The SANDSTONE & TERMITE RAILWAY
"Linking the Outback to the World"

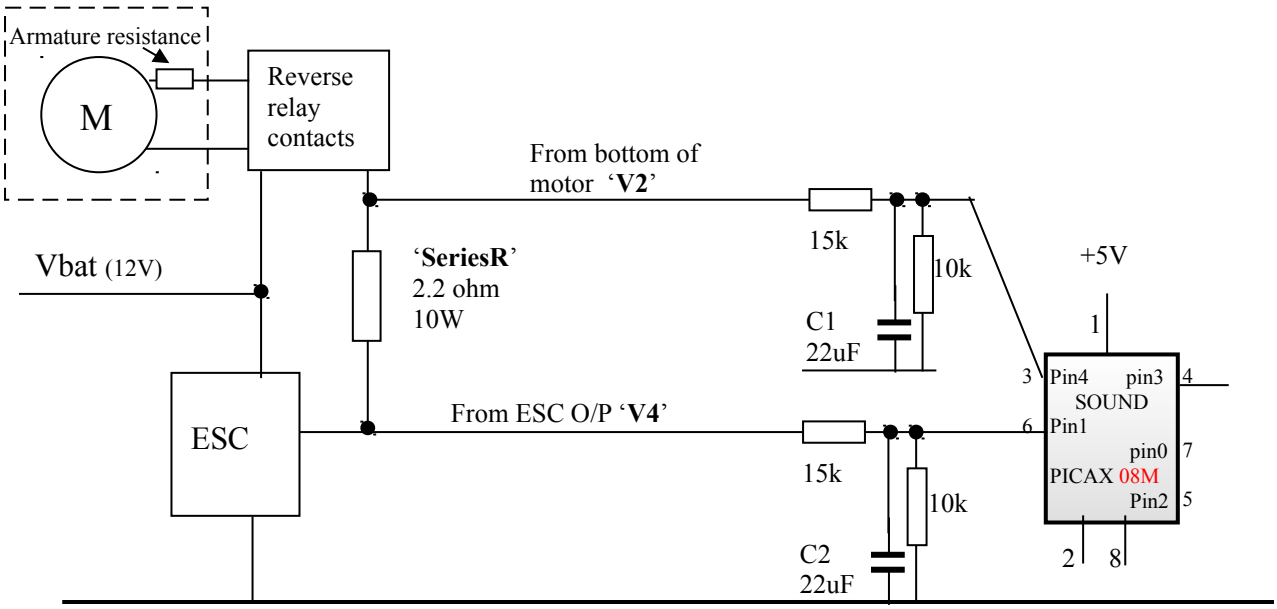
PICAXE TUTORIALS

6. Advanced Picaxe Motor Control

These are some thoughts on more advanced uses of Picaxe. I have not thoroughly tested all these and will not provide full programs. They are included to provide ideas to develop into you own programs.

Calculating 'True' armature voltage to determine speed

In the section on Sounds, we controlled the chuff rate using the motor voltage red in using READADC10 command.
But, the voltage across a motor is only an approximate indication of its speed. To calculate the true speed, we need to know the internal armature resistance of the motor, and the current in the motor. We can measure the motor current by putting a low resistance resistor in series with it and measuring the voltages on the 'top' and bottom' of this resistor using a Picaxe. This circuit shows the connections. Once the true speed voltage is calculated, it can be used to control the chuff rate as previously shown in *Tutorial 2* or to provide feedback control.



Then the Picaxe program has to calculate the voltage from:
armatureV=Vbat-V2-{ (V2-V4)*armatureR/seriesR}

(the internal armature resistance must be determined. A multimeter can give an approximate value. For G scale, 4 ohms is a typical value.)

NOTE that we need to know the battery voltage for this calculation – and it varies!!
The battery voltage is equal to V2 whenever V2 and V4 are equal. ie current =0, loco is stopped, but we can not easily measure it when the loco is moving.

The only problem is implementing the above equation with the limited maths capability of the Picaxe and the need to use integers only.
With Vbat around 12V and input resistors of 10k and 15k in circuit above, Vbat will be around 900 to 1000 counts in the program. V2 will be from 1000 counts (stopped) to around 200 counts at max current of 1amp. V4 will be in the range 1000 (stopped) down to zero counts at max speed.

A possible program flow might look like this: (assuming we've defined the symbols)

determine Vbat when loco is stopped
read in V2 and V4

armatureV=Vbat-V2 MIN 0 (an intermediate value in the calculation)
V1=V2-V4 (an intermediate variable) (value from zero when stopped, to about 40 when taking little current, to 200 at max current)
V1=V1*armatureR/seriesR
Because armatureR and seriesR are decimal numbers, we need to store them in the program as say 10 times their value.

Finally…
armatureV=armatureV-V1 measured in 'ADC counts'

An even more accurate equation takes the motor losses into account. Try measuring the loco current with its wheels in the air (Iloss). This represents the friction losses. (typical value G scale is 350mA)
Then use this equation:

armatureV=Vbat-V2-{ (V2-V4)*armatureR/seriesR} + Iloss*armatureR

Iloss*armatureR is a constant and can be entered as such. But Iloss has to entered as 'counts' equivalent to the voltage in the program. As the input voltages are reduced to 0.4 by the 10k & 15k voltage divider, we have to multiply the Iloss by 0.4 when entering it. So if Iloss is 350mA, enter it in the program as 0.35*0.4/5*1024=28 counts; and multiply by actual armatureR in ohms.
So the value of Iloss (350mA)*armatureR (4ohms) will be entered as 112 counts
Then we modify the above program in the final line to read:-
armatureV= armatureV-V1 + 112 measured in ADC counts.

I have never tried using this equation.

Compensating for battery voltage change

In both the above examples the calculation for speed depends on knowing the battery voltage.

The above program uses the fact that when V2=V4 there is no current, so the voltages V2 and V4 represent the battery voltage (at no load). It's not possible to determine the voltage when the loco is taking current – we need to know the battery voltage to calculate its speed!
It is possible to determine stopped battery voltage with just one analogue input (V4) but it is tricky. Again, when the loco is taking no current, the read-in voltage will be battery voltage, BUT how to determine if the loco is stopped? It's not possible to do it reliably using the motor ESC voltage as

the input. If we have defined 'stopped' as 'less than some speed', using the battery voltage in the calculation for speed, we are 'chasing our tail' and the program becomes unstable.

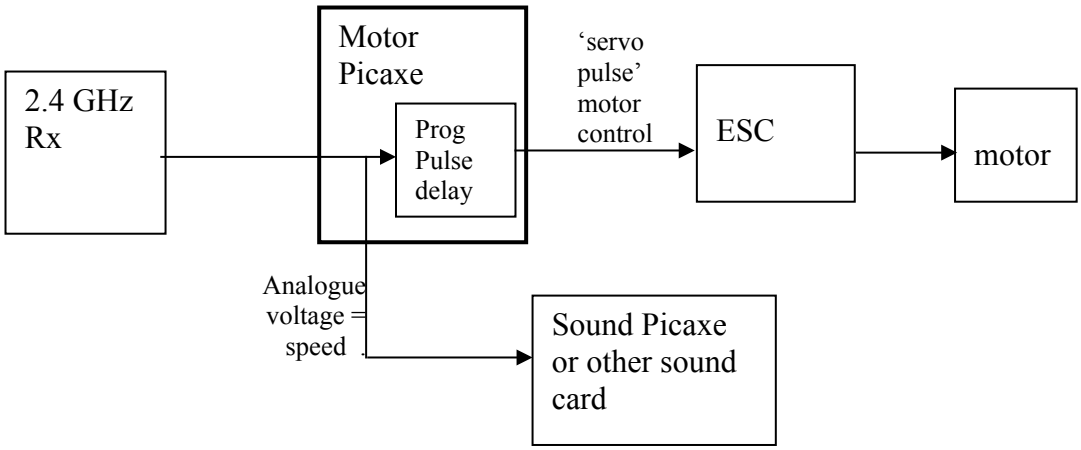
One solution is just to measure the voltage as soon as power is turned on. We KNOW the loco is stopped, so it's correct. But during running, the battery voltage will drop and the first value will become incorrect. Still, it's better than nothing, and turning the loco off and on again will get us a new value.
Another way is just to use a separate analogue input if you have one spare. Then you can monitor the voltage all the time, and even implement a 'low voltage' alarm.

Reading in servo pulses and controlling speed

In Tutorial 4 we saw how to read in the throttle pulses from a radio control receiver. You could use a servo signal in, to produce a PWM or DC voltage out – in effect a motor ESC. In this case you would read the input signal and pass it straight to a PWMOUT command. You might want to do this if you're using a higher battery voltage than the ESC can handle. Or you may want to delay the signal.

Delaying servo pulses

In some diesel locos I have delayed the motor voltage between the 2.4 GHz receiver and the motor, but used an **UN**delayed voltage to control the diesel sound. This gives the effect of the diesel engine revving up before the loco starts to move. Here's the block diagram:-



I use 3 variables as a 'shift register' to shift the servo/ESC pulses through. A few hundred milliseconds seems good. The output to the ESC uses the PWMOUT command on pin2, so this cannot be used for the analogue speed voltage. Instead I use the 'old' command PWM. This has to be 'refreshed' at intervals and feeds out to a RC circuit to filter the PWM and produce an analogue voltage.
As well as the delay feature, I also use an inertia feature when starting. But I found inertia is a bit of a drawback when trying to stop at a given spot, so the inertia feature is disabled when slowing.

Appendix: Byte usage in these example programs

word	byte	Steam chuff	diesel	Motor control
w0	b0		Seed (shift register)	Timecounts (pulsin)
	b1			
w1	b2	Speed input voltage	Speed input voltage	PWMspeed output
	b3			
w2	b4	chufftime	pausetime	
	b5	chuffcounter	Lookuptable ?	
w3	b6	offtime (ms)		
	b7			
w4	b8	soundpin	soundpin	
	b9	Temporary in cutoff calcs		
w5	b10	oldspeed1	oldspeed1	
	b11	oldspeed2	oldspeed2	
w6	b12	PWMvolume (PWM 15kHz)	PWMvolume (PWM 15kHz)	
	b13			