Evolving Rocket Control Systems for Launches to Orbital Flight
Experimental Design
CMPLXSYS 425
April 19th, 2018

Noah Bearman - nbearman@umich.edu
Owen Cannon - cannono@umich.edu
Joseph Donohue - donohuej@umich.edu
Divyansh Saini - divyansh@umich.edu

Introduction

In our accompanying literature survey, we introduced the problem as using a genetic algorithm to evolve a control system for a rocket to bring it to orbit efficiently. A more rigorous definition of our problem follows.

We set out to create a rocket control system capable of piloting a particular rocket from launch to a stable orbit of the planet Kerbin in the computer game *Kerbal Space Program* (KSP). KSP is a suitable platform for this experiment because it allows for the construction and physical simulation of rockets and aircraft composed of prefabricated modular parts. This game, while cartoonish, conducts a somewhat true-to-life real-time physics simulation, enabling us to explore realistic solutions to real world problems such as efficient launch trajectories. We define a stable orbit as any trajectory for which the apoapsis and periapsis are both above 70000 meters above sea level. This is the altitude at which the game no longer simulates the effects of drag, and so an object on such a trajectory will remain on that trajectory indefinitely. Reaching a circular orbit at 70000 meters above Kerbin is analogous to reaching low Earth orbit in the real world.

Control Systems

All control systems were tested on the same rocket. The rocket chosen was designed by us specifically for this task; it features several properties which make it ideal for our experiment. Firstly, it exhibits high levels of aerodynamic stability and control. With four active lift surfaces and four static stabilizers, once the rocket is moving straight, turbulence is not enough to knock it off course. Secondly, the rocket contains a single stage and is more than capable of reaching high orbits, and can therefore be classified as a single-stage-to-orbit ship (SSTO). We decided an SSTO would be optimal for this experiment, as requiring the control system to manage rocket stages adds a great deal of complexity to the solution space, and it was predicted that evolving a useful solution with staging would take much longer. This configuration also has plenty of fuel, allowing for a wide range between the minimal viable solution and the most optimal solution, which we predicted would be good for evolutionary progression. The overall simplicity and robustness of the rocket was thought to decrease the complexity needed for viable solutions.

The control systems we evolved took the form of neural networks. These neural networks polled the state of the rocket and sent controls at a rate of 10 Hz. The outputs of the network were limited to:

Roll force — percentage of the rocket's maximum torque capability currently applied to rolling vehicle around its longitudinal axis
Yaw force — percentage of the rocket's maximum torque capability currently applied to changing the heading of the vehicle
Pitch force — percentage of the rocket's maximum torque capability currently applied to changing the pitch of the vehicle
Throttle — percentage of the rocket's maximum thrust capability currently applied

Together, these constitute full control of the rocket. The inputs to the neural network were varied slightly throughout the course of the experiment, but included:

Roll — the current degree of roll around the longitudinal axis of the vehicle
Pitch — the current degree of pitch relative to the horizon
Heading — the current direction of travel relative to the horizon
Ground speed — the current speed of the vehicle relative to a fixed point on the surface of the planet
Altitude — the current height of the rocket above sea level
Prograde vector — the three values specifying the direction of travel of the rocket relative to the planet-sun orbital plane

Many more pieces of information could be given to the control system to improve its awareness, but increasing the number of inputs comes at a cost of both increased network complexity as well as computational complexity. As mentioned above, the KSP platform is a

real-time simulation, and so the network needs to be able to respond fast enough to maintain adequate control of the rocket. While calculating the state of the network likely would not increase to drastically with several more inputs, extracting more data from the game engine comes with a time cost and should be handled with care to prevent debilitating latency from crippling the control system.

Software and Algorithm

Using a modification to KSP which enabled remote procedure calls over sockets, we created a Python program to manage our genetic algorithm and interface with the game (found here: https://github.com/div3/EvolutionInSilico/tree/master/final). The mod used is called KRPC (and can be found here: https://krpc.github.io/krpc/). For our genetic algorithm, we used the neural evolution of augmenting topologies (NEAT). We used an existing implementation of NEAT for Python (found here: http://neat-python.readthedocs.io/). We settled on using NEAT for our genetic algorithm because NEAT specializes in minimizing solution complexity, which is of vital importance for real-time control systems.

A basic overview of the genetic algorithm we used is as follows: a population of control systems was initialized randomly. The game would be loaded into a preset pre-launch state, and the rocket would be launched. Shortly after the rocket is launched, control is handed to the neural network, which continually receives data from the rocket and sends control commands at a rate of 10 Hz. All four rocket controls can be sent from the control system to the rocket in a single tick. Flight of the vessel continues until an end condition is met. If the rocket fails to gain altitude at an acceptable rate, the flight is terminated; or once the rocket reaches a stable orbit, the flight is terminated; or if 200 seconds pass, the flight is terminated. After the flight is terminated, the fitness of the control system is determined; then, the pre-launch state is reloaded and the next neural network is placed in control of the rocket. After each individual control system has had a chance to conduct a flight, NEAT's intergenerational processes generate a new generation of control systems, and the process is repeated.

Our intention was to optimize the launch profile of this rocket. To optimize the launch is to reach a stable orbit with the most remaining fuel possible. In order to optimize a launch to orbit, however, the control system must be capable of reaching orbit to begin with. Evolving a system to do this proved to be rather difficult; we tailored our fitness function to cater to two different phases of a flight:

Sub-orbital: while the trajectory is sub-orbital, progress toward space is rewarded in both vertical height gained and eccentricity of the trajectory (how "arced" the trajectory is). We chose to reward individuals with tall, low-eccentricity (broadly arced, not straight up and down) trajectories because these trajectories are best suited for reaching orbit. When we rewarded individuals simply for altitude gained, the rockets would fly straight up, which makes reaching a stable orbit mostly impossible.

Orbital: after a trajectory becomes an orbit, we need a way to assess how good the orbit is. This way, we can rank two control systems which reach two different orbits. Since we sought a control system for efficient launches, we defined the better orbit as the orbit with a higher energy-to-fuel-cost ratio. This way, a rocket which reaches a very low orbit with some fuel

remaining is directly comparable to a rocket which reaches a very different orbit with no fuel remaining. Low eccentricity is also rewarded in flights that reach orbits; this represents our arbitrary decision that circular orbits are better than elliptical orbits.

Computational intensity was a major concern for our experiment. For one thing, the simulations run in real-time; the game was designed to be played by human players, meaning that simulating a rocket launch takes just as long as it would if a human were playing the game. The genetic algorithm approach benefits from larger population sizes and many iterations, but attaining this was difficult. In order to see evolutionary progress, we ran with fairly limited population sizes (10, 15, and 20 individuals per generation). We were able to run some simulations for more than 25 generations, but as the population improved, more of the trials ran to the 200 second time limit. This meant a single generation of 10 rocket flights took over half an hour. Our experiment would have benefitted from parallelization, but this is cost-prohibitive, as modern central processing and graphics processing units are needed to perform the work. Finally, we encountered an obstacle in KSP's software itself. We discovered what we believe to be a memory leak caused by repeatedly loading quicksave files (our method of reestablishing the pre-launch conditions), and this led to degraded performance of the game as time went on. After too long, the simulation would become so slow the neural networks could not effectively control the rockets. We had to design our experiment around this fact; we built in saving the state of the population, which allowed us to pause the genetic algorithm while the game was reloaded. All this is to say that running the experiment was quite time consuming, and more satisfactory results would have required significantly more time.

Conclusion

Initially, we started by evolving controllers with a very discrete set of outputs to the neural net where it would output a value between -30 and 30. The initial fitness function rewarded absolute height achieved by the rocket, this seemed to evolve rockets fairly quickly, but as the rockets reached above 70,000m we found that they would be travelling too fast to turn into orbit. We then updated the fitness function to also reward the absolute distance from the launch pad. This produced rockets that would travel a more natural path to orbit, gently curving at 45 degrees. But as the rockets would use maximum throttle pretty much all the time they were generally unable to get high enough to get into orbit. At this point, we did have rockets reaching into sub-optimal orbits, but we decided to switch to more fine-tuned throttle control. We expect this to produce better results.