Routing

File: src/train/controller.c, src/train/model.c

For routing, we have used Djikstra's algorithm to make the routing works. The routing will also consider other the train's location, by making sure it is able to avoid the path when a train is blocking a path. The run-time of our implementation should be O((|E|+|V|) log |V|). We also run through the algorithm every time we would like to move a train from a destination to another. The reason we chose this algorithm is for its good run-time for graph of positive edges, and ease of implementation.

Sensor Attribution

File: src/train/controller.c, src/train/model.c

The trains have to be initialized at a node position which have to be inputted manually. For every 10 ticks, we will send a sensor reading to the train set.. For every sensor being triggered, we will try to attribute to train based on the expected destination of every train. Our physics model handles the expected position of the trains. The train associated with a sensor, we update its position, and correct the model. If a sensor reading cannot be attributed to any of the trains, then we simply ignore it by assuming that it was an error, highly likely not triggered by a train. For trains which we do not expect to have sensor reading, we update an estimated position based on our model; therefore even if an expected sensor did not get triggered, our train tracking model can still recover from it.

Collision Avoidance

File: src/train/manager.c, src/train/driver.c

Same as TC2. To detect head to head, or head to tail collision, we simply check if a pair of train is about to get close to each other based on the estimated position. If we are getting closed, then we triggered speed 0 to stop the train to avoid collision. We judge a pair is closed if they are within stopping distance plus a delta error. For side-by-side collision, and switch coordination, we will reserve switch ahead. When the switch is reserved, then we can't switch it, since a train may depend on it, or we don't want a train to be on it while switching. When a train gets a reservation, then configure it based on the path of the train. If failed to reserve, then the train stop, waits for the reservation to be released. We have implemented a DFA to keep track a different state of the train.

Modelling/Error Handling

File: src/train/model.c

If the trains misses a sensor reading, it does not matter since our model keeps on running. If the sensor is not triggered long enough, then the train will timeout, and declared it as lost.

For the motion of the train, we have the following assumptions

1. Each speed level is mapped to an expected velocity, the velocity will stop accelerating once it reaches the expected velocity for current speed level.
   (Note: The following data is the measured average velocity in mm/s)

	10	11	12	13	14
01	327	390	455	515	595
24	335	397	470	530	620
58	303	368	438	500	600
74	470	535	580	630	630
78	265	320	370	430	495

2. During accelerating and stopping process, each train has a constant acceleration, which can be calculated based on (a) velocity and stopping distance, (b) accelerating process.
   a. For stop distance, we have

$$d_{stop} = v_{average} \cdot t = \frac{v}{2} \times \frac{v}{a}$$

   b. For accelerating process, we have either “accelerating didn’t finish in this section”

$$d_{section} = v_{average} \cdot t_{section} = \frac{1}{2}(v_{start} + v_{end}) \cdot \frac{v_{end} - v_{start}}{a}$$

   or “accelerating finished in current section”

$$d_{accelerating} = \frac{1}{2}(v_{start} + v_{expected}) \cdot \frac{v_{expected} - v_{start}}{a}$$

$$d_{constant} = v_{expected} \cdot (t_{section} - \frac{v_{expected} - v_{start}}{a})$$

$$d_{section} = d_{accelerating} + d_{constant}$$

Based on the above equations, and a lot of data measured from the actual trains, we can then estimate the location and the velocity of a train using integration:

$$d = \int_0^t v \, dt \approx d_0 + \frac{1}{2}(v_0 + v_{\delta t}) \cdot \delta t$$

$$v = \int_0^t a\, dt \approx v_0 + a \cdot \delta t$$

Notice that, although δt is supposed to be a small number, we cannot set it too small. Our program is using integers to store and calculate the numbers, so the errors may accumulate rapidly if we do too many calculations per second. After testing different δt, we think δt = 100ms is a reasonable number for our application.