Design, Simulation, and Testing of an ABS/GOX Hybrid Rocket Engine

Beni Snow, '19

Submitted to the
Department of Mechanical and Aerospace Engineering
Princeton University
in partial fulfillment of the requirements of
Undergraduate Independent Work.

Final Report

May 1, 2019

Michael Littman
Yiguang Ju
MAE 442
50 pages
File Copy

This thesis represents my own work in accordance with University regulations.

Abstract

Hybrid rocket engines are safer, cheaper, and less complex than liquid rocket engines, while providing many of their benefits, such as variable thrust and longer burn times. Solid motors are simpler than hybrids, but do not offer for the controllability of a hybrid. For these reasons, hybrid rocket engines are an ideal platform for investigating rocket thrust. This research project encompassed the design, simulation, and testing of such a rocket engine.

The engine used 3D printed ABS (acrylonitrile butadiene styrene) plastic as the solid fuel, and gaseous oxygen (GOX) as the oxidizer. ABS plastic is readily available and has the material properties that allows for the printing of complex fuel geometries without the hassle of curing resin based fuel grains. GOX is cheap, non-toxic, and much safer than liquid oxygen. The engine was built into a mobile test stand that can be used by future students to further explore rocket propulsion.

In addition to the physical engine, a MATLAB simulation of the rocket was developed and made publicly available, to allow for iterating on engine designs much more efficiently than actually building new components for each iteration.

The engine was successfully tested seven times and produced up to 650 Ns of total impulse. The simulation produced thrust curves that matched measured thrust within 17%. This represented a successful capability demonstration of what will undoubtedly be an ideal testing platform for students in the years to come.

Acknowledgements

Thank you to every single person in the MAE department who helped me with this project. It may have been an individual thesis, but I could not have done it without an enormous amount of help.

First, I have to thank Professor Littman for graciously agreeing to be my adviser for this project and to all the other professors I've had over these past four years.

To the School of Engineering and Applied Sciences, the MAE department in general, and the Morgan W. McKinzie '93 thesis fund in particular, thank you for providing the financial backing that made this project possible.

I would like to thank Al Gaillard and Glenn Northey in the machine shop for so much help not just on this project, but on many others over the past few years. To Al, thank you for helping me with a ton of manufacturing questions, and for teaching me how to weld. To Glenn, thanks for letting me set up Ansys on shop computer and for not being too mad when I tied up the printer for 80 hours. And to Linda Park in purchasing for not asking too many questions about what strange things I needed to buy.

Also, I would like to thank Jon Prevost and Mike Vocaturo for help with the control electronics, and of course to Jo Ann Love for watching out for the entire department.

To all my friends who humored me when I couldn't shut up about this project, and who supported me when I just couldn't stand it any more, thank you. A special shoutout to Coleman Merchant for helping with a lot of rocket design questions and to Nate Cope for all his help with editing, and to Coleman, Julian Castellon, and Robert Liu for helping with testing.

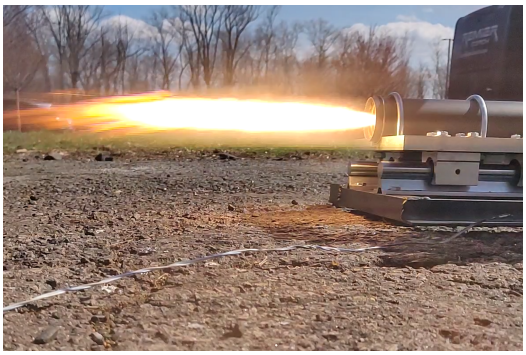
To the climbing team, thanks for being the most amazing group of people on this campus. I want to call out specific people for have been amazing friends over these past years, but I'm trying to keep this acknowledgements short, so I'll just assume that you know who you are.

To the Terrans who have provided a home away from home, ya'll are rocking it 24/7. Never change. Food = Love. Although I really should be giving some of you anti-acknowledgements, since you seem determined to distract me from actually completing this thesis.

Last but certainly not least, thank you to my parents for making this whole journey to Princeton possible. I don't know if you even remember this, but 10 years ago for a middle school science project you let me and a friend light off some solid rocket motors in the driveway. The test stand was a piece of wood held down by

some cinderblocks, and the only data measurement was a tape measure taped to the ground to measure the length of the exhaust plume, but you let us mess around, and letting me have the space to follow what I was passionate about was the first step towards me becoming an engineer.

On the left, we have a 2009 picture of an A8-3 solid motor with all of 2.3 Ns of total impulse, and on the right we have my engine, which is just about 300 times more powerful. Just imagine what a difference the next ten years will make.



Contents

List of Figures

List of Symbols

O/F	Oxidizer to fuel ratio	4
$\dot{m}$	Mass flow rate	6
C_d	Discharge coefficient	6
P_0	Pressure before choked flow contraction	6
A_t	Area of choked flow contraction	6
R_{0_2}	Individual gas constant of oxygen	6
T_0	Temperature before choked flow contraction	6
γ	Heat capacity ratio	6
λ	Nozzle correction factor	9
α	Nozzle half angle	9
T	Thrust	15
V_e	Exhaust velocity	15
A_e	Nozzle exit area	15
P_{atm}	Atmospheric pressure	15
P_e	Nozzle exit pressure	15
I_{sp}	Specific impulse	15
g_0	Standard gravitational acceleration	15
$\dot{m}_f$	Fuel mass flow rate	16
$\dot{r}$	Fuel regression rate	16
D_p	Fuel port diameter	16
L	Fuel length	16
ρ_f	Fuel density	16
a	Fuel regression constant	16
n	Fuel regression exponent	16
G_{ox}	Oxidizer flux	16
P_c	Chamber pressure	17
C^*	Characteristic velocity	17

Chapter 1

Introduction

Hybrid rocket engines burn solid fuel in the presence of a gaseous or liquid oxidizer to produce thrust. This is in contrast to liquid rocket engines, where both the fuel and oxidier are liquids, and solid motors, where both the fuel and oxidizer are solid. Solid motors, while relatively simple, cannot be throttled and suffer from low efficiency, as well as a limited range of propellants. Liquid engines, while more efficient and versatile, are incredibly complex and can involve dangerous and explosive fuel and oxidizer combinations. Modern hybrid rocket fuels are stable and non-explosive. They have been rigorously studied and can be safely stored and tested at much lower costs than a traditional liquid engine [8]. This makes hybrid rockets an ideal testbed for rocket propulsion research.

Currently, Princeton University has no small scale hybrid rocket engine test facilities suitable for undergraduate study. This project aimed to change that, by creating a small scale rocket engine that can be safely and usefully tested. To achieve this goal, the rocket was scaled down to a economical size, where a full test fire could be achieved using only 300 grams of fuel. The fuel/oxidizer combination was chosen to be ABS plastic, which can be 3D printed to skip the complicated process of curing fuel resins, and gaseous oxygen, which is cheap, readily available, and removes much of the risk and complexity of using a cryogenic oxidizer such as liquid oxygen [7].

In addition to building and testing the rocket engine, a MATLAB based simulation was developed. This simulation accurately models the thrust curve as a function of oxidizer flow, nozzle shape, and fuel grain geometry, determines the fuel consumed as well as several other useful parameters, such as chamber pressure, exhaust pressure, temperature, specific impulse, and the characteristic velocity. The simulation model

is available under the GNU public license on GitHub.[1]

This project drew on engineering skills from traditional manufacting, such as milling components for the test stand, as well as modern rapid prototyping, such as 3D printing the solid fuel. There was a non-trivial amount of electrical engineering as well to create the circuit board and power delivery system for the control electronics, as well as the ignition system. The collection and analysis of test data and the process of simulator verification necessitated rigorous data examination. The design of the injector relied on computational fluid dynamics, and nozzle designs were also iterated using CFD.

Overall, the project was ambitious, but achievable at a reasonable cost, with a high safety margin, and the result is a fully functioning engine that can now be used for further research into hybrid propulsion technologies.

[1] https://github.com/benisnow456/Hybrid-Rocket-Engine

Chapter 2

System Design

The engine and associated support infrastructure breaks down into four main subsystems:

1. Oxidizer feed system
2. Combustion chamber
3. Control electronics
4. Test stand

2.1 Oxidizer feed system

A hybrid rocket engine, by definition, uses a liquid or gaseous oxidizer. Gaseous oxygen (GOX) was selected as the oxidizer. GOX was selected due to its low cost, easy availability, and extremely well studied characteristics as a oxidizer.

The oxygen was supplied to the combustion chamber using a pressure feed. This system is much simpler than using pumps, which are the other common method of moving the oxidizer, but does have several drawbacks.

A key drawback is that the oxidizer tank cannot be fully used – it can only provide oxidizer when the internal pressure of the tank is greater than the pressure in the combustion chamber. However, as the system was only designed for ground testing, using up every gram of oxidizer was not a priority. Another drawback is that the oxidizer mass flow rate is not constant. As the tank pressure drops, the mass flow rate declines as well. This results in a shifting oxidizer to fuel (O/F) ratio

throughout the burn, which had to be accounted for in the calculations. However, these drawbacks are justified by the simplicity of such pressure-fed system.

Figure 2.1: Oxidizer feed system components

2.1.1 Tank

The oxygen tank used was a standard E size medical oxygen tank at 2,200 PSI, containing 24 ft^3 of gaseous oxygen, or 971 grams. These tanks are commonly available, have high purity, and contain more than enough oxygen for each test fire. Oxygen was sourced through Airgas, the university gas supplier. The tank was connected to a yoke that converted the CGA 870 fitting to a NPT 1/4" fitting to allow connection to the pressure regulator.

2.1.2 Pressure regulator

The pressure regulator used was a Swagelok general purpose pressure regulator, with an input pressure of up to 3,600 PSI and a variable output pressure of 0 - 500 PSI. The system was run at 500 PSI.

2.1.3 Pressure transducer

A pressure transducer was installed downstream of the pressure regulator. The sensor, made by TE Connectivity, can measure gauge pressures between 0 and 500 PSI. This sensor was used to measure the dynamic pressure of the oxygen after the

pressure regulator and before the contraction in the solenoid valve. After acquisition, the data from the load cell was smoothed using MATLAB's robust version of the LOWESS locally weighted scatterplot smoothing local regression function with a span of 10.

2.1.4 Valve

An ASCO Red-Hat solenoid valve was installed as the main control valve for the oxidizer delivery system. When a current is applied to a solenoid valve, the valve will actuate very rapidly. A ball valve, which is another commonly used option for similar purposes, would require a valve controller as well as having a much slower cycle time. The ASCO valve chosen is a normally closed valve, so that in the event of a power loss the engine would shut down. The valve runs on standard 120 V 60 Hz AC current, which was controlled by a 12 volt automotive relay.

In addition to the solenoid valve, a manual ball valve was installed in the oxidizer delivery system and was manually opened before each test, as an added safety feature.

2.1.5 Flow meter

Although not a permanent part of the oxidizer delivery system, a flow meter was used to calibrate to system. A Omega rotameter type flowmeter was used. Such a flow meter measures the standard cubic feet per minute, or SCFM, of air. A correction factor of 1.05 was used to a convert to SCFM of oxygen, and then a constant oxygen density at STP of 1331 g/m^3 was assumed to convert to grams per second [4] [1].

Initial calibrations were conducted using nitrogen, due to its low cost and chemical inertness, but for final calibration an oxygen tank was connected and then vented through the system for 30 seconds. The data obtained was fitted to a second order polynomial, which allowed for the mass flow rate at any point to be determined. The second order polynomial generated a very accurate fit with an R^2 value of 0.9971.

Although each tank should produce similar flow, a better method of determining oxidizer mass flow was desired. Methods such as a Venturi meter are well established to calculate mass flow of a compressible fluid, but such meters are costly, as they require exact machining and multiple precision sensors since the pressure differential created is small. A system that could rely on a single, less expensive sensor was desired.

The principle of choked flow was used to calculate the mass flow through the valve,

and, since mass is conserved, into the combustion chamber. When a compressible fluid flows through a constriction, the velocity will increase, until it reaches Mach 1. At this point, the flow becomes choked and the mass flow rate will vary linearly with pressure according to the equation [2]

$$\dot{m} = C_d P_0 A_t \sqrt{\frac{\gamma}{R_{0_2} T_0} \left(\frac{2}{\gamma+1}\right)^{\frac{\gamma+1}{\gamma-1}}} \tag{2.1.1}$$

where P_0 is the pressure before the choking point in the valve, as measured by the pressure transducer, R_{0_2} is the individual gas constant, which for oxygen is 259.84 J/kg K, T_0 is the temperature, assumed to be a constant 20 °C, and γ is the heat capacity ratio, which for oxygen is 1.40 [1]. A_t is the area of the valve orifice, which was calculated from the port diameter of the ASCO valve of 0.125". C_d is a discharge coefficient, which measures the ratio between the theoretical and actual discharge, and is normally a little less than 1. The best fit occurred when C_d was 0.77, as shown in figure 2.2.

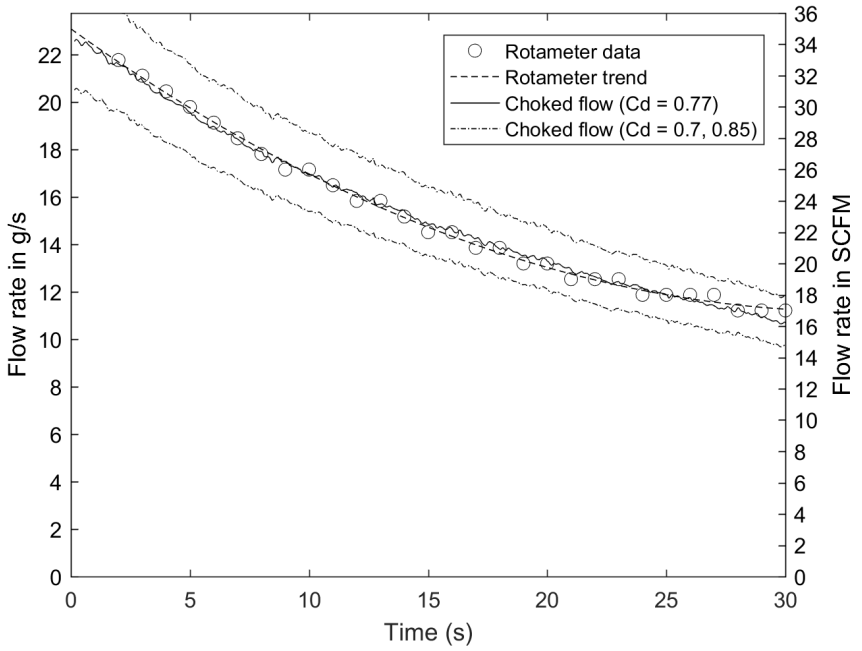


Figure 2.2: Oxidizer mass flow rate as a function of time

2.2 Combustion chamber

After the oxidizer flows through high pressure tubing, it enters the combustion chamber through an injector, flows over the igniter, and then over the fuel grain, and out the nozzle, generating thrust. The combustion chamber and nozzle were purchased through Loki Research and were originally intended for use with solid rocket motors, as shown in Figure 2.3, courtesy of the manufacturer.

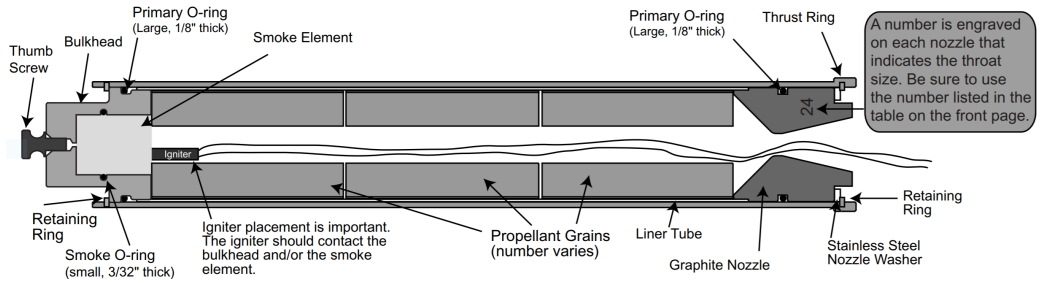


Figure 2.3: Combustion chamber and nozzle configuration

Some modification was necessary to allow for use in a hybrid engine, such as installing a port for the high pressure oxygen tubing where the thumb screw was originally installed, as well as modifying the nozzle since solid rockets typically have fuel regression rates three times or more higher than hybrids [10].

2.2.1 Injector

The orifice that allows the oxidizer to enter the precombustion chamber is critical to maintaining stable combustion. In a well-designed injector, there is a precombustion chamber that generates a recirculation zone, as shown in part (a) of figure 2.4, which is a diagram taken from reference [10]. In a poorly designed injector, there is no recirculation, as shown is part (b).

The precombustion chamber used for the engine was 1.25" long with a 1" diameter. The precombustion chamber was modeling using the CFD software Ansys FLUENT for oxygen flow at 22 g/s, which is the oxidizer mass flow at engine startup. Figure 2.5 shows the strong recirculation zones in the precombustion chamber.

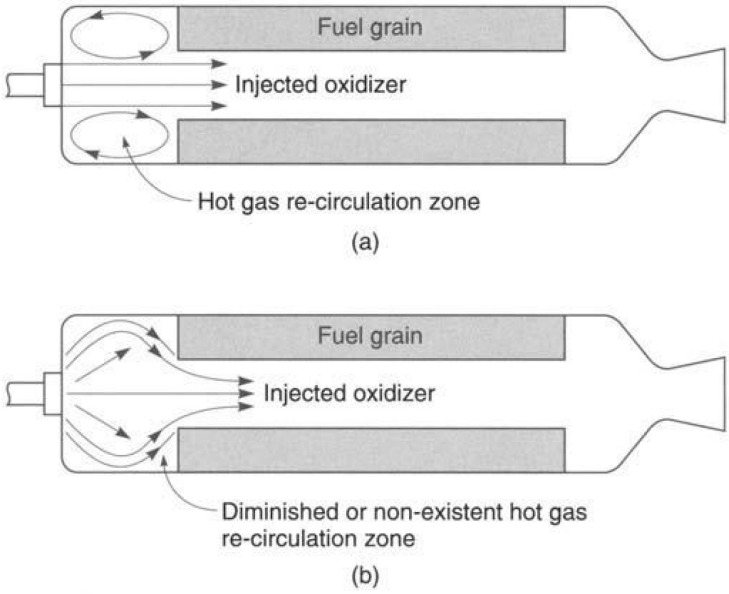


Figure 2.4: Ideal vs nonideal injector design

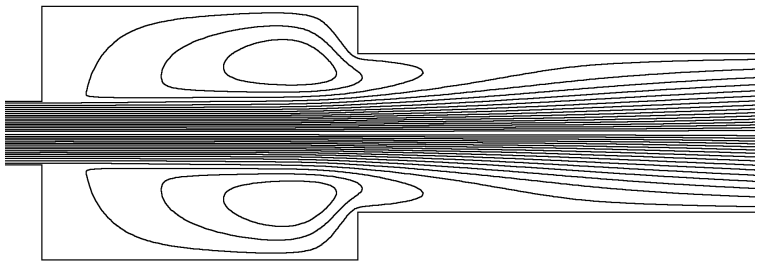


Figure 2.5: CFD model of the injector, showing the recirculation pattern.

2.2.2 Igniter

Immediately after flowing through the injector, the oxidizer flows into the pre-combustion chamber. This space was where the igniter was placed. The igniters used consisted of an e-match igniting a mixture of black powder and steel wool.

An e-match is a type of initiator that consists of wire leads that connect to a resistive element coated in a pyrotechnic mixture, in this case pyrogen. The e-matches used were Firewire initiators from MJG Technologies. When a current is applied, the resistive element heats up and ignites the pyrogen. The e-match was placed in a small plastic sleeve with half a gram of black powder and half a gram of steel wool. The burning pyrogen ignited the black powder which then burned the steel wool, generating sparks. Initially, an electronic system was considered using high voltage

sparks from an automotive coil, but the pyrotechnic ignition was deemed to be a more robust system.

2.2.3 Fuel

The fuel for the engine was FDM 3D printed ABS plastic using a Stratsys Dimension Elite 3D printer, printing their proprietary ABS-P430 filament at 100% infill. ABS is readily available for 3D printers, and using a 3D printed fuel removed the need for the complicated curing processes that some cast fuels require. Additionally, 3D printing allows for much more complex fuel grain geometries. A simple circular through hole was the only geometry tested, but novel fuel geometries are a promising area for future research.

Three segments were placed in the combustion chamber, as shown in Figure 2.3. Each segment has a outer diameter of 1.938", an inner diameter of 0.625", and a length of 2.375" for a total fuel volume of 308.6 cm^3. The fuel was weighed at 102.7 grams per segment, for a total initial fuel mass of 308.1 grams and a fuel density of 0.998 g/cm^3.

2.2.4 Nozzle

The nozzle was one of the most complicated parts of the design, and three different nozzles were tested. All nozzles were procured from Loki Research, and then two of them were heavily modified. All the nozzles feature a 30 degree angle exit cone, and came with a 0.85" exit diameter.

A simple cone nozzle, while easy to machine and modify, is not the ideal shape. A cone nozzle has a correction factor λ that relates the efficiency of an ideal nozzle to one with a half angle of α, which for all the tested nozzles was 15 degrees [10].

$$\lambda = \frac{1}{2}(1 + \cos\alpha) = 0.983 \tag{2.2.1}$$

The first nozzle, used in tests 1 and 2, was an unmodified #24 nozzle, which means that the throat diameter was 24/64" (9.525 mm). This nozzle has an expansion ratio of 5.14.

The second nozzle, used in tests 3 and 4, was a modified #19 nozzle. The throat area was decreased to the minimum that Loki Research provided since chamber pressure was calculated to be fairly low. The shoulder of the nozzle was shaved down to

allow it to better fit into the motor casing, and the overall length was also shortened so that the exit diameter was 0.7". This resulted in an expansion ratio of 5.56.

The third nozzle, used in tests 5 to 8, was also a modified #19 nozzle. After previous tests revealed significant overexpansion, even with a shortened bell, the bell was further shortened to a 0.5" exit diameter, which resulted in an expansion ratio of 2.84, although it did create a step nozzle.

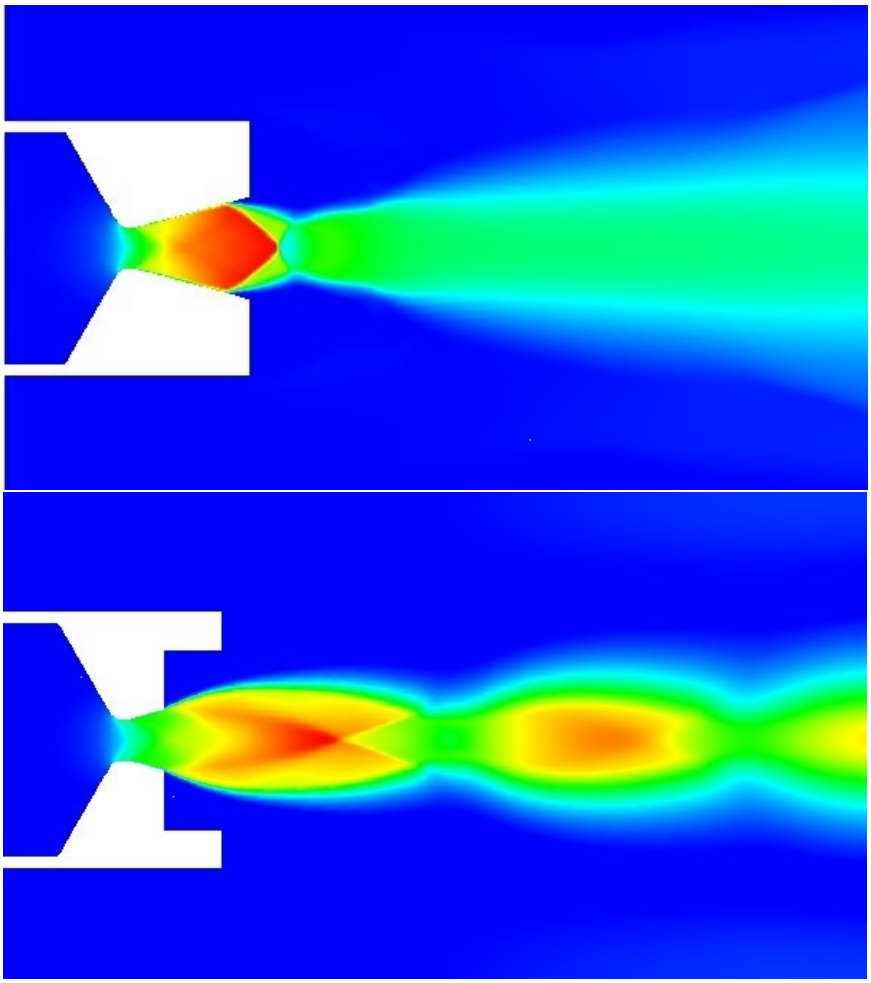


Figure 2.6: CFD model of two of the nozzle designs

The higher expansion ratios, while appropriate for the solid motors for which they were designed, resulted in flow separation for the lower mass flow rates in the hybrid engine. This was noted experimentally, and then CFD analysis backed up this observation, as shown in the top image in figure 2.6. This was the reason for the dramatically lower expansion ratio for the third nozzle, shown in the lower image in figure 2.6.

Ansys FLUENT was used to simulate the first and third nozzle design, shown

in figure 2.6. The first nozzle has obvious flow separation, while the third nozzle produces a much cleaner exhaust plume and shows the Mach diamonds that were occasionally observed in the actual exhaust plume. See figure 3.3.

2.3 Electronics Board

The microcontroller at the heart of the engine was an Arduino Uno. The Arduino, power systems, and all other electronics for the engine were all mounted on a 22" by 16" panel, which was then mounted on the same welded steel frame that held the pressure regulator, valves, oxygen tank, and combustion chamber. The electronics board is shown in figure 2.7, with the power supplies are marked in red, the Arduino in blue, the daughter board in green, and the relays in white.

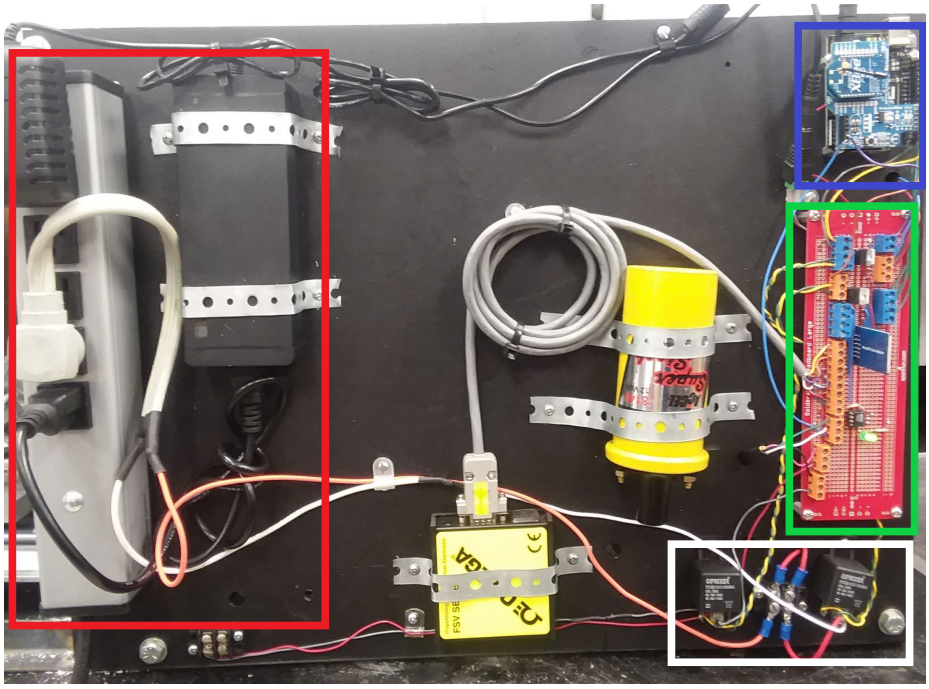


Figure 2.7: The control electronics board with major subsystems highlighted.

2.3.1 Power

Power was provided by means of the power strip mounted on the left side of the panel. The Arduino was powered by a 12 volt, 2 amp charger. The daughter board, since it provides power to the ignition system, was powered by a 12 volt, 10 amp AC/DC power supply. The solenoid valve received its power directly from an AC

outlet on the power strip, although the circuit was interrupted by an automotive relay to allow for the Arduino to control the valve. All other sensors and circuits were powered from the Arduino's 5 volt output.

2.3.2 Arduino

The entire engine was controlled via an Arduino Uno that read in commands from a computer, actuated the relays to control the oxidizer valve and the igniter, and read in data from the pressure transducer and load cell. The full Arduino code may be found in Appendix A.

2.3.3 Xbee

Communication between the Arduino and the control PC was accomplished using an Xbee Pro radio module. One Xbee module was connected to the computer using a USB dongle, and another was connected to the Arduino using a shield. This allowed for the engine to be started and stopped remotely, as well as stream data to the computer.

2.3.4 SD card

In addition to streaming the data over the Xbee serial connection, data was logged onboard by means of a SD card reader mounted to the daughter board.

2.3.5 Relays

Two 12 volt automotive relays were used for this project. One controlled the igniter, and the other controlled the oxygen solenoid valve. Each relay was set up as normally off, and activated using a Arduino pin to turn on a TIP120 Darlington power transistor, which allowed 12 V to flow to the relay.

2.3.6 Unused electronics

In addition to all the above components, the electronics board also contained a automotive ignition coil and a Omega valve controller. The ignition coil was originally intended for use in the electronic ignition system, but after switching to a pyrotechnic system the coil was no longer used.

The Omega valve controlled was to be used with a proportioning valve to allow throttling of the engine, but the valve never functioned properly and had to be excluded from the final design.

2.4 Test stand

The oxidizer delivery system, electronics board, and combustion chamber were all mounted to a steel test stand 36" long, 18" high, and 12" wide. The combustion chamber was mounted horizontally to a aluminum plate using steel U brackets, and the plate in turn was mounted to linear slider rails, allowing the chamber to slide freely. This entire assembly was connected to a load cell so that the entire horizontal thrust of the rocket engine would be placed on the load cell.

2.4.1 Thrust sensor

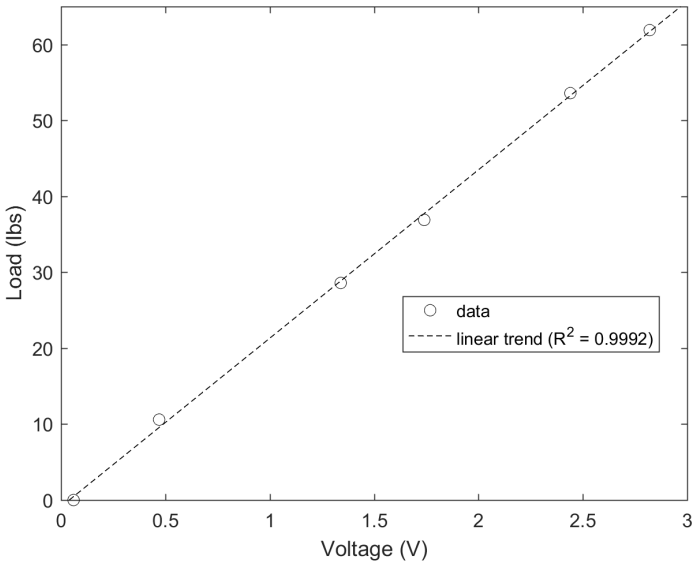


Figure 2.8: The load cell calibration data and linear trend.

A 50 kg rated S-type load cell was positioned in compression between the combustion chamber plate and the test stand to measure the thrust. The cell was excited with a 12 V input voltage and the output voltage of 2.0 mV/V was amplifed with a INA129P op-amp, and a linear regression run on the data.

The R^2 value of the regression was 0.9992, showing excellent linearity between the input load and the output voltage, as expected. See figure 2.8 After acquisition,

the data from the load cell was smoothed using MATLAB's robust version of the LOWESS locally weighted scatterplot smoothing local regression function with a span of a quarter of a second.

Chapter 3

Simulation and Results

The two main parameters that should be measured to characterize any rocket engine are its thrust and its efficiency.

Thrust is calculated by

$$T = \dot{m}V_e - A_e(P_{atm} - P_e) \tag{3.0.1}$$

where $\dot{m}$ is the total mass flow rate, V_e is the exhaust velocity, A_e is the nozzle exit area, P_{atm} is the atmospheric pressure, and P_e is the nozzle exit pressure.

Specific impulse is given by

$$I_{sp} = \frac{T}{\dot{m}g_0} \tag{3.0.2}$$

where g_0 is the standard gravitational acceleration of 9.8 m/s^2.

In order to determine a theoretical expected thrust, the exhaust velocity mass flow rate and exit pressure must be determined. These parameters are not constant over the course of a burn, and depend on other factors, such as chamber pressure, O/F ratio, oxidizer flow rate, and the physical parameters of the fuel. For this reason, a MATLAB simulation was built to accurately model the rocket engine.

3.1 Simulation

The simulation was built around NASA's CEA (chemical equilibrium with applications) software. CEA can output exhaust velocity and exit pressure, as well as chamber pressure, exhaust pressure, temperature, specific impulse, and the charac-

teristic velocity. For the type of problem being solved for rocket simulation purposes, CEA requires information on the fuel, oxidizer, O/F ratio, chamber pressure, and nozzle expansion ratio [6].

3.1.1 Fuel and oxidizer

The oxidizer input to CEA was simple – gaseous oxygen at 285 °K. The fuel input was more involved, since CEA's database did not contain ABS plastic. However, all that was required was to determine to molar mass, chemical formula, and specific enthalpy of formation, and then to add that information to the database.

ABS has a varying composition, but the PubChem value of 211.308 g/mol and chemical composition of $C_{15}H_{17}N$ was chosen as representative [3]. The enthalpy has been calculated to be -1097.4 kJ/kg, which results in a value of -231,900 J/mol, which is the format the CEA requires [9] [6].

3.1.2 Fuel regression and mass flow

The O/F ratio requires both the oxidizer mass flow rate and the fuel mass flow rate to be known. The oxidizer mass flow rate is determined by the choking flow, as shown is figure 2.2. The mass flow rate of fuel is given by [10]

$$\dot{m}_f = \dot{r}\pi D_p L \rho_f \tag{3.1.1}$$

where D_p is the diameter of the fuel port, which was 0.625" at the beginning of a burn with a new fuel grain, L is the total length of the fuel grains, and ρ_f is the density of the fuel. $\dot{r}$ is the fuel regression rate, i.e., the rate at which the fuel burns outwards and the fuel port expands. This is given by the equation

$$\dot{r} = aG_{ox}^{n} \tag{3.1.2}$$

where a and n are experimentally determined constants, and G_{ox} is the oxidizer flux, meaning the mass flow rate divided by the current fuel port area. Experimental data showed that the best fit occurred at n = 0.31, and a = 0.00016. With these coefficients, the simulated fuel burn and change in port diameter best fits the measured data.

3.1.3 Pressure and characteristic velocity

The chamber pressure is given by the equation [10]

$$P_c = \frac{C^* \dot{m}}{A_t}$$

where C^* is the characteristic velocity of the engine, $\dot{m}$ is the mass flow rate, and A_t is the throat area of the nozzle.

Characteristic velocity for a fuel combination is highly dependant on the O/F ratio, and slightly dependant on the pressure.

O/F ratios dropped from roughly 3.5 to 2 over the course of a typical burn, which as shown in figure 3.1 means the characteristic velocity increased throughout the burn. This was one reason why even though the total mass flow drops throughout the burn, the thrust stayed relatively constant.

The combustion chamber pressure reached almost 11 atm during the beginning of tests with a smaller throat area and dropped to as low as 5 atm at the end of a test using a larger throat area. However, as shown in figure 3.1, chamber pressure does not result in much of a change in characteristic velocity. This allowed for an initial value to be guessed, and then as CEA is rerun for reach time step and the pressure calculated, that guess is updated.

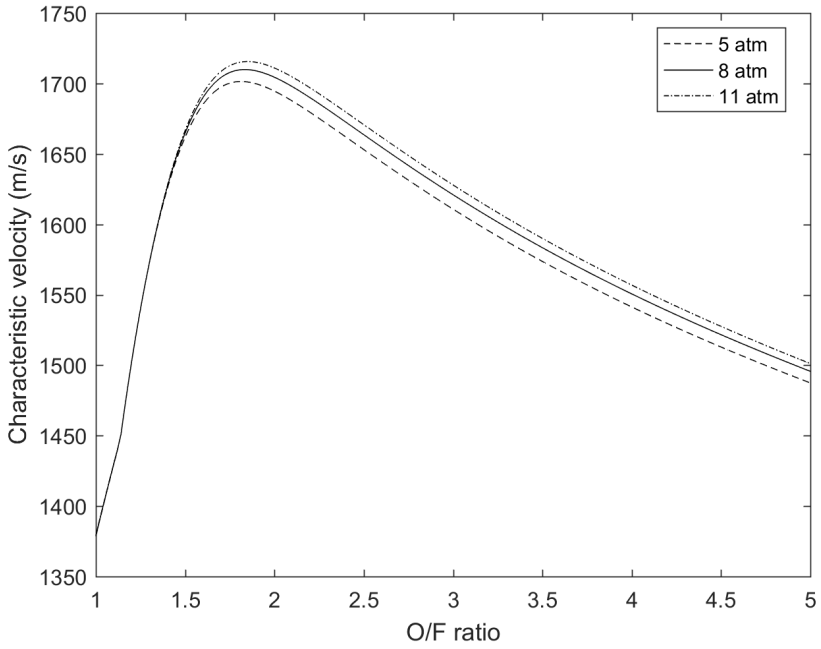


Figure 3.1: Effect on O/F ratio and chamber pressure on characteristic velocity.

Other simulation parameters

The simulation also takes in parameters such as the nozzle expansion ratio, which for the three nozzles tested was 5.14, 5.56, and 2.84. The simulation also requires the initial fuel port size, which for most tests was 0.625", although for two tests the fuel grains were reused, so the initial port size was larger.

CEA outputs the exhaust velocity and exit pressure, which are needed for the thrust calculation, as well as other useful parameters, such as combustion temperature. The thrust was then calculated using equation 3.0.1.

The assumptions behind CEA lead to a higher thrust than in a real engine. One assumption is that the nozzle is ideal, which a conical nozzle is not. As such, the exhaust velocity was multiplied by the loss coefficient λ, determined in equation 2.2.1.

Additional loses come from factors such as combustion instabilities, nozzle ablation, and friction. CEA makes several assumptions that ignore these effects, such as assuming 1D isentropic flow, no thermal or velocity lags, and no velocity at the inlet [5]. These losses are hard to characterize exactly, but thrust was scaled down to 70% in an attempt to estimate them. As is shown in the results section, this loss assumption produced a reasonably accurate model.

Simulation output

Figure 3.2 shows several of simulation outputs for a case of a 10 second burn using the nozzle with a #19 throat and a 0.5" exit diameter. These parameters correspond to test 5, the first test with the final nozzle design.

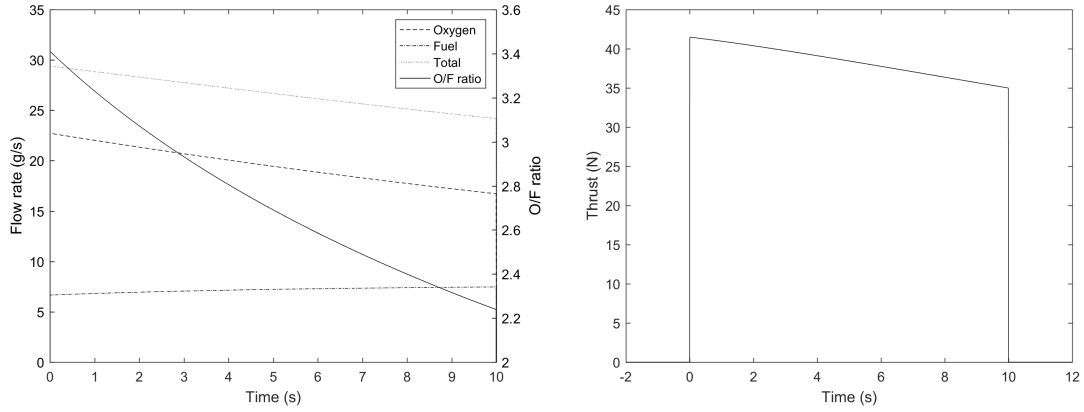


Figure 3.2: Simulation mass flow, O/F ratio, and thrust

3.2 Results

A total of eight of burns were conducted on the engine over three days of testing, with times ranging from 5 to 18 seconds. Of these, seven were considered to be total or partial successes, and one, test 4, was considered to be a failure. An overview of the tests may be found in table 3.1.

Test	Time	Mean thrust	Mean thrust (sim)	Percent difference	Fuel burned	Fuel burned (sim)	Percent difference
1	10 s	23.8 N	21.1 N	12.0 %	67.7 g	65.1 g	3.9 %
2	10 s	N/A[i]	26.8 N	—	75.2 g	77.3 g	2.8 %
3	15 s	33.7 N	35.4 N	4.9 %	105.6 g	106.4 g	0.8 %
5	10 s	36.5 N	40.7 N	10.9 %	66.9 g	69.3 g	3.5 %
6	5 s	36.1 N	42.0 N	15.1 %	32.5 g	33.6 g	3.3 %
7	15 s	43.5 N	36.7 N[ii]	17.0 %	110.9 g	109.7 g	1.1 %
8	18 s	31.8 N	35.7 N	11.6 %	128.4 g	123.3 g	4.1 %

Table 3.1: Summary of test data

In addition to the data captured from the sensors, high speed video was also taken. Stills from that video is shown in figure 3.3. As predicted in figure 2.6, the exhaust plume produced Mach diamonds, however, most of the time the plume was bright enough that the camera sensor oversaturated, so the diamonds were only visible during transient periods.

Test 1

The first test of the engine was primarily a systems check. The first nozzle design, with a #24 nozzle was used. A 10 second test produced 238 Ns of total impulse with a maximum I_{sp} of 115 seconds, for an average thrust of 23.8 N.

Test 2

Test two reused the fuel grain of test 1, meaning the starting port was not the normal 0.625", and was instead 1.06". However, due to a data logging issue, all the thrust data for this test was lost. Subsequent tests streamed data as well as logging it locally, to prevent a repeat of this error.

[i] Data on thrust for this test was lost.

[ii] This test was throttled, and the simulation was not designed to accurately model such a test.

Figure 3.3: Images of the engine being tested.

Test 3

Test three used the second nozzle design, and operated at up to 10.3 atm of pressure, compared to 6.5 atm of pressure for earlier tests. Additionally, the test ran for an extra 5 seconds.

The test produced 505 Ns of total impulse over 15 seconds, for an average thrust of 33.7 N, and had a maximum I_{sp} of 152 seconds, a significant improvement over earlier tests.

Test 4

Test 4 the only failure of this project. Due to a improperly connected snap ring used to retain the precombustion chamber, the engine catastrophically failed approximately one second into testing, resulting the destruction of the precombustion chamber and the combustor. The engine was rebuilt after this failure and testing was resumed, although much more care was taken to ensure that all components were properly set up before ignition.

Test 5

Tests five and onward used the third and final nozzle design. The test produced 365 Ns of total impulse over 10 seconds, for an average thrust of 36.5 N and a maximum I_{sp} of 147 seconds.

Test 6

Test six produced 181 Ns of total impulse over 5 seconds for 36.1 N of thrust and a maximum I_{sp} of 143 seconds.

Test 7

Test seven was different from all previous tests and throttling of the engine was attempted. This was accomplished by cycling the solenoid valve on and off as in a PWM cycle, with a cycle time of 80 ms. As the data in figure 3.4 shows, this was not very effective at producing a throttled effect, and just produced thrust spikes. Hybrid engines can be throttled, but without a variable flow valve, this engine can not. Because of the pressure spikes, the actual mass flow rates for this test could not be input to the simulation, so simulated values are based on the reference polynomial fit oxidizer flow rate determined in section 2.1.5.

The test produced 652 Ns of impulse over 15 seconds, for an average thrust of 43.5 N.

Test 8

Test eight was the longest test conducted, running for 18 seconds. However, as in test 2, data was not logged properly. After test 2 the code was modified to stream

the data, so not everything was lost. However, data was logged locally every 25 mS and only streamed every 500 mS, so the data could not be smoothed as well.

The burn generated 573 N of total impulse of 18 seconds, for an average thrust of 31.8 N, although the low sampling rate may effect the error on this measurement.

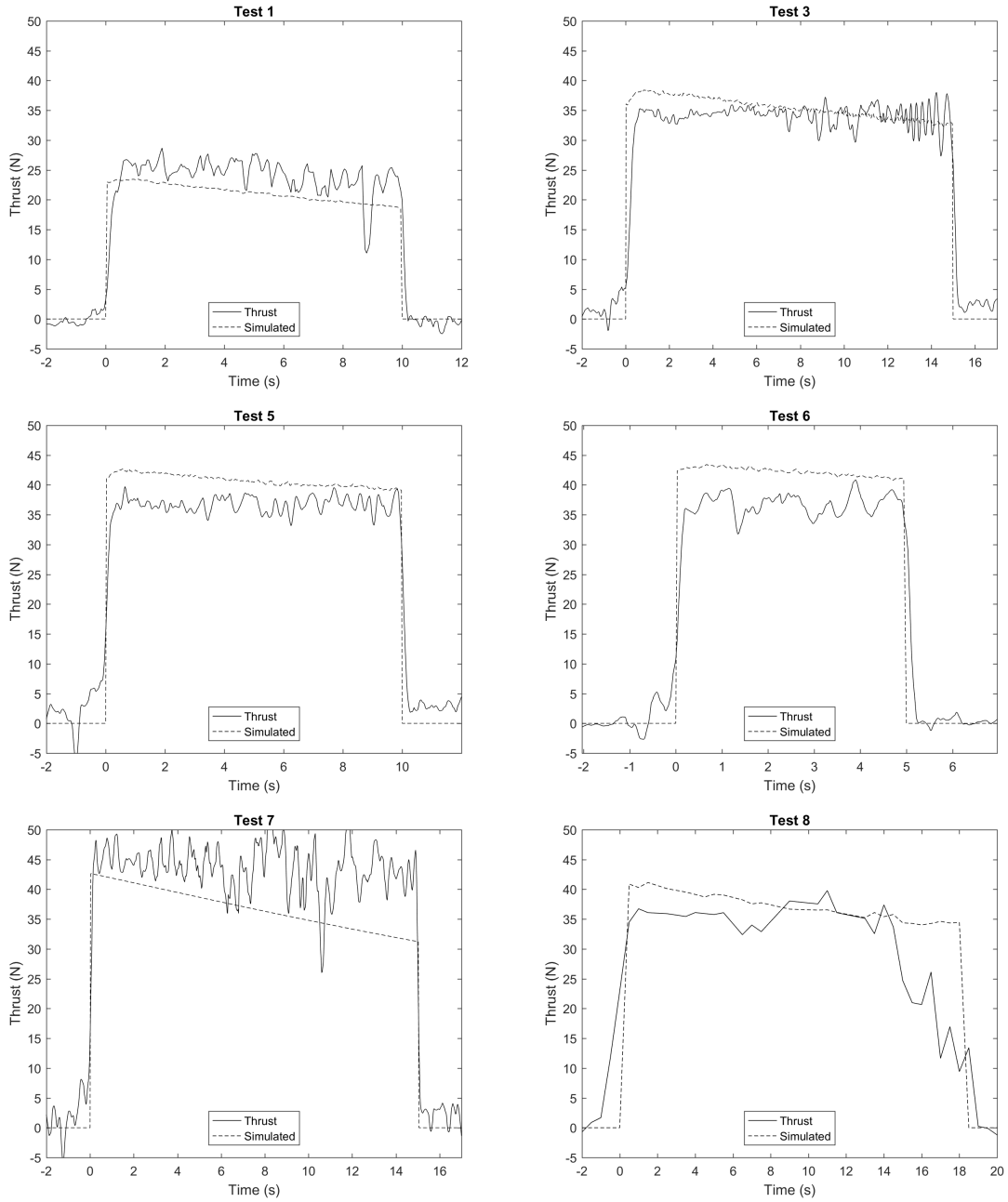


Figure 3.4: Real and simulated thrust data

3.3 Conclusion

The engine performed well, with only a single failure. Thrust was lower than hoped, due to low efficiency. Hybrid rockets normally operate with specific impulses over 200 seconds, while the engine produced in the 140 second range. Still, 140 seconds is respectable, and there is further work that could improve on this. For example, there has been some interesting work on helical fuel grains with FDM printed fuels [8]. These fuel grains increase fuel regression rates, which would increase thrust by increasing the mass flow, as well as increase the characteristic velocity by bringing the O/F ratio closer to the ideal ratio.

The model produced a accurate simulation of thrust, with errors ranging between 4.9% and 17%. Further refinement of the model over more tests could reduce these errors. The simulation of the fuel regression rate and total fuel consumed were very accurate, with errors between 0.8% and 4.1%.

The aim of this project was to design, simulate, and test a hybrid rocket engine. This was accomplished. The engine is now ready as a testbed for further experimentation and refinement.

Bibliography

[1] J.A. Dean. *Lange's Handbook of Chemistry*. McGraw-Hill, 1999.

[2] Philip Hill and Carl Peterson. *Mechanics and Thermodynamics of Propulsion*. Pearson, 1992.

[3] Cheng T Gindulyte A He J He S Li Q Shoemaker BA Thiessen PA Yu B Zaslavsky L Zhang J Bolton EE Kim S, Chen J. Pubchem database. Technical report, 2019.

[4] Matheson Gas. *Basic Flowmeter Principles*.

[5] NASA. *Computer Program for Calculation of Complex Chemical Equilibrium Compositions and Applications: Analysis*, 1994.

[6] NASA. *Computer Program for Calculation of Complex Chemical Equilibrium Compositions and Applications: Users Manual and Program Description*, 1996.

[7] Zachary W. Peterson. Closed-loop thrust and pressure profile throttling of a nitrous oxide/hydroxyl-terminated polybutadiene hybrid rocket motor. Master's thesis, Utah State University, 2012.

[8] Zachary S. Spurrier. Throttleable gox/abs launch assist hybrid rocket motor for small scale air launch platform. Master's thesis, Utah State University, 2016.

[9] Zachary W. Peterson Stephen A. Whitmore and Shannon D. Eilers. Analytical and experimental comparisons of htpb and abs as hybrid rocket fuels. *Annual AIAA/ASME/SAE/ASEE Joint Propulsion Conference*, 47, 2011.

[10] George P. Sutton. *Rocket Propulsion Elements*. John Wiley & Sons, 2001.

Appendix A

Arduino Code

A downloadable version of this code may be found in this project's github, https://github.com/benisnow456/Hybrid-Rocket-Engine

```
#include <SD.h>
#include <SPI.h>

boolean launch = LOW;
boolean record = LOW;
boolean control = LOW;
int percent = 0; // the percent valve will open

File myFile;
String fileName = "data.csv";
float calibration = 22.189; // calibration constant

// all times are in ms
unsigned long logFrequency = 25;
unsigned long streamFrequency = 500;
unsigned long ignitionDelay = 500;
unsigned long fullThrottleTime = 5000 + ignitionDelay;
unsigned long burnTime; // will be set by user input during startup

int minTime = 80; // DO NOT SET BELOW 80. Minimum valve cycle time

int ignitionPin = 2;
int valvePin = 3;
int pressurePin = A4;
int loadPin = A5;
int pinCS = 10; // chip select for the SD card

```

```
28 // Class for the ignition
29 class Ignition {
30
31   unsigned long prevTime = 0;
32   unsigned long startTime = 0;
33
34   // Constructor
35   public:
36   Ignition() {
37     pinMode(ignitionPin, OUTPUT);
38   }
39
40   void Update() {
41     unsigned long curTime = millis();
42     if ((launch) && (curTime - startTime > ignitionDelay)) {
43       digitalWrite(ignitionPin, HIGH);
44     }
45   }
46
47   void Initialize() {
48     unsigned long curTime = millis();
49     startTime = curTime;
50   }
51
52 };
53
54 // Class for the pressure transducer and load cell
55 class Sensor {
56
57   unsigned long prevLogTime = 0;
58   unsigned long prevStreamTime = 0;
59   unsigned long startTime = 0;
60   float offset; // calibration offset;
61
62   // Constructor
63   public:
64   Sensor() {
65     pinMode(pressurePin, INPUT);
66     pinMode(loadPin, INPUT);
67   }
68
69   void Update() {
70       long curTime = millis() - startTime;
71
72       if ((record) && (curTime - prevLogTime >= logFrequency)) {
```

```
73          int presValue = analogRead(pressurePin);
74          float presVoltage = (presValue/1023.0)*5;
75          float psi = (presVoltage-0.5)*500/4;
76
77          float loadValue = analogRead(loadPin);
78          float loadVoltage = (loadValue/1023.0)*5;
79          float weight = loadVoltage*calibration - offset;
80
81          prevLogTime += logFrequency;
82
83          String data = String(curTime) + "," + String(psi) + "," + String(weight);
84          data += ","  + String(launch) + "," + String(percent) + ",";
85          myFile = SD.open(fileName, FILE_WRITE);
86          myFile.println(data);
87          myFile.close();
88        }
89
90        if ((record) && (curTime - prevStreamTime >= streamFrequency)) {
91          int presValue = analogRead(pressurePin);
92          float presVoltage = (presValue/1023.0)*5;
93          float psi = (presVoltage-0.5)*500/4;
94
95          float loadValue = analogRead(loadPin);
96          float loadVoltage = (loadValue/1023.0)*5;
97          float weight = loadVoltage*calibration - offset;
98
99          prevStreamTime += streamFrequency;
100
101         String data = "Time: " + String(curTime) + "      Pressure: " + String(psi);
102         data += "      Load: " + String(weight) + "      launch: " + String(launch);
103         data += "      Percent: " + String(percent);
104         Serial.println(data);
105       }
106     }
107
108     void Calibrate() {
109         for (int i = 1; i <= 25; i++) {
110           float loadValue = analogRead(loadPin);
111           float loadVoltage = (loadValue/1023.0)*5;
112           float weight = loadVoltage*calibration;
113           offset += weight;
114           delay(20);
115         }
116         offset /= 25;
117         Serial.print("Offset calculated to be: ");
```

```
118           Serial.println(offset);
119       }
120
121       void Initialize() {
122         unsigned long curTime = millis();
123         startTime = curTime;
124       }
125
126   };
127
128
129   // Class for valve
130   class Valve {
131
132     boolean state = LOW;
133     unsigned long startTime = 0;
134     unsigned long prevTime = 0;
135
136     // Constructor
137     public:
138     Valve() {
139       pinMode(valvePin, OUTPUT);
140     }
141
142     void Update() {
143       unsigned long curTime = millis();
144
145       // full throttle burn
146       if ((!control) && (launch) && (curTime - startTime < burnTime)) {
147         digitalWrite(valvePin, HIGH);
148       }
149       // controlled burn, before controlled segment
150       else if ((control) && (launch) && (curTime - startTime < fullThrottleTime)) {
151         digitalWrite(valvePin, HIGH);
152       }
153       // controlled burn, during controlled segment
154       else if ((control) && (launch) && (curTime - startTime < burnTime)) {
155         int offTime;
156         int onTime;
157         if (percent >= 50) {
158           offTime = minTime;
159           onTime = percent*minTime/(100-percent);
160         }
161         else {
162           onTime = minTime;
```

```
163           offTime = (100-percent)*minTime/percent;
164         }
165         if ((state == LOW) && (curTime - prevTime >= offTime)) {
166           state = HIGH;
167           digitalWrite(valvePin, HIGH);
168           prevTime = curTime;
169         }
170         else if ((state == HIGH) && (curTime - prevTime >= onTime)) {
171           state = LOW;
172           digitalWrite(valvePin, LOW);
173           prevTime = curTime;
174         }
175       }
176       else {
177         digitalWrite(valvePin, LOW);
178         launch = LOW;
179       }
180     }
181
182     void Initialize() {
183       unsigned long curTime = millis();
184       startTime = curTime;
185     }
186
187   };
188
189
190   // Create instances of the classes defined above
191   Ignition ignition;
192   Sensor sensors;
193   Valve valve;
194
195   void setup()   {
196     Serial.begin(9600);
197     pinMode(pinCS, OUTPUT);
198     digitalWrite(pinCS, HIGH);
199     SD.begin();
200
201     if (!SD.begin(pinCS)) {
202       Serial.println("Card failed, or not present");
203     }
204
205     myFile = SD.open(fileName, FILE_WRITE);
206     myFile.print("TIME (mS),");
207     myFile.print("PRESSURE (PSI),");
```

```
208     myFile.print("LOAD (LBS),");
209     myFile.print("STATUS,");
210     myFile.println("PERCENT");
211     myFile.close();
212
213     Serial.println("Welcome to Kevin");
214     sensors.Calibrate();
215     Serial.println();
216
217     // read in total burn time
218     Serial.println("Enter the time in mS for the burn...");
219     while(1) {
220       if (Serial.available()) {
221         long command = Serial.parseInt();
222         burnTime = ignitionDelay + command;
223         Serial.print("Burn time set to: ");
224         Serial.println(command);
225         Serial.println();
226         break;
227       }
228     }
229
230     // determine what type of burn is requested
231     Serial.println(F("Enter 1 for a full throttle burn, or 2 for a controlled burn"));
232     Serial.println(F("Controlled burns will start after 5 seconds of full throttle..."));
233     Serial.println();
234     while(1) {
235       if (Serial.available()) {
236         int command = Serial.parseInt();
237         if (command == 1) {
238           Serial.println("You have selected a full throttle burn");
239           Serial.println();
240           break;
241         }
242         else if (command == 2) {
243            Serial.println("You have selected a throttled burn");
244            Serial.println("Enter the control percentage...");
245            while(1) {
246             if (Serial.available()) {
247               percent = Serial.parseInt();
248               if (percent != 0) { //if the percent has been updated
249                 Serial.print("Percent set to: ");
250                 Serial.println(percent);
251                 Serial.println();
252                 break;
```

```
253             }
254           }
255         }
256         control = HIGH;
257         break;
258       }
259     }
260   }
261
262   Serial.println(F("Commands you should know:"));
263   Serial.println(F("     Enter 1 to record data."));
264   Serial.println(F("     Enter 2 to end recording."));
265   Serial.println(F("     Enter 3 to launch."));
266   Serial.println(F("     Enter 4 to ABORT."));
267   Serial.println();
268 }
269
270 void loop() {
271   if (Serial.available()) {
272     char command = Serial.read();
273     switch (command) {
274       case '1':
275         record = HIGH;
276         sensors.Initialize();
277         Serial.println("Recording has begun");
278         Serial.println();
279         break;
280       case '2':
281         record = LOW;
282         Serial.println("Recording has ended");
283         Serial.println();
284         break;
285       case '3':
286         launch = HIGH;
287         valve.Initialize();
288         ignition.Initialize();
289         Serial.println("LAUNCH!");
290         Serial.println();
291         break;
292       case '4':
293         launch = LOW;
294         Serial.println("ABORT!");
295         Serial.println();
296         break;
297     }
```

```
298     }
299     ignition.Update();
300     valve.Update();
301     sensors.Update();
302 }
```

Appendix B

MATLAB Code

A downloadable version of this code may be found in this project's github, https://github.com/benisnow456/Hybrid-Rocket-Engine

B.1 Simulation class

```
1  classdef Simulation < handle
2
3      %% These properties are read only, or defined through initialization
4      properties (SetAccess=private)
5          time
6          mdot_O2
7          mdot_fuel
8          mdot_total
9          fuel_burned
10         OF
11         thrust
12         Ve
13         Pc
14         Pe
15         temp
16         Isp
17     end
18
19     %% These properties are hidden to simplify things
20     properties (GetAccess=private)
21         g0 = 9.81                  % standard gravitational acceleration
22         L_fuel = 0.180975;         % length of the fuel rod, in m
23         rho_fuel = 998;            % fuel density, in kg/m^3
```

```
24          gamma = 1.4;                % heat capacity ratio of oxygen
25          a = 0.00016;                % fuel regression rate coefficient
26          n = 0.31;                   % fuel regression rate exponent
27          Cstar_init = 1600;          % initial guess of Cstar, in m/s
28          P_atm = 101325;             % atmospheric pressure, in Pascals
29          nozzle_angle = 30;          % angle of the diverging nozzle
30          scaling = 0.7;              % scaling factor to include losses
31      end
32
33      methods
34          function obj = Simulation()
35          end
36
37          %% Main simulation function
38          function run(obj, D_throat, D_exit, D_port_init, mdot_O2, time)
39              A_t = pi*(D_throat/2)^2;
40              A_e = pi*(D_exit/2)^2;
41              r_init = D_port_init/2;
42              obj.mdot_O2 = mdot_O2;
43              obj.time = time;
44
45              assert(length(time) == length(mdot_O2),...
46                  'Time and oxidizer flow arrays must be the same length');
47              size = length(time);
48              dt = round(1/mean(diff(time)));  % steps per second
49
50              %% generate empty arrays to hold combustion parameters
51              r_p = zeros(size, 1);           % port radius
52              A_p = zeros(size, 1);           % port area
53              G_O2 = zeros(size, 1);          % oxidizer flux
54              rdot = zeros(size, 1);          % fuel regression rate
55              Cstar = zeros(size, 1);         % characteristic velocity
56
57              obj.mdot_fuel = zeros(size, 1); % fuel mass flow
58              obj.mdot_total = zeros(size, 1); % total mass flow
59              obj.OF = zeros(size, 1);        % O/F ratio
60              obj.thrust = zeros(size, 1);    % thrust
61              obj.Ve = zeros(size, 1);        % exhaust velocity
62              obj.Pe = zeros(size, 1);        % exhaust pressure
63              obj.Pc = zeros(size, 1);        % chamber pressure
64              obj.temp = zeros(size, 1);      % combustion temperature
65              obj.Isp = zeros(size, 1);       % Specific impulse, in s
66
67              burn_start = find(mdot_O2, 1);       % find start of othe burn
68              burn_end = find(mdot_O2, 1, 'last'); % find end of the burn
```

```
69              Cstar(burn_start) = obj.Cstar_init;
70              r_p(1:burn_start) = r_init;
71
72              % create CEA object
73              CEA_object = CEA;
74              CEA_object.setFuel('ABS', 100, 298.15);
75              CEA_object.setOxid('O2', 100, 298.15);
76              CEA_object.supar = A_e/A_t;
77
78              %% Loop to calculate combustion at each time step
79              for i=burn_start+1:burn_end
80                  r_p(i) = r_p(i-1) + rdot(i-1)/dt;
81                  A_p(i) = pi*r_p(i)^2;
82                  G_O2(i) = obj.mdot_O2(i)./A_p(i);
83                  rdot(i) = obj.a*G_O2(i)^obj.n;
84                  obj.mdot_fuel(i) = rdot(i)*2*pi*r_p(i)*...
85                      obj.L_fuel*obj.rho_fuel;
86                  obj.mdot_total(i) = obj.mdot_fuel(i)+obj.mdot_O2(i);
87                  obj.Pc(i) = obj.mdot_total(i)*Cstar(i-1)/A_t;
88                  obj.OF(i) = obj.mdot_O2(i)/obj.mdot_fuel(i);
89
90                  CEA_object.pressure = obj.Pc(i)/obj.P_atm; % input in atm
91                  CEA_object.OF = obj.OF(i);
92                  ioinp = CEA_object.input.rocket;
93                  data = CEA_object.run;
94                  Cstar(i) = data.CSTAR.EXIT1;
95                  obj.Ve(i) = data.Isp.EXIT1;
96                  obj.Pe(i) = data.P.EXIT1*100000; %CEA outputs P in bar
97                  obj.temp(i) = data.T.CHAMBER;
98              end
99
100             %set pressures to atmospheric if engine isn't on
101             obj.Pc(1:burn_start) = obj.P_atm;
102             obj.Pc(burn_end+1:size) = obj.P_atm;
103             obj.Pe(1:burn_start) = obj.P_atm;
104             obj.Pe(burn_end+1:size) = obj.P_atm;
105             %set velocity and Isp to 0 if engine isn't on
106             obj.Ve(1:burn_start) = 0;
107             obj.Ve(burn_end+1:size) = 0;
108             obj.Isp(1:burn_start) = 0;
109             obj.Isp(burn_end+1:size) = 0;
110
111             %Calculate and correct thrust
112             obj.thrust = obj.mdot_total.*obj.Ve + (obj.Pe-obj.P_atm)*A_e;
113             lambda = (1+cosd(obj.nozzle_angle/2))/2;
```

```
114                 obj.thrust = obj.thrust*lambda*obj.scaling;
115                 obj.Isp = obj.thrust./(obj.mdot_total*obj.g0);
116                 obj.fuel_burned = sum(obj.mdot_fuel)/dt;
117             end
118         end
119     end
```

B.2 Example

```
w = warning ('off','all');
close all;

%% Create a simulation object.
% The simulation class run method takes the inputs:
%    1) An empty simulation object
%    2) Nozzle throat diameter
%    3) Nozzle exit diameter
%    4) Initial fuel port diameter
%    5) Oxidizer mass flow array
%    6) Time array
%
% All units in the simulation are in SI.

%% Define simulation parameters
close all
burn_time = 10;
burn_delay = 0.5;
padding = 2;                    % seconds to display before and after burn
dt = 20;                        % steps per second

D_throat = (19/64)*0.0254;      % in meters
D_exit = 0.5*0.0254;            % in meters
D_port = (5/8)*0.0254;          % in meters

% These second order fitting coefficients were determined experimentally
O2_coeffic = [0.000010894432  -0.000720645427   0.023080625849];

%% generate a time array
size = (burn_time+padding*2)*dt+1;
time_array = linspace(-1*padding, burn_time+padding, size)';

%% generate an oxidizer flow array
burn_start = padding*dt;
burn_end = (padding+burn_time)*dt;
oxidizer_array = polyval(O2_coeffic, time_array+burn_delay);
oxidizer_array(1:burn_start) = 0;
oxidizer_array(burn_end+1:size) = 0;

%% run the simulation
sim = Simulation();
run(sim, D_throat, D_exit, D_port, oxidizer_array, time_array);

```

```
44  plot(sim.time, sim.thrust, 'k-');
45  ylabel('Thrust (N)');
46  xlabel('Time (s)');
```

Appendix C

User Guide

If you are reading this guide, you presumably want to test fire my engine. *Please* make sure you know what you are doing. While hybrid engines are safer than liquid engines, they are still complex systems that used improperly could have disastrous results. The engine has already exploded once. Take care that it does not explode again.

I encourage use to reach out to me at benisnow456@gmail.com before you go any further.

C.1 Setting up the engine

The engine requires a lot of individual steps to be completed before it can be fired. Missing a single step will almost certainly result in a misfire and/or catastrophic failure.

C.1.1 Oxygen

Connect a type E oxygen cylinder to the yoke. Make sure it is connected firmly. There is a chance that the yoke will leak slightly, so do not open the tank itself until right before you are ready to fire. Make sure the feed system looks like the image in figure 2.1, and that the manual valve is fully open.

C.1.2 Fuel

The fuel I used was 3D printed ABS. You can used all sorts of fuel types, but the simulation is only calibrated for ABS. I highly recommend trying more complex geometries, such as helical ports and wagon wheels. I did not have a chance to test these, but they a very interesting avenue of future research.

The fuel should be loaded into combustion chamber between the precombustion chamber and the nozzle.

Make sure that the o-rings on both the nozzle and precombustion chamber are replaced after each test.

If you skip this step, the engine **will** catch on fire.

I used size 223 Buna-N o rings, which are only rated to 250 F, and temperatures on the exterior of the engine case surpassed the boiling point of water, so I would definitely suggest Viton or high temperature silicone, even though it is more expensive. If you use Buna-N, do not fire the engine for longer than 15 seconds. I would assume that higher temperature O rings could be used for longer, but I haven't tested them.

The precombustion chamber and nozzle are held in by snap rings, and also a washer by the nozzle. Make sure these are seated correctly.

The entire combustion chamber assembly is connected with the three U bolts, which should be tightened only to finger tight.

C.1.3 Igniter

My pyrotechnic igniters are described in chapter 2. They are MJG firewires in the finger tip of a plastic glove with half a gram each of black powder and steel wool. The wire leads should be connected to the terminal junction just below and to the right of the power strip. Polarity does not matter. The igniter itself is glued into the precombustion chamber.

C.2 Electronics

This is the most finicky part of the engine. I lost data twice to a poor SD card connection.

Make sure that the 10 amp power supply is connected to the red daughter board, and the 2 amp charger powers the Arduino. Mixing these up will prevent the igniter from getting enough current to ignite.

A micro SD should be installed in the slot on the daughter board. Arduino can only read/write to SD cards formatted in a FAT32 file system, so make sure that the card is properly formatted. Make sure the SD card is empty before each test.

Ensure that all wires are connected correctly, and then place the USB dongle with the Xbee radio module to a computer's USB port. Open the Arduino IDE, and open the port that the dongle is connected to. Power up the Arduino and daughter board, and verify that the Arduino is transmitting. It will say "Welcome to Kevin" (my nickname for the engine) if it is transmitting properly. The Xbee pros have a range in the hundreds of feet, but they need line of sight.

At this point, just follow the prompts. Make sure that you are logging data before starting a burn, and that you stop logging data once the burn is finished.

I would not recommend using the controlled burn option, as it was not very effective and risks damaging the valves.

C.3 Testing

The engine is loud and dangerous. I would not stand any closer than 50 feet at a bare minimum, and further if you're trying anything novel.

Do a dry run before each actual burn. By a dry run I mean keep the oxygen tank closed and don't wire the igniter, but make sure the solenoid valve clicks open, the ignition relay switches, and data is logged and streamed.

Be aware that the engine is LOUD. As in, you can clearly hear it from hundreds of feet away. You probably don't need ear protection, but I never actually measured the sound volume, so I can't say for sure if the sound is at a safe level.

If you have everything connected, enter your burn time, start logging data, and press fire. After a half second delay, it will be obvious if it works.

Happy rocketing!