Navigation Simulator

Project Description

Liam Bindle

12/24/2015

This document outlines a navigation simulation software package for the University of Saskatchewan Space Design Team (USST).

One of the primary goals for this year on the USST is developing a software system which will allow the rover to drive autonomously. Our hopes are that by next June we have a system which will allow the rover to drive a route which is specified from a list of GPS coordinates.

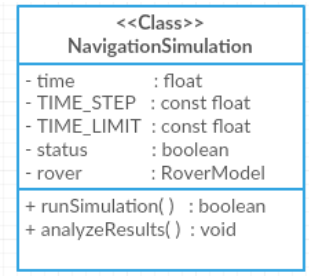
Testing such a system is very time consuming and it is not practical to do all of the testing with the rover, thus we need a method for testing and evaluating navigation algorithms.

A simulation which is able to accurately model how the actual rover drives is required so that we can test these navigation algorithms efficiently. The system must take an object-oriented paradigm so that additional models can be added or removed based on what we deem necessary as the project progresses.

The program will be divided into two parts—the simulator and the rover model. The simulator will be small and simple however the rover model will most likely become quite complicated. In the first section here, I will describe the simulator and how it will be used. In the next section I will describe the rover model and how that will work.

Note that what I have described in the following couple pages might seem a little overwhelming at first however it is actually a very simple program—there really isn't anything that is complicated. When reading it try not to overcomplicate it because it is really is as simple as it sounds(the framework especially).

Simulator



Above is the basic structure of the simulator.

time :: A variable which keeps track of the time during a simulation
TIME_STEP :: a constant which defines the size of the time step used in the simulation
TIME_LIMIT :: The simulation time limit. If the simulation has not been successful by this time then the simulation is said to have failed
Status :: a Boolean which indicates whether the simulation was successful or a failure
rover :: the instance of the RoverModel class

runSimulation() :: a the method which is called to run a simulation. This method will look something like the following. Note that this is just a basic pseudo code which is what I anticipate it will look like

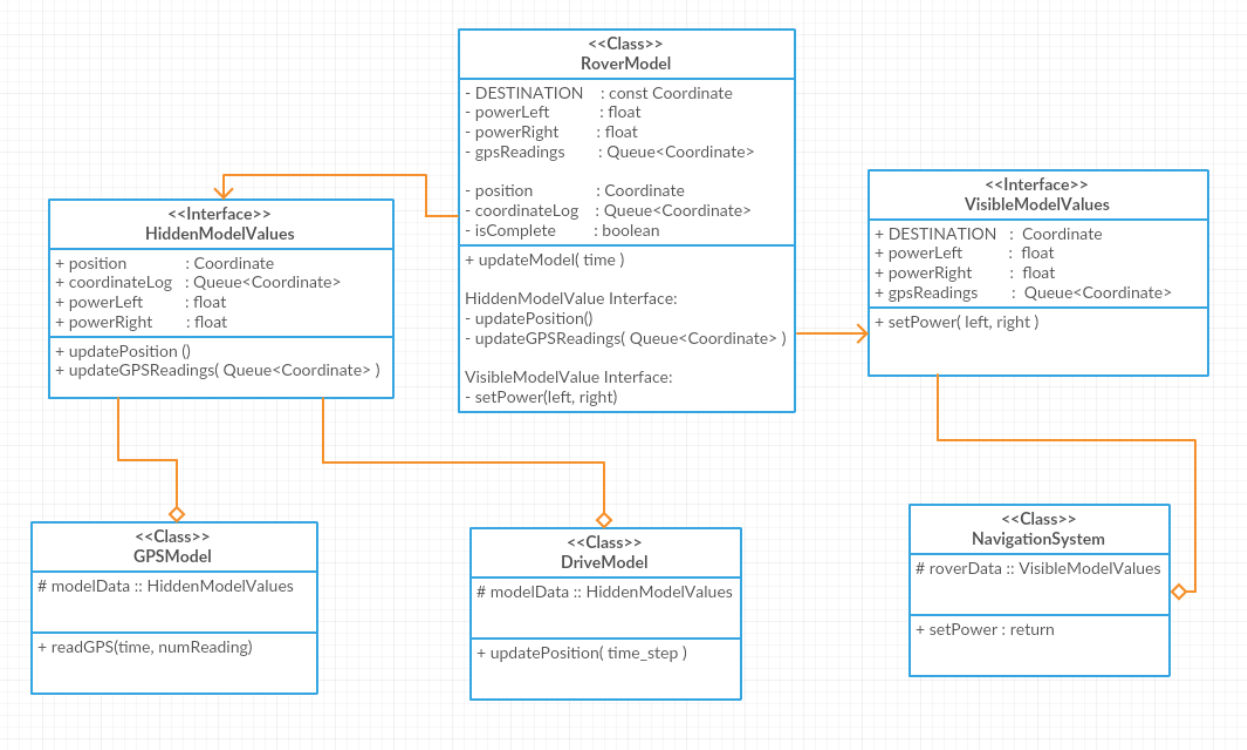
```
NavigationSimulation::runSimulation(){
    time = 0
    … do setup things …
    rover = new RoverModel( <constructor arguments>  )
    boolean notDone = true
    while(notDone){
        rover.stepTime(TIME_STEP)
        notDone = rover.simulationIsDone()
    }
    … check if the simulation was successful …
    if(successful)
        return true
    else
        return false
}
```

The only other method in the NavigationSimulation class is analyzeResults(). This method will be called after a simulation and will look at the results and show us what we want to see. Note that we will be using Numpy which is a python library which is similar to Matlab. This should allow us to plot results easily and visualize the results.

RoverModel

The rover model is a more complicated class. This class will contain multiple smaller models which will be used to emulate the real life rover. These sub-models will be models such as a GPSModuleModel which will simulate the GPS readings, and another will be the DriveModel which will model how the rover moves through our emulated environment.

These two basic models which I have described are going to the most fundamental models which we will for sure need however its possible we may need to add more if needed later. The following is a basic layout of the RoverModel



As you can see in the above UML diagram, the GPSModel and the DriveModel work with the “correct values”. These models outputs are generating emulated estimates of these “correct values”. For example the GPSModel takes the actual coordinate and then would generate a Queue of emulated GPS readings which would be distributed around the actual coordinate (it sets the GPS readings by calling the updateGPSReadings function which updates the gpsReadings queue in the rover model). The DriveModel takes the current coordinate (the “real” one) as well as the right and left drive power to then calculate the next position(it updates the position by calling the updatePosition(coordinate) function which sets the position in the rover model).

The NavigationSystem class is the most important class and will be the class which is actually used on the rover. This class can only see the drive power, destination coordinate, and gpsReadings (which are the modeled GPS readings). The Navigation system then sets the power to the right and left based off of these values and the GPS coordinates. This is the class which we will be changing a lot and hopefully eventually come up with a very robust solution.

By doing this over and over we should be able to simulate the rover driving. Obviously this is a simplified version but for the sake of explanation it should cover the general idea.

What to do

I haven’t set up the github repo yet however we can get started working right away! The following is a list of how we can start.

- First the framework needs to be laid out a bit. This means that all 7 classes/interfaces need to be written up (but not implemented yet). By this I mean just writing each of the classes and declaring all of the fields/methods, implement the simple ones and leave the more complicated ones for later.
- Once this is done I'd say that the first thing to work on is the Simulation class. This class should be pretty simple (don't worry about the analyzeResults method till later) and will allow us to test the RoverModel as we develop it
- Once that is done then we should start working on the interfaces and the implementation of these interfaces in the RoverModel class.
- Once we get to this point we should have a better understanding of where the project is and we can discuss what's next

If you need any clarification just let me know and I'll clear up whatever you need clarification on. I will set up the repo in the next couple days but for now let's just communicate by email on what we are working on.

Happy holidays,

Liam