CS 452 TC2

Authors

Sebastian Gaweda (20143503) and Shashwat Pratap (20638036)

Make and Run

checkout - git clone https://git.uwaterloo.ca/sgaweda/cs452-rtos-aka-trains.git
make - cd cs452-rtos-aka-trains/tc2/ && make
run - in gtk terminal run - load ARM/{user-name}/tc2.elf; go

Infrastructure

As part of our previous refactor, different components of the system were divided into different servers. A GUI server exists to process updates to state which need to be communicated to the user. There are 4 servers in the train control logic chain: Sensor, Switch, Train, and Navigation. Each of these servers is responsible for state related to their domain and communicating with servers dependent on that state information.

GUI Server

The GUI server is responsible for accepting state updates from components in the system and displaying those updates to the appropriate locations in the terminal.

Sensor Server

The sensor server is responsible for maintaining a list of recently triggered sensors. It makes requests for sensor data and parses those results to create a list of unattributed sensors. That list is then sent to the train server, and an attributed list of sensors is received. This attributed sensor list is used to update the list of recent sensors. This updated list of recent sensors is then passed on to the gui server.

Switch Server

The switch server is responsible for maintaining the current state of switches on the track. It receives switch command requests and updates its state before passing those messages on. It sends the gui server updates when a switch change occurs.

Train Server

The train server is responsible for 3 tasks: attributing sensors, maintaining/broadcasting train locations, and issuing train commands.

- **Sensor Attribution**
  Unattributed sensors are attributed in the train server using predicted sensors provided by the navigation server. These attributions are used to correct errors in train position and are propagated to the sensor server and navigation server.
- **Locations**
  Location updates are based on our models of train acceleration, velocity, and deceleration. Every tick, the train server updates the position of moving trains based on our calibration data and the train's direction and reports the updated positions to the navigation and gui servers.
- **Issuing Commands**
  When a train command is received, the state of the train is updated and the command is queued for sending. Reverse commands have delays introduced based on the speed of the train.

Navigation Server

The navigation server is responsible for 4 tasks: route finding, route reservation, sensor prediction, and issuing commands to control the trains and track.

- **Route finding**
  Route finding is implemented with Dijkstra's and uses a modified version of the provided track graph which supports reversing at merges.
- **Route Reservation**
  When a route is discovered for a train, the full route is reserved and the reserved nodes are removed from route finding calculations. As the train exits reserved nodes, the reservation is released so that the node can be used again in route finding calculations.
- **Issuing Commands**
  When a route is discovered for a train, the navigation server produces a list of location based commands that are triggered as the train reaches certain locations along its path. This satisfies a timing requirement of the train system, to make certain commands occur at certain times.

TC1

Routing and Switching

The navigation server implements Dijkstra's for route finding. We chose to use Dijksta's to make it easier to compute paths when running multiple trains. Given a starting position and an end position, we can compute the shortest path to the destination and direct the train there using the train control software.

Stopping

We have calibrated acceleration, velocity, and deceleration for train 58 and train 24. In order to stop at the correct point in the track, the navigation server uses the train speed calibration information to determine at what position on the track a train should issue commands regulating speed and stopping the train.

TC2

Route Finding

When a path from one location to another is requested by a train, the nodes of the path are reserved until the train is determined to have passed the reserved node. Locations provided by the train server give an estimate for where the train is positioned to facilitate releasing reserved nodes. If a node is reserved it is not used during subsequent path generations (until its freed), and so it is impossible for multiple trains to have paths which intersect. The reservation system can also be used to permanently block out sections of track. The track data was modified to add support for reversing at merges. To ensure that reverses occur at appropriate moments, the train model was modified to more accurately represent the position of the front of the train. To properly time switch and reverse, we create locations on the track that when passed trigger actions, and use the train location data being reported by the train server to detect when those locations have been passed.

Sensor Attribution

Each train expects the next two sensors in its path. If a sensor is triggered which is not expected by a train it is ignored.

Collision Avoidance

Collisions are avoided by generating mutually exclusive paths. In an effort to mitigate the effects of reserving long paths, nodes are released as trains pass them.

Robustness

Our goal was to create a system which prioritizes collision avoidance and recovery after detecting a collision may occur. Reservation provides basic collision avoidance. Sensor attribution ensures that our system doesn't get unexpectedly lost while driving along the track.