Faculty of Science, Engineering and Technology
Swinburne University of Technology

SWE30001 Real-Time Programming

The Model Railroad System 2015

Document Version: v2015.0
Last revised 2-Mar-2015
Author: R. K. Allen

1. Introduction

The Faculty of Science, Engineering and Technology, Swinburne, has an HO-Gauge model railroad for students to learn about practical real-time programming. Students enrolled in SWE30001 are required to develop a control system for the railroad. This is to be done, in teams, using the language Ada (ISO language Ada, mainly revision 1995 features – compiler version GNAT 2012 which actually supports Ada2012). The rest of this document provides some details of the hardware, a very brief specification of a control system, some details of the software environment and of the existing simulation software.

2. Hardware Environment

Figure 1 shows a simplified track layout (2003 version) and summarises the hardware environment of the software system to design. The 2008 layout is shown in Figures 2 and 4. This has 19 turnouts (points). Each turnout mechanism is moved by a “Tortoise” motor that moves the turnout between the “straight” and “turned” slowly, eg 4 seconds side-to-side. Turnout state (in-position or not) is sensed by a slider switch mechanism.

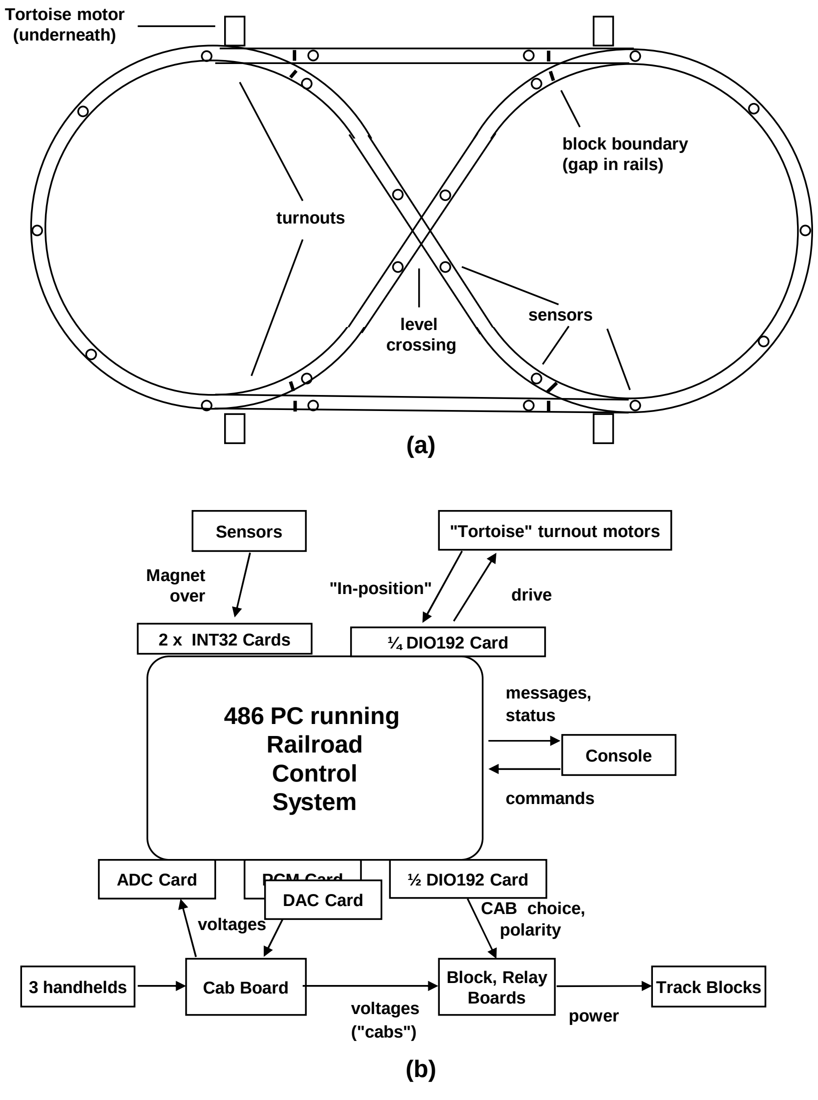
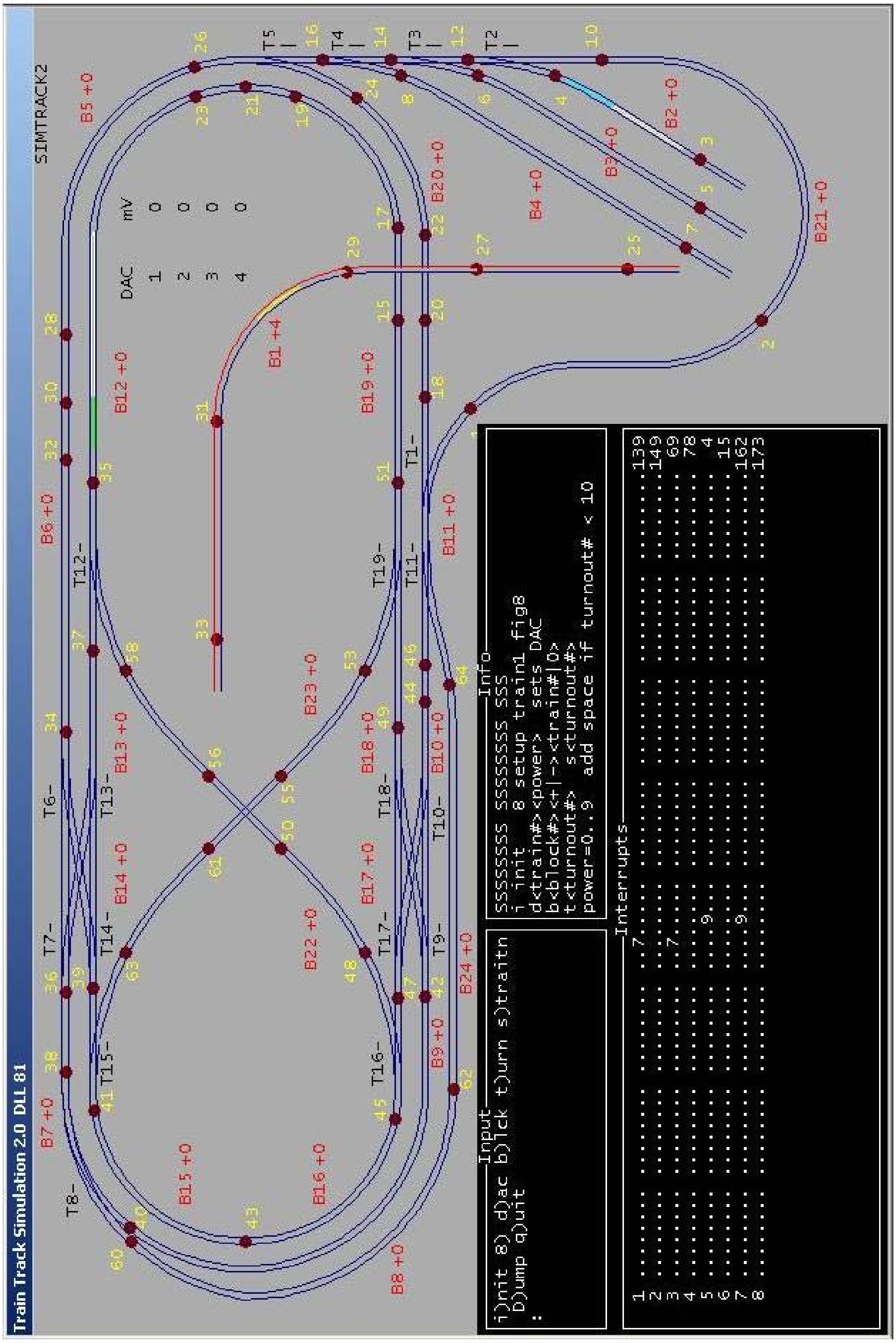


Figure 1: Overview of
(a) track layout (pre 2004, a 6 block system),
(b) hardware environment of current system (sound omitted)

The positions of the trains are sensed by Hall-effect magnetic sensors mounted on sleepers between the rails. Small magnets are mounted beneath each train (ahead of the front wheels of the locomotive & behind the rear wheels of the last carriage). Only the presence or absence of a magnet above a sensor is registered, as a single bit. There are



```
i)nit 8) d)ac b)lck t)urn s)traitn
 D)ump q)uit
:
```

```
SSSSSSSS SSSSSSSS SSS
i init   8 setup train1 fig8
d<train#><power>  sets DAC
b<block#><+|-><train#|0>
t<turnout#>  s<turnout#>
power=0..9  add space if turnout# < 10
```

```
 1 ........ ........ ........ ..7..... ........ ........ ........ ........139
 2 ........ ........ ........ ........ ........ ........ ........ ........149
 3 ........ ........ ........ ..7..... ........ ........ ........ ........ 69
 4 ........ ........ ........ ........ ........ ........ ........ ........ 78
 5 ........ ........ ........ ....9... ........ ........ ........ ........  4
 6 ........ ........ ........ ........ ........ ........ ........ ........ 15
 7 ........ ........ ........ ....9... ........ ........ ........ ........162
 8 ........ ........ ........ ........ ........ ........ ........ ........173
```

Figure 2: Screen shot of simple2.exe (2011). Three simulated trains are shown on blocks 1, 2 and 12. Note that only block 1 has been set up for its occupant, train 4, as indicated by “B1+4” and the red rail to the right of the train. (Right according to the direction of travel. The + indicates “normal” polarity.) Block boundaries aren’t shown here.

57 sensors (64 supported by the electronics): generally three guard the three branches of each turnout; others guard the approaches to crossings; the remaining sensors subdivide the longer sections of track.

The track is divided into 24 "blocks", that is, there are gaps in the rails to make 24 electrically isolated sections. (See Figure 4.) Most boundaries between blocks are within the left or right branches of the turnouts, 2.5 to 12 cm from the sensors. Each block is powered by a “block driver” containing a power transistor that follows reference voltages, known as "cabs". A “cab” is either the output of a digital-to-analogue converter (DAC) or ground (zero volts, “cab” zero) or a pulse-width modulated waveform (PWM, under development). There are 4 DACs, so up to 4 trains can run at once. The DACs are on an interface card, a CIO-DDA06/Jnr. (The voltage they deliver is determined by a 12-bit number.) All the blocks under a train must be associated with the same cab and hence deliver the same voltage so there is continuity of power (and no overloads of the power transistors) as the wheels cross block boundaries (the gaps between rails). Cab selection needs 3 bits (a number of type Cab_Id). As well the polarity has to be controlled separately for each block. This is done with double-pole change-over relays, driven by one bit. These actually take about 12 ms to change but that time can be ignored by control software.

In all there are six digital I/O cards plugged into the ISA bus of a 486 PC (or Pentium) and these act as interfaces to the external electronics. There are two INT32 cards that are connected to the Hall-effect sensors. If any of the bits changes then the appropriate card raises an interrupt request; which bit or bits have changed can be discovered by reading 8-bit input registers and comparing them with the previous values. As well as providing DACS, the CIO-DDA06/Jnr card has a 24-bit digital I/O sub-system that is used to read switches on three hand-held controllers. Another card, a CIO-DAS08, has analog-to-digital converters and is used to read the values of speed knobs on these controllers. The CIO-DAS08 card also has a digital I/O sub-system: 8 bits are used to control a sound system.

Another hardware subsystem is the console, that is, keyboard and screen. User input comes from the PC keyboard while the screen is for status and error messages. Finally the serial port COM1 can be used for logging.

3. Minimum Functionality of the Control System

The software must support the simultaneous operation of up to four trains controlled from the keyboard by single character commands or by three handheld controllers. One train (the tram), number 4, must go backwards and forwards on block 1 with no user control except overall speed; it only turns around at the ends (sensors 25, 33). At least one other train must support an "unmanned" mode of operation. The keyboard commands must include (functional names not keys):

Faster / Slower:	increase/decrease the desired speed (not the same as voltage) by some fraction eg 1/8th of the total range
Halt:	unless stationary, stop immediately, and set the desired speed to zero.
Left / Centre / Right:	take the left/right branch at all turnouts ahead for which there is a choice*.
Forward/Backward:	if stopped (a reasonable restriction) then reverse the desired direction of travel
Unman/Man:	set the train to unmanned/manned mode: when unmanned it will have predictable behaviour, eg always keep to the outer oval, but ignore (some) user commands.
Reset:	(all trains) initialise the **software** state of all the trains, assuming that the actual trains have been replaced on the track at standard positions, facing in standard directions. Ensure power is zero.
Sim_dump	(simulator) dump current state and history of the simulation to standard output.
Train_dump	(student code) state of each train, on the screen, for debugging.
Log_dump	(supported by package Slogger) writes a time-stamped log.

*If a train approaches a single turnout in the diverging direction there is a potential choice. The Centre command means “no preference” or “as is”, that is, the train must accept the current turnout position and go left or right accordingly. If the train approaches via the left or right branch, ie "converging", then there is no choice -- the turnout must be set appropriately and its position is said to be "required". Note that these keys control an internal state with values LEFT, NO_PREF, RIGHT; they are not single commands applying only to one turnout. In particular NO_PREF allows one train to automatically chase another while overall Manned.

Command keys are prescribed in Table 5 (at end). The control system must display at least train status and error messages on the PC screen (80 x 24 text).

Responsiveness: Except when it is unsafe to do so the control system must allow the users to control the trains. A train should not, for example, deviate onto a side branch of a turnout when the way ahead is blocked. Instead it must stop and, when the way ahead is clear, automatically resume moving at the original speed. (Variants: the user changes speed, direction or preference while the train is waiting; the way ahead has a speed limit.) If the user makes a request (Left or Right) to deviate from the current setting of a turnout too late for it to be obeyed safely, then the request may be ignored with respect to the current turnout. Conversely if the train has already stopped waiting for a turnout movement to complete or a block to become free, then new commands should be obeyed when safe to do so. For example, the user may change their mind and go Left or go Backward.

Safety requirements: The trains must be safe from collision and from derailment due to turnouts operating incorrectly or from any user command. Similarly the block drivers must be safe from short-circuit: all blocks under a train must have the same cab, and the electrical polarity from one side of the train to the other must be consistent.

Power-delivery strategy: In order to operate multiple trains, the power settings (cab and polarity) for a block are set appropriate to a train only when that train is about to enter it. Similarly these are set back to zero when the train leaves. (This aids debugging.)

Stop-start strategy: If a train reaches a sensor and it is unsafe to proceed then the train must stop (an “emergency stop”) and wait until it is safe – then it automatically starts moving. It must resume moving at the original speed and direction unless these have been changed by user commands while the train has been waiting. The position of a train is only known accurately at the edge of sensors. Therefore reaching the start of a sensor that guards a resource is the latest instant for checking whether it is safe to proceed, for example whether a turnout movement is complete. Similarly, that is the instant to check if the other train has vacated the next block so the CAB and polarity of that block can be set for this train.

Reverse-power braking: Don’t use this strategy! This has proved to be very unreliable in previous years.

Go-slow strategy: Note that because of inertia a train at full speed will coast forward up to 40mm when power is dropped to zero. In some places 25mm past the sensor is not safe – there might a collision with another train for example. More generally if the user decides to reverse out then keeping track of the train position is easier if the relevant magnet stops on top of the sensor, however see Topolog2 “Just past” below. The train speed may need to be limited in the section of track before a sensor where it might be required to stop. In the case of “boundary sensors” (those exactly over a block boundary, ie gap) the overshooting may cause the wheels to short-circuit the next block, eg if that next block cannot be allocated to this train because it is occupied by another train. Only a very small amount of overshoot is allowed in the sequence of chained turnouts numbered 5,4,3,2.

Direction of travel: (Refer to Figures 2 and 4) By convention in the simulator, train 4 (the tram) is always on block 1 and starts off in the vertical section between sensors 25 and 27. Other initial position conventions are built into the simulator but can be varied. Assume train 1 is on block 12. Irrespective of the train's orientation when power (more that about 4 volts*) is applied in the "normal" polarity to block 12, the train will move to the left/west. If all the blocks are set to "normal" polarity and the turnouts are set straight (except number 8 which should be turned), trains will travel **anti-**clockwise around the ovals (irrespective of the orientation of the locomotives). As can be deduced from the diagrams, when a train moves from one block to another the polarities of the two blocks must be the same except at the gaps near sensors 48 and 53 -- here the two blocks involved need to have opposite polarities. Otherwise the power will be short-circuited.

*Note: speed is **not** proportional to voltage! However each handheld knob and the set of discrete settings provided by faster-slower keys or buttons should control speed in a roughly linear way. For the knob there should be a zero-speed zone at the start. For the discrete settings the lowest setting should be very slow to aid debugging. The different physical locomotives have different speed characteristics and these vary with the amount of oxide on the wheels and track.

4. Software Environment

The ultimate target computer system (platform) is a 50MHz 486 PC (or 100MHz Pentium) running an executable without an operating system, ie “embedded” software. The executable contains a MarteOS kernel and is created by a GNAT/MarteOS cross-compiler hosted under Linux (on mercury). However there is simulation software that runs under Windows NT/2000/XP/7 (Win32) using the GNAT Win32 compiler. Develop train software first under Windows, linking into the simulator, and then, when sufficiently debugged, port it to MaRTE under Linux. To aid this there are four packages that support portability: Swindows, Io_Ports, Halls2 and Slogger. As well Topolog2 provides information about the layout independent of the simulator.

4.1 Swindows

The control console must use a non-scrolling 80x24 text window. The Swindows (“Swinburne simple windows”) package provides this on each platform. Swindows provides an 80x24 text window with text-mode sub-windows and gotoxy facilities. Under MarteOS this window occupies the entire screen and Swindows acts as an adapter API to other packages. The built-in MSDOS (or “Command prompt”) window provided by the various versions of Win32, does not support gotoxy (ability to write characters to any row-column position). Therefore the Win32 version of Swindows uses a third-party package, AdaGraph, which provides a very simple interface to some Win32 drawing facilities. AdaGraph is also used by the simulator to draw the track and trains. AdaGraph only supports one MS window on the desktop, so for the simulator, that window is split into two areas: an 80x24 "Swindows" area in the lower part; a graphical display of the simulated hardware in the upper part.

A simulator application is an ordinary EXE running under Win32 and there is always another MSDOS window and this window can be used for debugging etc using Text_Io. However all such extra Text_Io output must be eliminated from the target (“hardware”) version of the software as it would overwrite the other text at arbitrary places on the console.

Swindows provides keyboard input by polling the keyboard at 50 ms. Procedure Swindows.Get_Char blocks until a key is pressed – your control software must be based on this.

4.2 Io_Ports

On the target system all access to device registers is via I/O instructions that specify a 16-bit address in "IO space". Package Io_Ports acts as an Ada interface to these instructions. There are Swinburne versions of Io_Ports for the MarteOS (hardware) and Win32 (simulator) environments (and formerly the DOS environment).

In the simulation environment the version of Io_Ports calls methods in a large package, Simrail2, but only if the IO addresses are correct. Both versions log “interesting” Write_Io_Port events to Slogger.

4.3 Halls2

Simulated interrupts are done by "call-backs" from Simrail2 to some procedure in a student-written package. The corresponding version of package Halls2 has a procedure to install a pointer (or in Ada terminology, an *access* to) the specific protected procedure. The pointer is actually stored in Simrail2 and call-backs come from the one and only worker thread in Simrail2.

In the hardware version Halls2 actually handles the interrupts from the INT32 cards and the call-backs come from there. There is a separate handler thread for each card, and each handler has a 16 element buffer. When any bit on a card changes there is an interrupt. The handler procedure saves all 32 bits of that card plus house-keeping data in the buffer. This unblocks the appropriate worker thread which then empties the buffer, passing each set of 32 bits (plus a housekeeping/debugging byte) to the user-provided procedure.

4.4 Slogger

This package provides logging services safely within a multi-threaded system, efficiently. The Simrail2 version simply writes comma-separated numbers to a disk file and Slogger.Finalize closes it.

The MaRTE version stores the log in a 100kbyte buffer (like a RAM drive).and Slogger.Finalize (or Flush) writes the text to the serial port (RS232 COM1: at 57600 baud). The serial cable must connect to a Win32 PC running the program **read_log.exe**. This writes the data (space-separated) to a disk file. **Read_log** also performs calculations on log files and converts them to a form for subsequent display and analysis using Excel.

4.5 Topolog2

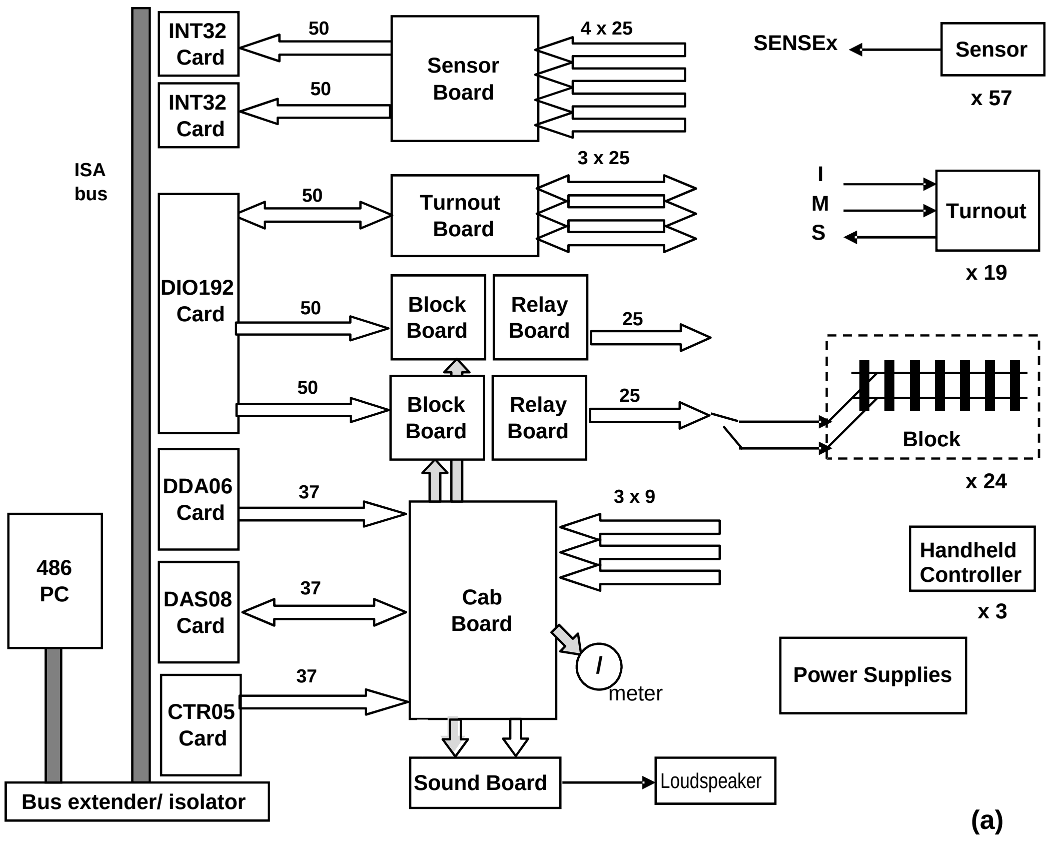
The control system should be based on a topological model of the track layout, ie what is connected to what. Because actual train speed cannot be predicted, accurately keeping lengths in the model is of limited use. A topological model is provided as an Ada package, Topolog2. This is a state-free package, in the sense that varying information must be provided by parameters to procedures and functions. Although these calls to Topolog2 are complex they do provide all the information needed for control system code. Use it.

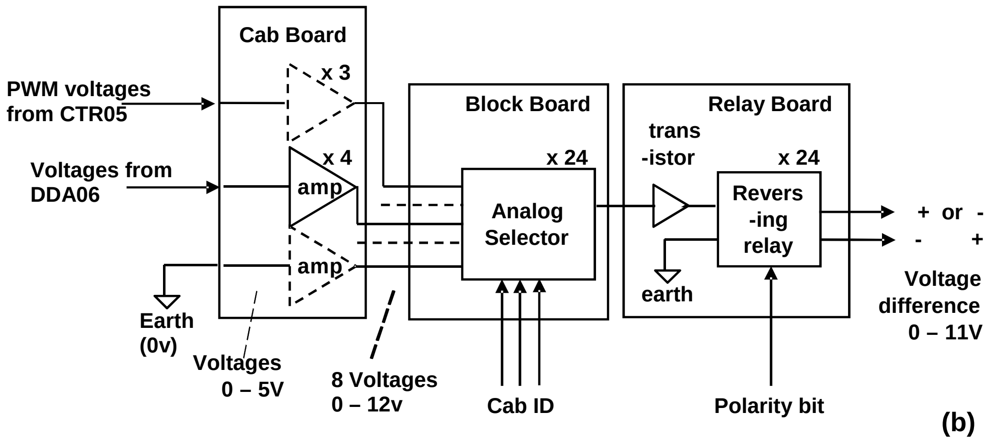
(2013) Topolog2 now provides “go slow” data. These are safe stopping distances for each sensor in each direction of travel. For example, in the reverse direction there is only 17mm from the centre of sensor 47 to the block boundary at turnout 16. So if s47 is ahead then the train cannot safely travel at full speed unless it can guarantee that block 16 is available and the turnout is straight.

Topolog2 uses both edges of a sensor, ie On event and Off event (not just “fire”). Each of the two magnets of a train is tracked. Of course the current front magnet is the most interesting. Most queries, eg Check_Entering_Turnout, is only valid if the front magnet is over a sensor.

Just past: Physically this is a position where the magnet has skidded “just past” a sensor, ie the train failed to stop in time. Logically Topolog2 recognizes this as a state. For decision-making this is very like stopping on top of a sensor, so queries like Check_Entering_Turnout can be called even though the train has skidded into that region of track denoted as “turnout”. Of course if the user decides to reverse the travel direction then, before it starts moving, the magnet position will be in the state “just before”. This state is needed because the user might reverse the direction again still with the train stationary, back to “just past”.

See Topolog2_notes.pdf and code comments.





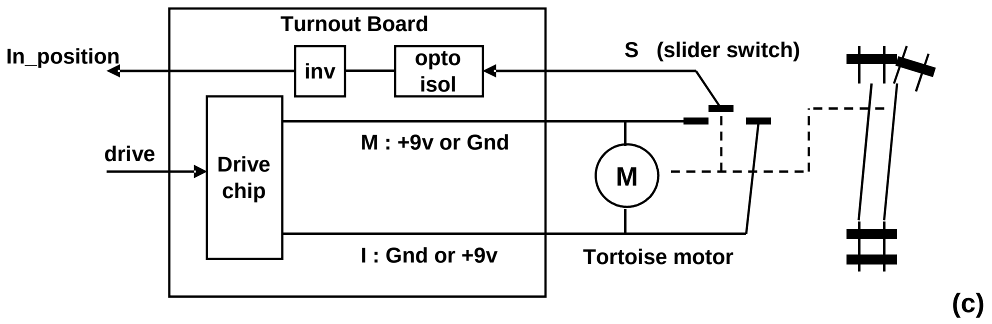


Figure 3: Diagrams of
(a) overall electronics,
(b) detail within the Block Driver and Relay boards
(c) mechanism for turnout movement (highly simplified)

5. Hardware Details

5.1 Electronics

The purpose-built electronics has eight printed circuit boards: one Sensor (Hall) board; one Turnout board; two pairs of Block and Relay boards; one Cab board that deals with "cab" voltages and hand-held controllers; one Sound board. Nine 25-wire ribbon cables connect these boards to the track layout. Eight wider ribbon cables connect the boards to the interface cards (plugged into an ISA "bus extender" which in turn is plugged into the computer's motherboard). Only a few opto-isolator chips are used (on Sound Board). (The pre-2008 system had many for electrical protection of the expensive interface cards.) There are also three hand-held controllers, which are essentially switch boxes with a speed knob. See Figure 3 for some details. Detailed circuit schematics are available. The Halls board and the layout itself have LEDs (light-emitting diodes) for debugging. An oscilloscope and a 34-channel logic analyser can be hooked up to the electronics for debugging and timing measurement.

5.2 Some notes on the turnout operation:

(See Figure 3(c).) Each turnout is driven by a Tortoise brand motor. This is a DC motor connected to the turnout via a plastic gear train and a piece of steel "piano" wire. Driver chips produce the correct polarity of 9 volts controlled by a single bit. When the motor reaches its limits it simply stops in a stalled state (and hence draws more current, about 20mA but the power supplies can cope with the overall load of 19 stalled motors). The mechanism also has a slider switch which is wired to control one bit that means "in position" (or not). The slider switch circuit shows "out-of-position" immediately and changes to "in-position" after 3+1 seconds. Actually the bit changes a little early, ie when the motor has moved across most of the way. NB: the slider switch is at the wrong end of the piano wire to detect friction and obstructions between the rails (cf the University of Northern Iowa system).

5.3 Interface Cards

See Tables 1 and 2. Two cards supporting digital I/O, namely the DIO192 and DDA06/Jr, use or emulate 8255 chips. Each deals with 3 bytes (registers) that can be configured for either input or output using a control register. When there are two 8255s (giving a total of 48 bits I/O) the bytes are designated PA, PB, PC, PCTL ("control"), and QA, QB, QC, QCTL (or A-Low, B-Low, … QCTL-High). The DIO192 board has four pairs of 8255s so provides 192 bits, but this system does not use the last quarter of the board. Note that reading the registers configured for output may always give zero.

register	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*	*Base address* = 16#220#
PA1	p	X	X	X	p	X	X	X	Blocks **1**, 2 (p = polarity, 0=normal, xxx = 3-bit numbers for CAB selection)
PB1	p	X	X	X	p	X	X	X	Blocks **3**, 4
PC1	p	X	X	X	p	X	X	X	Blocks **5,** 6
PCTL1	**X**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	Mode 0 and all bits output. (bit 7 is Mode Set Bit, 1=set, subsequent 0 enables tristate, ie connects to outside)
QA1	p	X	X	X	p	X	X	X	Blocks **7,** 8
QB1	p	X	X	X	p	X	X	X	Blocks **9,** 10
QC1	p	X	X	X	p	X	X	X	Blocks **11,** 12
QCTL1	**X**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	Mode 0 and all bits output. (bit 7 as above)
	7	*6*	*5*	*4*	*3*	*2*	*1*	*0*	*Base address* = 16#228#
PA2	p	X	X	X	p	X	X	X	Blocks **13**, 14
PB2	p	X	X	X	p	X	X	X	Blocks **15**, 16
PC2	p	X	X	X	p	X	X	X	Blocks **17,** 18
PCTL2	**X**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	Mode 0 and all bits output. (bit 7 as above)
QA2	p	X	X	X	p	X	X	X	Blocks **19,** 20
QB2	p	X	X	X	p	X	X	X	Blocks **21,** 22
QC2	p	X	X	X	p	X	X	X	Blocks **23,** 24
QCTL2	**X**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	Mode 0 and all bits output.
	7	*6*	*5*	*4*	*3*	*2*	*1*	*0*	*Base address* = 16#230#
PA3	*P*	*P*	*P*	*P*	*P*	*P*	*P*	*P*	position state of turnouts 16, 15, 14, .. 9

									(1 = in position, 0 = middle)
PB3					T	d	d	d	Drive bits for turnouts 19, 18 17 (0 = pull straight) zero for unused bits T is for testing, eg connected to a Hall input to cause an interrupt (to be installed during the semester)
PC3						*P*	*P*	*P*	position state of turnouts 19, 18, 17 Ignore other bits
PCTL3	**X**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	PB out, PA, PC in
QA3	d	d	d	d	d	d	d	d	Drive bits for turnouts 8, 7, 6 .. 1
QB3	*P*	*P*	*P*	*P*	*P*	*P*	*P*	*P*	position state of turnouts 8, 7, 6 .. 1
QC3	d	d	d	d	d	d	d	d	Drive bits for turnouts 16, 15, 14, .. 10, 9
QCTL3	**X**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	QA, QC out, QB in

Table 1: DIO192 register and bit usage (non-blank denotes bit is used).

The digital-to-analogue card DDA06/Jr has one 8255 section and 6 pairs of “voltage bytes”. Each pair takes a 12-bit number: 0 gives -5 volt, 2048 gives 0 volt and 4095 gives +5 volt. The protocol is to write the lower significant byte and then the most significant nibble. For this project 8-bit precision is plenty so Table 2 shows how to use only 8 bits (padding with zeroes). Failing to initialise the DDA06/Jr card or incorrectly giving a 12-bit zero produces a voltage that starts as -5 volt but the actual (not simulated) electronics converts this to maximum +13 volt – **beware**!! Also note that DA0 provides the reference voltage for CAB number 1, always used by train 1, DA1 for CAB 2, etc.

register	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*	*Base address = 16#240#*
DA0_lo	X	X	X	X	X	0	0	0	DA0 low byte (bottom 3 bits can be used, recommend 0)
DA0_hi	0	0	0	0	1	X	X	X	DA0 high nibble (top 4 bits always zero)
DA1_lo	X	X	X	X	X	0	0	0	DA1 low byte
DA1_hi	0	0	0	0	1	X	X	X	DA1 high nibble
DA2_lo	X	X	X	X	X	0	0	0	DA2 low byte
DA2_hi	0	0	0	0	1	X	X	X	DA2 high nibble
DA3_lo	X	X	X	X	X	0	0	0	DA3 low byte
DA3_hi	0	0	0	0	1	X	X	X	DA3 high nibble
DA4_lo									DA4 low byte (unused)
DA4_hi									DA4 high nibble (unused)
DA5_lo									DA5 low byte (unused)
DA5_hi									DA5 high nibble (unused)
PA				*t*	*t*	*d*	*b*	*r*	Base_address + 12 input from hand controller A tt = turn preference, 11=centre, 10=left, 01=right d = travel direction, 1=forward, 0=backward b = black button, 1=up “bell” r = red button, 1=up “horn”
PB				*t*	*t*	*d*	*b*	*r*	input from hand controller B
PC				*t*	*t*	*d*	*b*	*r*	input from hand controller C
PCTL	X	0	0	1	1	0	1	1	Base_address + 15 (PA, PCH, PB, PCL input) (X=1 set, subsequent 0 enables tristate)
	7	*6*	*5*	*4*	*3*	*2*	*1*	*0*	

Table 2: DDA06/Jr register and bit usage

(See Table 3.) The analogue-to- digital card DAS08/Jr has 8 pairs of “voltage bytes” configured differently from the DDA06. Each pair returns a 12-bit number: 0 means -5 volt (impossible from the handhelds), 2048 means 0 volt (approx) and 4095 means +5 volt (approx). The protocol is to write the channel to the CS register then anything (eg 0) to the upper byte. Then wait until the most significant bit of the CS register changes back to 0 (approx 25 microseconds later) and read the upper byte. For this project 8-bit precision is plenty so the lower byte can be ignored, however whatever precision is chosen when you compare old and new values you should treat the least significant bit (or the bottom 2 bits) as “noise”. That leaves 6-bit precision (64 possible values).

The card also has a digital output byte (used for the sound system) and a digital input byte (an unused 8-bit connector on the Cab board). These two bytes share a port address, PA.

register	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*	*Base address = 16#**310**#*
AD_lo	X	X	X	X	0	0	0	0	low nibble of measured voltage
AD_hi	X	X	X	X	X	X	X	X	(read) high byte (0volt is 128 approx, -5volt is 0)
									(write) start conversion of a channel (data written is ignored) The conversion will take about 25 microseconds.
CS									(base+2) Control and status
	e	-	-	-	-	c	c	c	(read) e=EOC (end of conversion) 1=busy 0=done ccc=current channel (of input multiplexor)
	0	0	0	0	0	C	C	C	(write) set channel 0..7 (For the railroad 0..2 selects one of the hand controllers A, B, C. The voltage controlled by its speed knob will be read.)
PA									(base+3)
	X	X	X	X	X	X	X	X	(read) 8 bits, unused but available on the Cab Board
	B	H	B	H	B	H	B	H	(write) output to sound system H=horn, B=bell for trains 4, 3, 2, 1 (1=on)
	7	*6*	*5*	*4*	*3*	*2*	*1*	*0*	

Table 3: DAS08/Jr register and bit usage

The two INT32 cards use Z8526 chips. These have several internal registers that can be accessed via the CS registers. Note complete INT32 code is provided in package Halls2.

register	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*	*Base address = 16#250# IRQ=5*
									(base+0) unused
PB1	X	X	X	X	X	X	X	X	(base+1) sensors 16,15,..9
PA1	X	X	X	X	X	X	X	X	(base+2) sensors 8,7,…1
PCS1	c	c	c	c	c	c	c	c	(base+3) control & status
QC1									
QB1	X	X	X	X	X	X	X	X	(base+5) sensors 32,31, .. 25
QA1	X	X	X	X	X	X	X	X	(base+6 sensors 24,23, . 17
QCS1	c	c	c	c	c	c	c	c	(base+7) control & status
									Base address = 16#258# IRQ=3
									(base+0) unused
PB2	X	X	X	X	X	X	X	X	(base+1) sensors 48,47, .. 41
PA2	X	X	X	X	X	X	X	X	(base+2) sensors 40,39, .. 33
PCS2	c	c	c	c	c	c	c	c	(base+3) control & status
QC2									
QB2	X	X	X	X	X	X	X	X	(base+5) sensors 64,63, .. 57
QA2	X	X	X	X	X	X	X	X	(base+6 sensors 56,55, .. 49
QCS2	c	c	c	c	c	c	c	c	(base+7) control & status
									Z8526 internal registers (accessed via CS registers: first cycle write internal address to CS, second read or write CS)
MIC	0	0	0	0	0	0	0	X	master interrupt control (X=1 reset, X=0 clear reset)
MCC									master configuration control
PAMS									port A mode specification
PACS									port A command & status
PADD									port A data direction
PAPP									port A pattern polarity
PAPT									port A pattern transition
PAPM									port A pattern mask
etc									Similarly for port B.
	7	*6*	*5*	*4*	*3*	*2*	*1*	*0*	

Table 4: INT32 register and bit usage

The CTR05 Counter-Timer card can provide 3 cabs that are pulse-width modulated (PWM). That is, instead of an analogue voltage varying continuously between 0V and 10V, the voltage is either 0V or 10V with the duty cycle changing. This mode of train operation gives better low-speed control, as the narrow 10V spikes "punch through" any oxide layer on the rails. You might decide to run train 1 on cab 5 instead of the normal cab 1. Refer to separate documentation.

5.4 Handheld Controllers

See Figure 6 and a separate document, handheld_pinout_v3.pdf, for details.

6. Some Details of the Simulation Software

The aim is to test the control software in the absence of the railroad hardware. Therefore the physical behaviour of the electronics and the trains is simulated. The simulation continually updates a graphics window that shows the simulated position of the trains as well as the polarity and CAB number of each block (eg B12+3), the voltages in mV (eg 8600) and the state of each sensor (by colour) and turnout (eg / for turned, m for middle).

The common interface between the control software and either the simulation software or the real system is:

(a) common definition packages: Unsigned_Types, Raildefs, Dio192defs, Dda06defs, Das08defs, Int32defs;

(b) the specification part of package Io_Ports, Halls2, Swindows and Slogger (these have very different package bodies on the two platforms);

(c) procedure Install_Int_Handling of package Halls2. This is used to install a procedure to later receive calls that correspond to interrupts from the INT32 cards. (The cards and interrupts are simulated within Simrail2.)

Notes

1. procedure Reset is used to initialise Simrail2. As mentioned in Section 4.1, the version of Swindows used with the simulator shares a screen window with the graphics display so Simrail2.Reset must be called before any other Swindows code. For details see example program(s) provided with the Simrail2 source.
2. It is important that no student software exploits any simulator support package, eg Simconst2, Simtrack2 as these **cannot** be present in the MarteOS system.
3. Internally the simulator, Simrail2, uses fixed time steps of about 10ms. IMPORTANT: The control software (student-written) must "yield" the CPU – it must essentially be event-driven. Your software is expected to have one or more worker threads. Also using tight loops for polling or timing (eg with procedure Exec_Load.Eat) for more than 1 or 2 milliseconds will undermine the simulation.
4. Violation of items 13, 15 and 16 below cause an error message and the relevant train becomes unusable, but simulation continues. Your code can call Simrail2.Dump at any time to supplement the information already in the MSDOS window.
5. Simrail2 does not itself write events to Slogger, but the sim version of Io_Ports does.

Physical characteristics that are simulated include

1. The relevant IO ports. (The interface cards have multiple registers, "ports", in the PC IO address space.) Accessing most other IO addresses produces error messages from Io_Ports.
2. The bits within these ports. Access to the data bytes of an interface card before it has been initialised and attempts to initialise it with the wrong bit pattern can raise exceptions.
3. The effect of these bits within the purpose-built electronics.
4. The time behaviour of the block polarity-reversing relays. (Actually this is only one 10ms step and can be ignored for control purposes but during stopping it does result in a little unavoidable coasting/skidding.)
5. The time behaviour of the turnout motors.
6. Reliable turnouts. (Pre-2013 unreliability was built-in. For example there was an x% chance of failing to switch across.) Now there is a random variation of up to 0.5 sec in time to change over.
7. The mapping of 12-bit numbers onto voltage and of voltage to steady-state train speed. This is only approximate. Note that about the first third of the positive voltage range gives zero speed.
8. The inertial characteristics of the trains (very approximate, as rolling stock not yet measured) especially those relating to sudden stops and start-up (this is affected by variations in electrical resistance at the wheel-rail interface). (Inertia is roughly implemented and is assumed the same even though real model locomotives have different masses and friction.)

9. The physical dimensions of the track including the positions of the sensors. (To be improved.)
10. The physical dimensions of the trains, including the front to rear wheelbase and the positions and effective widths of the magnetic sensors. The position of the magnets is assumed to be about 20mm ahead of (and behind at the rear) the contact point of the nearest wheel.
11. The ability to short-circuit the power supply if a wheel bridges a gap with different polarities or voltage across it. This is fatal to individual trains in the simulator but recoverable on the actual hardware.
12. The derailment of a train if it enters a turnout that is in the wrong position, or if it is over a turnout when the turnout moves.
13. The detection of a train entering a block with wrong power on it or a turnout occupied by another train.
14. (Not implemented) The ability of one train to contact another (crash or side-swipe).
15. (Not implemented) The variations of length between different locomotives and between different carriages/wagons. The effective width of the sensors is assumed to be 10mm for all.
16. (Not implemented) Handhelds and PCM board.

7. Required Keys for Control System

Keys for Trn 1	Keys for Trn 2	Keys for Trn 3	Keys for Trn 4	Command
q	**t**	**i**		Want train to go “forward”
z	**b**	**, comma**		Want train to go “backward”
w	**y**	**o**	=	Faster
x	**n**	**. dot**	-	Slower
a	**g**	**k**		Want train to go Left
d	**j**	**; semi**		Want train to go Right
s	**h**	**l**		Centre = “No preference” – left or right **“as is”**
c	**m**	**/ slash**	[	Halt / Bell** (** these keys toggle the state of notional
e	**u**	**p**	]	Horn** buttons for horn and bell)
1u	2u	3u		Change to “unmanned” mode (2 key sequence)
1m	2m	3m		Change to “manned” mode (from “unmanned”)
1d	2d	3d	4d	(optional) Dump attributes to screen (for debugging)
		S		Write internal state of simulator to stdout (for debugging)
		D		Dump slogger log to file (simulator) or serial line (hardware)
		R		Perform software reset of all trains, etc*

Table 5: Single-Key Commands Required for Control System

* Command R cannot be implemented by calling Simrail2.Reset (which only works for initialisation) – hence this can only be useful with the hardware version, where it can be very useful -- the user repositions the trains.

Note that the keys are arranged into pads in a fairly intuitive way (Figure 5); remember **W Y O** for the top of the pads. The digits act as “escapes” so forming 2-key command sequences. Other sequences involving more than one key press for a command are allowed **only** during program initialisation or as extra debugging commands.

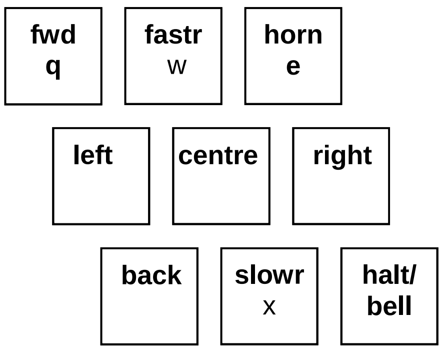


Figure 5: Keypad layout

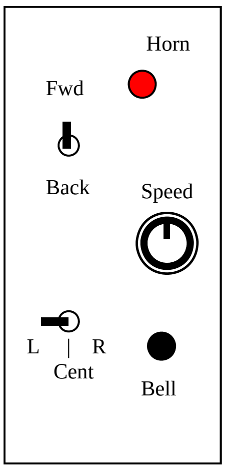


Figure 6: Handheld layout

8. Layout Numbering

(Figure 4 below) WARNING: don’t hard code sensor IDs, etc! Use Topolog2.
Note that 6-wheel locos like Diesel often get stuck on the curve of Turnout 1 (Toby is ok).

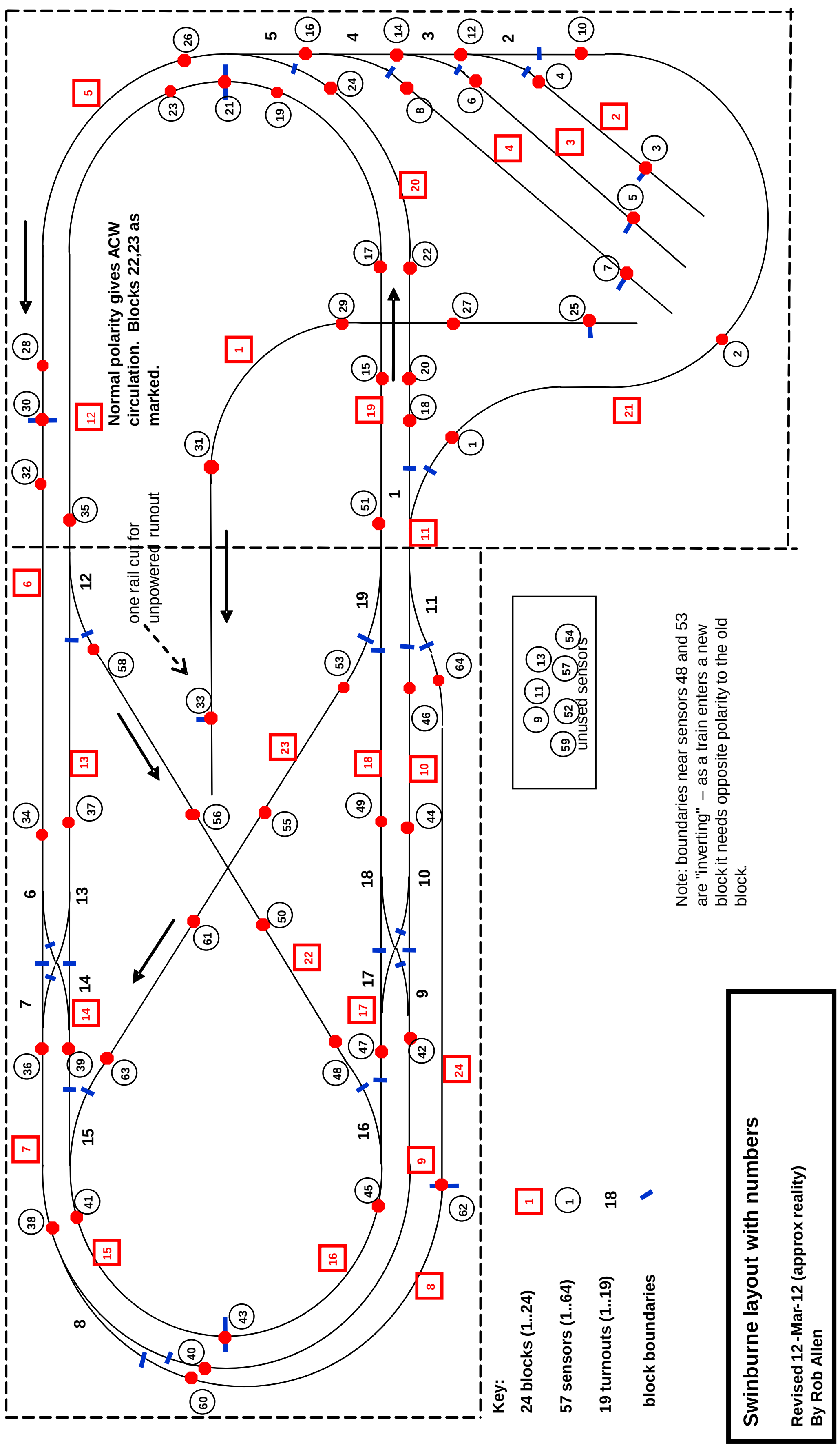

Normal polarity gives ACW circulation. Blocks 22,23 as marked.
one rail cut for unpowered runout
unused sensors
Note: boundaries near sensors 48 and 53 are "inverting" – as a train enters a new block it needs opposite polarity to the old block.
Key:
24 blocks (1..24)
57 sensors (1..64)
19 turnouts (1..19)
block boundaries
Swinburne layout with numbers
Revised 12-Mar-12 (approx reality)
By Rob Allen

Figure 4