Flight Motion and Control
X4: Foundations

John Pritchett
Freelance Flight Simulation Specialist

Overview

The purpose of this document is to present details about the new flight model that I have developed for Egosoft's X series. I will first present my own history with Egosoft and X, as well as my work on flight simulation leading up to this project. I will then provide details about the model, show what makes it unique relative to other flight models, and then provide examples of how the model is being used in X.

History

My history with Egosoft and its owner, Bernd Lehahn, goes back to 1999, just before the release of their first X game, X: Beyond the Frontier. I had co-authored an old text-based game called Tradewars 2002, which was a pre-Internet multiplayer space trader released in 1987 for Bulletin Board Systems (BBSs). By 1999, I had ported the game to an Internet server, to keep it alive after the decline of the BBS scene. Bernd was a fan of TW, so he contacted me in 1999 to discuss his plans to build a multiplayer X universe following the release of X:BTF. We had some interesting discussions, but ultimately it came to nothing, and we parted ways.

Fast forward to 2014, I found myself in position to join one of my biggest heroes, Chris Roberts, on his then-new spiritual sequel to Wing Commander, Star Citizen. Since I had been working as an indie physics programmer for several years, I jumped at the opportunity to develop CR's latest space combat simulation. If you're interested in details about the model I developed for SC, you can read my final IFCS design document [1] or watch a video covering that document [2].

Fast forward again to 2018, changes in the direction of Star Citizen brought my work on IFCS to an end, and I left the company. During the next two years, I spent my time building a new flight model for my own indie project, a space combat simulation set in the Tradewars universe. I continued to develop that project until, in 2021, Bernd, who had been following my work on Star Citizen, invited me to join the Egosoft team to develop a new flight model for X. By that time I considered X to be a legendary franchise, so I was honored to join the project.

Philosophy of Flight Model Design

Next I would like to say a few words about my goals for flight model development in general. My focus has always been more on flight control systems than on flight physics itself. The depth of the flight physics simulation depends on each game. In the case of X, its design priorities do not warrant a complex rigid body simulation. Instead, a ship's acceleration limits are designer-defined based on performance goals for each ship. But regardless of how the flight characteristics of a ship are defined, whether from a complex rigid body simulation or abstracted into a simple set of accelerations, once those accelerations are defined, my goal is to provide a control system that will translate a player's inputs into the realistic Newtonian accelerations needed to carry out those commands. A player may choose to control accelerations directly, or instead command velocity levels, or even positional goals. However the ship is controlled, whether by player or AI, the goal of my flight model is to simulate a real-world control system that can accomplish those commands, with accuracy and stability, without making simplifying assumptions such as constant-acceleration and infinite jerk, assumptions that sacrifice realism. To that end, my models are based on existing control systems, such as spacecraft and missile guidance and control systems, robotics motion planning and control systems, etc.

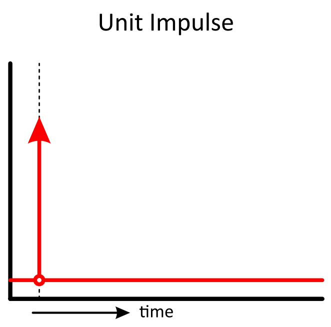
Scope of X Flight Model

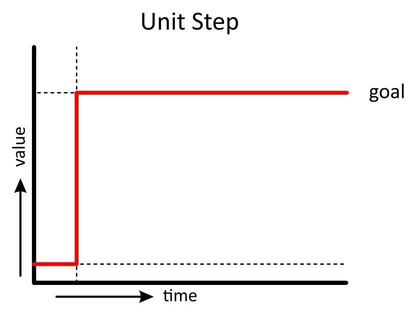
The X flight simulation has been around for many years, and its flight modes are mature and well proven. Therefore, it was not my goal to modify the existing X flight modes, but rather to improve the quality and feel of the ship motion commanded by those modes, to create a more realistic and natural look and feel for the ships, as well as to support greater variation in flight performance and ship characteristics. In order to accomplish this task, rather than use the 2nd order polynomial motion common to most videogames, or even the 3rd order motion used in many real-world systems, I rely on a unique motion algorithm that I've developed called Omega.

Omega Motion

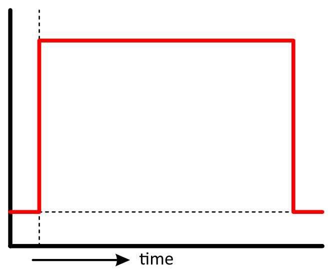
Omega is a nonlinear motion algorithm that models smooth, efficient, stable and precise motion, achieving cinematic-quality motion in realtime. At its core, Omega relies on a particular form of impulse function which, like all impulse functions, can be utilized within a control system to achieve a discrete change in state over a discrete amount of time. To illustrate this point, let's consider a well known impulse function called the Dirac Delta [3], an example of which is the unit impulse which drives an instantaneous change in acceleration in a system based on 2nd order polynomial motion.

The following graphs show the unit impulse as well as its integral or accumulation function, the unit step. Put simply, the impulse drives the change in some value while the step represents the value being changed.

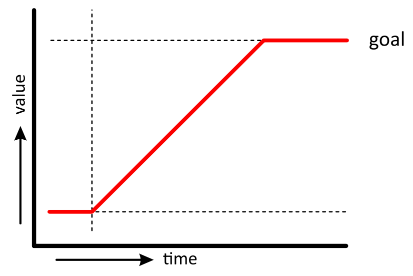




For 3rd order motion, the impulse will be box shaped, and the step will be linear.

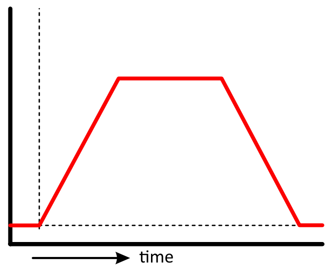


Box Impulse

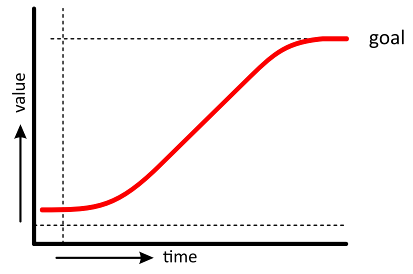


Linear Step

At 4th order, the impulse will be trapezoidal, giving us a desirable s-shaped quadratic step.



Trapazoidal Impulse

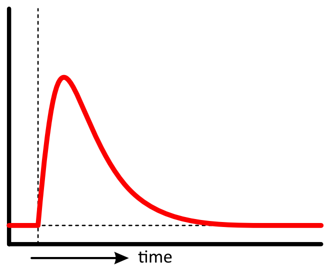


Quadratic Step

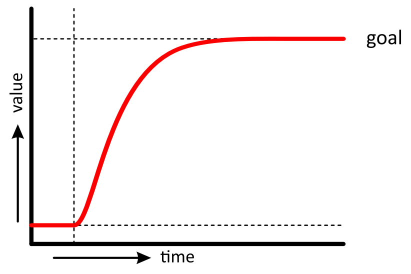
And so on.

Omega Impulse

Unlike polynomial-based motion systems, Omega employs a set of nonlinear impulse functions, the simplest being the well-known critically damped 2nd order harmonic oscillator [4], a.k.a. a damped mass spring system.



Omega Impulse

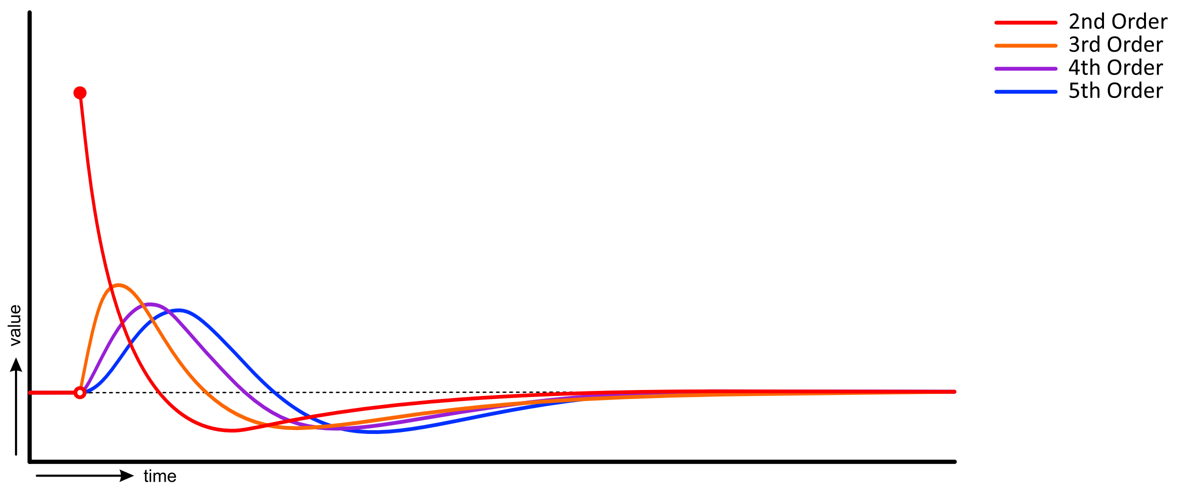


Omega Step

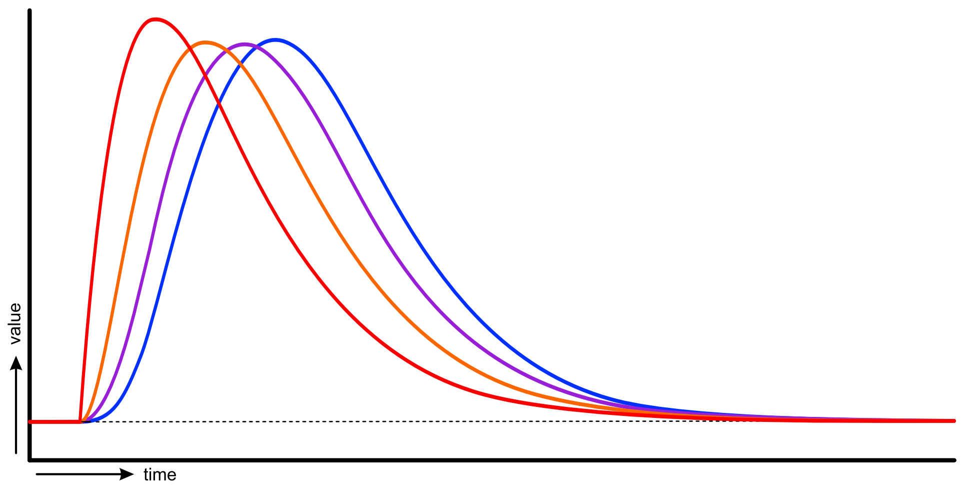
This curve exhibits a number of desirable characteristics. The impulse is front-weighted, making it highly responsive to changes in goal value while smoothly easing into that goal value over time. Also, the step is a smooth s-shape.

The Omega impulse is not limited to the standard 2nd order harmonic oscillator. There are infinitely many higher order impulses. The following graphs show the Omega impulse functions of 2nd order through 5th order, along with their integrals, the step functions, and their derivatives, which I call the wave functions.

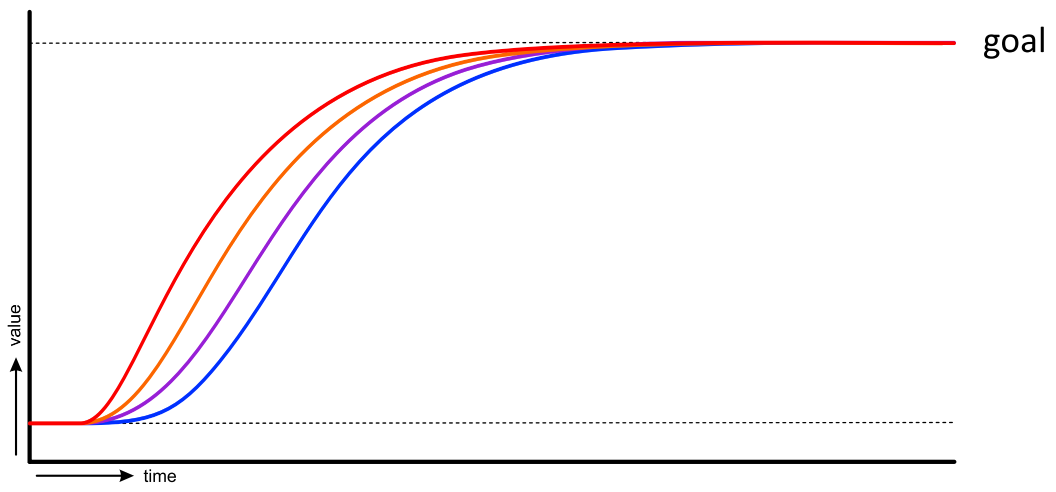
Omega Wave



Omega Impulse

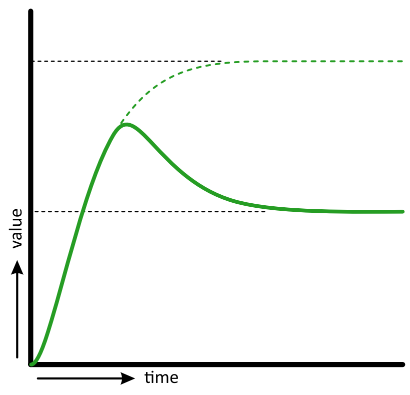


Omega Step

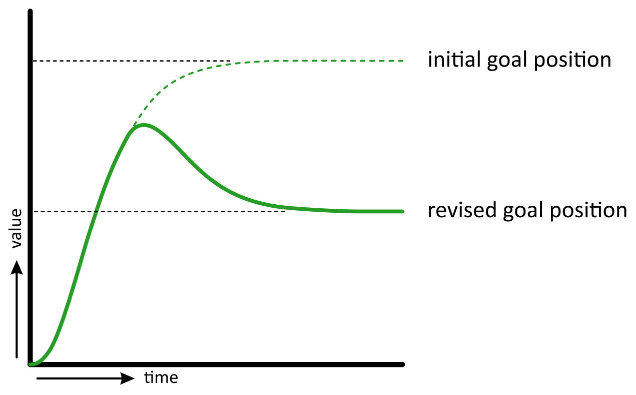


The higher the impulse order, the smoother the step response to a change in goal value, as demonstrated by the following graphs comparing 2nd and 3rd order Omega impulses as they seek one goal, then switch to another.

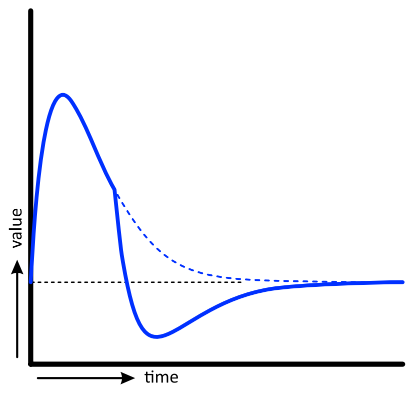
2nd Order Step



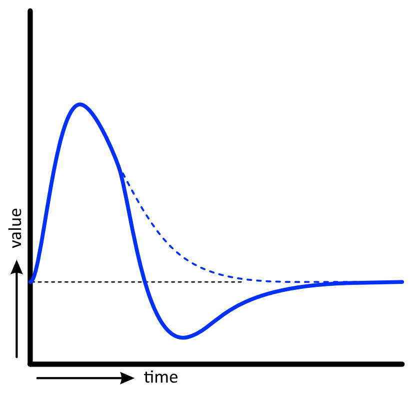
3rd Order Step



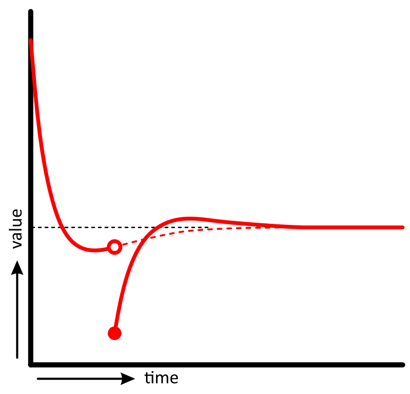
2nd Order Impulse



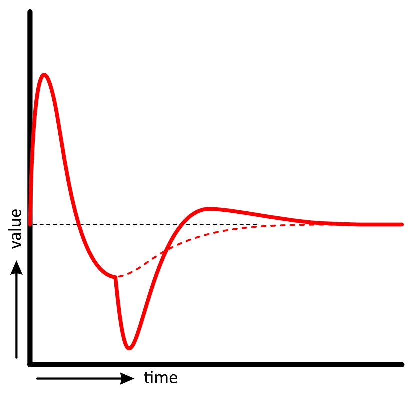
3rd Order Impulse



2nd Order Wave



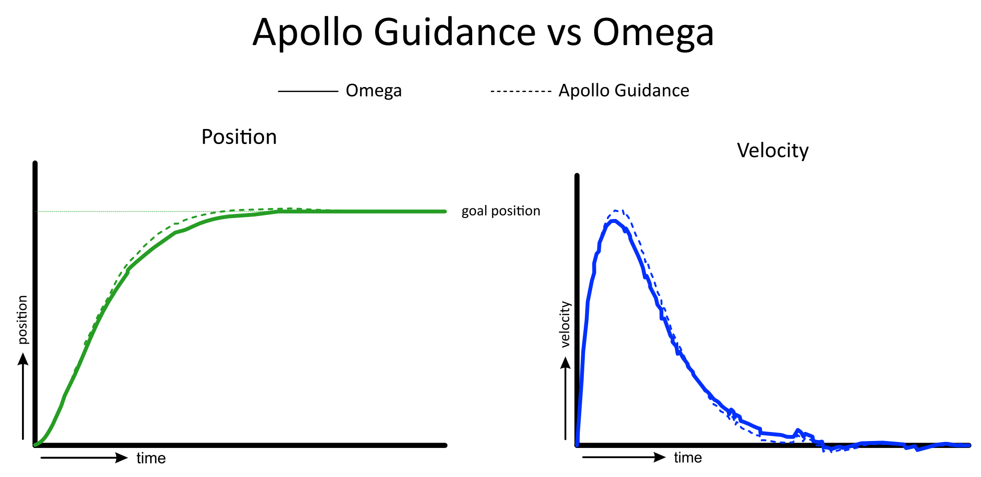
3rd Order Wave



For the X series, I have relied primarily on the 3rd Omega order, which, when used for positional control, guarantees continuity of position, velocity and acceleration. This provides superior smoothness compared to the 2nd Omega order, where an abrupt change in the positional goal causes a discontinuity in acceleration (see 2nd Order Wave graph above).

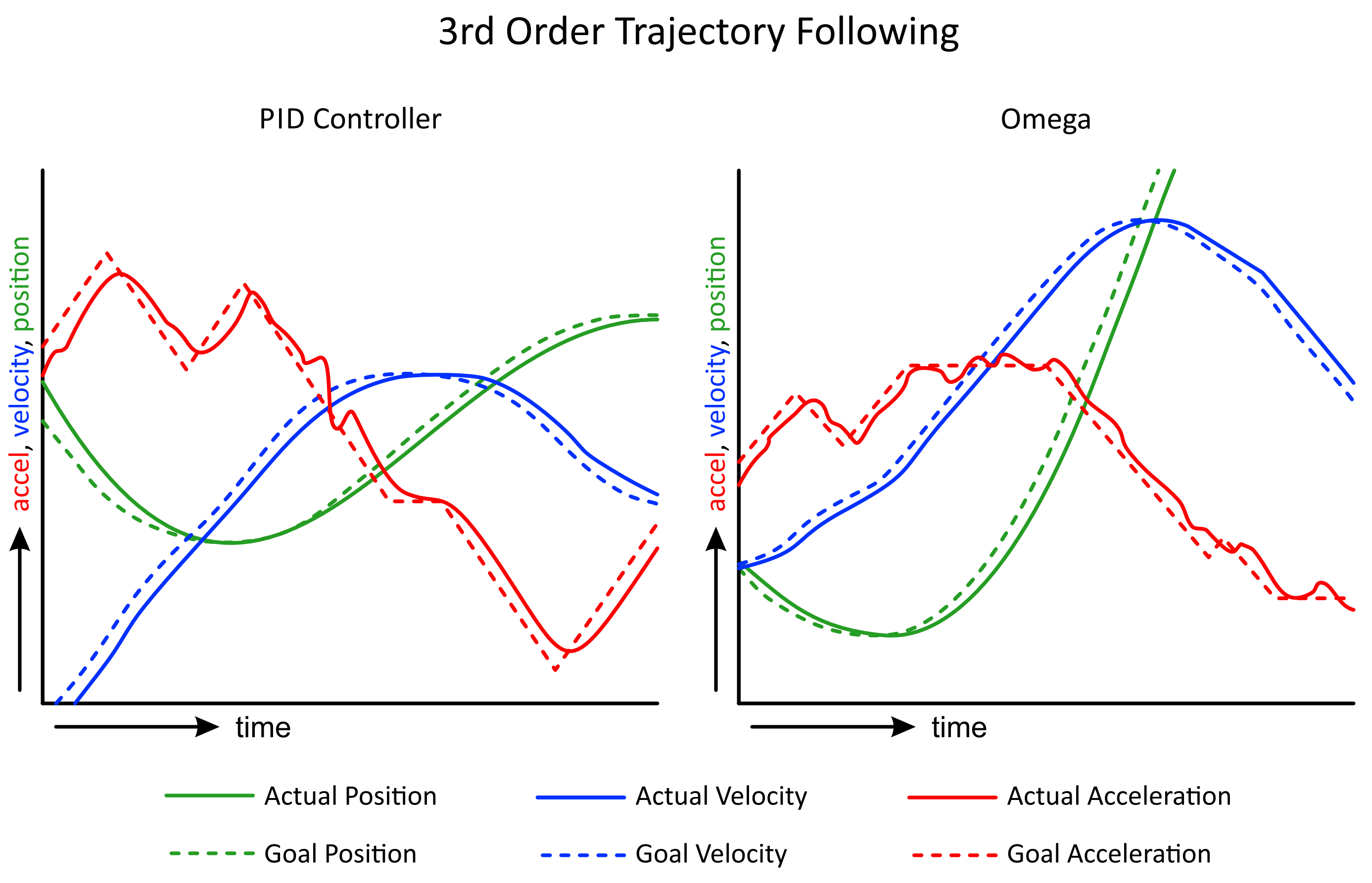
Omega IRL

Because the Omega step is a damped harmonic oscillator, it can be used to simulate systems that exhibit similar behaviors. In real-world applications, one example is the closed loop explicit guidance law used for the Apollo moon landing [5,6].



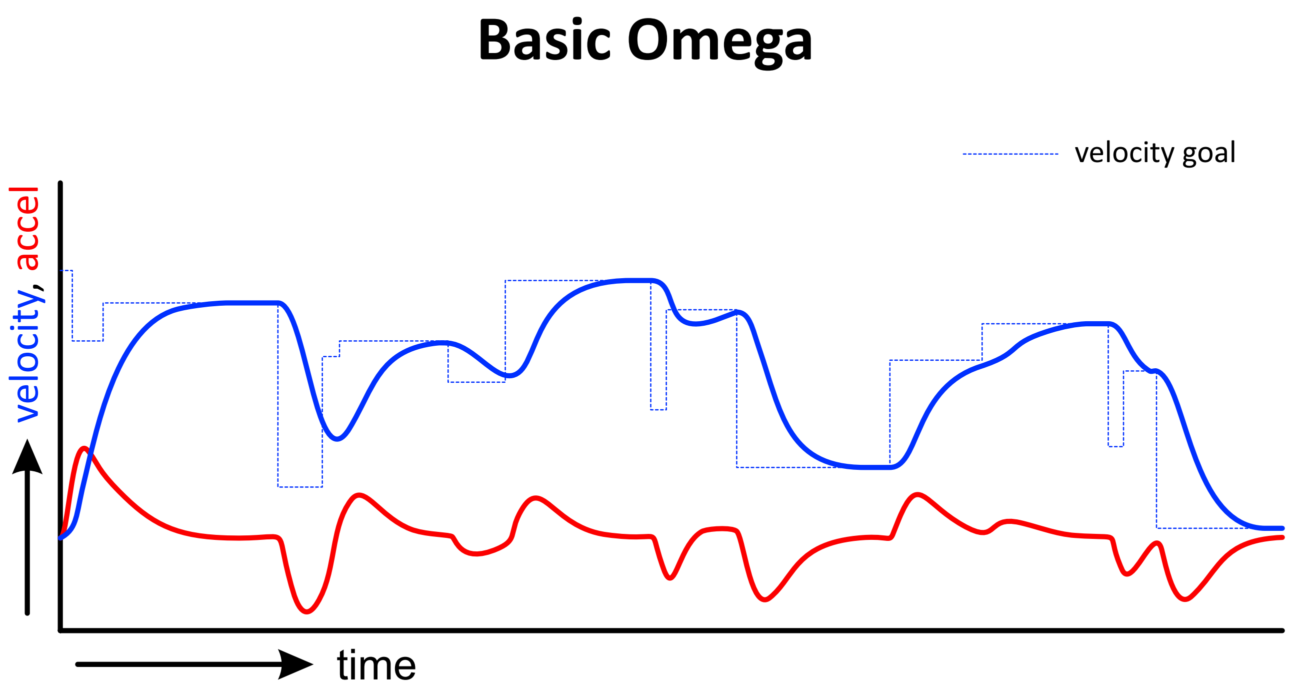
Note that the Apollo guidance law is a slightly underdamped harmonic oscillator, whereas Omega is critically damped.

Another example is a PID controller (Proportional-Integral-Derivative controller) [7], which is commonly used to provide error tolerant control, for example, for robotics. The following graph shows the actual trajectory of a robotic arm as it uses a PID controller to track 3rd order motion, compared to Omega-based tracking.



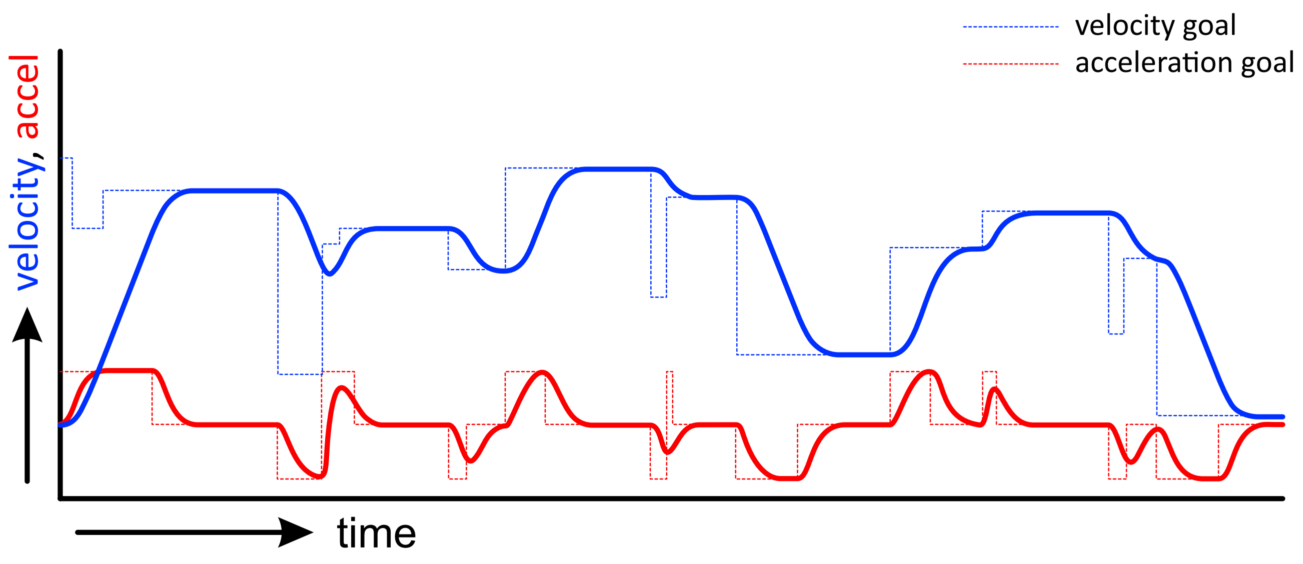
Omega in X

In the X flight model, the Omega algorithm is used as the primary method to generate acceleration curves that move a ship from its initial state to a goal state. In some cases, such as when we expect relatively short-duration changes in the control variable (acceleration, velocity or position), it is reasonable to use the basic Omega algorithm directly. Examples include linear and rotational reaction control (stabilizing attitude, station keeping relative to another vehicle, etc), as well as for maintaining zero lateral and vertical velocity during turns when flying under a nose-forward control mode.



Because of the shape of the Omega impulse function, which peaks early and then ramps out more gradually, without any cruising phase at all, it is not optimal to use this impulse directly to carry out very long duration changes in a vehicle's state. For example, if controlling velocity directly, a change in goal velocity from 0 to, say, 2000 m/s, at an average of say 4 gees, takes about 50 seconds. Using the Omega impulse to drive the acceleration curve, where the Omega step describes the change in velocity during this acceleration impulse, the acceleration will peak at about 2 times the average acceleration (in this case a spike up to about 9 gees) at about 9 seconds before gradually bleeding off toward zero acceleration as the vehicle approaches its final goal velocity at 50 seconds. While this kind of acceleration profile can be suitable for certain vehicles (it was used for the Apollo moon landing), in most cases it is desirable to ramp up quickly to a maximum cruising acceleration, hold that acceleration for an extended period of time, and then ramp down quickly to zero. In many motion planning applications, such as a robotics control system, this is accomplished by using a 2nd, 3rd or higher order polynomial equation to generate a guiding motion curve, and then using PID control to track that curve with error correction. For my model, I avoid the added overhead of this guiding equation by using an algorithm I call Switched Omega

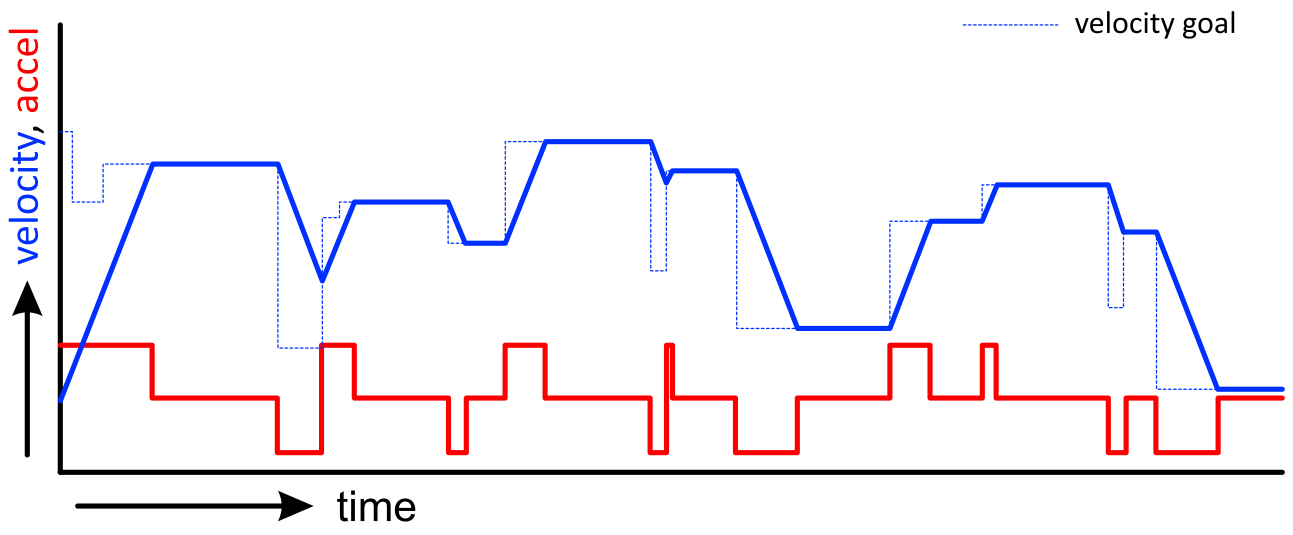
Switched Omega



With the Switched Omega algorithm, the step curve is used first to drive acceleration from its current value to its goal value, reaching that value with smoothness and accuracy, then this acceleration is held steady while velocity ramps up linearly toward its goal, and then finally, as we approach that goal, the algorithm switches so that the step curve will now drive the velocity itself toward its goal value. This allows the model to support both a maximum acceleration and a goal velocity with high precision and stability on both.

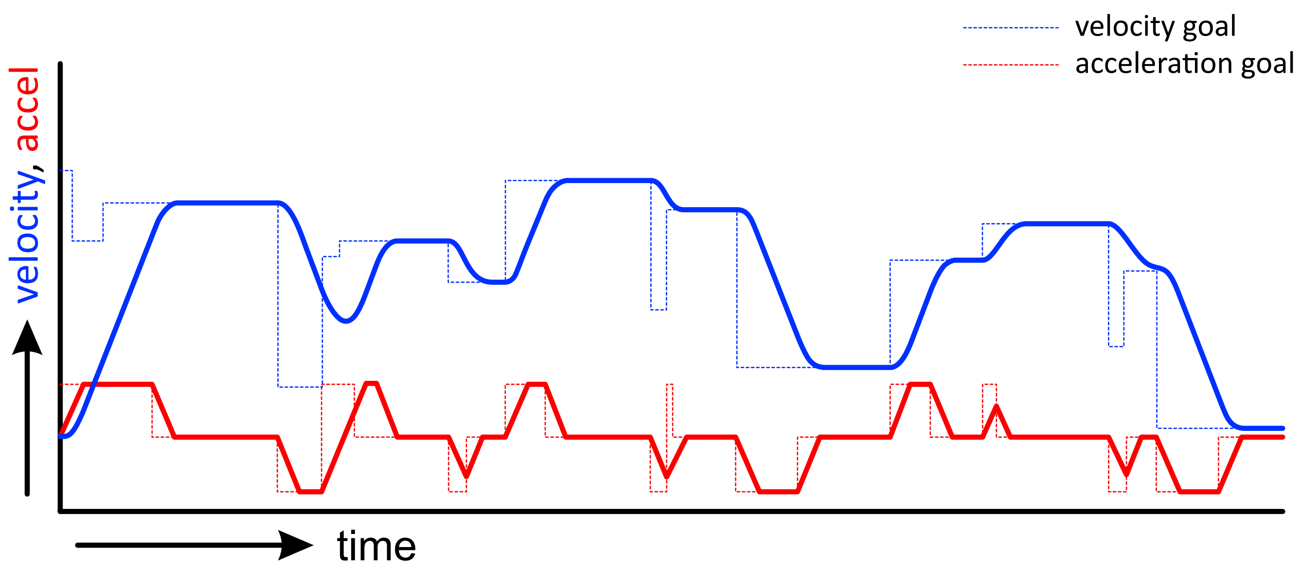
For comparison, here are examples of 2nd and 3rd order polynomial motion planning.

2nd Order

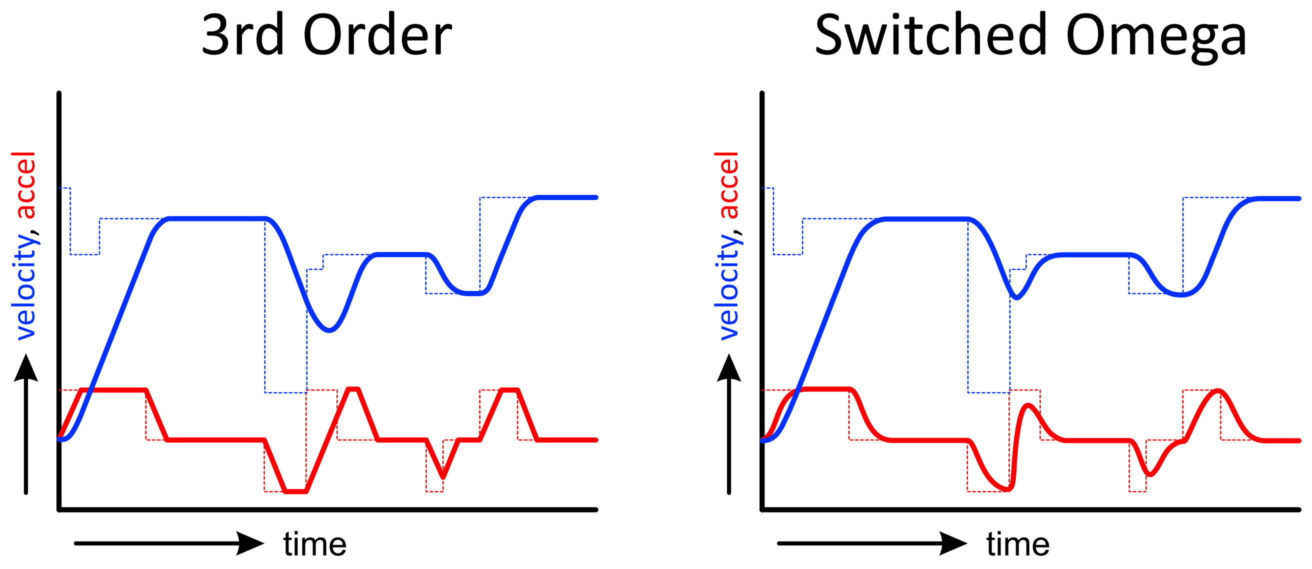


Switched Omega provides greater smoothness, but at a computational complexity similar to that of 2nd order.

3rd Order



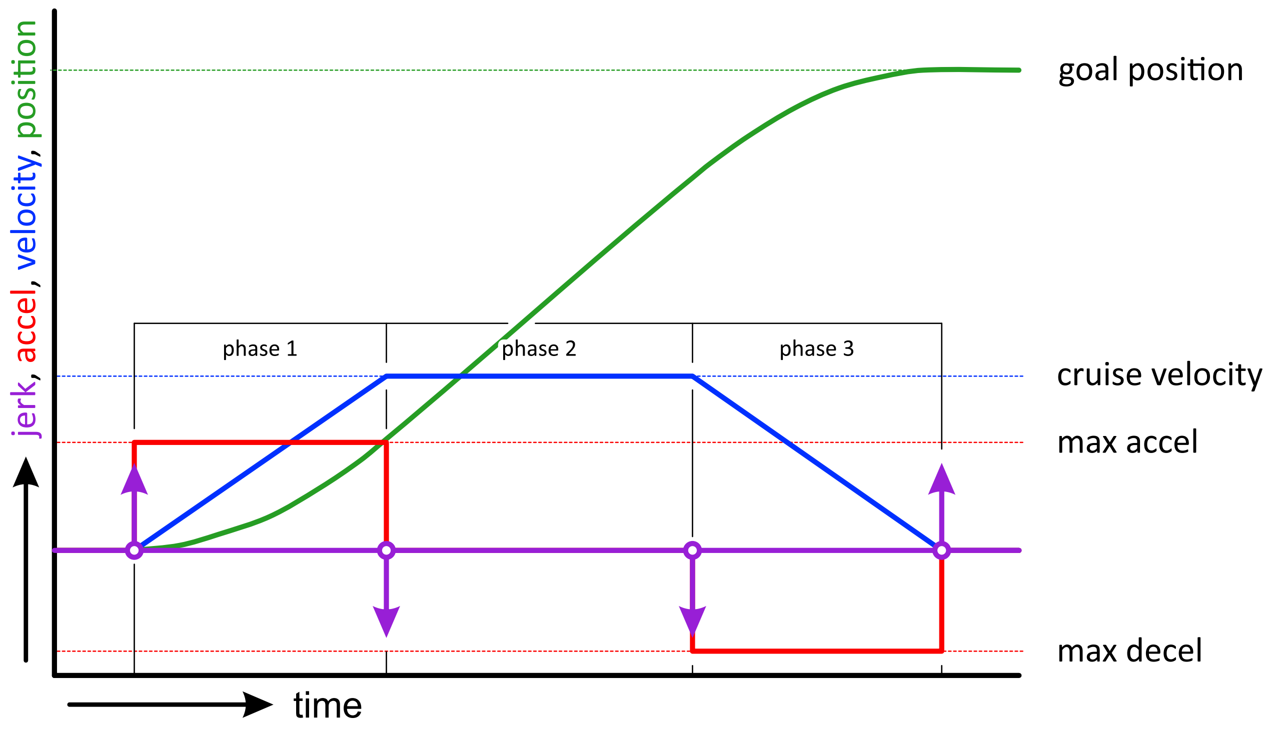
The following side-by-side of Switched Omega and 3rd order motion planning shows that Switched Omega achieves somewhat greater smoothness than 3rd order, but it does so at lower computational cost. In addition, it is error correcting, where 3rd order motion requires PID control to achieve error tolerance.



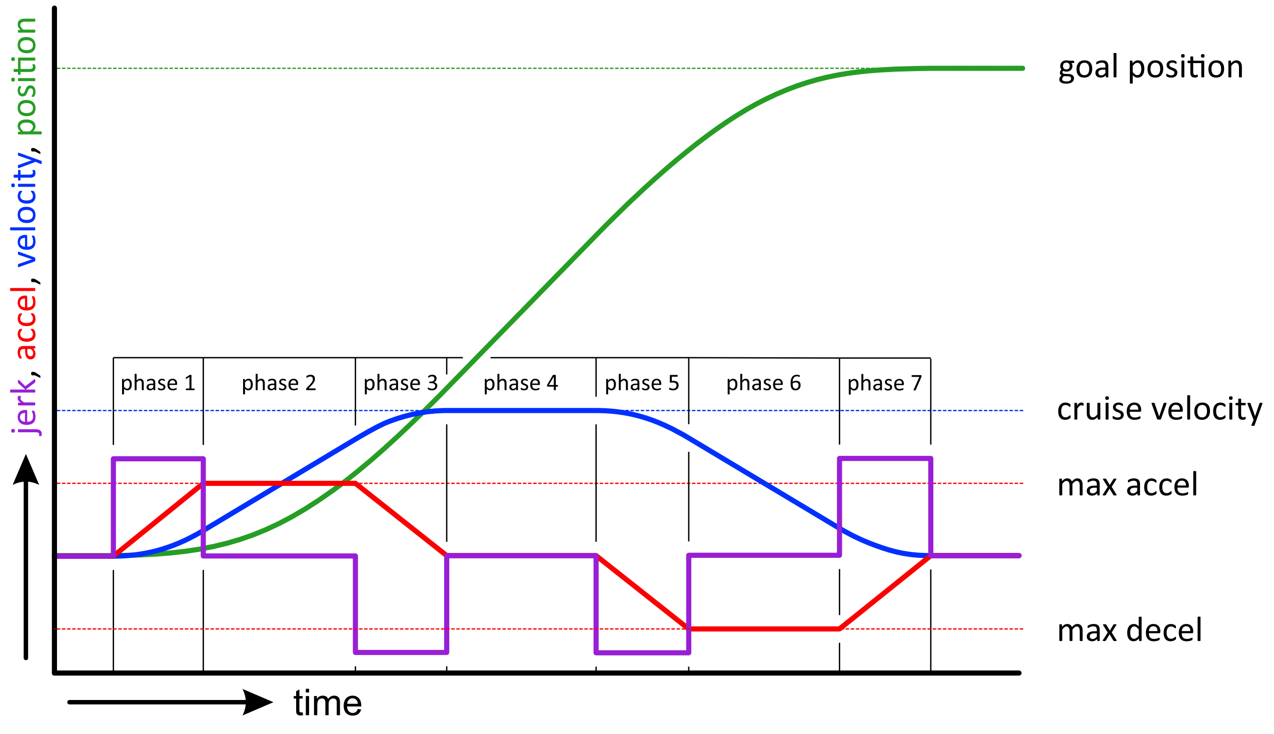
So far I have limited my presentation to simple velocity control. This is all that is necessary when providing support for direct player control or for dynamic AI steering control. However, in order to support more complex methods of control such as automated player motion as well as NPC waypoint or spline motion, positional control is required. Again, for short duration changes in position, such as a station keeping reaction control system, it is reasonable to apply the Omega step directly to an entity's position. However, this only allows for a limit on either peak or average velocity, with no limit on acceleration or jerk. Using the Switched Omega for positional control allows a cruising velocity as well as a limit on the peak or average acceleration, and this works well for relatively short duration velocity changes. In cases where long, sustained changes in velocity are expected, I have extended the concept of the Switched Omega to the Double Switched Omega, which supports positional control with a cruising velocity, maximum sustained acceleration and deceleration, and either peak or averaged jerk impulse limits.

For comparison, I will show sample positional control graphs using 2nd and 3rd order polynomial motion, and then Double Switched Omega motion.

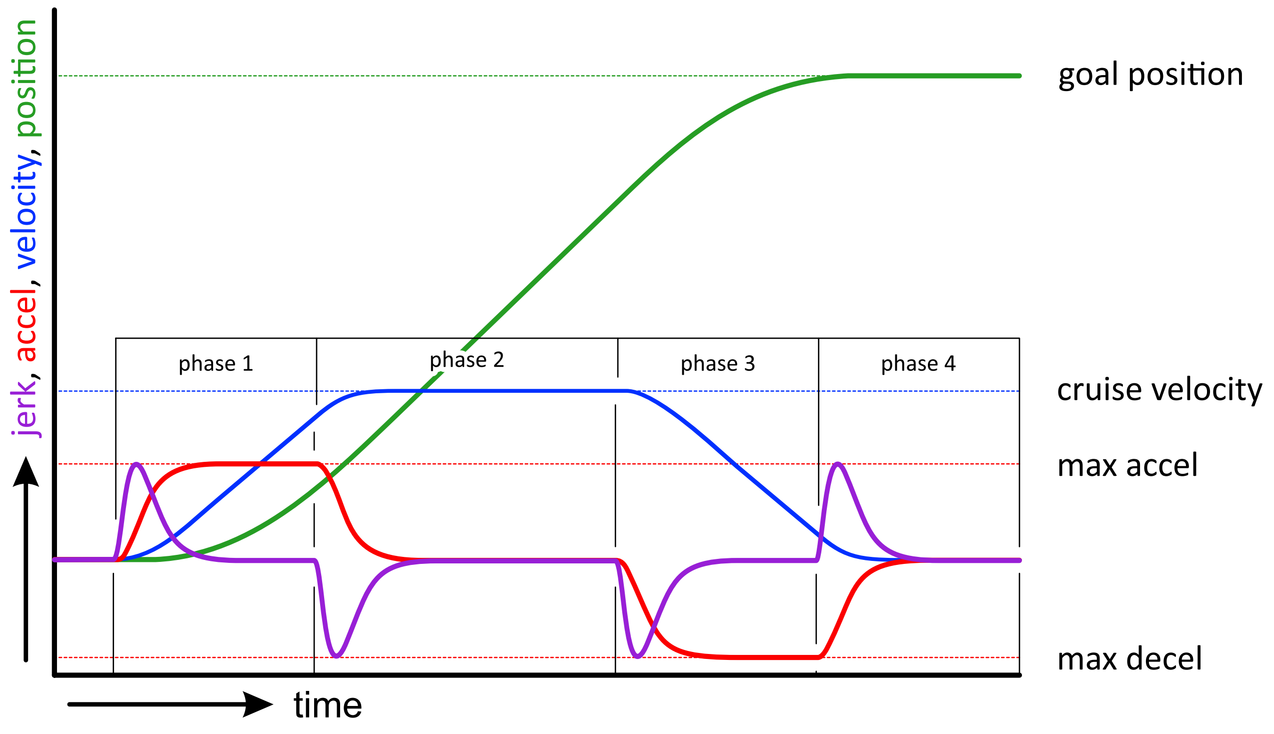
2rd Order Positional



3rd Order Positional



Double Switched Omega

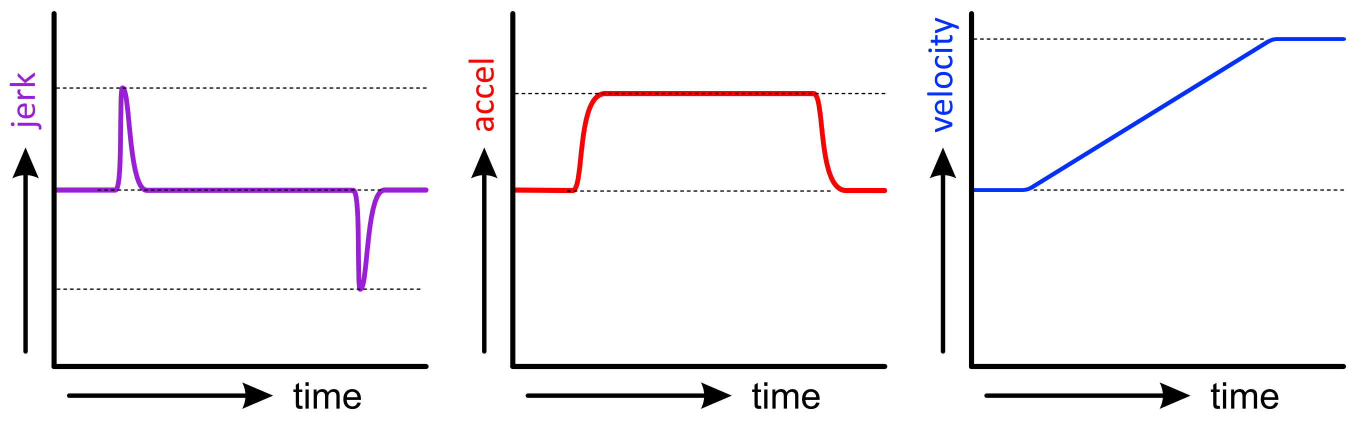


As with Switched Omega, Double Switched Omega control curves are smoother than 3rd order polynomial motion, with lower computational cost. In addition, Double Switched Omega has fewer phases of motion than 3rd order motion; only 4 phases compared to 3rd order's 7, and only 1 more than 2nd order's 3.

Tuning

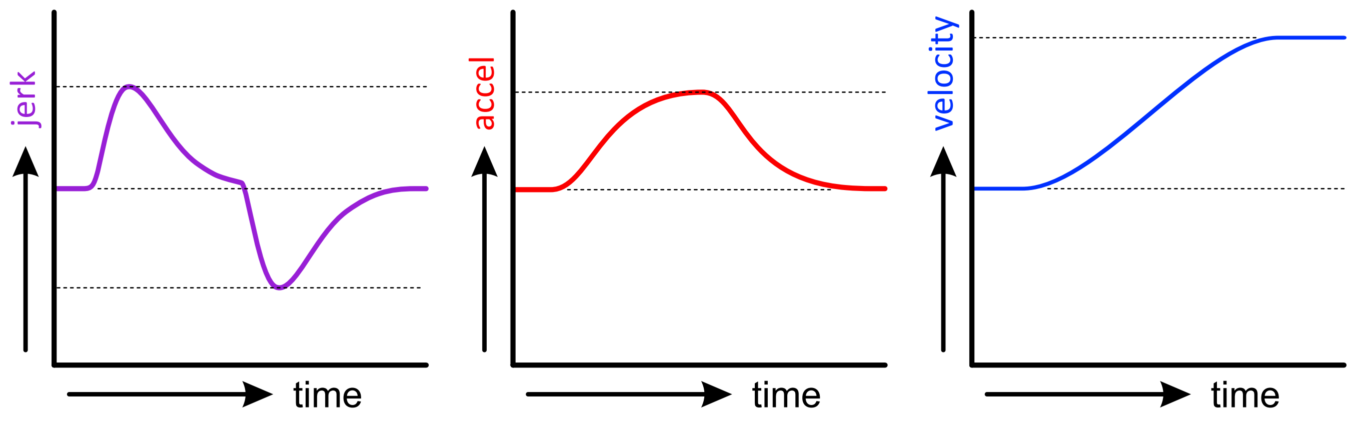
For both Switched and Double Switched Omega motion planning, designers have a range of jerk limits to work with. For minimum jerk smoothing, using velocity control as an example, the goal is for the acceleration ramp-up to be fast, but not instantaneous, so that the flight behavior is similar to 2nd order polynomial motion, but with a less stiff, more natural feel. In this case, acceleration will ramp up very quickly, just easing the edges of the 2nd order acceleration step, then hold at maximum acceleration while velocity ramps up linearly, and then ramp back down quickly as the ship arrives at the goal velocity.

Maximum Jerk



At the high end of the jerk smoothing range, again using velocity control as an example, the jerk limit is defined such that, for the maximum supported change in velocity, acceleration will ramp up to its maximum value and then immediately ramp back down toward zero, without any sustained cruising phase.

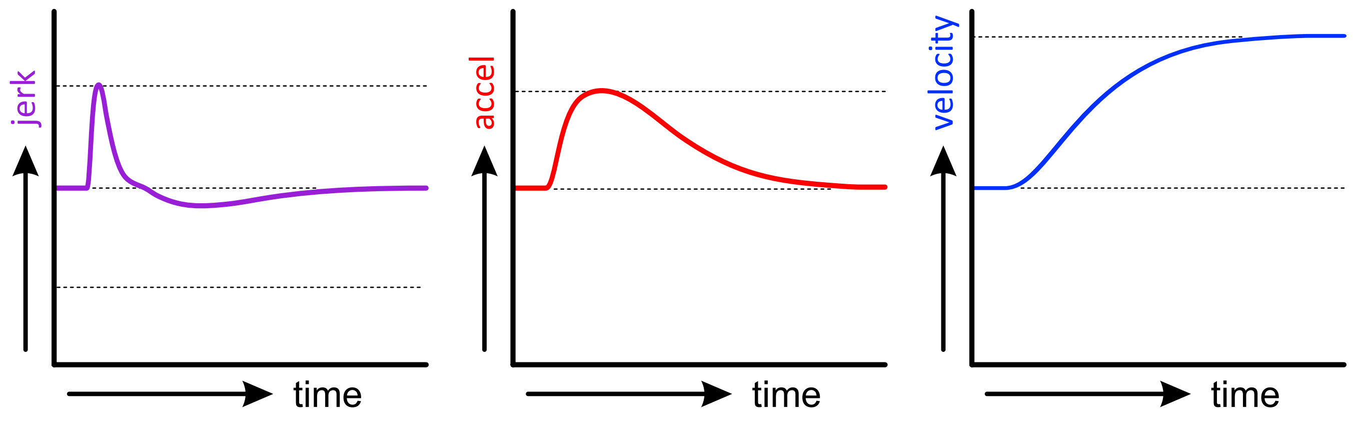
Minimum Jerk



This guarantees a maximum of velocity curve smoothing without saturating the acceleration curve. Once the acceleration curve is saturated so that it will never reach its maximum value, jerk begins to dominate so that changes in acceleration have very little impact on the performance of a ship. This upper limit on jerk smoothing guarantees that acceleration will always be the dominant characteristic for X ships, while jerk provides some additional variety in the feel of ships, in a range from highly agile and responsive like a formula one racer to a smooth and gentle ride like a luxury sedan.

In addition to this jerk range, designers can also choose to vary the ramp-in and ramp-out ratio, creating asymmetrical acceleration curves. For example, it is desirable for high performance ships to have a very fast ramp-in so that the ship responds quickly to changes in pilot input, but with a longer and more eased ramp-out to smooth the ship's flight characteristics.

Asymmetric Jerk

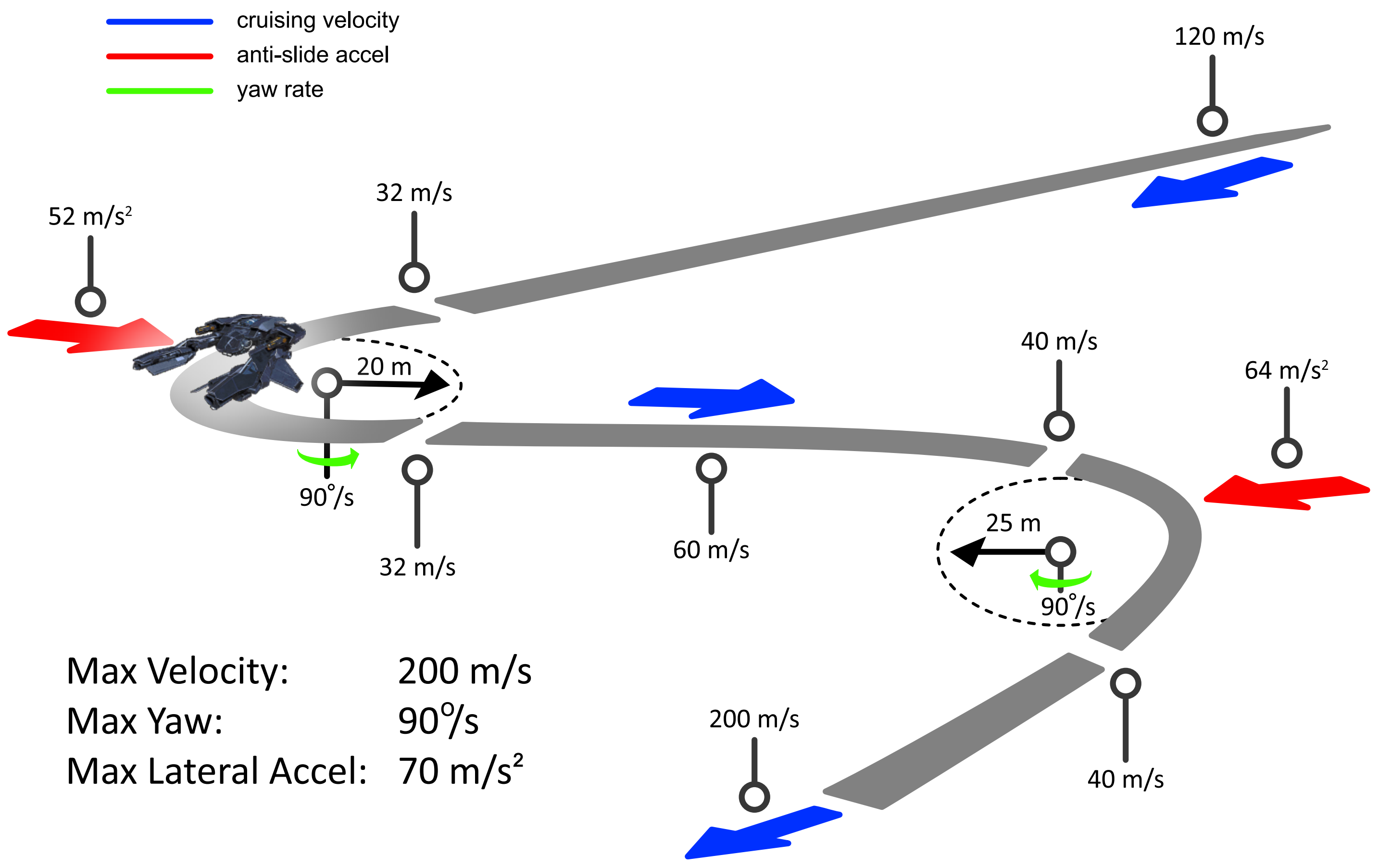


Spline Motion Planning

To wrap up this document, I will present the spline control system I developed for X. This system brings together all of the various elements of the new flight model. It allows the designer to designate any arbitrary spline path to be flown by any arbitrary ship, within that ship's flight constraints such as maximum cruising velocity, rotational velocities, acceleration, deceleration and jerk limits on each linear and rotational axis. Based on lateral and vertical acceleration limits, the allowed speed of a ship through a spline curve will be determined such that required centripetal acceleration does not exceed lateral or vertical acceleration capacities, achieving zero slide through a turn. In addition, based on constraints on the maximum rotational rates in pitch and yaw, the allowed speed of a ship through a spline curve is further limited so that the ship can always maintain, within a reasonable threshold, a nose-forward orientation.

Considering all of these constraints, a velocity profile is defined for the entire spline. This divides the spline into multiple segments, each of a designated length, ending at a designated velocity. The motion of the ship is then achieved by motion planning on each segment of the spline, such that the ship can enter each segment, ramp up velocity to maximum cruising speed, cruise until approaching the end of the segment, then ramp down toward the required ending velocity to safely enter the next segment of the spline.

This mode incorporates multiple control algorithms, including Double Switched Omega for forward flight along the spline, basic Omega for lateral and vertical error correction to keep the ship's path true to the spline, and Switched Omega for rotational control to keep the ship rotated into a nose-forward attitude.



Comparisons

The following videos show comparisons of Omega motion versus 2nd order polynomial motion in X4.

1) Rotational motion, cockpit view
2) Rotational motion, external view
3) Continuous rotational motion, cockpit view
4) Continuous rotational motion, external view
5) Linear motion, cockpit view
6) Linear motion, external view
7) Turrets
8) Weapons

Appendix

[1] Star Citizen IFCS 3.0 Design Document
[2] Noobifier's IFCS 3.0 Video
[3] Dirac Delta
[4] Harmonic Oscillator
[5] Apollo Lunar Descent Guidance
[6] A General, Explicit, Optimizing Guidance Law
For Rocket-Propelled Spaceflight
[7] PID Controller