cs452 : Winter 2017

The Kinematics of Train Calibration

Bill Cowan
University of Waterloo

1. Introduction

Implementing a train project produces an application program that writes bytes to interfaces and reads bytes from interfaces. Whatever is connected to the interface determines how the bytes are interpreted. From your previous courses you are familiar with the terminal's interpretation of bytes you write to it, and the meaning of bytes you read from it.

At a high level you understand the interpretation of bytes sent to and from the train controller: they set speeds and switches; they tell your application which sensors have been recently triggered. You learned which bytes mean what in assignment zero and in the fourth part of the kernel, and used this information to form mental plans that you carried out by monitoring the track with your eyes and typing commands at the terminal. Turning the commands into appropriate bytes is easy.

In the train project we replace you by an application that monitors the track without eyes, forms a plan without a brain and carries out the plan without fingers. Carrying out the plan you already know how to do; making it is not much harder; but you can't do much of either without the ability to determine the state of the trains and track, which is performed effortlessly by human vision and cognition. The remainder of this document describes a few techniques for doing so.

1.a. Calibration as a process

The ease with which we monitor the world using our senses and with which we build an internal model of the world by combining perception and memory is deceptive. Decades of research into computer perception fail to offer digital model-building of similar quality. Fortunately, the train controller offers us a small range of affordances, keeping the model we build simple.

This document shows a variety techniques for building mappings between a programmed model of the train set and the set itself. Some mappings you use are completely static. For example, you will most likely model the track with a graph, and most aspects of the graph-track

mapping do not change. But the state of the turn-outs and the positions of trains on the track do change in response to actions taken by the program. Building a model that integrates the discrete changes in turn-outs and continuous changes in train location is only the beginning of building a satisfactory model.

The relationship between the commands you give a turn-out and its state is direct; the relationship between the commands you give a train and its position on the track is indirect: what happens when you give a speed command to the train: over a long* time its velocity changes gradually, until it reaches a new constant velocity. To know a train's velocity at and time you need a calibration that tells you in quantitative terms what the change in velocity actually is. Thus, part of the state of the trains is a set of functions that describe how the train responds to speed change commands.

It is tempting to treat these functions as static properties of the trains: then you can measure them once and use the measurement for the rest of time, which means until the end of your final demo. Unfortunately they are actually not static: trains run more slowly as they get older, more quickly if the track has just been cleaned and not at all if they have reached EOL.

Under these circumstances we would like to think of a calibration as something dynamic, a process rather than a result. Keeping track of the train means that our program does many measurements of train position as it executes. We can use these to update the current calibrations of running trains: the result is a calibration that better decribes the current state of a changing train. One example is dynamic calibration of velocity (SECTION 5.D); there are many other places where similar approaches pay off.

2. Kinematics

You probably remember whichever elementary physics course you took dividing basic mechanics into two subjects, kinematics, the mathematics of motion, and dynamics, the mathematics of forces. The trains we use have simulated dynamics: a small light locomotive driven by an electric stepper motor has dynamics that are very different from those of a real freight train: several two hundred ton locomotives pulling two hundred ore cars, each of a hundred tons. The latter takes fifteen minutes and a dozen kilometres to reach travelling speed, and even more to stop, which we can't tolerate within a thirty minute demo.

Typical buyers of model trains, of which the trains course is not one, find the ordinary dynamics of electric trains unsatisfactory, and would like dynamics similar to real trains, but scaled down to fit into a basement. The

---

* 'Long' always means long compared to the temporal granularity of the controlled system, which for the train is about 10 milliseconds, during which the train moves about half a centimetre.

trains we use accomplish this by using a microcontroller inside the locomotive. When a command arrives asking for a change in speed the microcontroller runs a program that gradually changes the velocity of the locomotive with an acceleration that more or less models the scaled down performance of a real train. The desire of the model railroader for realism plays a role in the pseudo-dynamics of train behaviour we will use when considering what assumptions are reasonable, but that will be the extent of our interest in dynamics.

The basis of kinematics is the observation that the location of the train on the track is a time-varying function, $x(t)$. In practice, denoting the exact position of a train on a track with turn-outs* that divide the track into many connected segments. This problem is discussed, probably too extensively, in class. Discussing kinematics we usually are interested in local properties, so describing the position of a train by a single real number is quite satisfactory.

The velocity of a train†, $v(t)$, is the rate at which the train's location changes, its derivative:

$$v(t) = \frac{d}{dt}x(t) .$$

Integrating this equation gives us the distance travelled by a train between one time and another:

$$x(t) - x(t_0) = \int_{t_0}^{t} v(t')\,dt' .$$

In our discussion we also use higher order derivatives of location:

- acceleration, the derivative of velocity, $a(t) = \frac{d}{dt}v(t)$, and
- jerk, the derivative of acceleration, $j(t) = \frac{d}{dt}a(t)$.

---

* You probably know a turn-out as a switch, which allows one track to diverge into two or two tracks to merge into one. 'Switch', of course, is both a noun and a verb, an ambiguity that can make writing confusing. In class, and in these notes I try consistently to use 'switch' as a verb and the less common term 'turn-out' as a noun. That is, your software switches a turn-out.

† There are two closely related, but different concepts associated with a moving train. One is the number in the `tr` command, an integer between zero and fourteen, which is used by the microcontroller to decide how quickly to turn the wheels: we call this the 'speed' of the train. The other is the rate at which the train moves along the track, usually measured in cm/sec. We call this quantity the 'velocity'. Because the train's velocity changes gradually after a command to change speed is given we will be interested in the velocity as a function of time.

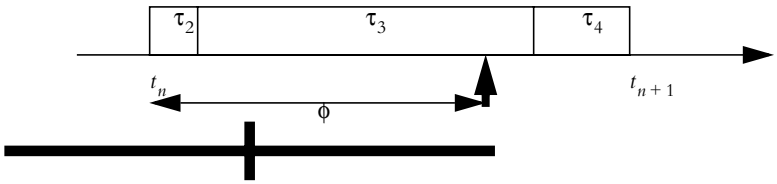
3. Measurement

Let us assume that a train triggers sensor n, at location s_n, at time T_n. Thus, $x(T_n) = s_n$, but our program does not know this fact because it only knows about sensor reports. When does it see the ensuing report and what can it conclude from it?

A sensor report begins with a request from the program for a sensor reading. Let us suppose that the request is initiated at time t_n, which is known to the program. After initiation, the request passes through several stages (each annotated with the time taken*):

1. it is encoded into a sensor request command ($\tau_1 \cong 0$ msec);
2. the command is transmitted through the train UART ($\tau_2 \cong 8$ msec);
3. the train controller gets results from the five blocks of sensors ($\tau_3 \cong 40$ msec);
4. the result is transferred back through the train UART ($\tau_4 \cong 40$ msec) and;
5. the result is decoded into a sensor report ($\tau_5 \cong 0$ msec).

These times are estimated based on a typical sensor polling rate of 11 Hz, and on the UART transmission rate of 2400 bps. The end of stage 5 is also known to the program. The diagram below, which is roughly to scale, shows the time line of this process. (The upward arrow indicates the last time as which a sensor trigger will be included in the current poll. The thick horizontal line is the interval during which the sensor must have been triggered to be included in poll n. The cross bar at its centre is the expectation $E(T_n)$.)



Assuming that the sensors are polled continuously then T_n is a random variable uniformly distributed on $[t_n - \tau + \phi, t_n + \phi]$, where ϕ is the phase, the time within request processing by which the trigger must occur to be reported on this poll, presumably near the end of time interval τ_3 and $\tau = \sum \tau_i$ is the amount of time taken by a single poll of the sensors. That is, the expectation of the time at which the sensor was triggered is

* I assume that you can figure out the numbers given on your own. If you have any doubts about a number ask about it in class.

$$\hat{t} = E(T_n) = t_n + \phi - \frac{1}{2}\tau = t_{n+1} + \phi - \frac{3}{2}\tau .$$

We know that the location of the train when the sensor was triggered is $x(\hat{t}) = S_n$, and can establish the location of the train at any other time if we know its velocity, v: $x(\hat{t} + t) = S_n + vt$. In particular, the position of the train. In particular, t_{n+1} is the earliest time at which a program can know that a sensor was triggered in the interval

$$[t_n + \phi - \tau, t_n + \phi] = [t_{n+1} + \phi - 2\tau, t_{n+1} + \phi - \tau] .$$

3.a. Difference Measurements

When we want to calculate velocity we make two measurements, usually at sequential sensors. The distance between the sensors we measure as $\Delta x = s_{n+1} - s_n$. The difference in time is the random variable $\Delta T = T_{n+1} - T_n$, which has expectation $t_{n+1} - t_n$. What is important here is that when we measure the difference the constant term in the error, which physicists call the systematic error, drops out. As much as possible we always try to measure differences. You will see another example of this principle in action when we measure stopping distances.

4. Stopping Distance

Let's start by making the most elementary of measurements. What distance does the train travel when it is stopping? The idea is simple: we wait until we receive a sensor report, then send a stop command to the train, wait until it stops, then measure the distance from the sensor to the position of the train. First we must think about what we consider the position of the train to be, then we will be ready to measure.

4.a. The Location of the Train.

The train locomotives that we use in this course are about twenty centimetres long. We want to position them to a fraction of their length and can do so only if we choose something smaller than the locomotive itself to be its position. One popular choice is the leading edge of the pick-up, the piece of metal between the wheels that contacts the little spikes in the middle of the cross-ties. Another is the front or the back of the locomotive.

The point you choose is called a fiduciary mark, a term used extensively in surveying. It doesn't matter which point you choose, only that you consistently use the same point.

When measuring track it is necessary to choose a fiducial point on the turn-outs, and you will save a lot of annoyance if you choose the same point on every turn-out.

4.b. Measuring the Stopping Distance

Measuring stopping distance is easy: send the train a 'speed zero' command and measure how far it travels after it receives the command.

But where is the train when the command is given? We start by giving the command immediately after receiving a sensor report. Doing so divides the time of travel after the sensor is triggered and before the train stops into four parts:

1. the time between the sensor trigger and the sensor report,
2. the time between the sensor report and the command to stop, which is negligible during the measurement,
3. the time between the stop command and the train starting to stop, and
4. the time that the train takes to stop after receiving the stop command.

Were we to define the stopping distance as the distance travelled during the fourth time interval we would have to remove the distance travelled during the first three intervals, the length of which can only be roughly estimated.

We easily measure the distance travelled during all four intervals and are fortunate that it is the distance most useful to us. To see why remember that we most often want the train to stop at a particular location Doing so involves something like the following sequence:

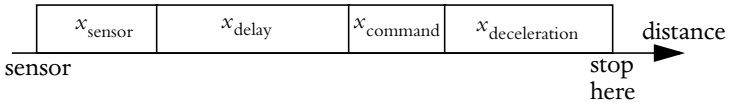
1. measuring back by the stopping distance from the desired stopping location to the point at which the stop command should be given,
2. finding the sensor before that location,
3. noting when the train triggers that sensor, and
4. delaying a calculated time, the distance past the sensor divided by the velocity, before giving the stop command.

In practice this amounts to the following.

1. Wait until your program receives a report of the sensor being triggered. At this point the train is past the sensor by an unknown amount.
2. Calculate the time to delay, and sleep for that long.
3. Wake up and give the stop command.

What algorithm should be used to calculate the time to delay?

The answer depends on the definition of stopping distance you use. Consider the break-up of the distance to stop into four components shown immediately below, where the train travels through the first three at the constant velocity v. If we define the stopping distance as $x_{\text{stopping}} = x_{\text{sensor}} + x_{\text{command}} + x_{\text{deceleration}}$ then $\tau_{\text{delay}} = x_{\text{delay}}/v$. This is very convenient: the stopping distance is defined as what is easiest to measure, and the delay time is easy to calculate



4.c. Taking Variability into Account

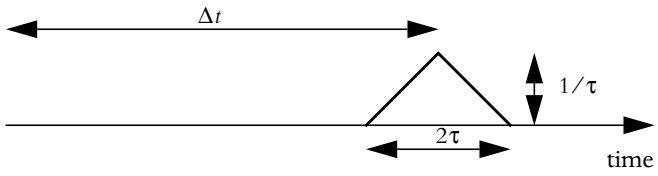
Because the actual location of the train varies when you get a sensor reading, the measured stopping distance should also vary. The probability distribution on the train location at a sensor reading is uniform on an interval $[t_n - \tau + \phi, t_n + \phi]$ you should see the stopping distance as uniformly distributed on an interval of width $v\tau$. (Don't concern yourself that you do not know the velocity of the train; we measure it in section 5 of this document, and you can then do the consistency check.)

5. Velocity

Velocity requires only a measurement of difference: that is, we measure the time at which the train triggers one sensor and the time at which it triggers the next sensor and subtract the first measurement from the second to know how much time it took the train to travel the measured distance between the two sensors. As discussed in section 3.a the systematic error drops out, substantially increasing the precision of the measurement.

5.a. Taking Variability into Account

A velocity measurement made in an context in which the sensors are polled continuously depends on the difference between two time measurements, each of which is a uniformly distributed random variable. The width of each distribution is τ, the polling interval, and the mean time between the two distributions is $\Delta t = E(T_2 - T_1) = E(T_2) - E(T_1)$. The distribution of the difference is straightforward to calculate. It is the tent-shaped distribution shown in the figure below. (A simple calculation shows that the standard distribution of the distribution is $\frac{\tau}{\sqrt{6}}$, which is independent of Δt. Thus, as the time measurement gets longer the signal-to-noise improves. This conclusion, however, depends on the velocity being precisely constant.)



In practical terms the key ratio limiting how well you can control the location of a train is $\Delta t / \tau$, where Δt is the temporal precision (or $v\Delta t$ the spatial precision) required by your project. When we say that the bandwidth of the communication channel between the ARM CPU and the train controller is the main bottleneck for improving the performance of your project, we are restating the fact that the bandwidth sets a lower limit on the value of τ.*

5.b. The Probability distribution for Velocity

The treatment so far acts as though we can get an estimate for velocity simply by dividing the distance travelled by the expectation of the time taken. Strictly speaking, however, velocity is a random variable, and we ought to calculate its expectation from its probability distribution. This may be important since the definition of V the random variable for velocity in terms of time is $V = \frac{x}{T}$ a non-linear relationship. Remember that the probability distribution for the actual time is

$$f_T(t) = \begin{array}{ll} \frac{1}{\tau}\left(\frac{t-\Delta t}{\tau}+1\right) & \Delta t-\tau < t < \Delta t \\ \frac{1}{\tau}\left(1-\frac{t-\Delta t}{\tau}\right) & \Delta t < t < \Delta t+\tau \end{array},$$

and zero elsewhere, in terms of the known values, Δt and τ.

The probability distribution for the velocity is $f_V(v) = |J| f_T(x/v)$, where J is the Jacobian of the transformation. A little algebra gives

$$f_V(v) = \begin{array}{ll} \frac{x}{\tau^2 v^3}(v(\tau-\Delta t)+x) & \frac{x}{\Delta t} < v < \frac{x}{\Delta t-\tau} \\ \frac{1}{\tau^2 v^3}(v(\tau+\Delta t)-x) & \frac{x}{\Delta t+\tau} < v < \frac{x}{\Delta t} \end{array}.$$

Using this distribution function we can calculate the expectation of the velocity, $E(V)$, which is

$$\begin{aligned} E(V) &= \frac{x\Delta t}{\tau^2}\left[\log\left(\left(1+\frac{\tau}{\Delta t}\right)\left(1-\frac{\tau}{\Delta t}\right)\right)+\frac{\tau}{\Delta t}\log\left(\frac{1+\frac{\tau}{\Delta t}}{1-\frac{\tau}{\Delta t}}\right)\right] \\ &= \frac{x\Delta t}{\tau^2}\left[\log\left(1-\left(\frac{\tau}{\Delta t}\right)^2\right)+\frac{\tau}{\Delta t}\left(\log\left(1+\frac{\tau}{\Delta t}\right)-\log\left(1-\frac{\tau}{\Delta t}\right)\right)\right], \\ &= \frac{x\Delta t}{\tau^2}\left[\left(1+\frac{\tau}{\Delta t}\right)\log\left(1+\frac{\tau}{\Delta t}\right)+\left(1-\frac{\tau}{\Delta t}\right)\log\left(1-\frac{\tau}{\Delta t}\right)\right] \\ &= \frac{x\Delta t}{\tau^2}\left[g\left(\frac{\tau}{\Delta t}\right)+g\left(-\frac{\tau}{\Delta t}\right)\right] \end{aligned}$$

where $g(\gamma) = (1+\gamma)\log(1+\gamma)$. We can expand this expression as a series in $\frac{\tau}{\Delta t}$, which would normally be much less than one. First for $\gamma \ll 1$

---

* The preceding statement is based on the centrality of sensor polling in the software architecture of a project. Other architectures may experience different limits.

$$\begin{aligned} g(\gamma) &= (1+\gamma)\left(\gamma - \frac{1}{2}\gamma^2 + \frac{1}{3}\gamma^3 - \frac{1}{4}\gamma^4 + \ldots\right) \\ &= \left(\gamma + \frac{1}{2}\gamma^2 - \frac{1}{6}\gamma^3 + \frac{1}{12}\gamma^4 - \ldots\right) \end{aligned}.$$

Then

$$\begin{aligned} E(V) &= \frac{x\Delta t}{\tau^2}\left[g\left(\frac{\tau}{\Delta t}\right) + g\left(-\frac{\tau}{\Delta t}\right)\right] \\ &= \frac{x\Delta t}{\tau^2}\left[\left(\frac{\tau}{\Delta t}\right)^2 + \frac{1}{6}\left(\frac{\tau}{\Delta t}\right)^4 + \ldots\right] \\ &= \frac{x}{\Delta t}\left(1 + \frac{1}{6}\left(\frac{\tau}{\Delta t}\right)^2 + \text{higher order terms}\right) \end{aligned}.$$

The lowest order term is the naive estimate we get if we ignore random variables; it is exact when the polling time is negligible. The correction term is very small, only 1% when $\Delta t = 4\tau$.* Short moves (SECTION 7) discusses other methods for handling velocity, which are appropriate when the time of travel is small.

5.C. Monitoring your velocity calibration

As a train travels your program uses reports of triggered sensors in order to know where it is on the track. You can† display these reports on the terminal as a way of monitoring the quality of your current calibration.

To be specific, when a train is travelling at constant velocity for its speed and it triggers a sensor at time t_0, your program knows which sensor it expects the train to trigger next, and the distance x_n to that sensor. Using the calibrated velocity for its speed, v_c. it can predict that the next sensor will be triggered at $t_p = t_0 + \frac{x_n}{v_c}$. If the next sensor is actually triggered at t_n, then your calibration is placing the train at the wrong location by $v_c(t_n - t_p)$. You can write the current values of t_p, t_n, $t_n - t_p$ and $v_c(t_n - t_p)$ somewhere in the terminal window, and whenever anything goes wrong you can easily check the success of your calibration.

5.D. Dynamic Calibration of Velocity

The previous section described how to use the predicted time of the next sensor trigger as a diagnostic for the quality of the current calibration. It is an easy extension to use this information to update the current calibration. Passing the next sensor allows you to calculate the average velocity of

---

* Remember that Δt is the time between triggering of sensor as the train travels.

† We say 'can' in this document, but the instructions for the milestones say 'must'.

travel between the two sensors: $v = \frac{x_n}{t_n - t_p}$. You can use this to update the calibration: a simple heuristic would update it as $v_c \leftarrow (1-\alpha)v_c + \alpha v$, where α is a parameter you can set at the terminal.

As well as the obvious refinement of a calibration as the train or track deteriorates this technique has obvious value if a favourite train dies at the last minute, forcing you to run with an uncalibrated train. You can start the demo with the new train and the old calibration using a relatively large value of α, such as one-quarter. Then your calibration will gradually improve as your demo proceeds. When you are satisfied you can adjust the value of α to a more normal setting, five to ten percent.

In this section we have assumed for simplicity that the train is travelling at a constant velocity. If you build a model of acceleration and deceleration (SECTION 8) a similar technique can be used to refine that model at run-time.

5.E. Moving a Train at an Intermediate Velocity

The simple calibration described above (SECTION 5.B) allows you to run a train at a limited number of constant velocities. If you want to run a train at a different constant velocity there is an easy heuristic. Associated with the velocity you wish, v, the are two speeds with velocities immediately above, v_a, and below, v_b, the velocity you wish to use. Ignoring acceleration and deceleration effects, which more or less cancel each other, we can run the train at the higher velocity β of the time and at the lower velocity $1-\beta$ of the time. β is determined by the solution of

$$v = \beta v_a + (1-\beta)v_b,$$

which is

$$\beta = \frac{v - v_b}{v_a - v_b}.$$

The result is an average velocity that is correct, and the velocity varies little enough to be almost unnoticeable.

If this approach is insufficiently precise there are improvements possible in the spirit of dynamic programming: every time a sensor is triggered test how far ahead or behind the train is, and adjust β accordingly.

5.F. Using Polling as a Constraint

6. Stopping

6.A. Stopping Anywhere on the Track

Knowing the stopping distance, defined as the distance the train stops after a sensor reading is obtained by the program, and the velocity it is possible to stop at any location on the track. Here is the procedure.

1. The train is given a destination, so many centimetres after or before a landmark.
2. The train finds a route to the destination. If the distance to the destination is less than the stopping distance then a longer route is needed.
3. The train measures back along the route to find the place where give the stop command, which is so many centimetres, x_s, past a sensor.
4. When the sensor is triggered the train delays for $\frac{x_s}{v}$, then gives a speed zero command.
5. The train then continues reading sensors until the train stops, using the sensor readings to update its estimate of the final stopping position.

The final step requires knowledge of the time it takes the train to stop, which is discussed in the following section.

6.b. Measuring the time taken to stop

Using the procedure in section 6.a, give the train a destination that puts the leading edge of its pick-up exactly on a sensor. When the sensor preceding the stop command is triggered read the time and when the destination sensor is triggered read the time again. The stopping time is the difference between the two time readings minus the delay interval*.

There are some tricky aspects to this simple idea; most are connected to the asymmetry between stopping just a little bit short, which gives you no feedback except a time-out, and stopping just a little bit long, which gives you a positive response with a time that is close to correct.

One term a group automated the distance and time measurement using binary search. Here's a rough description of their algorithm as I understood it.

1. Choose an interval on the track: at one end of the interval the train stops before the sensor; at the other end it passes the sensor.
2. Give the train a stop signal from the middle of the window.
3. If the sensor is triggered the middle becomes the passing end of the interval and the time taken to trigger the sensor after the stop command is the new lower limit of the stopping time. The distance from the middle of the interval to the sensor is the new lower limit of the stopping distance
4. If the sensor is not triggered the middle becomes the stops before end of the interval and the lower limits of the stopping time and distance are unchanged.
5. Repeat from step 2 until you are satisfied with the precision of the measurement.

---

* You should convince yourself that the definition of stopping distance correctly removes the systematic error for this method of calculating the stopping time.

The only problem with this algorithm is that it doesn't always work: it assumes that the train behaviour is deterministic whereas the actual train behaviour is stochastic, so it works only if the precision you require is low enough that the stochastic effects are negligible.*

The problem encountered is endemic to experimentation using human subjects, who have substantial variability in their responses. Thus, the experimenter wishes to build up a statistical description of the subject's behaviour. It is often solved using one of a collection of staircase methods. They start with bisection of an interval to discover the neighbourhood in which the behaviour changes. You then choose a step size close to the desired precision. Then, if the sensor is triggered you move the location at which the stop command is given away from the sensor by the step size; otherwise you move it towards the sensor by the step size. This procedure ensures that the measurements made are confined to the values for which stochastic behaviour is observed.

7. Short Moves

Calibration of stopping distance, stopping time and velocity makes it possible to stop a train at any position on the track provided the train travels far enough to get up to constant velocity.

The calibrated train moves we have seen so far have been long enough that the train gets up to a constant velocity then comes to a complete stop. The shortest moves having this pattern, and with the train moving at an interesting velocity require more than a metre of movement. On a crowded track there may not be enough free track to move the train so far; for a move with a deadline there may not be enough time. This section describes the simplest method I know for moving the train a short calibrated distance. It assumes that all train moves start with the train at rest, and finish with the train at rest, which covers almost every move required by a basic project; it integrates well with the long move calibration techniques described so far in the document.

A more comprehensive method of calibrating the position of the train when it is accelerating or decelerating is described in section 8, below.

7.a. Calibrating distance travelled

The basic procedure for calibrating distance travelled is very simple.

1. Start with the train stopped on the track.
2. Give the train a speed n command, where n is a fast enough speed that the train moves without stalling wherever it is on the track.
3. Delay for t_s seconds.
4. Give the train a stop (speed zero) command, and wait until it stops.

* You can get an idea of the size of the stochastic effects by running the algorithm several times and observing the range of values at which the algorithm terminates. You could, I suppose, repeat the binary search enough times to get a statistical description of the variability of the result, but that is likely to try the patience of both your partner and your classmates.

5. Measure the distance travelled, x, using a tape measure.

Repeat a few times, and convince yourself that the variability you see is consistent with what you saw when you were calibrating stopping distance and velocity.

At this point you can travel one particular distance with an error you understand. To extend the method choose different times and create a look-up table giving distance travelled as a function of time. The table should be dense enough that you are willing to interpolate linearly to get distances between the ones you have measured. You might also want to invert the table to give time as a function of distance to be travelled.

The table should be extensive enough to overlap with the moves you control by getting the train up to a constant velocity.

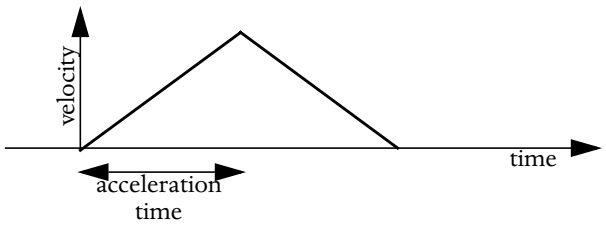
7.b. Calibrating the duration of a short move

The discussion of stopping has an extensive description of measuring the time taken to stop (section 6.b). The general idea is very similar.

1. Set up a short move using time t_s that will end with the pick-up just triggering a sensor.
2. Get the time, t_1, and start the short move.
3. When the sensor is triggered get the time, t_2.
4. If the move does not time out the difference, $t_2 - t_1$, is a lower limit on the duration of the move.

All the complication associated with measuring stopping time apply here, not to mention the possibility of devising an algorithm for automatising the solution.

Look for a relationship between the acceleration time t and the duration of the entire move. Some models of train kinematics (section 8) postulate a symmetry between acceleration and deceleration: deceleration is time-reversed acceleration. If so, a short move has a velocity function like the one drawn below. (The diagram shows constant acceleration so that I can draw it using only staight lines.) It is easy to see that constant



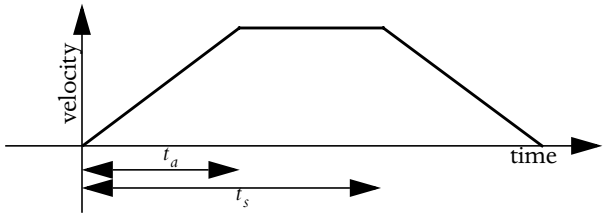
acceleration, which is necessary for a linear velocity function imposes the property $t_2 - t_1 = 2t_s$. Other assumptions impose other properties on short moves.

I am not advocating the symmetry described here, or the particular shape of the acceleration function, just using it as an example. (The few

details that follow are a preamble to SECTION 7.C.) While accelerating $(0 < t < t_a)$, the velocity is $v(t) = at$ and the distance travelled is $d(t) = \frac{1}{2}at_s^2$. While decelerating $(t_s < t < 2t_s)$, the velocity is $v(t) = a(2t_s - t)$, and the distance travelled is $d(t) = \frac{1}{2}at_s^2$. The total distance travelled during this move is $d(t) = at_s^2$. If life were this simple you could probably deduce the acceleration and deceleration functions from the short moves table.

7.C. INTEGRATING SHORT MOVES WITH LONG MOVES

A train accelerating from a standing start will eventually reach the constant velocity for its speed, as shown in the accompanying diagram. While accelerating $(0 < t < t_a)$ the train's velocity increases linearly, $v(t) = at$, and its position quadratically. At the end of acceleration its position is $x(t_a) = \frac{1}{2}at_a^2$. In the interval $t_a < t < t_s$ the train then maintains the constant velocity at_a and travels a distance $at_a(t_s - t_a)$. During deceleration $(t_s < t < t_s + t_a)$ the train's velocity decreases linearly $v(t) = a(t_s + t_a - t)$ and the train travels a further $\frac{1}{2}at_a^2$. Thus, the total distance travelled in this short move is $d(t_s) = at_at_s$.



We see that the total distance travelled in a short move is

$$d(t_s) = \begin{cases} at_s^2 & t_s < t_a \\ at_at_s & t_s > t_a \end{cases}.$$

The key observation is that when the short move time exceeds the acceleration time the distance travelled during the move becomes linear with the short move time. This result is independent of the acceleration characteristics of the train.

In practical terms, you may decide that your project will make all train moves start and end with the train stationary. If so, this analysis merges stopping with short moves and works for moves of any length.

8. Models of Acceleration

In this document I have at various times used hypothetical models of train kinematics to give concrete illustrations of one or another technique. 'Why,' you might ask, 'not measure a model of the train behaviour and fit it to a set of mathematical equations?'