# ECE 275 - Project 4

**D2L:** Project 4 (alpha)

**Due**: 10/26/2018, 11:59 PM

**D2L:** Project 4

**Due**: 11/02/2018, 11:59 PM

**Notes and Revisions:**

# Usage

Usage: catcar controlInputs stateOutputs

Command Line arguments

# Requirements Summary

Create a C++ program that reads control inputs (Input file stream)for a self-driving car from an input file. The program will simulate the car's state over the time horizon given by the control inputs and write the results to an output file (Output file stream). The major problem is: the input file was written in such a way that the control inputs are stored out of order! (Sorting algorithm)

# Assignment Name

The assignment name for this assignment is: catcar

# Kinematic Model Simulation

In this project, you will create a program that controls an autonomous vehicle from an input control file.

they were stored in order in a *queue*,

**They printed the nodes out by order of memory address of the list nodes, instead of by the node order prescribed by the next pointers.**

This program will read in **time** value at which this command should be executed, then the **commanded speed**, and **tire angle**. The duration of the command is the difference between this command's time value,

and the next time value (when the nodes are in order), at a maximum of 201 ms.

If there are any durations longer than 201 ms, the input file should be declared invalid, and no simulation should take place.

With this sequence of commands, you should **save the vehicle's state at each time step**.

During the execution of control commands, this program will record the vehicle's state as it moves (or stands still) and save the output states to a file.

1. **File Format**

## 2.1 controlInputs

For this assignment, the input text file will consist of control information, each line contains sample time (in seconds), commanded vehicle velocity (m/s), and tire angle rate (radians/s), with elements separated by whitespace (even the first number could be padded by a whitespace in the front). All values should be stored as doubles.

0.06 5 0.523 // sampling time (seconds), desired velocity (m/s), steering rate (rad/s)

0.02 20 -0.523

0.07 8 0

0.00 0 0

In this example, at time 0.06 the vehicle should take in as its input command a velocity of 5, with a tire-angle rate of 0.523, for 0.01 seconds.

The duration of 0.01 is determined by looking at the next time step (after the vector is confirmed to be sorted) and taking the difference (0.07 - 0.06).

Your program should run regardless of the unlikely control inputs, as long as the below conditions are met:

* *The first element of the sorted list must be at exactly time 0.*
* *Commanded tire angle rate must be between [-0.5236, 0.5236] radians/sec (i.e.,/ 6 radians/sec).*
* *Commanded velocity must be between [0, 30] m/s;*
* *Time values must be positive;*
* *Duration between sorted input objects must be between [5, 201] ms.*

If any of these are violated for any input value in the file, then the controlInputs file is deemed invalid, and an empty output file is written.

If at any time a line fails to parse, then the file should be declared invalid.

## 2.2 output File

This assignment should output a file with information on the state values of the vehicle throughout its journey. The format for this file is one called csv (comma separated values), and is common for inputting data into MATLAB. Each output entry is as follows:

t, x1, x2, x3, x4\n

*Timestamp, Xpos, Ypos, Angle, Heading*

Where t is the time at which this state value was measured, and the values in x are given by the kinematic equation (1). The precision for each value should be whatever is the default when using C++ iostream methods.

If the control inputs are determined to be invalid, you should create an empty file with the name provided from the command line.

# Vehicle Model

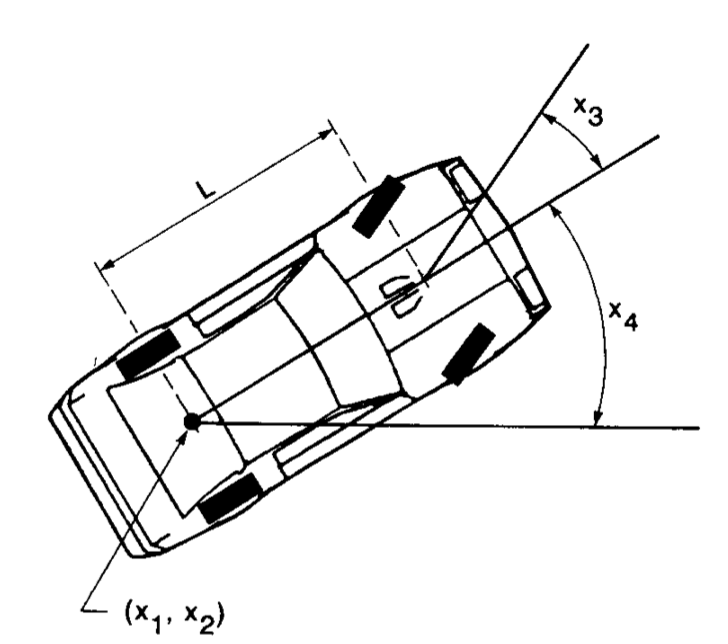
You are developing a vehicle simulator, which describes the kinematic motion of a front-steered, two-wheel drive vehicle. The vehicle you are simulating is visually depicted below. The equations of motion for this vehicle are provided in (1).

*x*1 = *u*1 cos(*x*3) cos(*x*4) (1)

*x*2 = *u*1 cos(*x*3) sin(*x*4)

*x*3 = *u*2

*x*4 = *u*1 (1/L) sin(*x*3)



*x*1 is translational forward motion (position x),

*x*2 is translational left/right motion (position y),

*x*3 is the tire angle,

*x*4 is the heading of the vehicle.

For control inputs, *u*1 represents vehicle velocity,

*u*2 represents angular rate of change for the tire angle.

The wheelbase L is defined in the Vehicle.h header file.

Using a simple discretization of these differential equations with a duration ∆*t*, we can use the following equations for motion:

*x*1(*t*+∆t)= *x*1(*t*) + ∆t *u*1(*t*)cos(*x*3(*t*)) cos(*x*4(*t*)) (2)

*x*2 (*t*+∆t) = *x*2 (*t*) +∆t *u*1(*t*)cos(*x*3(*t*)) sin(*x*4(*t*))

*x*3 (*t*+∆t)= *x*3 (*t*) + ∆t *u*2(*t*)

*x*4 (*t*+∆t) = *x*4 (*t*) + ∆t *u*1 (*t*) (1/L) sin(*x*3(*t*))

The value for *x*3 (tire angle) must always be between [􀀀-0.5236, 0.5236] radians/sec (i.e., / 6 radians/sec). If a value is commanded outside this range, then *x*3 should saturate using the above range. Ex: If the tire angle rate is commanded to be the value 0.7156, the tire angle rate should equal the maximum value of 0.5236.

The heading should always be between [0, 2). If the heading is a negative value, the heading should be converted into the range [0, 2) by repeatedly adding 2 . Ex: If the heading is -.5, the heading can be converted to -0.5+2 = 1.5.

Defined values are present for these ranges inside of State.h

# Class designs

The following class definitions must be used for the indicated classes.

Input, State, Vehicle (must use exact prescribed class definitions)

VehicleSimulator (define it at your discretion)

## 4.1 Classes that must use prescribed definitions

Please use the exact interfaces for Vehicle, State, and Input, or your alpha release may not compile. These are the only classes for which the design is fixed.

**4.1.1 Input**

The Input class holds the values for the u variables used in the kinematic model. Its interface is included on D2L as a resource for this Project.

**4.1.2 State**

The State class is similar to Input, and its interface is included on D2L as a resource for this Project. However, it does ensure that tire angle values and heading values stay within the designated ranges if the setters for the class are called.

**4.2 Vehicle**

The Vehicle class executes a control input for the designated duration. The interface is included on D2L as a resource for this Project.

The Vehicle keeps its own state, receives a control input, and updates its state.

The initial state value for the Vehicle is *x1 = 0, x2* =0, *x3* = 0, and *x4* = 0, e.g., (0, 0) position, tire angle of 0, and heading of 0.

The Vehicle class assumes that any invalid input values have been removed, so it does not do any error checking. The class does not permit anyone to update its state, except by providing an Input object through the stateUpdate method prescribed in the header file.

**4.3 Vehicle Simulator**

The vehicleSimulator class does the following:

* Reads all the control inputs from a provided file, as you read them in, don't worry about how they're sorted.
* After reading the control inputs the class puts them in order of their timestamp. The interface for the sort function should be:

void sort (); *// performs quick sort*

* Then, you have to validate the control input, The interface for this method should be:

*// should be called only after the vector is sorted*

*// returns true if the vector in Input objects is valid*

bool validate ();

The vector of Input objects is invalid if any of the criteria from Section 2.1 are discovered.

* Once you have an (ordered) set of control inputs, you have to store it in the output vector and pass it to the vehicle.
* Then, the class writes a vector of Output objects to a file of the provided filename.

**5. What goes in main, then?**

In your main function, you should instantiate your classes, check your arguments and return the usage statement. Put all the logic and error handling for file I/O, initialization of the Vehicle, etc., in your methods as defined in your other files and in the order that makes sense. You want your main to be very simple, so that if you want to reuse your code, you can do so without a main function.

# Alpha Submission

Your alpha assignment is to implement the Vehicle class methods, and turn in your Vehicle.cpp file to Project 4 (alpha) on ZyBook. We will use your file with our own main function and test files, to check the behavior of each of the Vehicle methods we prescribe. Use the same vehicle.h file given above and provided from the D2L, or be in danger of your alpha not compiling.