**Towards Development of a Real-Time Flight Simulation**

Dexter Watkins

ME 5267 – Aerospace Propulsion

Contents

[Thesis 2](#_Toc450213705)

[Introduction 2](#_Toc450213706)

[The Ultimate Goal 2](#_Toc450213707)

[Why is this useful? 2](#_Toc450213708)

[How am I going to get there? 3](#_Toc450213709)

[Background 3](#_Toc450213710)

[Real-Time Systems 3](#_Toc450213711)

[Hardware-in-the-loop 4](#_Toc450213712)

[Matlab, Simulink, and Stateflow 4](#_Toc450213713)

[The Simulation Premise 6](#_Toc450213714)

[Problem Definition 6](#_Toc450213715)

[Specified Properties and Assumptions 6](#_Toc450213716)

[The Matlab Solution 8](#_Toc450213717)

[Results 8](#_Toc450213718)

[The Simulink Translation 9](#_Toc450213719)

[Results 12](#_Toc450213720)

[The Stateflow Solution 13](#_Toc450213721)

[Issues with Matlab port 13](#_Toc450213722)

[Stateflow 13](#_Toc450213723)

[Model Setup 13](#_Toc450213724)

[Results 18](#_Toc450213725)

[Implementation of Real-Time Processing 19](#_Toc450213726)

[Results 20](#_Toc450213727)

[Conclusion 21](#_Toc450213728)

# Thesis

A basic real-time simulation has been developed using Mathworks’ Simulink and Stateflow that will provide the basis for code generation for future distributed real-time systems. The progression from a set of matlab code to this final real-time system is detailed, with comments towards future work.

# Introduction

## The Ultimate Goal

The ultimate goal of this and future work is to develop a hardware-in-the-loop real-time flight dynamics simulation. This simulation will act as testbed for the design and integration of individual real flight systems without having to fabricate ancillary components. In particular, the aim is to evaluate the high level trajectory planning and low level control of a miniature H202 monopropellant thruster.

## Why is this useful?

This real-time system (RTS) will fill in the gap between product design and flight testing. Flight testing is a rare occurrence with a low margin for failure. A catastrophic failure in-flight can lead to a total loss of the flight vehicle and the on-board prototype systems. In extreme cases, the lack of ample flight testing can lead to design flaws or omissions being overlooked and the loss of life when failure occurs during non-test flights. As such, a low-cost platform that can quickly and repeatedly provide a facsimile of a flight environment is of great use. Tests can be repeated with slight changes to system or flight parameters to quickly verify that the system works as intended.

A real life example of the importance of rigorous testing before proceeding to flight tests is the failure of Virgin Galactic’s SpaceShipTwo. On October 31, 2014, SpaceShipTwo broke apart on its fourth flight test. Analysis showed that the plane’s “feathering” system was prematurely initiated. The system was designed to rotate the craft’s twin tail booms upwards to stabilize the craft upon re-entry to earth’s atmosphere. The co-pilot, Michael Asbury, prematurely opened the feathering system at Mach 0.8 instead of the designed Mach 1.4. As such, the vehicle lost control and broke apart two seconds later. Statements from Virgin Galactic placed blame on the lack of safety features or a rigorous control system to prevent pre-mature deployment. It is entirely possible that this accident could be avoided in the future by rigorous hardware-in-the-loop testing of individual flight systems before proceeding to actual flight testing.



Figure 1 Virgin Galactic's SpaceShipTwo disintegrates on ascent due to an unforeseen pilot action

## How am I going to get there?

Development of this simulation platform can be divided into two steps. First, the design and development of a real-time flight dynamics simulation. Second, the fabrication of one (or more) testbeds in which flight hardware can be mounted and integrated with the simulation. The rest of this paper will focus on the first step in this process. In order to do this, a preliminary model will be made using mathwork’s matlab, simulink, and stateflow environments.

Starting from flight simulation code making use of matlab’s built in ordinary differential equation solvers, a simulink model will be made to mirror the matlab code’s behavior. Next, the simulink model will be optimized for the simulink and stateflow design environment. Last, the stateflow/simulink model will be modified to operate as a real-time system.

# Background

## Real-Time Systems

A real-time system describes hardware or software that is subjected to a *real-time* constraint such as waiting for an event or the need to deliver a response by a certain *deadline*. A real-time system must be designed such that delivery of data or control signal is *guaranteed* by its deadline. In a simulation’s context, this means that the system clock must run at the same speed as a real clock. In the context of the low-level control of a system, this means that no unforeseen delays in computation or hardware actuation can exist.

## Hardware-in-the-loop

Hardware-in-the-loop (HIL) is a technique used to develop and test complex, real-time systems. HIL simulation provides a mathematical representation of all dynamic systems that a system, or *plant*, is exposed to. From the plant’s point of view, is it operating in an actual system. This hoax is accomplished by accurate electrical emulations of all sensors and actuators related to the plant that act as the middleman between the plant and the simulation environment.

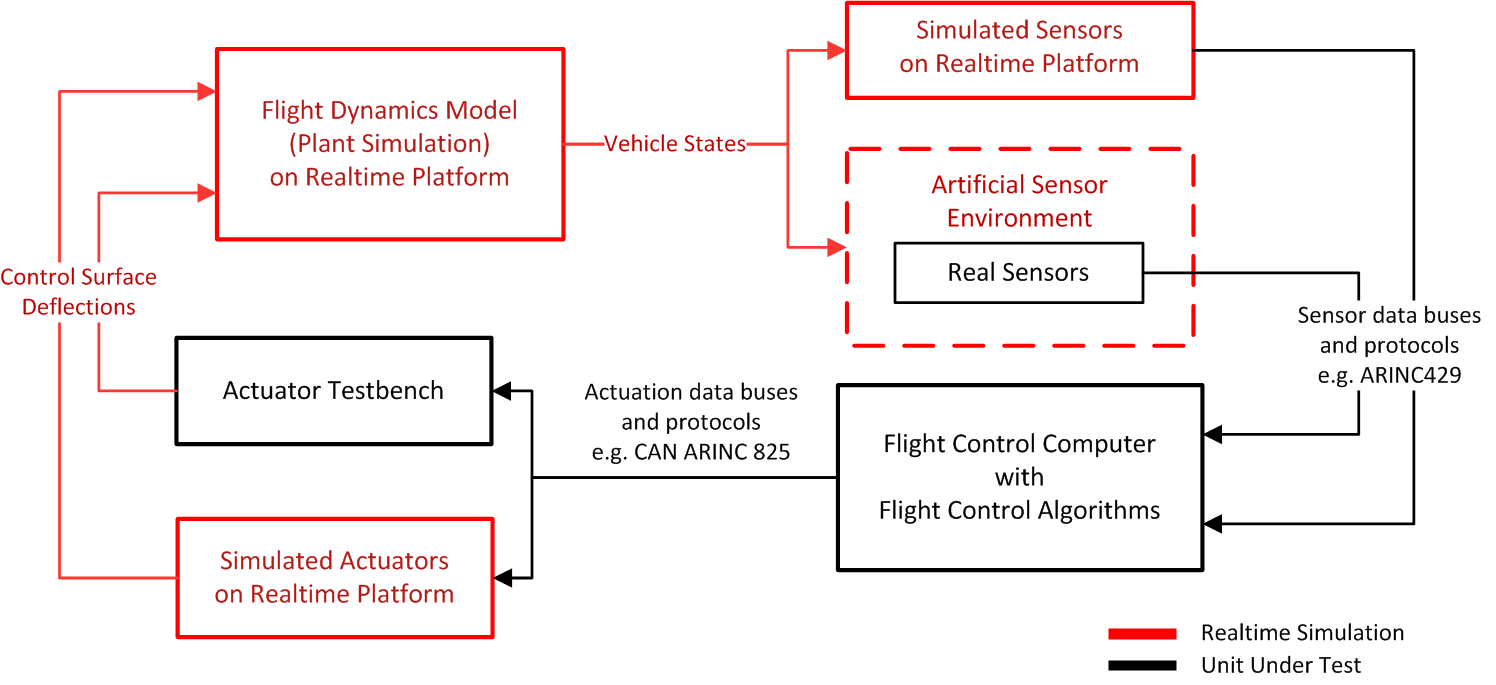


Figure 2 A sample design of HIL flight dynamics simulation

## Matlab, Simulink, and Stateflow

Matlab (Matrix Laboratory) is a platform optimized for computational mathematics. Matlab provides a propriety programming language that allows for the plotting of functions and data, creation and implementation of algorithms, and interaction with/generation of other languages such as C, C++, and Python. Of key importance for this work are matlab’s built in ordinary differential equation (ODE) solvers. Use of these solvers provides a numerical solution to ODEs found in the equations of motion of flight.

Simulink is a graphical programming environment suited for the design of models and simulations. In general, a simulink model is made by taking blocks that perform various functions and connecting their inputs and output to achieve a certain goal. Additionally, simulink can be integrated with matlab to call functions or scripts for when behavior is more accurately or efficiently defined using a non-graphical language.



Figure 3 Simulink model of a wind turbine

Stateflow is mathwork’s embodiment of a finite state machine design environment. It is tightly coupled with simulink, and provides the ability to create state machine and flow charts that directly influence model behavior.

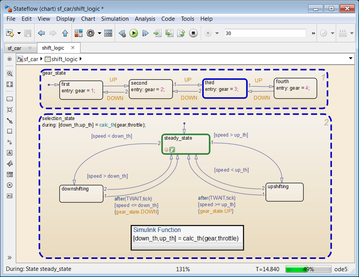


Figure 4 Stateflow States and Transitions

# The Simulation Premise

## Problem Definition

The two-dimensional flight of a high-powered model rocket will be simulated, from sitting on the launch pad to impact with the ground under no parachute. Vertical altitude and drift due to winds will be recorded, as well as the velocity and acceleration at all time steps. Variations in wind speed and air properties due to the atmospheric boundary layer will be taken into consideration.

## Specified Properties and Assumptions

The following tables detail the properties of the rocket and the assumptions made for the simulation.

Table 1 Simulation Properties

|  |  |
| --- | --- |
| Properties | |
| Dry Mass | 15.108kg |
| Cross-sectional Area | 0.015327 m2 |
| Drag Coefficient | 0.45 |
| Motor | Cesaroni L1720 |
| Rail Height | 5m |
| Winds@Ground | 0-10 m/s |

Table 2 Simulation Assumptions

|  |  |
| --- | --- |
| Assumptions | |
| Windspeed |  |
| Density | Standard Atmospheric Model |
| Rocket Orientation | Always parallel to apparent velocity |
| Drag |  |
| Gravity |  |

# The Matlab Solution

ODE45 is used to solve for the flight trajectory in matlab. The specifics of this solution have been detailed in previous class reports, so only a brief overview is given here. ODE45 is a built in function that uses the Runge-Kutta method for numerically approximating differential equations. It gets its name due to its method of approximating a solution using a fourth-order Runge-Kutta approximation, and comparing the solution to a fifth-order approximation. When the residuals between the two approximations fall below a predefined threshold, the fourth order solution is considered valid and presented to the user.

## Results

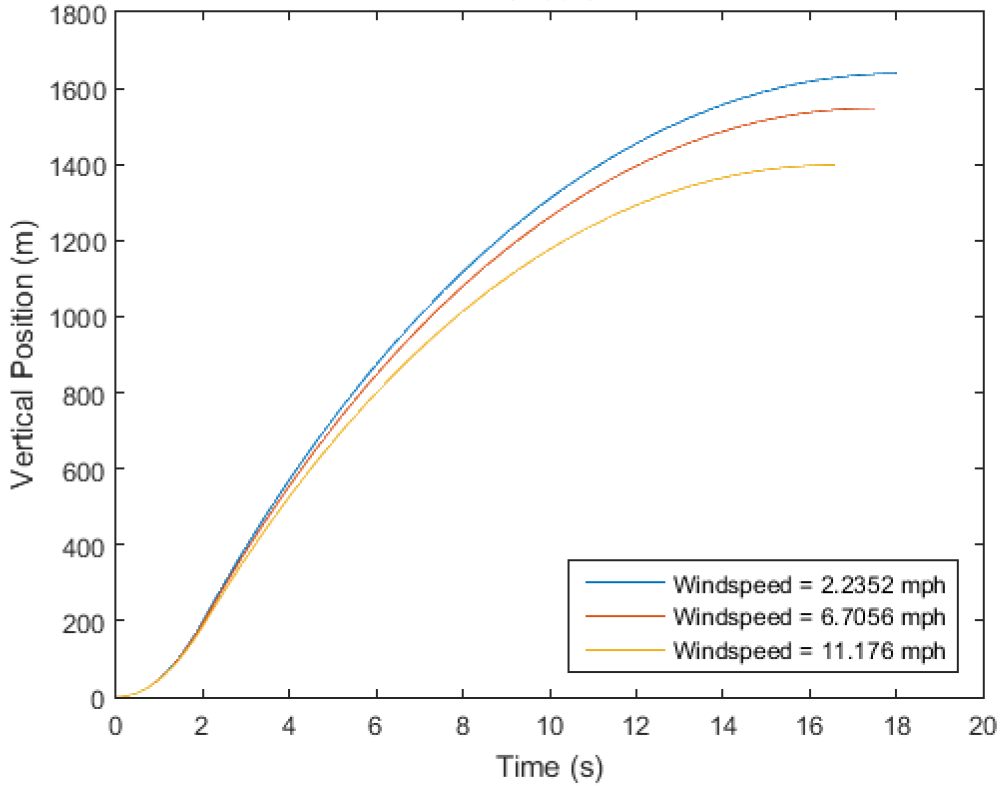


Figure 5 Matlab Solution Altitude vs Time

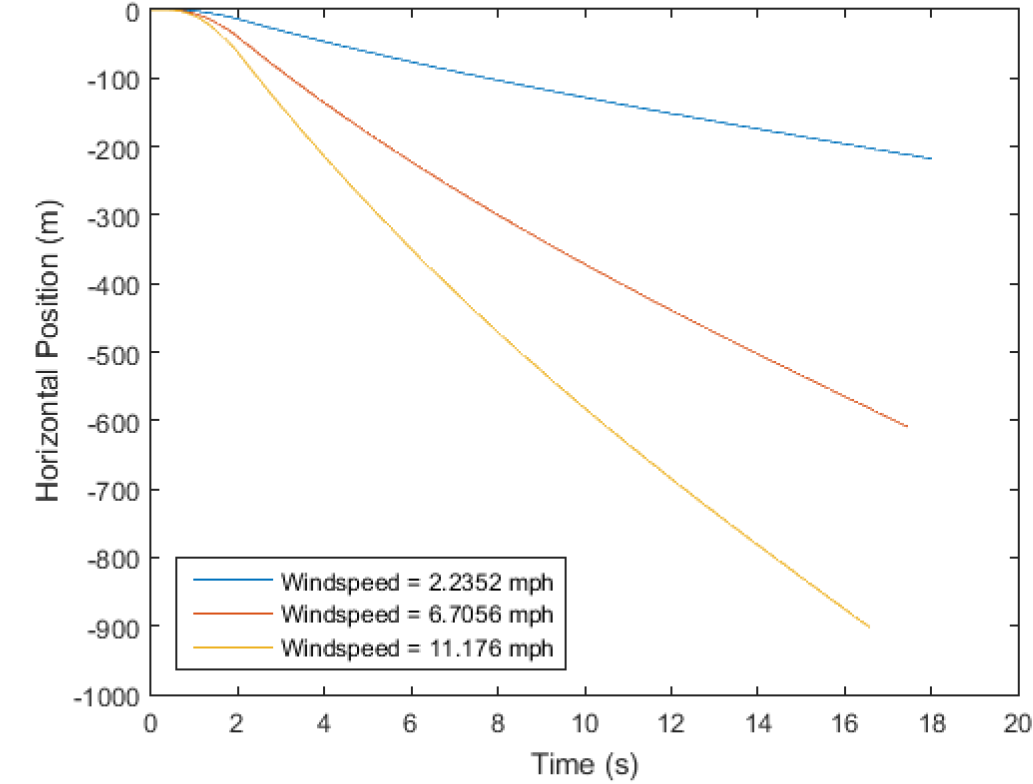


Figure 6 Matlab Solution Drift vs Time

These are the baseline results that future simulations will be compared against for accuracy.

# The Simulink Translation

The first step is to translate the matlab code into a simulink graphical model. No consideration is given for efficiency. Instead, matlab functions written for use with the ODE45 solver are directly embedded into simulink function call blocks. As such, this is a direct port from matlab to simulink.

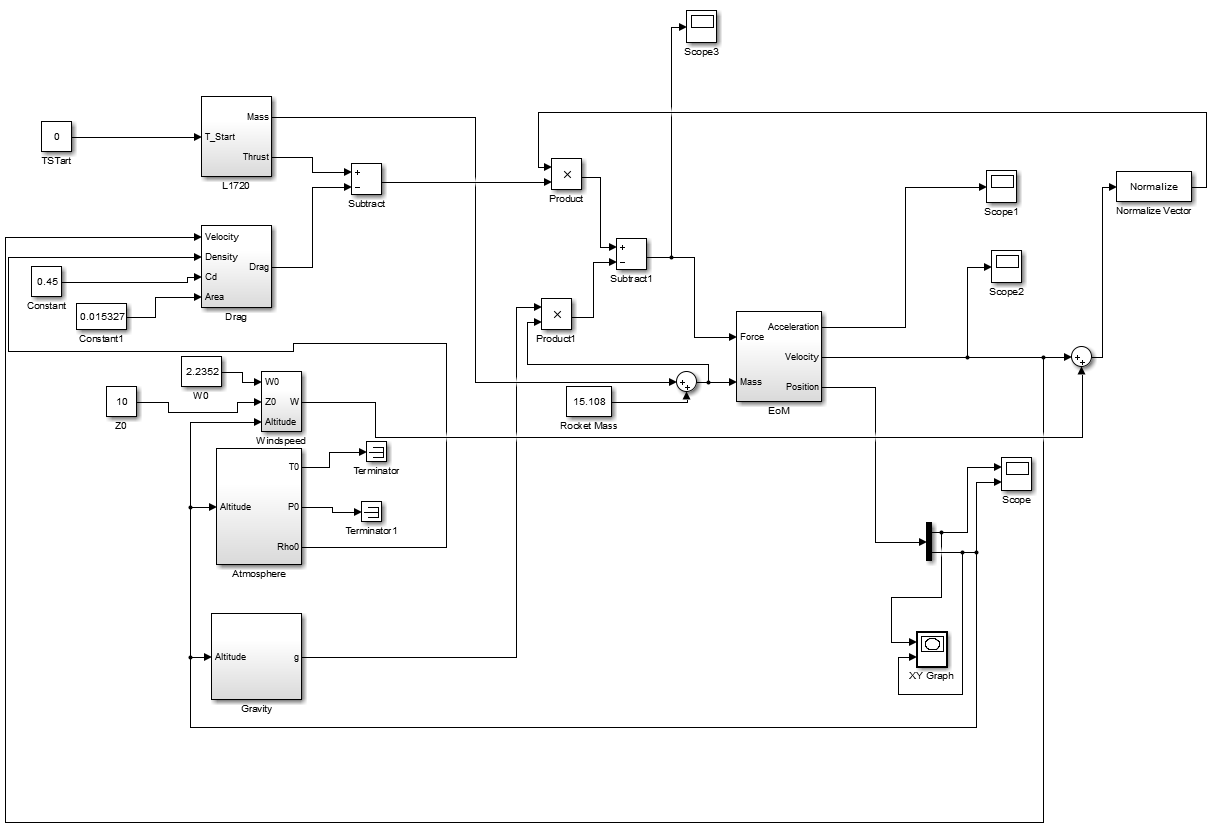


Figure 7 Simulink Block Diagram of Matlab Code

The block diagram shown above describes the data flow within the simulink model. Thrust and drag calculation blocks are used to determine the total force present on the rocket at any given time. Atmosphere and Gravity blocks are used to provide values such as density and the current gravitational acceleration for use in calculating drag and total force on the rocket. Force and mass and then fed into the equation of motion block for the rocket to deliver approximations of acceleration, velocity, and position at each time step.

Since this is a direct port from matlab code, most of the functionality within this block diagram is embedded into an *interpreted matlab function* block. For example, the thrust and mass of the solid rocket motor at any given time is define within the following block diagram.

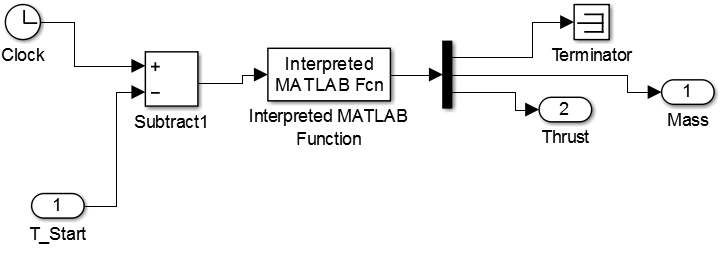


Figure 8 Interpreted Matlab Function for Thrust

This interpreted matlab function block simply calls a matlab function or script and outputs its results. While it saves time since it prevents the thrust curve from having to be redefined within simulink, it adds another layer of complexity to the simulation.

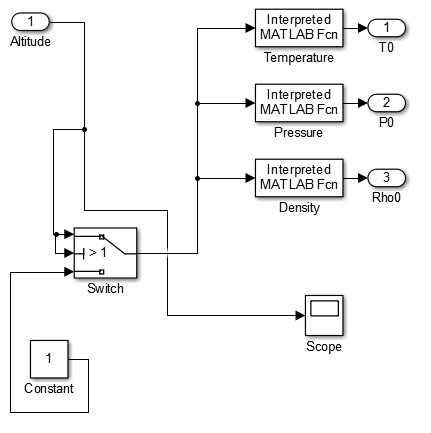


Figure 9 Interpreted mkatlab functions for calculating atmospheric properties

## Results

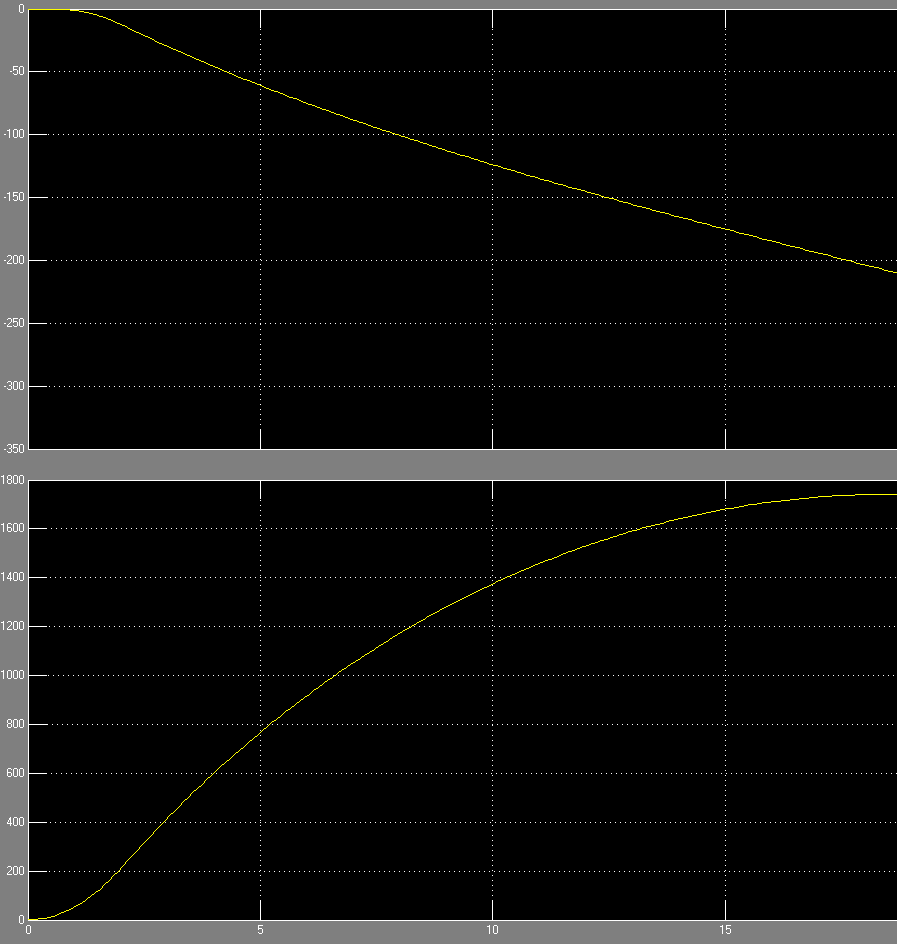


Figure 10 Drift (Top) and Altitude (Bottom) vs Time

As can be seen in the graph of Altitude and Drift vs time, the ported matlab code provides the same solution for the flight trajectory.

# The Stateflow Solution

## Issues with Matlab port

Directly porting from matlab leads to several design flaws and issues that add complexity to the system. First, the use of interpreted matlab function blocks significantly slows down the simulation. For use in a real-time system, this is an absolute no go. Additionally, transitions between specific portions of rocket flight, such as the time it is on the launch rail or in an unconstrained free fall, are much harder to define in simulink than in matlab. This led to needlessly complex block diagrams and workarounds that are almost more trouble than their worth.

## Stateflow

It is situations like this were stateflow shines. Stateflow adds the ability to define *states* of rocket flight and modify the behavior of the simulation accordingly. Independent simulink models can be run during each state of rocket flight. Since any single state will be well-defined, the simulink models run therein can be tailored to the specific behavior that will be seen instead of trying to be a jack-of-all-trades to deal with every situation.

## Model Setup

Utilization of stateflow and rewriting embedded matlab function in terms of simulink blocks leads to the following block diagram.

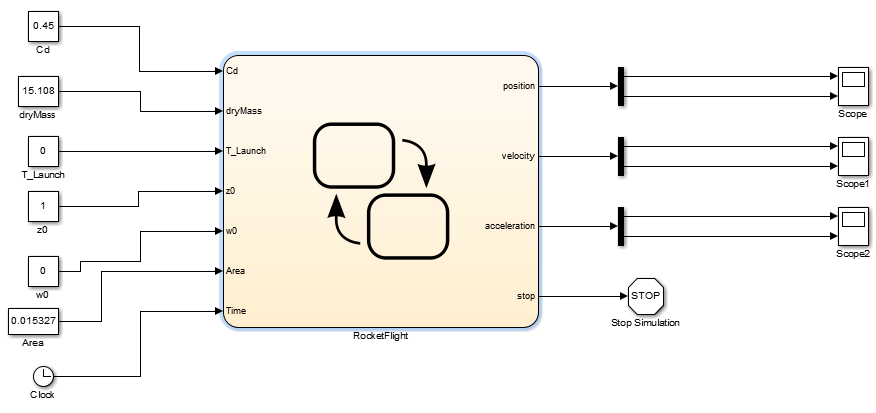


Figure 11 Updated block diagram using stateflow

The heart of the new block diagram is a stateflow chart with following states: *PreLaunch, Guide Rail, Unconstrained, Landed*. Transitions to go from one state to another are defined in terms of time and altitude. The simulation initial starts in the *PreLaunch* state. As soon as the simulation time exceeds the time of motor ignition, stateflow transitions to the *GuideRail* state. Within this state a simplified simulink model is run. Since it is known that the rocket in on the guide rail, simplifying assumptions can be made such as constant density and gravitation acceleration. This leads to a simpler simulink model with less computations that must be performed while on the guide rail. As soon as the rocket’s altitude exceeds 5m stateflow transitions to the *Unconstrained* state. Within this state gravity and density are no longer considered constant and as such the model is slightly more complicated. Once altitude is no longer positive, the rocket is considered to have landed and the *Landed* state ends the simulation.

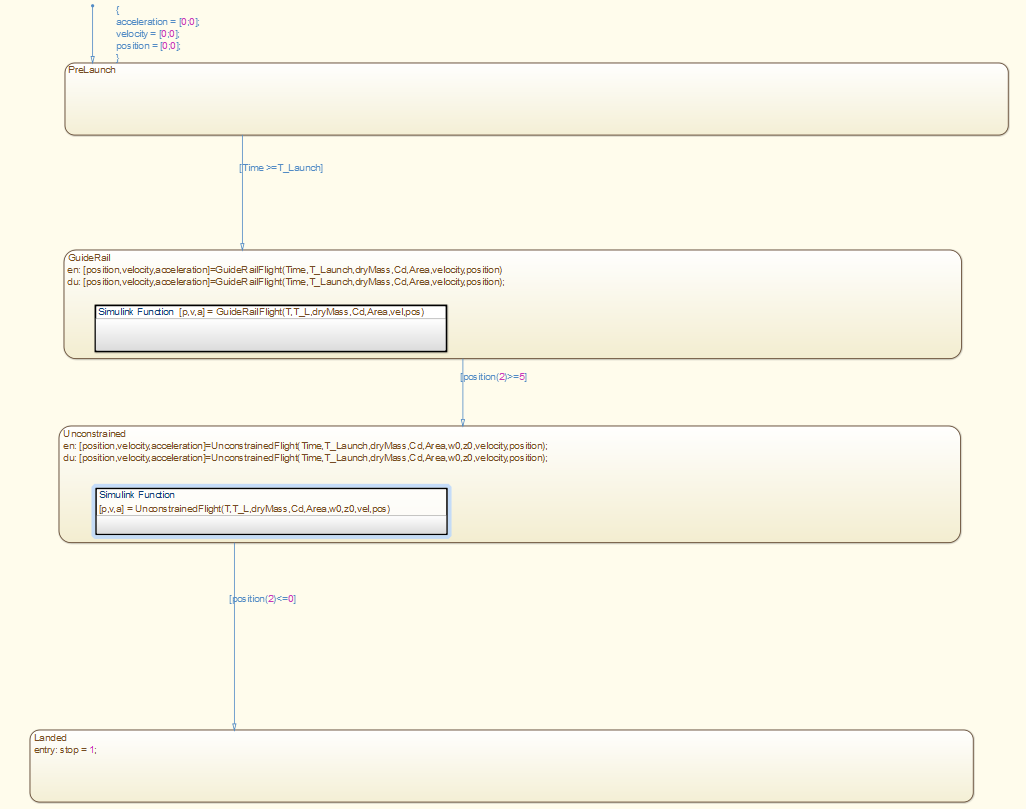


Figure 12 Stateflow States and Transitions

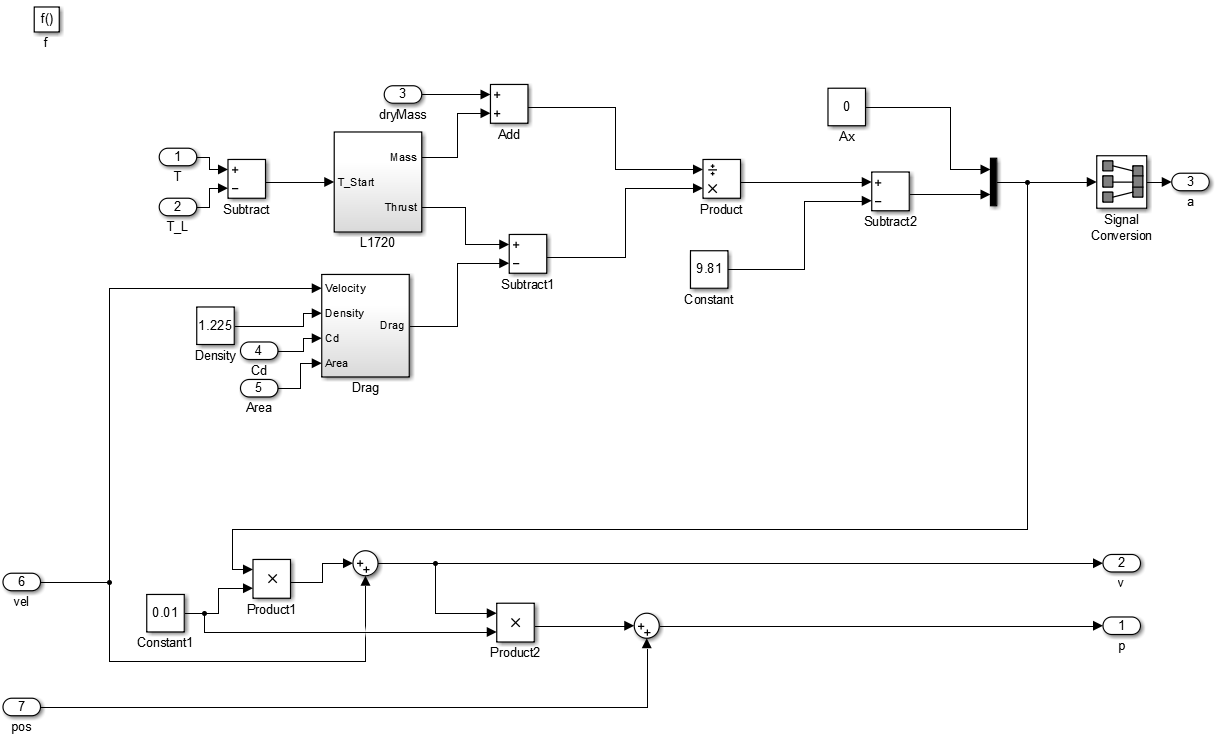


Figure 13 GuideRail State's Simulink Model

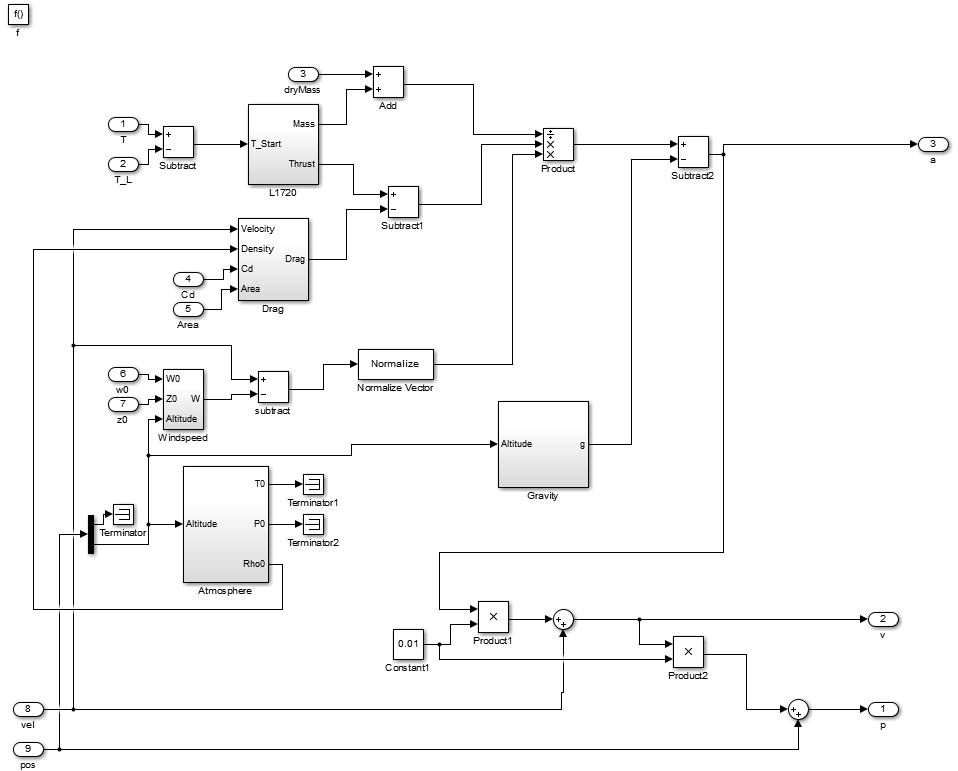


Figure 14 Unconstrained State's Simulink Model

An example of a matlab interpreted function block that was rewritten in terms of simulink blocks is shown below. Instead of calling a matlab function to determine the motor’s thrust curve and mass, lookup tables are used to quickly interpolate the solution.

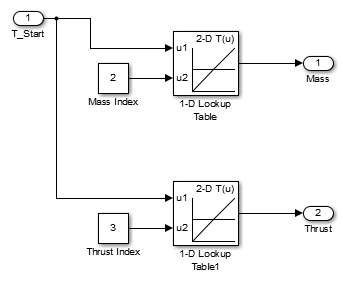


Figure 15 Motor thrust and mass curves redefined in terms of lookup tables

## Results

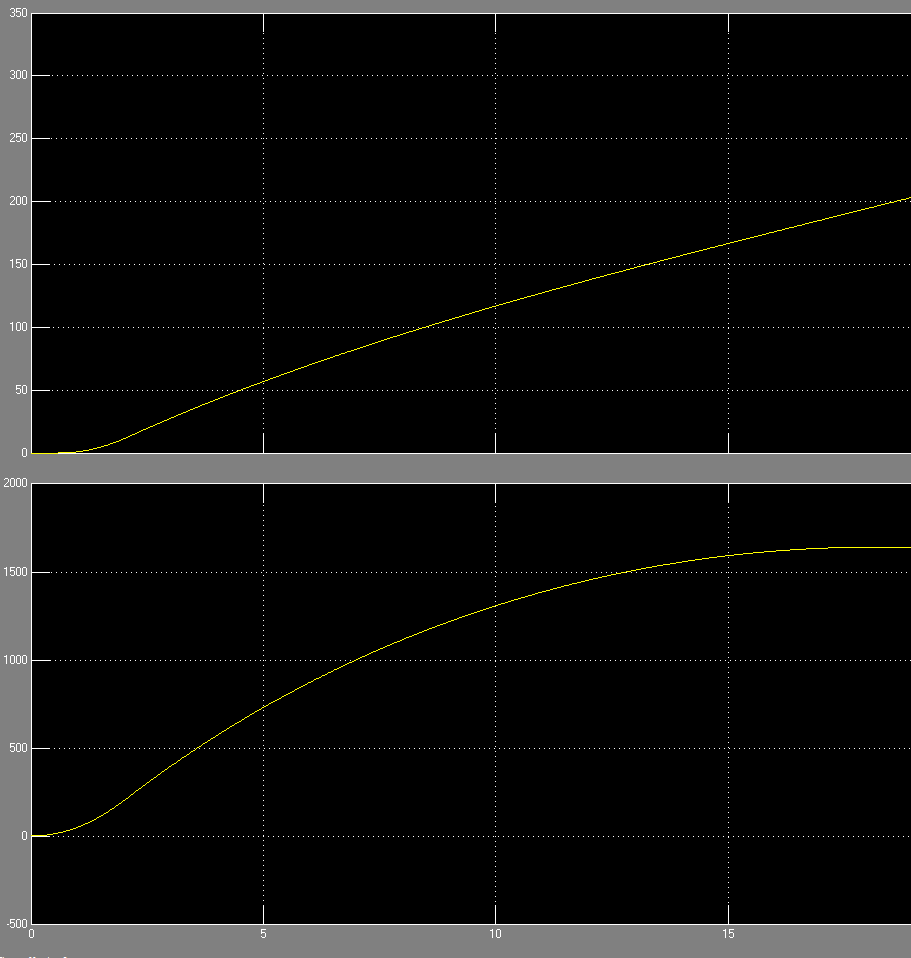


Figure 16 Drift (Top) and Altitude (Bottom) vs Time

As can be seen, the new stateflow model provided the same results. Execution time for this solution was more than halved by smartly implementing a stateflow and simulink solution.

# Implementation of Real-Time Processing

Due to current licensing issues between VUSE and Mathworks, Vanderbilt users do not currently have access to Simulink Real-Time, Mathwork’s version of real-time simulation. An alternative solution is to implement a user-defined block that compares the simulation clock time to real time and waits an appropriate amount of time before executing the next time step.

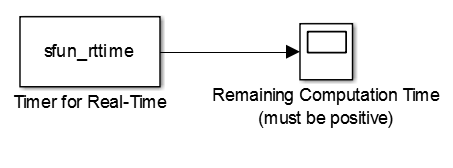


Figure 17 Simulink block for "soft" real-time simulation

While use of this block provides an appearance of real-time simulation, it is what is defined as *soft* real time. *Soft* real-time makes no guarantees of meeting all deadlines, while *hard* real-time is design in such a way as to guarantee these deadlines are met.

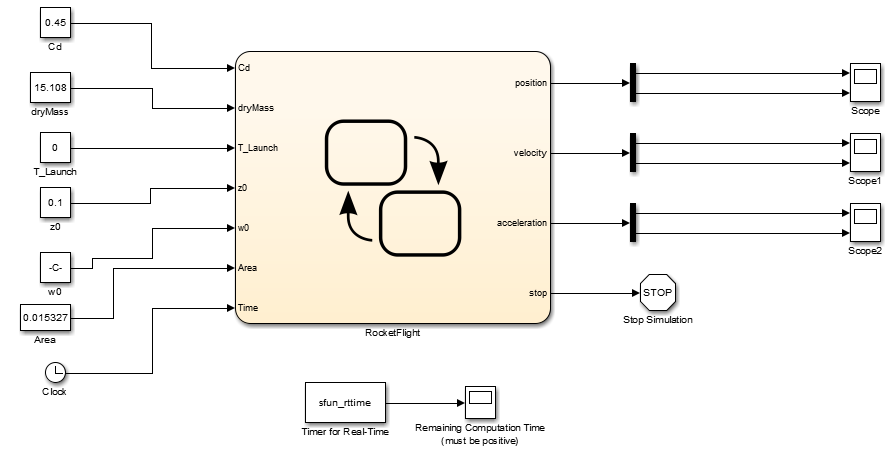


Figure 18 Stateflow with Real-Time Block

## Results

Shown is a chart of remaining computational time at each time step of the real-time simulation. A positive value means that all computations for that time step were performed ahead of the actual clock, allowing the model to meet all deadlines and then “wait” until the next time step. Any steps where the remaining computational time is negative means that the simulation did not meet its deadline and the solution is now slightly out of sync with the real clock. Adjusting the resolution of the simulation’s time step has a corresponding impact on the accurateness of the simulation with respect to real time.

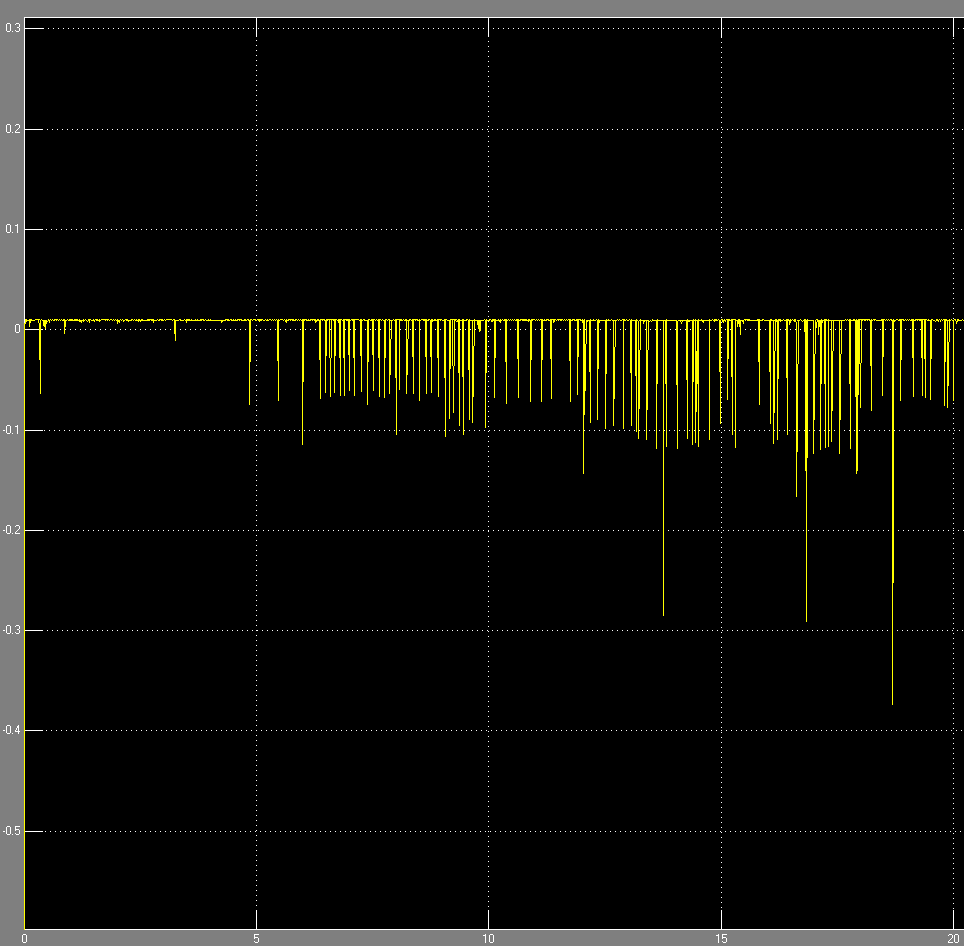


Figure 19 Remaining computational time with a time step of 0.01 seconds

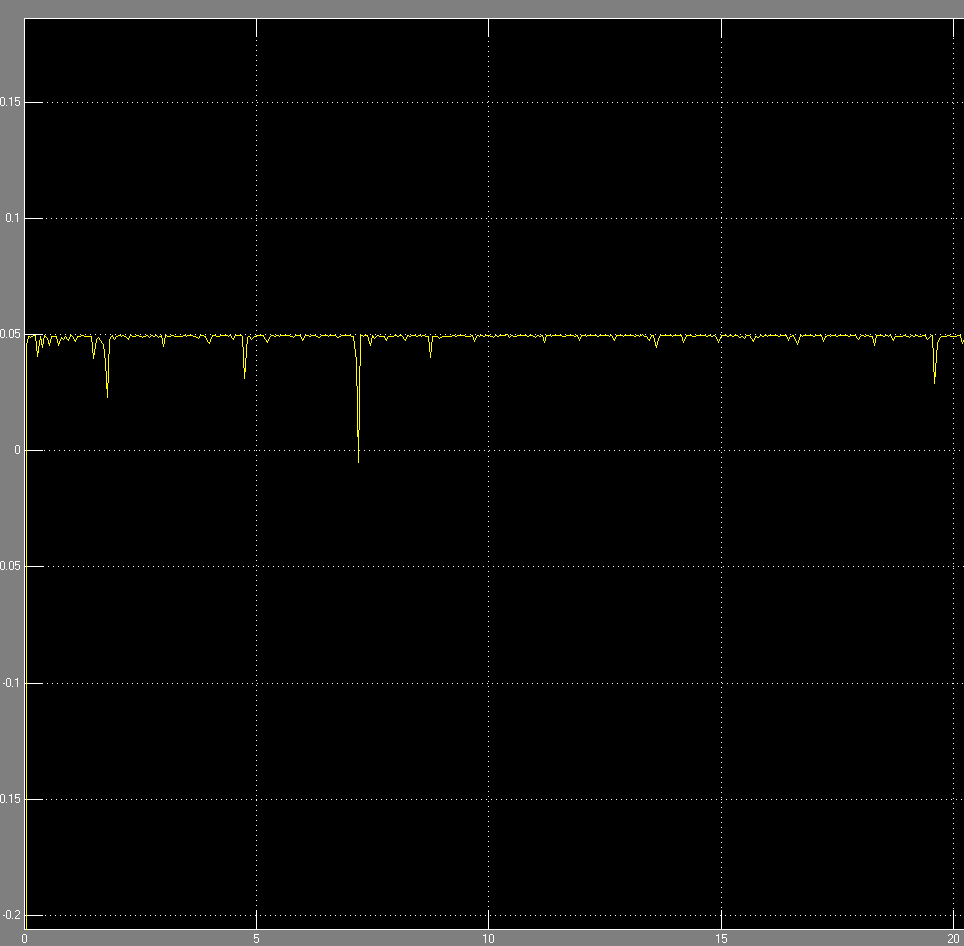


Figure 20 Remaining computational time with a time step of 0.05 seconds

# Conclusion

A valid stateflow and simulink model has been made that accurately reflects flight dynamics and runs in *soft* real time. This lays the basis for future work developing more advanced real time simulation systems for use with hardware-in-the-loop testbeds.