TPL – The theory

My aim was to see if I could create an automated layout with lots going on that didn't just run around in circles. Having looked at JMRI (briefly I must say) and DCC++ I began to wonder whether I could actually make a simpler automation system and run it entirely on the Arduino used for DCC++.

Some of the automation techniques I read about using python scripts in JRMI made my blood run cold… there's a lot I could say here but won't without a pint or two.

It seemed to me that basing an automation on "signals controlling trains" leaves a lot of complex technical problems to be solved… and wanting to be cheap, I didn't want to invest in a range of block occupation detectors or ABC braking modules which are all very well on circular layouts but not good at complex crossings or single line operations with passing places. Also I didn't want the automation to be an obvious cycle of movements… some random timings need to be introduced so that two trains don't always arrive at the same place in the same order, nor go on the same journey in a predictable cycle.

By reversing the usual assumptions, and slaughtering a few sacred cows, I think I have a workable, extendable and cheap solution. I have based the DCC bits on Locoduino which is a modified DCC++ as its easier for my code to call into it.

My small C++ library (lets call it TPL), sits between the layout owner and Locoduino so that the layout owner can write automation scripts in a form that is much more user friendly. In fact the automation is written in the Arduiono IDE as per a normal Arduino script but almost all the C++ boilerplate code is stripped away where you don't need to see it.

In order to keep costs down, I have implemented turnout controls using an I2C bus architecture where a single cheap PWM board can drive up to 16 cheap servos. Up to 4 servo boards can be chained together so you can handle up to 64 turnouts (or indeed servos for animations). As far as the layout owner is concerned, each turnout just has a number from 0 to 63 corresponding to the slot it's plugged into. TPL refers to turnout state as LEFT or RIGHT to avoid confusion.

Sensors are all cheap IR detectors connected 16 at a time to cheap I2C boards. I avoided magnetic sensors because they only detect the magnets (extra hassle, especially for visiting locos or rolling stock) and won't see, for example a long carriage parked over the sensor. For complex running, I need to know when the end of a train has passed (e.g. level crossing cleared) and also when a terminus has been reached regardless of whether the engine is pulling or pushing. Handling sensors in the logic is made easy because I have thrown away the concept of interrupts ("oh… sensor 5 has been detected… which loco was that and what the hell do I do now?") and instead have the route scripts work on the basis of "do nothing, maintain speed until sensor 5 triggers and then carry on in the script"

At present,   signals are connected direct to Arduino pins but in future (when the parts arrive from China) I will use the I2C bus as per the turnouts and sensors.

I have also added a simple relay switch that allows the code to switch the programming track onto the main track. This allows me to automatically detect the address of a loco on the programming track, then drive it onto the main track to join in the fun.

As I have not disabled the Locoduino/DCC++ text interface, you can still drive the system manually over the serial link.

OK … so how do I automate stuff?

The first thing to understand is the concept of a route… this is the steps required to get from A to B (but can also be the steps to handle an animation such as a level crossing, sludge farm or fairground)

These steps may be something like:

- Wait between 10 and 20 seconds for the guard to stop chatting up the girl in the ticket office.
- Move forward at speed 30
- When I get to sensor B stop.

Similarly the route from B to A could be something like this

- Wait 15 seconds for the tea trolley to be restocked
- Move backwards at speed 20
- When I get to A stop.

  Notice that the sensors at A and B are near the ends of the track (allowing for braking distance but don't care about train length)

  So this automation script could look like this:

```
BEGINROUTES
   SETLOCO(3)
    ROUTE(1)
       DELAYRANDOM(100,200)  // random wait between 10 and 20 seconds
       FWD(30)
       AT(2)        // sensor 2 is at the far end of platform B
       STOP
       DELAY(150)
       REV(20)
       AT(1)
       STOP
       FOLLOW(1)    // follows Route 1 again… forever
    ENDROUTES
```

The process starts at BEGINROUTES and in this case sets the loco address to 3 and drops through to Route(1) . If there are going to be multiple locos, it's a bit different as we will see.

Notice that the route instructions are followed in sequence by loco 3, the AT command just leaves the loco running until that sensor is detected. Although the above is trivial, the routes are designed to be independent of the loco address so that we can have several locos following the same route at the same time (not in the end to end example above!) perhaps passing each other or crossing over with trains on other routes.

The example above assumes that loco 3 is sitting at A and pointing in the right direction. A bit later I will show how to script an automatic process to take whatever loco is placed on the programming track and send it on it's way to join in the fun.

Now your Arduino script looks like this in the Arduino IDE:

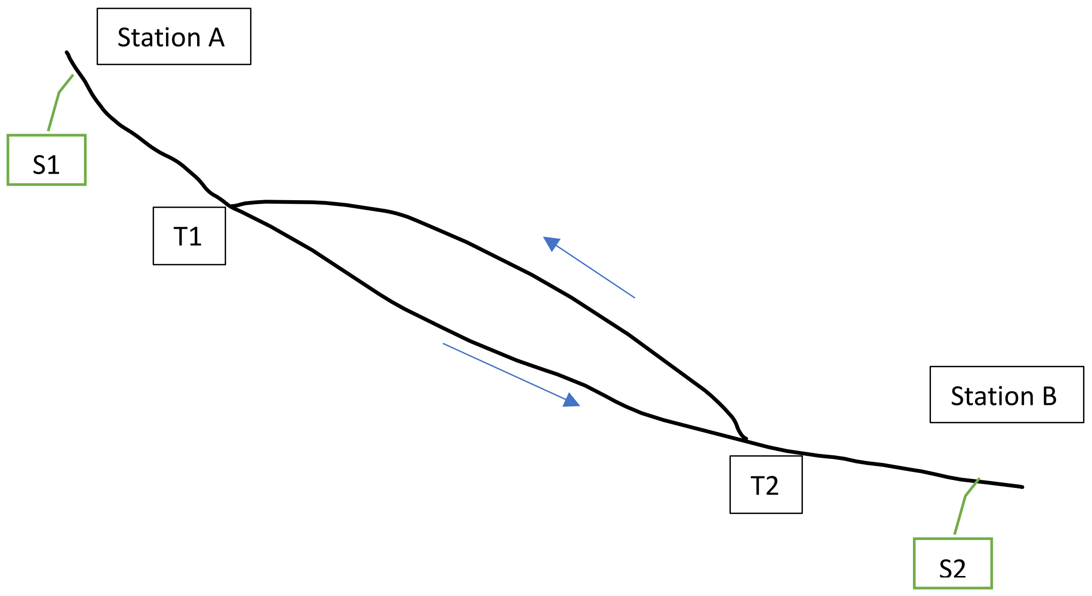
```
#include "TPL.h"
BEGINROUTES
  SETLOCO(3)
ROUTE(1)
  DELAYRANDOM(100,200)  // random wait between 10 and 20 seconds
  FWD(30)
  AT(2)         // sensor 2 is at the far end of platform B
  STOP
  DELAY(150)
  REV(20)
  AT(1)
  STOP
  FOLLOW(1)    // follows Route 1 again… forever

ENDROUTES

void setup(){
   Serial.begin(115200); // for diagnostics
   tplBegin(9,     // Arduino pin connected to prog track relay
            10,    // Number of contiguous sensor pins
            22,    // Arduino pin for signal zero
            8,     // Number of contiguous signals (2 pins each)
            16);   // Number of turnouts
  }

void loop() {
    tplLoop();
  }
```

OK, that was too easy, what about routes that cross (passing places etc) … lets add a passing place between A and B. S= sensors, T=Turnout number. So now our route looks like this:

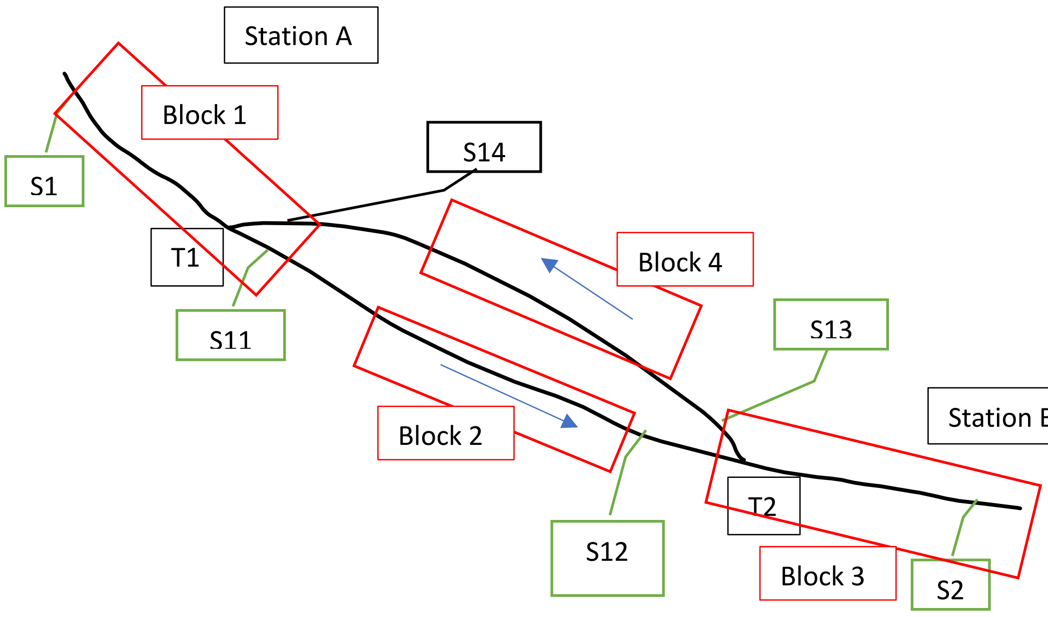


```
BEGINROUTES
   SETLOCO(3)
    ROUTE(1)
       DELAYRANDOM(100,200)  // random wait between 10 and 20 seconds
       TR(1)
       TL(2)
       FWD(30)
       AT(2)        // sensor 2 is at the far end of platform B
       STOP
       DELAY(150)
      TR(2)
      TL(1)
      REV(20)
       AT(1)
       STOP
       FOLLOW(1)    // follows Route 1 again… forever
    ENDROUTES
```

All well and good for 1 loco, but with 2 (or even 3) on this track we need some rules. The principle behind this is

- To enter a section of track that may be shared, you must RESERVE it. If you cant reserve it because another loco already has, then you will be stopped and the script will wait until such time as you can reserve it. When you leave a shared section you must free it.
- Each "section" is merely a logical concept, there are no electronic section breaks in the track.

So we will need some extra sensors (hardware required) and some logical blocks (all in the mind!):



We can use this map to plan routes, when we do so, it will be easier to imagine 4 separate routes, each passing from one block to the next. Then we can chain them together but also start from any block.

So… lets take a look at the routes now. For convenience I have used route numbers that help remind us what the route is for… any number up to 255 is Ok. Anyone want more than that and I will fix it.

```
BEGINROUTES
   … see later for startup
ROUTE(12) // From block 1 to block 2
  DELAYRANDOM(100,200)  // random wait between 10 and 20 seconds
  RESERVE(2)     // we wish to enter block 2… so wait for it
  TR(1)          // Now we "own" the block, set the turnout
  FWD(30)        // and proceed forward
  AFTER(11)      // Once we have reached AND passed sensor 11
  FREE(1)        // we no longer occupy block 1
  AT(12)         // When we get to sensor 12
  FOLLOW(23)     // follow route from block 2 to block 3

ROUTE(23)             // Travel from block 2 to block 3
  RESERVE(3)     // will STOP if block 3 occupied
  TL(2)          // Now we have the block, we can set turnouts
  FWD(20)        // we may or may not have stopped at the RESERVE
  AT(2)          // sensor 2 is at the far end of platform B
  STOP
  FREE(2)
  DELAY(150)
  FOLLOW(34)

ROUTE(34)  // you get the idea
  RERSERVE(4)
  TR(2)
  REV(20)
  AFTER(13)
  FREE(3)
  AT(14)
  FOLLOW(41)

ROUTE(41)
  RESERVE(1)
  TL(1)
  REV(20)
  AT(1)
  STOP
  FREE(4)
  FOLLOW(12)     // follows Route 1 again… forever

ENDROUTES
```

Does that look long? Worried about memory on your Arduino…. Well the script above takes just 70 BYTES of program memory and no dynamic.

If you follow this carefully, it allows for up to 3 trains at a time because one of them will always have somewhere to go. Notice that there is common theme to this…

- RESERVE where you want to go, if you are moving and the reserve fails, your loco will STOP and the reserve waits for the block to become available. (these waits and the manual WAITS do not block the Arduino process… DCC and the other locos continue to follow their routes)

- Set the points to enter the reserved area.. do this ASAP as you may be still moving towards them. (TPL knows if this is a panic and switches the points at full speed, if you are not moving then the switch is a more realistic sweep motion)
- Set any signals (see later)
- Move into the reserved area
- Reset your signal (see later)
- Free off your previous reserve as soon as you have fully left the block

Starting the system

Starting the system is tricky because we need to place the trains in a suitable position and set them off. We need to have a starting position for each loco and reserve the block(s) it needs to keep other trains from crashing into it.

For a known set of locos, the easy way is to define the startup process at the beginning of ROUTES , e.g. for two engines, one at each station

```
// ensure all blocks are reserved as if the locos had arrived there
RESERVE(1)  // start with a loco in block 1
RESERVE(3)  // and another in block 3
SENDLOCO(3,12) // send Loco DCC addr 3 on to route 12
SENDLOCO(17,34) // send loco DCC addr 17 to route 34
ENDPROG   // don't drop through to the first route
```

CAUTION: this isn't ready to handle locos randomly placed on the layout after a power down.

Some interesting points about the startup… You don't need to set turnouts because each route is setting them as required. Signals default to RED on powerup and get turned green when a route decides.

Startup can also SCHEDULE a "route" that is merely a decorative automation such as flashing lights or moving doors but has no loco attached to it. For example, using a signal connection to flash a red light on the pin for signal 7, green will turn it off!

```
ROUTE(66)
  RED(7)
  DELAY(15)
  GREEN(7)
  DELAY(15)
  FOLLOW(66)
ENDROUTES
```

Fancy Startup

With a small additional hardware relay, TPL can switch a track section between programming and mainline automatically.

Here for example is a startup route that has no predefined locos but allows locos to be added at station 1 while the system is in motion. Let's assume that the track section at Station1 is isolated and connected to the programming track power supply (via the relay). Also that we have a "launch" button connected where sensor 17 would be and an optional signal (ie 2 leds) on the control panel connected where signal 17 would be (see Signals below).

```
BEGINROUTES
     PROG_TRACK(0)  // start as program track connected to mainline

ROUTE(99)
     AFTER(17)      // user presses and releases launch button
     RESERVE(1)  // Wait until block free and keep others out
      PROG_TRACK(1) // power on the programming track
     GREEN(17)    // Show a green light to user
     // user places loco on track and presses "launch" again
     AFTER(17)
      READ_LOCO // identify the loco
      RED(17)   // show red light to user
      PROG_TRACK(0) // connect prog track to main
      SCHEDULE(12) // send loco off along route 12
      FOLLOW(99)   // keep doing this for another launch
```

The READ_LOCO reads the loco address and steps. For a small extra amount of C++ code I could make it detect a loco address clash and reprogram the loco to another unused address! By altering the script slightly and adding another sensor, it's possible to detect which way the loco sets off and switch the code logic to send it in the correct direction. (easily done with diesels!)

Signals

This is going to send the purists mad. Signals are now simply a decoration to be switched by the route process… they don't control anything.

GREEN(5) would turn signal 5 green and RED(5) would turn it red.

Currently TPL uses Arduino pins for signals and needs to know the first pin for signal zero. After that each signal uses two pins so if the signal zero pin is 40, then signal 0 uses pins 40-41, signal 1 uses 42-43 and so on. I guess we could do 3 lights etc if needed.

Sounds

You can use FON(n) and FOFF(n) to switch loco functions… eg sound horn

Numbers:

The code currently supports 64 Turnouts 64 sensors and signals according to the pins available. Its OK to use sensor numbers higher than the actual number of sensors connected. These can only be set, tested and reset in the scripts. If a sensor is set on by the script, it can only be set off by the script… so AT(5) SET(5) for example effectively latches the sensor 5 on when detected once.

You can give names to routes turnouts signals and sensors etc using #define statements.

Future plans

- Some of the constructs above are not yet in the code, or need cleaning up a bit. Its early days but world situation suggests I will have plenty of time on my hands.
- I want to add some more commands for controlling animations, such as SERVO, STEPPER and LED
- Issue messages on those fancy micro screens and on a status LCD

- Feel free to take a look at the code on https://github.com/Asbelos/TPL