SSC21-V-09

A Novel Approach to an Autonomous and Dynamic Satellite Control System Using On-Orbit Machine Learning

Maximum Wilder-Smith, Michael Pham, Matteo Gironda, Kevin Kwik, Michael Gee, Ryan Toomer,
Samuel Bennett, Nicholas Shewchuk, Ashley McBean, Tarek Elsharhawy
Bronco Space | California State Polytechnic University Pomona
3801 W Temple Avenue, Pomona CA 95136 USA; (323) 377-9601
maxwildersmith@gmail.com

ABSTRACT

Classical control methods require deep analytical understanding of the system to be successfully controlled. This can be particularly difficult to accomplish in space systems where it is difficult, if not impossible, to truly replicate the operational environment in a laboratory. As a result, many missions, especially in the CubeSat form factor, fly with control systems that regularly fail to meet their operational requirements. Failure of a control system might result in diminished science collection or even a total loss of mission in severe circumstances. Additionally, future SmallSat use cases (such as for orbital debris collection, repair missions, or deep space prospecting) shall place autonomous spacecraft in situations where mission operations cannot be fully simulated prior to deployment and a more dynamic control scheme is required. This paper explores the use of a student / teacher machine learning model for the purpose of training an Artificial Intelligence to fly a spacecraft in much the same way a human pilot may be taught to fly a spacecraft. With dedicated Artificial Intelligence & Machine Learning hardware onboard the satellite, it is also hypothesized that deploying an active learning algorithm in space may allow it to rapidly adapt to unforeseen circumstances without direct human intervention. Full development of a magnetorquer only control scheme was conducted with testing ranging from a software-in-the-loop 3D physics engine to a hemispherical air bearing, and finally a planned on-orbit demonstration. Further work is planned to expand this research to translational operations in future missions.

INTRODUCTION

In recent years small satellites, and particularly CubeSats, have exponentially increased in launch cadence. What a few years ago was only a niche industry is now quickly set to outpace the capabilities of traditional heavy launch missions in Low Earth Orbit (LEO). One aspect of space missions that continues to regularly challenge mission success in both small and large form factors is in the effective development of Attitude Determination & Control Systems (ADCS). In addition to the technical challenge of ADCS systems, a failure of an ADCS system can often result in seriously diminished performance or even loss of mission.

In order to ease the immense challenge of implementing ADCS and improve small sat reliability this team has proposed the use of novel artificial intelligence (AI) and Machine Learning techniques. A novel approach, based on a 2020 robotics paper by Lee, Joonho, et al. [1] was implemented to train an algorithm capable of robustly controlling a spacecraft.

Despite the rapid growth of the Cube Satellite industry failure remains abundant, especially in the university space. Langer and Bouwmeester presented a paper at the 30th Annual AIAA/USU Conference on Small Satellites in 2016 that examined CubeSat missions[2]. The paper, titled "Reliability of CubeSats – Statistical Data, Developers' Beliefs and the Way Forward" evaluated data from the CubeSat Failure Database which detailed 178 CubeSat missions at the time of their investigation. From examining these missions, they estimated that ADCS failure could be directly attributed for 3% of CubeSat failures after deployment. It should be noted that failure for unknown reasons also accounted for a sizeable number of CubeSats and could be unattributed failures of ADCS. A failure of a satellite to detumble (or an accidentally induced uncontrolled tumble such as with the MOVE-II CubeSat [15]) may also result in a loss of mission due to a loss of communication.

An example of a CubeSat that suffered ADCS failure is PicSat, which was intended for exoplanet observation. M Nowak, et al [3] detail how PicSat's onboard ADCS successfully de-tumbled the satellite post launch but was unable to orientate itself to point in the direction of the exoplanet. They suggest that the failure may have

been due to a high-level software issue with the system. As CubeSat mission continue to grow in number and complexity it is essential for more robust ADCS systems to become available for mission critical uses.

Existing Methods

The traditional methodologies employed in the design and tuning of an ADCS hinge on the feedback characteristics of the closed loop architecture. A system's dynamic response is impingent on the proper calibration of the various feedback gains and on the completeness and accuracy of the mathematical model used to derive such parameters [4]. As mentioned by de Ruiter, optimizing the control scheme for a space mission's actuators thus requires extremely precise knowledge of the spacecraft's physical properties, operating conditions, and control authority [5]. The nature of this process necessitates testing conditions that closely resemble the space environment. This is a significant bottleneck in the design and system integration process of a robust controls system, as achieving sufficient testing reliability relies on expensive equipment and thorough procedures. This level of rigor is especially unattainable for many university class CubeSat missions, leading to poor implementation or exclusion of ACDS altogether.

Appropriate testing and tuning however does not guarantee performance. Should an actuator or sensor on the satellite fail there is often little recourse that a small satellite can take to remedy it. Component and software redundancies are the only safeguards against unexpected events and conditions during mission operations. This is problematic in the limited design space for SmallSats (especially CubeSats) where adding redundant systems is often not possible due to size weight and power constrains. The ADCS then must be an incredibly high reliability sub-system for missions that require it. As a result, ADCS within small satellites that can afford this often take up a very large percentage of the overall cost of the satellite bus. This paper aims to address the design and testing overhead associated with the development of CubeSats' ADCS, as well as the lack of real-time adaptability of conventional approaches.

APPROACH

To overcome these issues of modern control systems we propose a novel control system that takes advantage of recent developments in artificial intelligence for robotic systems. In doing so, we hope to present a control system that is robust to changes in the satellite's properties via a machine learning through a derived abstract representation of the satellite. One that is versatile in the face of sensor or actuator failures, through action-command feedback loops. And finally, one that is quick to deploy through shared networks for satellites of similar actuation and sensor types.

The end result of this novel control architecture is a software product that is able to learn and actively adapt to its environment the same way a human pilot might. To accomplish this, a teacher-student policy is at the core of the control system architecture. This pair of neural networks involves first training a teacher model for longer periods of time on a rigorous simulation environment, before using the trained teacher to supervise the accelerated learning of a student model. The student model is fine-tuned with real world data from laboratory testing of a specific satellite's control mechanisms, while the teacher model is designed to be more generalized to any satellite of a similar type (same sensors and actuators). This would reduce the needed laboratory time for training, and the reuse of a pretrained complex teacher model would allow for reduced development time of control systems.

SIMULATION ENVIROMENT

One of the most crucial components of the proposed system is a robust simulation environment capable of directly interfacing with the machine learning framework during training. For this simulator, our team is building off the Blender 3D animation platform[6] as it allows for direct pythonic access, has a simple yet powerful UI and visualization framework, as well as a pre-existing physics engine. In the past Blender has demonstrated the compute capability to utilize complex machine learning algorithms for the purpose of quick and accurate simulations such as the Mantaflow project[7] an extensible framework for complex fluid simulation and research. Additionally, the animation engine's frame-by-frame seeking and property updates allow us a unique method for storing and computing properties and their histories throughout the simulation.

Physics Simulation

The angular velocity of the satellite describes the rate at which it rotates about its center of mass per unit time in the simulation. This vector quantity serves as one of the primary inputs for the control system and directly affects the satellite's ability to effectively send communications. While there are multiple ways to represent rotations in three dimensions, we decided to utilize quaternion notation for all functions involving angular velocity. Out of Blender's supported notations, quaternions are the most practical for expressing the satellite's behavior within the simulation, as they can describe three dimensional rotations without any degree of freedom issues [8]. Provided that the angular velocity calculation influences the orientation of the satellite,

and consequently the light sensor and magnetorquer calculations, avoiding this potential issue allows the simulation to provide consistently accurate measurements for the satellite.

In order to produce the angular velocity vector quantity, the difference in the satellite's orientation is calculated with respect to a specified time interval. For purposes of this simulation environment, the time interval is defined by a single frame step. Therefore, the quaternion describing the rotational transformation between the initial and final orientations for the time interval is calculated by multiplying the final orientation unit quaternion by the conjugate of the initial orientation unit quaternion. These operations are performed in the pyquaternion library [9]. Quaternion multiplication is noncommutative so the order in which these quantities are multiplied together is compulsory in producing the proper result [10]. The resultant of this operation will produce the angular velocity quaternion for the given time step. The interval of time to pass between frames is a variable that can be edited during preview, simulation and training.

The satellite's orientation is calculated progressively by utilizing the preceding frame's angular velocity and orientation data. The angular velocity quaternion itself is the product of three unit quaternions, each of which describes the specified change in orientation about the x, y, and z axes. By combining these quantities in the specified order, a single quaternion can be used to describe the multiple rotations necessary to emulate the satellite's behavior between each step of the simulation. Multiplying the quaternion describing the angular velocity with the unit quaternion describing the satellite's orientation at that instant [10].

To simulate the magnetic forces that interact with the CubeSat we use the pip pyIGRF package. This package allows us to find the magnetic field intensity at the location of the CubeSat by providing latitude, longitude, altitude, and time. This package utilizes the 13th and current iteration of the International Geomagnetic Reference Field which is a model for calculating the Earth's magnetic field produced by the International Association of Geomagnetism and Aeronomy [11]. Using the data calculated by pyIGRF we can simulate the interactions of the Earth's magnetic field and the CubeSat's onboard magnetorquers.

Spacecraft Simulation

For purposes of simulating the functionality of a satellite, it was necessary to provide approximations of the necessary input signals from the various sensors present. For the purpose of this initial simulation the virtual spacecraft was based on the 1.5U CubeSat: BroncoSat-1, which featured only a 9-Axis Inertial Measurement Unit (IMU with accelerometer, magnetometer, and gyroscope) and 6 photoresistors (one on each face) as sun sensors.

To produce a rudimentary approximation of the data received from the sun sensors, a trigonometric approach was utilized. Reference unit vectors are created originating from the satellites center of mass, each orthogonal to each other, to represent the default orientation of each face of the satellite. Each of these unit vectors are then rotated by the quaternion describing the orientation of the satellite at a given instance. A vector is then created, originating from the satellite's center of mass, to describe the direction between it and the simulated sun. The angle between the direction vector and the position vectors describing each face of the satellite is then measured and recorded. Subsequently, a corresponding voltage level for the simulated sun sensor is generated based on equation (1), which interpolates the voltages between the minimum and maximum values measured from extensive photoresistor testing trials.

$$V = 3.16 * \frac{90^o - \theta}{90^o} + 0.04V \tag{1}$$

If the angle between any given face unit vector and the direction vector is greater than 90 degrees, the generated value is 0.04 volts, as per the reference data collected through testing. To convert this voltage to the lux value received by the satellite, this value was multiplied by 0.5, as per the documentation provided for the sun sensor used in testing [12]. Additionally, if the satellite is positioned anywhere behind the sun, the generated voltage value will be recorded as 0.04 volts, as it is assumed that trace amounts of light will reach the photoresistors.

The gravitational acceleration experienced by the satellite at its altitude must be calculated to replicate the accelerometer data measured by an IMU in the simulation environment. This gravitational acceleration vector is calculated by utilizing the gravitational acceleration equation (2), where G is the gravitational constant, $\hat{r}$ is a unit vector whose direction is from the center of the earth to the satellite's center of mass, and m_e is the mass of the earth, [4].

$$\hat{g} = -G \frac{m_e}{|\hat{r}|^2} \hat{r} \tag{2}$$

For ease of development and integration with an actual satellite, interfaces between these simulated sensor readings and the actual method calls for sensor polling were developed. A similar interface is implemented for the actuator controls in the simulator and on the satellite. This allows a conventional python control system to be run directly in the simulator where it can get access to data analogous to what would occur on mission. Additionally, it allows for the evaluation of the control systems in failure conditions, where pins are disabled or the intrinsic properties of the sensors and actuators are manipulated mid operation. This allows for a means of testing and verifying robustness in the face of various failure conditions.

Blender Integration

The open-source nature of Blender has contributed to an easily modifiable user interface with established examples and documentation.

For this simulator we grouped all our custom interface elements onto one custom UI panel, shown in Figure 1, viewable in the 3D viewport. Here we include both controls for interacting with the simulation and data readouts. Text readouts are simple text put in the UI panel. This text is either readily available object data in Blender or generated from standalone python functions that access properties of the CubeSat within the simulation environment. Examples of calculated data includes gravitational and magnetic forces experienced by the CubeSat at any given point in time or space. Variables, such as CubeSat position or center of mass, allow both presentation of data and control of that aspect and are implemented by adding the objects properties to the panel. Blender automatically formats these by the number and type of data provided by that property.

For instance, location is populated to three inputs, one for each axis in Euclidean space. However, Blender is primarily an animation and 3D modelling suite, and as such does not have some of our needed functionality. For this reason, to track certain information such as center of mass of the CubeSat, we employ empties. Empties are objects within Blender that represent a single point in space, though as objects they also possess a sense of rotation and scale. We use empties as proxies that either possess the trait we want to track ourselves, such as the rotation of the empty, or are used to calculate the trait we want to track, such as through its position relative to the CubeSat. We then retrieve this trait and display the properties on the UI panel.

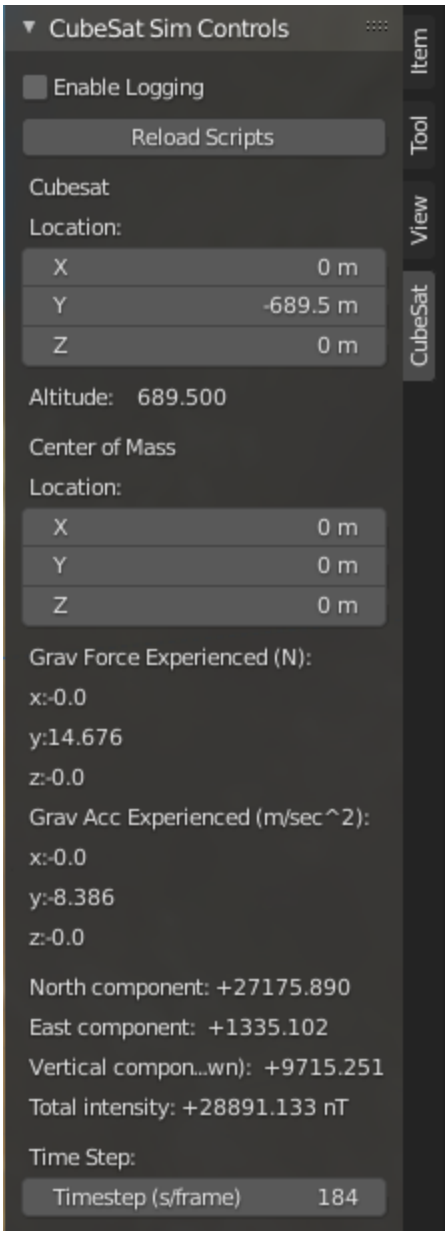

Figure 1: UI Panel

Blender implements its own internal workspace for writing and running python scripts that are stored within the blend file. This space is great for testing scripts, but for our purposes the ability to maintain and modify python scripts outside of the main blend file was preferential, one among the reasons being better integration with version control. To that end we created a script within the blend file that simply looks for a file named scripts.txt and executes the python scripts listed inside. This script is set to execute on opening the blend file so that Pythons scripts such as the UI panel are automatically run and presented to the user on startup.

MACHINE LEARNING MODEL

The machine learning model is based on the architecture presented in "Learning Quadrupedal Locomotion over Challenging Terrain." [1] with a different control scheme and simulation environment. It utilizes the TensorFlow library for python[13] which can be run directly within the Blender simulation environment. As this use case is designed for a CubeSat with three magnetorquers as the actuators, the output action generated by the model only consists of three duty cycles. The proprioceptive data then consists of the IMU data, power readings from six sun sensors, the current magnetorquer duty cycles, and time since last execution. In addition to the current sensor data, the robot state vector includes the previous error between the desired rotation and the actual rotation, as well as the previous duty cycle output. At the end of this robot state data, shown in Table 1, is the command for the desired angular velocity in Euler angles.

Table 1: Robot State Properties

Name	Dimension	Symbol
IMU Accelerometer	3	q_a
IMU Gyroscope	4	q_g
IMU Magnetometer	3	q_m
Solar Sensors	6	s
Magnetorquer Power	3	M_p
RTC Time Since Execution	1	Δt
Rotation Error	3	ω_{err}
Magnetorquer Power History	3	M_h
Desired Rotation	3	ω

This makes up the robot state data and is one of two inputs fed into action-generating network in both the teacher and student. The second input is an encoded latent state generated by each policies respective encoder. Both this generated latent state and the action output of final network in each policy are intended to be the same. In training, the teacher is first trained in the simulator using reinforcement learning as there is not a target actuator output for the model optimize for. Once the teacher performs adequately, the student is trained to mimic the teacher during knowledge distillation[14].

As the teacher generates actions and a latent state we want the student to mimic, the student can be trained through traditional supervised methods. Following the knowledge transfer between the two policies, the student model will be tested and fine-tuned in a laboratory environment before it is isolated from the rest of the system for use in an actual CubeSat.

Teacher Model

The first phase of the machine learning system is to create a robust teacher model trained off the simulator. This model is more complex than the one intended to run on the actual satellite and has access to information that the satellite would not normally have. This privileged information includes a global reference position, exact altitude, the center of mass and the actual direction of the sun. The data is then encoded to produce the latent state vector which serves as an abstraction of the environment. This latent state is then fed into the action-generating network along with the robot state data to produce some action. The action consists of three duty cycle, one for each magnetorquer, to interact with the simulator's magnetic fields.

For training of the teacher model, a reinforcement learning system is used where the reward is based on the new angular velocity that results from the output action of the model. Each action and rotational error is calculated on a frame-by-frame basis. Where the time step property serves as Δt in the robot state. In practice this would represent delays between running and updating the output of the control system due to power or computational constraints. This time step value is to be varied throughout training for a more robust solution for various hardware configurations. Once the generated actions produce the desired reaction of the CubeSat, the action-generating network is saved and its weights and biases are copied over to the student model.

Student Model

The second phase of this machine learning system is training the student model. By leveraging the knowledge distillation aspects of the teacher-student system[14], we create a smaller network capable of training and inferring from a smaller dataset than the teacher. In practice this means generating a simpler network capable of running on the limited hardware of the CubeSat, and one that requires less training time in physical testing facilities. This reduces both the development time and costs for future deployments of the control system.

Unlike the teacher model, the student only has access to the proprioceptive history and must infer the same latent state representation of its environment that the teacher generated. As there is now a known output value for each input, a standard supervised learning system is used for training the student's encoder. As the action-generating network ideally has the same input as the network in the teacher, robot state and

environmental latent state, the actions of the both the teacher and student should be the same for a given frame.

The supervised training results in a system where the simulation evaluates the teacher policy for a given configuration. The student uses the same action-generating network as the teacher, and updates its encoder to achieve the same latent state as the teacher and the same action. Following initial training in the simulation to ensure parity between the teacher and student policies' latent state, a final round of laboratory training will be used to verify and fine-tune the action-generating network. Like before, the training will be motivated by the reinforcement learning loop.

The training environment is shown in Figure 2 where a Helmholtz cage encloses a low-friction air bearing and is monitored by motion tracking cameras. Through the cage various magnetic fields can be generated for the CubeSat's magnetometers to interact with. The air bearing serves to reduce the impact of friction on the effects of the actuators. Finally, the motion tracking cameras allow for external monitoring and data collection for analysis.

The final trained student policy is then saved as a stand-alone model, taking the robot state and proprioceptive history as input, and generating an action for the magnetorquers as output. This smaller isolated model is then to be deployed on the CubeSat. As the model can be exported with standard graph formats, it need not be run on a python flight computer and can be executed on system capable of performing inference on a TensorFlow network, and that possess a sufficient computing capability to run within a reasonable time. Additionally, as the latent state output of the two policies are similar and the action-generating networks are the exact same, changes to the outputs or additional training within the simulation environment can easily be ported computer through the exported model.

CURRENT STATUS & FUTURE WORK

Currently, the AI system has been created in Python, but awaits completion of the simulation environment and an improved training dataset before full implementation on a flight system. One of the most serious issues encountered by the team in the course of its implementation of the AI system is a lack of useable training & reference data for magnetorquer only control systems. In the literature review there does not seem to exist any on-orbit data that is both publicly accessible and continuous that would act as a reference for real world magnetorquer only control.

Additionally, while a combined 3-Axis Helmholtz coil and hemi-spherical air bearing system has been created for ground testing of the BroncoSat-1 magnetorquers control scheme, it has proven incredibly difficult to separate the effect of the magnetorquers from ambient disturbances. Issues such as ambient air currents, an unrepresentative moment of inertia, and standard CubeSat flight hardware complications have stalled a laboratory validation of the AI control system.

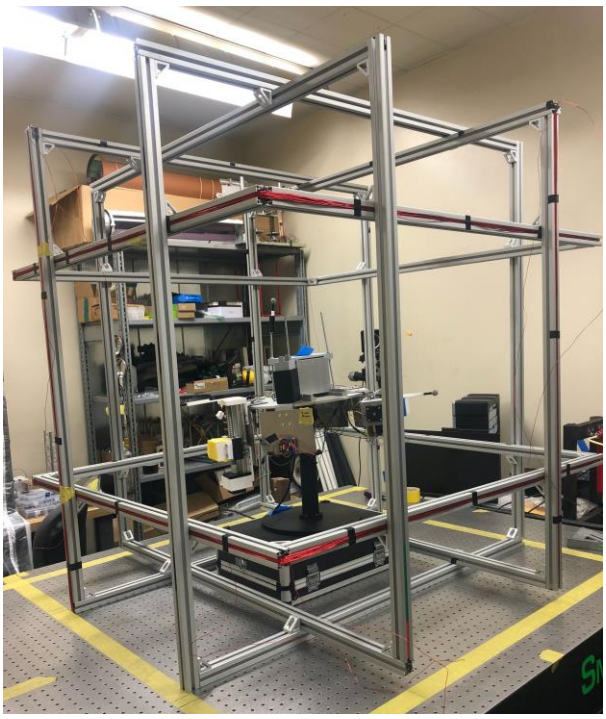

Figure 2: Laboratory Testing Facility

Improvement of the simulator and testing facility is ongoing, and it is believed that within a few months significant progress shall be made towards laboratory validation of the AI control system. Work is under way to incorporate more complex actuators into the simulator, such that the control system can generate outputs to reaction wheels and similar systems. With a more powerful actuator it is also believed that laboratory test and training of the control system shall be greatly simplified. The ability to model translational motion within the simulation environment is also planned.

With regards to the testing facility, software and procedures are being developed to measure and control the CubeSat and environment more accurately. This will allow for analysis of the performance of the proposed control system as well as comparisons to a traditional system in real-world environments. A planned overhaul of the hardware interface system in the simulator is intended to allow the execution of

complete flight software with communication, error creation and reporting, as well as payload interaction. This will then allow for verification of system robustness, and an evaluation of the proposed system in traditional failure conditions.

Finally, once testing and validation are complete, the trained model would be deployed on a CubeSat with an appropriate AI-accelerated computer. This would allow for the collection of mission data that can be used to make revisions to the system and potentially open up the use of the proposed system as an alternative to traditional control systems for other missions. Especially within the university space, this is expected to have huge benefits for mission reliability and simplifying the implementation of ADCS in CubeSats.

ACKNOWLEDGEMENTS

This work, over the course of the last year during the COVID-19 pandemic, would not have been possible without the incredible efforts of the Bronco Space Student Research Group, the Colleges of Engineering and Science at Cal Poly Pomona, and the support of our advisors and industry partners.

References

1. Lee, Joonho, et al. "Learning Quadrupedal Locomotion over Challenging Terrain." *Science Robotics*, vol. 5, no. 47, 2020, doi:10.1126/scirobotics.abc5986.
2. Langer, M., Bouwmeester J. "Reliability of CubeSats – Statistical Data, Developers' Beliefs and the Way Forward." 30th Annual AIAA/USU Conference on Small Satellites. https://digitalcommons.usu.edu/cgi/viewcontent.cgi?article=3397&context=smallsat
3. M. Nowak, S. Lacour, A. Crouzier, L. David, V. Lapeyrère, and G. Schworer "Short life and abrupt death of PicSat, a small 3U CubeSat dreaming of exoplanet detection", Proc. SPIE 10698, Space Telescopes and Instrumentation 2018: Optical, Infrared, and Millimeter Wave, 1069821 (24 July 2018); https://doi.org/10.1117/12.2313242
4. J., D. R. A. H., Damaren, C., & Forbes, J. R. (2013). *Spacecraft dynamics and control an introduction*. Wiley.
5. Carletta S, Teofilatto P, Farissi MS. A Magnetometer-Only Attitude Determination Strategy for Small Satellites: Design of the Algorithm and Hardware-in-the-Loop Testing. *Aerospace*. 2020; 7(1):3. https://doi.org/10.3390/aerospace7010003
6. Community, B. O. (2018). Blender - a 3D modelling and rendering package. Stichting Blender Foundation, Amsterdam. Retrieved from http://www.blender.org
7. Xie, Y., Franz, E., Chu, M., & Thuerey, N. (2018). tempoGAN. *ACM Transactions on Graphics*, *37*(4), 1–15. https://doi.org/10.1145/3197517.3201304
8. "Kinematics of Moving Frames." *Ocw.mit.edu*, Massachusetts Institute of Technology, 2004, ocw.mit.edu/courses/mechanical-engineering/2-154-maneuvering-and-control-of-surface-and-underwater-vehicles-13-49-fall-2004/lecture-notes/lec1.pdf.
9. Wynn, Kieran. *Pyquaternion*, kieranwynn.github.io/pyquaternion/.
10. Graf, Basile. "Quaternions and Dynamics." *ArXiv.org*, Cornell University, 18 Nov. 2008, arxiv.org/abs/0811.2889.
11. Alken, Patrick. IAGA V-MOD Geomagnetic Field Modeling: International Geomagnetic Reference Field IGRF-13, 19 Dec. 2019, www.ngdc.noaa.gov/IAGA/vmod/igrf.html.
12. *Ambient Light SensorSurface - Mount ALS-PT19-315C/L177/TR8*. Everlight, 24 Dec. 2013, cdn-shop.adafruit.com/product-files/2748/2748+datasheet.pdf.
13. Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
14. Abbasi, S., Hajabdollahi, M., Karimi, N., & Samavi, S. (2020). Modeling Teacher-Student Techniques in Deep Neural Networks for Knowledge Distillation. *2020 International Conference on Machine Vision and Image Processing (MVIP)*. https://doi.org/10.1109/mvip49855.2020.9116923

15. Ruckerl, Sebastian, et. al. 2019. “First Flight Results of the MOVE-II Satellite” *Proceedings of the 33rd Annual Small Satellite Conference*, A Look Back : Lessons Learned, SSC19-WKI-07. https://digitalcommons.usu.edu/smallsat/2019/all 2019/49/