CS Capstone Problem Statement

October 9, 2017

Inferno

Prepared for

OSU - Phoenix Solar Racing

Prepared by

Group 43
Inferno
Logan Kling

Abstract

This Problem Statement is a detailed explanation for building a simulation for a solar car. This document starts broad, asking how to solve OSU Solar Teams testing problem, then by answering that with this project. It then describes why we will choose to program this in C++11. The general consensus for the GUI is that it should be pretty simple. We agreed on having the program store all specs of the car that significantly affect its performance in a text file. We also wanted to calculate what the most efficient speed given those parameters would be. We also determined that we should try to finish our simulator well before the car was finished, as we do have data on the last car. This is entirely physics based as well, so with the right variables, equations, & a good map API we should be able to simulate the car with a surprisingly small subset of its data. When the car is finished, we will take its stats into account. Lastly, we established some bonus goals. Most of which were impractical, but goals we should still aim for.

1 The Problem & Solution

The OSU Phoenix Solar Team would like to test their car more than they are currently able to. The problem is that testing the car is tedious, & they will almost certainly wish to test different designs & parts for the car. The race that these engineers participate in is up to 2000 miles, so they can't exactly test 100 different designs. A simulation of such a car can test 100 designs, though. A simulation can test far more than 100 designs, & it has the added benefit of not wearing out the solar car.

2 How We Plan to Implement this Simulation

We're allowed to code this in any language as long as it can be run on Windows, Mac, & Linux, but because this simulation will almost certainly be processor intensive C++11 will be the best language to use. It's lower than Java & C#, both of which have been used by one of their electrical engineers, but it still has room for plenty of libraries—even if some of those necessary libraries will vary on different operating systems. Java is incredibly slow & restrictive as it compiles everything written into it to byte code first. This allows Java applications to run on wildly different operating systems with no conscious effort from the programmer being necessary. However, the necessary process of compiling that bytecode every time one wishes to use the program is incredibly slow on top a fully compiled Java program also requiring more resources (RAM especially) on average. While C# has a lot to offer, it's still slower than C++11 in most cases, & the recommended languages were Java, & C++. Even using C++11, I predict we will have a tough time writing this program well enough to simulate a 2000 mile drive in under 30 seconds.

The next issue is the Graphical User Interface (GUI). Despite this being only a small part of the program, it's easily the most important. The OSU Solar team want a simple GUI that's easy to use. This means a minimal menu bar (likely with only a file menu) & the most useful features as buttons with helpful icons & clear, understandable labels underneath each one. In the area below that will be a blank box that will be filled with the stats of a simulated drive after it has been run by the user. User implies someone from the Solar Team.

Some of the data that should be modifiable via GUI. This data includes the weather which affects how much sun this solar car will receive, the difference in altitude from start to finish, the batteries initial charge, & the set time or speed for the car. However, these values vary a lot more than physical constraints, such as drag, weight of the car, the friction of the wheels, and the weight of the wheels. The user will need to tweak values associated directly with the specs of the car, but not anywhere near as often as they need to change conditions such as the weather. While a small subset of these values can & should be modifiable within the program itself, I predict we may end up with close to 50 variables dealing with this car. I cannot yet imagine a GUI that would make dealing with all of those any more pleasant than a text file. While the client believed 50 variables were far more then they needed, they did agree that if we needed that many, that having the user change some of them (namely the more obscure, less modifiable ones in terms of the car) from a text file wasn't a bad idea.

The biggest part of this program will be our system of equations to handle these variables. Some of these equations are easy, such as the one for rotational inertia which simply calculates that every time the car needs to accelerate or decelerate, the wheels will be effectively double their weight. However, calculating how much sun the car receives is going to be an extremely complicated set of equations, each with a slightly different level of significance. Testing & analyzing these equations would be practically impossible without data from the previous cars race. Not everything from that car will translate to this one nicely (such as how much sun it received) but others, such as air resistance will be much easier to translate to this car. The data the OSU Solar Team has on their last car is massive. They have data for the latitude, longitude, elevation, velocity, charge of each of the three battery packs, the solar arrays power usage, its absorption, the motor usage, & the energy used per mile for every minute of their last race. This is 174 pages of raw data, so that means we'll have to write a small program to process that data to make it useful to us.

3 Bonus Goals

Lastly, we have some ridiculously ambitious bonus goals, which we will hopefully implement. We do want to "finish" this project very early so as to be able to improve upon it greatly. The most difficult of our goals is to be able to map the altitude minute by minute in order to calculate how much energy will need to be used far more accurately. I've been unable to find an API with accurate enough altitude records that would be plausible to hook into. In theory, this would allow us to drastically improve the accuracy of our simulation, though it would also slow it down by an order of magnitude because of the need to fetch topographical data so frequently. It would also be nice to update weather conditions on the fly. This would also slow down the simulation, though nowhere near as much as the altitude, as weather is far less precise. The least ambitious of our bonus goals is to record the angle of the sun along with the direction of the car. This would be much easier to implement as we likely wouldn't need any additional API for it. With some basic algebra & the map API that we would already have, this wouldn't be all that difficult to calculate.