TC2

How to build

Run make from the root directory

Servers

Several new servers were introduced compared to TC1.

cbServer

First there is a cbServer (callback server) that implements a publish/subscribe pattern for sensor notifications. The RegisterCb function is called by a calling task to subscribe to sensor notifications, and tasks can subsequently desubscribe. When a task is subscribed, it receives a Send when any sensor is triggered. Hence the **cbServer** is used by wf_sensor, by experiment drivers, and by the demo train server.

trainServer

The trainServer (in trainServer2.c) functions similarly to TC1, except now it tracks two trains. Sensor attribution is simple - if a train is at sensor s, then the sensor in front of s is attributed to that train. If a sensor is not attributed to exactly 1 train, trainServer ignores the sensor and issues a warning, because other parts of the system should have prevented it.

The trainServer also does maintains local reservations. A train reserves the last sensor it tripped, the next sensor, all the landmarks in front up to stop_distance, and from that point, all the landmarks until the next sensor. This is necessary because we only do processing on every sensor read. If the train fails to reserve a landmark, it immediately halts and issues a warning.

The reservation is maintained as follows: when a new sensor is attributed to a train, all reservations for that train are dropped, and the new reservations (as calculated above) are

Known bug: if a train fails to reserve a sensor and halts, it could halt very close to the failed-to-reserve sensor, which is then not reserved (but it really should be reserved, because the halting train might be near it). This could cause another train to stop too late and hit the sensor. However this might require 3 trains to trigger (the failed reservation requires 2 trains, and the crashing train might need to be separate).

Note that manual control of the train does not use the trainServer's reservation system.

In addition, manual tracking of reversing trains now works. When a train is reversed, the trainServer marks it as reversing, and in that state, for a sensor s, we attribute the sensor in front of s as well as the reverse of s, to that train. Once the reverse of s is attributed to the train, it is no longer reversing.

Known bug: if the train barely stops on a sensor and reverses without triggering the reverse of the sensor, we do not track it.

pathServer

The pathServer is called by haltat to find a path between two landmarks. It is a stateful server, and it stores its own reservations (distinct from the trainServer local reservation) for each train. When a train requests a path the pathServer reserves the entire path and excludes it from pathfinding. Haltat is responsible for asking the pathServer to drop reservations when it is done.

pathServer uses Dijkstra's algorithm directly on the track_data graph. Floyd-Warshall is not used except in the rt command for cli testing.

There is no support for time-based avoiding or paths that reverse.

Robustness

Unattributed sensors are ignored. This seems to work best in practice. I never got an inconsistently skipped sensor so I didn't support them. However, on track B there is a sensor that is consistently not tripped by the train. I mark these as node type BROKEN_SENSOR so that the attribution code skips over them but pathfinding is allowed to go through them.

Two of the central turnouts must be set in a certain way on track A or the train will derail. In addition, one of the turnouts on the right that goes from the inner to outer loop is stuck. I mark the ususeable edges as having infinite distance so dijkstra doesn't consider them to be valid edges.

UI

An ANSI scrolling region is used for the cli. This way the HUD can be updated very slowly. In addition, the dirty-buffer code I wrote for snake can be used to repaint the track GUI. However, currently the whole track is just redrawn.

Calibration

Stop distance was measured manually and confirmed to be within the 115 ms tolerance. This is done until the distance is less than the length of the train.

Speed was measured by using 20 samples of the inner loop (same loop as in TC1). We share the code for online calibration. A ringbuffer is used to store 20 inverse velocities which is averaged and displayed. Note: inverse velocity is used for online calibration, because the sample mean (of the set of velocity measurements) is a biased estimator for the true velocity.

The online calibration is commented out because I found that it had to be set to pretty low to not screw up the precision of the velocity estimate. However, I know it works because it is the same code used to gather offline calibration for trains 58, 78 and 1.