

C++ assignment 2

Building on the previous work using objects in C++, the purpose of this assignment is to give you experience of designing object orientated programs (almost) from scratch. Provided (<http://seis.bris.ac.uk/~db9052/teaching/>) is a simple simulation of a set of cars driving around in a circle, written in C++ *but without using objects*. Your task is to rewrite and extend this program using objects and object inheritance.

Overview of the simulation

For the purposes of this simulation, each car has a position x_i and velocity v_i in polar coordinates where the subscript i denotes the i -th car in the circle. The motion of each car is governed by the optimum velocity model (a system of ordinary differential equations):

$$\frac{dx_i}{dt}(t) = v_i(t), \quad (1a)$$

$$\frac{dv_i}{dt}(t) = a(V(x_{i+1}(t) - x_i(t)) - v_i(t)), \quad (1b)$$

where a is an arbitrary constant governing the rate of acceleration/deceleration of the cars and V is the optimum velocity function

$$V(h) = \tanh(bh - 2) + \tanh(2), \quad (2)$$

where b is an arbitrary constant governing the headway needed to obtain the maximum velocity. (The distance from one car to the car in front $x_{i+1} - x_i$ is called the *headway*).

This model has interested researchers in recent years due to its replication of spontaneous traffic jams; try changing the number of cars in the simulation provided from 20 to 25 and note the different dynamics.

To step forwards in time the explicit Euler scheme is used to integrate (1):

$$x_i(t + \Delta) = \Delta v_i(t), \quad (3a)$$

$$v_i(t + \Delta) = \Delta a(V(x_{i+1}(t) - x_i(t)) - v_i(t)), \quad (3b)$$

where Δ is the *step-size* of the integrator. (You will learn more about numerical methods for integrating differential equations next year; explicit Euler is the simplest method possible.) The step-size is fixed to a small value; if the step-size is set too large then the numerical integrator becomes unstable and produces rubbish.

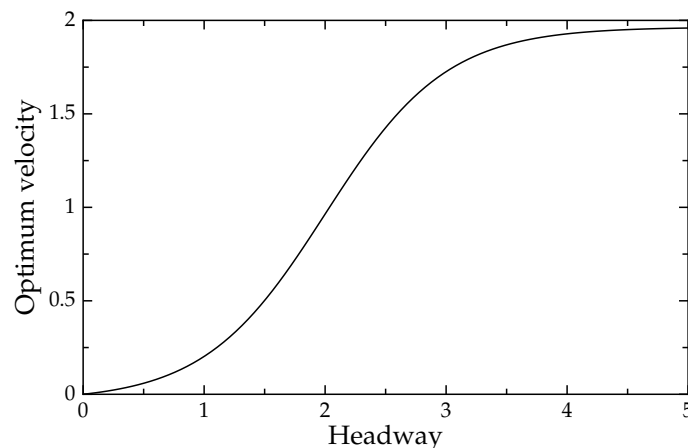


Figure 1: The optimum velocity function.

More detailed instructions

Note: read all the instructions before starting; some of the later instructions may influence how you answer the earlier questions (i.e., how you design your objects).

For this assignment:

1. Go over the code provided and give a *brief* description of what each part does.
2. Determine what objects are appropriate for the simulation (use the previous exercise sheets for inspiration):
 - What data does each object need?
 - What functions does each object need?
 - Justify your choice. This is a crucial part of software design; often there are many possible choices, why have you chosen the particular design you have?
3. Rewrite the code provided to use the objects you have designed.
4. Extend the code with additional sub-classes (object inheritance) to create different types of vehicle:
 - e.g., vans and lorries which may be drawn differently and have different acceleration characteristics.
5. Modify the code so that vehicles can appear and disappear *during* the simulation. Extra marks will be awarded for use of linked lists and/or *correct* usage of the standard template library (STL). (Use of the standard template library will require additional background reading.)

This assignment is intended to be open-ended. Completing the above will get you a decent mark; additional marks will be awarded for creativity and programming style. Possible extensions include

- using a figure-of-eight track,
- adding user controlled traffic lights,
- ...

This assignment counts for 30% of the overall course mark. You should submit your code and a *short* report (no more than 6 pages) describing your work by 5:00pm on Friday 14th May (week 24). Please submit your work via the online coursework system as C++ code and a PDF report (see the course webpage for the appropriate links). Brief feedback will be provided by the end of week 26. Please note that reports will not be returned to you.

If you have any problems, please email me, David Barton, at <david.barton@bristol.ac.uk>.

Side note: frame-rate control

Some of you may have noticed in the bouncing ball example that the balls can suddenly increase in speed. This occurs when the step-size used for integration is too large. Since the step-size is taken from the system clock, this can occur whenever you interrupt the program execution (moving the window, etc). To avoid this problem, the code provided uses a fixed step-size and some frame-rate control code to attempt to maintain a constant integration rate.