cs452 – Spring 2016
Stopping

Bill Cowan
University of Waterloo

I. Stop When the Train Hits a Sensor

The train is travelling at a constant velocity v; it hits sensor m at location S_m at time t; its position $x(t) = S_m$. What happens next? You find out that the train has triggered a sensor by requesting a sensor report from the train controller at time t_1, which may be before or after t. After a time interval τ_1 the response returns: note that $t < t_1 + \tau_1$. Between the train hitting the sensor and your software knowing that it hit the sensor $t_1 + \tau_1 - t$ seconds have elapsed, during which the train has travelled $v(t_1 + \tau_1 - t)$ cm. to

$$x(t_1 + \tau_1) = S_m + v(t_1 + \tau_1 - t).$$

Some of these times are not directly measurable but may be estimated.

Assume that the 'speed zero' command is issued immediately. τ_2 seconds elapse while the command is transmitted by your program and received by the train controller and then by the train's microcontroller which the enters its 'speed zero' procedure. The location of the train when the procedure starts is

$$x(t_1 + \tau_1 + \tau_2) = S_m + v(t_1 + \tau_1 + \tau_2 - t).$$

The procedure imitates the gradual slowing of a real train, taking τ_3 seconds to slow the train to a stop at

$$x(t_1 + \tau_1 + \tau_2 + \tau_3) = S_m + v(t_1 + \tau_1 + \tau_2 - t) + \sigma_3.$$

$\sigma_3 \neq v\tau_3$ because the train is decelerating.

Only a few of these quantities are easily measurable. The location S_m and the distance σ_3 are easily measured using a tape measure. The time t_1 and the time interval τ_1 are easily measured. The time interval $\tau_2 + \tau_3$ is also measurable, but not so easily. Note that 'measurable' does not imply exactly measurable.

I.1. Stopping Distance

You are free to define stopping distance however you like. Here are a few alternatives. They are evaluated according to scenarios likely to occur during your project.

- $d_s = \sigma_3$. This has little to recommend it because without very intrusive measurement, which we forbid, you have no idea when to emit a command that will be executed by the train starting at a specific time.
- $d_s = v\tau_2 + \sigma_3$. This is the distance the train travels between giving a stop command within the application program and the time that it stops. It is useful when the program maintains internally the exact space-time location of the train.
- $d_s = v(\tau_1 + \tau_2) + \sigma_3$. When we discuss measurement errors you will see that measuring the location of a train often generates an estimate that is, on average, behind by $v\tau_1$, which might make this a good definition of stopping distance.

The definition you choose must stay the same throughout your project. (Please see §2 before making a final choice.)

I.1. a. Polling the Sensors

It is a common strategy to have a server task that uses a worker to poll the sensors and gives out information about recently triggered sensors to it clients. Polling runs continuously, a sensor request immediately following the end of the previous sensor report. A rough model of polling would assume that it operates cyclically with cycle time π. For example, if a sensor query is initiated at t_1, other queries will be initiated at $t_1 + \pi$, $t_1 + 2\pi$, and so on. Because the new query is produced immediately on reception of the results of the previous query $\tau_1 = \pi$ when the sensors are polled. The cut-off time for a sensor trigger to be reported on poll n is $t_1 + n\pi - \phi$, where ϕ is the time between when the sensor is read and when the reading is time-stamped.

The triggering of a sensor is equally likely at any time in the cycle. Thus, if the triggering of the sensor is reported at $t_1 + n\pi$ the sensor was actually triggered before $t_1 + n\pi - \phi$ and after $t_1 + (n-1)\pi - \phi$. The distribution of possible trigger times is $f(t) = 1/\pi$ for $(n-1)\pi < t - t_1 + \phi < n\pi$, and zero otherwise. The mean of this distribution is $\mu = -t_1 + \phi + (n-1/2)\pi$; the width is π; the standard deviation is $\pi/\sqrt{6}$.

The assumption that polling is perfectly periodic is, of course, violated when commands are being given to change the speed of a train or the direction of a switch, which are of higher priority. Nonetheless, periodic polling is a good approximation and will be used in the remainder of this document.

The polling period, π, is easily measurable. In previous terms students have measured it to be a little less than 100 milliseconds.

I.2. Stopping Time

Stopping time is important. Suppose we give a speed zero command. Between the command and the train stopping we do not know the position of the train. The expiration of the stopping time resumes knowledge of the train's position.

The simplest method for measuring stopping time – Give a speed zero command while starting your stopwatch, then stop the watch when you see the train stop – doesn't work very well. There are two reasons.

1. It is hard to see the moment when the train stops. Among other things the temporal granularity for vision driven responses is several hundred milliseconds.
2. The train stops too quickly for stopwatch or second hand measurement to be sufficiently precise. Assume, just to find out what values to expect, that the train travels at 50 cm/sec, has a stopping distance of 50 cm and stops with constant acceleration. The stopping distance $\sigma_3 = (1/2)a\tau_3^2$ and the stopping time is $\tau_3 = v/a$. Eliminating a, $\tau_3 = 2\sigma_3/v$. The stopping time is about two seconds, too short to time with a second hand or stopwatch.

The next section shows a better way of measuring stopping time.

II. Constant Velocity Calibrations

II.1. Calibrating Velocity

The expressions in the above section all assume that the velocity, v, is constant, but unknown. How do we measure it? Assume we are polling the sensors: sensor m is triggered on poll n; sensor $m+1$ is triggered on poll n'. The difference between the means of the time distributions is

$$-t_1 + \phi + (n' - 1/2)\pi - (-t_1 + \phi + (n - 1/2)\pi) = (n' - n)\pi.$$

Notice how many of the terms in the time measurement drop out when you subtract. As a general rule achieving your answer from the difference between two similar measurements always benefits you.

II.1. a. Instrumenting Velocity Calibration

One important reason for reading sensors regularly is keeping track of the position of a train. Other useful information also comes available, and on the principle of sqeezing sll possible information from each use of the train communication bottleneck we should use it as completely as possible.

One use is monitoring calibration quality: deterioration of the calibration is as early sign that train performancee is changing, which is common when the trains are heavily used. Each time the train passes a sensor it is easy to use the calibration to compute its time at the next sensor. Then, on arriving at the next sensor one can compare the calibration prediction to the actual time, giving an error that can be expressed in milliseconds or, multiplying by the velocity, in cm. Putting this result onto the terminal display as the demo progresses gives real-time feedback about the quality of the calibration.

II.1. b. Dynamic Calibration

Another example of squeezing extra utility from sensor readings is dynamic calibration. Each time a train passes two sensors you can, dividing the distance by the time, calculate its average velocity between the sensors. The result can be used to update the calibration. A common algorithm mixes a fraction α of the new measurement into the old one to get a revised value:

$$v_{revised} = \alpha v_{new} + (1 - \alpha) v_{previous}.$$

The calibration them improves to match the condition of the train and the track as the demo proceeds. Small values of α gradually tune a calibration when conditions are stable; large values of α rapidly create a new calibration.

When a favourite train dies close to the demo, dynamic calibration allows you to have a calibration that improves as the demo proceeds.

II.1. c. Predetermined Velocities

The procedures we have seen so far in this section enable you to drive the train at one of the velocities corresponding to one of the discrete speeds provided by the microcontroller. If, however, you wished one locomotive to follow another closely you wouldn't be able to do so: each locomotive travels at a different velocity when set to the same speed. If both were set to speed ten, for example, the one behind would either fall further and further behind or catch up and collide with the one in front.

The obvious way to solve this problem works fine.

- The two locomotives cross sensors in quick succession. Establish the time between successive triggerings of the same sensor as the measure of distance between the locomotives.
- If the time is greater than the target time then increase the speed of the following locomotive.
- If the time is less than the target time then decrease the speed of the following locomotive.

Clearly, changes in speed must be frequent to minimize variation of the inter-locomotive distance.

Driving a train at a pre-determined velocity is a simple adaptation of this procedure. Once you have accelerated the train to a constant velocity, measure its velocity as it passes the next two sensors. If its velocity is above the velocity you desire decrease the speed; if it's below, increase it. Most likely you will be changing its speed frequently between speeds that give velocities immediately near the desired velocity.

The train does not actually maintain an exact velocity but oscillates near the velocity you want. In fact the train is always either accelerating or decelerating, as in the short moves section of the acceleration document, with feedback control managing the velocity.

II.2. Stopping

You can now stop anywhere on the track. The way one does so without thinking too much is the following.

1. Given the place you want to stop, $S_m + \Delta$, subtract the stopping distance, d_s, from Δ. (If the answer is negative measure the stopping location from

the sensor before m and subtract again, continuing to go back until you reach sensor m' where the difference, $\Delta' - d_s$, is positive.)
2. Wait until you receive a report that sensor m' has been triggered.
3. Wait a further $(\Delta' - d_s)/v$ seconds.
4. Give the speed zero command.

This should do it but we must analyse the error to choose the correct definition of stopping distance. (We assume sensor polling in what follows. Generalizing to non-periodic sensor reading is straightforward.)

On average the sensor reading is received $\phi + \pi/2$ seconds after the sensor was triggered. During that time the train has travelled $v(\phi + \pi/2)$ cm. Before the speed zero command is given the train travels $\Delta' - d_s$ cm. After the speed zero is given the train travels $v\tau_2 + \sigma_3$ cm. The total distance should be Δ' for a perfect stop, which gives

$$d_s = v(\phi + \pi/2 + \tau_2) + \sigma_3,$$

the third of the definitions above. It is usually the best definition, but remember carefully what it includes when integrating acceleration and deceleration.

Knowing the velocity at which the train travels for a given speed, and knowing the stopping distance you can now stop the train at any location. We now show how you can use this capability to measure the time it takes the train to stop.

II.3. Time to Stop

We stop a train in a particular location because we want it to be exactly there, which it is only after it has stopped moving. How long does it take a train to stop? We can measure in the following way.

1. Find a location on the track that it exactly the stopping distance before a sensor.
2. Run the train at constant velocity towards that location.
3. When the train is the stopping distance away from the location measure them time and give the speed zero command.
4. When the sensor triggers measure the time. The difference beween the two time measurements is the stopping time.

This procedure triggers sensors twice, so it's important to look carefully at the time sequence.

The first sensor trigger determines the most recent location of the train. If thie report is uncorrected then the train is ahead of our measurement by $v(\phi + \pi/2)$ cm. When the speed zero command is given the train travels a further $v\tau_2$ while the command is on its way to the train, and σ_3 while the stopping procedure is carried out. At the very end of this travelling the sensor is triggered and there is a wait of $\phi+\pi/2$ seconds until the sensor report reaches your program. This, if your train position is correct the stoping time is correct; if your train position is behind by $v(\phi + \pi/2)$ cm., then your stopping time is overestimated by $\phi + \pi/2$ seconds, which is usually inconsequential.

As a check that your stopping is being implemented correctly look carefully at the sensor and pick-up without moving the train. The distance of the sensor beyond the leading edge of the pickup is within the random error of your stopping distance. If the train stops before the pickup interacts with the sensor then your stopping distance is too long.

II.3. a. Automated calibration

The time to stop ideas above can be elaborated into an automated calibration procedure that calibrates the stopping distance and the stopping time simultaneously. The idea is to try to stop on a sensor: when the sensor is triggered you shorten your estimate of the stopping distance/time and when the sensor is not triggered you lengthen your estimate. Iteration with progressively smaller changes gets you to the point where the pickup just touches the sensor enough to activate it.