CS 452 PF

Names: Robert Elder, Christopher Foo
ID #: 20335246, 20309244
Userids: relder, chfoo

Date due: July 31, 2013

Running

The executable is located at `/u/cs452/tftp/ARM/relder-chfoo/pf-submit/final-demo.elf`.

The entry point is located at `0x00045000` or `%{FREEMEMLO}` It *must* be executed with caching enabled. (Caches not enabled by the program itself due to time constraints):

```
load -b %{FREEMEMLO} -h 10.15.167.4 ARM/relder-chfoo/pf-submit/final-demo.elf
go -c
```

There are also other files:

`kern_simulation.elf` A train simulation that can support up to an arbitrary number of 10 trains.

`kern_test.elf` Runs 390 rock paper scissor tasks. (That's a lot of tasks!)

`orientation.elf` Program used to collect sensor data for train orientation.

Commands

tr TRAIN SPEED Set the train speed.

rv TRAIN Slows, stops, and reverses train. The final speed is hard coded to 5.

sw SWITCH DIRECTION Changes the turnout direction. DIRECTION is either S or C.

q Quits the program.

map NAME Sets the current track. NAME should be A or B.

go TRAIN Begins the train route finding process. The train should start up, find position, and go to a random destination.

gf TRAIN Like `go`, however, this make the train go forever by running `go` in an continuous loop.

num TRAINS Set the number of trains to be used.

paint Causes the interface to redraw itself.

rt Resets the train system by stopping the trains, clearing reservations, and clearing train engine states.

rps Runs Rock Paper Scissors program.

Pressing 'CTRL+Z' will cause the program to dump out a list of tasks information and statistics. This is considered a debug operation, and as such it can cause future instability in the program.

Pressing CTRL+C will cause the program to exit immediately without shutting down the tasks.

Getting Started Quickly

To start up two trains

1. Select the appropriate map using the `map` command.
2. Set the number of trains to be used using `num` command.
3. Enter the first train using `go TRAINNUM1 0`
4. Enter the second train using `go TRAINNUM2 1`
5. If things go wrong, use the `rt` command

Description

Assert

The assert statement, as usual, is enhanced to show Thomas The Tank Engine. Please do not be alarmed when you see it.

When an assertion failure occurs, the Stop command will be sent to avoid train collisions.

Train Navigation

File: `route.c`, `tracks/track_data.c`, `train_logic.c`, `train_data_structures.h`

Train navigation is currently accomplished using a combination of fast and naive graph search algorithms, as well as a server called the SwitchMaster that is responsible for updating the positions of switches.

We have broken down the problem of navigation to anywhere on the map into two basic problems: The first is navigation to a point while considering the map as a directed graph. In this situation we only consider moving in the forward direction. In this context, it is now possible to navigate to anywhere on the map from all nodes because the graph is considered to be a directed one. In the second case, we consider the map as an undirected graph, where any shortest path can be found by finding the shortest route in the undirected graph. We can then express the problem of navigation between two points in the undirected graph as multiple navigations in a directed graph, while adding direction reversals in the middle.

To find a destination, a simple depth first recursive algorithm is used to build up a Route Info array. The Route Info array contains information about each track node and the switches it needs to switch. The algorithm avoids blacklisted switches.

Undirected Graph Model

In order to accurately model the train and its motion around the track, as well as predicted future positions on the track, we required another representation of the track to complemented the directed model that was provided. It is for this reason that we have created a undirected graph model of the track based on the directed graph model. This model also includes the trains as nodes, which enables us to apply standard graph-based algorithms to any nodes on the track graph, including the trains themselves. This has significant advantages for tasks such as sensor attribution, collision detection, and route planning. The advantage of including the trains as nodes in this model means that in this representation, we do not need sensor data to make decisions about what actions to take, and can rely on the current state of the model that has been predicted based on last sensor observations. The undirected graph model allows us to consider route planning, independent of the number of reversals that are required on the route. The other advantage is that trains are included as

nodes so that the shortest distance between two trains can be calculated down to the micrometer at any point in time, as long as their approximate speed is known.

Sensor triggering can be used to infer observed train speeds, which can be used to simulate the motion of the train in a near continuous time manner.

Undirected Graph Data Structure

The undirected graph model is built from the directed track node data. Pointers are added to the directed nodes that point to the corresponding undirected graph nodes, and vice versa. The undirected graph model is implemented as an adjacency list. Since every node in this graph can have a maximum of 3 adjacent nodes, this significantly shortens the run time and memory requirements of many graph processing algorithms.

Dijkstra's

Dijkstra's algorithm has been implemented for the undirected graph nodes. The implementation of this algorithm is the standard one, with a run-time of $O(|E|+|V|)$. Testing has been done with a simulated track where multiple trains are sent on a random-walk around the track millions of times, calculating the shortest distance at each step. Valgrind was also used to preclude the possibility of programming errors.

Routing and Navigation

Currently, we use a simple recursive graph search algorithm for calculating paths. This will soon be replaced by the much more accurate Dijkstra's algorithm once the undirected graph model is incorporated into the routing. Once we have determined a series of nodes that we need to navigate through, we determine the set of switches that need to be changed from their current state, up until we possibly end up changing that same switch again (for re-entrant paths that only involve moving forward). The switches are queued in the order in which they need to be switched so that the closest switch will be the first one to change. If the train triggers a sensor that is not on the path it was expected to take, a warning is printed for debugging purposes.

Model

The model of the train uses an estimated speed of the train computed using simple linear interpolation of the ideal speed of 45cm/s. Ideally, it should be using a finely calibrated tables, however, this could not be implemented within the time provided but the currently code is ready to support this.

Sensor readings augment the train model. It currently snaps the train to the location of the sensor. Ideally, we should use a rubber band method that gently interpolates the differences and jitter.

A simulation build has been provided that behaves similarly to the actual build. It randomly uses a sensor as the initial position.

Stopping

For stopping we use a roughly approximated table for each train that will tell us how many millimeters before a sensor we need to issue a command to slow down. This table was derived from empirical measurements and still needs a bit of calibration. This is especially true on a specific train level, since different trains require different stopping distances.

A list of speeds for each node during stopping has also been determined empirically. Nodes that are near switches have a lower speed to avoid stopping on top of a switch. We risk the trains getting stuck on curves because it is preferred that trains become stuck rather than derailed by an activating switch.

Velocity

Our trains move at a speed of 45 cm/s and we maintain this speed using a feedback control mechanism. The observed train speed is calculated by dividing the known track length between two sensors, and dividing this by the observed time taken to travel between them. The trains use a floating point speed setting to avoid sending too many train speed

commands and to dampen noise. The floating point speed setting is casted to an int and the command is issued if needed. The algorithm slowly increases the train speed when it arrives at a sensor too slowly, and decreases the speed quickly when it arrives too fast.

Sensor Malfunctions

Sensor malfunctions are accounted for by maintaining a list of sensors that are known to malfunction on each track. We use a blacklist of sensors to remember which sensors should not be navigated to, and which should be ignored when determining the train position.

Reservations

The provided track nodes have been modified with an extra field called `reserved`. It holds the train number of the reservation. Once the destination and route is calculated, all the nodes in the route are reserved. Once the train reaches its destination, the nodes are released from reservation. The concept of switch reservations is taken care of, because while a train has reserved a switch, no other can attempt to queue a switch change.

A train will always check the node ahead to see if the node is reserved. If the node is reserved, it will stop and wait in the `WAIT_FOR_RESERVATION` state. During this state, it will generate a random number between 1 and 100. If the number is 1, it will reverse direction and attempt to find a new destination.

On-the-fly switching

Although the current reservation prohibits multiple trains using switches by reserving it for the duration of the route, on-the-fly switching is implemented in case we were able to reduce the reservations needed. Each train looks ahead at the next 2 switches and computes the distance to them. If the switches are less than 50cm or the train will pass by them within 2 seconds, it will switch them to the correct route. We use this generous amount of time because the train model is not entirely accurate and switching early will reduce trains caught on the switches.

Lost train detection

Lost trains are detected by the model if the train has been found to be not on its current path. It is able to detect this by running the train through the internal model and state of the switches. For example, if the switch queue is backlogged and the command to active the switch has not been sent yet, the model will move the train based on the current state of the switches. However, the model may not reflect reality, so the model will keep a counter of how many times it has determined that it is on the wrong part of the track.

If a sensor reading has confirmed that it went to the wrong branch, the train will be put into the lost state and it can be concluded that the model was correct. If a sensor reading has snapped the train onto the correct route, then the switch has actually been put into the correct state and it can be concluded that the model has computed an incorrect train speed.

Ideally, it would be best to use timeouts and determine windowed distances to the sensors beyond each branch. However, only a portion of this feature was implemented but it was disabled. If given more time to fine tune the calibration, this feature would be useful for detecting incorrect train behaviour and allow use to recover from a lost train state.

When a train is lost in wrong location state. It will do nothing to be safe, however, it is possible for the train to recover by computing a new route.

Train Switch Master

The Switch Master is responsible for picking up switch commands from the Train Server and calling Train Command Server. This task is a worker that removes the burden of waiting for train commands to complete.

Train Engine Client

The Engine Client is responsible for picking up train speed commands from the Train Server and calling the Train Command Server. Like the Switch Master, the task is a worker hired by the Train Server.

Train Engine States

Name	Description
IDLE	The engine is stopped and waiting.
FINDING_POSITION	The engine is moving slowly and waiting for a sensor
RESYNC_POSITION	The engine has drifted from its calculated position and is attempting to find its location
FOUND_STARTING_POSITION	The engine has found its location
WAIT_FOR_DESTINATION	The engine is waiting for a destination to be calculated
GOT_DESTINATION	The engine has calculated its destination
WAIT_FOR_ALL_READY	The engine is waiting for other engines to be found and ready
RUNNING	The engine is running at high speeds to the destination
AT_DESTINATION	The engine is at the destination and stopped.
NEAR_DESTINATION	The engine has slowed down and is waiting for a sensor report.
REVERSE_AND_TRY_AGAIN	The engine is in a direction that provides no destination and is reversing to find a new sensor.
WAIT_FOR_RESERVATION	The engine has stopped and is waiting for the track to become unreserved
WRONG_LOCATION	The engine has entered an unauthorized section of the track

GO

The go command operates as following:

1. Set the train speed to 5.
2. If a sensor is hit, the location of the train has been found.
3. Reserve the current location in the reservation system.
4. If there are other trains that need to find their location, wait for them.
5. Pick a random destination.
6. Calculate a route to the destination.
7. If there is no possible route to destination, reverse the direction and go to step 5.
8. Activate the switches that do not overlap other routes or require switching multiple times.
9. Speed up the train to 14.
10. Read sensors and compute the speed, location, and distance to update the state of the train engine.
11. Using the sensor data and feedback control system, adjust the speed to achieve a speed of 45 cm/s.
12. If the next node is a switch that needs to be activated, switch it.
13. If the distance to destination is within the stopping distance, slow the train down.

14. If the next node is reserved, wait until it is cleared. If it is not cleared and a random number generator generates a true condition, reverse direction and go to step 5.

15. Wait for a sensor and stop.

For an iterative version of the go command, see GF command which will iteratively use the go command after a train reaches its destination.

GF

The gf command operates as following:

1. Do steps 1-15 of the go command

2. Wait for 4 seconds

3. Goto step 5

Train Scoring

In this deliverable, trains are given points and they must optimize their high score.

The trains are given or removed points depending on what kind of events occur.

Points Awarded	Event
10	Train found its initial position
-10	Train decided to not go anywhere
100	Train found arrived at its destination
-100	Train went off course
-5	Train slowed and sped back up unnecessarily
5	Train found another path when the original was blocked
-5	Train computed a speed that was physically impossible
-5	Train computed a speed that was negative
5	Train computed a speed that was reasonable

UI Servers

Files used by UI servers: `ui.c`, `ansi.c`, `maps/map_gen.py`, `maps/map_a.txt`, `maps/map_b.txt`

UI Server

- `atoi()` no longer throws an assertion failure on bad input

The UI Server is responsible for drawing the textual user interface. It draws a header, the time since start up, a system load indicator expressed in percentage, the command prompt, table of sensors readings, an ASCII diagram of the track layout, a table of train status, and a scrolled area of train information.

The command prompt supports up to 80 characters. Once this limit is reached, no input will be accepted and displayed. It supports backspace. Pressing the Enter key will execute the command and a response will be displayed under the command prompt. If an error occurs, it will be shown in yellow.

When a sensor is triggered, the UI Server will display an bold number on the table. Sensor data for the UI is cached by the Train Server so displayed sensor readings may not reflect actual state. Sensor states in the Train Server, however, reflect actual states.

The ASCII map shows sensors as X and bold X with underline. Switches are shown as U, C, or S which represent Unknown, Curved, or Straight. The ASCII map code was generated through a script from a text file.

A green highlight shows the destination of the first train. A yellow highlight shows the destination for other trains.

A black highlight shows the reservation of the first train. A red highlight shows the reservation for other trains.

The model's location of the train is indicated by a bold digit. If the model train and the actual train is on a sensor it will show a bold digit with underline.

Some of the hilights of the UI are found in figure 4.

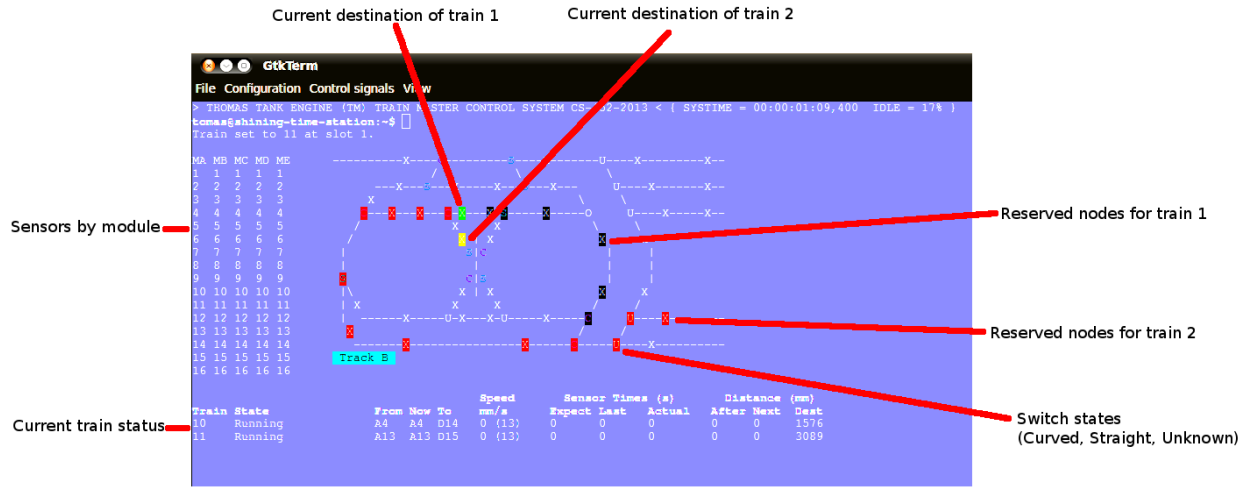


Figure 1: Figure 4

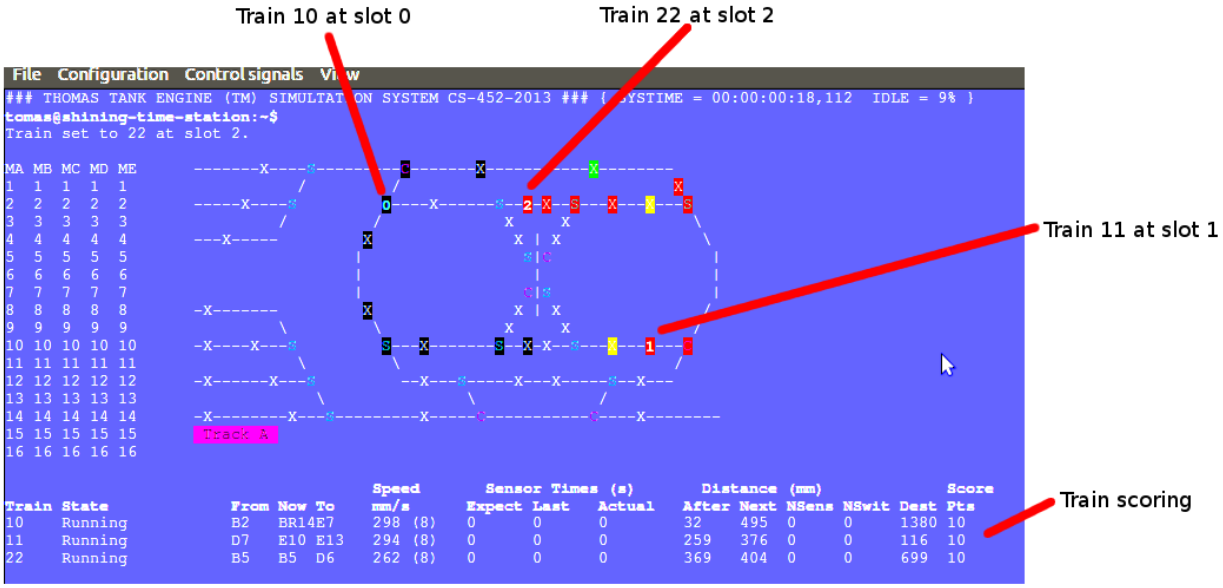


Figure 2: Figure 5

UI Timer

The UI Timer is responsible for sending a message to the UI Server. The timer tells the UI to update the clock and system load on the screen.

UI Keyboard Input Task

The UI Keyboard Input task is responsible for calling `Getc` and sending the character to the UI Server.

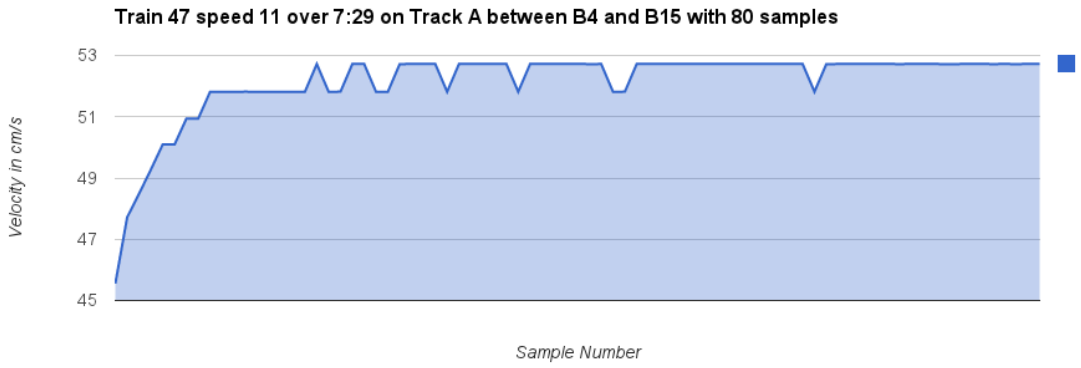
UI Print Message Task

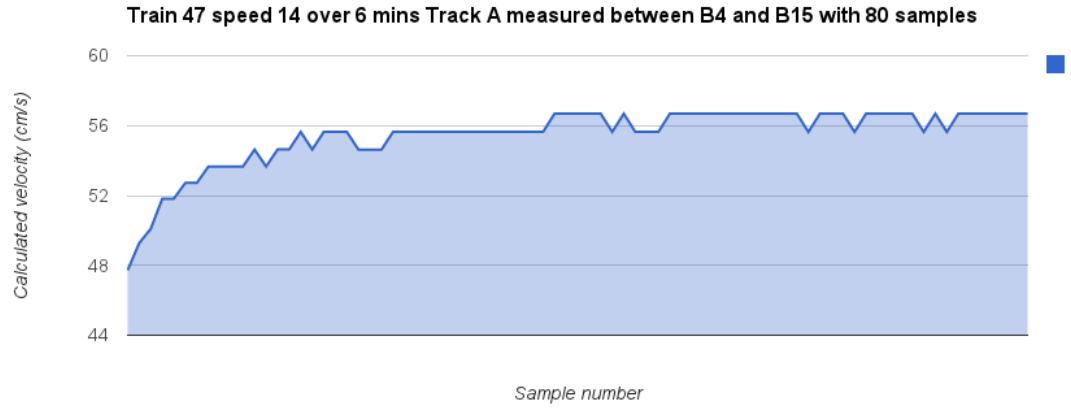
This task is responsible for printing messages into the scrolled area. It uses the ANSI feature to set scrolling areas. It is separate from the UI Server as messages may be from higher priority tasks like the Train Server. It is called via the `PrintMessage` call. This method was implemented as a non busy-waiting alternative for debug messages.

Real Time Worst Case Execution Analysis

Since this deliverable significantly depends on the accuracy of the simulated train model, an analysis was done into the sampling error, and how worst-case execution can affect the error in determining where train actually are.

It was observed that trains speed up over a very long period of time. After initial acceleration, trains can continue gaining velocity of up to another 10 cm/s. This takes place over a period that stablizes after about 1-3 minutes, however in some cases, speedup of less than 1 cm/s is still observed 7 minutes after the train has started. The graphs below show a plot of how the train velocity increases over time for two different speeds.


Train 47 speed 11 over 7:29 on Track A between B4 and B15 with 80 samples
53
51
49
47
45
Velocity in cm/s
Sample Number

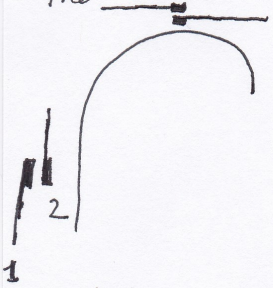

Train 47 speed 14 over 6 mins Track A measured between B4 and B15 with 80 samples
60
56
52
48
44
Calculated velocity (cm/s)
Sample number

Automated Orientation Detection

An attempt was made to determine train orientation automatically in order to improve the accuracy of predictions made by the train model. This improvement would come from knowing which side of the train the sensor is on.

Hypothesis

It was reasoned that curved sections of track were likely to have more friction, causing the train to move more slowly. To detect orientation, the idea was that the orientation difference would cause certain wheels to move over more or less tracks depending on where the sensor was.



In the diagram 2 will move over more curved track

Method

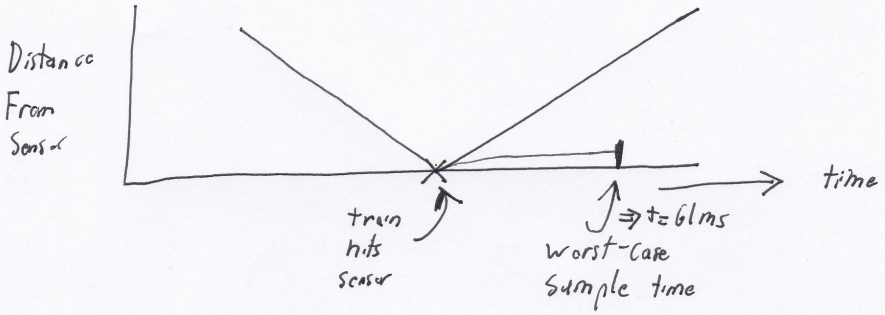
On track A, the time was measured for each lap of 80 laps between Sensor B4 and B15. This was done in a polling loop for high accuracy. Speed 14 was used and testing was done with train 45 and 47.

Conclusion

It looks like there is no stable difference more than 10ms required to travel between these sensors. The test is typically confused by transient effects like the several minute train accelleration period

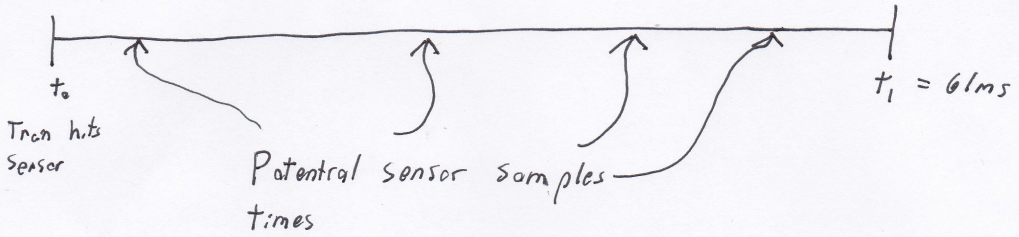
Worst-Case Error in Sensor Sampling

linear motion over a sensor:



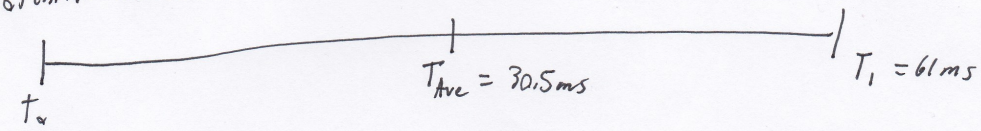
Batch sensor query method has a max frequency of once per 61ms

At train speed = 50 cm/s, 61ms = 3.05 cm



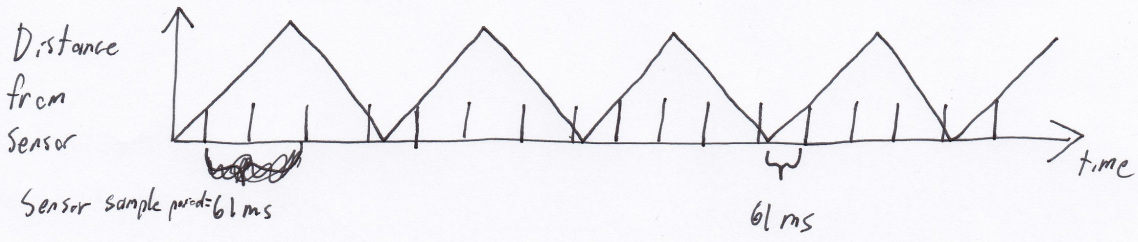
Problem: Determine actual sample time with greater certainty

Potential solution: Run trains a lot, and take average.



Worst-Case sampling situation can still exist!!

Consider train moving in a circle

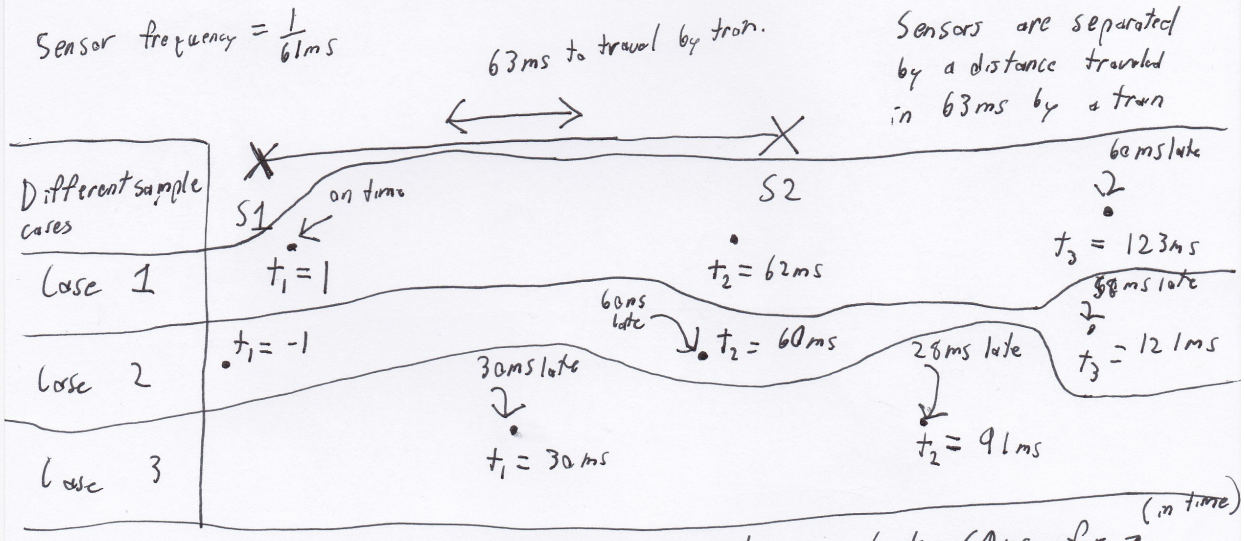


Train movement period = $n \cdot 61$ ms

$\Rightarrow$ If sensor polling period evenly divides train movement period, we could end up averaging over worst case (or best case). So uncertainty at 50 cm/s is still 3.05 cm!

$\Rightarrow$ Periodic motion is not necessary to encounter this problem. people usually calculate velocity on line by calculating Δt between two adjacent sensors.

Sensor frequency = $\frac{1}{61 ms}$



$\Rightarrow$ Total worst case error is always about 60ms for any sensor polling offset when distance between sensors is almost evenly divisible by sensor polling period.

But noise in system will save us if we sample enough?

Kinda...

- Trains can take up to 3 mins to reach steady-state velocity going from 47.5 cm/s to 56 cm/s.
- If trains are "hot" they go at max speed almost immediately.
- Temperature doesn't actually seem to change this, tried cooling down a train with ice for ~25 mins, no effect.
- Once a train reaches steady-state velocity (after running for 10 minutes) measured time between B4 and B15 (7 cm apart) on track A is about 1.22691S seconds with a standard deviation of 0.009811 seconds. Sample size was 80 with train 45 on speed 14.
- Batch sensor query period on track A was measured to be 6.1829 $\times 10^{-2}$ seconds with a standard deviation of 7.088 $\times 10^{-6}$ seconds (100 samples done reading ES)

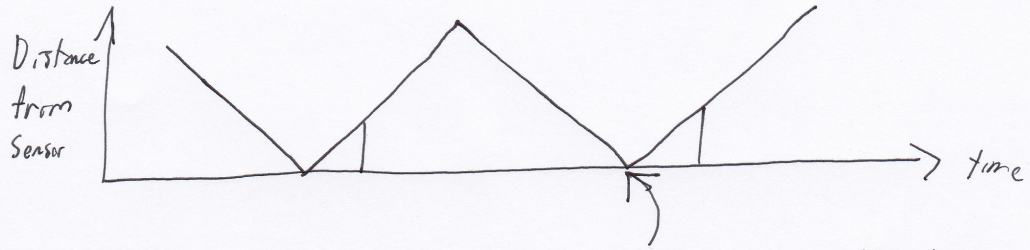
Conclusion

After steady-state velocity is reached, the majority of noise will come from motion of the train, not the sensors. $\sigma_{train} \approx 0.010$

$\sigma_{sensor} \approx 0.000007$

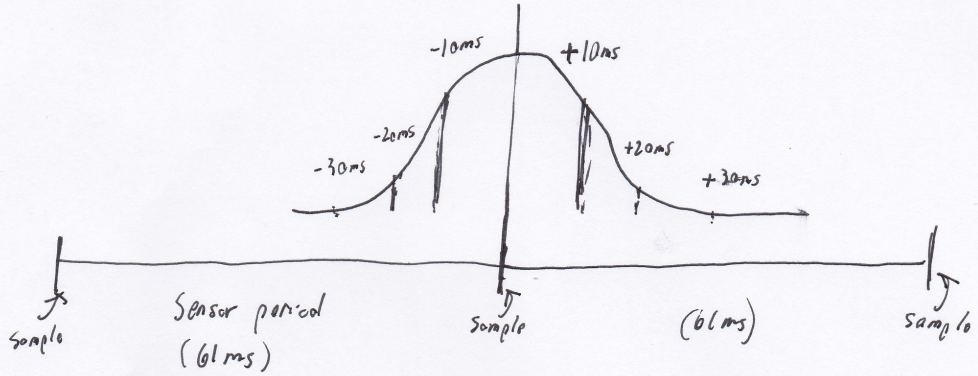
Effect of noise on period with periodic motion

Based on ~~trains~~ trains traveling from B4 to B15 w/ train 45



Train arrives slightly later or earlier

(Assuming gaussian distribution)



Question

When does noise cause sampling to cover all points of the potential sampling domain?

Answer:

I don't know exactly, but after N steps of a random walk that follows a normal distribution, the resulting distribution is $N(0, n\sigma^2)$ where N is a gaussian distribution centered at 0, with ~~σ~~ $\sigma \approx 1$. Therefore, in a 100% confidence interval of 6 we will travel 61 ms away in a minimum of 6 samples, but the ~~cumulative~~ distribution function is still not evenly distributed. This solves the worst-case situation with periodic motion but not with Δt measurements between close sensors, because there will be less variance, and require more samples

Source Code

The source code is located at `/u4/chfoo/cs452/group/pf-submit/io/project-final/`. It can be compiled by running `make`.

Source code MD5 hashes:

```
chfoo@nettop37:~/cs452/group/pf-submit/io/project-
final$ md5sum */*/* */* *
bd0a0df5b9fbc588bdc203efe3c6570d  tracks/tests/Makefile
bf8d2b5291390f9f727f7fa766bf9348  tracks/tests/tests.c
50ef0e1e3c71ab1e795fc3d39f75ef9d  include/bwio.h
9af226f127c1fd759530cd45236c37b8  include/ts7200.h
94944e9febc4db1bb344fff990ed7e9e  maps/map.h
3dfa3ed141445a72c20840b384c1ebb9  maps/map_a.c
c6adb76c95a6ae7986d03cd416d5837e  maps/map_a.h
703f1eeadf245074517591baa0844a37  maps/map_a.txt
7834e70e1b89501a33508f8ad5f0624a  maps/map_b.c
eba8710b29615da70e7165571efd99d8  maps/map_b.h
bd274a1bb75033031b39937bbf837f85  maps/map_b.txt
ead84e8315fd7e45f0e8e631197b9150  maps/map_gen.py
ec02c471a6b2bf494ff65b7a6e740a97  tracks/parse_track
md5sum: tracks/tests: Is a directory
1a1aac0b745639b84fe74f1839547512  tracks/track_data.c
1352f3743944badbb8c2399e6fb2ccd4  tracks/track_data.h
e33dcce364a34b75f722eb3d272626cb  tracks/track_node.h
9b68c8cea9bd88aad8571cae7f35a03f  tracks/tracka
f01263617106358e3fea00e84f9b324a  tracks/trackb
7b05a2c3e87fd049cebd40cf67fd0ef1  tracks/undirected_nodes.c
fc5b08361b98de3499737c9681addd33  tracks/undirected_nodes.h
612547069490bb70f5f6c9e1a565045b  45-run
a4e8acaf80c4680a85fbf8729afcb900  80-times
c7be4a987b71b07d49d973a6a61545a7  Makefile
1a5d522885e2e71cd9b940bd52ff9b42  Screenshot-1.png
e613d497f4ddd240605c62968fcc8b98  Screenshot-2.png
e92b7c25883384cd034329580bdb0e5d  Screenshot.png
41010359d05f8a46d7d402a38b0b9693  Screenshot3.png
0dc64506433fa8e40520a29acdae7984  ansi.c
cc47d9653ed272a2d23a743ab186914d  ansi.h
b8c8b5fafcd1fd43beaeee7da1e5550f  buffer.c
04c39523dd006155ba353fb3ba1dddfb  buffer.h
ad48b92a01b68f1b8e33f95a9590e7f9  clock.c
f798d08d32ce37146d8013b821f740f5  clock.h
d79855f9ffb6a0003409ebb81290b47f  figure1.jpg
ea9ed6320aea54e698752e9a9b94adc5  figure2.jpg
97543aad843c35a031e79c5faf4ca957  figure2.png
4bc0f85c30a9d3bfaf7d355123aadf58  figure3.jpg
9adce26681f68a082f5c45bf7833c0ed  figure4.jpg
8c879f7e1e375bc7199895c9ef74d8e3  figure4.png
f6f62ebe51da7f2d8cddd6c74c53801a  figure5.png
8b4dec2e4a8518bb27bca5929531dc82  final-demo.elf
f085cf6938e4ca9624555ba76eb5b45b  ice-test
md5sum: include: Is a directory
```

```
796800c7dc1bbd2d2444ff3ad2046a51  ioflags.jpg
cac2aaebb371f2ab8150cdbe1e7f5528  kern.c
d41d8cd98f00b204e9800998ecf8427e  kern.h
aec1db7c3d6f93aa7b9b6fd98e47b4db  kern_simulation.elf
3f559a290b3a3a5fb91d2262d7a8d47d  kern_test.elf
6152637f1334fd74e0eb806912affc59  kernel_irq.c
db3b8b5c5eaa48d2e5bab408ffd172c2  kernel_irq.h
bd3f47ad7601caa6f6a64dbbd77ae784  kernel_state.h
md5sum: maps: Is a directory
5439df921ac46fd07959e43125fefa91  memory.c
b16265e8b0bfe3a510b3a25e05b8674a  memory.h
adcff2244ac92050360eacd7ab4f5dd9  message.c
e43a3792e6748d3227db13b2d9f9c549  message.h
615b2439e1f227fc8451bce70c045e11  nameserver.c
f9335969b8c71be878a915c26e7a606c  nameserver.h
08703117df738f05b4ba289925ee7bf9  notifier.c
3fd892b4a7ec6c055cdad49ad7449b59  notifier.h
78a32a3a80cad8a4cc40de1ce18fbe29  orex.ld
9d5e4e2a134be237d62d888af793de04  orientation.elf
96282407319e88eb23bb90a8daf06b9f  priorities.h
cf633eed1c5eaa9cb54a2f74f1d34fa2  private_kernel_interface.c
299821b9f1a7a97ec90a3b8863f67045  private_kernel_interface.h
f0d95167eb82b7426ca76cd33b627b9c  public_kernel_interface.c
c19a9aaa189bbaa1547f52e59009f999  public_kernel_interface.h
63c2ccbe48bb263149cfdc1d0cbe0370  queue.c
e12b085b2bf8cd425365b345831841f2  queue.h
092ccec4bf20645fcde14470e074e8ea  random.c
7b31c57ff692317d816c839156382596  random.h
cfea4010ae3135b177e3ecc8e29e1b42  readme.pdf
bf88193205eacf0d76502ea124bf7bdc  readme.rst
d24874efbd4544370f6347b1cae9cca9  readme.tex
3477616820d4a1447b5a569e246409bf  robio.c
5763b2a44810b6d0afafc27fb88cc7de  robio.h
f6b6225f4f50d316f154ee5927de4080  route.c
2fe7d2acdae03abb1904c8a460f4d53c  route.h
155b6b3d1816287618cc197aec5d5884  rps.c
6eee23bcabb82e39ca885de1563eca4f  rps.h
02566388717be1765b35028f7f16bf39  scheduler.c
0b1101123bcff9dbbf9d39542c35aacb  scheduler.h
6b8f1fdb8b0676c61b8ee263e43a18b4  sensor-periods
fba4eb1fd2006e2d70124be70af02282  swi_kernel_interface.s
00f9f65864243bdd18687e7a849c72a1  task_descriptor.c
34b26bd48a79c0a2572ca700e9ea4283  task_descriptor.h
33f883f692ee8388a7f1be0b1409c73f  tasks.c
0d3699b1a8224eb6995bb042834f66b5  tasks.h
126838fe0a43c2de5588e2f0f961d784  test_uart.c
5b820ca4fce39820f678a6080fd594ef  test_uart.h
md5sum: tracks: Is a directory
a068a724bb13ffa4e64a8902738adbae  train.c
b82348475b05867e134a4cccf9602d77  train.h
7c3a92aa318aa46c34a85a1f6d96a349  train_abstraction_layer.c
```

```
a379252956109aaae75059ecb6dbcfd6  train_abstraction_layer.h
d6df7a88aec394cbddc22e290307b83f  train_data_structures.h
926e48905f6cb40ce35d6df164408878  train_logic.c
da12dfebda37a969e5de4a109618ee15  train_logic.h
d4ad03272947d96db7fbd9528ca11ced  uart.c
1a8185a782b5c582a6ba13127ae1a1e3  uart.h
ef0786d9cd6e0d698c0bfc3a6fff0851  ui.c
a7fd7dab78620ea996b01012367b0b76  ui.h
5b609bdd0235c3858e16c053b8e53bfd  va_list_def.h
```

Elf MD5 hash:

```
chfoo@nettop37:/u/cs452/tftp/ARM/relder-chfoo/pf-submit$ md5sum *
8b4dec2e4a8518bb27bca5929531dc82  final-demo.elf
aec1db7c3d6f93aa7b9b6fd98e47b4db  kern_simulation.elf
3f559a290b3a3a5fb91d2262d7a8d47d  kern_test.elf
9d5e4e2a134be237d62d888af793de04  orientation.elf
```

Git sha1 hash: `e9e737e07135cc8864f6c8e1e89bf6cf0fbeb63a`

Appendix

System Calls

`Create` Returns the new task id, `ERR_K_INVALID_PRIORITY -1`, or `ERR_K_OUT_OF_TD -2`

`MyTid` Returns the current task id

`MyParentTid` Returns the parent task id. The parent task id is always returned regardless of the parent's state.

`Pass` (Rescheduling happens as normal in the background.)

`Exit` Task is marked as `ZOMBIE` (and rescheduling happens as normal in the background).

`Send` Sends a message to the given task ID. `-3` code is not implemented.

`Receive` Blocks until a message is received. Returns the size of the message which will be typically `MESSAGE_SIZE 16`

`Reply` Replies a message to the task. On errors `-3 -4`, an assert will fire before returning to aid in debugging.

`RegisterAs` Prepares a `NameServerMessage` structure with a message type of `REGISTER_AS` and sends the message to the Name Server. `0` is always returned because the Task ID is hard-coded and the call should never send to the wrong task.

`WhoIs` Prepares a `WHO_IS` message type and sends it to the Name Server. As noted in `RegisterAs`, we either return a Task ID or 0 if the task has not been created. However, the task ID returned may be in a zombie state.

`AwaitEvent` Marks the task as `EVENT_BLOCKED`. The task will be unblocked by the Scheduler. This call always returns 0 and the user task will be responsible for obtaining the data themselves. `AwaitEvent` supports only 1 task per event type.

`Time` Wraps a `Send` to the Clock Server. It first queries the Name Server for the Clock Server and then sends a `TIME_REQUEST` message. It expects back a `TIME_REPLY` message and returns the time.

`Delay` Similar to `Time`, it sends a `DELAY_REQUEST` message and expects back a `DELAY_REPLY` message.

DelayUntil Similar to `Time`, it sends a `DELAY_UNTIL_REQUEST` message and expects back a `DELAY_REPLY` message.

TimeSeconds, DelaySeconds, DelayUntilSeconds Same as above but in seconds. It simply converts the ticks into seconds before calling the system calls. These calls are simply for convenience.

Getc Sends a message to either Keyboard Input Server or Train Input Server. It will block until the servers have a character to return.

Putc Sends a message to either Screen Output Server or Train Output Server. The servers will place the character into the server's Char Buffer.

PutString Formats the string and calls `Putc` for every character.

PutcAtomic Like `Putc`, but accepts multiple characters and guarantees the characters are placed into the queue sequentially. This call is useful to ensure that two byte commands are not separated by a single byte command.

SendTrainCommand Sends a message type `TRAIN_COMMAND` to the Train Command Server. The call is for convenience.

PrintMessage Similar to `PrintMessage`, but this sends the string to the UI Print Server to be displayed on the lower half of the screen using a `UI_PRINT_MESSAGE` message type

Priorities

Task	Priority
Clock Notifier	0
Clock Server	0
First Task	0
Name Server	1
Administrator	2
UART Bootstrap	3
Train IO Notifier	4
Train Input Notifier	4
Train Output Notifier	4
Keyboard Input Notifier	4
Screen Output Notifier	4
Train Input Server	5
Train Output Server	5
Screen Output Server	6
Keyboard Input Server	6
Train Server	7
UI Print Task	7
Train Command Server	8
Train Switch Master	8
UI Server	8
Train Sensor Reader	9
Train Engine	9

... continued on next page

Task	Priority
Train Server Timer	10
UI Keyboard Input	12
UI Timer	13
RPS Test Start	15
RPS Server	16
RPS Client	31
Idle Task	31

Raw data collected from experiments:

All numbers shown in this section are the number of clock ticks for a clock that ticks 508000 times per second.

A test was done by cooling a train down for 25 minutes to better understand the relationship between temperature and speed. The results are inconclusive at best, but do not show any significant effect on train speed for the given test.

```
Sensor in front with 45 speed 14.  Train has not been run in a while, and ex-
hibits the slow speedup effect.

RedBoot> go -c
delta is 713033
delta is 700574
delta is 700574
delta is 675654
delta is 675641
delta is 663154
delta is 663140
delta is 650720
delta is 650720
delta is 638247
delta is 638246
delta is 638194
delta is 638234
delta is 638207
delta is 638246
delta is 638246
delta is 638246
delta is 638220
delta is 625786
delta is 625786
delta is 638220
delta is 638245
delta is 638247
delta is 625787
delta is 638247
delta is 638247
delta is 625786
delta is 625787
delta is 625760
delta is 625786
delta is 625787
```

```
delta is 625773
delta is 638247
delta is 625785
delta is 625787
delta is 625786
delta is 625786
delta is 625747
delta is 625786
delta is 625746
delta is 625786
delta is 638247
delta is 625786
delta is 625746
delta is 625787
delta is 625747
delta is 625787
delta is 625746
delta is 625773
delta is 625774
delta is 625746
delta is 625774
delta is 625761
delta is 625787
delta is 625786
delta is 625759
delta is 625773
delta is 625773
delta is 625785
delta is 625773
delta is 625747
delta is 625786
delta is 625734
delta is 625774
delta is 625787
delta is 625787
delta is 625772
delta is 625786
delta is 625720
delta is 625746
delta is 625786
delta is 625760
delta is 625786
delta is 625786
delta is 625746
delta is 625720
delta is 625773
delta is 625747
delta is 625760
average was 634134.
```

```
delta is 638260
delta is 625786
delta is 625786
delta is 625787
delta is 638246
delta is 638246
delta is 625747
delta is 625759
delta is 625786
delta is 638220
delta is 625787
delta is 638247
delta is 625786
delta is 625760
delta is 625786
delta is 625786
delta is 638247
delta is 625786
delta is 625772
delta is 625774
delta is 625787
delta is 625773
delta is 625773
delta is 625773
delta is 625787
delta is 625760
delta is 625772
delta is 638247
delta is 625747
delta is 625773
delta is 625787
delta is 625786
delta is 625759
delta is 625760
delta is 638260
delta is 625787
delta is 625747
delta is 625786
delta is 625760
delta is 625774
delta is 625746
delta is 625786
delta is 625760
delta is 625773
delta is 638247
delta is 625786
delta is 625760
delta is 625773
delta is 625786
```

```
delta is 625786
delta is 625786
delta is 625786
delta is 625760
delta is 625786
delta is 625774
delta is 625747
delta is 625759
delta is 625720
delta is 625787
delta is 625787
delta is 625787
delta is 625786
delta is 625786
delta is 625760
delta is 625786
delta is 625773
delta is 625760
delta is 638219
delta is 625773
delta is 625760
delta is 625787
delta is 625787
delta is 625759
delta is 625707
delta is 638247
delta is 638246
delta is 638234
delta is 625799
delta is 625761
average was 627824.

same test again

RedBoot> go -c
delta is 2476
delta is 625774
delta is 625786
delta is 625707
delta is 625786
delta is 625787
delta is 625720
delta is 625785
delta is 625787
delta is 625747
delta is 625773
delta is 625761
delta is 625786
delta is 625786
delta is 638233
delta is 625759
```

```
delta is 625774
delta is 625787
delta is 625786
delta is 625774
delta is 625787
delta is 625786
delta is 625760
delta is 625786
delta is 625786
delta is 625787
delta is 613246
delta is 625760
delta is 625774
delta is 625786
delta is 625747
delta is 625774
delta is 625773
delta is 625787
delta is 625785
delta is 625786
delta is 625800
delta is 625759
delta is 625774
delta is 625761
delta is 625746
delta is 613247
delta is 625773
delta is 625760
delta is 625760
delta is 625785
delta is 625747
delta is 625774
delta is 625787
delta is 625786
delta is 638206
delta is 625760
delta is 625746
delta is 625706
delta is 625773
delta is 625747
delta is 625747
delta is 625787
delta is 625786
delta is 625774
delta is 625747
delta is 625786
delta is 625721
delta is 638207
delta is 625760
delta is 625746
delta is 625785
```

```
delta is 625747
delta is 625787
delta is 625786
delta is 625773
delta is 625719
delta is 625747
delta is 625786
delta is 625785
delta is 625760
delta is 625787
delta is 625773
delta is 625787
average was 625926.


delta is 625747
delta is 625786
delta is 625786
delta is 625760
delta is 613326
delta is 625747
delta is 625785
delta is 625746
delta is 625720
delta is 625746
delta is 613327
delta is 625747
delta is 625787
delta is 625760
delta is 625785
delta is 625747
delta is 613260
delta is 625746
delta is 625774
delta is 625787
delta is 625734
delta is 613287
delta is 625786
delta is 625760
delta is 625747
delta is 625774
delta is 625787
delta is 625746
delta is 625787
delta is 625787
delta is 625759
delta is 613313
delta is 613314
delta is 625787
delta is 625786
```

```
delta is 613285
delta is 613326
delta is 625787
delta is 625787
delta is 625747
delta is 625786
delta is 625774
delta is 625747
delta is 625786
delta is 625721
delta is 625787
delta is 625721
delta is 625787
delta is 625706
delta is 613326
delta is 638246
delta is 613260
delta is 613313
delta is 625785
delta is 625707
delta is 625747
delta is 625760
delta is 625720
delta is 625759
delta is 613287
delta is 625787
delta is 625746
delta is 625720
delta is 625760
delta is 625733
delta is 625787
delta is 625786
delta is 613300
delta is 625734
delta is 625787
delta is 625786
delta is 625759
delta is 613247
delta is 625747
delta is 625760
delta is 625773
delta is 625732
delta is 613313
delta is 625786
average was 623553.

Speed test with 47 speed 8 10 laps before using ice.  sensor in back.


 i  asdfasdfasdf
delta is 1012196
```

```
delta is 1012195
delta is 999762
delta is 999762
delta is 999735
delta is 999749
delta is 999722
delta is 999722
delta is 999762
average was 1002511.
asdfasdfasdf


delta is 987275
delta is 1037089
delta is 987288
delta is 974830
delta is 987288
delta is 974788
delta is 987249
delta is 1012156
delta is 1037156
average was 998346.


speed test with 47 speed 8 80 laps after icing for about 25 mins sen-
sor in front.


iRedBoot> go -c
asdfasdfasdf
delta is 1024629
delta is 1024656
delta is 1111943
delta is 1012209
delta is 999761
delta is 999735
delta is 1012222
delta is 1099470
delta is 999735
delta is 999762
delta is 999736
delta is 999761
delta is 999749
delta is 1012222
delta is 999735
delta is 1074549
delta is 987289
delta is 987275
delta is 987302
delta is 987302
delta is 987262
```

```
delta is 987289
delta is 987289
delta is 987301
delta is 987288
delta is 987262
delta is 987262
delta is 999762
delta is 987302
delta is 987303
delta is 974802
delta is 999762
delta is 974815
delta is 974788
delta is 974762
delta is 974802
delta is 974828
delta is 974829
delta is 974829
delta is 974829
delta is 987223
delta is 987302
delta is 974762
delta is 974828
delta is 987289
delta is 987262
delta is 987262
delta is 987302
delta is 987262
delta is 974816
delta is 987288
delta is 987275
delta is 1024682
delta is 974828
delta is 987302
delta is 974803
delta is 974802
delta is 974841
delta is 987289
delta is 974803
delta is 974828
delta is 974789
delta is 987302
delta is 974816
delta is 987289
delta is 999762
delta is 974828
delta is 974829
delta is 974828
delta is 987301
delta is 987275
delta is 974789
```

```
delta is 974762
delta is 974816
delta is 974828
delta is 974789
delta is 974762
delta is 987302
delta is 987262
average was 990908.


continuing in the other direction...



Aidelta is 1049615
delta is 987288
delta is 987262
delta is 987249
delta is 987262
delta is 987302
delta is 987250
delta is 987289
delta is 987301
delta is 987288
delta is 974815
delta is 987302
delta is 974829
delta is 987289
delta is 987276
delta is 1049628
delta is 974817
delta is 1037155
delta is 974815
delta is 987289
delta is 1024669
delta is 987275
delta is 987302
delta is 974789
delta is 987288
delta is 987289
delta is 987288
delta is 987262
delta is 987289
delta is 987289
delta is 974762
delta is 974828
delta is 974828
delta is 987289
delta is 987290
delta is 987289
delta is 974802
```

```
delta is 974801
delta is 1012182
delta is 987288
delta is 987249
delta is 987289
delta is 1012183
delta is 1012235
delta is 987289
delta is 987235
delta is 987262
delta is 987249
delta is 987236
delta is 974829
delta is 987262
delta is 974828
delta is 974815
delta is 987276
delta is 987289
delta is 987302
delta is 974815
delta is 974842
delta is 974762
delta is 974764
delta is 974828
delta is 987289
delta is 987288
delta is 987289
delta is 974828
delta is 1037129
delta is 987302
delta is 1012222
delta is 974789
delta is 974749
delta is 987249
delta is 987262
delta is 987262
delta is 987302
delta is 974762
delta is 974762
delta is 974762
delta is 974842
delta is 1049589
average was 988695.



Reading data for E5 using batch sen-
sor polling method.  Done in polling loop, holding down sensor.

31409
```

31408
31409
31408
31409
31408
31409
31422
31408
31409
31408
31409
31408
31409
31407
31409
31409
31408
31409
31421
31409
31409
31408
31409
31408
31409
31408
31409
31409
31408
31421
31408
31409
31408
31409
31409
31408
31409
31408
31409
31408
31409
31408
31409
31422
31408
31409
31408
31408
31408
31409
31409
31408
31409

```
31408
31409
31422
31408
31409
31408
31409
31408
31409
31408
31408
31409
31408
31409
31421
31409
31409
31408
31409
31408
31409
31408
31409
31408
31409
31408
31409
31408
31422
31408
31408
31409
31408
31409
31409
31408
31409
31408
31409
31408
31422
31409
31408
31409
31408
31408
31408
31409
31408
```

Reading data in B5 in polling loop, using individual sen-

sor class polling method.

12699
12698
12699
12698
12699
12685
12699
12698
12699
12698
12685
12698
12473
12698
12699
12698
12699
12460
12698
12699
12473
12698
12699
12698
12699
12460
12698
12699
12698
12686
12698
12699
12698
12699
12685
12698
12699
12698
12699
12685
12698
12699
12699
12685
12698
12699
12698
12699
12684

12698
12699
12698
12699
12685
12698
12699
12698
12699
12685
12699
12698
12699
12685
12698
12699
12698
12699
12685
12698
12699
12699
12698
12685
12699
12698
12699
12685
12698
12699
12698
12699
12685
12699
12698
12698
12699
12685
12699
12697
12699
12698
12685
12699
12698
12699
12685
12699
12698
12699