Reaction Control System for High Altitude Balloons

Characterizing the Cold Gas Thruster for Reaction Control

Name: _____

Max Huggins

A thesis presented in partial fullfillment for

Honors in the Bachelor's of Science Degree

Faculty Mentor: _____

William Slaton, Ph.D.

Reader: _____

Carl Frederickson, Ph.D.

Department of Physics and Astronomy

University of Central Arkansas

April 30, 2020

Reaction Control System for High Altitude Balloons

Characterizing the Cold Gas Thruster for Reaction Control

Max Huggins

Abstract

Several methods exist which allow scientists to collect various types of data from Earth's atmosphere. High altitude balloons are one of these methods, which provide scientists a cost effective, simple system for data collection. At the cost of simplicity, however they lack control. Sporadic winds in the atmosphere affect data collection for sensors on board these balloon's payloads. A reaction control system would allow stabilization through regions of the atmosphere to allow better data collection. Here, the development of a reaction control system utilizing cold gas thrusters has been outlined. From the decision of fuel to the methods of characterization. An analysis has been done to characterize the behaviour of the thrusters, but due to an incomplete data set the characterization was unable to be completed. A high-level outline on future work has been laid out along with the experimental and analytical scripts.

COVID-19

This was a two semester project, performed in last year of a Bachelor's of Science Degree. The first semester was spent working to develop the background knowledge and the experiment. The second was spent actually setting up the experimental apparatus and data acquisition system; as well as collecting data. However, the University of Central Arkansas began online instruction March 13, 2020 and access to the research labs was limited due to COVID-19. Because of this, some necessary data collection was not completed and progress on the project was halted. The analysis and results that could be completed are discussed here.

In hopes of future work being completed, an outline will be presented for later work to be done.

Contents

Nomenclature

ϵ	Expansion ratio
ϵ^*	Area ratio yielding maximum force for the theoretical value
ϵ_m	Area ratio yielding maximum force for the measured value
γ	Ratio of specific heats
ω_z	Angular velocity about z axis
ρ	Density
a_w	Acceleration caused by wind
C	A constant relevant to the enthalpy at stagnation
C_D	Discharge coefficient
F	Force or thrust
F_w	Force caused by wind
H	Enthalpy
I_w	Impulse caused by wind
$I_{sp,m}$	Measured specific impulse
I_{sp}	The specific impulse
I_{sp}^*	Theoretical specific impulse

J	The mechanical equivalent of heat
KE	Kinetic energy
M	Mach number
m_p	Mass of propellant
m_{pd}	Mass of the payload
n	Number of moles
$N1$	No expansion, trial 1
$N2$	No expansion, trial 2
$O1$	Optimum area ratio, trial 1
$O11$	Overexpanded area ratio 1, trial 1
$O12$	Overexpanded area ratio 1, trial 2
$O2$	Optimum area ratio, trial 2
$O21$	Overexpanded area ratio 2, trial 1
$O22$	Overexpanded area ratio 2, trial 2
$O31$	Overexpanded area ratio 3, trial 1
$O32$	Overexpanded area ratio 3, trial 2
$O41$	Overexpanded area ratio 4, trial 1
$O42$	Overexpanded area ratio 4, trial 2
P	Pressure
Q	Heat put into a system
R	Universal gas constant

r	Radius
S	Entropy
T	Temperature
t	Time
U	Internal energy
$U11$	Underexpanded area ratio 1, trial 1
$U12$	Underexpanded area ratio 1, trial 2
$U21$	Underexpanded area ratio 2, trial 1
$U22$	Underexpanded area ratio 2, trial 2
$U31$	Underexpanded area ratio 3, trial 1
$U32$	Underexpanded area ratio 3, trial 2
$U41$	Underexpanded area ratio 4, trial 1
$U42$	Underexpanded area ratio 4, trial 2
V	Volume
W	Molecular weight of the gas
W	Work
X_c	Represents some variable, X, in the nozzle chamber
X_e	Represents some variable, X, in exit plane of the nozzle
X_f	Represents some variable, X, at some final state
X_i	Represents some variable, X, at some initial state
X_s	Represents some variable, X, at stagnation

X_t Represents some variable, X, in the nozzle throat

X_x Represents some variable, X, at some point x

Chapter 1

Introduction

1.1 Background

High altitude balloons (HABs) are large balloons to which one or several payloads are attached that can reach high altitudes. These payloads are insulated packages typically holding scientific equipment for data collection. Here, high altitude balloon payloads will be referred to as HABPs. Some examples of sensors that may be on board the HABPs are: temperature, altitude, pressure, wind, radiation, and so on. These can provide information for meteorologists and researchers about conditions in high levels of the atmosphere with prolonged data collection. This is unlike other methods of atmospheric data collection like rockets which can also reach these altitudes but do it over a much shorter time-span. Satellites offer longer lifetimes, but much higher prices. HABs can also provide safer, more cost effective options for data collection. However, there are some downfalls of using balloons as the vehicle for sensitive instruments. They can be subject to high winds and any on board sensors will be affected by this. For example, during the 2017 total solar eclipse any payloads sent up to observe this would have been subject to winds throughout their flight. This means that camera footage observing the sun's corona would never be stable and set on the subject it is trying to observe. The cameras would be passive observers controlled by the wind rather than the scientist. Another example is Geiger counters observing radiation in the atmosphere. These may have inconsistent data depending on what part of the payload is in its path from the sun. Wind speed sensors will also be subject to the forces of the winds. Keeping up with

the inconsistent direction of the payload is a difficult task and the sensor is provided wind speed that is not in any particular direction. This is not ideal to determine the direction of the winds and would require much post-processing to match up the direction the payload was facing at a given wind speed data point. These are only three specific examples, but winds will affect many other data collection systems. The goal of this project is to develop a system that stabilizes the payload against wind speeds.

1.2 Outline

The ultimate objective here is to fly a reaction control system on board a HABP. Getting to this point requires several steps. A high-level overview is listed:

1. Determining viability of different RCSs
2. Choosing a RCS
3. Characterizing the chosen RCS
4. Implementing the RCS
5. Ultimate objective

1.2.1 Options

The relevant options that were considered for the RCS can be divided into two categories. The first is of the gyroscopic type. These can offer both passive an active stabalization. Reaction wheels (RWs) are commonly used in satellites and telescopes in order to change the attitude of a craft with extreme accuracy. There are also control moment gyroscopes (CMGs) which are essentially reaction wheels mounted on a gimbal controlled by a motor. These produce much higher torques than RWs and more typically provide active stabilization. Both CMGs and RWs rely on electricity as their energy source; while crafts like the International Space Station along with others can utilize solar panels ([14] and [16]) for the longevity of missions, HABs cannot. They would rely on a battery to store all of the energy used to move the craft. Another drawback to these gyroscopic systems is their slow reaction time

([13]). These are typically not used for stabilizing a craft throughout a flight. Rather they provide reactions to torques from solar radiation pressure, gravity gradient, magnetic fields, micrometeorite impact, and internal effects like gas leakage and moving parts [4]. In the case of these scenarious it is not a matter of time and unpredictability. The reason for their slow reaction times is that they rely on motors spinning up to change the direction of the craft. The other type of option consdiered here is the thruster type. This takes the form of both combustion and cold gas thrusters. These cannot offer the same amount of accuracy as the former, but can provide extremely fast changes in momentum. They are more commonly used for stabalizing crafts utilizing rockets and rely on compressed air energy storage (CAES) or chemical energy storage rather than electricity.

Of the two, the thruster types have more desireable properties. Primarily in their ability to provide faster changes in the attitude of the craft. Now, the consideration to be made is between the combustion or cold gas thruster. Obviously, sticking combustible fuel on board a HAB is not considered safe so the CGT is more desireable. Also, the design of a CGT system (CGTS) has a lower level of complexity ([10]). The choice of system for this project is the CGT.

1.2.2 Fuels

An in-depth analysis was not necessarily done regarding the choice of the CGT for stabalization in the previous section. So it is important to determine the viability of several types of common fuels. To determine the viability of a CGT, a variable called the specific impulse (I_{sp}) must be defined. Equation 1.1 provides the definition of the specific impulse.

$$I_{sp} = \frac{F}{dm_p/dt} \tag{1.1}$$

Where F is the force and dm_p/dt is the change in mass of propellant with respect to time. This value provides a *standard* so to speak for characterizing fuels as it describes the force production per some amount of mass spent. Different fuels have different $I_{sp}s$ and generally

speaking the higher the value the better the fuel. [1] Additionally, given an impulse, the amount of fuel required to counteract the impulse can be determined. This describes stabalization well. Looking at some sample data from [3], the amount of fuel required to stabilize a flight can be estimated. The raw flight data was not given in form of a .txt or .csv, but plots of the raw data are shown. From this, a digitizer tool was used to extract numbers from the gyroscopic plots. Specifically, the angular velocity about the axial direction (ω_z). Multiplying this by the radius of rotation (r) and then taking the derivative with respect to time gives the accerlation (a_w) due to the forces (F_w) of the wind. Multiplying this by the mass of the payload (m_{pd}) then determines the force in the relevant direction due to the wind. Using the theoretical values for the specific impulse of different gases, the mass of fuel needed for a flight could be determined. In other words,

$$a_w = \frac{d(\omega_z r)}{dt} \tag{1.2}$$

$$F_w = m_{pd} a_w \tag{1.3}$$

The integration of all these values would provide the total impulse caused by the wind (I_w).

$$I_w = \int |F_w| dt \tag{1.4}$$

$$m_p = \frac{I_w}{I_{sp}} \tag{1.5}$$

In reality equations 1.2, 1.3, and 1.4 are performed for each data point throughout the data set. The derivative is the difference in two points next to each other and the integral is a trapezoidal sum throughout the flight. Also, since the force would take both positive and negative values, the absolute value of each force term was taken.

The most commonly used gas in a CGT is by far nitrogen; the reason for this is reasonable propellant storage density, performance and lack of contamination concerns [11]. Some other options are $CO_2,\ H_2,\ and\ He$; H_2 having the highest specific impulse. Obviously, the specific impulse is not the only important variable when considered what makes the best fuel for

[1] It is worth noting the units for I_{sp} are Ns/kg, but most sources will record the value in units of s. Multiplying m_p by the acceleration due to gravity is the only difference.

a system. Factors such as safety, availability, cost, energy storage density, and so on all contribute to the choice of gas. The theoretical specific impulse values of the gases listed above are shown in table 1.1. Along with the specific impulse, the mass required for each gas to stabilize the payload for the total trip is recorded. In the third column, the mass

Gas	I_{sp} (s)	$Mass$ (g)	$Mass\ Ozone$ (g)
H_2	296	172	5
He	179	284	8
N_2	80	636	18
CO_2	67	760	21

Table 1.1: The specific impulse of several gases and how much fuel is required from them during a flight

of fuel needed for the time the payload spends in the ozone layer is also recorded. It is not always necessary to stabilize the payload throughout the entirety of the flight. In fact, the target may be a specific altitude region so in this case stabilization in other regions is irrelevant. To put these numbers into perspective, the CO_2 cannisters that mountain bikers use to refill their tires are typically 16g cartridges. The CO_2 and the cartridge together weigh approximately 60g. The maximum mass allowed by the FDA for the type of HABP dealt with here is 2721.554g; meaning one of these 16g cartridges is only approximately 2% the total mass of the payload. A 25g cartride is still only approximately 3.8% the total mass and well within the volume constraints as well. So, at a high-level view, the CGTS is a seemingly viable option.

This analysis also provides insight as to some design considerations that should be made for the system. The goal is *not* to maximize the force overall, but rather to maximize the force per unit mass spent in the system. The type of force required to stabilize the system is on the scale of 1×10^{-3} N as was previously determined in the analysis of some example flight data. This means the thruster doesn't have to be powerful necessarily, but it should be efficient to conserve on fuel.

Since several trials will be recorded availability of the gas is largely important to this project.

Out of these gases the availability of CO_2 is by far the highest. Even with the lowest I_{sp}, it is still an attractive option. Additionally, the actual implementation of the CGT is a distance away from this point in the project. As each gas should behave similarly in the system, it would not be a difficult task to switch the type of gas at any given point during the project. If further analysis finds the specific impulse to be of more importance then N_2 would be the most likely option for the aforementioned reasons.

1.3 Characterizing

The characterization of the RCS is referring to fitting the actual results of the system to the pre-exisisting theory's results. This is reffering to the theory discussed in chapter 2. From a high-level, it is likely the assumptions made to generate the mathematical equations do not fit the actual behaviour of the system. In fact, the book, reference [7], from which the actual nozzle theory is taken states the theory produces an error of approximately 10% for actual rocket engines. The goal in characterizing the system is to create a proportionality between the predicted and actual results in order to specify the predicted results for the system developed here. To accomplish this, an on-ground testing rig is built consisting of the same parts to be integrated into the actual payload. The on-ground rig measures variables of interest in the system that will allow the comparison between the predicted and actual results. This will be discussed in more detail in chapter 2, section 2.3. In the high-level outline provided, this is the section where progress was halted due to the COVID-19 crisis. From this point further in the project is considered an outline for future work.

1.4 Integration

After characterization of the system, the actual integration into a HABP must be made. This is a straightforward task and involves adding another thruster and solenoid to the on-ground testing rig and removing some sensors that are no longer necessary. Then fitting the plumbing into the HABP. Additionally, the scripting for stabilization must be developed further and specified for the characteristics determined. A CAD rendering of the plumbing

system is shown in figure 1.1. This integration would include simulating windy conditions

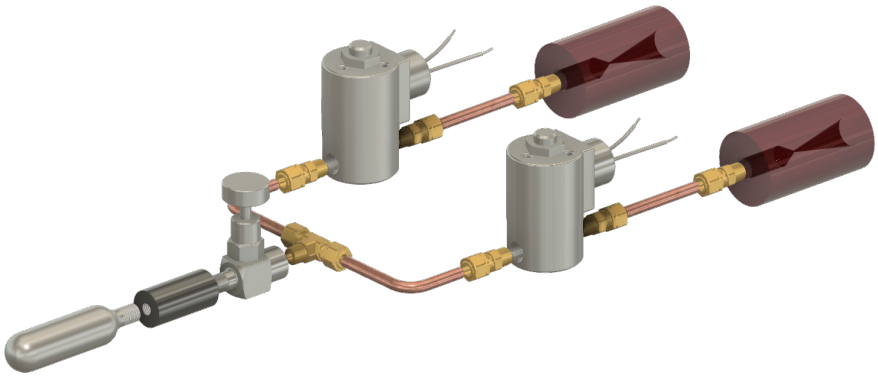


Figure 1.1: CAD rendering of the plumbing system to be used in the actual payload

and determining impulse data from accelerometer sensors to see how the system would work for actual flights. As well as determining the type of feedback system to be used for the RCS script. Examples of these are all negative feedback systems using some type of sensor such as a magnetometer, photoresistor, accelerometer, etc to provide the controller a reference for the pointing direction. This is the point where the amount of fuel needed for a flight objective could be well defined and the necessary changes could be implemented.

1.5 Flight

After confidence in the system has been established from the on-ground testing and simulations, the goal and best test is a flight of the RCS. Here, it is important that the sytem is integrated *with* a data acquisition system and acts as an *aid* in the data acquisition. The flight objective has not been well-defined yet. Some vague examples are listed:

- Point at a ground target for a duration of the flight
- Take footage of only the sun for a duration of flight
- Record radioactivity with the sensor unobstructed towards the sun through the ozone layer

The system should be robust enough such that completion of these tasks is not a matter of altering the CGTS in any way, but rather altering the scripting and feedback system.

Chapter 2

Theory

The objective of this section is to derive the relationships used for the analysis of the data collection. First, the general thermodynamic relationships will be established, then they will be applied to converging-diverging nozzle. At this point, the relevant relationships for the data will be established and the experimental design can be discussed in detail.

2.1 General Thermodynamic Relationships

The theory used here is mostly taken from pre-existing nozzle theory. The source for this project is reference [7]. Much of the derivation for the equations to be used are based on the following assumptions and rely on Classical Thermodynamics:

1. The propellant is homogeneous throughout the nozzle.
2. The propellant behaves like a perfect gas.
3. There is no friction between the propellant and the nozzle walls
4. The system is adiabatic
5. The propellant flow is one-dimensional
6. The propellant flow is stagnate in the nozzle chamber
7. The propellant velocity is uniform accros any cross-section normal to the nozzle axis

Reference [7] refers to three other assumptions, but they either do not apply or are redundant for the CGTS. Additionally, all of the variables are clearly defined in the nomenclature section but will not be defined along the discussion here.

Starting with the first law of thermodynamics (FLT),

$$dQ = dU + PdV \tag{2.1}$$

and according to the second law,

$$dQ = TdS \tag{2.2}$$

defining the specific heats at constant volume and pressure respectively,

$$c_V = \left(\frac{\partial Q}{\partial T}\right)_V \tag{2.3}$$

$$c_P = \left(\frac{\partial Q}{\partial T}\right)_P \tag{2.4}$$

and using Joule's equation to define a perfect gas

$$\left(\frac{\partial U}{\partial V}\right)_T = 0 \tag{2.5}$$

substitutting 2.1 into ∂Q in 2.3

$$c_V = \left(\frac{\partial U}{\partial T}\right)_V \tag{2.6}$$

Also, substitutting 2.2 into ∂Q in 2.3

$$c_V = T\left(\frac{\partial S}{\partial T}\right)_V \tag{2.7}$$

so

$$c_V = \left(\frac{\partial U}{\partial T}\right)_V = T\left(\frac{\partial S}{\partial T}\right)_V \tag{2.8}$$

similarly with c_P

$$c_P = \left(\frac{\partial U}{\partial T}\right)_P = P\left(\frac{\partial V}{\partial T}\right)_P \tag{2.9}$$

this means 2.2 can be written as

$$dQ = c_V dT + PdV \tag{2.10}$$

Considering a perfect gas, the equation of state is

$$PV = nRT \tag{2.11}$$

substitutting $n = m/W$ provides

$$PV = \frac{mRT}{W} \tag{2.12}$$

so the equation for 1 unit of mass is

$$PV = \frac{RT}{W} \tag{2.13}$$

differentiating PV gives

$$d(PV) = VdP + PdV \tag{2.14}$$

$$= \frac{R}{W}dT \tag{2.15}$$

so

$$PdV = \frac{R}{W}dT - VdP \tag{2.16}$$

Defining a new constant, γ

$$\gamma = \frac{c_P}{c_V} \tag{2.17}$$

from this, 2.16, 2.10, and 2.4

$$c_V = \frac{R}{W(\gamma - 1)} \tag{2.18}$$

similarly,

$$c_P = \frac{\gamma R}{W(\gamma - 1)} \tag{2.19}$$

It can also be determined for an adiabat, $dQ = 0$, that

$$P_i V_i^{\gamma} = P_f V_f^{\gamma} \tag{2.20}$$

From the previous relations and the definition of enthalpy,

$$H = U + pV \tag{2.21}$$

the enthalpy per unit mass can now be expressed as

$$H = \frac{\gamma RT}{W(\gamma - 1)} \tag{2.22}$$

2.2 The Nozzle

Until this point, all of these relations are quite general. Figure 2.1 displays a typical converging diverging nozzle. It consists of three important regions. The furthest left is the chamber, it is here that we assume the variables to be stagnate. Stagnate meaning the velocity of the gas is zero. The next section going in the $+x$ direction is the nozzle throat. This is the smallest cross-sectional area of the system. Next is not a particularly important point, but represents the variables at any point, x, in the nozzle. Lastly is the exit plane of the nozzle. To apply

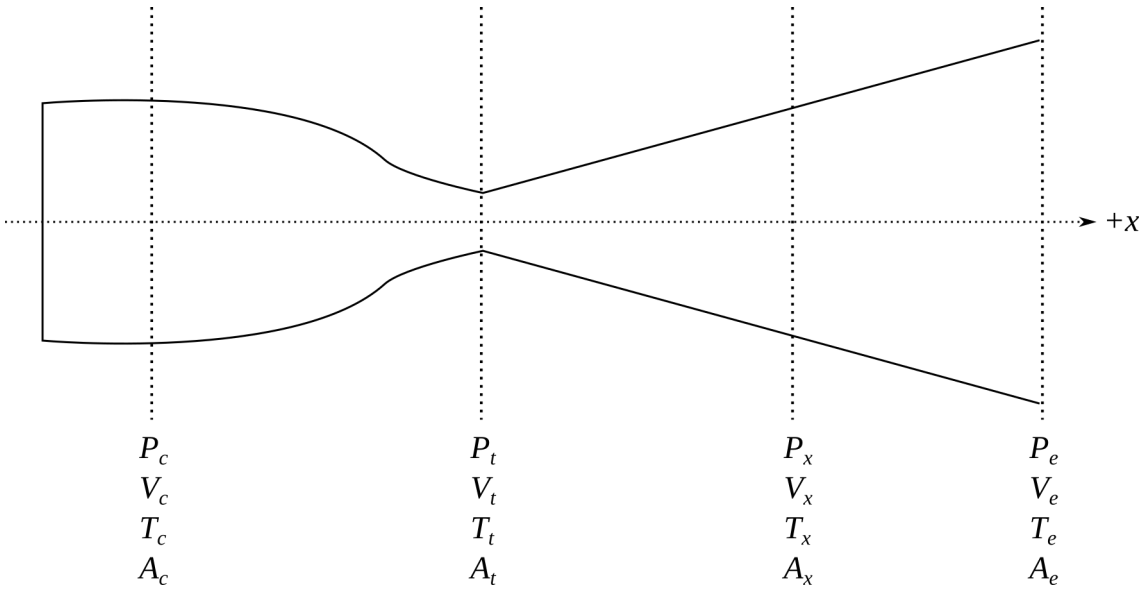


Figure 2.1: Simplified nozzle, with axis and some variables defined.

the previous definitions to a nozzle, the kinetic energy of 1 unit of mass will be considered along with the enthalpy. If the flow of the gas is considered, the enthalpy will increase by the amount equal to the kinetic energy of the gas per 1 unit of mass, given the gas exchanges no heat with the environment

$$KE = \frac{v^2}{2} \tag{2.23}$$

$$H_x = \frac{\gamma R T_x}{W(\gamma - 1)} \tag{2.24}$$

$$H_f = H_x + KE \tag{2.25}$$

A region where the gas is stagnant ($v = 0$) can be considered, here the enthalpy is a constant. Substitutting 2.13 and the specific volume ($V = 1/\rho$) as well,

$$H_s = C = \frac{\gamma P_s}{W\rho_s(\gamma - 1)} \tag{2.26}$$

$$= \frac{\gamma}{(\gamma - 1)}\frac{P_x}{W\rho_x} + \frac{v_x^2}{2} \tag{2.27}$$

Now, considering an adiabatic system

$$\frac{V_s}{V_x} = \left(\frac{P_x}{P_s}\right)^{\frac{1}{\gamma}} \tag{2.28}$$

From 2.26 and 2.28 it can be found that

$$\frac{P_x}{P_s} = \left(1 - \frac{v^2}{2C}\right)^{\frac{\gamma}{\gamma-1}} \tag{2.29}$$

2.2.1 Mach Number

An important parameter is the mach number, which is defined as the ratio of the gas velocity at some point to the velocity of sound in the gas. Here, this will be manipulated to determine the parameter in terms of temperature variables.

$$M_x = \frac{v_x}{v_{sound}} \tag{2.30}$$

$$= \frac{v_x}{\sqrt{\gamma R T_x}} \tag{2.31}$$

$$\tag{2.32}$$

This can be substitutted into the previously defined relationships to find

$$\frac{T_s}{T_x} = 1 + M_x^2\frac{\gamma - 1}{2} \tag{2.33}$$

From 2.4 and the fact that the net heat change is equal to the change in kinetic energy per unit mass, the

$$dQ = c_P dT \tag{2.34}$$

$$= c_P(T_x - T_{x+dx}) \tag{2.35}$$

$$= \frac{(v_{x+dx}^2 - v_x^2)}{2J} \tag{2.36}$$

Where J is defined as

$$J = \frac{W}{Q} \tag{2.37}$$

Substitutting 2.19, using the stagnation conditions for the initial values, and solving for v_x,

$$v_x = \sqrt{\frac{2\gamma R T_s}{(\gamma - 1)W}\left(1 - \left(\frac{P_x}{P_s}\right)^{\frac{\gamma-1}{\gamma}}\right)} \tag{2.38}$$

Now, the velocity of the gas at any point x is in terms of variables in regions where flow is stagnated. The temperature can be found similarly,

$$T_x = T_s - \frac{v_x^2}{2Jc_P} \tag{2.39}$$

substitutting 2.38 and 2.39 into the mach number

$$M^2 = \frac{2}{(\gamma - 1)}\left(\frac{T_s}{T_x} - 1\right) \tag{2.40}$$

Lastly,

$$\frac{M_{x+dx}}{M_x} = \frac{v_{x+dx}}{v_x}\sqrt{\frac{T_x}{T_{x+dx}}} \tag{2.41}$$

2.2.2 Area Ratio

The expansion ratio is the single most important parameter when designing an efficient nozzle. This is the ratio of the exit plane area to the throat plane area. To introduce the variable, the mass flow rate will be defined.

$$w = \frac{dm}{dt} \tag{2.42}$$

In terms of the geometry of the nozzle, the velocity of the gas, and the density

$$w = A_x v_x \rho_x \tag{2.43}$$

The mass flow rate must be constant throughout any given time in the system. This is the equation of continuity for the flow. The ratio of any two points for a unit mass can now be analyzed,

$$\frac{A_{x+x}}{A_x} = \frac{V_{x+dx} v_x}{V_x v_{x+dx}} \tag{2.44}$$

Using 2.28 for the general case, 2.40, and 2.41

$$\frac{A_{x+dx}}{A_x} = \frac{M_x}{M_{x+dx}} \left(\frac{T_x}{T_{x+dx}} \right)^{\frac{1}{2}} \left(\frac{1 + \frac{(\gamma-1)}{2} M_{x+dx}^2}{1 + \frac{(\gamma-1)}{2} M_x^2} \right)^{\frac{1}{\gamma-1}} \tag{2.45}$$

The expansion ratio then can be expressed as

$$\epsilon = \frac{A_e}{A_t} = \frac{M_t}{M_e} \left(\frac{T_t}{T_e} \right)^{\frac{1}{2}} \left(\frac{1 + \frac{(\gamma-1)}{2} M_e^2}{1 + \frac{(\gamma-1)}{2} M_t^2} \right)^{\frac{1}{\gamma-1}} \tag{2.46}$$

To determine the best value for the expansion ratio, the mass flow rate will be maximized and using previously found relationships the value for the expansion ratio will be found. Starting with substitutting 2.38 into 2.44.

$$w = \frac{A_x}{V_x} \sqrt{\frac{2\gamma R T_s}{(\gamma - 1) W} \left(1 - \left(\frac{P_x}{P_s} \right)^{\frac{\gamma-1}{\gamma}} \right)} \tag{2.47}$$

and

$$\left(\frac{T_x}{T_s} \right)^{\frac{1}{\gamma-1}} = \left(\frac{P_x}{P_s} \right)^{\frac{1}{\gamma}} \tag{2.48}$$

Also,

$$\frac{1}{V_x} = \frac{1}{V_s} \left(\frac{T_x}{T_s} \right)^{\frac{1}{\gamma-1}} \tag{2.49}$$

substitutting 2.49 and 2.48 into 2.47 and rearranging

$$w = \frac{A_x}{V_s} \sqrt{\frac{2\gamma K T_s}{(\gamma - 1)} \left(\left(\frac{P_x}{P_s} \right)^{\frac{2}{\gamma}} - \left(\frac{P_x}{P_s} \right)^{\frac{\gamma+1}{\gamma}} \right)} \tag{2.50}$$

This can now be maximized by differentiating with respect to P_x and set equal to zero. This gives

$$\frac{dw}{dP_x} = \frac{A_x}{2V_s} \left[\frac{2\gamma K T_s}{(\gamma - 1)} \left(\left(\frac{P_x}{P_s} \right)^{\frac{2}{\gamma}} - \left(\frac{P_x}{P_s} \right)^{\frac{\gamma+1}{\gamma}} \right) \right]^{-\frac{1}{2}} * \frac{d}{dP_x} \left(\left(\frac{P_x}{P_s} \right)^{\frac{2}{\gamma}} - \left(\frac{P_x}{P_s} \right)^{\frac{\gamma+1}{\gamma}} \right) = 0 \tag{2.51}$$

Obviously,

$$\left(\left(\frac{P_x}{P_s} \right)^{\frac{2}{\gamma}} - \left(\frac{P_x}{P_s} \right)^{\frac{\gamma+1}{\gamma}} \right) \neq 0 \tag{2.52}$$

so

$$\frac{d}{dP_x} \left(\left(\frac{P_x}{P_s} \right)^{\frac{2}{\gamma}} - \left(\frac{P_x}{P_s} \right)^{\frac{\gamma+1}{\gamma}} \right) = 0 \tag{2.53}$$

differentiating and solving for P_x/P_s

$$\frac{P_x}{P_s} = \left(\frac{2}{\gamma+1}\right)^{\frac{\gamma}{\gamma-1}} \tag{2.54}$$

Now, substitutting 2.48 it can be seen that

$$\frac{T_x}{T_s} = \frac{2}{\gamma+1} \tag{2.55}$$

which when compared to 2.33 it is seen that they are equal when if $M = 1$, and if $x = t$, then the optimum value for P_t can be determined. When $M = 1$ at the throat is not necessarily the only condition which maximizes w but it is also a condition which satisfies maximum force production of the nozzle. This can be seen quite easily. If the assumption is made that the conglomorate of particles exiting the nozzle act as a rigid body with some collective velocity and momentum, then it can be said the force generatd by that body is

$$F_{gas} = \frac{d(mv_e)}{dt} \tag{2.56}$$

Also, the exhaust velocity is considered to be constant with time, so

$$F_{gas} = \frac{dm}{dt}v_e \tag{2.57}$$

$$= wv_e \tag{2.58}$$

Other forces acting on the nozzle are due to the pressure differences of the exit plane and ambient pressures, where $P = F/A$,

$$F = wv_e + (P_e - P_a)A_e \tag{2.59}$$

Here it can be seen that maximizing w, maximizes the force produced by the nozzle. Substitutting 2.47 and 2.38

$$F = A_t P_s \sqrt{\frac{2\gamma^2}{\gamma-1}\left(\frac{2}{\gamma+1}\right)^{\frac{\gamma+1}{\gamma-1}}\left(1-\left(\frac{P_e}{P_s}\right)^{\frac{\gamma-1}{\gamma}}\right)} + (P_e - P_a)\,A_e \tag{2.60}$$

From this, the I_{sp} can also be determined using 1.1. In a similar method, the following expression is found

$$I_{sp} = \left[\frac{2\gamma RT_s}{(\gamma-1)W}\left(1-\left(\frac{P_e}{P_s}\right)^{\frac{\gamma-1}{\gamma}}\right)\right]^{\frac{1}{2}} + \frac{(P_e - P_a)}{P_s}\frac{A_e}{A_t}\frac{1}{C_D} \tag{2.61}$$

where C_D is the discharge coefficient and is defined by equation 2.62.

$$C_D = \sqrt{\frac{\gamma\rho_c}{P_s}}\left(\frac{2}{\gamma+1}\right)^{\frac{\gamma+1}{2(\gamma-1)}} \tag{2.62}$$

2.3 Relationships

Looking at equations 2.61 and 2.60 it can be seen that the variables are not easily measured. It would be simpler to measure the temperature values instead. A change of variables can be performed plugging 2.20 and 2.40 into them. The following expressions are obtained:

$$F = A_t P_c\left(\sqrt{\frac{2\gamma^2}{\gamma-1}\left(\frac{2}{\gamma+1}\right)^{\frac{\gamma+1}{\gamma-1}}\left(1-\frac{T_e}{T_c}\right)} + \left(\left(\frac{T_e}{T_c}\right)^{\frac{\gamma}{\gamma-1}} - \frac{P_a}{P_c}\right)\epsilon\right) \tag{2.63}$$

$$I_{sp} = \left[\frac{2\gamma R T_c}{W(\gamma-1)}\left(1-\frac{T_e}{T_c}\right)\right]^{\frac{1}{2}} + \left[\left(\frac{T_e}{T_c}\right)^{\frac{\gamma}{\gamma-1}} - \frac{P_a}{P_c}\right]\frac{A_e}{A_t}\sqrt{\frac{P_c}{\gamma\rho_c}}\left(\frac{\gamma+1}{2}\right)^{\frac{\gamma+1}{2(\gamma-1)}} \tag{2.64}$$

These equations are the basis for this experiment. In determining values for these variables the theoretical and experimental values can be compared. Let I^*_{sp} denote the theoretical specific impulse and $I_{sp,m}$ denote the measured specific impulse. As these are functions of ϵ ($I^*_{sp}(\epsilon^*)$) it could be found a proportionality between I^*_{sp} and $I_{sp,m}$ such that

$$I^*_{sp}(\epsilon^*) = I_{sp,m}(\epsilon_m * e_{ff}) \tag{2.65}$$

where e_{ff} denotes some efficiency factor. The same procedure can be done for the force equations. Determining this would allow for the specify the optimum expansion ratio for this system at various ambient pressures.

Chapter 3

Methods

3.1 Characterizing the CGT

The objective is to determine the difference between the measured specific impulse and the theoretical values. To determine the experimental value for the specific impulse is a matter of measuring the force production of the nozzle and the change in mass of the system. This is done with the use of two force sensors. One measuring the change in mass of the CO_2 cylinder and the other measuring the force production of the nozzle. To determine the theoretical values for the same system requires measuring $T_c,\ T_e,\ P_c,\ and\ P_a$ where the ambient pressure is a constant and can be determined from the forecast that day. The same variables are needed for the predicted force equation. This information is coming from equations 2.63 and 2.64. It is important that these variables are represented well by the data collected, however the experimental setup used in the data collection presented here does a poor job of this. The setup and methods will be presented here along with solutions to the problems faced.

3.1.1 On-Ground Testing Rig

A simplified schematic of the experimental apparatus is shown in figure 3.1.

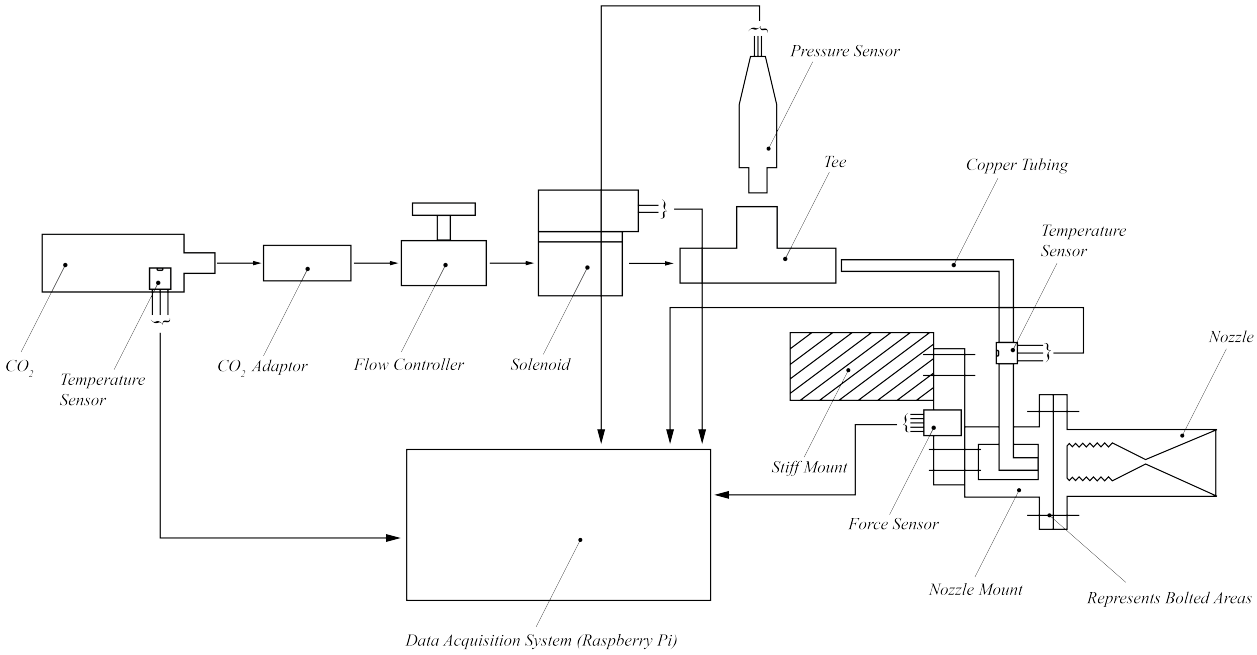


Figure 3.1: Simplified on-ground testing rig

Hardware

All of the parts, except the CO_2 adaptor and pressure transducer were ordered from McMaster-Carr. The adaptor was custom machined and the transducer was purchased from a third party seller with no datasheet provided. However, the voltage output from it varied linearly with pressure and could be calibrated with a simple linear regression function. The solenoid is a 24V DC 3000psi solenoid valve, it was controlled by a simple digital signal run through a push pull (NPN-PNP pair) amplifier. Each temperature sensor was a type k thermocouple, interfacing with a AD8495 breakout baord made by Adafruit. These breakoutboards interfaced with the data acquisition system via a 10-bit analog to digital converter. The force sensor used was a 1kg load cell interfacing with an HX711 (24-bit analog to digital converter) breakout board. To handle several different voltage levels, several linear regulators were used to provide a 12V, 5V, and 3.3V supplies. The solenoid, along with the regulators, was powered directly by the 24V lab bench power supply. The importance of having a current capable supply to handle all the relatively large power consumption is paramount. Without it data sensors provide erroneous values with massive errors ([15]). Additionally, a stable and low noise supply should be used. The solenoid measured a serious resistance of 60Ω and was connected

directly to a 24V supply. This can cause shifting in the ground level if care is not taken to provide a very low resistance, low inductive path to ground. For more information regarding the schematics of this setup, see Appendix A, section A.2. The plumbing system used is 1/4" OD copper tubing with compression fittings that allowed adaptations between the copper pipe and the 1/8" National Pipe Threading (NPT) that was used for the non-pipe fittings. The choice of this plumbing system is the high pressure rating and relatively low weight. The most massive part of the system was the solenoid valve. One budgeting approximately 19% the maximum allowed weight of the HABP. On final thought, a choice was made to omit the flow controller because the mass flow rate should be limited by the nozzle throat.[1] For sake of simplicity these devices are referred to as "sensors" preceeded by the variable they measured. So P_c sensor, T_e sensor, T_c sensor, and F sensor. Additionally, γ is dependent on temperature, but only changes slighty in the temperature differences noticed here. In later models, γ will be set to be a function of temperature for greater accuracy. For now it is not necessary because the force has a small dependence on γ compared to the temperature. A complete list of parts and schematics is available in Appendix A section A.3.

Nozzle Manufacturing

Several options were considered for the manufacturing of nozzles. These included the processes listed below:

- Fused deposition modeling (FDM) 3D printing
- Resin 3D printing
- Machining by hand
- Modeling and casting by hand

However, in dealing with such small parts modeling and casting by hand was immediately discarded. Modeling differences of expansion ratios on the scale of parts of a millimeter was out of the question. Machining by hand is both costly in terms of money and time. Also, such small nozzles would present difficulties in the machining process. FDM 3D printing

[1] This assumption will be discussed further in Chapter 5

was by far the most accesible out of these, but there are problems related to both accuracy and actual function of the nozzles. FDM 3D printers typically use a .4mm nozzle. This limitation on accuracy is relatively large considering the nozzle throat is designed to by no larger than .625mm in diamter. Also, the .4mm nozzle refers to the actual size of the hole in the nozzle, *not* the size of the filament extruded from the nozzle. FDM printers typically produce layer heights no smaller than .1mm. In addition to the problem of accuracy, the parts are relatively porous. The method of FDM printing calls for the extrusion of a hot filament onto a cooled layer of filament to produce a 3D model. This does well creating structural parts not dependent on high tolerances and do not need to be air-tight, but for a CGT nozzle it will not work well. Much of the fuel would be lost in the layers of the nozzle and the friction between the layers and exhaust gases would be moreso than any of the other methods. This leaves resin printing, often referred to as stereolithography (SLA) printing. This method has been out of reach for many in years past, but recent manufacturers have developed robust, inexpensive resin printers. These printers can provide layer heights of .01mm and do not rely on laying lines of hot filament on top of other layers. Rather, they use two main methods of creating the part. The cheaper versions use a 5.5x3.5in UV LCD screen with resolutions of approximately 2560x1400 pixels. These resolutions are much finer than the FDM printer's capabilities. Additionally, these layers are formed by a chemical curing of one layer of resin to the previous. Given no impurities in the resin, this will provide an air-tight seal in the part. Figure 3.2 shows a pressure test that was performed on a printed part[2], similar to the designed nozzles. The part was essentially an end cap, filled in increments of 20psi up to 400psi. It was held at 400psi for 6 hours and it held that pressure with no noticeable change in the needle on the guage throughout that time.[3] Similar tests with FDM printed parts held no pressure, even after being treated with an epoxy coating. For comparing the accuracy of the parts, several prints were done. Figure 3.3 displays an example of the difference in detail and figure 3.4 shows a closeup of the two at the same scale.

[2]This part has been soaked in IPA to reveal interior details of the part

[3]The o-ring may have been noticed, this is not a necessary part for typical NPT fittings, but was added to ensure a seal with the resin part.

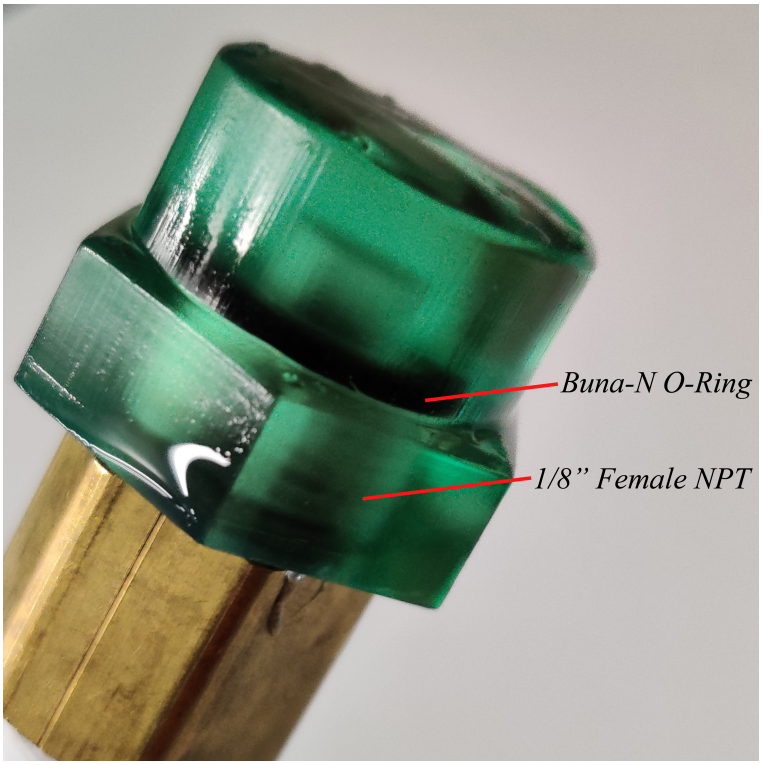


Figure 3.2: Pressure test of resin print from the ELEGOO MARS 3D printer

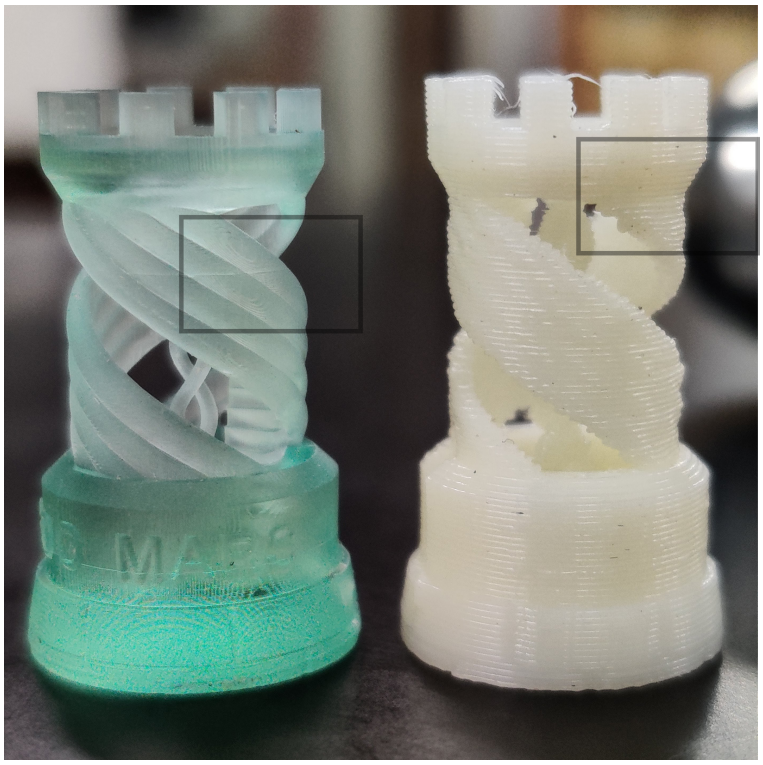
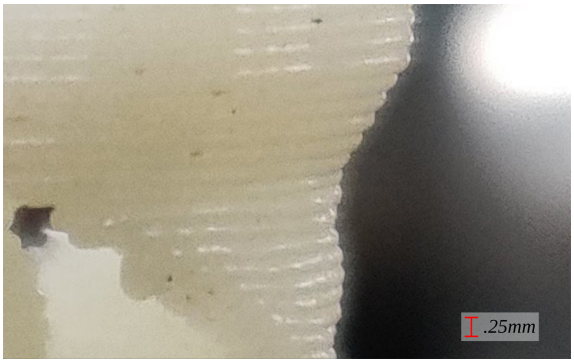


Figure 3.3: Two 3D printed rooks for detail comparison. On left, resin and on right, FDM

(a) Resin printed rook closeup



(b) FDM printed rook closeup

Figure 3.4: Rook closeups

The layer height for the resin printed part was set to .05mm and the FDM is .25mm. The difference is easily noticed.

Nozzle design was done in two programs. First, the nozzle geometry was made in OpenSCAD. An open source program was used to set the parameters for the nozzle geometry. A .stl file was generated from OpenSCAD and imported into Blender. From here, the actual functional

parts were added to the model like the hex flatends, NPT fitting, exterior walls, and labelling. Figure 3.5 shows several nozzles begin cleaned in isopropyl alcohol (IPA). The actual nozzle geometry can be seen while the part is covered in IPA. There were a total of 10 nozzle

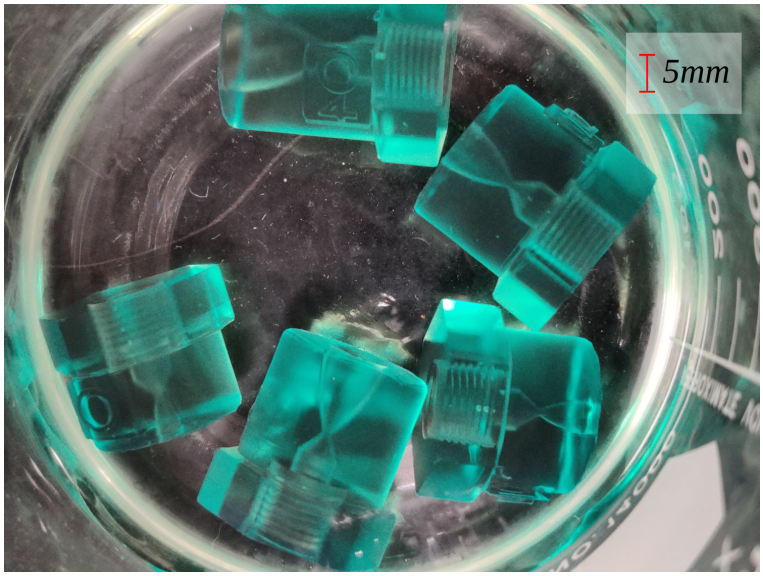


Figure 3.5: Several 3D printed nozzles bathing in IPA

expansion ratios used with 2 trials each. The expansion ratio kept A_t constant and varied A_e. There were 4 over expanded, 4 underexpanded, 1 optimum, and 1 with no expansion. The naming scheme goes something like ERT, where E stands for either U (Underexpanded), N (None), or O. O stands for either optimum or overexpanded. R is the ratio number (1-4) except in the case that there is no third character. In this case the O stands for optimum, which there is only one of. T stands for trial number which can be either 1 or 2. So for example, U22 is underexpanded, ratio number 2, trial number 2. O1 is optimum ratio, trial number 1. These are explicitly defined in the nomenclature section.

Data Acquisition System

The data acquisition system used here was a Raspberry Pi 3. The language used for scripting was Python. The project is hosted in a GitHub repository.[4] There are two main scripting sections in the repository. The scripting for the on-ground testing rig was developed first. In

[4]Physical URL: https://github.com/maxmhuggins/RCS_HAB/tree/master/On_Ground_Testing/Code

this section there are classes for each of the data sensors. These allow for simple implementation of each of them in the main experiment file. Essentially, each of these sensors is defined as an object in the experiment file and this is the file which initiates each trial. This is a relatively straightforward file that allows for the calibration of sensors if need be. Calibration for the P_c sensor is a simple linear regression plotting observed voltages against the known pressures. The slope for this line allowed a conversion betwen observed ADC voltages and pressure values. The force sensor calibration used a method of known masses. Known masses are added to the scale and the voltage values were measured. These values are averaged and the unit conversion is formed by the division of the read voltage and the known mass. This is done for each mass and the unit conversions are averaged. The Adafruit breakout boards for the thermocouples do not necessarily require calibration. However, in the datasheet ([1]) there is a relevant section discussing error analysis and generally the devices are accurate to $\pm 2^oC$ and $\pm 1^oC$ initially. Major sources of error come from differences in the reference junction on the PCB. To ensure the thermocouples were reading similar values they were testing next to each other and a laser thermometer was used as a reference value. Both performed similarly. Most important to this project is that the thermocouples produced similar voltage values.

The analysis for the data was done under another section and is all object oriented. These scripts include an analyzer class and a file for generating plots using the analyzer class. All of the computation is completed in the analyzer class and the figure generator is simply that, a figure generator. For ease of access the analyzer and experiment Python files are displayed in Appendix A, section A.1.

3.1.2 Data Collection Simplified

The data taken was for the variables in equation 2.63. The code began the experiment after all sensors were calibrated and functioning properly. Data collection ran until the force production was below a certain threshold for a predefined number of points.[5] After it was determined over the mass of the CO_2 canister was measured. All of this data was written

[5]This is discussed further in Chapter 5

to two seperate .txt files. One included the calibration values for the sensors and changes in mass, the other included P_c, T_c, T_e, and F. The previous was not recorded in a useful format and later work had to be done to resolve this. In the future this should be updated. The latter was simply comma delimited and worked fine for reading into python for analysis. The data files can be found at this GitHub repository[6] as well.

[6]Physical URL: https://github.com/maxmhuggins/RCS_HAB/tree/master/On_Ground_Testing/Data_and_Analysis/Experimental_Data_Analysis/Refined_Data_Files

Chapter 4

Analysis

4.1 Initial Analysis

To begin the analysis, simple plots were made to look at the sensor data over time. Immediately, it was found that the temperature data is quite poor! The resolution of the ADC is evident in the discrete steps that the temperature takes. Figure 4.1 shows an example of the temperature data. The limited resolution of the ADC can be seen quite clearly. However, this

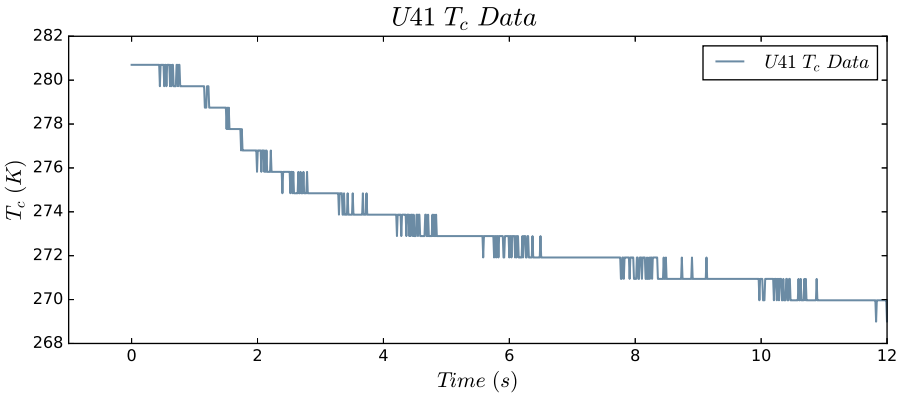


Figure 4.1: Example plot of the temperature over time for a trial

is not terribly problematic and several things can be done to smooth the curve. A function from the Pandas Python library that handles smoothing well was used to compensate for this. It uses a moving averages method. The smoothed plot is shown in figure 4.2. There are additional problems that come with smoothing data, though. The total number of points is

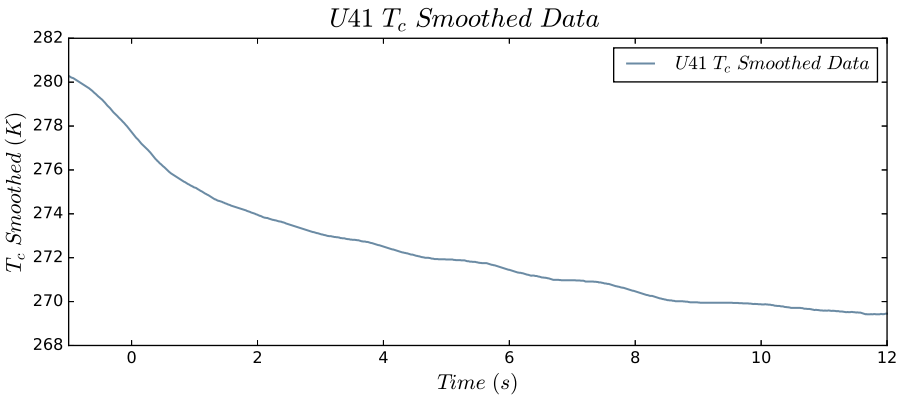


Figure 4.2: Example plot now smoothed

typically reduced and therefore all of the data that is going to be used must then be smoothed by the same factor. Also, the data is shifted to the left as can be seen from figures 4.1 and 4.2. Alternatively, a function could be fit to the data and this would allow the possibility of a continuous data set. For now though, there are bigger problems with the data recorded. Figure 4.3 shows the two different temperature data sets plotted against time for one of the trials. It can be seen immediately that at some point the T_c line drops below the T_e line.

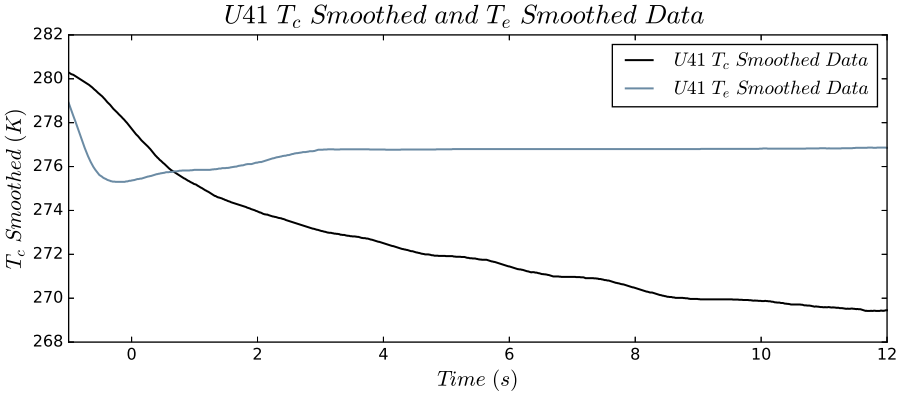


Figure 4.3: Smoothed T_c and T_e plotted with time for one of the trials

Looking back at equation 2.63 this is troublesome. This means that $\left(1 - \frac{T_e}{T_c}\right) < 0$ giving an overall negative sign under the square root. Obviously the force produced is not imaginary, so either the data collected for these temperatures is not representative of the actual variables or the theory is wrong. It is obvious that the former is the problem for several reasons. The first is that the T_c variable represents the temperature directly before the converging section

of the nozzle. T_e is the temperature of the exit plane of the nozzle. The two thermocouples are not placed in these regions. Admittadely, there was not enough attention when inspecting the theory to develop the experiment. Under normal circumstances some simple changes would be made and new data would be taken, however due to the closure of the university nothing can be done about collecting new data. Because of this we will proceed with what analysis can be completed and use a more general approach to characterizing the CGT. First, the predicted force is plotted with the experimental force in figure 4.4. As expected, segments

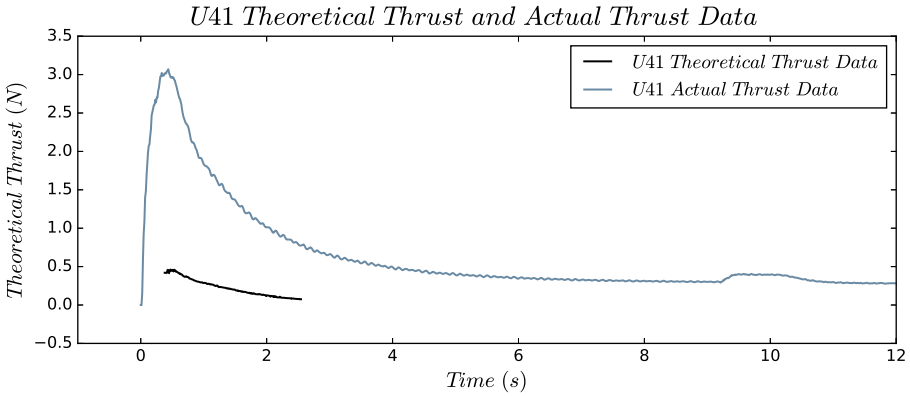


Figure 4.4: First attempt at plotting the thrust and predicted thrust against time

where the force is undefined due to a runtime error in the Numpy Python package being used for the square root function.

4.2 Attempt at Reconciliation

The most obvious way to resolve the negative sign is to include an absolute value around the problematic term. This solves the runtime error, but is unfounded. The optimum nozzle expansion ratio should have the largest thrust production, but $T_e's$ dependence on the exit area is essentially removed on the expansion ratio in doing this. This means the thrust will be highest for the value which has the greatest expansion ratio which is geometry O4. This can be seen in figure 4.5. Several trials have been plotted there and included the predicted force according to equation 2.63. O4 is indeed the largest thrust curve. Also, in viewing these thrust curves it is important to recognize the predicted force is the predicted force per unit mass, but the actual curve is per some different mass value. However, the unit for the

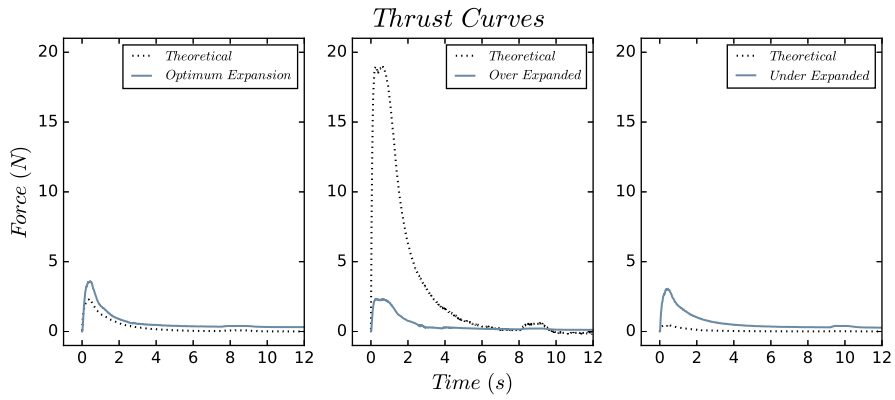


Figure 4.5: Several trials thrust curves plotted with the predicted thrust

mass can be set to some convenient unit and be compared through all the data. This won't change the actual analysis and design conditions.

Tacking on an absolute value will not only fail to resolve the issue here but it is also unfounded. The next thought is to solve for T_e in terms of the other available variables. It turns out that T_e can be put in terms of $T_c,\ P_c,\ A_e,\ \rho_c,\ and\ w$. Unfortunately, there are several problems with this. The first is that the change in mass, the mass flow rate, was only measured as an initial and final value. Meaning the total mass used throughout the experiment is known, but the mass flow rate throughout is not. For now, the assumption will be made that the mass flow rate is constant.[1] Next, as can be seen from equation 4.1 T_e cannot be solved algebraically. It is not difficult to solve for it computationally though.

$$T_e = T_c \left[\frac{\gamma - 1}{2\gamma P_c \rho_c} \left(\frac{w}{A_e} \right)^2 \left(\frac{T_c}{T_e} \right)^{\frac{\gamma+1}{\gamma-1}} + 1 \right]^{-1} \tag{4.1}$$

The simplist method is to use the relaxation method as described in [12]. The process is to iterate the equation by plugging in $T_e = 1$ on the right side, finding a new value for T_e on the left side, substitutting this value back into the right side, and so on. T_e should converge to a value and that is the solution for T_e. Using this method, I've made the same plots in figure 4.6 as in figure 4.5. The similarity between the two figures is quite apparent. At first glance, they nearly look the same, but looking at the far right plot the theoretical curve can be seen to be different from the other. The reason for this is because the assumption made

[1] Later, it will be seen this is not the case.

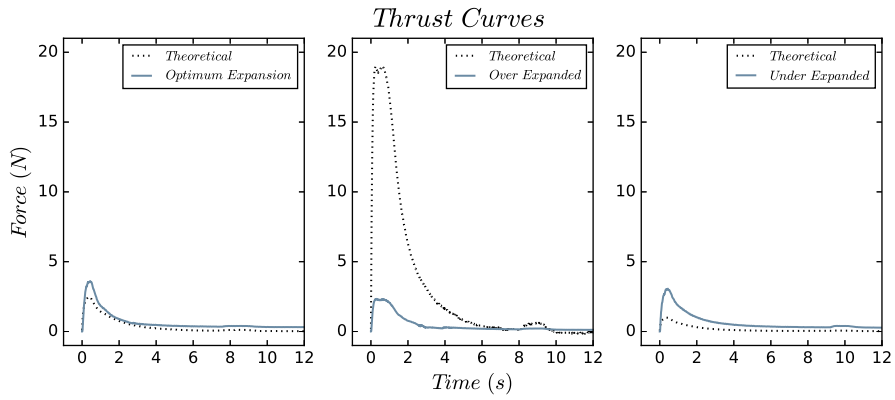


Figure 4.6: Several trials thrust curves plotted with the predicted thrust

about the mass flow rate being constant through time was naive. The mass flow rate, at any point x can be defined by equation 2.42.

$$w_x = A_x v_x \rho_x \tag{4.2}$$

Both the velocity and density of the gas have dependence on pressure and temperature so w is *not* constant with respect to time. That is not to say it is not constant at any given time throughout the entire system, however. If T_e is not represented correctly it is clear that the result is that the nozzle with the largest expansion ratio is predicted to function best because of equation 2.63's dependence on A_e.

Specific impulse values for the experiment can technically be determined. Unfortunately, this doesn't tell much because the theoretical values cannot be calculated for the same reasons as the force. Additionally, it is not ideal to look at the specific impulse over the entire experiment because the temperature and pressure are so transient. Determining these values is a simple trapezoidal sum over all time for the force data and division by the change in mass over the time of the experiment. These values are tabulated in table 4.1. In the end, there is not much reconciliation that can be done for the poor data collection.

Geometry	$Trial\ 1\ I_{sp}\ (s)$	$Trial\ 2\ I_{sp}\ (s)$
N	61.1	54.0
U4	58.0	44.1
U3	33.5	52.8
U2	62.5	60.3
U1	33.9	62.6
O	57.5	37.3
O1	58.6	59.5
O2	50.5	48.3
O3	29.0	60.3
O4	44.6	56.0

Table 4.1: Experimental specific impulses

Chapter 5

Discussion

Unfortunately, no serious results were obtained from the data collected in this project. It can be seen, though, that the values found are close to the known values from reference [2]. The $I'_{sp}s$ dependence on temperature and pressure means that these values are meaningless without the theoretical values at the same pressure and temperature. Reference [2] is using values for the maximum possible I_{sp} values. However, the importance of this research is the development of the system and scripting. The focus on both readability and generalizability is the value here. Additionally, the work done here has allowed the realization of certain errors. The solutions to these errors are presented below.

The most obvious problems with this setup are shown in the analysis. The exit plane temperature is not well represented and neither is the mass flow rate. Additionally, the stagnation conditions for the actual system are not well-defined.

5.1 Hardware

Here, the problems regarding the hardware of the project are discussed.

5.1.1 ADC

The analog to digital converter is used to convert the analog voltage signals from the variable sensors to digital signals that the Raspberry Pi can read in. The resolution of the device is

defined by the number of *bits* it has. The MCP3008 is a 10-bit ADC used for the temperature sensors and the , which means the number of levels (N) available for storing voltage signals is 1024. In other words,

$$N = 2^{bits} \tag{5.1}$$

The HX711 is a 24-bit ADC, meaning $N = 16777216$. The HX711 has the capability of being 16384x more accurate than the MCP3008. The poor resolution is clearly seen in the raw temperature plots, such as the one shown in figure 4.1. The reason the HX711 was not used for the temperature data, however was the open source library interfacing with it was meant to be used specifically for the load cell. The chip was also on a breakout board and did not have other channel options. Most of the 24-bit ADCs require the use of a standard communication protocol, I2C. Finding open source libraries for these higher resolution devices is difficult and the user would likely have to build one themselves. However, this is not completely out of reach as there are many sources providing information on the subject and implementations in Python do exist. This was not a complete necessity though and was put aside for future design iterations.

5.1.2 Pressure Regulator

The stagnation conditions discussed in chapter 2, were not well defined prior to the first run of trials. Because of this, data that is meant to be stagnant is not. Also, since the CO_2 was spent so quickly it caused some of it to fuse into a solid. This is definitely not ideal for several reasons. The first is that solid CO_2 can block plumbing and cause issues for the RCS. Next, is the previously mentioned issue of stagnation. If the CO_2 changes temperature so rapidly, the chamber pressure will also be decreasing rapidly. This creates an ill-defined stagnation condition and an optimum nozzle geometry is impossible to correctly determine. With the use of a pressure regulator the stagnation conditions could be clearly determined. Also, this could allow the use of a feedback system that would add heat to the system such as to condition the temperature to stay constant even through a continuous discharge. It may also be possible to create the same effect by decreasing the nozzle throat radius. Either way, this should most likely be done because the mass flow rate for the system is too high.

5.2 Software

Changes to the software include a better method for stopping each experiment. The current method as discussed in Chapter 4 is to determine when the force value dropped below a certain threshhold for a specified amount of time. The force sensor sometimes lost its zero position and created problems for stopping data collection. A more robust method would be using time derivatives of the force to classify when the data collection should end. This would create more consistent data files and plots. The scripts should also be updated for the addition of the new hardware mentioned above.

Chapter 6

Conclusion

The over arching goal of this research project is to develop a reaction control system for high altitude balloons. The work completed includes an analysis on viability of the system, an experimental design, data collection, and analysis on it. The experimental design had errors associated with it and because of this the analysis did not provide system characterization. Next steps include correcting the experimental errors, finishing the analysis, integrating the CGT into a HABP, and flying the system. Additional thoughts on future work are the possibility of attempting another axis of stabilization. In this project, only two thrusters were ever really considered. This is because the payload does not see many perturbations in the upward or downward direcitons due to the nature of the balloon system. The biggest consideration here is not just fuel limitations but also the weight limitations. The heaviest part of the system is by far the solenoid; to add another axis of stabilization means adding two more solenoids.

In correcting the experimental errors, the options are not left open. There are clear solutions that can be proceeded with quickly. After this though, the project is much more open-ended. Actually integrating the cold gas thruster system into the high altitude balloon payload is not clearly defined here. There are several interesting methods to go about implementing this as an effective reaction control system and the carryover is quite similar to the systems used to stabilize rockets, satellites, etc. Because of this there is paramount information available for proceeding. The many applications of the RCS show its robust nature and the possibilities with it.

Bibliography

[1] Inc. Analog Devices. Ad8494/ad8495/ad8496/ad8497 (rev. c). https://www.analog.com/media/en/technical-documentation/data-sheets/ad8494_8495_8496_8497.pdf, 2018. Datasheet for thermocouple amplifier.

[2] Assad Anis. Cold Gas Propulsion System - An Ideal Choice for Remote Sensing Small Satellites. In Boris Escalante, editor, *Remote Sensing - Advanced Techniques and Platforms*. InTech, June 2012.

[3] Emanuel Bombasaro. Titan 1 flight data report. page 44, april 2016.

[4] Roland A Boucher. Electrical propulsion for control of stationary satellites. *Journal of Spacecraft and Rockets*, 1(2):164–169, 1964.

[5] Charles Gurrisi, Raymond Seidel, Scott Dickerson, Stephen Didziulis, Peter Frantz, and Kevin Ferguson. Space Station Control Moment Gyroscope Lessons Learned. page 16.

[6] Max Huggins. Rcs for habs. https://github.com/maxmhuggins/RCS_HAB, 2020.

[7] NH Langton. *Space Research and Technology.: Rocket Propulsion*. University Press, 1970.

[8] Edward B. "Magrab and Donald S." Blomquist. *"The Measurement of Time-Varying Phenomena: Fundamentals and Applications"*. "John Wiley and Sons, Inc", 1971.

[9] Ralph" "Morrison. *"Grounding and Shielding Techniques in Instrumentation"*. "John Wiley and Sons, Inc", 1977.

[10] Juergen Mueller and Juergen Mueller. Thruster options for microspacecraft - A review and evaluation of existing hardware and emerging technologies. In *33rd Joint Propulsion Conference and Exhibit*, Seattle,WA,U.S.A., July 1997. American Institute of Aeronautics and Astronautics.

[11] Juergen Mueller and Juergen Mueller. Thruster options for microspacecraft-a review and evaluation of existing hardware and emerging technologies. In *33rd joint propulsion conference and exhibit*, page 3058, 1997.

[12] Mark Newman. *Computational Physics*, volume 1. Mark Newman, 1 edition, 2012.

[13] Shahin S Nudehi, Umar Farooq, Aria Alasty, and Jimmy Issa. Satellite attitude control using three reaction wheels. In *2008 American Control Conference*, pages 4850–4855. IEEE, 2008.

[14] David J Richie, Vaios J Lappas, and Phil L Palmer. Sizing/optimization of a small satellite energy storage and attitude control system. *Journal of Spacecraft and Rockets*, 44(4):940–952, 2007.

[15] F.N.H." "Robinson. *"Noise and Fluctuations"*. "Oxford University", "Oxford", 1974.

[16] Panagiotis Tsiotras, Haijun Shen, and Chris Hall. Satellite attitude control and power tracking with energy/momentum wheels. *Journal of Guidance, Control, and Dynamics*, 24(1):23–34, 2001.

Appendix A

Appendix Title

A.1 Scripting

Here, two primary files are included. The first is the file that runs the experiment for data collection. To see all of the necessary class files for this, see reference [6]. This is shown below:

```
import __init__ as I

#===Constants===#
ForceThreshold = .25

#===ADC-Channels===#
PressureChannel = 0
CO2TempChannel = 1
PipeTempChannel = 2

#===BCM-Channels===#
OutputEnable = 22
SolenoidPin = 23
DT = 5
SCK = 6

#===Instantiating-Sensors===#
I.OutputEnable(OutputEnable)
```

```
Solenoid = I.SND.Solenoid(SolenoidPin)
PressureSensor = I.PR.PressureTransducer(PressureChannel)
ForceSensor = I.HX(DT, SCK)
CO2TempSensor = I.TEMP.TemperatureTransducer(CO2TempChannel)
PipeTempSensor = I.TEMP.TemperatureTransducer(PipeTempChannel)

#===Initializing=Data=Lists===#
TimeData = []
ForceData = []
PressureData = []
CO2TempData = []
PipeTempData = []
COUNTDOWN = range(0,61)

#===Beginning=the=Experiment===#
try:
    print("First, define the nozzle's geometry. This should be one of these:\
   nN \nU4 \nU3 \nU2 \nU1 \nO \nO1 \nO2 \nO3 \nO4")
    NozzleGeometry = input('Nozzle geometry\n >>>  ')
    print("Now, define the trial number for that geometry\n >>>  ")
    trial = input('Trial number\n >>>  ')

    print('This is the start of the solenoid valve test. The valve should open
    for 1 second and then close for another second.')
    I.time.sleep(1)
    start = 0
    while start != 1:
        Solenoid.SolenoidOPEN()
        I.time.sleep(1)
        Solenoid.SolenoidCLOSE()
        I.time.sleep(1)
        start = int(input('If the test was successful, then press 1 to
   continue.\n >>>  '))

    Solenoid.SolenoidCLOSE()

```

```
52      start = 0
53      while start != 1:
54          InitialCO2Mass = float(input('Input the initial mass of the CO2 (g):\n
        >>>  '))
55          start = int(input('Press 1 to continue.\n >>>  '))
56
57      start = 0
58      while start != 1:
59          start = int(input('Insert CO2, then press 1 to continue.\n >>>  '))
60
61      #PressureSensor.Calibrate()
62
63      start = 0
64      while start != 1:
65          start = int(input('Open main flow valves and close N2 valve, then
        press 1 to continue.\n >>>  '))
66      ForceSensor.set_reference_unit(141846.78588704733)
67  ##    ForceSensor.CalibrateHX711()
68      ForceSensor.reset()
69      ForceSensor.tare(20)
70
71      start = 0
72      while start != 1:
73          testforce = round(ForceSensor.get_weight(1), 2)
74          testpressure = round(PressureSensor.getPressure(), 2)
75          testtempco2 = round(CO2TempSensor.getTemperature(), 2)
76          testtemppipe = round(PipeTempSensor.getTemperature(), 2)
77          print('\n\nForce: {}N\nCO2 Temp {}C\nPipe Temp {}C\nPressure {}psi\n'.
        format(testforce, testtempco2, testtemppipe, round(testpressure /
        6894.757,1)))
78          start = int(input('If all sensors are functioning correctly, press 1
        to continue.\n >>>'))
79      print('Starting in...')
80      for i in COUNTDOWN:
81          print(COUNTDOWN[len(COUNTDOWN)-i-1])
82          I.time.sleep(1)
```

```
83      ForceSensor.reset()
84      ForceSensor.tare(20)
85      ForceTester = 200
86      StartTime = I.time.time()
87      while ForceTester >= 0:
88          Solenoid.SolenoidOPEN()
89          TimeData.append(I.time.time() - StartTime)
90          ForceDataPoint = ForceSensor.get_weight(1)
91          ForceData.append(ForceDataPoint)
92          PressureData.append(PressureSensor.getPressure())
93          CO2TempData.append(CO2TempSensor.getTemperature())
94          PipeTempData.append(PipeTempSensor.getTemperature())
95          if ForceDataPoint < ForceThreshold:
96              ForceTester = ForceTester - 1
97          else:
98              pass
99
100     Solenoid.SolenoidCLOSE()
101
102     file = open('../Data/ExperimentalData/{}_trial_{}.txt'.format(
        NozzleGeometry, trial), 'w')
103     for n in range(len(PressureData)):
104         file.write(str(TimeData[n]) + ',' + str(ForceData[n]) + ',' + str(
        PressureData[n]) + ',' + str(CO2TempData[n]) + ',' + str(PipeTempData[n])
        + '\n')
105     file.close()
106
107     I.time.sleep(5)
108     start = 0
109     while start != 1:
110         FinalCO2Mass = float(input('Input the final mass of the CO2 (g):\n >>>
          '))
111         start = int(input('Press 1 to continue.\n >>>  '))
112
113     ChangeInMass = InitialCO2Mass - FinalCO2Mass
114     PressureSlope = PressureSensor.CalibrationSlope
```

```
115     PressureIntercept = PressureSensor.CalibrationIntercept
116     ForceReferenceUnit = ForceSensor.REFERENCE_UNIT
117 
118     file = open('../Data/ExperimentalData/{}_trial_{}CalibrationValues.txt'.
    format(NozzleGeometry, trial), 'w')
119     file.write('Nozzle Geometry: {}, trial: {}'.format(NozzleGeometry, trial)
    + '\n' + 'CO2 Initial Mass: {}kg'.format(InitialCO2Mass/1000) + '\n' + '
    CO2 Final Mass: {}kg'.format(FinalCO2Mass/1000) + '\n' + 'Change in CO2
    mass: {}'.format(ChangeInMass/1000) + '\n' + 'Pressure Calibration Slope:
    {}'.format(PressureSlope) + '\n' + 'Pressure Calibration Intercept: {}'.
    format(PressureIntercept) + '\n' + 'Force Reference Unit: {}'.format(
    ForceReferenceUnit))
120     file.close()
121     print('Geometry {}, trial {} completed.'.format(NozzleGeometry, trial))
122 except KeyboardInterrupt:
123     print('great job... you made toast')
124 
125 finally:
126     I.RGPIO.cleanup()
```

The second scripting file shown is the analysis class. This holds all of the necessary code for analyzing the data collected from experiment.py. Again, see reference [6] to see all the additional files.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Apr  2 14:01:21 2020

@author: max
"""
#==================Importing Modules=========================================#
import numpy as np
import mpmath as mp
import pandas as pd
import matplotlib.pyplot as plt
plt.style.use('classic')
from CoolProp.CoolProp import PropsSI
#==================Analyzer Class============================================#
class Analyzer:
    #==============This initializes class objects=============================#
    def __init__(self, Geometry, Trial):

        ExitRadii =     [
            5, 5, 4, 4, 3, 3, 2, 2, 1.59323, 1.59323,
            1, 1, .9, .9, .8, .8, .7, .7, .625, .625
                        ]

        self.Geometry = Geometry
        self.Trial = Trial
        self.AreaRatios = []
        self.ForceData = []
        self.TimeData = []
        self.ChamberTempData = []
        self.ActualExitTempData = []
        self.ChamberPressureData = []
```

```
33          self.Masses = []
34          self.Identifiers = []
35          self.ActualTime = []
36          self.g = 9.80665
37          self.R = 8.31446261815324
38          self.W = 0.1178664
39          self.gamma = 1.305
40          self.P_a = 101354.62277
41          self.ThroatRadius = .625e-3
42          self.THRESH = .01
43          self.UNIT = 1e-1
44          self.N = 10
45          self.k = self.gamma - 1
46          self.m = self.gamma + 1
47          self.ThroatArea = np.pi * self.ThroatRadius**2
48          self.ExitRadii = list(reversed(ExitRadii))
49          self.Identity = self.Geometry + str(self.Trial)
50          self.color_value = '#6b8ba4'
51          self.thickness = 1.5
52          for i in self.ExitRadii:
53              self.AreaRatios.append((i*10**(-3))**2 / self.ThroatRadius**2)
54          #=====Extracting intrinsic data=================#
55          columns = np.loadtxt(
56              '../Refined_Data_Files/Calibration_Files/Mass.txt'
57              , delimiter=',', dtype='str')
58
59          for column in columns:
60              self.Identifiers.append(column[0])
61              self.Masses.append(column[1])
62
63          for i in range(0,len(self.Identifiers)):
64              if self.Identifiers[i] == self.Identity:
65                  self.Mass = float(self.Masses[i])
66                  self.ExitArea = np.pi * (self.ExitRadii[i]*10**(-3))**2
67                  self.AreaRatio = self.AreaRatios[i]
68
```

```
69          columns = np.loadtxt('../Refined_Data_Files/Data_Files/{}_trial_{}.txt
    ',
70                               .format(self.Geometry, self.Trial), delimiter=',')
71
72          for column in columns:
73              self.TimeData.append(column[0])
74              self.ForceData.append(column[1])
75              self.ChamberPressureData.append(column[2])
76              self.ChamberTempData.append(column[3] + 273.15)
77              self.ActualExitTempData.append(column[4] + 273.15)
78          #=============================================#
79          #==Giving each geometry its calculated properties==#
80          counter = 0
81          while counter < self.N:
82              for i in range(1, len(self.ChamberPressureData)):
83                  self.ActualTime.append(self.TimeData[i])
84                  if 0 <= self.ChamberPressureData[i] <= self.THRESH:
85                      pass
86                  else:
87                      change =    (
88                                  self.ChamberPressureData[i] /   (
89                                      self.ChamberPressureData[i-1]
90                                                                  )
91                                  )
92
93                      if 1 - self.THRESH <= (
94                              change <= 1 + self.THRESH):
95                          counter = counter + 1
96
97              self.TotalTime = self.TimeData[i]
98
99          self.MassFlowRate = self.Mass / self.TotalTime
100         self.ExitTempData = self.SolveExitTemp()
101         self.PredictedForce = self.PredictedThrust()
102         self.ExperimentalIsp = self.SpecificImpulse()
103         self.PredictedIsp = self.PredictIsp()
```

```
104         #=================================================#
105     #=======Function for performing a trapezoidal sum=========================#
106     def TrapezoidalSum(self):
107         summer = 0
108         for i in range(0,len(self.ActualTime)-1):
109             AreaSection = .5 * (self.ForceData[i] + self.ForceData[i+1]) * (
110                 self.ActualTime[i+1] - self.ActualTime[i])
111             summer = summer + AreaSection
112         return summer
113     #=======Function to calculate the experimental Isp=========================#
114     def SpecificImpulse(self):
115         Isp = self.TrapezoidalSum() / (self.Mass * self.g)
116         return Isp
117     #=======Computational analysis to solve for T_e============================#
118     def SolveExitTemp(self):
119         ExitTemp = []
120         for j in range(0,len(self.ChamberTempData)):
121             T_e = 1
122             T_c = self.ChamberTempData[j]
123             P_c = self.ChamberPressureData[j]
124             Y = self.gamma
125             k = self.k
126             m = self.m
127             w = self.MassFlowRate
128             A_e = self.AreaRatio * self.ThroatArea
129             rho_c = PropsSI('D','T',T_c,'P',P_c ,'CO2')
130 
131             test_values = []
132             counter = 0
133 
134             while counter < self.N:
135                 term_1 = (k / (2*Y*P_c*rho_c))
136                 term_2 = ((1e2*w) / A_e)**2
137                 term_3 = mp.root(T_c / T_e, k/m, k=0)
138 
139                 T_e =   (
```

```
140                         T_c /   (
141                                     term_1 * term_2 * term_3 + 1
142                                 )
143                         )
144
145                 test_values.append(T_e)
146                 counter = 0
147
148                 for i in range(1,len(test_values)):
149                     if 0 <= test_values[i] <= self.THRESH:
150                         pass
151                     elif 1 - self.THRESH <= (
152                             test_values[i] / test_values[i-1]
153                             ) <= 1 + self.THRESH:
154                         counter = counter + 1
155
156             ExitTemp.append(T_e)
157
158         return ExitTemp
159     #=========Function to predict the thrust given geometry's properties=====#
160     def PredictedThrust(self, T=None):
161
162         PredictedForce = []
163         for i in range(0,len(self.TimeData)):
164             if T is None:
165                 T_e = self.ExitTempData[i]
166             else:
167                 T_e = self.ActualExitTempData[i]
168             T_c = self.ChamberTempData[i]
169             P_c = self.ChamberPressureData[i]
170             P_a = self.P_a
171             A_t = self.ThroatArea
172             Y = self.gamma
173             k = self.k
174             m = self.m
175             Q = ((2*Y**2) / k) * (2 / m)**(m/k)
```

```
176             TR = T_e / T_c
177             E = self.AreaRatio
178             if T is None:
179                 NewForce = A_t * P_c *  (
180                     mp.root((Q - Q*TR), 2, k=0) +   (
181                                                 mp.root(TR, k/Y, k=0) - (
182                                                                     P_a/P_c)
183                                                     ) * E
184                                         )
185             else:
186                 NewForce = A_t * P_c *  (
187                 np.sqrt(Q - Q*TR) +   (
188                                             mp.root(TR, k/Y, k=0) - (P_a/P_c)
189                                                 ) * E
190                                     )
191             PredictedForce.append(self.UNIT*NewForce)
192         return PredictedForce
193     #=======First attempt to fix the thrust=================================#
194     def FirstThrustFix(self):
195         PredictedForce = []
196         for i in range(0,len(self.TimeData)):
197             T_e = self.ActualExitTempData[i]
198             T_c = self.ChamberTempData[i]
199             P_c = self.ChamberPressureData[i]
200             P_a = self.P_a
201             A_t = self.ThroatArea
202             Y = self.gamma
203             k = self.k
204             m = self.m
205             Q = ((2*Y**2) / k) * (2 / m)**(m/k)
206             TR = T_e / T_c
207             E = self.AreaRatio
208 
209             NewForce = A_t * P_c *  (
210                 mp.root((np.abs(Q - Q*TR)), 2, k=0) +   (
211                                             mp.root(TR, k/Y, k=0) - (P_a/P_c)
```

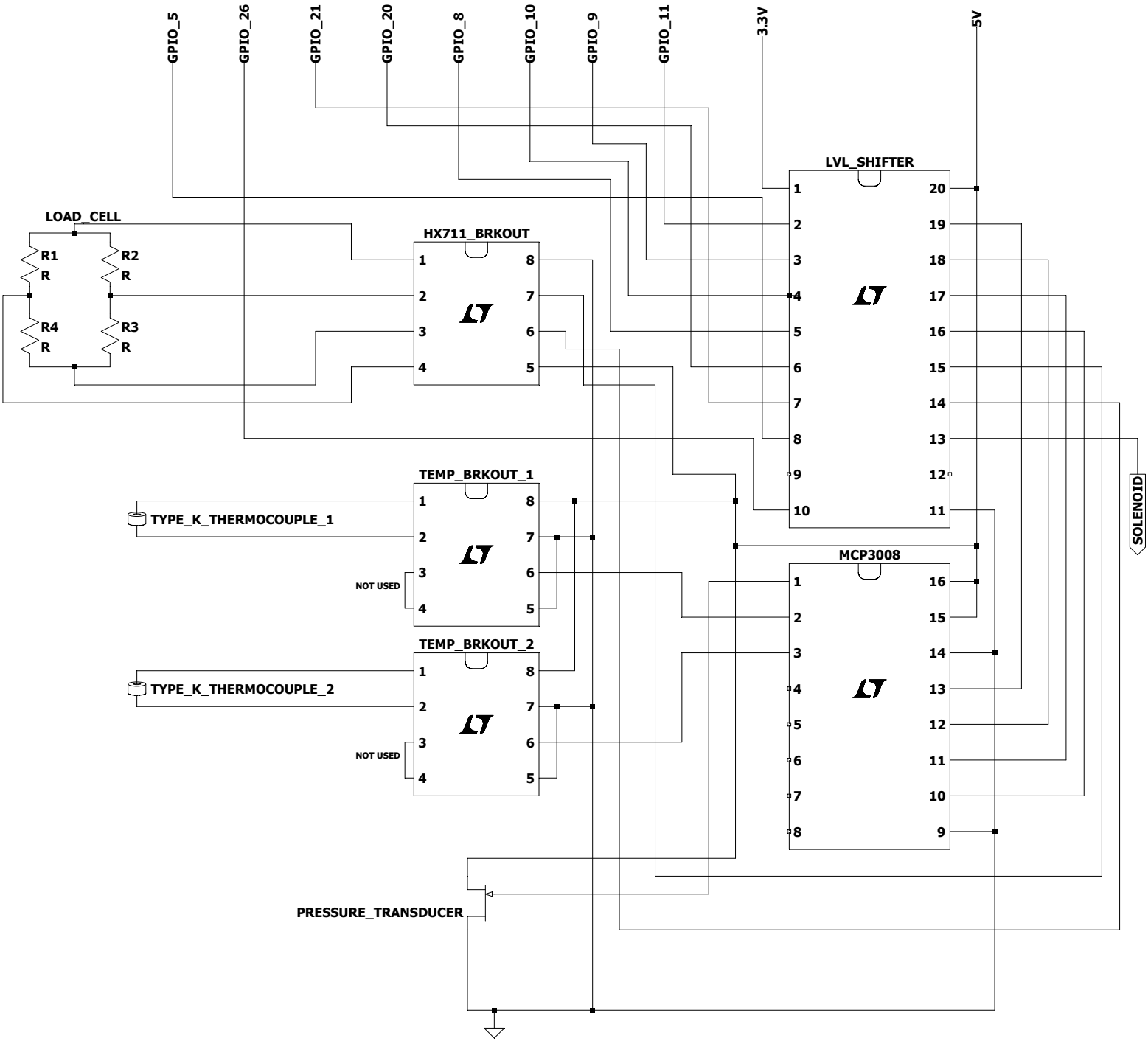
```
                                                    ) * E
                                            )

            PredictedForce.append(self.UNIT*NewForce)
        return PredictedForce
    #=======Function to determine theoretical Isp=========================#
    def PredictIsp(self):
        PredictedIsp = []
        for i in range(0,len(self.TimeData)):
            T_e = self.ExitTempData[i]
            T_c = self.ChamberTempData[i]
            P_c = self.ChamberPressureData[i]
            P_a = self.P_a
            Y = self.gamma
            k = self.k
            W = self.W
            R = self.R
            rho_c = PropsSI('D','T',T_c,'P',P_c ,'CO2')
            self.C_D = np.sqrt((self.gamma * rho_c) / P_c)
            C_D = self.C_D

            SQRT_TERM = np.sqrt(((2*Y*R*T_c) / (k*W)) * (1 - (T_e/T_c)))

            NewIsp = (self.AreaRatio / C_D) * (SQRT_TERM + (
                (T_e/T_c)**(Y/k) - (P_a/P_c)))

            PredictedIsp.append(NewIsp)
        return PredictedIsp
    #=========An attempt to resolve poor data to calculate Isp============#
    def ActualIsp(self):
        PointIsp = []
        for i in range(0,len(self.TimeData)):
            PointIsp.append(self.ForceData[i] / self.MassFlowRate)

        return PointIsp
    #=========Thrust curve plotter=========================================#
```

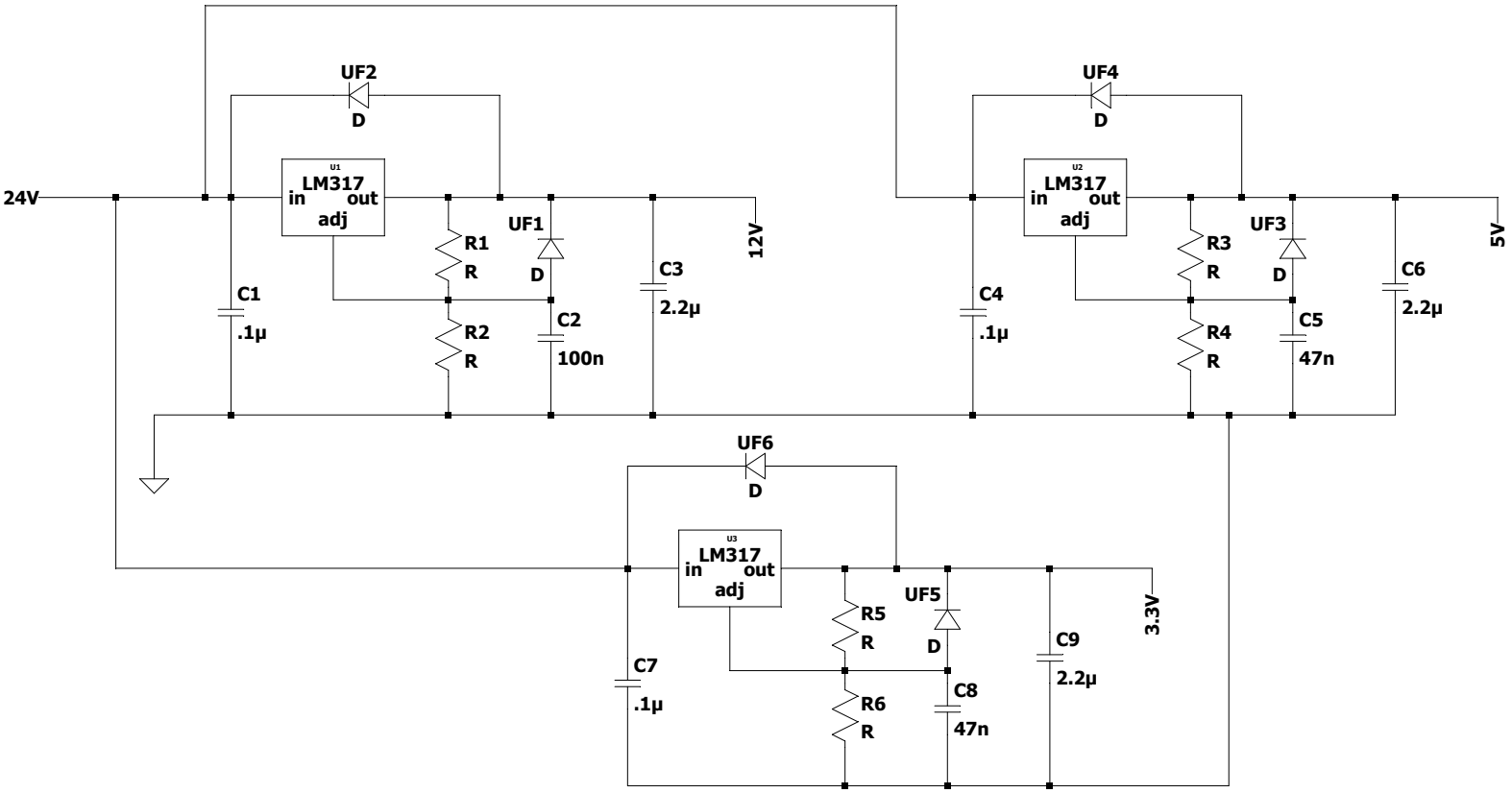
```
248     def PlotThrust(self):
249 
250         size = 20
251         size_config = .8
252         fig = plt.figure(1, figsize=(10,7))
253         fig.suptitle('${}\ Thrust\ Curves$'.format(self.Identity),
254                      fontsize=size)
255         plt.xlim(-1,12)
256         if max(self.PredictedForce) > max(self.ForceData):
257             ymax = (max(self.PredictedForce))
258         else:
259             ymax = (max(self.ForceData))
260         plt.ylim(-round(.05*ymax, 2), round(1.05 * ymax, 2))
261         plt.xlabel('$Time\ (s)$', fontsize=size_config*size)
262         plt.ylabel('$Force\ (N)$', fontsize=size_config*size)
263         plt.plot(self.TimeData, self.PredictedForce,
264                  label='$Theoretical$', lw=self.thickness,
265                  linestyle='dotted', color='black')
266         plt.plot(self.TimeData, self.ForceData, label='$Emperical$',
267                  color=self.color_value)
268         plt.legend()
269         plt.show()
270     #=========Smoothing function for noisy datasets=========================#
271     def Smoother(self, DataSet, N):
272         return pd.Series(DataSet).rolling(window=N).mean().iloc[N-1:].values
273     #=========Generic plotter that makes them somewhat pretty===============#
274     def PlotAnyDataSet(self, DataSet1, Variable1, Unit, path,
275                        DataSet2=None, Variable2=None, x_data=None):
276         if x_data is None:
277             x_data = self.TimeData
278         else:
279             x_data = np.linspace(-1, 12, len(DataSet1))
280 
281         size = 20
282         size_config = .8
283         fig = plt.figure(1, figsize=(11,4))
```

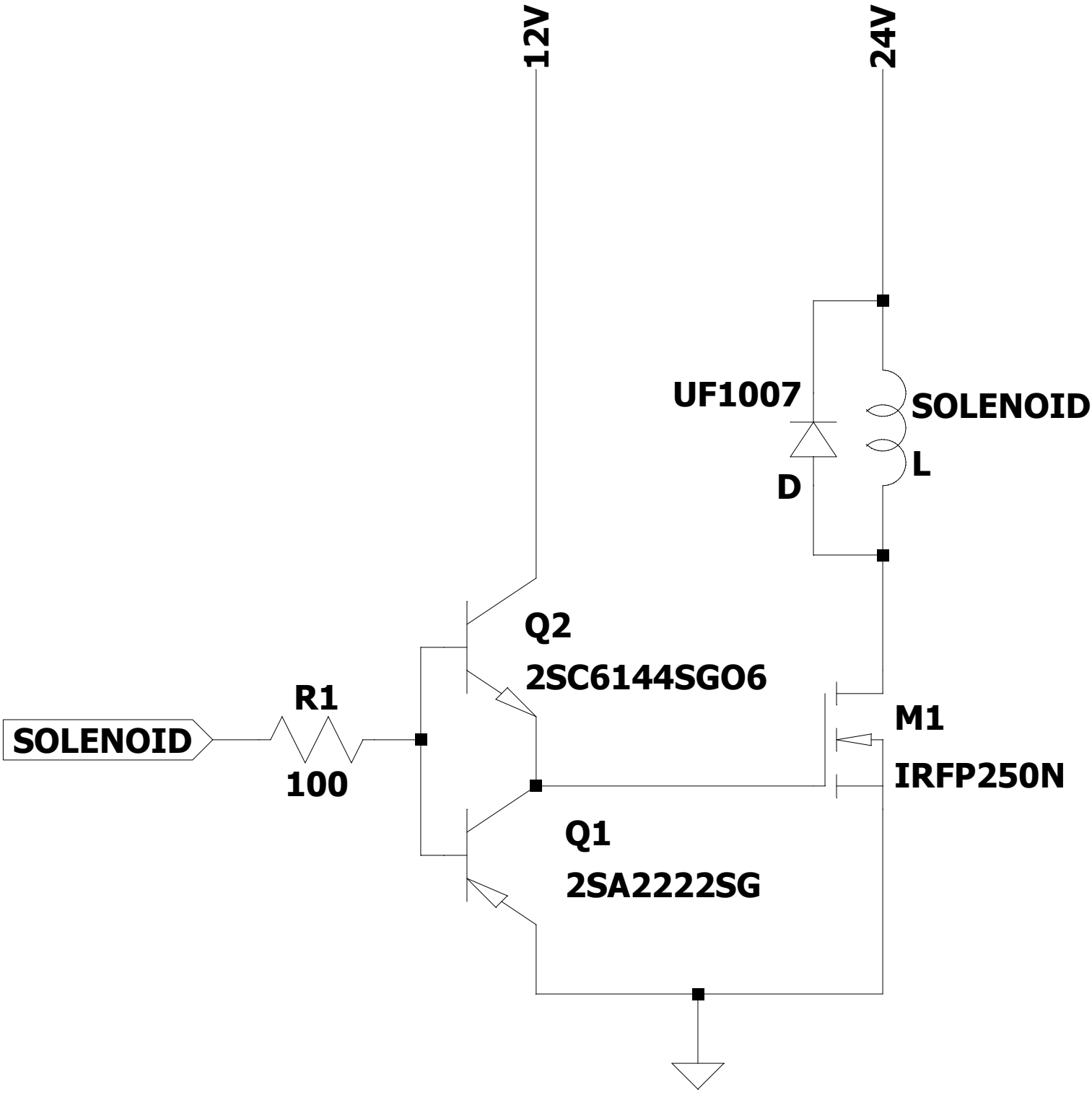
```
284         plt.xlim(-1,12)
285         plt.xlabel('$Time\ (s)$', fontsize=size_config*size)
286 
287         if Variable1 == 'Temperature':
288             plt.ylabel('$Temperature\ ({})$'.format(Unit),
289                        fontsize=size_config*size)
290         else:
291             plt.ylabel('${}\ ({})$'.format(Variable1, Unit),
292                        fontsize=size_config*size)
293 
294         if DataSet2 is not None:
295             fig.suptitle('${}\ {}\ and\ {}\ Data$'.format(
296                 self.Identity, Variable1, Variable2), fontsize=size)
297 
298             plt.plot(x_data, DataSet1, color='black', lw=self.thickness,
299                      label='${}\ {}\ Data$'.format(
300                          self.Identity, Variable1))
301             plt.plot(x_data, DataSet2, color=self.color_value,
302                      lw=self.thickness, label='${}\ {}\ Data$'.format(
303                          self.Identity, Variable2))
304         else:
305             fig.suptitle('${}\ {}\ Data$'.format(
306                 self.Identity, Variable1), fontsize=size)
307 
308             plt.plot(x_data, DataSet1, color=self.color_value,
309                      lw=self.thickness, label='${}\ {}\ Data$'.format(
310                          self.Identity, Variable1))
311         plt.legend()
312         plt.savefig('{}'.format(path), bbox_inches = "tight")
313         plt.close()
314     #=============================================================#
315 
316 #=================================================================#
```

A.2 Schematics

Here, the most relevant circuit schematics are provided. Shown below is the entire circuit schematic used for interfacing the Raspberry Pi with the sensors, the power supply handling, and the push pull amplifier for the solenoid respectively.


GPIO_5
GPIO_26
GPIO_21
GPIO_20
GPIO_8
GPIO_10
GPIO_9
GPIO_11
3.3V
5V
LVL_SHIFTER
LOAD_CELL
R1
R
R2
R
R4
R
R3
R
HX711_BRKOUT
TEMP_BRKOUT_1
TYPE_K_THERMOCOUPLE_1
NOT USED
TEMP_BRKOUT_2
TYPE_K_THERMOCOUPLE_2
NOT USED
MCP3008
SOLENOID
PRESSURE_TRANSDUCER


24V
UF2
D
U1
LM317
in
out
adj
R1
R
UF1
D
C1
.1µ
R2
R
C2
100n
C3
2.2µ
12V
UF4
D
U2
LM317
in
out
adj
R3
R
UF3
D
C4
.1µ
R4
R
C5
47n
C6
2.2µ
5V
UF6
D
U3
LM317
in
out
adj
R5
R
UF5
D
C7
.1µ
R6
R
C8
47n
C9
2.2µ
3.3V


12V
24V
UF1007
SOLENOID
D
L
Q2
2SC6144SGO6
R1
SOLENOID
100
M1
IRFP250N
Q1
2SA2222SG

A.3 Parts List

Here a parts list is included of the purchased equipment for the project.

Seller	Item	Description	Price per Unit (USD)	Quantity	Subtotals (USD)
McMaster-Carr	7833K73	High-Pressure Precision Flow-Adjustment Valve	28.73	2	57.46
McMaster-Carr	5220K158	Inline Tee for 1/4" Copper Tube OD x 1/8 NPTF Male	10.00	1	10.00
McMaster-Carr	8955K112	General Purpose Copper Tubing	28.29	1	57.72
McMaster-Carr	5272K299	Straight Adapter for 1/4" Tube OD x 1/8 NPT Male	4.81	12	28.29
McMaster-Carr	5485K21	Straight Connector, 1/8 NPT Male	1.40	1	1.40
McMaster-Carr	50785K26	Reducing Adapter, 1/4 NPT Female x 1/8 NPT Male	1.58	1	1.58
McMaster-Carr	50785K75	Tee Connector, 1/8 NPT Female	3.73	1	3.73
McMaster-Carr	1190N21	Stainless Steel, 24V DC, 1/8 NPT Female, 3000 Max PSI	138.97	3	416.91
Amazon	B07K2JQ6CG	30 Packs x 16g Threaded CO2 Cartridges	25.90	1	25.90
Amazon	B07PFY54TV	25g Threaded CO2 Cartridges, 6 Pack	24.99	1	24.99
Amazon	B07DN3L9N2	G 1/4" Male Thread x 1/4" NPT Female	14.99	1	14.99
Amazon	B07GJGTWQT	G1/4" Pressure Transducer Sensor (0-1000psi)	23.19	1	23.19
Amazon	B07YGN2QGL	UV Resin Curing Light for SLA/DLP 3D Printer	25.99	1	25.99
Amazon	B07KSYRW34	Strong and Precise High Resolution 3D Printing Resin	49.95	2	49.95
Amazon	B06XWVZHZJ	8 Channel Logic Level Converter Bi-Directional High	7.99	1	7.99
Amazon	B07K2ZHMRF	ELEGOO Mars UV Photocuring LCD 3D Printer	229.99	1	229.99
				Subtotal:	1030.03