University of Colorado - Boulder

ASEN 4013

Rocket Demonstration Project

Transient Propellant Grain Burnback

Author: Connor O'Reilly[a]

[a]107054811

Professor:
James Nabity

April 29, 2020

College of Engineering & Applied Science
UNIVERSITY OF COLORADO BOULDER

Contents

I. Introduction

The Aerotech RMS-29/60-F62T Rocket motor is a commercial off the shelf high power rocket motor. It uses Blue Thunder^{TM}, and the exact composition is unknown to the public. In this project, numerical and analytical methods are used to predict the performance of the F26T motor. Using the available characteristics of BlueThunder^{TM} and the provided thrust curve, a theoretical model was produced to investigate different aspects of a solid propellant propulsion system. Propellant composition, grain burn back models, and other methods were used to try and accurately model the available vendor data. For the numerical portion NASA Chemical Equilibrium Analysis is used to determine thermodynamic and transport properties for the estimated propellant mixture. The analytical portion uses MATLAB to iteratively evaluate derived equations for the calculation of a thrust curve for the theoretical model. This demonstration shows even without proprietary information or a physical motor at hand, the performance of a real rocket motor can be modeled and determined.

II. Problem Statement

A. Sketch

Below is a sketch for the Solid-Propellant rocket engine. It includes some important Dimensions and State variables.

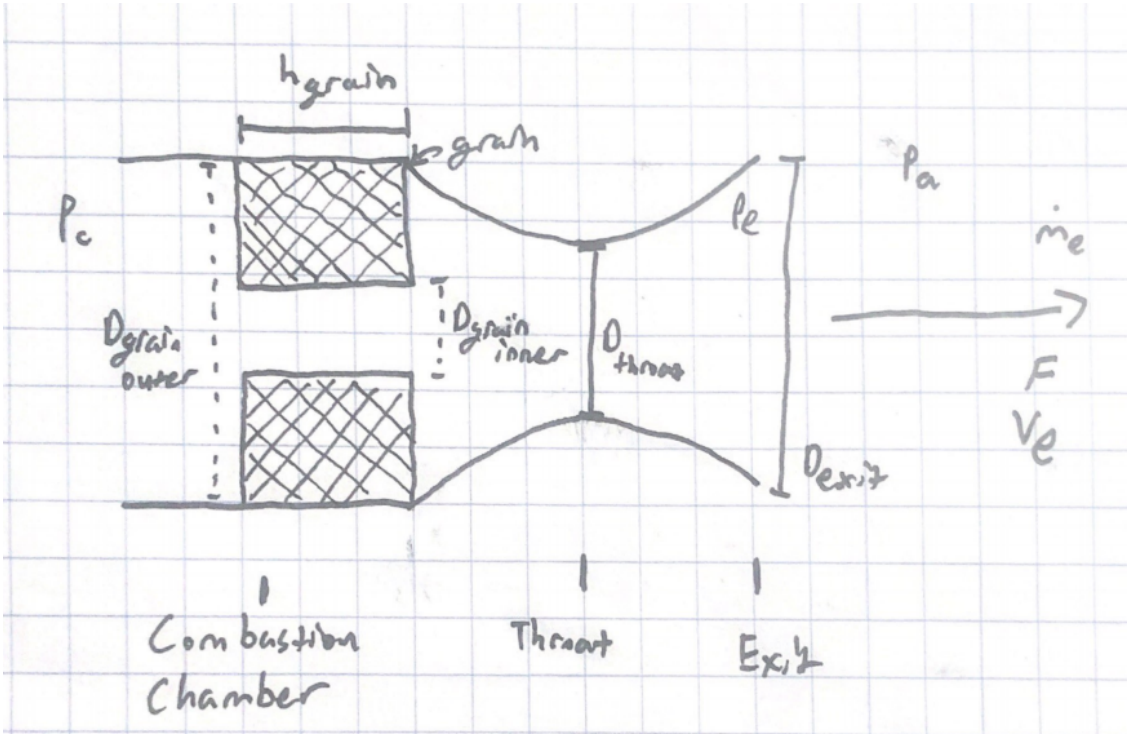


Figure 1: Simplified sketch of Rocket Problem

B. Given Information

Using the provided SolidWorks assembly drawing of the RMS-29/60 rocket motor the following dimensions were determined

- D_{exit}= 0.231 [inches]
- D_{throat}= 0.123 [inches]

Grain dimensions were given with

- $D_{grain,outer}$= 0.908 [inches]
- $D_{grain,inner}$= 0.177 [inches]
- h_{grain}= 1.505 [inches]

The Ambient pressure for the Aerotech testing at sea-level and the testing in Boulder, CO were provided with

- $P_{sea-level}$= 101,325 [Pa]
- $P_{Boulder}$= 83,491 [Pa]

A Thrust-Time Profile for the F62T Motor with Blue Thunder™ propellant was provided and digitized using the digitizer found at the following address. `https://automeris.io/WebPlotDigitizer/`

Lastly the Burn Rate Exponent and Coefficient were provided in the Vendor's Blue Thunder™ Propellant data

- n= 0.321
- a= 0.047

III. Method

A. Governing Principles, Governing Equations and Simplifying Assumptions

1. *Governing Principles*

Shown below are some of the Governing Principles that were utilized during the analysis of this experiment.

- Conservation of mass, momentum and energy
- 1^{st} law of thermodynamics - Energy can be transformed from one form to another, but can be neither created nor destroyed
- 2^{nd} law of thermodynamics – entropy increases (for irreversible processes); isentropic relationships can be used to determine state properties at any station even for an irreversible process
- Equations of Motion (Newton's Laws)
- Physics - Gas Laws
- Thermodynamics - Thermodynamically favored chemical reactions occur and the final composition minimizes Gibbs free energy (favors equilibrium combustion)
- Isentropic relationships - used to determine state properties at any station and not neccessarily for a process

2. *Governing Equations*

Shown below are some of the Governing Equations that were utilized for Thrust Prediction, and Thrust Prediction Analysis.

- Ideal Thrust Equation
- Burning Rate Equation
- Mass Flow Rate of Propellant Equation
- Characteristic Velocity Equation, C^*
- Nozzle exit velocity, V_e
- Total and Specific Impulse Equations
- Chamber Pressure and Stability Equations

3. Simplifying Assumptions

- Isentropic from the chamber through the nozzle.
- Complete Combustion (No Grain remains)

B. Equations and Code Employed

1. Equations Employed

- Lumped parameter equation to calculate chamber pressure

$$p_c = \left[\frac{a \cdot \rho_p \cdot A_{burn} \cdot c*}{A_{throat}}\right]^{\frac{1}{1-n}} \tag{1}$$

- St. Robert's equation

$$\dot{r}_b = aP_c^n \tag{2}$$

- Equation to determine surface area of a hollow cylinder with outer radius r_1, internal radius r_2 and height h.

$$SA = 2\pi h(r_1 + r_2) + 2\pi(r_1^2 - r_2^2) \tag{3}$$

- Mass Flow rate of Propellant gas

$$\dot{m} = (\rho_{propellant,solid} - \rho_{propellant,gas}) * r_b * A_{burn} \tag{4}$$

- Ideal Thrust Equation

$$F_i = \frac{\dot{m}_e V_e}{g_c} + (P_e - P_a)A_e \tag{5}$$

- Mach Number for Isentropic Flow

$$M_e = \frac{V_e}{a} \tag{6}$$

- Total Impulse of a Rocket

$$I_t = \int_0^{t_b} F dt \tag{7}$$

- Average Specific Impulse for a Rocket

$$\overline{I_{sp}} = \frac{I_t}{w_{propellant}} \tag{8}$$

- Average effective exhaust velocity

$$\overline{C} = \overline{I_{sp}} \cdot g_0 \tag{9}$$

2. Code Employed

transient.m is the main script for the analysis. All inputs for calculations are initialized, and the digitized data provided from the vendor will be downloaded. Then utilizing *burn_geometry.m* and *thrust_calc.m* the theoretical thrust will be calculated until no grain remains. Afterwards, the predicted results will be plotted against the vendor provided data and motor performance values are determined.
burn_geometry.m calculates the burn area for a cylindrical propellant grain with propellant grain height, propellant grain radius, propellant linear ablation as inputs. The outputs which are the current burn area and current burn cavity volume.
thrust_calc.m applies the simplified momentum conservation integral to the rocket engine to calculate thrust and the characteristic velocity for a given time.
CEAinput.m defines a class to call NASA CEA from within a MATLAB project.
RUN_CEA.m will call NASA CEA within *thrust_calc.m*, it takes inputs of chamber pressure, supersonic area ratio and will return a structure containing sonic velocity, Mach number, nozzle exit pressure, and C^*.

C. Solution Procedure

Initially the Vendor Provided Thrust Curve will be digitized and downloaded in the *transient.m* script. Following variables needed for computation will be initialized in the *INPUTS* sections of the main script. Some variables will be initially set to estimates these, include the c-star efficiency, time step, propellant composition and characteristic velocity. Variables for grain rate, grain displacement and time will be initially set to zero. An equation to model the burn area is derived in terms of grain radius, grain height and burn displacement. The final equation used is shown below and explained further in the result section.

$$A_{B3} = 2\pi(r^2_{grain,outer} - (r_{grain,inner} + r_b)^2) + 2\pi(h_{grain} - 2r_b)(r_{grain,inner} + r_b) \tag{10}$$

After the burn area for each time step is calculated the chamber pressure and burn rate will be computed using the Lumped Parameter equation 1 and Saint Roberts Equation 2 respectively. Burn displacement is then determined iterativly by the following equation.

$$r_n = r_{n-1} + \dot{r}_b * \Delta t$$

Equations to determine propellant mass flow and Ideal Thrust are derived and implemented in the *thrust_calc.m* MATLAB script using outputs from *RUN_CEA.m*.

$$F_{model} = \dot{m}_e(M_e * a) + (P_e - P_a)A_e$$

$$\dot{m} = (\rho_{propellant,solid} - \rho_{propellant,gas}) * r_b * A_{burn}$$

The time step will increase and computations will continue until no propellant grain remains. After the model is complete, adjustments to input parameters are investigated and implemented to accurately fit the theoretical thrust curve to the vendor thrust curve. performance characteristic for both the model and claimed curves are calculated and compared.

IV. Results

A. Digitizing Vendor Data

The provided F26T thrust curve in the assignment document was digitized and downloaded into a file named "Thrust.csv". Using MATLAB, the Data was plotted and is shown in the figure below.

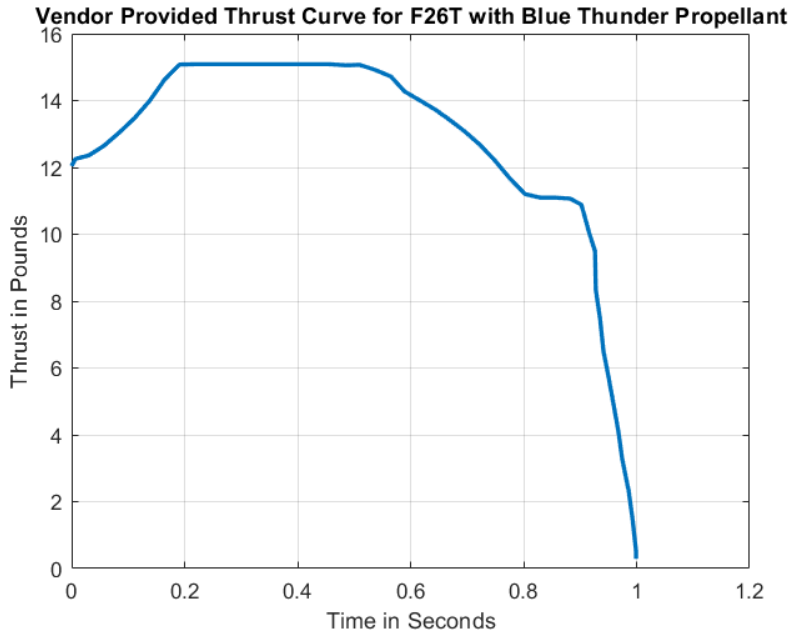


Figure 2: Digitized F26T from Vendor

Using equation 7, the total impulse for the claimed thrust curve was computed. This was completed by finding the area under the thrust curve using the MATLAB *trapz* function. These Values are shown below.

	Total Impulse [N-sec]
Vendor Claimed	50
Thrust.csv	57.873

The value computed is similar and has the same order of magnitude as the claimed total impulse. One major factor causing variance from the claimed value can be from the digitizing process. The Thrust Curve was digitized multiple times causing the predicted total impulse to vary ±1 [N-sec].

B. Defining Equations for Burn Area

Due to the motor having a Cylindrical perforated grain the burn area was modeled after equation 3. The grain will burn from the interior surface of the tube but it is unknown how the grain will burn at both ends of the cylinder. Equations to compute burn area were derived in terms of grain radius, grain height, and burn displacement for three cases. The first case assumes that the grain does not burn at either surface, the case two assumes the grain will burn at only one end of the cylinder, and lastly the third case assumes the grain burns at both ends. These equations are shown below with the subscript specifying the case.

$$A_{B1} = 2\pi(r_{grain,outer}^2 - (r_{grain,inner} + r_b)^2) + 2\pi(h_{grain})(r_{grain,inner} + r_b) \tag{11}$$

$$A_{B2} = 2\pi(r_{grain,outer}^2 - (r_{grain,inner} + r_b)^2) + 2\pi(h_{grain} - r_b)(r_{grain,inner} + r_b) \tag{12}$$

$$A_{B3} = 2\pi(r_{grain,outer}^2 - (r_{grain,inner} + r_b)^2) + 2\pi(h_{grain} - 2r_b)(r_{grain,inner} + r_b) \tag{13}$$

The first section of each equation describes the surface burn of the grain while the second section describes the interior burn of the grain. Computation of Burn Area is located in the $burn_geometry.m$ MATLAB script.

C. Propellant Mass Flow and Motor Thrust

Propellant Mass flow rate for the model was computed using equation 4 which is shown below and implemented in the *thrust_calc.m* MATLAB script.

$$\dot{m} = (\rho_{propellant,solid} - \rho_{propellant,gas}) * r_b * A_{burn}$$

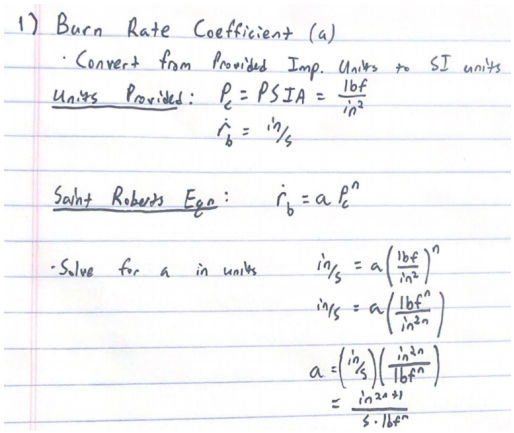
Motor Thrust for the theoretical model was computed using equation 5. With the assumption that flow through the nozzle is Isentropic, the exit velocity can be found using Isentropic Relations and output values from *RUNCEA.m*. The relation and substitution are shown below and the equations are implemented in *thrust_calc.m*.

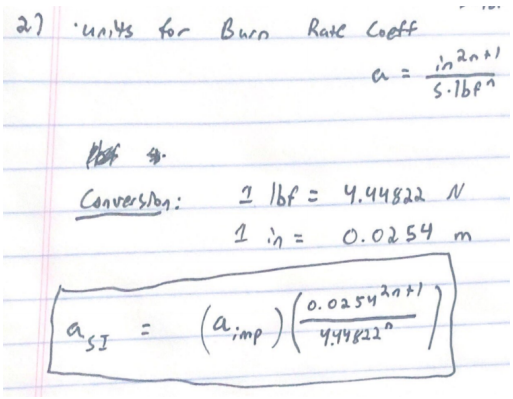
$$V_e = M_e * a$$

$$F_{model} = \dot{m}_e(M_e * a) + (P_e - P_a)A_e$$

D. Adjustment of Input Parameters

The first input parameters that were investigated were the burn rate coefficient and burn rate exponent. Both these values are determined experimentally therefore they should not change from the vendor provided values. The only adjustment made was to convert the burn rate coefficient from the provided Imperial Units to SI units. This conversion was obtained using St. Robert's equation and is shown below.





The burn rate coefficient was calculated to be $6.9947 * 10^5$. To determine the propellant composition, the provided description for the visual effect was useful. As the Blue Thunder^{TM} burns, it produces minimal smoke. This is a widely known characteristic of using Ammonium perchlorate as an oxidizer. In addition to the minimal smoke, Blue Thunder^{TM} also produces a violet-blue color when burning. Looking at different metal additives for rocket fuel it is noticed that Aluminum [Al] burns at high temperatures with an "electric arc, light blue" color. Comparing the provided Chamber Temperature of 2616.532 K and the Combustion temperatures of Aluminum mixed with an oxidizer which range around 2500 K to 3000 k, it can be assumed that a combination of aluminum and HTPB are used as fuel. Now to determine the composition of the propellant, using the provided chamber temperature of 2616.532 and molecular weight of 22.959, this was compared to corresponding values on the Typical Propellant Compositions figure. It was determined that the oxidizer should consist of 78% to 66% of Ammonium perchlorate, 18% of organic polymer binder and 4% to 20% of aluminum. After running multiple tests with different percentages of Al and NH4CL04, it had little affect on the total impulse of the model. Instead the average characteristic velocity was used for comparison. The proposed composition is shown below. With a C* efficiency of 75% the average calculated characteristic velocity was 2692.750 $\frac{m}{s}$- which is close to the vendor value of 2314.930 $\frac{m}{s}$.

	Chemical Formula	weight by mass
Primary Oxidizer	NH4CLO4(I)	72
Binder	C4H6,(HTPB)	10
Additive	AL(cr)	18

Table 1: Propellant Composition

As stated earlier, it is unknown how the grain will burn at the ends of the cylinder. To determine which case to model the burn area with, theoretical thrust curves for all cases were compared to the vendor supplied curve and are shown below.

Figure 3: Case 1

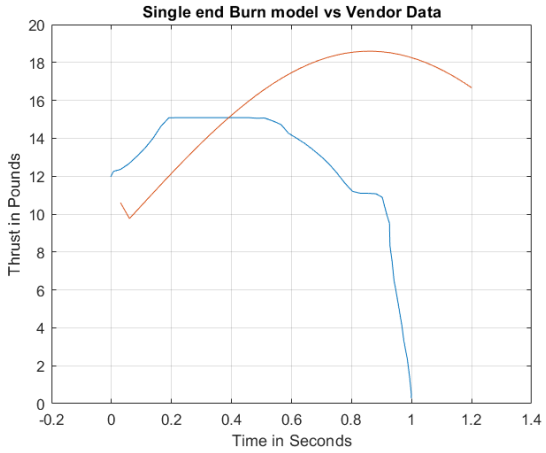


Figure 4: Case 2

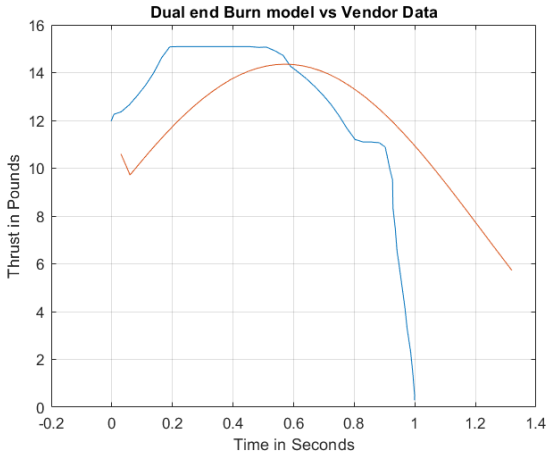


Figure 5: Case 3

Without computations, it can be seen that the burn back model accounting for surface burn at both ends of the cylinder best models the vendor provided data. Below is the final adjusted Thrust Curve with the Vendor provided curve followed by performance characteristics.

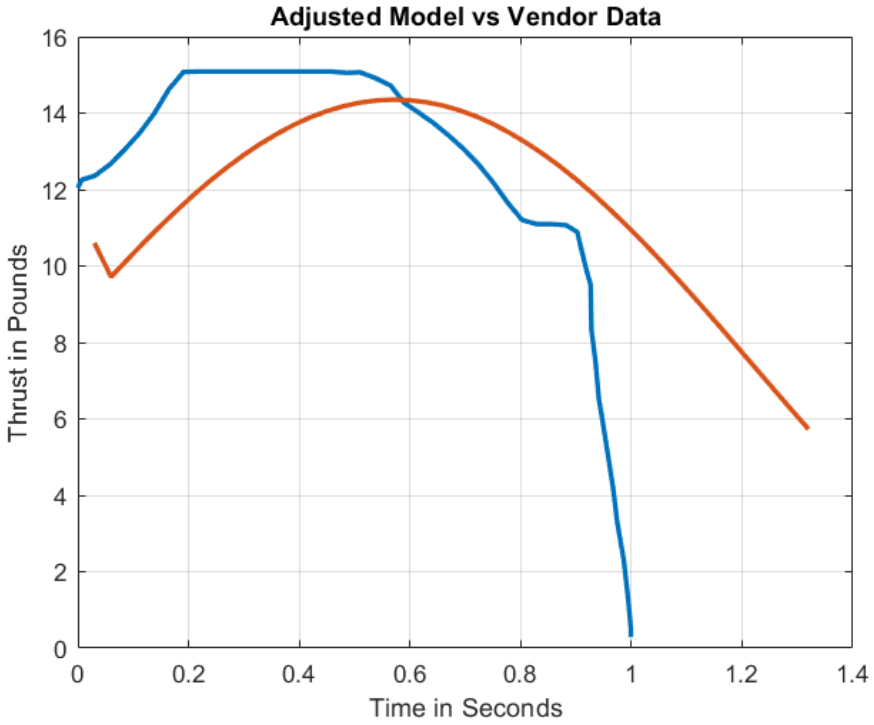


Figure 6: Adjusted vs Vendor Curve

	Maximum Thrust [N]	Total Impulse [N-s]	Avg. Isp [sec]	Avg. C* [$\frac{m}{s}$]	Action time [sec]
Model	14.353	67.3188	274.490	2692.750	0.9632
Digitized	15.088	57.873	235.977	2314.930	0.9932

Table 2: Analysis of model and vendor motor performance

All values computed using the theoretical model have the same order of magnitude are close to the values from the vendor data.

E. Comparison

The shape of the vendor provided thrust curve surprised me. This is due to the fact that the F26T rocket motor has a cylindrical perforated grain, the thrust curve should be progressive with thrust increasing until no grain remains. Instead, the curve shows a constant thrust from about 0.2 sec to around 0.5 sec which is usually associated with a Rod and Tube solid propellant. In addition, there is a section of steadily decreasing thrust from 0.6 to 0.8 seconds followed by another constant thrust section from around 0.8 to 0.9 seconds. In an ideal situation, once no grain remains the thrust curve should decrease to zero thrust almost immediately. This trend, could be accounted for from machining errors in the production of the motor, production in the grain or other real world irregularities; which an ideal rocket motor would not experience. The produced thrust curve is parabolic, although, the thrust is progressive from 0.1 to 0.5 seconds; once maximum thrust is reached the curve begins a regressive trend.. For a cylindrical grain as stated before the thrust curve should drop off once max thrust is reached which is not shown in any model.

V. Discussion of Results

A. Validity of Given Information and Assumptions

A major concern with the provided information is the shape of the grain used. As stated earlier the vendor claimed thrust curve does not resemble a thrust curve using cylindrical perforated grain. This may be the reason why the theoretical model does not accurately describe the Vendor provided data. In addition the Theoretical model has a longer burn time when compared to the vendor provided data. This may cause some concern in the provided data for grain dimensions or it could also be caused by the theoretical model being ideal.

B. Error Associtated with Functions and Measurements

Previously stated, there is some precision error introduced when digitizing the vendor provided thrust curve - if quantified the below equation can be used. Multiple Digitized data files were used and it was determined the error was small and had little affect on calculated values. Lastly, the error associated with *thrust_calc.m* is can be associated with the accuracy of the CEA code and the accuracy in determining the propellant composition. Thrust Calc relies heavily on of *Run_CEA.m* as well as *CEAinput.m* and the calculated burn displacement to compute the predicted thrust for the model. The uncertainty in the predicted thrust and predicted mass flow is shown below.

$$\delta \dot{m}_p = \sqrt{(\frac{\partial \dot{m}_p}{\partial rho_g}\alpha)^2} \tag{14}$$

$$\delta T_{model} = \sqrt{(\frac{\partial T_{model}}{\partial \alpha}\alpha)^2 + (\frac{\partial T_{model}}{\partial M_e}M_e)^2 + (\frac{\partial T_{model}}{\partial P_e}P_e)^2 + (\frac{\partial T_{model}}{\partial \dot{m}_p}\dot{m}_p)^2} \tag{15}$$

VI. Concluding Remarks

A. Major Findings

Although the theoretical model does not visually represent the vendor provided thrust curve accurately, all performance values calculated are within 85% of the provided data. But this must be taken cautiously due to the fact that the theoretical model still has grain remaining after 1.0 sec. If the model continued after 1 sec thrust would still be produced, causing the performance values calculated earlier to increase. The burn back model had a greater affect on the ideal thrust produced than initially expected. But in retrospect it makes sense, if the propellant grain does not burn at the ends the grain will last longer, causing the motor to produce thrust longer.

B. Lessons Learned

The major lesson learned is that Rocket science is hard and projects should be started earlier. In addition theoretical models can predict the performance of rockets quite well. This can help in determining propellant characteristics for real world applications.

C. Future Work

As shown in figure 6 the Theoretical model does not do a great job of modeling the true performance of the rocket. For future work, the burn back model could be investigated further. Investigating different shapes of grain that represent the vendor curve better like rod and tube or star shaped grain.

VII. Acknowledgements

Thank you professor Nabity help and support throughout the project and everything and everything else in this crazy time.

VIII. References

Mattingly, Jack D., and Keith M. Boyer *Elements of Propulsion, Gas Turbines and Rockets*. 2nd ed. AIAA Education Series, AIAA, New York, 2006, cahpter 10.

Nabity, James "ASEN 4013: Foundations of Propulsion Rocket Demonstration Project", Aerospace Dept., Univ. Colorado Boulder, 17 March 2020.

Nabity, James "ASEN 4013: Foundations of Propulsion, Lecture 15: Chemical Rockets", Aerospace Dept., Univ. Colorado Boulder, 17 March 2020.

Nabity, James "ASEN 4013: Foundations of Propulsion, Lecture 14: Overview of Rocket Propulsion", Aerospace Dept., Univ. Colorado Boulder, 19 March 2020.

"Flame test" Wikipedia, Wikimedia Foundation, 1 April 2020, *https* : *//en.wikipedia.org/wiki/Flame*$_t$*est*.

IX. Appendices

A. Matlab Code

1. *transient.m*

```
close all
clear all
clc

t(1) = 0; % [s] initial time
rb(1) = 0; % [m] initial burn grain displacement

%% Digitized Data
T_ven = csvread('Thrust.csv');
T_ven(:,2) = T_ven(:,2);
%Plot Digitized Vendor Data
figure('Name','Digitized Data');
plot(T_ven(:,1),T_ven(:,2),'LineWidth',2);
hold on
xlabel('Time in Seconds');
ylabel('Thrust in Pounds');
title('Vendor Provided Thrust Curve for F26T with Blue Thunder Propellant');
grid on ;
xlim([0 1.2])
ylim([0 16])


%% INPUTS
cstar_eff = .75; % [-], cstar efficiency
t_step = .03; % [s] time step
P_atm = 101325; % [Pa] ambient pressure
a = 0.000069947; % [-] burn rate coefficient
n = 0.321; % [-] burn rate exponent
cstar = 1500; % [m/s] characteristic velocity
h_grain = 1.505; % [in] motor grain height
r_grain_inner = 0.177/2; % [in] motor grain radius
r_grain_outer = 0.908/2;
r_throat = 0.123/2; % [in] throat radius
r_exit = 0.231/2; % [in] exit radius
Mass = 0.025 ; % [kg] Propellant mass

```

```
%% CONVERSIONS
h_grain = h_grain*0.0254; % [m] motor grain height
r_grain_inner = r_grain_inner*0.0254; % [m] motor grain inner radius
r_grain_outer = r_grain_outer*0.0254; % [m] motor grain outer radius
r_throat = r_throat*0.0254; % [m] throat radius
r_exit = r_exit*0.0254; % [m] exit radius

%% QUANTITY CALCULATIONS
Vol = h_grain * ( (r_grain_outer)^2 - (r_grain_inner)^2 ) * pi(); % [m^3]
rho_p = Mass/Vol; % [kg/m^3]
A_throat = pi()*(r_throat)^2; % [m^2]
A_exit = pi()*(r_exit)^2; % [m^2]
AR_sup = A_exit/A_throat; % supersonic area ratio

V_burn = 0; % [m^3]
j = 1;
while rb < r_grain_outer && rb < h_grain % while there is unburned grain
    remaining
    [A_burn(j)] = burn_geometry(r_grain_inner,r_grain_outer, h_grain, rb); % [
        m] burn area, burn cavity volume
    Pc(j) = ((a * rho_p * A_burn(j) * cstar) / (A_throat)).^((1)/(1-n))/1e6; %
         [MPa] chamber pressure
    burn_rate(j) = a*(Pc(j)*10^6)^n; % [m/s] burn rate
    rb = rb + burn_rate(j) * t_step; % [m] updates burn displacement
    burn_rate(j)
    % delta_Vol = ; % [m^3/s] rate of change in burn cavity volume
    [T_predicted(j),cstar] = thrust_calc(P_atm, Pc(j), A_exit, rho_p,
        burn_rate(j), A_burn(j), AR_sup);
    cstar = cstar*cstar_eff; % [m/s]
    %action time
    if j == 1
        t(j) = t_step;
    else
        t(j) = t(j-1) + t_step;
    end
    j = j+1;
end
%Plot Digitized Vendor Data
figure('Name','Digitized Data');
plot(T_ven(:,1),T_ven(:,2),'LineWidth',2);
hold on
plot(t,T_predicted,'LineWidth',2)
xlabel('Time in Seconds');
ylabel('Thrust in Pounds');
xlim([0 1.4])
title('Adjusted Model vs Vendor Data');
grid on ;


%% Performance computations

%maximum thrust calculation
max_t_ven = max(T_ven(:,2));
max_t_mod = max(T_predicted);
%total impulse from digitized data
```

```
I_ven_tot = trapz(T_ven(:,1),T_ven(:,2));
%Convert to SI units (lb-s to N-s)
I_ven_tot = I_ven_tot * 4.44822162;
%total impulse from model
I_mod_tot = trapz(t,T_predicted);
%Convert to SI units (lb-s to N-s)
I_mod_tot = I_mod_tot * 4.44822162;

%avg spec imp
isp_av_ven = I_ven_tot/(Mass * 9.81);
isp_av_mod = I_mod_tot/(Mass * 9.81);
%avg effective velocity
avg_cstar_ven = isp_av_ven*9.81;
avg_cstar_mod = isp_av_mod*9.81;
%action time
% 10% max T
T_10_ven = max_t_ven * .1;
%ven
for i = 1:length(T_ven)
    if(T_ven(i,2) <= T_10_ven)
       time_fin = T_ven(i,1);
       break
    end
end
act_ven = time_fin;

% 10% max T
T_10_mod = max_t_mod * .1;
k = 0;
%ven
for i = 1:length(T_predicted)
    if((k == 0) && (T_predicted(i) >= T_10_mod))
        time_int = t(i);
        k = k + 1;
    end
    if((k == 1)&& (T_predicted(i) <= T_10_mod))
       time_fin = t(i);
       break
    end
end
act_mod = time_fin - time_int;
```

2. burn_geometry.m

```
function [Ab] = burn_geometry(ri,r0,h,rb)

    if rb >= r0 % motor is burnt out
        Ab = 0; % [m^2]
    else % there is grain remaining
        %% BURN AREA
        Ab = 2*pi()*(r0^2-(ri+rb)^2) + 2*pi()*(h-2*rb)*(ri+rb); % [m^2] total
            burn area

        %% BURN VOLUME
        % Vb = ; % [m^3]
    end
```

3. thrust_calc.m

```
function [ThSM_en, Cs] = thrust_calc(Pa, Pc, Ae, rho_p, burn_rate, A_burn, AR_sup)% delta_Vol)
    Pc_en = Pc * 145.038; % [psi] chamber pressure

    %% CEA RUN
    ERROR = 0;
    try % tests if there is any CEA output
        % OUTPUT1 GIVES VALUES IN THE CHAMBER
        % OUTPUT3 GIVES VALUES AT NOZZLE EXIT
        [Output1, Output3] = RUN_CEA(Pc_en, AR_sup);
    catch
       ERROR = 1;
    end

    if ERROR == 1 % sets Mach and alpha to zero if output DNE
        alpha = 0;
        Mach = 0;
        Pe = Pa;
        Cs = 0;
        rho_g = 0;
    else
        alpha = Output3.a; % [m/s] sonic velocity
        Mach = Output3.Mach; % Mach number
        Pe = Output3.P * 1e5; % [Pa] nozzle exit pressure
        Cs = Output3.Cstar; % [m/s] characteristic velocity
        rho_g = Output1.rho; % [kg/m^3] propellant gas density, chamber
    end

    %% MASS FLOW CALCULATION
    m_dot = (rho_p - rho_g)*A_burn*burn_rate; % [kg/s] propellant mass flow rate

    %% THRUST CALCULATION
    ThSM =  m_dot * (Mach*alpha) + (Pe - Pa) * Ae; % [N] thrust SI
    ThSM_en = ThSM * 0.224809; % [lbf] imperial thrust to match curve data
```

4. RUN_CEA.m

```
function[SM_results1,SM_results3] = RUN_CEA(Pc_en, AR_sup)
    SM_inputs = CEAinput();          %cea input class object for easy input and definition

    % set conditions for the CEA run of H202
    SM_inputs.ox1      = 'NH4CLO4(I)';              %primary oxidizer
    SM_inputs.ox1T     = 536;                       %primary ox temp (R)

    SM_inputs.ox2      = 'AL(cr)';                  %primary oxidizer
    SM_inputs.ox2T     = 536;                       %primary ox temp (R)

    SM_inputs.fu1      = 'C4H6,butadiene';                %primary fuel
    SM_inputs.fu1T     = 536;                       %primary fuel temp (R)

    SM_inputs.ox1wt    = 72;                         %primary oxidizer weight (by mass) in total
```

```
15      SM_inputs.ox2wt     = 10;                       % secondary oxidizer
            weight (by mass) in total
16      SM_inputs.fu1wt     = 18;                       % primary fuel weight (by
            mass) in total
17
18      SM_inputs.Pc        = Pc_en;                     %chamber pressure (psi)
19
20      SM_inputs.supar     = AR_sup;                   % nozzle expansion ratio
21
22      %% Run CEA
23
24      SM_inputs.runCEA();                             %execute CEA for the above
            conditions
25      SM_results1 = SM_inputs.getCEAresults(1,'si');
26      SM_results3 = SM_inputs.getCEAresults(3,'si');          %1 for chamber
            conditions (2 for throat, 3 for nozzle exit: requires arg for area
            ratio or exit pressure), 'en' for english units
```

5. CEA_input.m

```
1  % CEA input class for easier use of NASA Chemical Equilibrium Analysis
2  %
3  % Drew Sherman
4  % Purdue University
5  % Made 4/12/2017
6  %
7  % Mitch Woolever
8  % University of Colorado Boulder
9  % Modified 4/20/2017
10 %
11 % Defines a CEA object with all the properties required to run CEA
12 %   (SEE BELOW)
13 %
14 % METHODS:
15 %   runCEA(obj): used by a CEA object to run CEA for the values (usually
16 %        defined in a parent code using '.' notation
17 %
18 %   readCEAout(obj,location): reads the Detn.out file of CEA for the values
       the
19 %        user requested using obj.out command. The function returns a structure
20 %        with all relevant data from the CEA run
21
22
23 classdef CEAinput
24    properties
25       Filename = ''; %specify name for output file(.txt)
26
27       % specify oxidizer & fuel conditions inputs
28        ox1    = '';              %% primary oxidizer
29        ox2    = '';              %% secondary oxidizer
30        fu1    = '';              %% primary fuel: RP-1
31        fu2    = '';              %% secondary fuel
32        ox1chem    = '';          %% optional primary oxidizer chemical formula
              if required (captilize chemical symbols)
33        ox2chem    = '';          %% optional secondary oxidizer chemical
              formula if required (captilize chemical symbols)
```

```
34          fu1chem    = '';          %% optional primary fuel chemical formula if
                requied (captilize chemical symbols)
35          fu2chem    = '';          %% optional secondary fuel chemical formula if
                 requied (captilize chemical symbols)
36
37          ox1wt = 0;             %% wt fraction of primary oxid in total oxid [1]
38          ox2wt = 0                %% wt fraction of secondary oxid in total oxid
                [1]
39          fu1wt = 0;             %% wt fraction of primary fuel in total fuel [1]
40          fu2wt = 0;               %% wt fraction of secondary fuel in total fuel
                [1]
41          ox1T  = 0;               %% optional input of primary oxid temperature
                which enthalpy is evaluated [degR]
42          ox2T  = 0;               %% optional input of secondary oxid
                temperature which enthalpy is evaluated [degR]
43          fu1T  = 0;               %% optional input of primary fuel temperature
                which enthalpy is evaluated [degR]
44          fu2T  = 0;               %% optional input of secondary fuel
                temperature which enthalpy is evaluated [degR]
45          ox1H  = 0;               %% optional input of primary oxid enthalpy of
                formation [cal/mol]
46          ox2H  = 0;               %% optional input of secondary oxid enthalpy
                of formation [cal/mol]
47          fu1H  = 0;               %% optional input of primary fuel enthalpy of
                formation [cal/kg]
48          fu2H  = 0;               %% optional input of secondary fuel enthalpy
                of formation [cal/mol]
49
50
51          Pc     = [];             %% Chamber pressure [psia]
52          OF     = [];             %% mixture ratio [wt oxid/wt fuel]
53          Phi    = [];             %% equivalence ratio
54
55          Pe     = []              %% optional input for exit pressure
56          PR     = [];             %% pressure ratios [Pc/Pe]
57          subar = [];              %% subsonic area ratios [A/At]
58          supar = [];              %% supersonic area ratios [A/At]
59
60
61
62          CR     = 0;              %% chamber contraction ratio [Ac/At]
63          flow   = 'eq';           %% flow type [eq or fz]
64          out    = 'isp ivac cp p t gam m'; %%'aeat p t'; %maximum eight output
                parameters in one call to CEA
65          %outlab= ['Area ratio' 'Temperature' 'Pressure' 'xH2O' 'xCO2' 'xCO'];
66
67      end
68      %%
69      methods
70          function runCEA(obj)
71            %finds and runs CEA given the properties of the CEA object defined
72
73
74            cd './CEA Code'                %find the CEA code directory (should
                  be one level down from this class def)
```

```
75
76          %call input generation function
77          inpgen_rocket_3(obj.ox1,obj.ox2,obj.ox1wt,obj.ox2wt,obj.ox1T,...
78              obj.ox2T,obj.ox1chem,obj.ox2chem,obj.ox1H,obj.ox2H,obj.fu1,...
79              obj.fu2,obj.fu1wt,obj.fu2wt,obj.fu1T,obj.fu2T,obj.fu1chem,...
80              obj.fu2chem,obj.fu1H,obj.fu2H,obj.Pc,obj.OF,obj.Phi,obj.PR,...
81              obj.subar,obj.supar,obj.CR,obj.flow,obj.out);
82
83          % execute CEA600
84          system('CEA600.exe');
85
86          cd '..'                         %return to the parent directory
87        end
88
89      function CEA_out = getCEAresults(obj, dataLoc, units)
90          %location = 1: chamber, location = 2: throat, location = 3: exit
91          %units: 'si' or 'en' for SI or English units returned
92          %
93          % this function returns a structure with all the relevant output
94          %  data from CEA
95
96          % INPUTS:
97          %   dataLoc: 1,2,3: chamber, throat, exit
98          %   units: 'si' or 'en' ONLY!
99          %
100         % OUTPUTS:
101         %   CEA_out: structure with Gamma, P0, T0, rho, Cp, mu, k, Pr, a
102         %
103
104         %% Setup output
105
106         % output file
107         directory = '.\CEA Code';
108         FID = fopen(fullfile(directory, 'Detn.out'));
109         CEA_output = fscanf(FID,'%c');
110         fclose(FID);
111
112         %---------- Extract Output from File --------------%
113
114         %% Stagnation Properties
115
116             % find Mach
117             Mach_index = strfind(CEA_output, 'MACH NUMBER') + 15;
118             end_ofline_index = strfind(CEA_output, 'TRANSPORT PROPERTIES (
                    GASES ONLY)') - 1;          %finds endline
119             line_of_Mach = CEA_output(Mach_index:end_ofline_index);
                            %makes array with only the mach values
120             Mach_array = sscanf(line_of_Mach, 'c
                    P0_index = strfind(CEA_output,' P,BAR') + 15; end_ofline_index =
                    strfind(CEA_output,' T,K') - 1; line_of_P = CEA_output(P0_index : end_ofline_index); P_array =
                    sscanf(line_of_P,' P0 = P_array(1); T0_index =
                    strfind(CEA_output,' T,K') + 15; end_ofline_index =
                    strfind(CEA_output,' RHO,') - 1; line_of_T = CEA_output(T0_index : end_ofline_index); T_array =
                    sscanf(line_of_T,' T0 = T_array(1); C_index = strfind(CEA_output,' CSTAR') + 15; C_index =
                    C_index(1); end_ofline_index = strfind(CEA_output,' CF') - 1; line_of_C = CEA_output(C_index :
```

$end_ofline_index); C_array = sscanf(line_of_C,' C_star = C_array(1); Isp_index = strfind(CEA_output,' Ivac') + 15; end_ofline_index = strfind(CEA_output,' Isp') - 1; line_of_Isp = CEA_output(Isp_index : end_ofline_index); Isp_array = sscanf(line_of_Isp,' Isp_index = strfind(CEA_output,' Isp') + 15; end_ofline_index = strfind(CEA_output,' MASSFRACTIONS') - 3; line_of_Isp = CEA_output(Isp_index : end_ofline_index); Isp_atm_array = sscanf(line_of_Isp,' Cf_index = strfind(CEA_output,' CF') + 15; end_ofline_index = strfind(CEA_output,' Ivac') - 1; line_of_Cf = CEA_output(Cf_index : end_ofline_index); Cf_array = sscanf(line_of_Cf,' eps_index = strfind(CEA_output,' Ae/At') + 15; end_ofline_index = strfind(CEA_output,' CSTAR') - 1; line_of_eps = CEA_output(eps_index : end_ofline_index); eps_array = sscanf(line_of_eps,' ifdataLoc == 1Isp = Isp_atm_array(1); Isp_vac = Isp_array(1); Cf = Cf_array(1); eps = 1; elseIsp = Isp_atm_array(2); Isp_vac = Isp_array(2); Cf = Cf_array(2); iflength(eps_array) < 2eps = eps_array(1); elseeps = eps_array(2); endendgamma_index = strfind(CEA_output,' GAMMAs') + 15; end_ofline_index = strfind(CEA_output,' SONVEL, M/SEC') - 1; line_of_gamma = CEA_output(gamma_index : end_ofline_index); gamma_array = sscanf(line_of_gamma,' rho_index = strfind(CEA_output,' RHO, KG/CUM') + 15; end_ofline_index = strfind(CEA_output,' H, KJ/KG') - 1; line_of_rho = CEA_output(rho_index : end_ofline_index); rho_array = sscanf(line_of_rho,' iflength(gamma_array) > 1rho_array(1) = rho_array(1) * 10^(rho_array(2)); rho_array(2) = rho_array(3) * 10^(rho_array(4)); iflength(gamma_array) > 2rho_array(3) = rho_array(5) * 10^(rho_array(6)); endendcp_index = strfind(CEA_output,' Cp,') + 15; cp_index = cp_index(1); end_ofline_index = strfind(CEA_output,' GAMMAs') - 1; line_of_cp = CEA_output(cp_index : end_ofline_index); cp_array = sscanf(line_of_cp,' mu_index = strfind(CEA_output,' VISC,') + 15; mu_index = mu_index(1); end_ofline_index = strfind(CEA_output,' WITHEQUILIBRIUM') - 1; line_of_mu = CEA_output(mu_index : end_ofline_index); mu_array = sscanf(line_of_mu,' k_index = strfind(CEA_output,' CONDUCTIVITY') + 15; k_index = k_index(2); end_ofline_index = strfind(CEA_output,' PRANDTL') - 1; end_ofline_index = end_ofline_index(1); line_of_k = CEA_output(k_index : end_ofline_index); k_array = sscanf(line_of_k,' Pr_index = strfind(CEA_output,' PRANDTL') + 15; Pr_index = Pr_index(1); end_ofline_index = strfind(CEA_output,' WITHFROZEN') - 1; line_of_Pr = CEA_output(Pr_index : end_ofline_index); Pr_array = sscanf(line_of_Pr,' a_index = strfind(CEA_output,' SONVEL') + 15; end_ofline_index = strfind(CEA_output,' MACHNUMBER') - 1; line_of_a = CEA_output(a_index : end_ofline_index); a_array = sscanf(line_of_a,' P_index = strfind(CEA_output,' P, BAR') + 15; end_ofline_index = strfind(CEA_output,' T, K') - 1; line_of_P = CEA_output(P_index : end_ofline_index); P_array = sscanf(line_of_P,' T_index = strfind(CEA_output,' T, K') + 15; end_ofline_index = strfind(CEA_output,' RHO,') - 1; line_of_T = CEA_output(T_index : end_ofline_index); T_array = sscanf(line_of_T,' ifisempty(strfind(CEA_output,' MASSFRACTIONS')) = 1mass_index = strfind(CEA_output,' MASSFRACTION') + 19; end_ofline_index = strfind(CEA_output,' *THERMODYNAMIC') - 6; line_of_massf = CEA_output(massf_index : end_ofline_index); split = strsplit(line_of_massf,' '); expression =' *'; value_array = zeros(length(split)); names = zeros(1, length(split)); massf = struct(); fori = 1 : length(split)matchStr = regexp(split(i), expression,' match'); compounds = matchStr1, 1; comp_array(i) = compounds(1, 1); forj = 1 : length(compounds(1, 2 : end))value_array(i, j) = str2double(compounds(1, j + 1))/1e5; endmassf.(comp_arrayi) = value_array(i, dataLoc); endmolef = []; elsemolef_index = strfind(CEA_output,' MOLEFRACTION') + 19; end_ofline_index = strfind(CEA_output,' *THERMODYNAMIC') - 6; line_of_molef = CEA_output(molef_index : end_ofline_index); split = strsplit(line_of_molef,' '); expression =' *'; value_array = zeros(length(split)); molef = struct(); fori = 1 : length(split)matchStr = regexp(split(i), expression,' match'); compounds = matchStr1, 1; comp_array(i) = compounds(1, 1); forj = 1 : length(compounds(1, 2 : end))value_array(i, j) = str2double(compounds(1, j + 1))/1e5; endmolef.(comp_arrayi) =$

$value_array(i, dataLoc); endmassf = []; endifstrcmp(units,'en') == 1cp_array = cp_array * 1000 * 2.388e-4; mu_array = mu_array * 0.000067197/12; k_array = k_array * 100/1000 * 0.001927/144; a_array = a_array * 3.28084 * 12; rho_array = rho_array * 3.61273e-5; T_array = T_array * 1.8; P_array = P_array * 14.5038; P0 = P0 * 14.5038; T0 = T0 * 1.8; C_star = C_star * 3.28084; Isp_vac = Isp_vac * 3.28084; Isp = Isp * 3.28084; endCEA_out = struct('Mach', Mach_array(dataLoc),' P0', P0,' T0', T0, ...'Cstar', C_star,' Ivac', Isp_vac, ...'Isp', Isp,' CF', Cf,' GAM comp_array; endendend$