# 1. A description of how to access, make and operate your program.

# DOWNLOADING THE SOURCE CODE

Download the zip file

goo.gl/u1c8ov

click File->Download

this will download the m1\_submission.zip file

Unzip the zipfile will result in a m1\_submission folder

REBUILD

compile the program in m1\_submission folder

make clean && make

copy the compiled elf file onto tftp

cp kernel.elf /u/cs452/tftp/ARM/ur\_user\_name/

change the permission of the copied file

chmod 744 /u/cs452/tftp/ARM/ur\_user\_name/kernel.elf

load the program onto the board

load -b 0x00218000 -h 10.15.167.4 "ARM/ur\_user\_name/kernel.elf

type go

go

## LOADING AND RUNNING THE PRE-COMPILED VERSION

load -b 0x00218000 -h 10.15.167.4 "ARM/z23he/m1.elf;go

# 2. A description of the algorithms and data structures used to calibrate the train and the track, to maintain knowledge of the train position, and to recover from errors. Remember to tell us how they work and why you chose them.

## TRAIN SPEED CALIBRATION - CONSTANT

The real speed for a train's constant speed is calculated by mapping a speed with each and every sensor. A path was designed for the train to run, the path covers every sensor on the track that is not an entry or exit, time data is recorded across each sensor and a table was created to calculate the velocity at each sensor.

For each sensor a speed, an array initialization function was written that takes in an array and fills it in with the speed.

## TRAIN SPEED CALIBRATION - SHORT MOVE

The short move is calculated by first timing stops. The train starts stationary, a train speed is then sent to the train, after a timed delay, a speed 0 is sent to the train. From this we can adjust the timed delay to find out how long the trains would run given the delay. A series of time from 500ms to 4000ms was sent, and the distance was measure for each. From these data we can plot the distance vs delay time needed. A best-fit cubic function was generated from these points. The program uses this function to time the go and stop commands to the trains for short moves across a specific distance.

## TRACK SERVER

The track server is the task that knows the condition of the tracks, and the task that performs path finding. More specifically, the track server knows about all the switch statuses. The track task is also task which performs the switch setting. Any commands given to switch the switch status is performed by this task.

The switch status is stored in a single unsigned integer. Switch 1 to 18 is represented by bit 1 to 18, where switch 0x99 to 0x9C is stored on bit 19 to 22 with 0 representing curved and 1 representing straight.

## PATHFIMD - MILESTONE 1

The path find algorithm uses the path track data on the course webpage, and performs BFS on it. The path find algorithm once finds a path will path the switches to the correct direction. This is a simple path finding algorithm and will not be feasible for milestone 2.

The path find algorithm also takes in a train's stopping distance. Based on the path found, it finds the last sensor it will trigger before the stopping distance and how far will it travel after that sensor to before a stop command needs to be issued. This information will be returned to the caller to time the stop command

## Sensors Server

Sensors requests are done using a server courier and notifier. The notifier will query for all the sensors and send the result back to the server when all the queries come back. The servers will than parse the information to handle requests accordingly.

The Sensor server can handle the following requests:

### Sensors dump request:

Use this to get the latest raw bytes that were returned from COM1. This is typically called by the sensor display task to get the sensors that were last triggered.

### Wait request:

This is a blocking request which is used to block the client until the specified sensors are triggered. The client can specify more than one sensor to wait on.

## Train position and stopping

The train position and speed is controlled by the command center tasks which consist of a server that generates a notifier for each train as well as stop request

The command center knows where the trains are at all time in term of the sensor it last triggered and when it was triggered. It also has the velocity of the train that was measured. these information is sufficient for it to calculate an estimated location of the train.

The server handles 3 types of client requests:

### Init train request:

Use this to set up the trains’ initial location as well as initialize its parameters. It creates a notifier and courier for the train which notifies the server when the train triggers the next sensor.

### Set train destination:

Use this set up a destination for the train. The server will ask the track server for when to stop the train. The server will convert this information in terms of the sensor and time after sensor trigger. The server will than create a stopping notifier that will fire when the train needs to stop.

### Get train info:

This is a request for the information of a train. It will return the trains’ previous and next sensor.

# 3. Files and MD5s

e63ad8bf4a7a1bf48c274eb4ffe81ad2 ./include/ts7200.h

5f801609c2ee3af70a7bc962f4a74494 ./kernel/ker\_ent\_exit.asm

e2954b851b40c8c4e3b76c28f39b5071 ./kernel/kernel.c

63c88d650a2a57dc902976fab873073c ./kernel/kernel.h

605a1438ac7738c0c486430e52b165e5 ./kernel/queue.h

0c954df6f2664adecbfe0a6cd994e4b6 ./kernel/debug.c

57d7ecc932e069e6f3c8f5044ea66a6f ./kernel/queue.c

fe178ee05cba742a7b79153dbb1e4edf ./kernel/nameserver.c

b9b7470560031946dab7f959b1bace6d ./kernel/int\_ker\_ent\_exit.asm

4c7903df0ada55219827ec269eba5185 ./kernel/nameserver.h

480ac30b62493a9d2346b956dacdf350 ./kernel/debug.h

705d2e36b1b9281289480d95c647d372 ./lib/libbwio.a

1bb0bc970ef579c94d89f187060a1406 ./lib/bwio.c

3fb593b9f5f7c393fd4e34fbf8b8d940 ./lib/bwio.h

099eb61a5ee6c949b237b29fc44b05f6 ./lib/libutil.a

a22badbdb79426bdbe6feca347286b4b ./lib/util.c

b6eee62abf5a4d2a81d09cf5d1061bcc ./lib/util.h

b6c5125ddef0fd675ce94cdec50f05be ./lib/a.out

d06ccb8138dabc4f74db4b7a00a1de6e ./orex.ld

3bc27c207ef23ce6b28f87dca9a6d628 ./track/trainspeed.c

f5458f5fca06a8c9502f34a1d9fbe741 ./track/track\_data.c

57648ac7519921743c912d61da28bcf2 ./track/track\_node.h

15e0112dc091029d65a447facaf8eaec ./track/track\_data.h

017fcbf35e1a1ff8261bcbb54248fbe6 ./track/trainspeed.h

a00bf7f767939dd2245f777865bda780 ./tasks/Tasks.c

acfd9a58c040faf90fbc782c937f4883 ./tasks/Tasks.h

556c534a45ec488baaf5c336608d4fd0 ./tasks/clockserver.c

f0da0dd5b3ebffb8597c305dead61152 ./tasks/clockserver.h

1dda09b8a58f5b2284d5bf58a85d3b8a ./tasks/rps.c

4ada5de8073ec8c6fab66719f12c8f66 ./tasks/rps.h

16b8cc4b0c5a9595daae77e6c77fcac9 ./tasks/comservers.h

b9995e378d01540d3cb80bf4466930b3 ./tasks/comservers.c

0d85ccbaca0fcb48d56f5096a880732e ./tasks/interface.c

295932c5338e6f732183ebcdbcf94988 ./tasks/interface.h

1087cce0b8d3c00f7700d8de4085ea72 ./tasks/sensors.c

92d5ff702ef2cb59c13beeea5a0143ca ./tasks/train.c

3626d473b41ccde18a2ff0ec9019cde9 ./tasks/train.h

465b3299aa349f2c3251cfea0c4eefcd ./tasks/sensors.h

8e968b8a65e44cf1bb447241337986ee ./tasks/commandcenter.c

832bdf46c9c1cec90574de35572bc089 ./tasks/commandcenter.h

5396b32d5d9ae2e7d20ae3749b5bb1a3 makefile