Development of a Simulation Environment and Drag Controller for High-Powered Rockets

University of Minnesota
Samuel Courtier, Devin Vollmer

December 2013

AEM 4495, Research and Design of Aerospace Systems

Abstract

This paper outlines a methodology for simulation and altitude control of a variable-drag, high-powered rocket. To this end, we modified the MATLAB-based Cambridge Rocketry Toolbox (CRT), and investigated the practical implementation of pitot tubes to supply airspeed data for drag control systems.

We attempted to launch a rocket to a target apogee of 337.2 m above ground. Our simulation predicted an altitude of 344.3 m above ground, and our flight data indicated that the rocket reached an altitude of 344.1 m above ground. This suggests that our modified (patched) CRT possesses exceptional modeling potential. The rocket's failure to reach the specific target can be attributed to incorrect selection of the target altitude. This was due to the use of insufficiently powerful modeling software (used as a stop-gap while our own solution was still in development), as it was later found that the control system implemented has an effective range of 342.1 m to 348.5 m.

We concluded that using a commercial altimeter that provides velocity readings is a more practical approach for generating drag control inputs than the combination of barometer and pitot tube.

Forward and Acknowledgments

This report is designed to meet two objectives in the most condensed manner we could manage. First, it seeks to broadly outline our most salient research and experimental efforts over the course of the semester. And second, it attempts to provide the University of Minnesota Rocket Club with guidance in the use and expansion of our methods for future projects.

We would like to thank the University of Minnesota Rocket Club for its supporting resources, Seth Frick and David Escobar for their technical assistance, Gary Stroick for going well out of his way to make our test launches possible, and Gary Balas for donating his time to act as our advisor and for providing us with essential insight and critique throughout the process.

Contents

1 Introduction

Apogee targeting is a common feature of almost all major rocketry competitions. The University of Minnesota's newly founded Rocketry Club has set out to develop an inexpensive, standardized, and efficient methodology for launching high-powered rockets to pre-specified altitudes of up to $100,000$ ft within the next five years.

These launch vehicles will feature stability systems to ensure a constant vertical orientation. Thus, an effective strategy for achieving a target apogee is to design a rocket that will overshoot its goal under nominal conditions, and then intelligently apply an air brake throughout flight to eliminate the excess kinetic energy and mitigate the effects of uncertainty from factors like variable motor performance and unpredictable atmospheric conditions. This regime has proven effective in past attempts by other schools (e.g. Century College's 2012-2013 USLI rocket), but is in need of a great deal of refinement to attain the precision sought after by the U of M team.

To achieve meaningful results in a single semester, and to ensure that implementation would require the fewest components possible (and would therefore be easily accessible to rocket club members), we reduced the problem for altitude determination and drag control to a one-dimensional ballistic model. The air braking system we designed is simple to build and its performance was readily verified using DATCOM and other aerodynamic analytical models.

Another research objective was to compare the effectiveness of two approaches for altitude and velocity measurement; a pitot tube and barometer combination versus a commercial altimeter with on board filtering and velocity estimation. The rational was that, given a relatively stable upward trajectory, numerically deriving altitude readings to yield velocity may introduce more overall inaccuracy than would using a pitot tube.

2 Methods

The experiment was intended to span the entire semester with September and October spent on the development of analytical models for the launch vehicle, wind tunnel tests, and simulations. The remaining months were to be spent on four test-launches and custom simulator refinement.

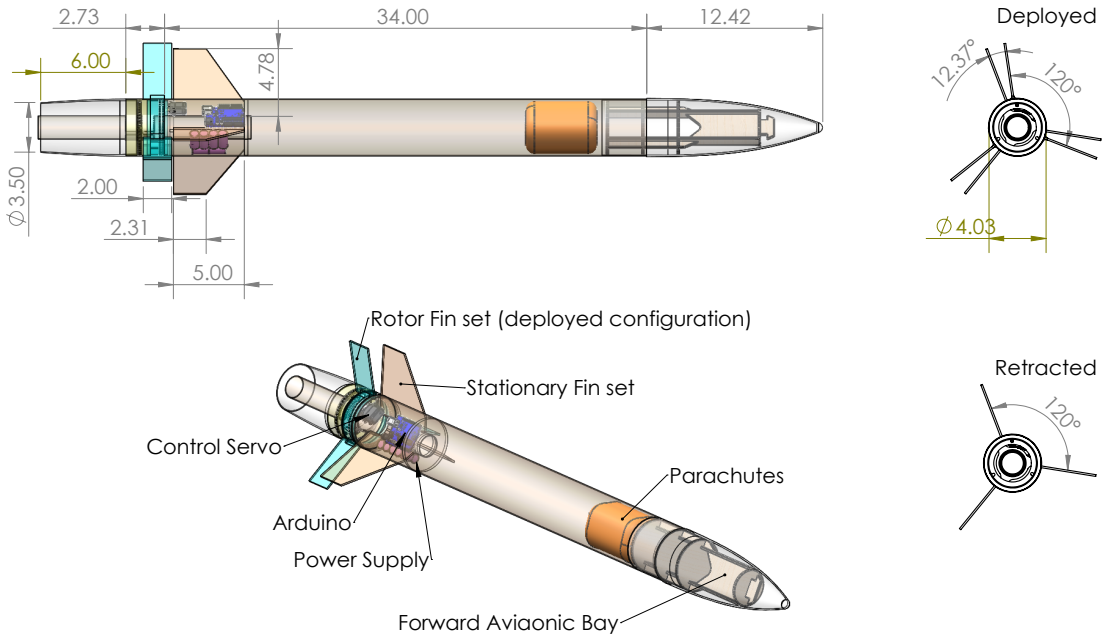

Figure 1: 3-View Drawing of the launch vehicle used in the experiment. All units are in inches.

The purpose of the first launch was to examine the performance of an Ardupilot flight computer to determine the feasibility of re-purposing the unit for high-powered rockets—the Ardupilot was found to be inappropriate for our needs. The second and third tests were done to verify the dynamic models of the deployed an retracted air-brake configurations of the vehicle (See Figure 1), and the final launch was a full test of the controller's ability to prevent the rocket from exceeding its target apogee. The final three launches provided a basis for the assessment of our dynamic models and simulation environment.

Wind tunnel tests were conducted at 37.0 m/s to verify the DATCOM analytical models used to profile the dynamic characteristics of the rocket. This was the top speed of the wind tunnel—time constraints prohibited an investigation of the drag effects at lower Reynolds numbers. Primarily though, this test enabled us to fully characterize the drag brake. This was a crucial step in establishing an accurate simulation environment. Results from the tunnel compared with the analytical solution are plotted in Figure 8.

After validating DATCOM models using wind tunnel data, a linear relation between drag coefficient and angle of attack was approximated from 0 to 6 degrees. The modified CRT [8] code accepted these linear relations as .cvs files. A MATLAB function identical to the chosen control law was created to operate within the modified CRT simulation environment, enabling accurate modeling of actively controlled rocket launches.

The target altitude for the final launch was found using estimations from Open Rocket rocketry simulator in order to determine the mean altitudes for both the brake-deployed and brake-retracted configurations for the vehicle. This software was limited in its ability to model complex aerodynamic interactions and was incapable of modeling air brake deployment control. Instead, the strategy involved making assumptions regarding uncertainties in motor performance to analytically derive probable apogees [6],[5]. The target was then chosen to be 2σ away from the mean apogee of this derived probability distribution. This distribution represented the propagated uncertainties in motor performance. The rational was that, in the unlikely event of an under-performing motor, the vehicle could still reach its target without the aid of an air brake.

Altitude was measured on all flights using the StratoLogger SL100 commercial altimeter by PerfectFlite. The SL100 was chosen to be the standard of measurement due to its trusted performance in past Rocket Club projects, and its use as the standard altitude measurement device of the NASA USLI competition. An Arduino with barometric sensor and pitot tube was also used as the prototype flight computer to control the deployment and retraction of the air brake. With both of these instruments, it was possible to evaluate the performance of the Arduino flight computer as well as the SL100.

3 Theory

3.1 Analytical Model

Drag force contributes significantly to the deceleration of the rocket and was the parameter of interest for our control system. The DATCOM analytical model for rockets provided a means by which to derive the profile and interference drag coefficient for the rocket, as well as the modified $C_{d,b}$ for the drag brake. We define body drag $C_{d,body}$ with the parameters illustrated in figure 6,

$$C_{d,body} = C_{f,body}\left[1 + \frac{60}{l_{TR}/d_b^3} + 0.0025\frac{l_b}{d_b}\right]\left[2.7\frac{l_n}{d_b} + 4\frac{l_b}{d_b} + 2\left(1 - \frac{d_d}{d_b}\right)\frac{l_c}{d_b}\right]. \tag{1}$$

These equations were key to building the analytical model. They are described in [1].

We can estimate the form drag by evaluating the ratio between the nose cone's base diameter d_b and the tapered diameter of the boat tail d_d [1]

$$C_{d,base} = 0.029 \frac{(d_d/d_b)^3}{\sqrt{C_{d,body}}}. \tag{2}$$

Drag due to the fins is dependent on Reynolds Number Re and $C_{f,fin,v}$ depends on the criticality of Re. According to [1], $Re_T = 5 \times 10^5$, so

$$C_{f,fin,v} = \begin{cases} \frac{1.328}{\sqrt{Re}} & \text{if } Re < Re_T \\ \frac{0.074}{Re^{1/5}} - \frac{B}{Re} & \text{if } Re > Re_T \end{cases}. \tag{3}$$

The parameter B is defined in Equation 4 as

$$B = Re_T \left(\frac{0.074}{Re^{1/5}} - \frac{1.328}{\sqrt{Re}} \right), \tag{4}$$

and the Reynolds number is determined using the fin mid-cord length l_m , velocity V, air density ρ, and dynamic viscosity μ

$$Re = \frac{\rho V l_m}{\mu}. \tag{5}$$

For our design, we used variations of the fin drag equation to calculate drag contributions in the deployed and retracted flight modes. See Figure 7. When deployed, the drag was modeled using a total of six fins, three of which had the stationary mid-cord span length of $l_m = l_{m,stat}$ and three with $l_m = l_{m,roto}$ as seen in Figure 7. It was also necessary to calculate the fin planform area A_{fp} and fin exposed area A_{fe} for each configuration. Fin thickness t_f, was constant for both configurations, but the viscous coefficient $C_{f,fin,v}$ had to be computed for each fin set in the deployed configuration.

Thus, for each case, we can determine the fin drag $C_{d,fin}$ to be

$$C_{d,fin} = \begin{cases} 2C_{f,fin,v} \left(1 + 2\frac{t_f}{l_m}\right) \left(\frac{4nA_{fp}}{\pi d_b^2}\right) & \text{if retracted} \\ 2C_{f,stat,v} \left(1 + 2\frac{t_f}{l_{m,stat}}\right) \left(\frac{4nA_{fp}}{\pi d_b^2}\right) + 2C_{f,roto,v} \left(1 + 2\frac{t_f}{l_{m,roto}}\right) \left(\frac{4nA_{fp}}{\pi d_b^2}\right) & \text{if deployed} \end{cases}. \tag{6}$$

where

$$A_{fe} = \frac{1}{2}(l_r + l_t) l_s, \tag{7}$$

and

$$A_{fp} = A_{fe} + \frac{1}{2} d_f l_r. \tag{8}$$

Interference effects due to fins were also modeled using equation 9 to formulate $C_{d,i}$

$$C_{d,i} = 2C_{f,fin,v} \left(1 + 2\frac{t_f}{l_m}\right) \frac{4n(A_{fp} - A_{fe})}{\pi d_f^2}. \tag{9}$$

For the body of the vehicle, increase in drag coefficient as a function of angle of attack $C_{d,(\alpha),body}$ includes the coefficients δ and η. We assumed angles of attack to be lower than 6 deg, and formulated a linear approximations to the data from [1]

$$C_{d,(\alpha),body} = 2\delta\alpha^2 + \frac{3.6\eta(1.36l_{TR} - 0.55l_n)}{\pi d_b}\alpha^3. \tag{10}$$

We used a similar formulation for the fins and introduced the parameters k_{fb} and k_{bf} to include interference effects [1]

$$C_{d,(\alpha),fin} = \alpha^2 \left[4.8\frac{A_{fp}}{\pi d_f^2} + 3.12(k_{fb} + k_{bf} - 1)\left(4\frac{A_{fe}}{\pi d_f^2}\right)\right], \tag{11}$$

with

$$k_{fb} = 0.8065R_s^2 + 1.1553R_s, \tag{12}$$

and

$$k_{bf} = 0.1935R_s^2 + 0.8174R_s + 1. \tag{13}$$

Here, R_s is the fin section ratio, which is given in terms of l_{TS}, the total span of the vehicle, and d_f the diameter of the vehicle at the fins

$$R_s = \frac{l_{TS}}{d_f}. \tag{14}$$

The drag coefficient is fully defined in terms of the sum of all drag contributions at zero angle of attack $C_{d,(0)}$, drag coefficient as a function of angle of attack $C_{d,(\alpha),body}$, and the fin drag coefficient $C_{d,(\alpha),fin}$. Combined,

$$C_d = C_{d,(0),body} + C_{d,base} + C_{d,i} + C_{d,(\alpha),body} + C_{d,(\alpha),fin}. \tag{15}$$

We can account for compressibility effects present in high speed flight using the approximation in Equation (16) [3]

$$C_d{}' = \frac{C_d}{\sqrt{1 - Ma}}. \tag{16}$$

All together, this equation enables the drag characterization of a given vehicle with respect to angle of attack. This was important for building accurate parameters for use in the modified CRT simulation.

3.2 Summary of Drag Controller Algorithm

Our control scheme involved toggling between two states: high drag (rear fins deployed) and low drag (rear fins retracted), according to whether an apogee prediction model (Section 3.3) indicated, respectively, an above-target or at-or-below target maximum height was likely. Due to the time needed to fully deploy/retract the rear fins, a delay between successive toggle signals was set. A MATLAB implementation of the algorithm is presented in Appendix C.

3.3 Apogee Prediction Model

Our controller design required the prediction of apogee given post-burnout flight conditions reported by sensors. For computational efficiency, we used a simple ballistic model; given initial altitude z_i, and gravitational acceleration g, the equation

$$z_{max} = z_i + \frac{v_t^2}{2g} \log\left(\frac{v_i^2 + v_t^2}{v_t^2}\right), \tag{17}$$

with terminal velocity

$$v_t = \sqrt{\frac{2mg}{C_d A_{ref} \rho}}, \tag{18}$$

gives the apogee height [7]. Here, m, A_{ref}, and C_d are the mass, reference area, and coefficient of drag, respectively. Because our fights were low altitude, the air density ρ was approximated as constant.

In the actual implementation, z_i was derived from barometer readings, v_i was given by pitot tube readings, C_d was determined by the drag brake deployment state, and the remaining variables were measured in the lab.

4 Results

Apogee (m)	Retracted	Deployed	Actively Controlled
CRT	386.3	371.1	344.3
SL100	384.4	x	344.1
Arduino	395.0	x	383.5

Table 1: Apogee results from two launches compared with modified CRT simulation results. After the first launch, the vehicle experienced a hard landing that resulted in the scrubbing of a second launch to collect brake-deployed data. The subsequent repairs made for the actively controlled launch added an additional 159 g of mass to the airframe.

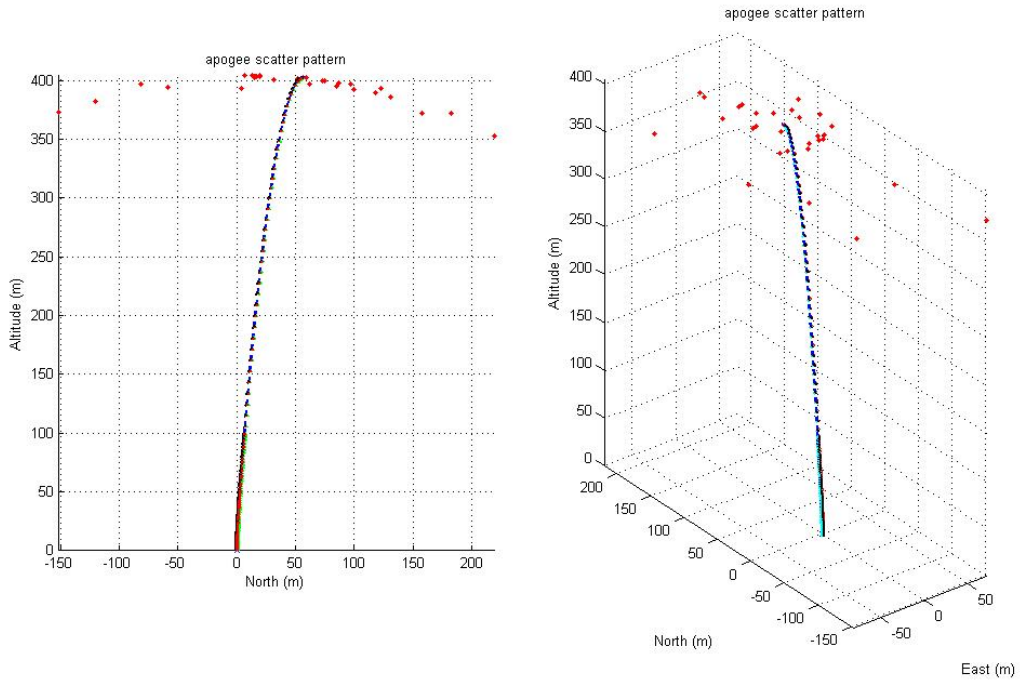

Figure 2: Output from the modified CRT simulation. The simulation predicted a mean altitude of 344.3 m. It used the dynamic data gathered from wind tunnel tests. The red dots are apogees predicted using a Monte Carlo method to generate a statistical model for flight given the uncertainties in the atmosphere. The trajectory of the simulated rocket is also plotted.

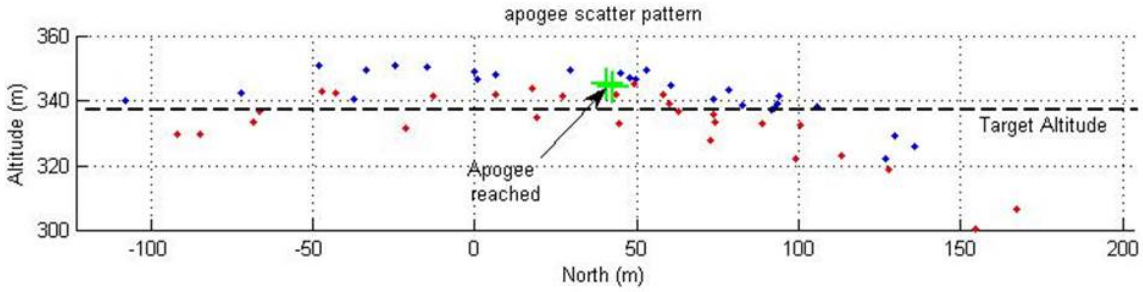

Figure 3: Modified CRT simulation data assuming a retracted (blue dots) and fully-deployed (red dots) configurations for two different launch scenarios. The average apogees were found to be 348.5 m for the retracted configuration and 342.1 m for the deployed configuration. This range of possible target altitudes is smaller than initial estimates.

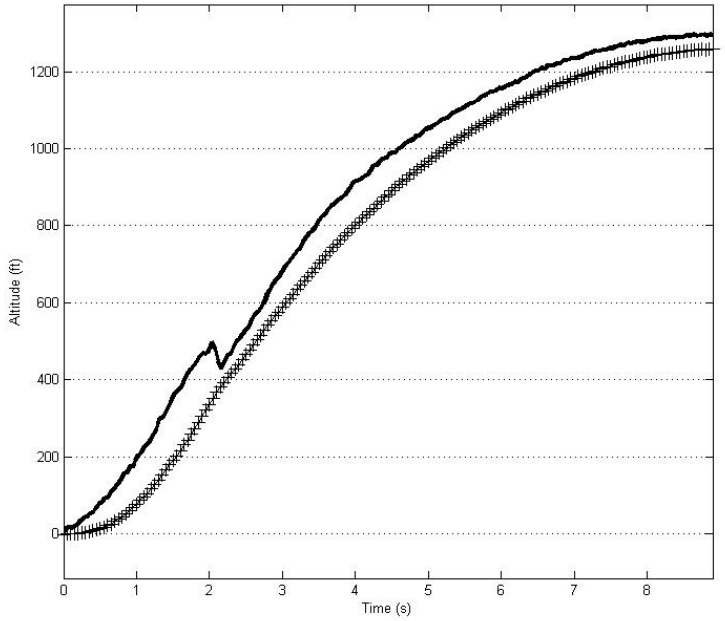

Figure 4: Results from second test launch of the vehicle matched to the apogee event for both the SL100 (+) and Arduino (black line). The purpose of this test was to confirm the dynamics of the brake-retracted configuration as modeled in DATCOM. In this test, as seen in figure 5, the Arduino reported a higher altitude than the SL100, but this difference is not as dramatic as it was in the final launch. The difference in apogee indicated by the two sensors was 11.0 m. The spike in the Arduino data could have been caused by a high acceleration period disrupting the signal connections to the sensors.

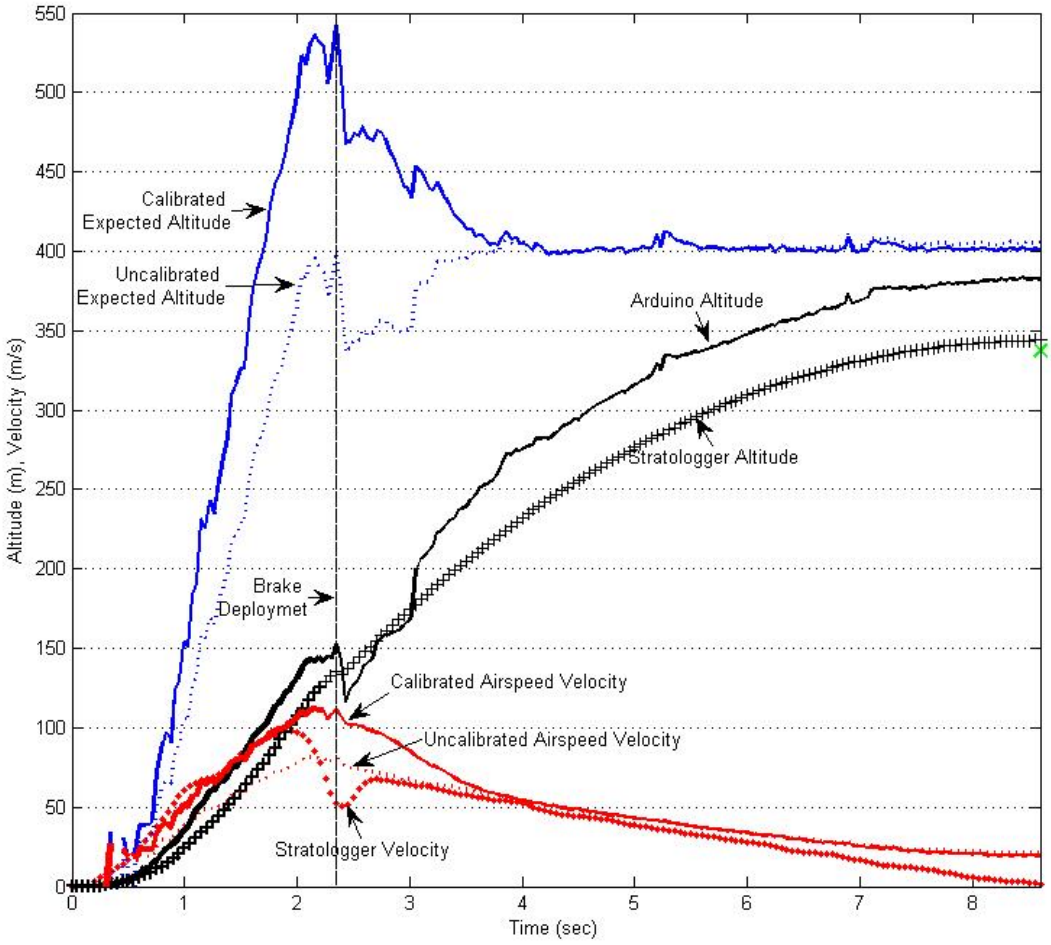

Figure 5: Results from the final test launch of the vehicle matched to the apogee event for both the SL100 (+) and Arduino (black line). Together are the altitudes, velocities, and predicted altitudes from the Arduino with and without calibration applied to the pitot tube. The period of bold line-widths indicate motor thrusting. The green '×' on the right-hand edge is the target altitude of the vehicle. The Arduino did not appear to capture altitude as accurately as in the previous launch, and the noticeable dip coinciding with the increase in drag due to brake deployment suggests that the system is susceptible to fault under accelerations.

5 Discussion

From Figure 5, the reported Arduino altitude (black line) does not correspond well with the SL100 altitude (+). It was found that the Arduino reported an apogee of 382.5 m and the

SL100 reported an apogee of 344.1 m. After the launch, it was observed that the barometric sensor for the Arduino became coated in fuel exhaust from the motor at some point during the launch—a factor that could have upset the performance of the sensor. The sudden data displacement shortly after burnout suggests that a reassessment of barometric sensor performance is necessary. On the previous launch (see figure 4), it was also observed that the pressure sensor on board the Arduino reported an additional 11.0 m of apogee height when compared to the SL100. This is in contrast to a difference of 38.4 m on this launch, giving additional evidence that the sensor may be flawed or may require more intensive calibration.

The below-freezing conditions at the launch site necessitated compensatory pitot tube calibration. This was not programmed into the Arduino during the launch, and the resulting velocity as reported in real-time during the flight is plotted in Figure 5 as small red dots. After the launch, the proper calibration was applied and the calibrated airspeed velocity (red line) was found to agree with the values reported from the SL100 (large red dots), giving strong evidence that pitot tubes can provide accurate real time velocity. The divergence in the pitot tube velocity when compared with the SL100 velocity is due to the fact that the SL100 is using the vertical changes in position to numerically derive the velocity, whereas the pitot tube uses a flow velocity that may always be present even if the rocket has reached apogee.

The Arduino uses the pitot tube velocity as v_i in the maximum altitude prediction algorithm found in Equation (17). The results are plotted in Figure 5 as blue dots for uncalibrated pitot tube velocity, and as a blue line for calibrated pitot tube velocity. Once the maximum predicted altitude is greater than the target altitude, the brake is deployed. The delay between the Arduino sensing that its trajectory will go past the target and the actual deployment of the brake can be attributed to two software issues. The first being that a bug in the code resulted in a delay of 0.2 seconds after the maximum altitude determination and until the command to change the fin position was actually sent. The second is the mechanical delay of the brake to reach its fully deployed position. The vertical black dots in Figure 5 account for these delays and indicate when the brake reached its fully deployed position.

After comparing this with a fully characterized simulation using the modified CRT, it was predicted that an actively controlled vehicle would reach a mean apogee of 344.3 m. This result corresponds extremely well with the SL100 result of 344.1 m, and strongly suggests that the modified CRT environment has the potential to model ascent control systems accurately. However, because this result is close to both the predicted range limits of the air brake, there is inconclusive evidence that the air brake, as designed, has the ability to reach a particular target within the operational range—between 342.1 m to 348.5 m as seen in Figure 3.

6 Conclusions

The results show that we can accurately model the flight of a high-powered rocket using experimental data to guide the development of a DATCOM model as well as the control systems integrated into these rockets. This began by using OpenRocket [9] and RASAero [10] to develop the initial dynamic models, but it was found that wind tunnel data used in conjunction with DATCOM models yielded the most accurate results when input into a modified version of the CRT. This is also highlights CRT's ability to be used as a design tool to determine C_d requirements for altitude targeting systems—an ability the Rocket Club did not previously posses.

However, it was found with further analysis that the change in C_d provided by the air brake is not sufficient to reach the target altitude of 337 m, indicating that the original assumptions regarding the performance of the fins were optimistic. This does not immediately compromise the system's precision however, and future launches my be conducted to verify the air brake's ability to consistently hit a target of 344.1 m.

The data from the final launch indicates that pitot tubes can be used to capture the velocity in real time for a high-powered rocket, but problems arise when the rocket experiences none-zero pitch or yaw, since the current flight computer cannot account for orientation. For future developments involving pitot tubes, it is recommended that the entire system exist in a temperature controlled environment that is at room temperature, and that an Inertia Measurement Unit (IMU) be integrated with the Arduino to resolve vertical velocity components.

It must be stated though, that for a simple altitude control system, a pitot tube is not necessarily the optimal solution. The results have shown that the SL100 can effectively capture altitude with fidelity nearly equivalent to that of the Arduino. If this can be utilized to compute the maximum predicted altitude in real time, then a pitot tube is not needed for accurate altitude targeting.

A Appendix: Application of the UMN CRT Patch

A patch was designed for version 2.3 of the Cambridge Rocketry Toolbox. The patch enables the use of a drag control algorithm, provides a useful template for constructing a simulation, and fixes a minor bug in the CRT. It consists of eight files:

- `README.txt` instructions on how to apply the patch.
- `ascentcalc.m` a modified ascent trajectory calculator that allows the user to simulate an ascent under a customizable drag control scheme,
- `rocketflight_monte.m` a modification of the original CRT file that fixes a bug pertaining to string comparison.
- `DragControl.m` a file in which a drag control scheme can be specified,
- `UMNCRTTemplate.m` a file that guides a user in entering empirical data to be used in the flight simulation, and constructs the simulation. Once run, a .csv file of mean flight variables recorded throughout the ascent is output to the CRT folder, and
- `atmoData.csv, dragData.csv`, and `thrustData.csv` files in which tabular empirical data is entered.

To apply the patch, the files are copied into the root folder of the CRT.

B Appendix: The UMN CRT Template

The following template facilitates the entry of empirical data as inputs to a Monte Carlo simulation produced by the patched CRT. The template was designed so that the U of M Rocket Team can better leverage its test data in simulations (e.g. wind-tunnel data) rather than use analytical methods like Barrowman Analysis. Example values are assigned in this template.

```
function [headers, RDT] = UMNCRTTemplate
%% READ BEFORE USING
%%% This template walks the user through the process of entering empirical
%%% data for a Monte Carlo simulation of a rocket flight using the
%%% Cambridge Rocketry Toolbox (CRT) UMN Patch. Accompanying this template
%%% is a file named DragControl.m. In this file, a user-defined control
%%% algorithm whose inputs are time, drag coefficient, altitude, and ascent
%%% velocity and whose outputs are an updated drag coefficient can be
%%% constructed. This code will run whenever the patched CRT is used.
%%% Please consult the Cambridge Rocketry Toolbox manual for more detailed
%%% information.
```

```
%% Specify the file-path of the engine thrust data.
%%% This is the file-path string of an engine thrust data file. The file
%%% must be plain text with comma- or space-separated values. The first
%%% column should consist of times in seconds. The second column should
%%% consist of corresponding thrust values in Newtons. The table can have
%%% any number of rows. The website http://www.thrustcurve.org/ is a useful
%%% resource here.
thrustDataFilePath = '../thrustData.csv';

%% Specify the file-path of the drag coefficient data.
%%% This is the file-path string of a drag coefficient data file. The file
%%% must be plain text with comma- or space-separated values. It should
%%% have the following structure:
%%% 0       Re_1  Re_2  Re_3  ...
%%% alpha_1 CD_11 CD_12 CD_13 ...
%%% alpha_2 CD_21 CD_22 CD_23 ...
%%% alpha_3 CD_31 CD_32 CD_33 ...
%%% .       .     .     .
%%% .       .     .     .
%%% .       .     .     .
%%% where CD_ij is the drag coefficient at angle of attack alpha_i (in
%%% radians) and Reynolds number Re_j (with rocket body length as the
%%% characteristic dimension). The table must contain two or more
%%% observations and alpha_1 must be 0.
dragDataFilePath = '../dragData.csv';

%% Specify the file-path of the atmospheric data.
%%% This is the file-path string of an atmospheric data file. The
%%% file must be plain text with comma- or space-separated values. The
%%% first column should contain altitudes from 0m up to some unspecified
%%% high altitude. Columns 2 to 6 contain corresponding data for the
%%% following atmospheric variables:
%%% Column 2: Easterly wind component (m/s),
%%% Column 3: Northerly wind component(m/s),
%%% Column 4: vertical wind component(m/s),
%%% Column 5: atmospheric density (kg/m3 ),
%%% Column 6: atmospheric temperature (K).
%%% The file must have one or more rows.
atmoDataFilePath = '../atmoData.csv';

%% Specify the pre-ignition rocket mass.
%%% This is the total mass of the rocket prior to
```

```
%%% engine ignition.
mass0 = 2.550;  % kg

%% Specify the post-burnout rocket mass.
%%% This is the total mass of the rocket after its fuel
%%% has been ejected.
mass1 = 2.414;  % kg

%% Specify the pre-ignition center of mass.
%%% This is the distance between the tip of the nose
%%% cone and the rocket's center of mass prior to engine ignition.
cm0 = 0.77;    % m

%% Specify the post-burnout center of mass.
%%% This is the distance between the tip of the nose
%%% cone and the rocket's center of mass after its fuel has been ejected.
cm1 = 0.694;    % m

%% Specify the pre-ignition inertia tensor.
%%% This is a 3 by 3 matrix representing the inertia tensor of the rocket
%%% prior to engine ignition.
inertia0 = [0.01 0 0; 0 0.31032 0; 0 0 0.31032];   % kg*m^2

%% Specify the post-burnout inertia tensor.
%%% This is a 3 by 3 matrix representing the inertia tensor of the rocket
%%% after its fuel has been ejected.
inertia1 = [0.01 0 0; 0 0.31032 0; 0 0 0.31032];   % kg*m^2

%% Specify the rocket body length.
RBL = 1.41;    % m

%% Specify the center of mass to nozzle exit distance.
%%% This is the average of the pre-ignition and post-burnout distances
%%% between the center of mass of the rocket and the nozzle exit.
lcn = 0.683;    % m

%% Specify the rocket center of mass to engine center of mass distance.
%%% This is the average of the pre-ignition and post-burnout distances
%%% between the center of mass of the rocket and the center of mass of the
%%% fuel.
lcc = 0.55;    % m
```

```
%% Specify the center of pressure to nose cone tip distance.
%%% This is the distance between the rocket center of
%%% pressure and the nose cone tip.
Xcp = 0.845;    % m

%% Specify the frontal cross-sectional area of rocket body.
Ar = 0.008107;     % m^2

%% Specify the coefficient of normal force.
%%% This is the derivative of the normal force
%%% coefficent with respect to angle of attack.
Cna = 8.71;

%% Specify the launch rail length.
Rl = 1.7;     % m

%% Specify the launch rail angle of declination.
Ra = 0;     % deg

%% Specify the launch rail bearing.
%%% This is the direction in which the launch rail points in degrees from
%%% north.
Rb = 0;     % deg

%% Specify parachute characteristics.
%%% From the CRS manual:
%%%% "If the rocket uses a single parachute deployment at apogee then
%%%% paratab is a two-element array containing the coefficient of drag of
%%%% the parachute, and the area [m^2] of the parachute respectively, e.g.
%%%% [Cd,Ap]. If the rocket is using a dual deploy system then paratab has
%%%% four elements and the 3rd and fourth elements contain the coefficient
%%%% of drag, and the area of the second parachute, e.g. [Cd1,Ap1,Cd2,Ap2].
%%%% If the first stage of the rocket recovery is drogueless then the first
%%%% two elements of paratab should approximate the drag on the separated
%%%% rocket. As a good first approximation a coefficient of drag of 0.6 and
%%%% an area equalling the side-view planfom area of the rocket can be
%%%% used."
paratab = [0, 0, 0.3, 0.01];

%% Specify main chute deployment altitude.
%%% This should be set to 0 if the rocket has a
%%% single-deploy system.
```

```
altpd = 200;  % m

%% Specify the number of Monte Carlo iterations to run.
%%% This must be at least 2.
noi = 30;

%% Specify Monte Carlo coefficients. Note that all simulations are run with
%% a stochastic wind turbulence model. See rocketflight_monte.m in the CRT
%% folder for implementation details.
Cdms=0.0;   % Random mulstiplier for drag coefficient
Cpms=0.0;   % Random multiplier for center of pressure
CNms=0.0;   % Random multiplier for normal force coefficient
Cdpms=0.0;  % Random multiplier for parachute drag coeficient
Cddms=0.0;  % Random multiplier for drogue drag coefficient


%%% ########################################################################## %%%
%%% ########################## END OF TEMPLATE ########################### %%%
%%% ########################################################################## %%%
%%% What follows should only be edited in rare circumstances by
%%% knowledgable users of CRS.

%% Build rocketflight inputs.
%%% Consult 3.3 of the CRS instruction manual for more information on what
%%% follows.

%%% Import thrust data, drag data, and atmospheric data.
thrustData = importdata(thrustDataFilePath);
dragData = importdata(dragDataFilePath);
atmoData = importdata(atmoDataFilePath);

%%% Build INTAB1.
t = thrustData(:, 1);
t0 = t(1);
t1 = t(end);

INTAB1(:, 1) = t;
INTAB1(:, 2) = thrustData(:, 2);

%%%% Linearly interpolate mass and inertia through engine burn.
burnInterp = @(v) interp1([t0, t1], v, t);
```

```
INTAB1(:, 3) = burnInterp([mass0, mass1]);
INTAB1(:, 4) = burnInterp([inertia0(1, 1), inertia1(1, 1)]);
INTAB1(:, 5) = burnInterp([inertia0(2, 2), inertia1(2, 2)]);
INTAB1(:, 6) = burnInterp([inertia0(3, 3), inertia1(3, 3)]);
INTAB1(:, 7) = burnInterp([inertia0(1, 2), inertia1(1, 2)]);
INTAB1(:, 8) = burnInterp([inertia0(1, 3), inertia1(1, 3)]);
INTAB1(:, 9) = burnInterp([inertia0(2, 3), inertia1(2, 3)]);
INTAB1(:, 10) = burnInterp([cm0, cm1]);

%%%% Estimate the mass expulsion rate.
Mdot = (INTAB1(end, 3) - INTAB1(1, 3)) / (t1 - t0);

INTAB1(:, 11) = Mdot * (lcn^2 - lcc^2);

%%%% Build INTAB3
INTAB3 = [Cna, Xcp];

%%%% Build INTAB
INTAB = {INTAB1, dragData, INTAB3, [RBL, Ar], paratab};

%%% Simulate flight.
[Ascbig,Desbig,Landing,Apogee] = rocketflight_monte(...
    INTAB, atmoData, altpd, Rl, Ra, Rb, noi, ...
    'CDmult', Cdms, 'CPmult', Cpms, 'CNmult', CNms, ...
    'CDPmult', Cdpms, 'CDDmult', Cddms);
[headers, RDT] = flight_variables('flightData', Ascbig{1}, ...
    INTAB, atmoData, Rl, Ra, Rb);
AnimateRocket(Ascbig{1});


end
```

C Appendix: MATLAB Drag Controller Model

The following drag control algorithm was incorporated into the CRT patch in order to serve as a basic example for the U of M Rocket Team. It can be modified so that users may specify a custom drag control scheme to be run as part of the CRT ascent trajectory simulation.

```
function [CdOut] = DragControl(tt, Cd, z, v, holder)
%%% This function can be used to specify a drag control scheme. It is run
```

```
%%% whenever a simulation is run.
%%% tt is the time from lift off (s).
%%% Cd is the drag coefficient.
%%% z is the altitude (m).
%%% v is the ascent velocity (m/s).
%%% holder is a cell array which can be used to store values whose states
%%% are preserved outside of the main update loop (the user may wish to
%%% investigate the persisent keyword as well).

    function zmax = maxHeight(zi, vi)
        m = 2.255;
        g = 9.81;
        A = 0.008107;
        rho = 1.4;
        vt = sqrt(2*m*g/(Cd*A*rho));

        zmax = zi + (vt^2 / (2*g)) * log((vi^2 + vt^2)/vt^2);
    end

target = 337;
depDelay = 0.4;

%%% holder{1} stores the time of last brake deployment
%%% holder{2} stores drag brake state.

if size(holder, 1) == 0
    holder{1} = tt - depDelay;
    holder{2} = 0;
end

if tt - holder{1} >= depDelay
    holder{2} = maxHeight(z, v) >= target;
    holder{1} = tt;
end

if holder{2} == 1
    CdOut = Cd + 0.2;
else
    CdOut = Cd ;
end
end
```

D Appendix: Additional Figures

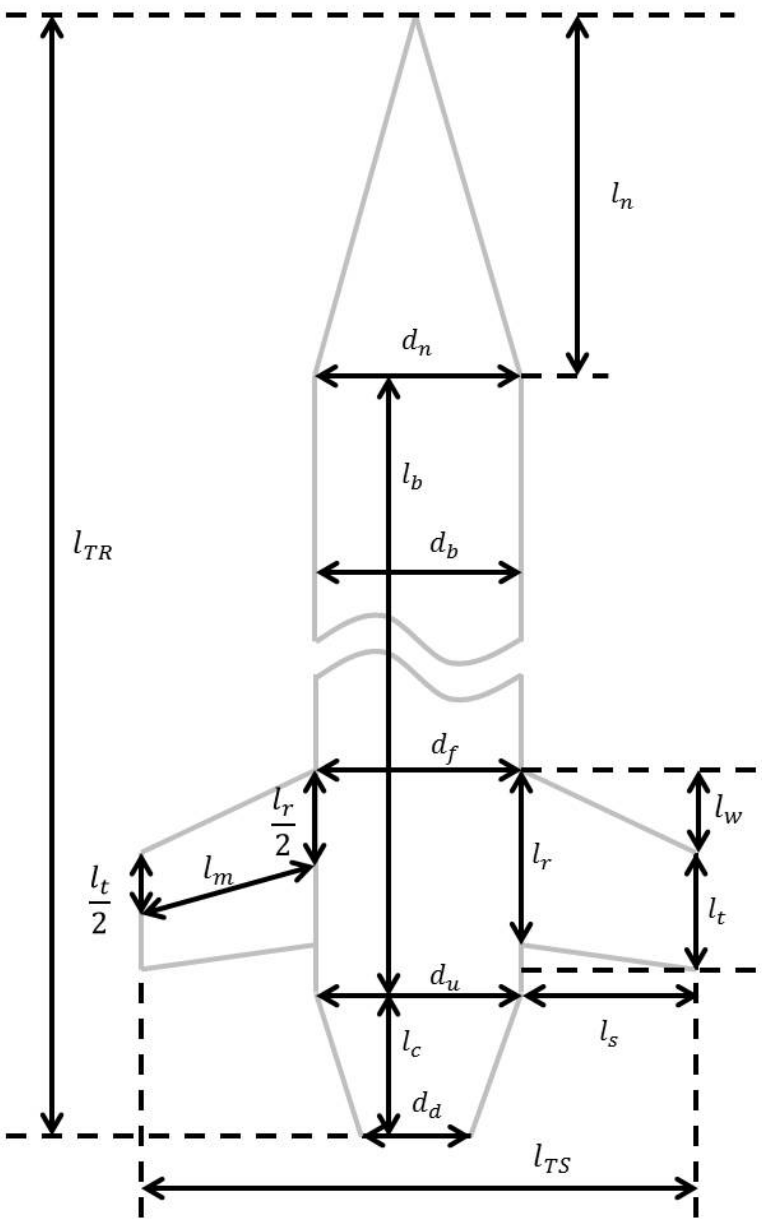

Figure 6: Diagram of parameters used to model the DATCOM model.

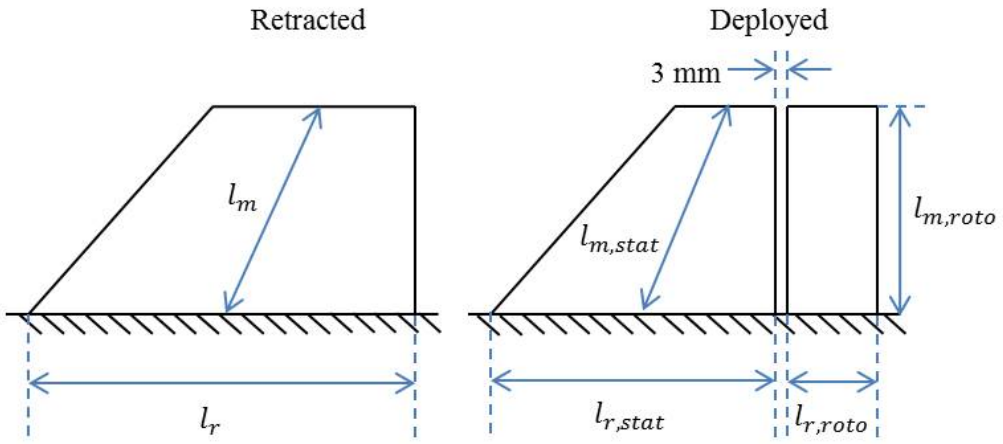

Figure 7: Diagram of analytical model used for retracted and reployed configurations for fins. The small secondary fin in the deployed configuration is treated as an additional fin, resulting in additional drag.

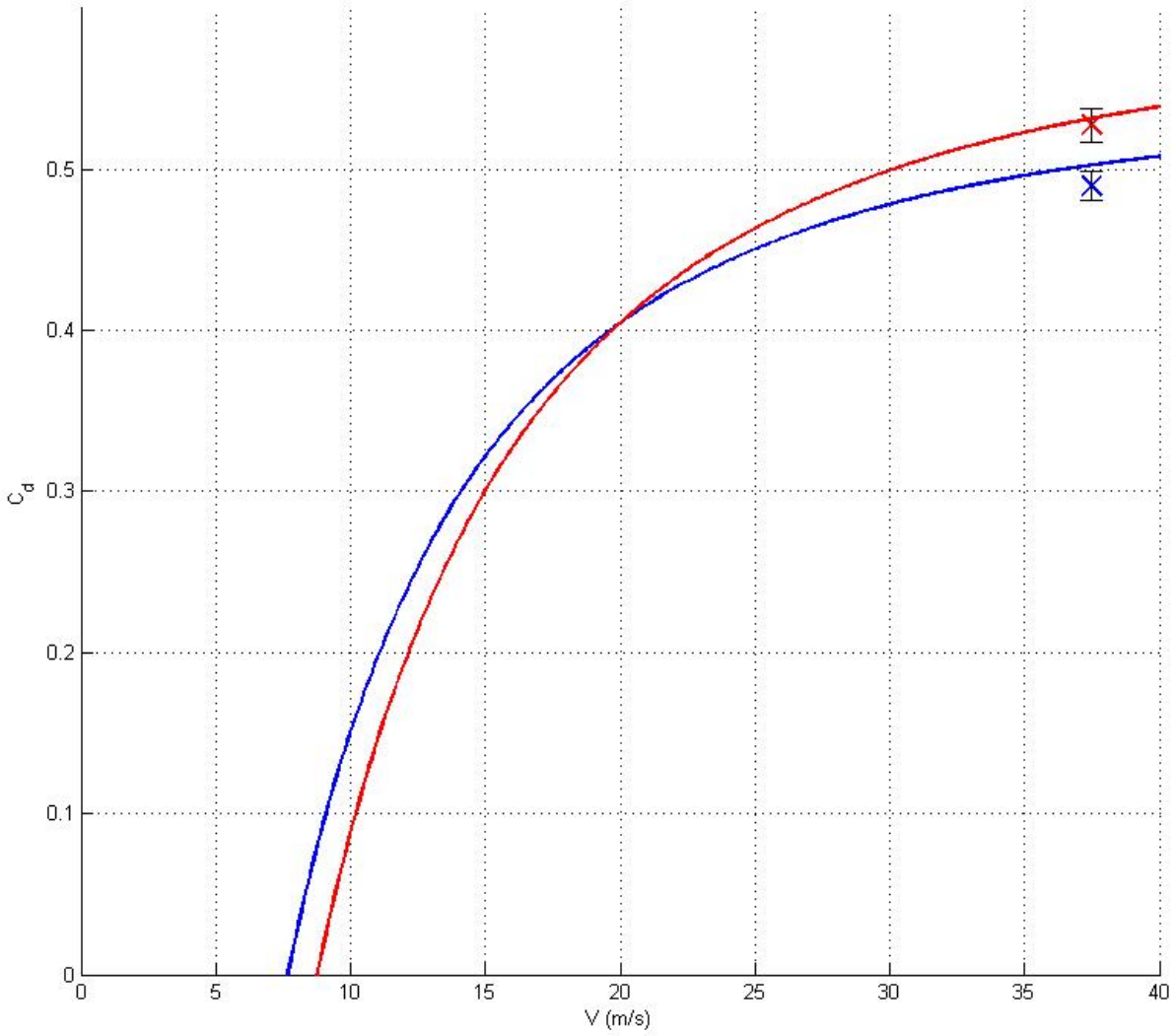

Figure 8: Results from the wind tunnel plotted with the analytical solutions derived from Equation (15).

Figure 9: Wind tunnel testing of full size vehicle in Akerman Hall's closed return tunnel.

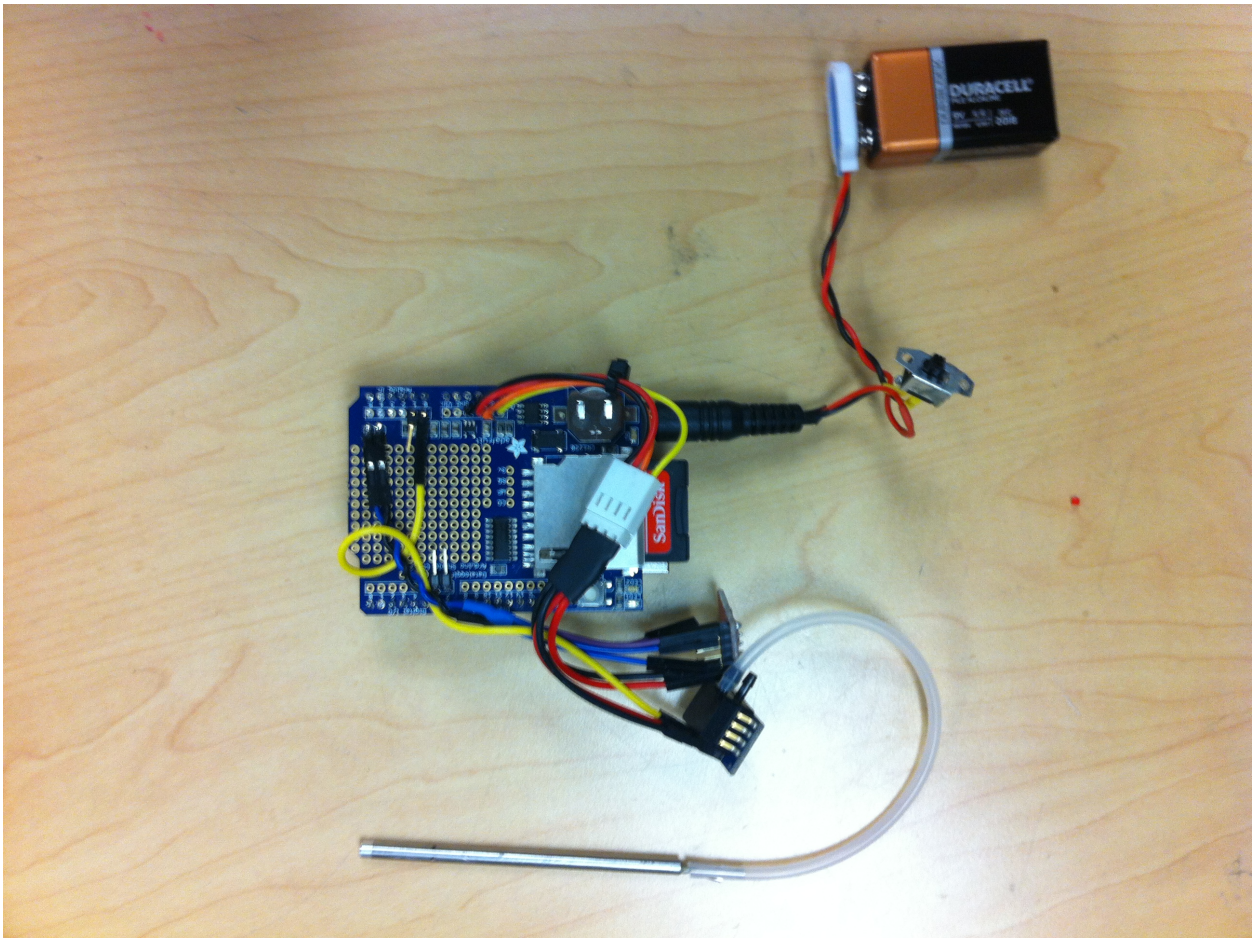

Figure 10: Arduino with pitot probe (MPXV7002), barometric senor (BMP085), and Adafruit data logging shield. The entire system, except for servo power, was powered from a standard 9 volt battery.

Figure 11: An inertia swing test was conducted to determine the inertia tensor for the vehicle.

Figure 12: 'Gertrude' assembled prior to launch.

Figure 13: Landing site after second launch that resulted in a broken motor mount and the scrubbing of the third launch.

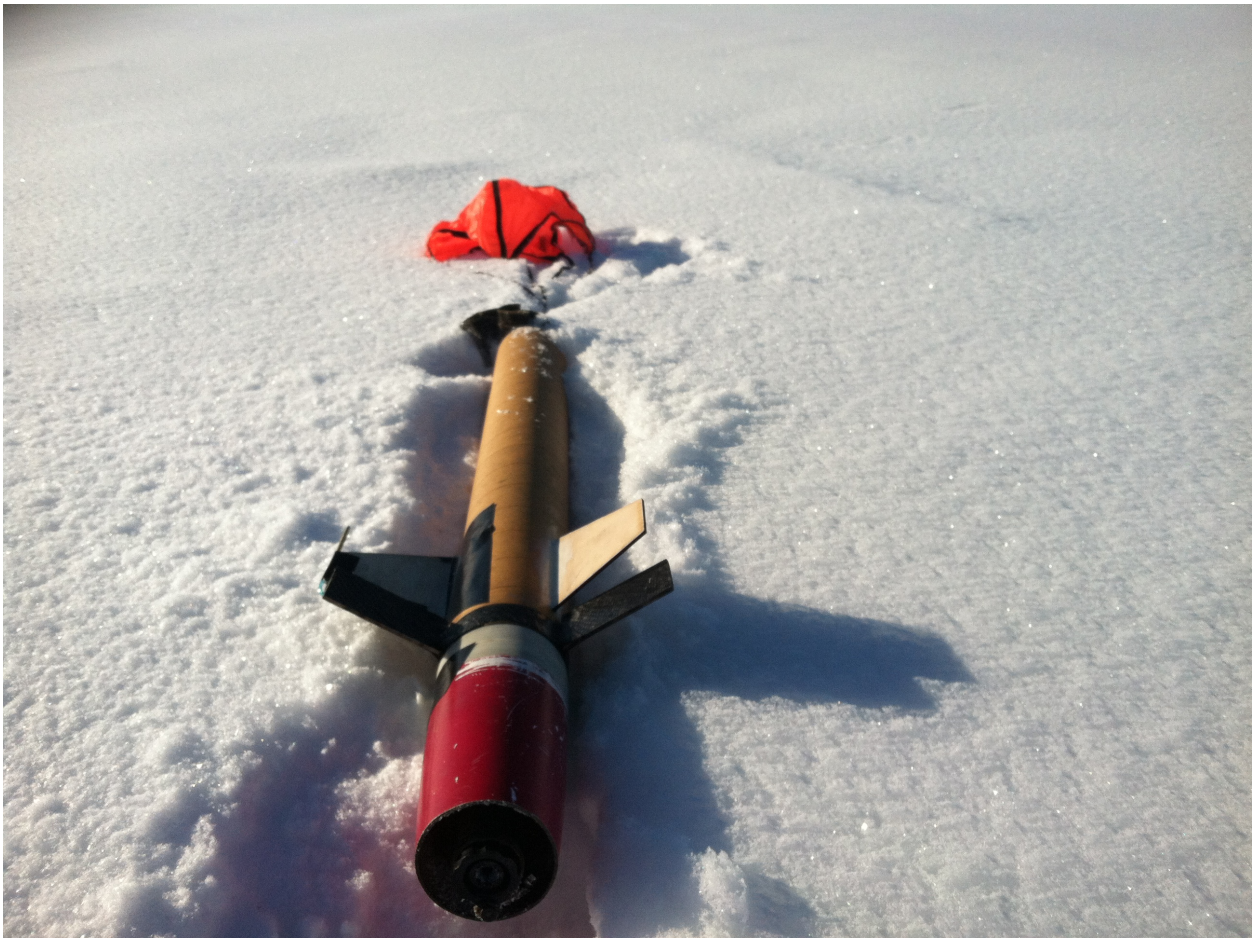

Figure 14: Landing site after final launch with a successful and safe recovery.

References

[1] Box. S. Bishop. C.M. and Hunt. H., 2009, "Estimating the dynamic and aerodynamic parameters of passively controlled high power rockets for flight simulation", Cambridge Rocketry, pp 8-14.

[2] Mandell, G.K., Caporaso, G.J., Bengen, W.P., 1973, Topics in Advanced model Rocketry. MIT press classics.

[3] Blakelock, J., 1995 Aerodynamics, aeronautics, and flight mechanics Wiley, New Jersey.

[4] Theodore, K., 1943 "Turbulence and Skin Friction", J. of the Aeronautical Sciences, Vol. 1, No 1, pp. 1-20. http://www.cfd-online.com/Wiki/Skin_friction_coefficient

[5] "Motor Statistics" ThrustCurve Hobby Rocketry, http://www.thrustcurve.org/motorstats.shtml

[6] Culp, T., 2011 "Rocket Equations Quick Reference" Rocket Mime http://www.rocketmime.com/rockets/qref.html

[7] Benson, T., 2011, "Flight Equations with Drag," National Aeronautics and Space Administration, http://www.grc.nasa.gov/WWW/k-12/airplane/flteqs.html

[8] http://cambridgerocket.sourceforge.net/download.html

[9] http://openrocket.sourceforge.net/

[10] http://www.rasaero.com/