uavEE:
A Modular, Power-Aware Emulation Environment for Rapid Prototyping and Testing of UAVs

Mirco Theile*, Or D. Dantsker*, Richard Nai*, Marco Caccamo*,
*University of Illinois at Urbana-Champaign, USA, {mircot, dantske1, rnai2, mcaccamo}@illinois.edu

***Abstract*—State of the art design and testing of avionics for unmanned aircraft is an iterative process that involves many test flights, interleaved with multiple revisions of the flight management software and hardware. To significantly reduce flight test time and software development costs, we have developed a real-time UAV Emulation Environment (uavEE) using ROS that interfaces with high fidelity simulators to simulate the flight behavior of the aircraft. Our uavEE emulates the avionics hardware by interfacing directly with the embedded hardware used in real flight. The modularity of uavEE allows the integration of countless test scenarios and applications. Furthermore, we present an accurate data driven approach for modeling of propulsion power of fixed-wing UAVs, which is integrated into uavEE. Finally, uavEE and the proposed UAV Power Model have been experimentally validated using a fixed-wing UAV testbed.**

I. Introduction

In recent years, we have seen an uptrend in the popularity of Unmanned Aerial Vehicles (UAVs) driven by the desire to apply these aircraft to areas such as precision farming, infrastructure and environment monitoring, surveillance, surveying and mapping, search and rescue missions, weather forecasting, and more. All the mentioned application scenarios require the design of UAVs that carry a high performance embedded computer system, which run flight management and mission critical software. The developement and testing of avionics for unmanned aircraft is an iterative process that involves many test flights and multiple revisions of software and hardware.

While the long term goal of this research is the design and development of a solar-powered, long-endurance UAV for real-time on-board data processing [1, 2], we now present our work to develop a modular, power-aware UAV Emulation Environment (uavEE). The uavEE uses high fidelity simulators to simulate the flight behavior of the aircraft. It emulates the avionics hardware by providing a ROS based control flow interface between the hardware and the simulator (FS One®[3, 4, 5] or X-Plane[6]).

We introduce a step-wise prototyping process that allows the user of uavEE to first focus on the software development and testing, followed by hardware integration and emulation, drastically improving debugging efforts. The cheap and quick deployment in the emulation environment saves hours at the field, dramatically increasing the success rate of autonomous flight tests. The system level diagram in Figure 1 presents a few of the possibilities that the highly modular uavEE can offer. We briefly describe the possibilities that arise when integrating fault modeling into the emulation environment. In

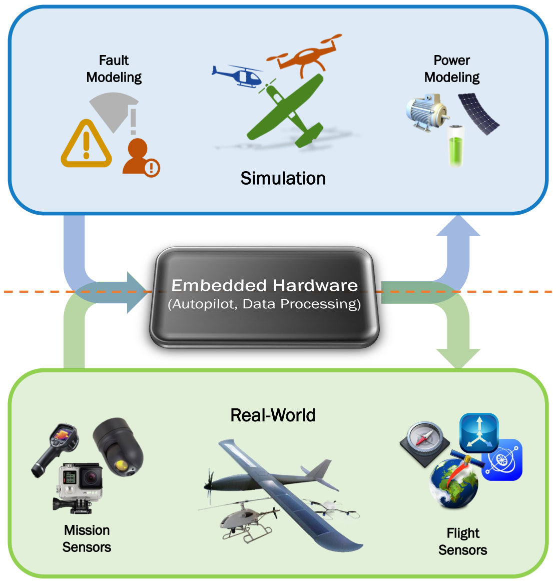


Fig. 1: A system level diagram of the UAV Emulation Environment (uavEE).

this work we focus on power modeling, since it is a crucial element for long-endurance solar-powered flight.

The power on a UAV is consumed by its propulsion subsystem, actuators, and avionics, as well as mission components. Even for an efficient sailplane, the propulsion power consumption takes the majority of the energy. Therefore, the integrated power model focuses on the propulsion power. Previous works have separately looked at aircraft power modeling [7, 8, 9, 10, 11, 12] and propulsion system modeling [13, 14, 15, 16, 17] with varying degrees of assumptions, and simulation for flight control purposes [4, 18, 19, 20]. In contrast to the existing literature we present a data driven approach. We use a physical model of the propulsion power, which is fully derived in [21], and use the non-linearities as kernel functions of a linear regression algorithm. This way we can easily train the power model with flight data and then use the linear parameters for power estimation of future flights. For this approach no aerodynamic parameters of the airplane are needed which accelerates the modeling process. We validate the proposed power model by means of flight experiment using a highly accurate data acquisition system [22]. To the best of our knowledge, this is the first accurate and portable data driven approach for modeling of the propulsion power of fixed-wing aircraft.

In summary, the main contributions of this work are:

1) Development of a real-time UAV Emulation Environment (uavEE) that interfaces with an existing high-fidelity flight simulator to simulate flight dynamics and emulate the avionics hardware;
2) Integration of an accurate UAV power model using a novel data driven approach based on non-linearities of a physical model of the aircraft;
3) Experimental validation of the uavEE integrated with the proposed power model using a fixed-wing UAV testbed, which has a 1.59 m wingspan and a mass of 3.92 kg.

The uavEE has already been instrumental for the research activity of this group on UAVs by allowing for rapid testing and debugging of autopilot and path planning software. In the case of developing our autopilot, uavAP, more than 90% of autopilot software bugs hidden in the avionics software were discovered by flying the avionics computer within the emulation environment, dramatically cutting both the time and cost of development. The UAV emulation environment has also been used multiple times as an educational tool for student class projects on UAVs, and therefore we have released it as open source[1]. The emulation software can also be interfaced with other flight simulators (*e.g.* FlightGear, Gazebo) at minimal porting cost by exporting the same communication interface that is used with FS One®and X-Plane.

The paper is structured as follows: In Section II, we describe the Emulation Environment and the hardware and software used for modeling and testing. In Section III, we describe our data driven power model approach, which we evaluate alongside uavEE with an example flight in Section IV. In Section V we conclude the paper and give an outlook for future work.

II. Emulation Environment

In this section we describe our Emulation Environment for rapid prototyping of Unmanned Aerial Vehicles (uavEE). The Emulation Environment consists of an embedded hardware, a Robot Operating System (ROS) based ground-station running on a Linux machine, and a simulator. The goal of the emulation environment is to imitate real flying conditions as closely as possible to let the embedded hardware (it is running the autopilot in real flight) think that it is actually flying. The hardware and software setup and interfaces for the real flight as well as the simulated flight are shown in Figure 2. The graphic uses a coloring scheme to classify the state in which the components are used:

- Yellow: Only used in real flight. Inactive or disconnected in the emulation.
- Blue: Only used in the emulation. Inactive or disconnected in real flight.
- Green: Used in both, real flight and emulation.

[1]The executable of FS One augmented with an emulation interface needs to be purchased under proper software license by the company (InertiaSoft) that sells the flight simulator.
uavEE: https://github.com/theilem/uavEE.git
uavAP: https://github.com/theilem/uavAP.git

This section is structured as follows: First we describe the embedded hardware and the software architecture on it. Afterwards we give an introduction into the ROS environment followed by a description of the hardware and software integration. To conclude, we present a simplification of the uavEE to allow even faster prototyping of autopilot software.

A. Embedded Hardware

The embedded hardware used in emulation is the same embedded hardware that is used to control the aircraft in real flight, which is the Al Volo FC+DAQ [23]. The software and interface schematics is depicted on the embedded hardware side of Figure 2. In real flight, the Emulation Interface is inactive and the Al Volo backend is active. The backend collects sensor information and sends them through an API to the autopilot. The autopilot calculates the appropriate control based on a prior set mission and returns an actuation command through the API. The backend takes the actuation command and sends it to the actuators to control the aircraft. During this process the autopilot is continuously sending status information through a radio interface to a ground-station for monitoring. Additionally the ground-station can utilize the same radio link to send commands to the autopilot.

In this setup the goal of the emulation environment is to imitate the functionality of the Al Volo backend as closely as possible to ensure that the autopilot is oblivious about whether it is in a simulation or not. Therefore, in emulation, the emulation interface performs the exact same task as the backend except that it uses a link to the autopilot interface in the ROS environment for sensor and actuation data.

The autopilot software that is used is custom designed and is described in section IV. However, it is conceivable that the modularity and abstraction of the interfaces in the emulation environment allow the usage of other embedded hardware and autopilot software with minimal porting efforts.

B. ROS Environment

To support the embedded hardware and the autopilot in real flight as well as in emulation a ground-station is needed for continuous monitoring of the aircraft status and for mission command. Our ground-station is part of a ROS environment, as visualized on the left side of Figure 2, which enables more features than only a ground station user interface (UI).

First described in [24], ROS is an open source robot operating system. It supports the design and programming of nodes for specific functionality and allows communication among them through ROS topics. Every node has the ability to publish messages to, or subscribe on messages from a topic which creates a runtime connection between publisher and subscriber. Additionally nodes can offer services to other nodes. Messages on topics are usually used for periodic data exchange and services for aperiodic requests. Further information, installation manuals, and tutorials can be found in [25].

1) Real Flight and Emulation: During real flight, the autopilot status information is sent to the Radio Com node which distributes the information in the ROS environment. The Ground Station UI node displays a graphical representation of

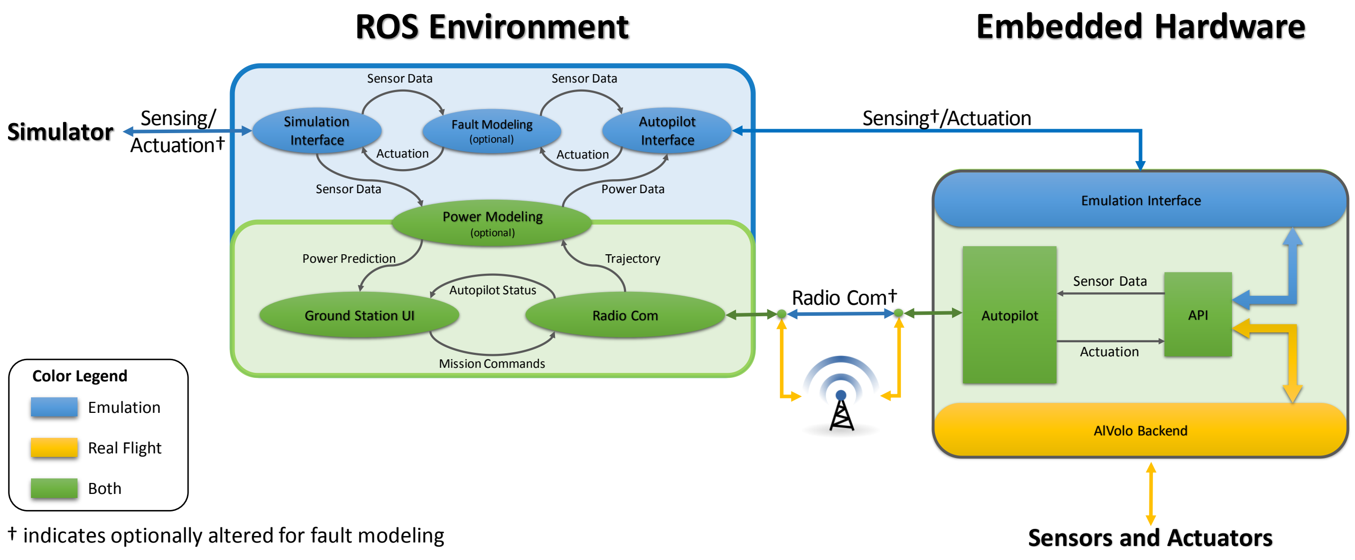


Fig. 2: Full setup schematics showing uavEE setup as well as real flight setup.

the status information. Besides displaying status information, the ground station can send user commands to the autopilot through the Radio Com node by calling a ROS service. These commands can include the tuning of parameters, the selection of missions, or override settings, etc.

In emulation, this setup is extended by a simulation interface and an autopilot interface node. The simulation interface communicates with a flight simulator to retrieve simulated sensor data. This data is passed to the autopilot interface, which forwards it to the embedded hardware. The actuation response from the autopilot is received in the autopilot interface and sent through the simulation interface back to the flight simulator.

Figure 2 shows that for the real flight setup, the ROS Environment is only connected to a radio interface to receive status information from and send commands to the autopilot on the embedded hardware. The embedded hardware is then connected to actual aircraft sensors and actuators to retrieve sensor data and perform control. In the emulation setup, the radio link is shortcut by a direct link, which does not effect the autopilot or the Radio Com node.

For power awareness, we use a high-fidelity simulator, FS One® [4, 5]. FS One® uses a full-flight-envelope 6-DOF aerodynamics model, created using component-build up and strip methods, and a fourth-order Runge-Kutta integration scheme to perform real-time flight simulation at 400 Hz. The simulator offers a large variety of aircraft, which allows the possibility to select an aircraft very similar to the actual testbed aircraft. The simulation leads to highly realistic flight data which is important for accurate power modeling. The downside of using FS One® is that the software runs on standard Windows machines, which leads to another inter-device communication in the setup. In contrast, the interface with X-Plane is based on a plug-in that directly creates a ROS node from the simulator. The benefit is that X-Plane and the ROS Environment of uavEE can run on the same device. However, X-Plane and most high fidelity simulators require significant computation, thus they might need to run on a separate machine. Therefore, the selection of the simulator has to be made dependent on the available hardware.

As before for the embedded hardware, the modularity of the emulation environment allows for the usage of most simulators with little to no porting efforts, as long as the simulators provide an interface for sensor data and actuation commands.

The physical setup of the emulation environment consists of a Windows computer running the FS One® simulator which is connected via RS232-Serial to a Linux computer running the ROS environment. The Linux computer is connected via two RS232-Serial links to the embedded board, the aforementioned Al Volo FC+DAQ. The embedded hardware runs a two core Atom 500 MHz processor with 1GB of LPDDR3 RAM. The software used for the emulation environment is our uavEE and for the autopilot is our uavAP, which are both open source available (link in the Introduction).

2) Power and Fault Modeling: The main reason for utilizing the ground station ROS environment for the emulation is that it allows for adding additional nodes enabling countless possibilities for simulation scenarios, two of which are power and fault modeling. For fault modeling, the sensor data from the simulator can be manipulated to simulate sensor failures or sensor spoofing. Furthermore, the actuation command can be altered, which can simulate actuator failures. Altering the radio communication can test the resilience against third party attacks in the radio link. The fault modeling is out of scope but will be addressed in future work.

The power modeling node in the ROS environment has two functionalities. The first is to estimate the current power consumption based on the sensor data and forward this information through the autopilot interface to the autopilot. In real flight the Al Volo backend provides power information to the autopilot using on-board power sensors. To have this information available in the emulation environment a power model is needed. Supplying the autopilot with power information is therefore a emulation-only function of the power modeling ROS node. The second functionality is to predict the power consumption of a given trajectory ahead of the flight. The power prediction is out of scope for this paper as we aim to first describe the basic power-aware emulation environment, which estimates power consumption based on inertial aircraft data. The power model is derived and explained in section III.

C. Simulation Environment

For rapid prototyping of the autopilot software it can be suboptimal to need to deploy the software on the embedded hardware to test it. Therefore, the emulation environment supports the direct deployment of the autopilot on the ground-station machine. The configurable autopilot interface node can be set up to start the autopilot process and use the API to communicate with it, similar to the embedded hardware side in Figure 2. If the ground station computer can be used to program the autopilot this direct deployment allows fast testing and debugging.

In this setup the autopilot software is deployed on a desktop machine rather than the embedded hardware making the emulation environment a simulation environment since the autopilot is only simulated and the actual embedded hardware is not used. After a few quick iterations in testing and debugging, the autopilot software can then be deployed on the embedded hardware e.g. to test its real-time behavior and the tasks' schedule on the embedded hardware. This step-wise testing process drastically improves the speed of debugging by looking at one problem at a time.

III. Power Modeling

To integrate power-awareness into uavEE we take use of the power model derived in [21]. The power flowing from the battery to the propulsion system can be modeled with

$$P_{bat} = \frac{K_i}{\eta_m \eta_p} \frac{\cos^2 \gamma}{v \cos^2 \phi} + \frac{K_p}{\eta_m \eta_p} v^3 + \frac{m}{\eta_m \eta_p} (g \sin \gamma + a) v \quad (1)$$

To simlify this equation we combine the physical and aerodynamical parameters to write

$$P_{bat} = A \frac{\cos^2 \gamma}{v \cos^2 \phi} + B v^3 + C (g \sin \gamma + a) v \quad (2)$$

in which the only variables are γ as the vertical climb angle, ϕ the roll angle, v the total velocity and a the forward acceleration of the UAV. To find the parameters for a given aircraft we use machine learning techniques.

A. Regression with Kernel

A typical linear regression for multiple input and single output (MISO) linearly maps input values to the output value. This can be represented with

$$y = \mathbf{w}\mathbf{x} \quad (3)$$

where y is the output value, $\mathbf{x}$ are the input values, and $\mathbf{w}$ is the prior unknown weight vector. Using linear regression these weights are found by minimizing a quadratic cost function over training data. If we have non-linear dependencies between our output and input values we cannot directly use this method. If the non-linearity is known, we can, however, use the so called "Kernel Trick" to create a linear regression. In short, we can map our actual input values $\mathbf{\Theta}$ with kernel functions $\mathbf{f}$ to $\mathbf{x}$ which is then linearly mapped to y, which results in:

$$y = \mathbf{w}\mathbf{f}(\mathbf{\Theta}) \quad (4)$$

In the case of the power model our input vector is defined as

$$\mathbf{\Theta} = [\gamma, \phi, v, a]^T \quad (5)$$

After compressing the constant factors in (1), which resulted in (2), we can write the weight vector as

$$\mathbf{w} = [A, B, C] = \left[\frac{K_i}{\eta_m \eta_p}, \frac{K_p}{\eta_m \eta_p}, \frac{m}{\eta_m \eta_p} \right] \quad (6)$$

the kernel functions as

$$\mathbf{f}(\mathbf{\Theta}) = \left[\frac{\cos^2 \gamma}{v \cos^2 \phi}, v^3, (g \sin \gamma + a) v \right]^T \quad (7)$$

and P_{bat} being y. Any solver for regression, e.g. Matlab, can approximate $\mathbf{w}$ given a set of training data for $\mathbf{\Theta}$ and y.

B. Training and Testing

To select training data we need to take a look at the assumptions that are made in the derivation of the power model. The first assumption is that we can neglect the angle of attack α, which offsets the actual climb angle from the aircraft pitch angle. If this assumption does not hold for the training data, a wrong relation between climb and power is learned. An example of this can be seen in Figure 3. Here the bad training data is coming from stalled gliding flights, where the aircraft climbs for a short time and then stalls. Since the motor is not engaged the regression assumes that only little power is needed to climb, or to fly in general. Therefore, this training set leads to an underestimation of the power consumption. The good training set was taken from two 400s flights with low angle of attack.

Additionally, the velocity v in the regression model is airspeed. If GPS speed is used and there is wind, the dependency towards velocity will be wrong. Therefore, if no pitot tube for airspeed measurement is available, it is advisable to train the model on windless flights.

Testing is not as vulnerable to these two assumptions since the effects of angle of attack and wind usually average out over time. Therefore, the estimation of the total consumed propulsion energy, which is the cumulated estimated propulsion power consumption, remains accurate.

IV. Evaluation

In this section we validate the proposed power model by comparing estimated with measured energy consumption, followed by a detailed comparison of the estimated and measured power consumption given an autonomously flown trajectory. Afterwards, a comparison between real and emulated flights shows the accuracy of the emulation environment as well as the applicability of the power model in emulation. First we describe the hardware and software used for the evaluation.

A. Hardware and Software

The hardware used for the actual flight is split into aerodynamic hardware, the aircraft itself, the computational hardware, and sensor and actuation management.

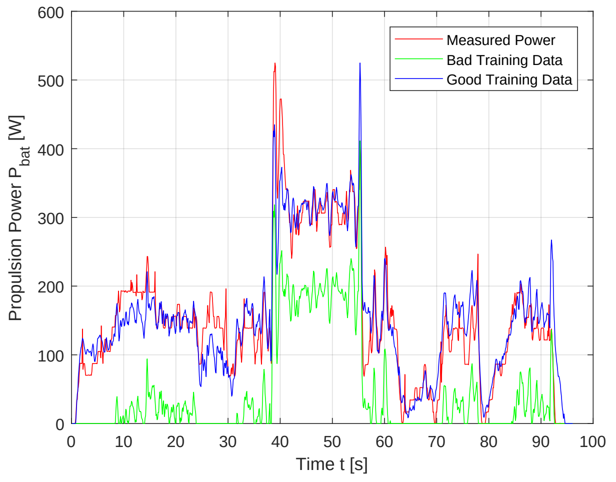


Fig. 3: Comparison of power estimation to measured power using good and bad training data sets.

Fig. 4: Completed flight-ready aircraft.

1) Aircraft: A fixed-wing trainer-type radio control aircraft, which was built for previous avionics development [26, 27, 28], was used for the evaluation. The aircraft built was a Great Planes Avistar Elite [29], which has a 1.59 m wingspan and a mass of 3.92 kg. The aircraft has the following control surfaces: 2 ailerons (roll), 2 flaps, 1 elevator (pitch), and 1 rudder (yaw). The completed flight-ready aircraft is shown in Figure 4 and its specifications can be found in [27].

2) Avionics: The aircraft was instrumented with an Al Volo FC+DAQ 400 Hz flight computer and data acquisition system [23], which incorporated the open source uavAP autopilot. Our custom designed uavAP autopilot, mentioned in section II, deployed onto an Al Volo FC+DAQ is based on a modular and configurable framework. The framework allows the easy integration of different planning and control algorithms. For detailed information about uavAP, the interested reader is directed to the GitHub page found in the Introduction.

B. Power Model Evaluation

In order to evaluate the power model we first look at the modeled total consumed propulsion energy over a full flight. For this validation the regression model was trained on two 400s flights from shortly after take-off to shortly before landing, to eliminate ground effect disturbances[2]. The trained model is then tested on a third flight. The resulting measured and estimated energy consumption are shown in Figure 5. It

[2]The resulting weights are $\mathbf{w} = [1130.97, 0.01353, 6.3444]$

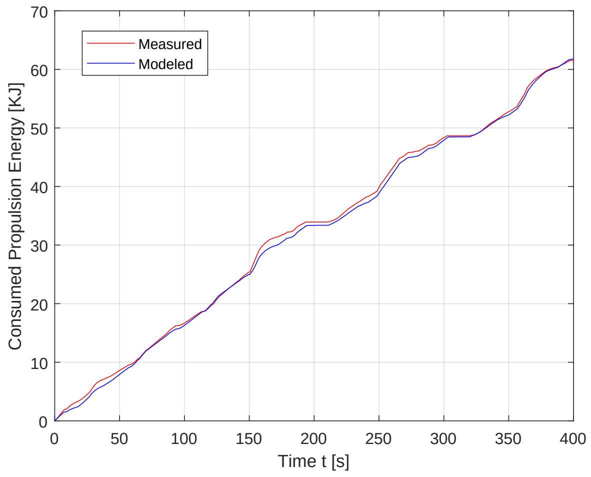


Fig. 5: Comparison of consumed propulsion energy measurement to estimation using the regression model.

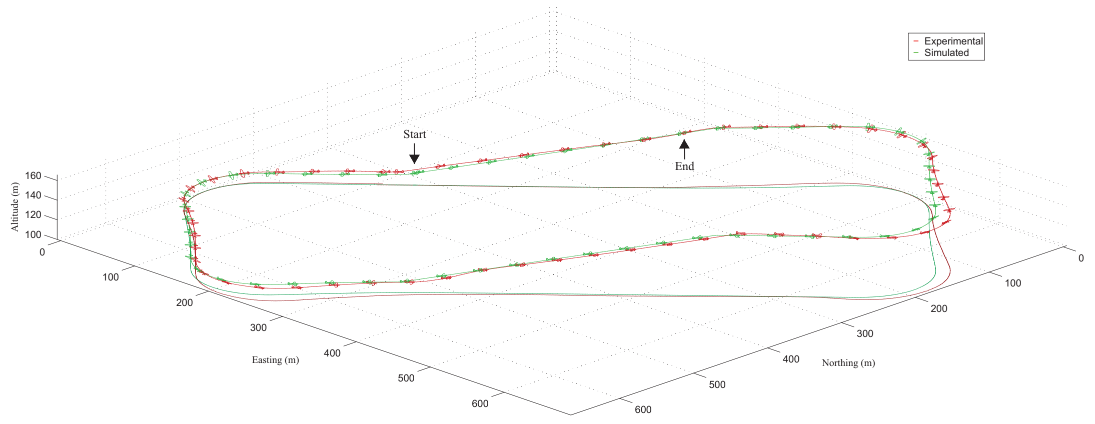
can be seen that all disturbances during the flight average out and that the power model perfectly estimates the consumed energy at the end of the flight.

We will now go into detail and look at the estimated power from the 100s to 200s ticks. Figure 6 shows the trajectory, the power consumption and the total consumed energy of the extract. For now we focus on the red and blue lines. In the power Figures 6b and 6c the red lines show power measurements during the flight. The blue lines show the estimations of the power model using inertial and GPS data from the flight. It can be seen that for the given trajectory, which contains turns, climbs, and straight lines the power estimation is nearly congruent to the measured power. The difference in the beginning can be explained by an offset angle of attack due to wind.

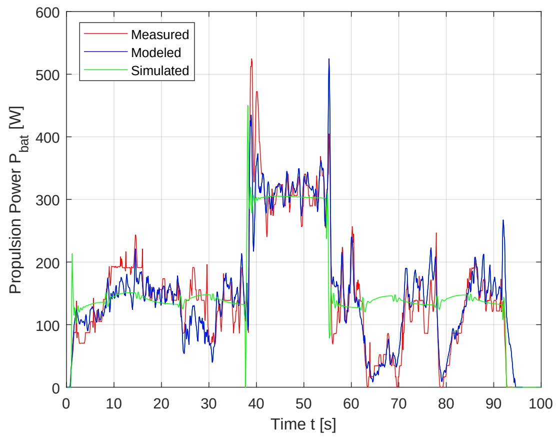
C. Emulation Validation

Since the power model yields highly accurate results, when applying it on the measured inertial data, we can use it to show the applicability of the emulation environment. For this a similar aircraft is flown in the emulation environment, using the same embedded hardware and the same autopilot settings. The resulting trajectory can be compared to the real flight trajectory in Figure 6a. It shows that the trajectory-following-behavior in uavEE is similar to real flight.

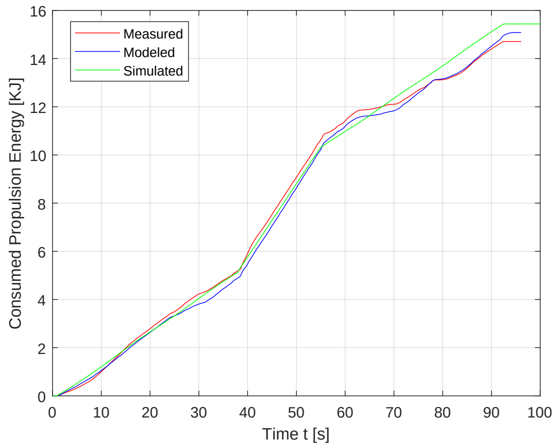
The trained power model is applied on the inertial data coming from the simulator. The resulting power and energy curves can be seen in Figures 6b and 6c, respectively. It can be seen that the power consumption data of the simulator is less noisy than the measured power. The overestimation during the last two turns can be linked to a tail-wind in the real flight. As explained before, when looking at the total energy consumption these disturbances cancel out which we can observe in Figure 6c. The graph shows that for this specific extract of the trajectory we have a slight overestimation of the consumed energy in the simulator. However, the results of the power model applied in uavEE is very similar to the real flight and shows that uavEE has the capability to accurately estimate power and energy consumption.



(a) Comparison of aircraft path for experimental (red) and simulated flight (green) results. The airplane is plotted at 6x scale and every 2 seconds.



(b) Comparison of propulsion power from experimental measured (red), experimental modeled (blue), and simulated (green) results.



(c) Comparison of propulsion energy consumed from experimental measured (red), experimental modeled (blue), and simulated (green) results.

Fig. 6: Evaluation of power model comparing experimental to simulated results for circuit maneuver performed with the Avistar unmanned aircraft

V. Conclusion and Future Work

We described and evaluated uavEE, our modular, power-aware emulation environment for UAVs. The uavEE creates a connection between high fidelity flight simulators and autopilot software running on embedded hardware. The step-wise prototyping process introduced allows the user of uavEE to first focus on the software development and testing, followed by hardware integration and emulation, drastically reducing debugging efforts. The cheap and quick deployment in the emulation environment saves hours in the field, dramatically increasing the success rate of autonomous flight tests.

The modularity of the emulation environment, the uavEE, as well as the autopilot, the uavAP, create a highly customizable solution for the development of embedded software for any aircraft, not limited to fixed wing. Using fault modeling, the autopilot can be tested regarding robustness and cyber-security, which is becoming ever more important. The integration of our accurate power model allows power-aware autopilots to be tested in uavEE. The data driven approach of the power model is easy to use and allows for fast integration of other aircraft. Together with the power model, uavEE opens up the benefits of emulation to countless long-endurance flight applications, including solar UAVs.

The evaluation of uavEE was performed with actual flight testing using the same embedded Al Volo hardware and uavAP autopilot software and showed that the resulting flight path, as well as the measured and modeled power data, are highly similar. Since the software for uavEE, as well as the uavAP, is available as open source, other research and education groups can benefit from this powerful tool.

Using the work presented in the paper we can further develop a long endurance solar UAV. For this we need power-aware path planning and control algorithms, which we can rapidly test in the uavEE. Extending the power model to incorporate mission power estimation will allow us to develop planning algorithms to take into account the computational power of the on-board data processing tasks. Furthermore using the fault modeling possibilities of uavEE we can assess the autopilot for safety regarding failures as well as third party attacks. For this, uavAP can be extended to use robust control, e.g. L1 adaptive control [30].

Acknowledgments

The material presented in this paper is based upon work supported by the National Science Foundation (NSF) under grant number CNS-1646383. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the NSF.

We would like to thank Prof. Renato Mancuso and Andrew Louis for their help with flight testing, ground station, development, and emulation environment development.

References

[1] Real Time and Embedded System Laboratory, University of Illinois at Urbana-Champaign, "Solar-Powered Long-Endurance UAV for Real-Time Onboard Data Processing," http://rtsl-edge.cs.illinois.edu/UAV/, Accessed Mar. 2018.

[2] O. D. Dantsker, M. Theile, M. Caccamo, and R. Mancuso, "Design, Development, and Initial Testing of a Computationally-Intensive, Long-Endurance Solar-Powered Unmanned Aircraft," AIAA Paper 2018-4217, AIAA Applied Aerodynamics Conference, Atlanta, Georgia, Jun. 2018.

[3] InertiaSoft, Inc, "FS One RC Flight Simulator," http://www.fsone.com/, Accessed Oct. 2017.

[4] M. S. Selig, "Real-time flight simulation of highly maneuverable unmanned aerial vehicles," *Journal of Aircraft*, vol. 51, no. 6, pp. 1705–1725, Nov.-Dec. 2014.

[5] M. Selig, "Modeling propeller aerodynamics and slipstream effects on small uavs in realtime," AIAA Atmospheric Flight Mechanics Conference, Toronto, Ontario, Canada, Aug. 2010.

[6] Laminar Research, "X-Plane 11," http://www.x-plane.com/, Accessed Mar. 2018.

[7] J. S. Lee and K. H. Yu, "Optimal path planning of solar-powered uav using gravitational potential energy," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 53, no. 3, pp. 1442–1451, June 2017.

[8] C. Grano-Romero, M. Garca-Jurez, J. F. Guerrero-Castellanos, W. F. Guerrero-Snchez, R. C. Ambrosio-Lzaro, and G. Mino-Aguilar, "Modeling and control of a fixed-wing uav powered by solar energy: An electric array reconfiguration approach," in *2016 International Conference on Power Electronics*, June 2016, pp. 52–57.

[9] X. Gao, Z. Hou, Z. Guo, J. Liu, and X. Chen, "Energy management strategy for solar-powered high-altitude long-endurance aircraft," *Energy Conversion and Management*, vol. 70, no. Supplement C, pp. 20 – 30, 2013.

[10] S. Hosseini, R. Dai, and M. Mesbahi, "Optimal path planning and power allocation for a long endurance solar-powered uav," in *2013 American Control Conference*, Jun. 2013, pp. 2588–2593.

[11] B. B. Lee, P. Park, C. Kim, S. Yang, and S. Ahn, "Power managements of a hybrid electric propulsion system for uavs," *Journal of Mechanical Science and Technology*, vol. 26, no. 8, pp. 2291–2299, Aug 2012.

[12] J. Ostler and W. Bowman, "Flight testing of small, electric powered unmanned aerial vehicles," ser. U.S. Air Force T&E Days Conferences. American Institute of Aeronautics and Astronautics, Dec 2005, 0.

[13] D. Karabetsky, "Solar rechargeable airplane: Power system optimization," in *2016 4th International Conference on Methods and Systems of Navigation and Motion Control (MSNMC)*, Oct 2016, pp. 218–220.

[14] H. B. Park, J. S. Lee, and K. H. Yu, "Flight evaluation of solar powered unmanned flying vehicle using ground testbed," in *2015 International Conference on Control, Automation and Systems*, Oct 2015, pp. 871–874.

[15] P. Lindahl, E. Moog, and S. R. Shaw, "Simulation, design, and validation of an uav sofc propulsion system," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 48, no. 3, pp. 2582–2593, July 2012.

[16] J. B. Bradt and M. S. Selig, "Propeller performance data at low reynolds numbers," in *AIAA Aerospace Sciences Meeting, Orlando, Florida, Jan. 2011.*

[17] J. K. Shiau, D. M. Ma, C. W. Chiu, and J. R. Shie, "Optimal sizing and cruise speed determination for a solar-powered airplane," *AIAA Journal of Aircraft*, vol. 47, no. 2, pp. 622–629, Mar 2010.

[18] W. Khan and M. Nahon, "Modeling dynamics of agile fixed-wing uavs for real-time applications," in *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, Jun. 2016, pp. 1303–1312.

[19] E. N. Johnson and S. Mishra, "Flight simulation for the development of an experimental uav," AIAA Paper 2002-4975, AIAA Modeling and Simulation Technologies Conference, Monterey, California, Aug. 2002.

[20] FlightGear Flight Simulator, "FlightGear," http://www.flightgear.org, Accessed Oct. 2017.

[21] O. D. Dantsker, M. Theile, and M. Caccamo, "A high-fidelity, low-order propulsion power model for fixed-wing electric unmanned aircraft," AIAA/IEEE Electric Aircraft Technologies Symposium, Jul. 2018.

[22] O. D. Dantsker, G. K. Ananda, and M. S. Selig, "GA-USTAR Phase 1: Development and Flight Testing of the Baseline Upset and Stall Research Aircraft," AIAA Paper 2017-4078, AIAA Applied Aerodynamics Conference, Denver, Colorado, Jun. 2017.

[23] Al Volo LLC, "Al Volo: Flight Systems," http://www.alvolo.us, Accessed Mar. 2018.

[24] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *2009 ICRA workshop on open source software, Kobe, Japan*, vol. 3, no. 2, p. 5.

[25] Open Source Robotics Foundation, "Robot Operating System," http://www.ros.org, Accessed Jan. 2018.

[26] R. Mancuso, O. D. Dantsker, M. Caccamo, and M. S. Selig, "A low-power architecture for high frequency sensor acquisition in many-DOF UAVs," in *Cyber-Physical Systems (ICCPS), 2014 ACM/IEEE International Conference on*, Apr. 2014, pp. 103–114.

[27] O. D. Dantsker, R. Mancuso, M. S. Selig, and M. Caccamo, "High-frequency sensor data acquisition system (sdac) for flight control and aerodynamic data collection research on small to mid-sized uavs," in *AIAA Applied Aerodynamics Conference, Atlanta, Georgia, June 2014.*

[28] O. D. Dantsker, A. V. Loius, R. Mancuso, M. Caccamo, and M. S. Selig, "Sdac-uas: A sensor data acquisition unmanned aerial system for flight control and aerodynamic data collection," in *AIAA Infotech@Aerospace Conference, Kissimee, Florida, Jan 2015.*

[29] Hobbico, Inc., "Great planes avis-

tar elite .46 advanced trainer rtf," http://www.greatplanes.com/airplanes/gpma1605.html, Accessed Oct. 2013.

[30] N. Hovakimyan and C. Cao, *L1 adaptive control theory: guaranteed robustness with fast adaptation*. SIAM-Society for Industrial and Applied Mathematics, 2010, vol. 21.