorithm        ::=  Var ['('Map')'];
Map              ::=  MapItem|MapItem, Map
MapItem          ::=  Var ':' (Number|Var|Exp)
Exps             ::=  Exp|Exp, Exps
Exp              ::=  Func(Exps)|Exp Func Exp|
                      '(' Exp ')' |Value
Number           ::=  Sign Digits|Sign Digits'.'Digits
Sign             ::=  '+' | '-' | ''
Integer          ::=  Sign Digits
Digits           ::=  Digit|Digits, Digit
Digit            ::=  {0, 1, 2, ..., 9}
Vars             ::=  Var|Var, Vars
Var              ::=  {a, b, c, ...}
String           ::=  {"a", "b", "c", ...}
```

Fig. 2: PILOTS trainer grammar.

is also an optional `test features` and `test labels`, which can be used to validate the accuracy of a trained model without needing to create another program to test it. Validating the model can ensure that the model has not over-fit on the training dataset. The last subsection of the model specifies the `algorithm` to use as well as specify any arguments it may take. The arguments can signify settings such as the learning rate of a neural network or what method of error calculation to use. Furthermore, if multiple models are listed then each of the models are trained and validated so the user can see which model resulted in the best performance on the given data.

B. PILOTS Architecture

Figure 3 shows the connection between the Machine Learning (ML) component and other PILOTS programs. Within the ML component, there are three types of files used to perform the operations: training data, algorithm implementations, and trained models. Training data consists of data used by models to update internal parameters to reduce error in prediction. Model code files are implementations of machine learning algorithms (currently written in Python) using a specific Application Programming Interface (API) so it can used by the ML component. Lastly, trained models are serialized models that can be used by PILOTS programs to predict values. In summary, trainer files take model code files, pass them training

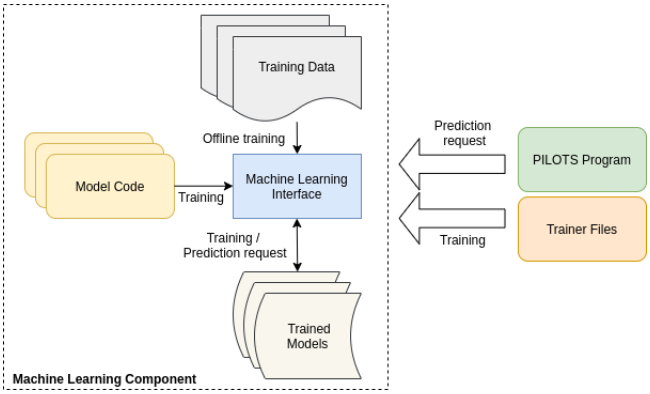


Fig. 3: PILOTS and machine learning interface.

data, and compile them into a trained model file which can be used by other PILOTS programs.

The machine learning component consists of an HTTP server which responds to requests from the PILOTS trainers and PILOTS programs.

Model code is the term used to describe the Python implementation of the machine learning algorithms. As long as the Python source code implements certain API functions it is considered a valid model code file. This means any type of Python library or program with Python bindings can be used within PILOTS with ease.

Model files are the serialized representation of a trained model. They are created as the result of running a trainer file and can be used by any PILOTS program.

The PILOTS compiler translates PILOTS programs and trainer files into Java for platform independent execution. The compiler consists of two parts: a parser and a code generator, each of which has a version meant for standard PILOTS programs and one for trainer files. The parser uses JavaCC[4] to convert the grammar into an abstract syntax tree, which the code generator then converts into a valid Java program.

Once a trainer file is compiled into the Java source file, it is compiled using the standard Java compiler. When the program is run in conjunction with the machine learning component server, it creates a model file which can be used in other trainer files or PILOTS applications.

C. Applications

A simple example of a program that uses a machine learning model can be seen in Figure 4: the Twice program but using a linear regression model to estimate the b data stream [3]. It uses the a data stream as input to the "twice_model". The other input data stream, b, is twice the value of a and the output data stream o outputs the error of the model's estimate of b. This model was trained using the trainer file shown in Figure 5.

This trainer program trains a linear regression model using the columns a and b from a training dataset called "twice_training.csv". In this data, b, is approximately twice

[4]https://javacc.github.io/javacc/

```
program prediction_twice;
  inputs
    a, b (t) using closest (t);
    b_prime using model(twice_model, a);
  outputs
    o: b - b_prime at every 1 sec;
end
```

Fig. 4: PILOTS prediction_twice program.

```
trainer twice_model;
  data
    a, b using file("twice_training.csv");
  model
    features: a;
    labels: b;
    algorithm:
      LinearRegression(OrdLeastSq: true);
end;
```

Fig. 5: PILOTS twice_model trainer program.

that of a. The linear regression model was also specified to use the ordinary least squares method.

A more advanced trainer which trains a neural network to learn a three input XOR gate can be seen in Figure 6. It collects columns a, b, c, and x from the "data" file, where they are listed as columns "A", "B", "C", and "Q" in the file. Also $i1$, $i2$, $i3$, and o are collected from the "xor" file. The three neural network models using slightly differing settings are trained using a, b, and c as inputs and x as the output. Each model is also validated with the testing data given by $i1$, $i2$, $i3$, and o. The program will return the final results of training all of the models so that the user can see which model was most effective on the dataset. Training multiple models is no different in PILOTS which means that users can quickly try multiple models to see what works best on the data. These four models resulted in final accuracies, which may vary each time the model is trained, of 90.4%, 0.5%, 99.9%, and 1.1%, respectively. These results shows several things very quickly: that the hyperbolic tangent and logistic activation functions severely decrease the accuracy of the model (comparing models 1 to 2, and models 3 to 4) and that increasing from one layer to two increased overall accuracy (comparing model 1 to 3).

IV. Aircraft Weight Estimation During Take-off

A. Background

During take-off an aircraft is accelerating due to thrust and being slowed from friction with the air and the runway. Knowing these forces and the aerodynamic properties of the aircraft we can calculate the mass using Newton's second law of motion.

$$m = \frac{T - D - F}{dV/dt} \tag{1}$$

```
trainer xor_model;
  data
    a, b, c, x using file("tr_data.csv",
                          A, B, C, Q);
    i1, i2, i3, o using file("xor.csv");
  model
    features: a, b, c;
    labels: x;
    test_features: i1, i2, i3;
    test_labels: o;
    algorithm:
      NeuralNetwork(hidden_nodes: 100,
                    hidden_layers: 1,
                    relu: true);
      NeuralNetwork(hidden_nodes: 100,
                    hidden_layers: 1,
                    logistic: true);
      NeuralNetwork(hidden_nodes: 50,
                    hidden_layers: 2,
                    relu: true);
      NeuralNetwork(hidden_nodes: 50,
                    hidden_layers: 2,
                    tanh: true);
end;
```

Fig. 6: Training multiple models to learn XOR.

T is thrust, D is aerodynamic drag, F is rolling resistance, and V is airspeed. The complexity in this approach is that each force can be difficult to compute.

Thrust of the aircraft depends on air density, propeller pitch, propeller diameter, propeller rpm and airspeed. There is no simple equation to calculate it directly, and is often looked up in tables.

Aerodynamic drag, equation 2, depends on the coefficient of drag C_D, air density ρ, airspeed V, and cross sectional area S.

$$D = \frac{C_D * \rho * V^2 * S}{2} \tag{2}$$

Lastly, rolling resistance, equation 3, is dependent on the coefficient of friction between the landing gear and the runway surface μ, and the force being applied to the runway which is result of weight W and lift L.

$$F = \mu(W - L) \tag{3}$$

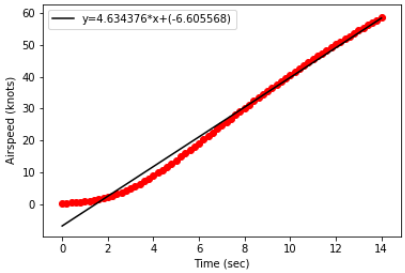
Whereas the other forces can be accurately calculated using properties of the aircraft, rolling resistance poses an issue. Different runway surfaces and weather conditions will change the friction between the landing gear and the runway. More importantly, the dependency on weight to calculate rolling resistance makes the physics approach more difficult during take-off. We will discuss a data-driven model which can estimate aircraft weight using only previous take-off trials that does not require knowledge of the aerodynamic properties of an aircraft.

B. Data-driven Weight Estimation

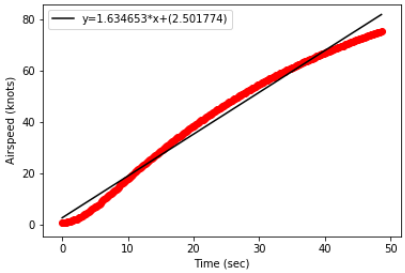
If an aircraft is heavier, then it takes longer to take-off, assuming all other variables are held constant. This can be seen in practice and in theory. This is because a heavy aircraft, given the same engine thrust, takes longer to accelerate to take-off speed. The model that will be discussed in this chapter uses the relationship between weight and acceleration to estimate weight. Furthermore, it only uses data that is available from basic avionics on an aircraft such as the Cessna 172. This means that this model can be used in nearly any aircraft, as long as it has an accurate measurement of airspeed and atmospheric conditions such as temperature, air pressure, and altitude.

Looking at the airspeed over time gives a curve. Figure 7 shows the linear approximations of the velocity curve. As can be seen in Figure 7a, the curve takes time at the beginning until the curve accelerates at a roughly constant rate because during this time the propeller is getting up to speed. The acceleration of the aircraft would be approximated to the derivative of the fitted curve. As mentioned previously, if the aircraft is heavier then it takes longer to take-off, because the heavier aircraft has less acceleration. This relationship between weight and the velocity curve can be shown in Figures 8 for fixed atmospheric conditions.

Furthermore, in real-life situations, aircraft do not take-off in identical atmospheric conditions. Even within a single day, weather conditions can change greatly, so it is important to be able to account for atmospheric conditions. This model uses density altitude, which is defined as the altitude with respect to standard atmospheric conditions, as an encapsulation of all atmospheric properties since it includes air pressure, temperature, and altitude within a single value. However, we are disregarding the effect of humidity and wind. For a fixed weight, Figure 9 shows the effect of density altitude on the take-off velocity curve.



(a) Trial with density altitude of -1,809ft and weight of 1,811lbs.



(b) Trial with density altitude of 6,551ft and weight of 2,934lbs.

Fig. 7: Linear approximation of velocity curve.

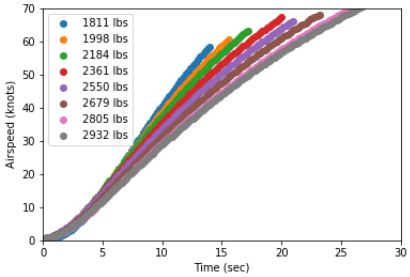


Fig. 8: Effect of weight on velocity curve.

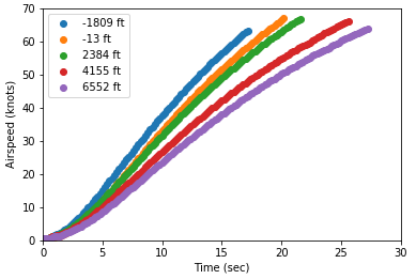


Fig. 9: Effect of density altitude on velocity curve.

C. Weight Estimation Method

In this section, we will discuss how the weight estimation model predicts the weight of a given trial. Including how the model uses the training data, and interpolates between training data points to get a final estimate.

1) Density Altitude: Density altitude represents what altitude the current conditions would be at in standard atmospheric pressure and temperature. The calculation used assumes completely dry air (0% humidity). Similar to density altitude, pressure altitude (PA) is the altitude with respect to standard air pressure, disregarding the effect of temperature.

$$PA = \text{Elevation} + 1000\ \text{ft} * (29.92\ \text{inHG} - \text{Pressure}) \quad (4)$$

Elevation is the altitude of the aircraft above mean sea level in feet and pressure is the atmospheric pressure at the location of the aircraft in inches of mercury.

$$ISA = 15°\text{C} - (1.98°\text{C}) * \left(\frac{PA}{1000\ \text{ft}}\right) \quad (5)$$

International Standard Atmosphere (ISA) temperature is the temperature at a specific altitude in standard atmospheric conditions.

$$DA = PA + (118.8 \text{ ft/°C}) * (OAT - ISA) \tag{6}$$

Outside air temperature (OAT) is the current air temperature. Finally, to calculate density altitude, we use equation 6 for its ease of calculation and very good approximation of true density altitude which is sufficient for our model.

2) Supervised Training: Training data of this model consists of take-off trials labelled with the true weight. Each take-off trial consists of atmospheric conditions (temperature, pressure, and altitude) which are used to calculate the density altitude and the velocity throughout the take-off which is used to estimate the acceleration of the aircraft. The stored data after training consists of three values: density altitude, estimated acceleration, and true weight for each take-off trial.

3) Interpolation: Once all training data is collected, the model calculates two values used to interpolate the final result. The two values represent the effect that a change of density altitude has on the acceleration, and the effect that a change in acceleration has on the weight. These two values are called δ_{da} and δ_{acc}, given by equations 7 and 8.

$$\delta_{\text{da}} = \frac{\Delta\text{acceleration}}{\Delta\text{density altitude}} \tag{7}$$

$$\delta_{\text{acc}} = \frac{\Delta\text{weight}}{\Delta\text{acceleration}} \tag{8}$$

These values can be understood as quantifying the relationships shown in Figures 9 and 8, respectively. However, instead of showing the relationship in terms of the velocity curve, it is with respect to the approximated acceleration. Figure 10 shows the relationship between density altitude and acceleration and Figure 11 shows the relationship between weight and acceleration. The δ_{da} is the slope of the linear approximation of the relationship shown in Figure 10 and, similarly, δ_{acc} is the inverse of the slope of the linear approximation of Figure 11.

To calculate δ_{da}, the model finds the average of values of δ_{da} between pairs of training data points with similar weights, $\pm 0.1\%$. Similarly, to calculate δ_{acc}, the model finds the average values of δ_{acc} between pairs of points with similar density altitudes, $\pm 0.1\%$. If the dataset does not have enough data points with similar weights and density altitudes to calculate δ_{acc} and δ_{da}, then the model will not be able to accurately estimate weight. This can be solved by adding structure to data collection by collecting data using specific weights and density altitudes, or increasing the size of the dataset.

An example PILOTS program to train a weight estimation model is shown in Figure 12. The *WeightEstimator* model is a custom algorithm written in Python which implements the model described in this section. We tried using various other algorithms including neural networks. Our approach is far simpler to train, adjust for various parameters (*e.g.*, properties of the dataset, effect of density altitude), and it

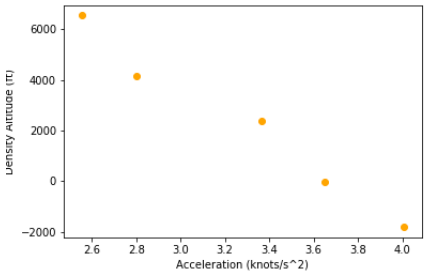


Fig. 10: Relationship between density altitude and estimated acceleration.

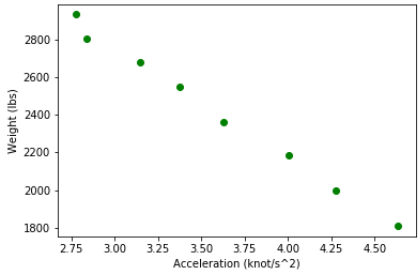


Fig. 11: Relationship between aircraft weight and estimated acceleration for density altitude of -1,809ft.

is more explainable than a neural network approach. This program trains a model using data from KRNO. Furthermore, the model uses data points with density altitude values within $\pm 0.2\%$ to calculate δ_{acc} and uses data points with weights within $\pm 0.1\%$ to calculate δ_{da}. It is useful to tune these values slightly for each training dataset to improve accuracy. For a more uniform dataset these values are expected to be smaller.

4) Weight Prediction: When the model is given new data with an unknown weight, it calculates the density altitude and the acceleration from the slope of the velocity curve. It then finds the closest stored data point, closest being the point that minimizes the difference in the density altitudes and the difference in the acceleration values as in equation 9. The

```
trainer weight_model_krno;
  data
    // Airspeed, pressure, temperature,
    // altitude, and actual weight
    va, prs, tmp, alt, curr_w using
      file("data_krno.csv", );
  model
    features: va, prs, tmp, alt;
    labels: curr_w;
    algorithm:
      WeightEstimator( da_close: 0.2, w_close:
     0.1 );
end;
```

Fig. 12: Weight estimation model trainer.

model then takes that data point and interpolates to get a final estimate.

$$e = (\mathrm{DA}_1 - \mathrm{DA}_2)^2 + (a_1 - a_2)^2 \tag{9}$$

Given the closest stored data point from the training data, equation 10 shows how to interpolate this value to improve accuracy.

$$\text{weight} = (\text{closest weight}) + (\delta_{\text{acc}} * (\delta_{\text{da}} * E_{\text{da}} - E_{\text{acc}})) \tag{10}$$

Error in density altitude, E_{da}, is the difference between the closest point's density altitude and the input's density altitude. Similarly, error in acceleration, E_{acc}, is the difference in the two data points' accelerations.

Multiplying E_{da} and δ_{da} gives the error in the acceleration due to the density altitude. Therefore, adding the value with E_{acc} gives us the total error in acceleration. Note, the negative sign on E_{acc} is due to the inverse relationship between the error in acceleration and weight. Meaning a positive value of E_{acc} would result in a negative adjustment in the weight. Lastly, multiplying the total error in the acceleration by δ_{acc} gives the weight adjustment due to the error in the slope, which gives us the updated final weight estimate.

D. Experimental Results

The data was collected in X-Plane 9 using a Cessna 172SP. Data streams such as airspeed, temperature, air pressure, and altitude were collected during standard procedure[5] take-offs at two airports: Albany International Aiport (KALB) and Renoe-Tahoe International Aiport (KRNO), using a selection of density altitudes (five for each runway), and weight classes. There are eight weight classes: five are considered safe take-off weights from extremely light to maximum take-off weight and three weights which are considered overweight consisting of 5%, 10%, and 15% over maximum take-off weight. There are 80 total take-off trials in the training dataset.

The density altitude values for each airport are a selection of atmospheric conditions that could happen in that location.

These weight classes were selected as a uniform partitioning of weights that represent a reasonable possible low-end take-off weight up to a very extreme overweight condition.

The model was validated using a wide selection of weights and density altitudes, distinct from the training dataset. This included take-offs from KALB, KRNO, and Minot International Airport (KMOT). KMOT was used to see how well the model can be used on runways that the model had no training with; furthermore, the altitude of KMOT is between the altitudes of KALB and KRNO.

The results of four different models were tested. There are two models trained using data from only one airport (KALB and KRNO), to see how the model can transfer from one airport to another without direct knowledge. Referred to as the KALB and KRNO models. Another model is trained on the complete dataset and is referred to as the total model.

[5]According to Cessna 172 Information Manual [7]

TABLE I: Average percent error based on runway of testing trial.

Testing Airport	KALB Model	KRNO Model	Total Model	Ensemble Model
KALB Accuracy	3.22	13.00	10.24	3.32
KRNO Accuracy	-14.90	1.10	1.51	2.26
KMOT Accuracy	-2.07	12.34	11.63	4.68
Overall Average	-6.81	6.78	5.87	2.9

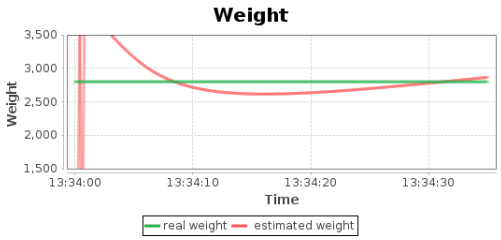


Fig. 13: Weight estimation curve.

Lastly, there is an ensemble model that aggregates the results from the two models trained only on one airport. This model uses the average density altitude between all training trials as a transition point (approximately 6000ft). When given a density altitude below this point the model uses the KALB model and above which uses the KRNO model.

Table I shows the average final results of each model. This table also shows that some models performed better on certain runways than others. As can be seen in Table I, the KALB model underestimated estimates on KRNO trials and the KRNO model overestimated on KALB trials. This is likely due to some differences between the two runways that is not properly accounted for by the model, such as runway slope or surface. Lastly, the ensemble model outperforms the other models.

From a safety standpoint, it is important to be able to quickly alert the pilot about an overweight condition so that they have enough time to take appropriate action. For large aircraft, V_1 is the take-off decision speed: the speed above which take-off should not be aborted. Meaning, if there is an issue before reaching V_1, the pilot has enough time to abort take-off and stop safely; however, if an issue arises after V_1, the aircraft must complete take-off to not overrun the runway. However, in small aircraft such as the Cessna 172SP this speed is not calculated. To mitigate this, we have come up with a reasonable decision point for a small aircraft: either before the aircraft reaches 55 knots or 1000ft before the runway ends, whichever comes first. This is because 55 knots is the speed at which standard procedure says to begin lifting the nose of the airplane. However, with heavy aircraft in high density altitudes it may take time for the aircraft to reach 55 knots. The aircraft needs to abort take-off with enough time to come to a full stop; 1000ft is more than enough runway to stop even on fairly short runways [6].

We used two points to indicate when the model produces useful results. This is when the model converges within 10%

[6]based on maximum landing distance ground roll in Cessna operating manual

TABLE II: Speed of model convergence.

Testing Airport	% of Trials Converged	Avg Time <10%	% of Trials Converged	Avg Time <5%	Avg Take-off Length
KALB	100	8.4	90	12.6	23.5
KRNO	100	10.5	80	18.1	29.3
KMOT	75	11.5	75	17.5	25
Overall	96.7	9.7	83.3	15.75	26.75

TABLE III: Distance traveled and speed at slowest 10% convergence point.

Testing Airport	Slowest Trial Speed (knots)	Slowest Trial Distance (ft)	Runway Length (ft)
KALB	40.6	328	8500
KRNO	45.6	919	11001
KMOT	40	492	7700

of the actual weight and when the model converges within 5%. These points are sooner than the point where the curve reaches a minimum which could also be used as a point of convergence. As can be seen in the weight estimation curve in Figure 13, the model first overestimates the weight of the aircraft then converges to a final value. When the model first converges within 10%, the model gives a rough estimate. This gives us an idea of whether the estimate will be overweight or underweight. The estimate then stabilizes and converges within 5% and gives an estimate that is close to its final estimate. Table II shows how many testing trials converged and how quickly. Considering only the trials which converged, Tables III and IV show the speeds and distances of the slowest trials to converge to 10% and 5%, respectively. It can be seen that the slowest trials still converged before the decision distance point even if the runways were half as long.

E. Cessna 172R N4207P Accident

On August 25, 2014 a Cessna 172R airplane, N4207P, crashed in Willoughby Hills, Ohio shortly after take-off from Cuyahoga County Airport (KCGF). The private pilot and three passengers died in the crash. The aircraft was loaded to 2,622lbs, approximately 166lbs over its maximum gross weight of 2,457lbs (106% max gross weight), and was within safe ranges for the center of gravity. The airplane became airborne approximately 2000ft down runway 6 which is 5500ft long. The pilot noted that the aircraft was slow to climb and tried to turn back to land. The increased weight and steep turning angle used likely caused the aircraft to stall.

We created a scenario using X-Plane with a Cessna 172SP loaded to 106%, total of 2720lbs, taking off from KCGF runway 6. We replicated the atmospheric conditions of the day of the crash, 24°C and 30.09 inHG. Wind was reported as 10 knots from 140°, which would be a crosswind and would have negligible effect on take-off distance. However, we omitted wind entirely from our simulation as we currently solely use density altitude to encapsulate atmospheric conditions; this is an area of future work. The report noted that the aircraft had approximately 36 gallons of fuel at the time of take-off. The center of gravity was calculated from the information given in the report, 4.3 inches forward from the standard center of gravity which is within the safe range.

TABLE IV: Distance traveled and speed at slowest 5% convergence point.

Testing Airport	Slowest Trial Speed (knots)	Slowest Trial Distance (ft)	Runway Length (ft)
KALB	56	1083	8500
KRNO	64	2297	11001
KMOT	61.9	1444	7700

```
program weight_experiment;
  inputs
    // Airspeed, pressure, temperature,
    altitude,
    // and actual weight
    va, prs, tmp, alt, curr_w (t) using
    closest (t);
    // Estimate weight
    est_w (t) using model(N4207P_model, va,
    prs, tmp, alt);
  outputs
    va, curr_w, est_w, e at every 200 msec;
  errors
    e: (est_w-curr_w)/curr_w * 100;
end;
```

Fig. 14: PILOTS N4207P program.

```
trainer N4207P_model;
  data
    // Airspeed, pressure, temperature,
    altitude,
    // and actual weight
    va, prs, tmp, alt, curr_w using
      file("N4207P_training_data.csv");
  model
    features: va, prs, tmp, alt;
    labels: curr_w;
    algorithm:
      WeightEstimator();
end;
```

Fig. 15: N4207P weight estimation model trainer.

We collected 24 take-offs from KCGF runway 6. We used three distinct density altitudes: 860ft, 2049ft, and 3237ft, and the same eight weight categories from the previous experiments. We trained a weight estimation model using this data, see Figure 15. Figure 14 shows the full PILOTS pilots program.

We then ran several take-off trials using the exact conditions mentioned previously to replicate the N4207P take-off. The entire take-off was approximately 27s, and around 1800ft long, slightly faster then the approximate reported 2000ft of take-off distance of N4207P. The model produces the weight estimation curve as shown in Figure 16. However, on average, the model was able to estimate with a final accuracy of 3% and was able to estimate within 10% error after only 6s and within 5% error after 8s, as can be seen in Figure 17. After 8 seconds, our simulation only traveled 200ft. This means that the model could have alerted the pilot of an overweight condition well before the halfway point of the take-off giving ample time to react safely to this alert.

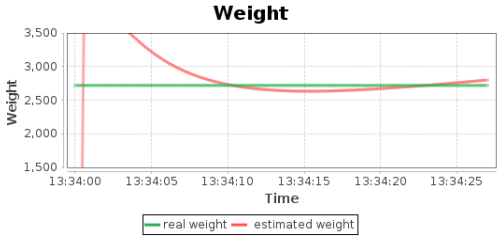


Fig. 16: N4207P weight estimation curve.

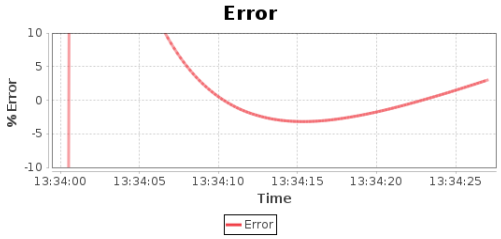


Fig. 17: N4207P weight estimation error curve.

V. Related Work

There are many Domain Specific Languages (DSLs) for machine learning applications [8]. OptiML is a DSL for machine learning which can take advantage of heterogeneous resources (CPUs and GPUs) [9]. MXNet is a ML library which combines symbolic expression with tensor computation [10]. These languages differ from PILOTS in that they are far more imperative and procedural in style. ScalOps [11] and Pig Latin [12] uses the low-level procedural style of MapReduce but with the high-level declarative style of SQL. The declarative SQL-like style of these DSLs is similar to the declarative nature of PILOTS; however, these DSLs are focused on processing large-scale data whereas PILOTS is focused on dynamic data-driven systems, fault detection, and fault recovery.

There exists some prior work on aircraft weight estimation. Imai, Galli, and Varela [5] used PILOTS to detect underweight conditions during cruise phase using data from the Tunin-ter 1153 accident. Lee and Chatterji [13] created a model which uses fuel usage to estimate take-off weight required to complete a given flight plan. Similarly, Sun *et al.* [14] created a model which can estimate the weight of an aircraft by analyzing the fuel consumption at various phases of flight and calculating a final weight probability distribution. Alligier *et al.* [15] created a mass estimation system used to help predict the rate of climb of an aircraft. Each of these models uses detailed physics-based model whereas our model uses the analytical redundancy to estimate weight in real-time. Furthermore, our model aims to provide information that may be critical to the safety of the pilot and passengers as well as provide that information before the aircraft becomes airborne, which none of the previous models can do.

There are several patents which detail systems to estimate aircraft weight before or during take-off. Several use the landing gear in some manner, either by using pressure sensors within the landing gear [16] [17], or by measuring the bending of structural members of the aircraft which includes landing gear and wings [18]. Another details a system which automates the weighing of passengers and luggage and feeds this directly to a computer which calculates weight and center of gravity [19]. One system [20] describes measuring the acceleration of an aircraft, using an accelerometer, and using Newton's second law of motion to estimate weight of an aircraft during take-off. This system must know aerodynamic properties in order to calculate values such as lift. This approach differs from our approach in that our model has no prior knowledge of the aircraft and only uses data from previous take-offs. Furthermore, our model requires no extra hardware such as an accelerometer or other sensors and can be used in various aircraft with no changes.

VI. Conclusions and Future Work

We introduced the new declarative extensions to PILOTS to learn models from data. Our extensions allow data scientists to use PILOTS to create systems using machine learning algorithms quickly and easily without a strong background in programming. Furthermore, it allows for users with a back-ground in programming to easily integrate machine learning algorithms into PILOTS. These features in PILOTS are more declarative and model-agnostic than other similar domain specific languages and libraries.

We also discussed our method of data-driven aircraft weight estimation. Our method can be used on most fixed-wing aircraft with no additional hardware and provides useful results to the pilot in real-time during take-off. We showed that our method is capable of preventing accidents such as the August 2014 N4207P accident where an overweight condition caused the aircraft to crash shortly after take-off.

Our weight estimation model was trained on data from small fixed-wing single-engine aircraft. Further research can look at other types of aircraft, such as larger multi-engine aircraft and commercial jets, where we expect our model to produce accurate results before reaching take-off decision speed. Future work includes creating an ensemble system to estimate weight during all phases of flight such as climbing, descending, land-ing, and turning. Even taxiing could also potentially provide enough information to detect weight discrepancies. Besides overweight conditions, such a model would also be able to detect certain failures of the aircraft, *e.g.*, fuel leaking which decreases aircraft weight, an engine failure which decreases thrust, or wing surfaces icing which decreases lift. The exact source of a discrepancy may not always be easy to isolate; however, being able to alert the pilot to a potential fault is critical to maintaining safety.

In data-driven systems, there is a lot of inherent randomness and uncertainty: sensor errors, communication delays and failures, and much more. Without the ability to maintain correctness in these conditions, an application is not useful in the real world. However, it is difficult to be able to be correct in all possible types of conditions, due to computational, temporal, or model-based constraints. Making models that

have a clearly defined envelope of correctness allows the user, or other applications, to know when the result may not be correct. Breese *et al.* introduce the concept of formal safety envelopes which shows which properties of a model hold under which assumptions [21]. Furthermore, modeling uncertainty quantification and propagation and devising ways to communicate uncertainty to system users, can enable better aeronautical decision making.

ACKNOWLEDGEMENT

This research is partially supported by the Air Force Office of Scientific Research, Grant No. FA9550-19-1-0054 and National Science Foundation Grant No. 1816307. We would also like to thank everyone who reviewed drafts of this paper for their valuable feedback.

REFERENCES

[1] S. Leasca. (2018, Feb.) When accidents are most likely to happen during a flight. [Online]. Available: https://www.travelandleisure.com/travel-news/when-most-fatal-accidents-occur-on-flights

[2] S. Imai and C. A. Varela, "Programming spatio-temporal data streaming applications with high-level specifications," in *3rd ACM SIGSPATIAL International Workshop on Querying and Mining Uncertain Spatio-Temporal Data (QUeST) 2012*, Redondo Beach, California, USA, November 2012.

[3] S. Imai, S. Chen, W. Zhu, and C. A. Varela, "Dynamic data-driven learning for self-healing avionics," *Cluster Computing*, Nov 2017. [Online]. Available: http://rdcu.be/yJNh

[4] S. Imai, E. Blasch, A. Galli, W. Zhu, F. Lee, and C. A. Varela, "Airplane flight safety using error-tolerant data stream processing," *IEEE Aerospace and Electronics Systems Magazine*, vol. 32, no. 4, pp. 4–17, 2017. [Online]. Available: http://www.brightcopy.net/allen/aesm/32-4/index.php/6

[5] S. Imai, A. Galli, and C. A. Varela, "Dynamic data-driven avionics systems: Inferring failure modes from data streams," in *Dynamic Data-Driven Application Systems (DDDAS 2015)*, Reykjavik, Iceland, June 2015.

[6] S. Imai, F. Hole, and C. A. Varela, "Self-healing data streams using multiple models of analytical redundancy," in *The 38th AIAA/IEEE Digital Avionics Systems Conference (DASC 2019)*, San Diego, CA, September 2019. [Online]. Available: http://wcl.cs.rpi.edu/papers/DASC2019_imai.pdf

[7] Cessna Aircraft Company, *Information manual, Cessna Aircraft Company 1981 model 172P*. Cessna Aircraft Company, 1980.

[8] I. Portugal, P. Alencar, and D. Cowan, "A preliminary survey on domain-specific languages for machine learning in big data," in *2016 IEEE International Conference on Software Science, Technology and Engineering (SWSTE)*, June 2016, pp. 108–110.

[9] A. K. Sujeeth, H. Lee, K. J. Brown, T. Rompf, H. Chafi, M. Wu, A. R. Atreya, M. Odersky, and K. Olukotun, "OptiML: An implicitly parallel domain-specific language for machine learning," in *ICML*, 2011, pp. 609–616. [Online]. Available: https://icml.cc/2011/papers/373_icmlpaper.pdf

[10] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems," 2015.

[11] M. Weimer, T. Condie, R. Ramakrishnan *et al.*, "Machine learning in ScalOps, a higher order cloud computing language," in *NIPS 2011 Workshop on parallel and large-scale machine learning (BigLearn)*, vol. 9, 2011, pp. 389–396.

[12] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, "Pig Latin: A not-so-foreign language for data processing," in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 1099–1110. [Online]. Available: https://doi.org/10.1145/1376616.1376726

[13] H. tae Lee and G. Chatterji, *Closed-Form Takeoff Weight Estimation Model for Air Transportation Simulation*. Aerospace Research Central, 2012. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2010-9156

[14] J. Sun, J. Ellerbroek, and J. M. Hoekstra, "Aircraft initial mass estimation using bayesian inference method," *Transportation Research Part C: Emerging Technologies*, vol. 90, pp. 59 – 73, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0968090X18302626

[15] R. Alligier, D. Gianazza, and N. Durand, "Learning the aircraft mass and thrust to improve the ground-based trajectory prediction of climbing flights," *Transportation Research Part C: Emerging Technologies*, vol. 36, pp. 45 – 60, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0968090X13001708

[16] C. K. Nance, "Aircraft weight and center of gravity indicator," U.S. Patent 5 548 517, Oct. 22, 1998.

[17] M. A. Long and G. E. Gouette, "Aircraft weight and balance system," U.S. Patent 7 967 244, Nov. 16, 2006.

[18] C. D. Bateman, "Weight, balance, and tire pressure detection systems," U.S. Patent 4 312 042, Jan. 17, 1992.

[19] R. Stefani, "Aircraft weight and balance system," U.S. Patent 6 923 375, Sep. 29, 2003.

[20] H. Miller, "Takeoff weight computer apparatus for aircraft," U.S. Patent 4 490 802, Jan. 04, 1982.

[21] S. Breese, F. Kopsaftopoulos, and C. A. Varela, "Towards proving runtime properties of data-driven systems using safety envelopes," in *The 12th International Workshop on Structural Health Monitoring*, Stanford, CA, September 2019. [Online]. Available: http://wcl.cs.rpi.edu/papers/IWSHM19_brees.pdf