CS452 – Spring 2012
Project Milestone 1

Bill Cowan
University of Waterloo

A. Introduction

We require two milestones in the development of your project.

1. The first milestone requires you to track the train in real-time. In essence, this means that as a single train moves around the track you update a display showing its position in real-time. To demo this milestone you must implement a rudimentary route-finding algorithm. Be certain that it is compatible with the more elaborate route-finding required for the second milestone.
2. The second milestone consists of tracking the position of two trains simultaneously, finding the shortest route to a given destination in the presence of unusable sections of track, and implementing a method which ensures that two simultaneously moving trains do not run into one another.

For each of these milestones you will give us a short demo, and hand in documentation describing the algorithms and data structures used to realise your objectives.

Comment. This documentation will be part of your final project documentation, so doing it well will save you time in the long run.

B. Description

B.1. General Description

The track contains a collection of 'landmarks': turn-outs[*], sensors, whatever else you wish. You know the distances between them and the topology of the track with respect to them.[†] We expect the demo program to allow the operator to do the following by typing at the terminal keyboard.

[*] I use 'turn-out' for what you normally call a switch because it is easy to confuse the noun, switch, and the verb, switch.

[†] Please note that the two tracks have different topologies and distances. The differences are small, but significant. You are allowed to choose which track is used in your demo, but you should be prepared to run on both tracks in case of track malfunction during the demos.

1. Drive a train around the track, switching turn-outs and changing speed. This is the capability required by the fourth kernel assignment, so you should have it already functioning.
2. There should be a calibration for the train that allows you to stop it at a given location by automatically calculating the time at which a stop command should be given.
3. You should be able to find a route, preferably the shortest roote from where it is to a given location, and to follow it successfully.
4. The calibration should be used to provide a display on the terminal, from which we can read:
   i. the train's location in real-time in the form of a landmark, a distance (in centimetres) before or beyond it plus the direction of travel of the train,
   ii. the expected and actual time (or distance) of the train at the previous landmark, and
   iii. the information provided in your final kernel assignment, including the .

Your program is not expected to be robust against errors in sensor reports or the state of the turn-outs.

B.2. Technical Details

Here are a few technical details, some hints and some fine points about what you are and are not allowed to do.

1. You may start the train anywhere you choose on the track, but the train should determine its initial position and the direction it considers to be 'forward' by moving until it trips sensors.
2. You may use calibration files you have created on other occasions or you may build a calibration file at the beginning of the demo.
3. You may use the distances given in the course account, or you may measure your own.
4. Some trains actually have twice as many speeds as you think. They travel a little faster when they decelerate into a new speed than when they accelerate into it. You should check this by measurement.
5. The trains have programmed acceleration and deceleration profiles. You may calibrate them, and many of you will want to do so because knowing the profiles strongly enhances performance during the final demo.*
6. Using the command sequence, 'speed 0' followed immediately by 'reverse', which stops the train instantly, is not allowed. Among other things, it is hard on the locomotives.

* This is more essential than you might think. In your completed project trains will stop to allow other trains to pass. You must know when to give the stop command to have the train stop in a safe location. Also, routes that have reverses, often require the train to accelerate and decerate several times without tripping sensors.

7. There are a couple of curves on the track where trains stall if they are travelling at too low a speed. Your trains must be run fast enough to prevent stalling.
8. The brightness of the headlight tells you how much electrical resistance there is between the track and the train. When cleaning the track and engine contacts to decrease the resistance remember that there are two places where electrical contacts are made between the engine and the track.
9. There is no way to determine whether the pick-up is at the back of the locomotive or at its front. A common strategy simply assumes that the locomotive is twice as long as it actually is, with the pick-up in the middle. If you want to do better, a simple method is to start the train in a known orientation, and to keep track of the orientation as the train travels.
10. The assignment is trivial if you run the trains slowly enough. We expect to see them running at speeds 8 to 12 during your demo.

This list may grow as questions come in.

c. Hand in

Hand in the following, nicely formatted and printed.

1. A description of how to operate your program, including the full pathname of your executable file which we may download for testing.
2. A description of the algorithms and data structures used to calibrate the train and the track, to maintain knowledge of the train position, and to recover from errors. Remember to tell us how they work and why you chose them. We will judge your program primarily on the basis of this description.
3. The location of all source code you created for the assignment and the MD5 hash of each file. The code must remain unmodified after submission until the assignments are returned.
4. A list of all files submitted.