CS 452 – Train Control 3

March 28, 2017

Zhengkun Chen

Student ID: 20557054

Xiwen Liu

Student ID: 20468618

Operating Instructions

Full pathname to executable: /u/cs452/tftp/ARM/z283chen/kernel.elf

To load and run the program in the Redboot terminal, run the following command:

> load -b 0x00218000 -h 10.15.167.5 "ARM/z283chen/kernel.elf" ; go

The executable can also be created by going into the directory "kernel" and calling make:

> cd kernel

> make kernel.elf

And then copy the elf to wherever you want to run it.

Data Structures and Algorithms

Calibration

Data structures:

- The track structure from track_data.c, from the course website, was used to store the track's information.
  - Additional fields index and switch direction were added to the track structure to better reflect the state of the track dynamically.
- A 4 dimensional array train_velocity[train number][speed][start sensor][end sensor] stores the number of ticks needed for a train to travel from one sensor to the next, at a set speed.
- A 2-d array default_speed[train number][speed] stores the default velocity of a train, if there is no data in train_velocity.
- A 3-d array train_acc[train number][speed][type] stores a specific train's acceleration at different types of sensors. For example, E14/15 and E5/6 are considered the same "type" of sensor, since they lie on the same track orientation.
- These arrays offer constant time access, are easy to manage, and have very little overhead.

Algorithms:

- As a baseline, the velocity of the train at a number of different sensors was calculated by recording the average time difference of when a sensor and the following sensor was activated. During execution, the times are dynamically adjusted as the trains move, using a weighted average of the new time and the old time.
  - Entering the command "cal [train number] [speed]" will print the new velocities, which can then be imported into our program.
  - All time is measured in ticks, where 1 tick is 0.01 seconds.
  - All distances are measured in millimetres.

Path & Route Finding

Data Structures:

- The track structure from track_data.c is used to represent the graph of the track. This structure is a linked list of nodes, where a node is a sensor, switch, or an end of the track.

Algorithms:

- A path is found with BFS, since the graph of the track is sparse. The path is the shortest path in terms of number of nodes. If switches need to be flipped in the path, the switches are flipped when the train is two sensors before the switch. If the train is not on the running on the path we found, the path finding algorithm will detect that and cancel the path finding to be safe.
- The route found will avoid any sections of track that have been already reserved by another train.

Stopping

Data Structures:

- A stop wait queue keeps track of the amount of delay time and the sensor number that the train needs to stop at.

Algorithms:

- A worker task sends the stop command to the train when it is required. This way, the worker task does not block the calling task.
- To determine when the stop command should be sent to the train, the stopping distance of the train was measured. Once a path has been determined, the stopping distance is subtracted from the end point to determine the distance of the train from the preceding sensor when the stop command needs to be issued. Then the distance is converted into a delay time based on the experimental initial velocity at the preceding sensor.

Train Position

Data Structures:

- Data about each train is kept in a structure, which contains its current speed, location, previous sensor activated, and its predicted path.
- A worker for each train was created to handle routing.

Algorithms:

- To find the train's predicted path, the latest sensor activated is used to traced the next three sensors in the path, using the track data graph.
- Path correction: If a set of "noisy" sensors actually forms a valid path (i.e. occurs in order), chances are the train is being repositioned to another location than we expected. We correct the path accordingly.
- When a new route is requested through a command or programmatically, the train worker is unblocked so that it can find and travel along a route. After the train reaches the destination, the worker task is blocked, until a new route is needed.

Short Moves

Data Structures:

- The model was developed by measuring the distance travelled vs the time interval between sending the start and stop command. Then a quadratic regression was done on the collection data to find a model for distance as a function of time.

Algorithms:

- To calculate the amount of time to wait between the start and stop commands, the root of the quadratic regression formula is found.
- A square root function fast_sqrt was implemented, which uses Newton's method to approximate the numerical value of the square root of a given number.

Error Detection

Data Structures:

- The train structure contains the previous sensor information needed to detect errors.

Algorithms:

- Malfunctioned sensors or invalid sensor input are ignored. Using a BFS, all sensors within 2 levels of the latest valid sensor are found. If a new sensor input is not one of the valid sensors for the train, it is ignored.
- If a sensor is valid, missed switches are detected by checking if a branch exists between the sensor and the previous sensor. A path from the previous sensor that has the same time as the predicted time is found, and if this path does not match the actual sensor, then a switch must be in the wrong position. Then the program is updated to match the track data.
- Missed sensors are detected by finding the path from the previous sensor to the current sensor, and checking if the path contains any other sensor.

Collision Avoidance

Data Structures

- The track was divided into sections that can be reserved by a train. The sections are defined as an array and were defined to be from sensor to sensor. If a section contains a switch, both branches of the switch are defined to be in the section. The reservation sections were generously defined to reduce the change of collision.
- A track server controls the sections of the track and whether a reservation can be made.

Algorithms

- When a train is not moving, it only has the current section it is in reserved.
- When a train is moving, it has the next three sections ahead of it in its path reserved. Three sections were chosen because this is enough distance for the train to come to a stop at the fastest calibration speed.
- If two trains are heading towards each other, they will come to a stop. To avoid a deadlock, their current path is canceled, and a new destination is generated for each train.
- For short moves, the entire path of the move is reserved, since the total distance is expected to be less than a few sections.

Sensor Attribution

Data Structures

- Each train structure contains the current and previous sensor activated, as well as a set of predicted sensors.

Algorithms

- Once new sensor data is received, the sensor data is sent to each train. If the new sensor data matches the train's set of predicted sensors, it is attributed to the train. Otherwise it is ignored by the train.

Commands:

- staf [train number] [sensor name]
  - "**St**op **af**ter" sends the stop command when the train triggers the chosen sensor.
- move [train number] [distance]
  - This moves the train by a short distance, under 40 cm. The distance is in millimeters. For example, "move 64 280" will move train 64 by 28 centimeters.
- mot [train number] [time ticks]
  - "**Mo**ve **t**ime" sets the speed to 4, and after the specified number of ticks, where 1 tick is 0.01s, the stop command is sent.
- goto [train] [sensor]
  - Sends the specified train to the sensor. This command if called when the train is not in motion. The train will come to a stop at the specified sensor.
- block [section]
  - Programmatically blocks a section of the track so that no train travel over it.
- unblock [section]
  - Programmatically unblocks a section of the track.
- en
  - Enables routing that avoids blocked parts of the track.
- dis
  - Disables routing that avoids blocked parts of the track.
- pos [train] [sensor]
  - Sets the position of the train to a sensor.
- demo
  - Runs the program created for Train Control 3.
- zmove [train] [sensor 1] [sensor 2]
  - "Zmove" finds the shortest path from sensor 1 to sensor 2, including reversals. The command will move the train from sensor 1 to sensor 2 using a series of short moves.

Things to note

- Train 64 was calibrated on track A, on speeds 6, 8, 10 and 12.
- Train 63 was calibrated on track A for speed 8 and 12.
- The program will work best with the above two trains on the track.

Source Code

Git repository: https://git.uwaterloo.ca/x272liu/cs452/tree/zmove_tc3

Sha1 of the commit: c32cd8910cbca9040dbc28064224d917ace2547e

All files in the repository are part of this submission.