72nd International Astronautical Congress (IAC), Dubai, United Arab Emirates, 25-29 October 2021.

IAC-21,D5,1,x65885

The Architecture of a Safe Low Cost Earth Based Lunar Landing Test Bed for the Validation of Experimental Flight and new Technologies

Michael Smat[a]*, David Barnhart[b], Antariksh Narain[c], Isabel Brieler[d], Dimitri Gianousopolous[e], Anirudh Sharad[f], Reese Weingaertner[g], Hubert Wang[h], Jose Orozco[i], Thanh Tran[j], Shreya Nagpal[k], Noah Foster[l]

[a] *University of Southern California, Los Angeles, California, 90089,* msmat@isi.edu
[b] *University of Southern California, Los Angeles, California, 90089,* barnhart@isi.edu
[c] *University of Southern California, Los Angeles, California, 90089,* antariks@usc.edu
[d] *University of Southern California, Los Angeles, California, 90089,* brieler@usc.edu
[e] *University of Southern California, Los Angeles, California, 90089,* gianouso@usc.edu
[f] *University of Southern California, Los Angeles, California, 90089,* asharad@usc.edu
[g] *University of Southern California, Los Angeles, California, 90089,* weingaer@usc.edu
[h] *University of California, San Diego, La Jolla, California, 92093,* hwwang@ucsd.edu
[i] *University of California, San Diego, La Jolla, California, 92093,* jmo16@ucsd.edu
[j] *University of California, Berkeley, Berkeley, California, 94720,* thanhtran@berkeley.edu
[k] *University of California, Berkeley, Berkeley, California, 94720,* shreyan@berkeley.edu
[l] *Wichita State University, Wichita, Kansas, 67260, nafoster@schockers.wichita.edu*
* Corresponding Author

Abstract

A propulsive landing on the surface of an extraterrestrial body requires a robust vehicle with a guidance, navigation and control (GNC) system that is reliable, efficient and repeatable. Developing algorithms for these systems involves the creation of a mathematical model to simulate reality, and the testing of physical hardware to validate the results produced by the simulations. Unique design considerations for the structures are required for off-nominal flight in 1G to avoid damage yet still allow the vehicle to re-fly quickly. The validation of experimental control algorithms requires the development of necessary infrastructure to iterate through a virtual to physical testing process, which is both time and cost intensive. The University of Southern California's (USC) Space Engineering Research Center (SERC) in collaboration with the University of California at Berkeley (UCB) and the University of California at San Diego (UCSD) has developed such an infrastructure for an earth-based lunar landing test bed capable of validating experimental GNC algorithms with measures designed into both the hardware and software of the vehicle to mitigate failures in the event of off nominal flight conditions, allowing for innovative landing solutions to be repeatedly tested at a higher rate. The Lunar Entry Approach Platform For Research On the Ground (LEAPFROG) is a flight vehicle funded under a NASA Artemis STEM Competition Pilot award with the goal of supporting a nation-wide competition among universities. Powered by a central 300 N thrust turbine jet engine, the vehicle includes a cold gas attitude control system (ACS) to maintain stability, and a gimbal controlled by linear actuators to achieve thrust vector control (TVC) responsible for translation of the vehicle. Structurally, a number of innovations are built in for safety and reliability, including a composite based chassis and roll cage designed using Ansys Composite PrepPost (ACP) to support and protect the critical hardware, as well as a mechanical fuse allowing the frame and legs to avoid excessive loading in the event of a free fall. Additionally, the software architecture monitors competition teams' inputs during flight that can override the controls and land the vehicle safely in the event of a policy violation. This paper will expand on the design and analyses of the features implemented in the structural and software designs that ensure a safe validation of innovative GNC algorithms on this lunar landing platform for use worldwide as a low-cost testbed for advanced technology testing.

Keywords: lunar lander, control systems, lander structures, NASA Artemis Challenge

1. Introduction

1.1 NASA Artemis Challenge

NASA's Artemis Student Challenges are a set of NASA-funded competitions and initiatives that aim to engage undergraduate and graduate students and increase interest in the NASA Artemis Mission. These challenges revolve around finding solutions to common problems and anticipated hurdles during space flight and exploration.

The LEAPFROG (Lunar Entry and Approach Platform for Research On Ground) challenge was organized by the University of California, San Diego in collaboration with the University of Southern California's Space Engineering Research Center (SERC) and the University of California, Berkeley's

Space Sciences Laboratory. This challenge aimed to engage undergraduate and graduate students across the nation in a Lunar Lander skills competition, wherein competitors gain and illustrate the ability to control a lunar lander prototype.

The first version of LEAPFROG, Generation-0, was built by undergraduate students at USC's SERC in 2006 as a reusable ground-based flight simulator for lunar lander technology testing. Since then the project cycled through various iterations, changing designs based on project requirements at each stage. This version of LEAPFROG made for the NASA Artemis Challenge aimed to iterate on the past versions of the vehicle and create a deliverable that could be distributed to teams of students across the country in a safe and educational way.

1.2 Vehicle Design Overview and Approach

To promote the vehicle's ability to safely validate experimental flight control algorithms, LEAPFROG was designed with five major subsystems: structures, propulsion, attitude control system, software, and avionics. Each subsystem was developed in accordance to the testing schedule created at the beginning of every semester. Priority for the team's resources were given to those systems which were required for the most immediate testing campaign, which isolated the subsystems from one another during their development, allowing each to be fully validated prior to its integration with the vehicle and other subsystems. This approach aligns with the primary goal of the vehicle: to create a test bed with features embedded into the design that mitigate failures when off-nominal flight conditions occur.

Figure 1 and Figure 2 show two views of the vehicle with the various subsystems and its global coordinate system which will be referred to throughout this paper.

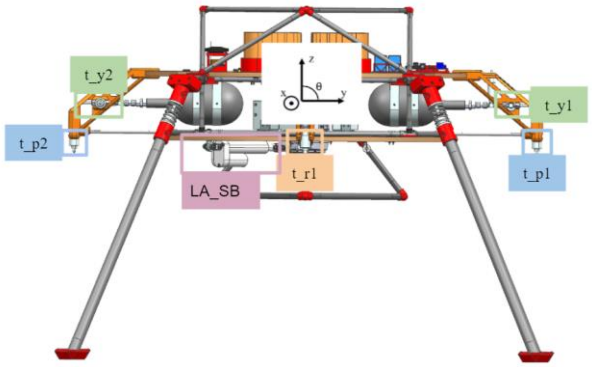

Figure 1: Side view of vehicle with labelled coordinate systems

2. Vehicle Design

2.1 Structures

The structure was designed such that all hardware required for the various subsystems could be rigidly fixed to the vehicle in their appropriate positions while remaining under the maximum weight as determined by the maximum engine thrust and flight performance goals. To do so, the structure was designed with four main components: its chassis, mounting platforms, legs, and roll cage. All of which were made from composite materials to maximize the vehicle's strength to weight ratio.

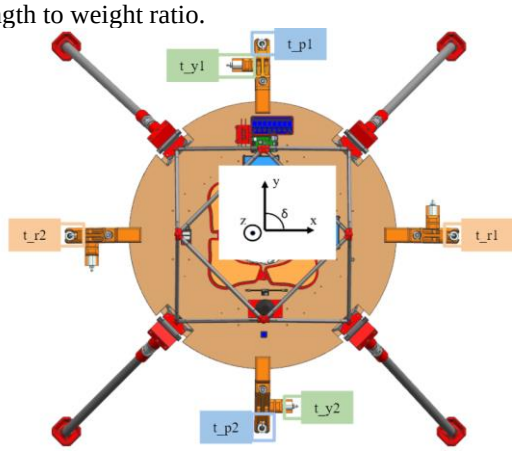

Figure 2: Aerial view of vehicle with labelled coordinate systems

The chassis consists of COTS half-inch, twill weave carbon fiber tubes from Rockwest configured in an octagonal structure as seen in Figure 3. On the first level, the central octagon supports the thrust of the engine, and the outer octagon supports the reaction forces of the linear actuators used for thrust vectoring the engine as well as providing mounting points for the struts used to connect the top layer to the bottom layer. The primary function of the top layer is to mount the engine fuel tanks, attitude control system air tanks, and avionics. The tubes were connected to each other using commercial off the shelf (COTS) threaded and unthreaded clevis connectors from Dragon Plate.

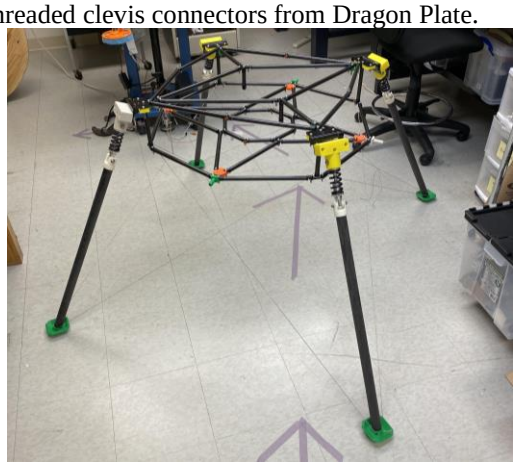

Figure 3: Assembled carbon fiber chassis with legs integrated

A chassis structural analysis was performed using an ABAQUS truss analysis simulation to determine the stresses within the chassis when the vehicle free falls from a 5-meter height. The result showed that the vertical carbon fiber struts experience significant tension and bending from the pulling of the

strings attached to the legs. This led to the addition of the mechanical fuses which will disconnect the strings and protect the chassis in the event of a hard landing.

The mounting platforms provide a greater surface area to which the hardware can be fastened and adds rigidity to the chassis against moments in the x and y directions as established by the coordinate system in Figure 2. Both platforms are composite sandwiches manufactured in house with five pound density foam from FiberGlassSupply and two layers of fiberglass fabric impregnated with two part epoxy resin from WestSystems. The lay-up sandwich was placed under vacuum for 3 hours until cured. The fiberglass allows fasteners to clamp flanged mounts to the platforms without penetrating the foam.

A protective vehicle roll cage was designed consisting of both an upper and lower structure. The upper roll cage was designed to protect the various avionics hardware and fuel tanks mounted on the top shelf of the vehicle in the event of a roll over. The lower roll cage was designed to protect the jet engine and bottom platform from being damaged in the event of the mechanical fuses disengaging. Both roll cage structures were analyzed using ABAQUS truss analysis simulation to confirm their protective capabilities in the intended situations.

2.2 Vertical and Horizontal Propulsion

LEAPFROG utilizes a JetCat P300 Pro engine, shown in Figure 4, to enable it to lift off the ground and simulate various levels of gravity. This is a COTS air-breathing jet turbine engine fuelled by kerosene with the performance metrics as shown in Table 1.

Figure 4: JetCat P300 Pro engine

This engine simplifies the propulsion system due to its integrated design. The JetCat P300 Pro houses within itself the ECU, fuel pump, start valves, fuel filter, starter, pressure sensor, igniter, and glow plugs. Using an electric starter required a large energy output and therefore battery but allowed space and mass to be used on an auxiliary starting system. The only propulsion-related hardware required to be integrated onto the vehicle is the remaining parts needed to complete the fuel system.

Table 1: Performance metrics for JetCat P300 Pro engine

Metric	Value	Units
Mass Flow Rate	0.5	[kg/s]
Weight	2.73	[kg]
Diameter	13.2	[cm]
Length	38.1	[cm]
Exhaust Gas Temperature	480-750	[C]
RPM at Idle	35000	[RPM]
Max RPM	105000	[RPM]
Thrust at Idle	14	[N]
Thrust at Max RPM	300	[N]
Exhaust Gas Velocity	2160	[km/h]

The fuel system, as shown in the aerial view of the vehicle in Figure 5, consists of two Jet Model Products (JMP) T-33 fuel tank sets, COTS tanks made from fiberglass by Jet Tech. The four primary tanks are positioned about the center axis of the vehicle, forming a rough toroidal configuration about the air-inlet hole on the top platform. This allows the center of gravity of the fuel system to remain aligned with the center axis of the vehicle. However, this is only true assuming that the fuel draw distribution is even across the four tanks. To ensure this would occur, the fuel lines cut from each tank are the same length, resulting in an equal pressure drop over the path traveled from each tank. Additionally, the filling procedure of the tanks ensures that each begins with the same mass of fuel. During the hover flight testing campaign, tests were conducted to validate this assumption.

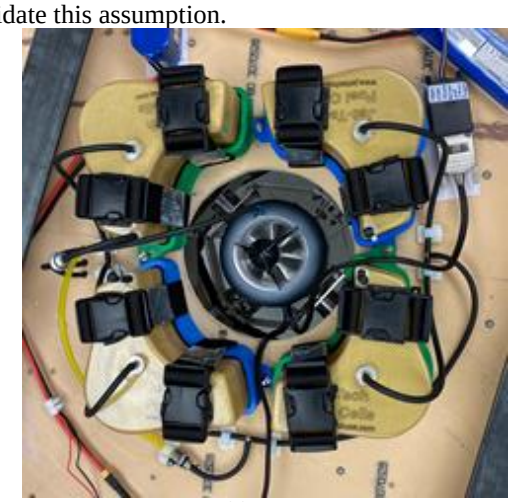

Figure 5: Aerial view of fuel plumbing integrated on vehicle

Each tank has a capacity of 52 ounces. This volume would allow for a flight time of approximately 8 minutes. However, due to weight limitations, each tank is only filled with 0.8 kg of fuel resulting in a flight time of approximately 3 minutes. To decrease weight and complexity, no sensors were used to monitor the fuel levels during flight. Instead, a flight time calculator was

created to determine how much fuel remained in the tanks during tests. Mounting platforms for the tanks were 3D printed to match the shape of the tanks, allowing them to be strapped directly to the platforms. This design made the tanks rigid during flight, but able to be removed while weighing the tanks during the fueling process.

To enable the vehicle to translate in X or Y axes, an in-house manufactured gimbal shown in is used for thrust vectoring the engine. Consisting of three rings and including attachment points both to the main structure and to the linear actuators, this enables the engine to be controlled in both pitch and yaw.

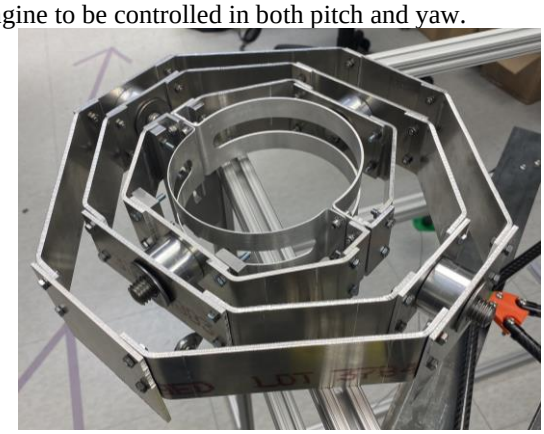

Figure 6: Gimbal used to thrust vector the engine

2.3 Control Systems

The LEAPFROG control system performs two primary functions: ensuring the safe flight of the vehicle and performing legal commanded maneuvers. This design allows the vehicle to navigate as commanded through three-dimensional space as directed by the user, so long as the flight commands provided do not endanger the vehicle. This design philosophy for the vehicle control system was implemented to meet the design goal of providing a safe and reusable platform for testing and competition surrounding flight software.

The high-level command architecture of the LEAPFROG vehicle is split into two actors. The first is the competition or test code, which controls the gimbal and throttle of the engine as well as yaw of the vehicle to autonomously navigate translationally through three dimensions. The second actor is the vehicle flight software (FSW), which itself serves two purposes. Firstly, the FSW constantly actuates the cold gas subsystem to maintain the roll and pitch of the vehicle at level. Secondly, the FSW serves as a safety net for the test code. Because the test or competition code will, by nature, not be rigorously tested before implementation, it carries a higher degree of risk than would normally be acceptable. This is mitigated by allowing the flight software to automatically override illegal or dangerous commands provided by the test code. Some examples of such prohibited commands are putting the vehicle into too aggressive of a maneuver, or postponing landing beyond the point at which the vehicle runs out of fuel. The FSW also allows for manual commands to be sent in real time from the ground station, overriding the autonomous modules if it becomes apparent that the vehicle is behaving unsafely.

To further isolate test code from the vehicle's FSW, the two modules are separated into different hardware. The FSW is run on a Raspberry Pi controller while the test code is run on a PX4 capable Cube, or PixHawk. This decision was made such that even if the test code were to, for instance, crash the process, the FSW itself will continue to run and allow the vehicle to safely land. The autonomous command flow architecture is shown in Figure 7 below.

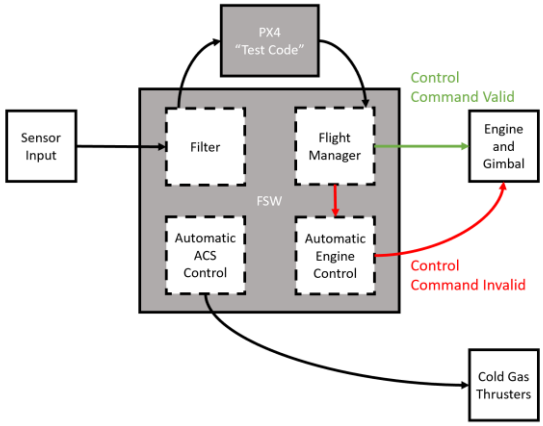

Figure 7: Control flow diagram. Flight manager determines whether control commands sent by PX4 Test Code is legal, and decides to use either command or built-in automatic control.

Further description of the LEAPFROG control system begins with detailing variables which the system must keep track of and control: state variables. The state variables of the system fall under one of two categories: commanded state variables, and supplementary state variables. Commanded state variables are those which are set to target a desired value, upon which effectors are utilized to bring the measured value to the desired value via the implemented control scheme. Supplemental state variables describe aspects of the system which influence its behavior and may also be influenced by setting a desired state, but do not have a set point commanded. A list of state variables tracked by the control system is shown in Table 2 below.

To illustrate this nomenclature, an example of the cold gas system is provided. The set point of the roll axis is to keep the vehicle level with respect to the ground. Thus, the roll angle is a commanded state variable. Cold gas thrusters are utilized to maintain the vehicle at level, and as the thrusters are fired, the amount of mass in the air tanks changes. This affects the dynamics of the vehicle and is kept track of using pressure transducers. However, the mass of the tanks is not controlled, so it is a supplementary state variable.

The LEAPFROG control scheme maintains safe operations by bounding certain state variables into a legal range. The legal ranges of certain states – subject to change – are provided in Table 2. Should the test code send a command to the flight software that brings the vehicle outside of these legal ranges, the command will be overridden, and automatic emergency procedures will take over. These include righting the vehicle, slowing translation to a standstill, and performing a controlled landing.

Table 2: System state variables. Control variables in white boxes, supplementary state variables in gray boxes.

V_x	Translational velocity in x	2 m/s
V_y	Translational velocity in y	2 m/s
V_z	Translational velocity in z	1.5 m/s
Alt	Altitude above ground	0-6 m
R,P,Y_vehicle	Vehicle roll, pitch, yaw	0, 0, 0-360 degrees
Th_percent	Percentage of total thrust	85-100%
R, P_gimbal	Roll and pitch angle of gimbal	$\|\alpha,\beta\| < 10$ degrees
M_fuel	Mass of kerosene fuel	M > 15%
M_gas (1-4)	Mass of any cold gas tank	M > 15%

The overall control loop of the system can be broken down into three main sections: sensors, the control logic, and the effectors. The sensors of the system update the measured state variables, the control logic takes in the states and commands to actuate the effectors appropriately, and the effectors influence the physical state of the vehicle. The negative feedback loop continues indefinitely to minimize the error as new commands arrive and disturbances are imparted to the system.

The LEAPFROG control system is generally divided into two control schemes: engine control for translation and attitude control for rotation. These two schemes together have a mutually exclusive, completely exhaustive control over the six dynamic degrees of freedom of the vehicle. The negative feedback control loop including these two control schemes are illustrated in the figure below.

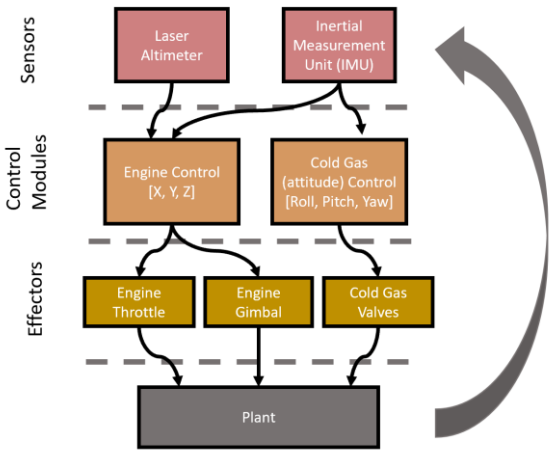

Figure 8: Sensors, control modules, and effectors visualized in a control loop.

Navigation and controlling the translational motion of the vehicle in flight is the primary goal of the user-implemented control test code on the Pixhawk. The Pixhawk will be fed filtered altitude and acceleration data from the sensors, and can output commands to affect the gimbal angle and the engine thrust.

In the case that a state's bounds have been exceeded, the flight manager will instead enable the flight software's automated engine control and will bypass the commands sent from the Pixhawk. The automated controller (FSW Engine Control) will attempt to lower the vehicle's translational velocity and initiate a landing sequence. This architecture is visualized in Figure 9 below.

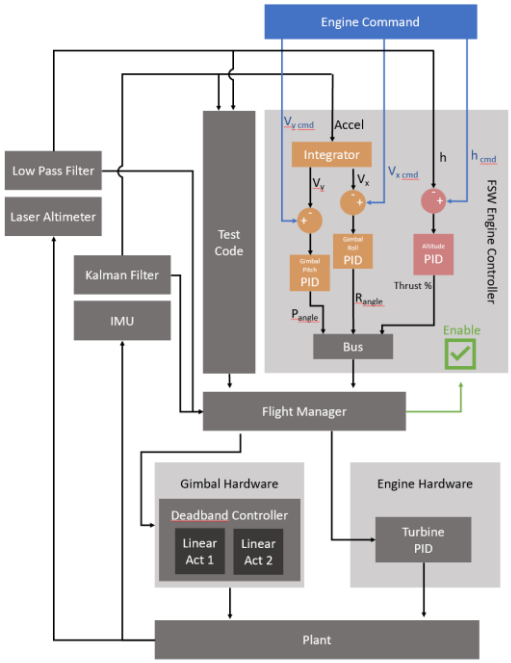

Figure 9: Engine control loop illustrated. The altitude control is shown in red, while the X / Y velocity control is shown in orange.

The cold gas system, or attitude control system (ACS) governs the rotational degrees of freedom of the vehicle: roll, pitch, and yaw. The goal of the controller is to maintain the vehicle in level flight and oriented in the correct direction by actuating the vehicle's six cold gas thrusters. Roll and pitch control are outside the scope of the PixHawk controller and are always controlled autonomously.

The ACS feedback loop begins with sensor measurement via the onboard WT61 IMU. This sensor has a built-in integrator and Kalman Filter to provide absolute angle feedback at a maximum rate of 100 Hz and minimum rate of 10 Hz. The cold gas thrusters which are utilized in the ACS are actuated via electronic relays with a 10 ms operation time and 5 ms release time. Most critically, the maximum on/off switching rate for the relays is approximately 2 Hz.

At the heart of the control loop are two discrete PID controllers. Measurements from the IMU are taken at a rate of 10 Hz, wherein the error is passed into the controller. To transition between the rates of the sensor and the thrusters, a moving average of the previous 5 outputs is performed, resulting in a total control output value. This is then converted into a percentage - this percentage dictates the length of the thruster pulse over the next 0.5 seconds. In this way, the ACS uses pulse-width modulation to accommodate the binary on/off behavior of the thrusters and the maximum switching rate.

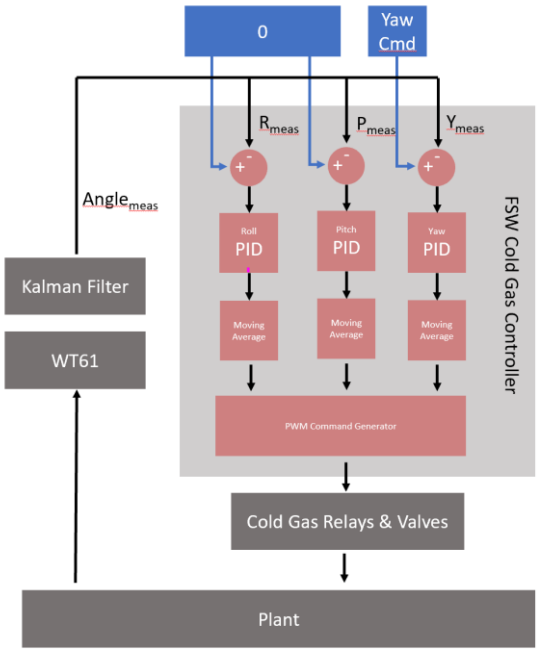

Figure 10: ACS control loop illustrated. Roll, pitch and yaw are controlled by parallel PID controllers, with roll and pitch set to an angle of zero permanently.

Although the different controllers have been separated as much as possible to aid in simplicity and robustness, there still remained couplings between the control systems. Some of these were determined to be negligible, such as the vertical force induced by the 3N cold gas thruster vs the 300N engine. However, it was determined that the distance between the center of gravity and center of thrust would determine how negligible the other coupling effects would be, such as the moment the engine makes on the vehicle when gimballing. Due to this, tooling was developed during the manufacturing and integration of the vehicle to ensure all hardware would be positioned as designed. Additionally, testing procedures were performed following the vehicle's integration to quantify the location of the center of gravity.

Certain states of the system are not commanded or controlled but do affect the way that the vehicle responds to control forces. This includes the mass properties of the vehicle, which change as the engine and ACS fuel are depleted through the firing duration. The loss of this mass changes the moments of inertia, total mass, and center of mass of the vehicle. In a Simulink six degree of freedom dynamics simulation of the vehicle, the controllers were tuned so as to maintain adequate control throughout the entire duration of flight.

With the controllers tuned to satisfaction in simulation, Simulink C++ Code generation is used on the controllers to convert them into discrete C++ objects. These controller objects are then implemented into the ROS2 flight control software and run at a constant loop rate. By using code generation, it is ensured that the controllers on the vehicle behave as close to identically on our real time embedded hardware as in the simulation.

While the Simulink simulation is used to help drive the development of the control system, testing with hardware was required prior to their integration on a flight test. Information regarding the control system test beds is in Section 3.1.

2.4 Software

The software is developed on the backbone of ROS2. It leverages ROS publisher subscriber, server client and parameters for inter-node communication. ROS2 was chosen for its serverless architecture, IPC (Inter-process communication), and compatibility with several plugins that simplify the software implementation. Every sub-component runs as an independent node to leverage the multi-processor system, threads are used to run programs responsible for continuous data collection and monitoring. Each communication protocol UART and I2C has a thread running which sends and receives data on the port assigned to it at the time of initialization. The solution is divided into 4 sub-packages namely, actuators, sensors, communication, and monitoring.

Communication module establishes network and port communication with different peripherals. It parses the data and makes it available to the nodes. As different sensors and actuators have different packet structure being transmitted, it provides functionality for encoding and decoding each packet. The sensor package consumes the communication module to get raw data and process it to generate information by implementing filters and data pre-processing. Once the data is processed, it is published to be consumed by other modules.

Actuator module also uses the communication module, primarily to communicate with the engine. The other actuators are solenoid valves (digital) and linear actuators (PWM). The engine has different commands for setting engine states and retrieving information like health and run statistics from it. All the controls are wrapped into functions that are available as services. The monitoring node is responsible for setting up communication between the vehicle and ground station. It also monitors system health and resource information. There are simple contingency plans defined in case of communication loss or component failure which takes precedence in priority over any other tasks, keeping the vehicle safe as much as possible.

Once the power is connected, the system boots up and starts the program. The program initializes ROS nodes mentioned above and checks all systems are working. Then it tries to establish a connection with the ground station. If it fails, it shuts down the system after the timeout, else sends a heartbeat to the ground station. The user can now send commands to test individual components and monitor the vehicle wirelessly. Once all the system is tested and verified, the GCS can be used to send flight commands and waypoints. The onboard control algorithms generate a plan and send TVC commands to move the gimbal and control engine thrust. The system status is relayed back to the ground station for real time analysis.

During the flight testing phase, we implemented a few features to the software architecture based on our flight tests. The first feature is the ability for the ground station to label each test flight, streamlining the data processing stage of our test flights. The second feature is maintaining the ACS system while the vehicle goes into a shutdown sequence. This is a safety-focused procedure since the vehicle must always attempt to stabilize itself.

An important discovery during our flight testing phase was that the location of the engine's glow plug did not allow for our engine to start upright. Our solution was to tilt the engine 10 degrees in the pitch direction using the gimbal, allowing the glow plug to ignite the fuel.

In order for LEAPFROG to hover, the vehicle must maintain its altitude and attitude. In order to achieve altitude hover, the SEN0259 laser altimeter continuously publishes height data to the engine node. Utilizing a function call in the flight manager node, the engine node receives the laser altimeter data and passes the data into the altitude PID controller helper library. The altitude PID controller generates a thrust value back to the engine node which we send a thrust service request to the JetCatP300. Using this communication protocol, we are able to verify that the thrust values change when the height of the vehicle changes.

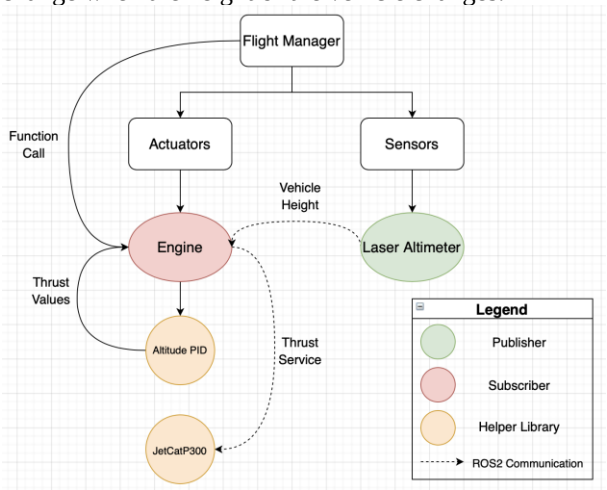

Figure 11: Communication protocols for altitude hover

To achieve attitude hover, the MPU6050 sensor continuously publishes IMU data to the ACS node. Utilizing a function call from the flight manager node, the ACS node receives the IMU data and passes it to the attitude PID controller helper library. The attitude PID controller generates a direction and time back to the ACS node which we then actuate the corresponding cold gas thrusters. Using this communication protocol, we are able to verify the polarity of the ACS response when disturbing the vehicle. Because of noisy IMU data, it is a challenge to validate and tune the ACS response.

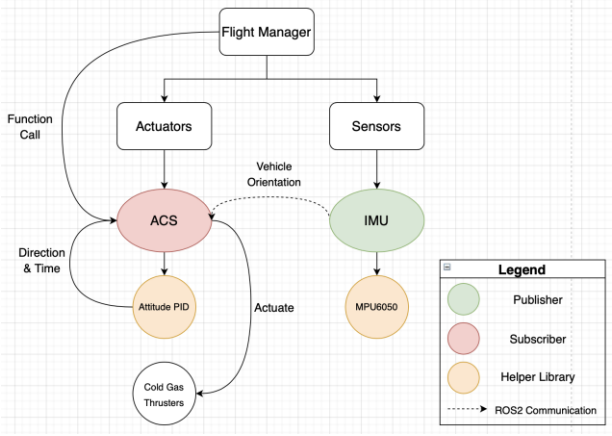

Figure 12: Communication protocols for attitude hover

By leveraging the ROS2 communication system, we are able to manage ground station inputs to control the desired actuators and sensors on our vehicle. The publisher / subscriber method allows for simple communication between sensors and actuators. The server / client method allows for control over the functionalities of the vehicle, such as enabling and disabling the ACS.

Our current obstacle is the noisy IMU data. We believe we must pass the IMU data through a Kalman filter before sending it to the ACS node in order for the vehicle to get a more accurate representation of its orientation. Another possible solution is to purchase an IMU with built-in noise filters.

2.5 Avionics

The avionics onboard need to be low cost, power efficient and reliable. To achieve this the system is divided into 5 parts namely, compute, ACS, TVC, communication and power. The compute module should provide enough headers to connect all components and enough computation power. The solenoids in ACS should have fast actuation and sensors to provide orientation information of the vehicle. TVC system should provide gimbal action to the engine and control the engine thrust with telemetry. The system should have enough power for a single flight and redundancy for backup. Lastly, the communication should be encrypted and operate in the RC bandwidth without overloading it.

Communication between the vehicle and ground station is done via bi-directional RF 900MHz band. RFD900x transceiver is used which provides AES256 encryption over air and UART communication at 57600 bps between devices. To get relative position and orientation of the vehicle there are a pair of MPU6050 IMU sensors and SEN0259 LiDAR sensor. The IMU sensor measures the acceleration along the axes and orientation is calculated using the same with a filter in the middle using I2C protocol. LiDAR polices distance information from the surface at 100 Hz upto 12 meters using UART at 115200 bps. The absolute position of the vehicle is collected using a GPS module HERE3 which shares data using UART connection.

The flight system to control the orientation of the vehicle is managed by the ACS. It is a configuration of 4 + 2 cold gas solenoid valves which get actuated using a relay board. The gimbal is moved using a pair of linear actuators placed perpendicular to each other. These actuators need to be light weight and should have enough force to move the engine at full thrust. Each actuator has a 2” stoke length which provides a gimbal angle of 10 degree in each direction. To provide lift and translation JetCat P300 engine is used to provide a thrust upto 300N. The engine is industrial grade and provides communication and control over UART.

The software for the avionics control runs on a Raspberry Pi 4. To have a clean design, power distribution and connection a Pi shield has been added. It provides multiple 5V and 3.3V lines and removes the requirement of using a voltage splitter. There are 2 batteries on the vehicle, one to power the engine using 22.2V 6S LiPo and another 7.4V 2S LiPo to power the solenoid valves. There is a UBEC voltage regulator connected to the 7.4V battery to provide constant 5V to the Pi. The Pi internally has a 3.3V regulator for running the system and providing power to sensors connected to it. All the avionics including engine and batteries share a common ground to avoid a current backflow, which could damage the electronics.

The system wiring (ACS) is completely new, flexible, thinner, clean and provides slack in case of vehicle topple. The re-routing has been done in such a way that the wiring is all internal and connects all the components from under the housing, rather than the outside of the vehicle. The wiring now caters to the electronics individually and is therefore easily replaceable in case of damage. It’s been done in such way that the overstretching won’t cause any damage to

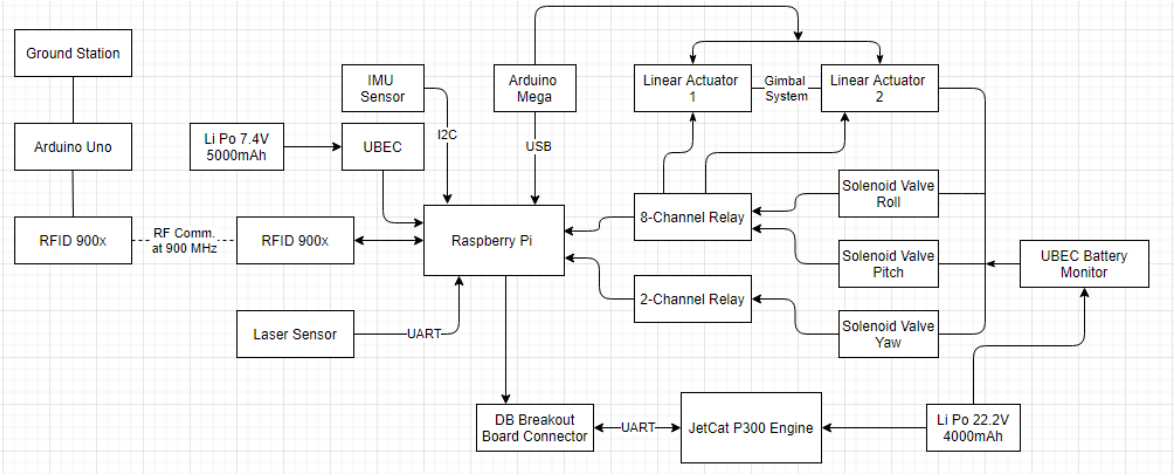

Figure 13: High-level connection diagram for on-board avionics

the avionics as well as the Attitude Control System during extreme flight measure. All the electronics are stationed together under one housing and provide rigid support during static, tether as well as flight testing. The housing protects the electronics from topple and extreme weather conditions.

3. Testing Campaigns

3.1 Control System Test Beds

Prior to the full integration of the vehicle, the two control systems, the TVC and the ACS, were validated on their own stands that operated separately from the flight vehicle. This allowed us to isolate problems that were encountered in both of those systems prior to their integration with the other subsystems. Additionally, it allowed us to capture important characteristics of the systems required for the development of the control algorithm, such as response times, maximum overshoot, time to steady state, and others.

The attitude control system controllers are tested via air bearing test stand. In this case, the vehicle, or its mass analogue, is placed on an air bearing stand with the IMU and the cold gas thrusters. Then, a disturbance is applied to the air bearing vehicle. Parity is checked to ensure that the correct thrusters are firing, and settling time is measured to ensure that the controller is behaving as expected. Yaw tests are performed to ensure the vehicle behaves properly with respect to yaw commands. After the air bearing test, the altitude controller and the attitude controller are tested in tandem on the vehicle via a tether test, explained in Section 3.4.

With this test successfully demonstrated, it is planned to move into flying the vehicle in earnest, with takeoff, translation, and landing sequences all demonstrated. This would demonstrate working performance of the thrust vector control, altitude control, roll & pitch control, and yaw control.

3.3 Static Hot Fires

As mentioned in the above section, LEAPFROG's propulsion system consists of a JetCat P300 Pro turbine engine that is gimbaled using a gimbal manufactured in-house at the SERC lab and two linear actuators. The goals of the static hot fires were oriented around validating, demonstrating, and quantifying various aspects of the propulsion system, and are shown in Table 3.

Table 3: Goals of static hot fires categorized by subsystem

Sub-System	Goal
Propulsion	Validate engine startup procedure
	Validate fuel feed system
	Characterize engine performance metrics
Control Systems	Quantify force output of actuators
	Validate gimbal actuation
	Demonstrate no interference between ACS and engine firing
Structures	Determine temperatures seen on vehicle due to plume
	Validate rigidity of gimbal
	Validate structure's ability to hold engine at full thrust
Software	Demonstrate communication between vehicle and ground station
	Demonstrate ability to immediately break from command sequence
	Demonstrate ability to throttle the engine
Avionics	Validate power consumption of engine
	Validate ground station equipment
	Quantify IMU bias

Characterizing the JetCat P300 Pro engine required metrics that were needed for the control algorithms to be quantified, such as the response time of the engine over a variety of command sequences and the maximum overshoot of the RPM. Figure 14 illustrates this data collected on one firing, demonstrating the correlation between the ability for the engine to proceed through its firing states and the exhaust gas temperature. Additionally, the tests provided engine metrics needed for the development of other systems, such as the fuel flow rates at various thrust levels and the power consumption rates at various thrust levels.

These static hot fires were conducted using a stand design specifically to hold the vehicle for these

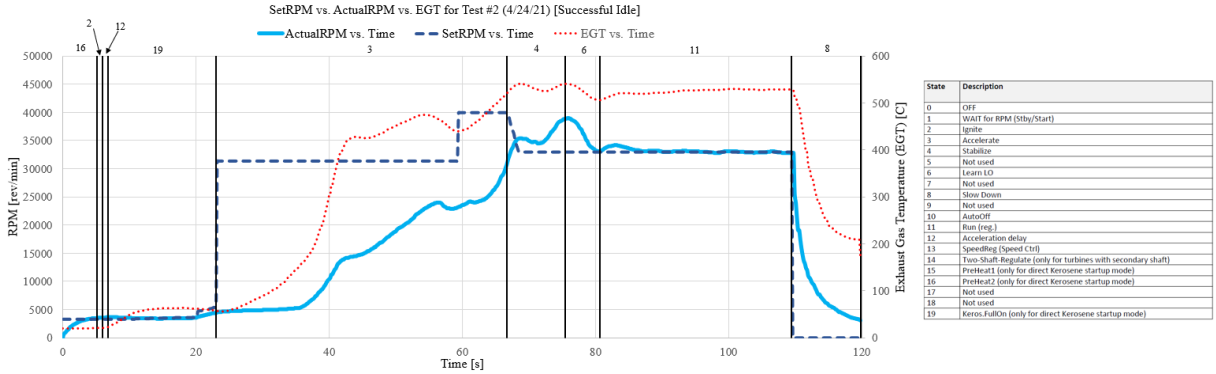

Figure 14: Data collected during static hot fire plotted over the various engine states

tests. The vehicle integrated into this stand with all of the associated hardware is seen in Figure 15.

Figure 15: Static hot fire stand prior to test

3.4 Tethered Tests

Rather than move directly from the static hot fires to a free flight test, the team instead decided to run the first flight tests with a tethered test stand to minimize the potential for irreparable damage. As shown in Figure 16, it uses a tether harness both above and below the vehicle to localize which subsystems of the vehicle are tested. The bottom tether allows for the restriction of translational motion, so no use of the TVC is required, and tests can focus exclusively on testing the ACS' control of rotational motion with all 6 degrees of freedom. The top tether acts as a safeguard against free fall should the engine abruptly cut out.

Figure 16: Integrated LEAPFROG supported by the tether test stand

Additionally, the tethered test makes it possible to conduct the first flight tests on-site at the SERC lab, meaning that more, shorter tests can be performed and allowing for more targeted testing of the different systems. If the only option had been to travel to a safe location for free flight, due to the time requirements of such a trip, it's likely that longer tests and fewer trips would have been prioritized. However, this would have been suboptimal, as the purpose of testing is to uncover issues and bugs in the vehicle, and a flexible testing campaign is necessary.

Before performing an actual flight test using the tethered stand, it was necessary to validate the stand itself by performing a "dry run" of the tethered flight test. During this test, the vehicle was hung from the engine hoist using the tether harness to validate that the test stand was capable of catching the vehicle, should it begin to fall. Additionally, this time was utilized in order to perform a test of the ACS system, which was successful in decreasing the amount of swinging and spinning the vehicle had been doing while hung.

Since the fuel is capable of keeping the engine running for roughly 6 minutes, it's clear that the vehicle flight time is limited by the pressurized ACS air tanks, and therefore a maximum hover of 1 minute was planned. The procedure for the tether test stand was simply to perform all of the engine checks completed before each static hot fire, and then to command an engine hover at increasing heights for increasing times with the ACS program running. The maximum hover height that the test stand was designed to support is 2 meters. By starting with a short hover for a short amount of time, the team can validate the landing procedures and limit the potential damage to the vehicle should the vehicle come down sub-optimally.

4. Distribution and Accessibility

4.1 NASA Artemis STEM Pilot Project

Through California's Space Grant, USC, UCSD, and UCB were supported by NASA's STEM Pilot program for this effort. The ARTEMIS LEAPFROG team was tasked to build and deliver multiple flight-capable lunar lander prototypes that could execute tasks in Earth's gravity and atmosphere.

Additionally, the team was tasked with organizing a national competition centered around this vehicle as part of NASA's goals to encourage hands-on training for undergraduate and graduate students that promote learning, teamwork, research, and enthusiasm surrounding the Artemis project.

The initial plan for this competition called for competitors to develop and demonstrate Artemis-relevant systems engineering skills by building a lander with materials provided by the LEAPFROG team and then carefully flying it through a physical obstacle course. However, due to the COVID-19 pandemic, this version of the competition was altered, and a software challenge only was introduced to accommodate the circumstances. Instead of gathering teams from all over the country to compete at one location, a full simulation was built and communicated to teams from across the United States to develop at their facilities.

Although the pandemic stymied progress on an in-person national competition during the summer of 2021, plans to hold this competition as originally

envisioned during the summer of 2022 are in deliberation.

4.2 2021 Software Challenge

The LEAPFROG Software Simulation Competition launched in the summer of 2021. The competition was open to all university-affiliated students throughout the United States. Competition registration required teams to have one faculty member as a contact point.

The first webinar was hosted on April 7th, 2021. We introduced an overview of the LEAPFROG Competition as a whole, and the software required to setup your simulation environment. This included the basics of PX4, ROS, Gazebo, and MavRos.

The second webinar was hosted on April 30th, 2021. The teams learned how to add new plugins to the PX4 Software specific for the LEAPFROG vehicle and simulation, and how to link the various software elements together. We also provided more detailed instructions on setting up the simulation environment, and how it operates.

The third webinar was hosted on May 12th, 2021. We demonstrated how the teams might modify the behavior of the LEAPFROG vehicle within the simulation, and the teams also learned about our competition scoring rubric.

The fourth webinar was hosted on May 21st, 2021. We broke down the scoring criteria in great detail, showed an initial "lunar world" with craters for the team's use, and described how teams upload their code/algorithms to the Software Challenge Github. We also described how winners will get kits upon the final scoring and notification.

We registered teams from New Mexico State University, New Mexico Tech, University of Texas at Austin, and University of Illinois at Urbana-Champaign. We also had an internal team from UC Berkeley to test various competition aspects.

At the end of the summer, we congratulated the University of Illinois at Urbana-Champaign for winning the inaugural LEAPFROG software competition! Their team successfully navigated the LEAPFROG vehicle and landed safely in a crater in our competition simulation environment.

5. Conclusion

The LEAPFROG Artemis Challenge to be held in the Summer of 2022 will be an integrated demonstration of the LEAPFROG vehicle. We will have three flight vehicles in three regions of the country in which universities across the nation will have the opportunity to fly their unique navigation based code on LEAPFROG. Through the safety architecture developed, the team is confident that the vehicle will be able to detect and recover from off nominal flight conditions, allowing university teams to repeatedly test their innovative flight and landing algorithms on an Earth based lunar lander testbed.

Acknowledgements

The authors thank the California Space Grant Authority from the University of California San Diego, Dr. John Kosmatka from UCSD, Mr. Dan Zevin from UC Berkeley, the facility personnel at the Information Sciences Institute, and the past alumni of the LEAPFROG program from 2006 on!