Advanced Traction Control Turorial V1.8

Overview

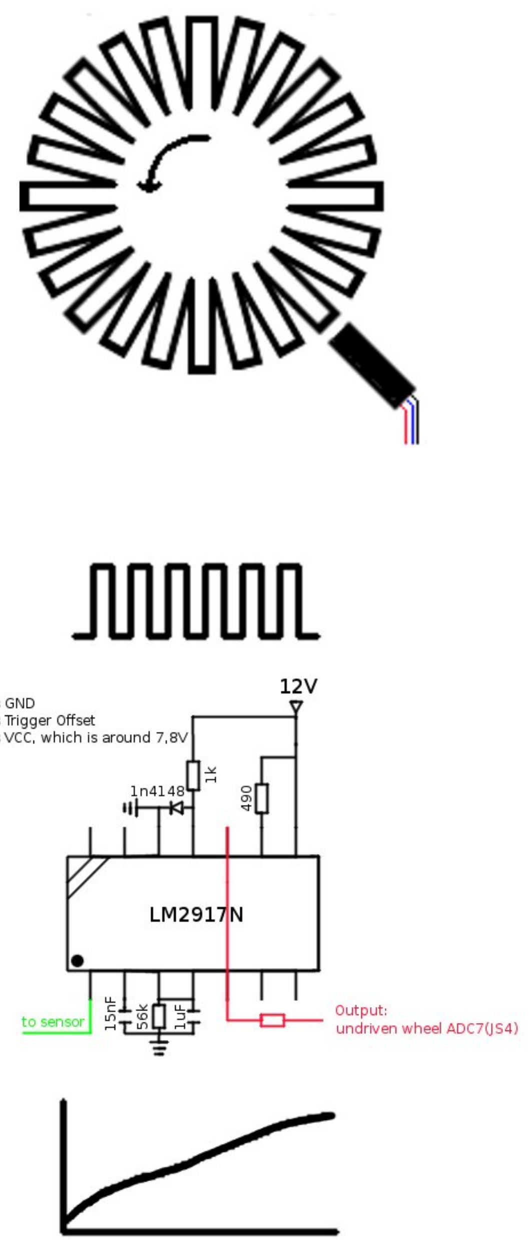
Test Vehicle:	FS Car of 2008
Weight:	250kg
Power:	≅75HP
Tires:	Front 6"
	Rear 7,5"

Test Vehicle:	FS Car of 2009
Weight:	225kg
Power:	82HP
Tires(accel):	Front 6"
	Rear 7,5"
Tires(normal):	Front 7"
	Rear 7"

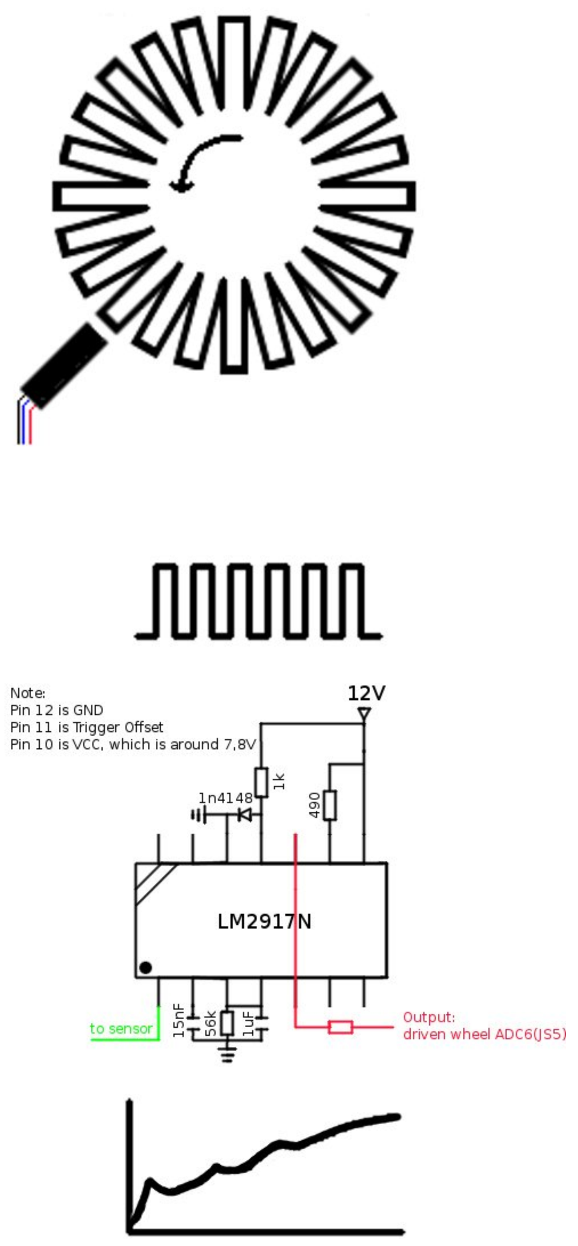
©Dipl.-Ing. Patrick Menschel
Email: Menschel.P@posteo.de

2008 FS Car Layout

Front



Rear



©Dipl.-Ing. Patrick Menschel
Email: Menschel.P@posteo.de

Changelog

23. April 2009	Initial Document
19. July 2009	Swap to Fuzzy Assembly
22. July 2009	Added Fuel Cut Option
23. July 2009	Code Section removed as it just keeps confusing the reader.
28. July 2009	Outline/Summary added, Clean Up
01. Aug 2009	Added Spark Cut Option
01. Sept 2009	2009 FS Car, Controller Tuning Section added
08. Sept 2009	2009 FS Car Circuit instead of 2008, Tuning Section Clean Up
20. Sept 2009	Integration with Launch
16. Dec 2009	Clean Up, new Options figure
17. Dec 2009	Datalogs added, draft status removed
15. Jan 2010	Author's Note, Benchmark,GPL Compliance added
29. Jan 2010	Circuit Calc Cleanup as 2008/9 FS Car were mixed up
16. Feb 2010	Handshake with other spark/retard controller modes implemented
19. Feb 2010	Cut ties with Megasquirt 2 Extra Project
20. Feb 2010	Last and now unofficial Release of AdvTC within Megasquirt
27. Feb 2010	Code Section Easy Version added
28. Feb 2010	Final Document
12. May 2017	Updated contact information

Inhaltsverzeichnis

©Dipl.-Ing. Patrick Menschel
Email: Menschel.P@posteo.de

1 Basics of Traction Control

1.1 Physical Basics

Traction, Slip, those are empty words without a proper definition, so we'll discuss the physics behind them for the record.

Traction is „what keeps you on the road", more specifically there's the friction coefficient called µ which is multiplied with gravity, weight and tire surface to get the actual force of friction.
µ depends on a whole bunch of things, material combinations, rubber mixture, surface conditions, temperature,..., the higher it is, the better for you.
As common to all of nature there's an equilibrium of forces, means there's a counterforce, usually a sum of certain forces that deform your tire as it moves on the road combined with the torque from the powertrain, not to mention lateral gravity while cornering.

Slip is not exactly a counterpart to Traction as many "experts" claim. A better description is that slip is a reaction to or a function of traction, actio is followed by reactio. Since it is a function, it can be controled by influencing the input.
Mathmatically, Slip is the fraction/percentage your wheel moves faster, be it ether rotation or lateral movement, than what friction would be able to compensate for.
If there's no lateral movement, you can say slip is the percentage that the driven wheel rotates faster than the undriven wheel.

$$Slip = \frac{RPM_{drivenWheel} - RPM_{undrivenWheel}}{RPM_{undrivenWheel}}$$

Note: This is the most convenient way to define slip for automotive applications.

1.2 What is optimal? Is there a constant?

Way through different studies and papers there's been a couple of "experts" claiming that a slip of 0,2 or 20% is optimal for tarmac, 0% for ice, and so on.
With that knowledge, you would be able to tweak a traction control for that specific surface but it won't work on a different one.
Should mean, all of those laboratory results are of no actual use as the road/surface/material/fluids/... you drive on:

- changes from one meter to the next (dynamic),
- which is not predictable (non-linear),
- changes based on weather/temperature and other things you can't influence at all.

Additionally it even depends on the rubber mixture of your tires, it's profile and even the weight distribution of your vehicle. To sum it up, that's a serious problem since that tire/road system that you want to regulate somehow is changing by chance, so there's no mathmatical way to calculate what will happen next as PID controllers rely on.
The problem is non-trivial and classical engineering can't provide a sufficient solution.
Fortunately Computer Science / Informatics came up with their own field of controller theory decades ago and provide the key to solve this problem, that is **Fuzzy Logic**.
If you loose traction while accelerating, you're smart enought to reduce throttle until slip reduces. More precisely you feel the car slip, start to determine the state that it slips and react on it, so the key part is to determine the car's state and have a rule how to act in that case.

So far, so good. We have a couple of states and rules for that but a machine can't actually use those since most of human made states/rules overlap, are partly true, partly false and may contradict each other. So we'll start to determine/weight the value of truth and enter the world of fuzzy computing.

1.3 Basics on Fuzzy Logic

Most people claim they don't understand fuzzy even one bit. That's funny since the human being is a spendid example for fuzzy logic.
The all world example:
There's a glass of beer on the table, neither full nor empty, some would say half full, others would judge it to be half empty and their reaction is even more curious. Some would take the bottle next to it and fill it up again, others would empty it right away. This is a fuzzy controller. People judge the state depending on membership functions they use unconsiously.
„Yep, that glass is definetly more empty than full.", „What to do with an empty glass as it is not fully empty?", „Enjoy the last sip." or „Fill it up as a full glass of beer is more pleasing to the eye than an empty one."
Even the reaction is weighted and what people do is to sum up all of the unconciously weighted stuff in their head and start to act depending on their individual output. Sure way, you can do that but that stupid computer does only know 0 and 1, false or true.
Not completely correct, no one ever said that you have to determine a state with only one bit. You can use a whole byte (8bit) to define a state of truth where 255 would be fully true and 0 would be not true at all. So you need a couple of membership functions that determine the truth value for each input value.
Empty(Beer) = 50, Full(Beer) = 205 , and so on. That's the fuzzyfication step, to assign your states a truth value. Now for the rules. Empty consequents to fill up, Full consequents to empty it. Since we only have two rules/ one variable we skip the rule evaluation here and directly move to the defuzzification step.
We're still in the fuzzy realm, think of drink as negative and fill as positive effect on the level of beer, so we assign drink a negative value (-50) and fill a positive (50). These are the fuzzy singletons/possible reactions. Now multiply the truth of each fuzzy output with their singleton value, sum up and divide it by the overall truth value to get the real Output.

$$Output = \frac{205 \times -50 + 50 \times 50}{255} = -30$$

You will drink some more. Yes, that's bad for your health so we'll change the singletons a little to make you drink less.

$$Output = \frac{205 \times -50 + 50 \times 100}{255} = -20$$

Yeah, as we increase the singleton for fill up, you drink less. Correct, you will drink it anyways sometime. But we can also change the membership functions for full and empty to influence on the truth values instead of the singletons. We may build you a pyramid glass that has most of it's volume at bottom means the glass always looks sort of empty, you will keep on filling it and presumably drink less.
If you're not completely dizzy/drunk by now, I assume you have learned the basics of fuzzy.
Note: You can use even more than 8 bits to determine the truth value and raise precission respectively.

2 A fuzzy Traction Controller

2.1 Input Variables

To adjust a logic based controler you need to know your key variables that are traction and slip.
The easiest way is to use wheel rpm sensors. Any undriven wheel will provide a valid pickup for Traction as you can assume that it is “driven“ by the friction from the corresponding surface. That gives you µ. For Slip you need both the driven wheel and the undriven wheel (that you already have). So install an rpm pickup on a driven wheel and calculate slip as per given formula. Now you have slip or lambda how it’s called in certain papers. Now you only need their derivates deltamu and deltalambda. On microcontroler systems this is commonly done by a simple subtraction from the last known value and the current value within a known loop time. See the source code for that.

2.2 Membership Functions

We need 4 Membership functions for starters. We want to determine if deltamu or deltalambda are positive or negative. So we’re using triangular membership functions that start at one end of a specific area and constantly get more true as they reach 255 which is the highest truth value in 8bit fuzzy. Take a look in the code how it’s actually done and laugh to your hearts content how easy it is. ;)
Now we have fuzzy inputs deltalambdanegative, deltalambdapositive, deltamunegative and deltamupositive.

2.3 Fuzzy Rules

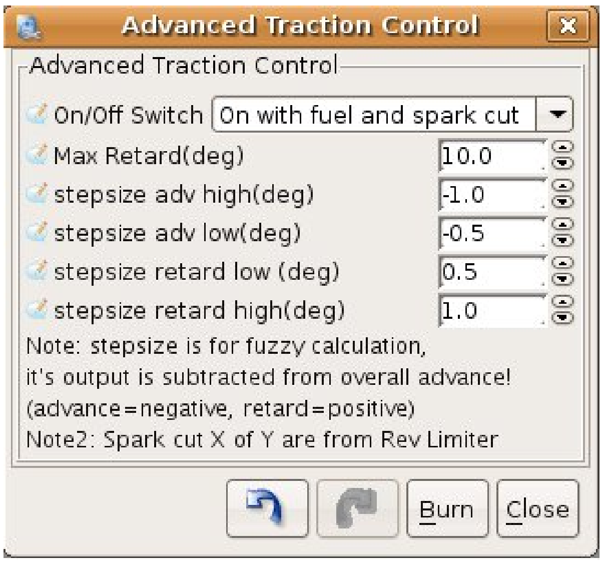
These rules were inspired by an IEEE paper from the early ’90s I read somewhere, didn’t find it anymore ;P . They used brake torque as actuator that is the most stupid idea from my POV. We use the spark angle instead as a change in spark angle influences engine torque without any noticable time delay thus a very powerful actuator. You can drive at Wide Open Throttle while controling engine output with the ECU. Now for the rule evaluation.

- **if** deltamu < 0 **and** deltalambda < 0 **then** advance spark angle very much
  (there is enough traction for more engine power)
- **if** deltamu > 0 **and** deltalambda < 0 **then** advance spark angle a little
  (gaining traction, so add some power)
- **if** deltamu < 0 **and** deltalambda > 0 **then** retard spark angle very much
  (already lost traction, cut power immediately)
- **if** deltamu > 0 **and** deltalambda > 0 **then** retard spark angle a little
  (loosing traction, decrease power)

These rules sum up as follows. Slip is regulated to the point where slip may occur but the vehicle still accelerates. That’s the key part of the controller, peak slip targetting/estimation. The beautiful aspect is that you don’t have to know what slip is optimal for that particular surface and since the controller doesn’t depend on real figures, it does not care about any change of surface.
Note: Mathmatically the „and“ of different truth values is a min-operation on both that is supported by the HCS12 instruction set.

2.4 Safety Features, Fuzzy Singletons and Output Calculation

The first aspect of every controler is to limit the output, the value that is subtracted from spark advance. Assume 10 degrees retard are enough to reach a powerfull reaction on torque, so limit output to be a positive number from 0 to 100 (degree with one decimal) for starters. This way you can garantee it's neither adding something on advance and blow your engine that way, nor moving advance into the ATDC region. Additionally if the driven wheel rpm is less than or equal undriven wheel rpm, the controller output is set to 0, as there is no slip thus no regulation required. This is feasible as a reset state, so every occurance of slip has the same start conditions.
The fuzzy singletons are the output values for each fuzzy output (if that output would be fully true), a step size for given conditions. Retard is positive, Advance is negative due to the fact that we subtract that value from overall advance at the end of the controller loop. They are limited in Megatune/Tunerstudio, so you should not be able to insert wrong signed values here. But take care the low value is always less than the high value, should be logic. As an option, fuel or/and spark may be cut if the controller is at max retard and still wants increase retard. This is a safety feature for very low µ conditions. Slip should fall back into the region where spark advance can control slip again. It's not intended as an actuator!
Take a look at figure 1.



Advanced Traction Control

On/Off Switch	On with fuel and spark cut
Max Retard(deg)	10.0
stepsize adv high(deg)	-1.0
stepsize adv low(deg)	-0.5
stepsize retard low (deg)	0.5
stepsize retard high(deg)	1.0

Note: stepsize is for fuzzy calculation,
it's output is subtracted from overall advance!
(advance=negative, retard=positive)
Note2: Spark cut X of Y are from Rev Limiter

Abbildung 1: Settings Dialog

2.5 A more or less true Graph

OK, here's a graph (figure 2) that illustrates the controller output, well actually it illustrates what comes out of the rule evaluation. Keep in mind that the truth values of all fuzzy outputs are multiplied with their respective singleton, get summed up and divided by the overall truth to get the real output.

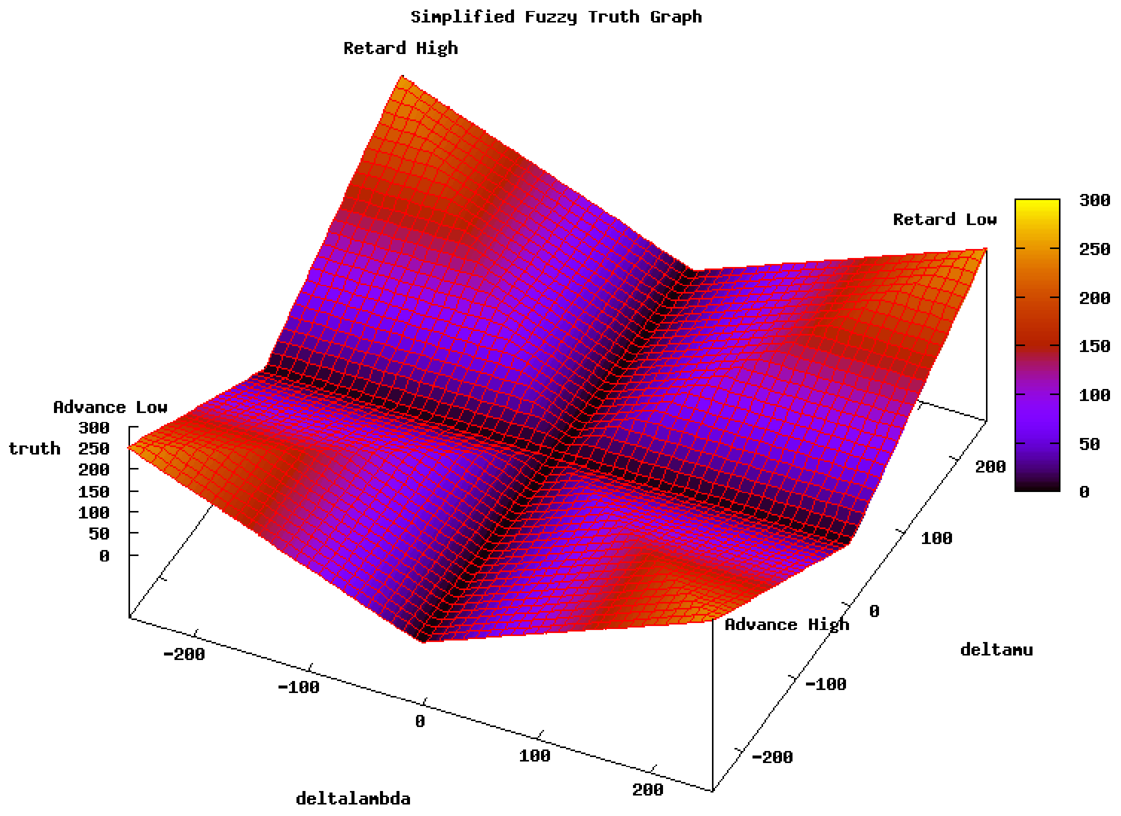


Abbildung 2: Fuzzy Graph

©Dipl.-Ing. Patrick Menschel
Email: Menschel.P@posteo.de

2.6 A Datalog

At first, let's look at Acceleration (figure 4). Driven wheel aka egt6temp immediately slips and is countered by controller output aka status4 until it stabilizes. As the undriven wheel aka egt7temp accelerates, controller output decreases and launch is complete. The graph speaks for itself.

The other point is skid pad aka cornering. Naturally the car reacts more sensitive to slip while cornering and will spin out if the reaction is too slow.
Additionally the front usually drives on a slightly larger radius than the rear, so we can't prevent slip before it occurs, thus our goal is not peak slip targetting but to counter slip as far as possible while cornering using very high step sizes. In vehicle dynamics this should force a state where the steering is close to neutral and cornering is done by rear slip. Figure 5 shows a few light steps on the throttle while cornering with acceleration settings. The reaction comes quite quick but it won't stop the car from spinning out at WOT.

Abbildung 3: Track Day (-4 deg Celsius, little amounts of snow and ice on track)

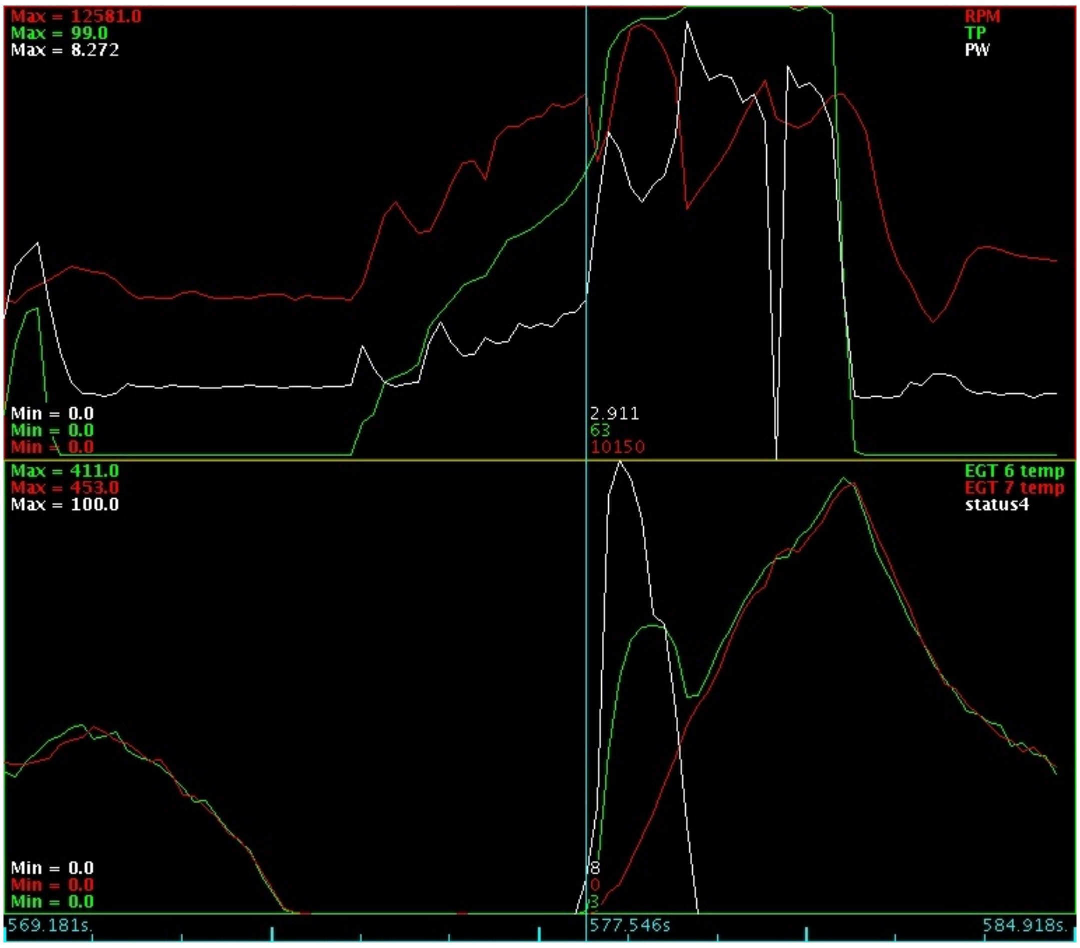


Abbildung 4: Acceleration Trial

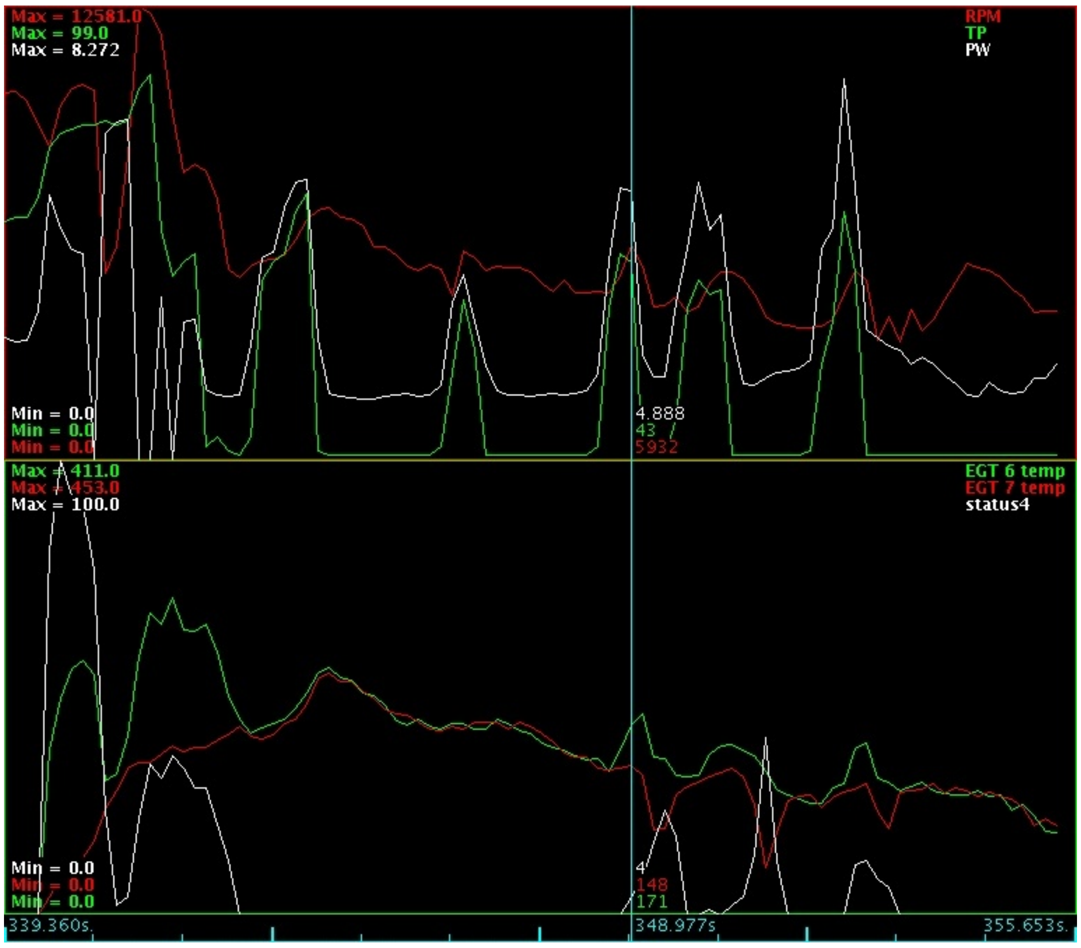


Abbildung 5: Skidpad Trial

Note: There's a sudden loss of traction approximately 7m after start due to a snow field.

3 Sensors, Wiring and Circuits

3.1 Signal Specification

As MS2 is a LQFP 48 package with limited IO Pins where all Timer Channels are already in use, the most appropreate way for signal input are the spare ADC Channels AD6(JS5) and AD7(JS4). So we need an input voltage (0-5V) based on wheel RPM, and the common chip for frequency to voltage is a LM2917N (charge pump circuit). Means neither the idea is new nor the circuit, it was already on the MS1 Sourceforge pages, labeled as experimental and vanished somewhere on a dump.

3.2 Sensors and Trigger Wheel

With the 2009 FS Car we use hall sensors in combination with a 12 tooth rotor, the inner nut of the wheel bearing to be exactly. The car's top speed is about 170km/h, it's 1,64m per Wheel Revolution, means the wheel rotates about 30 times per second, results in a frequency of about 360Hz, say 400Hz including safety.

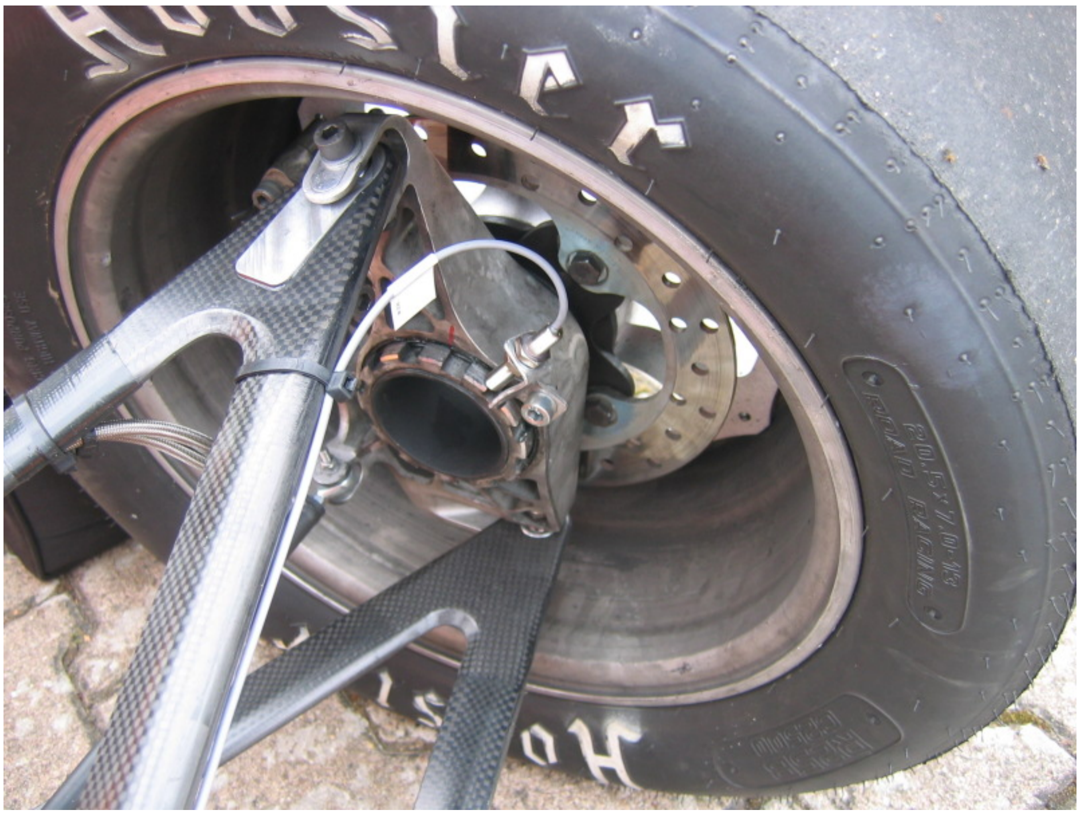


Abbildung 6: RPM Sensor Front

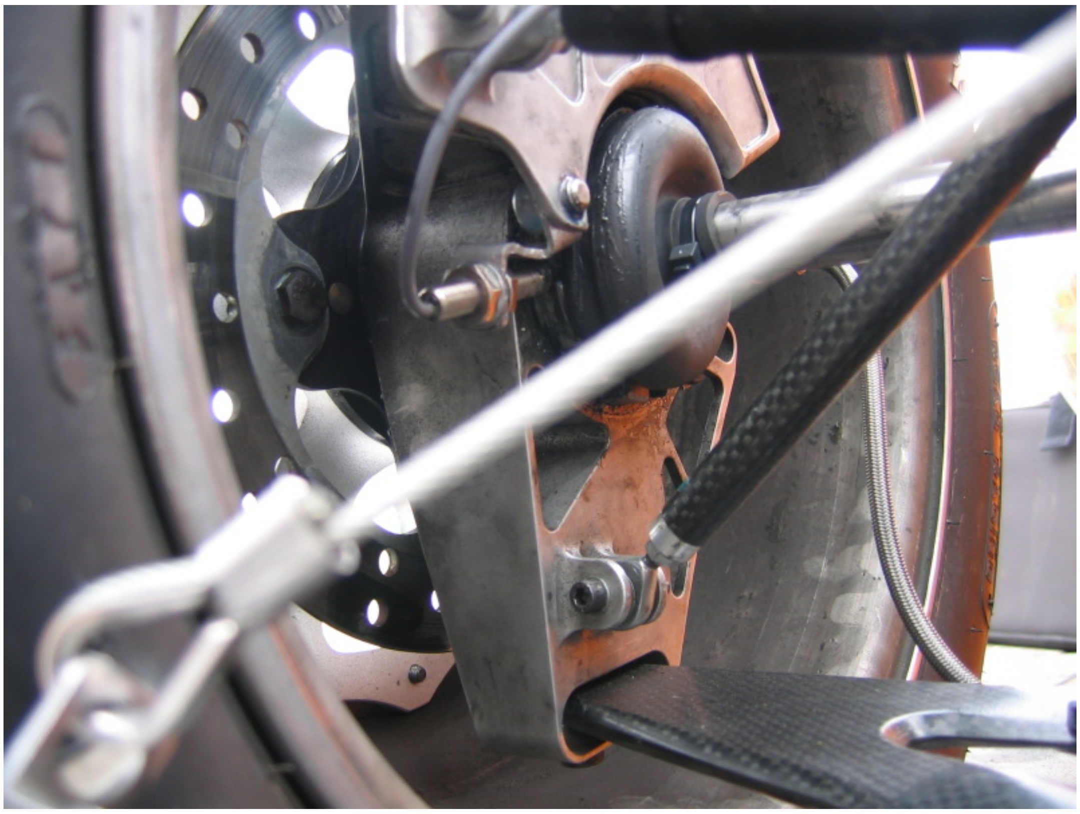


Abbildung 7: RPM Sensor Rear

3.3 Circuit Calculation

Easy! The datasheet has plenty of examples in it, except for the most crutial information that you need a DIP14 component for hall sensors, since Pin 11 is the trigger offset, that must be some low but positive Voltage to provide the internal positive/negative crossing that triggers the charge pump. Means a 1N4148 diode and a 10K resistor will do the job on Pin 11 except you have one of those nasty 0,85V Hall Sensors ;P . In that case use a resistor instead to raise trigger offset over the voltage level that the hall sensor has when active.
Calculating the charge capacitor on Pin 2 and the flow resistor on Pin 3 can be done by the formula:

$$OutputVoltage = InputFrequency*SupplyVoltage(Pin9)*FlowResistor*ChargeCapacitor$$

$$7,8V * 360Hz * 0,000000015Farad * 100000Ohm = 4,21V$$

In our case 7,8V on VCC (due to the 490Ohm Zener Regulation, 12V before that), 15nF Capacitor, 100kOhm Resistor. Results in about 4,2V at Top Speed. Some Air upwards, good scaling.
Note: There's another bad joke within the datasheet, VCC is PIN9, not the VCC they use in their Example Circuits. Also note that the decoupler on the output voltage(pin 5) is for signal dampening, prevent spikes.

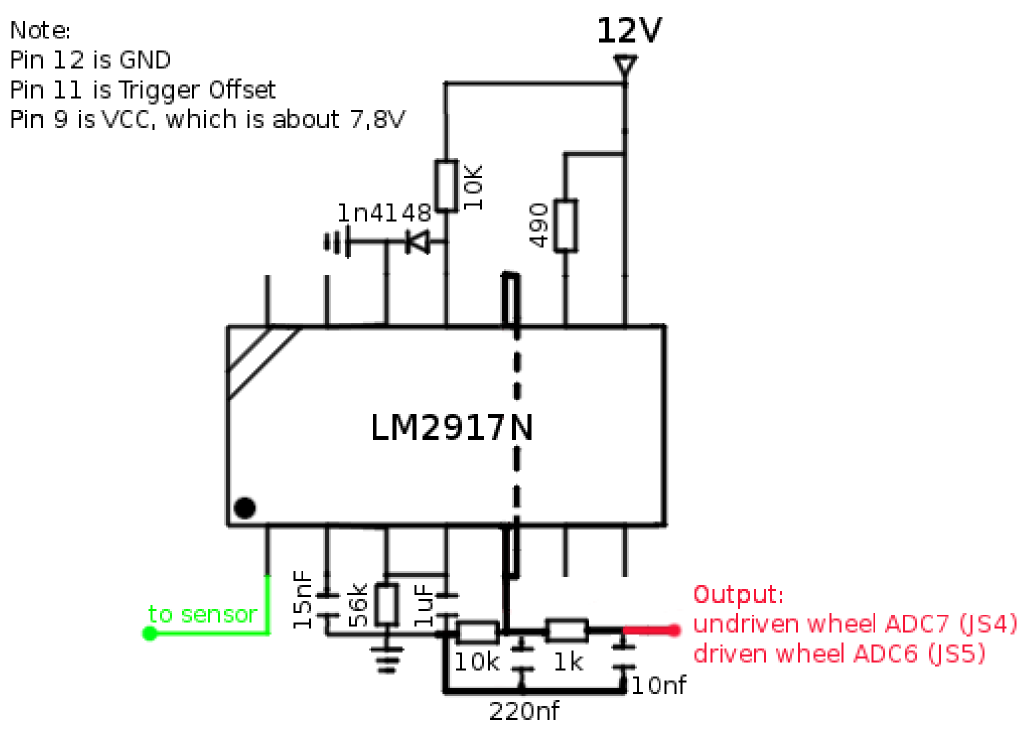


Abbildung 8: LM2917N Circuit Schema from 2009 FS Car

4 Code

This is an early Code Version partly in C and Assembly, assumed to be more easy to understand than the stack only assembly code that came out in the end.

Listing 1: Active Control in Main Loop

```c
if (flash8.advTcOn){//only act on enabled
  DISABLE_INTERRUPTS
  ultmp = lmms;
  ENABLE_INTERRUPTS
  //get time for update cycle
  if (ultmp > advTcUpdateClk){//update
    advTcUpdateClk = ultmp + 78;
    // 10ms ( 1s = 7812 x .128 ms) clk
    advtc_fuzzy(); //see advtc.s
  }//end update loop
  outpc.status4 = advTcRetardValue;//log output
  lsum -= advTcRetardValue;
}//end advTc loop
```

Listing 2: Fuzzy Controler in Assembly

```asm
advtc_fuzzy:
;calculate current slip here
   ldd drivenwheeladc
   subd undrivenwheeladc
   lble advtc_no_tc ;brach to the end
   ; branch if no slip

   pshd ;increase stack pointer by 2,write d

;calculate change of undrivenwheel aka deltamu
   ldd undrivenwheeladc
   std 15,SP ;tmp1 buffer
   subd advTcLastundrivenAdc

;limit to 8bit signed
   cpd #0xff80 ; -128
   bge advtc_lt
   ldd #0xff80 ; -128
   bra advtc_lt2

advtc_lt:
    cpd #0x7f; 127
    ble advtc_lt2
         ldd #0x7f; 127

advtc_lt2:
```

```
;d is now 8bit signed range
;save undriven adc for next loop
    movw 15,SP , advTcLastundrivenAdc
;calculate the corresponding fuzzyinputs deltamupos/neg

   clra ;discard high byte

   pshb ; push low byte on stack
   ldx #fuzzyinputoutput ;set pointer
   ldab #0x7f ; 127
   subb 0,SP ; b on stack
   stab 0,X ; deltamunegative!

   pulb
   addb #0x80 ; 128
   stab 1,X ; deltamupositive

;calculate change of slip in percentage*10  aka deltalambda
   puld; already have slip pullfrom stack,decrease SP by 2
   subd #1 ; prevent reach of slip state when 0 Speed
   ldy #1000
   emul
   ldx 13,SP ; tmp1 adc7 buffer
   inx ;never divide by 0
   ediv
   tfr Y,D
;limit output to 16bit signed, we know its positive
   cpd #0x7fff ; 32767
   ble advtc_lt3
   ldd #0x7fff ; 32767
   ;slip calculation and limit verified working perfect
advtc_lt3:
;buffer that value for a moment

   std 13,SP ;tmp1
;limit output to 8bit signed
   subd advTcLastSlipValue
   cpd #0xff80 ; -128
   bge advtc_lt4
   ldd #0xff80 ; -128
   bra advtc_lt5
advtc_lt4:
   cpd #0x7f; 127
   ble advtc_lt5
   ldd #0x7f; 127
advtc_lt5:
```

```
;now store advTcLastSlipValue for next loop
   movw 13,SP , advTcLastSlipValue

;calculate the corresponding fuzzyinputs deltalambdapos/neg
   ldx #fuzzyinputoutput ;set pointer
   pshb
   ldab #0x7f ; 127
   subb 0,SP
   stab 2,X ; deltalambdanegative

   pulb
   addb #0x80 ; 128
   stab 3,X ;deltalambdapositive


FUZZYEVAL:
; rule evaluation here
   ldx #fuzzyinputoutput
   clrb
   ldaa 2,X
   mina 0,X
   staa 4,X
   ; rule1 if deltamunegative and deltalambdanegative >> advance much higher

   ldaa 3,X
   mina 0,X
   staa 7,X
   ; rule3 if deltamunegative and deltalambdapositive >> retard much lower

   ldaa 2,X
   mina 1,X
   staa 5,X
   ; rule2 if deltamupositive and deltalambdanegative >> advance some higher

   ldaa 3,X
   mina 1,X
   staa 6,X
   ; rule4 if deltamupositive and deltalambdapositive >> retard some lower


;now do defuzzification aka true output step calculation
;first init variables
   ldd #0
   std 13,SP ;tmp1 is multiplied truth
   std 17,SP ;tmp2 is overall truth

   ldab #4 ; ctmp1 loop variable
;now set pointers
   ldx #fuzzyinputoutput+0x4
   ldy #flash9.advTcFuzzySingletons


;*************************************************************
;anything above here is verified working flawless

advtc_defuz:
;loop, store loop variable
   stab 35,SP ;ctmp1

;now get singleton
   ldab 0,Y
;signed extent to 16bit
   sex B,D
;move to Y
   sty 10,SP
   ;utmp1 buffer pointer to Singletons, need another index register
   tfr D,Y

;get output truth value
   clra ;cleanup
   ldab 0,X ;load output truth

;add to overall truth
   addd 17,SP
   std 17,SP

;get output truth value again
   clra
   ldab 0,X

   emuls ;signed multiply d*y
   addd 13,SP
   std 13,SP
;add to multiplied truth

   ldy 10,SP ;restore pointer
   inx ; increase both pointers
   iny

   clra ;cleanup
   ldab 35,SP

   dbne B,advtc_defuz
;**************************************************************
;load loop variable, autodec and branch back to start if not 0
;verified to loop 4 times which is correct
   ldd 13,SP
   ldx 17,SP
   idivs
;compute real output
   tfr X,D
;add this step to overall output
   clra
   addb advTcRetardValue
;output limitation 0
   cmpb #0
   bge advtc_lt6
   movb #0, advTcRetardValue
   bra advtc_end

advtc_lt6:
;output limitation advTcMaxRetard
   cmpb flash9.advTcMaxRetard
   blt advtc_lt7 ;was ble previously
   movb flash9.advTcMaxRetard, advTcRetardValue
   brclr flash9.advTcOn,0x06,advtc_end
   bset outpc.status3,#0x04 ; status3_traction_active, toggle fuel cut
   bra advtc_end

advtc_lt7:
;didnt touch the borders
   stab advTcRetardValue
   brclr flash9.advTcOn,0x06,advtc_end
   bclr outpc.status3,#0x04 ; status3_traction_active, restore fuel
   bra advtc_end

advtc_no_tc:
;branch for no slip reset runtime variables here, keep record of undriven wheel

   clr advTcRetardValue ;saves a byte
   movw #0, advTcLastSlipValue
   movw undrivenwheeladc , advTcLastundrivenAdc

advtc_end:
rts
```

5 Controller Tuning

5.1 Hey, why does it need tuning, didn’t you tell otherwise before?

Although the controller concept is flawless in its basics, there are feasible things to change with individual installations to squeeze out the last milliseconds. So we tend to an example from our 2009 FS Car. As 600cc have very limited torque in relation to the tire surface, the tires’ grip slows the engine down to rpm ranges that have little torque, a serious problem with the 2008 car that had absolutely nothing below 8500rpm. Fortunately the 2009 engine with customized cams has its maximum torque with 63Nm at 7000rpm, so this effect is less important. Thus our goal is to produce more slip to counter the slow down of the engine which would cause us loss of time to recover to max torque. During acceleration trials we achieved good times but the driver complained “ It feels wrong, I might be faster with a little more slip“. Therefore we want to shift the controller balance towards advancing spark/adding torque.

5.2 Aggressive Controller Strategy

The first thing to understand is that for peak slip targetting during acceleration, the step sizes for retard low and advance low are active. If we decrease retard low and increase advance low, we reach the effect that retarding when slip increases becomes weak and advancing when slip decreases becomes strong and in neutral state, controller output has a (fast) decreasing attitude.
Note: You may want to check that on the stim.
Means the controller balance has shifted towards advancing spark/adding torque. This is a more or less nasty way to cause more slip that may be feasible to heat up the tires and increase traction this way, however the assumption is made that the road surface provides (“ascending“) friction in general.
Note that we didn’t change the behaviour when traction is lost/recovered.

5.3 Tuning for Slippy Conditions

Slippy conditions require a different strategy since you commonly find “decending“ friction behaviour. For example, friction on ice produces a water film that decreases traction, same with heavy rain as water is “sucked“ in between tire and surface, called “aquaplaning“.
For these conditions you would do exactly the opposite as above, increase retard low and decrease advance low thus only little amounts of torque are added when slip decreases and recovery is “dampened“. Additionally you may want to increase the amount of each step size to reduce reaction time.

5.4 Other Controller Strategies

As shown in the previous examples you can influence on controller balance by changing the step sizes/fuzzy singletons but if you do so, you need a valid reason for that. In theory a neutral balanced controller should work best overall, acceleration, cornering and fast varying surfaces, so test with decreasing the amount of low step sizes while preserving balance before changing the controller balance.

5.5 Stepsize Collection

These are intended as a start point for your own evaluations.

Warm/Dry:	-1.0, -0.6, 0.3, 1.0 (unbalanced/tested)
Warm/Wet:	-1.2, -0.8, 0.8, 1.2 (balanced)
Cold/Dry:	-1.0, -0.6, 0.6, 1.0 (balanced/with fuel cut/tested)
Cold/Icy:	-1.5, -1.0, 1.0, 1.5 (balanced/with fuel cut)

6 Benchmark Criteria

As the name says, it's Benchmark time, something intended especially for acceleration and the quarter mile guys will love it ;). Ok, let's start with defining some criteria for a good tuned controller/system, basically anything that adds to our time, creates torque loss and so on.

- **Time for closing the clutch:** That's part of Launch Control, as it depends on the engine rpm, traction will slow down the engine rpm while the clutch closes, the less RPM you have, the faster the clutch should close completely, be cautious about this as you may burn the clutch. The time can be easily calculated since you know what ADC value represents which wheel rpm and gear ratio and engine rpm as well.

- **Time to reach maximum controller output:** This is something, you can tweak with the step sizes, the higher they are the less time it takes.

- **Maximum controller output amount:** This gives a hint if your tires match the engine torque, e.g. if it's too high you may need wider tires and so on.

- **Time to peak slip:** In the datalogs, you see that the driven wheel makes an upslope and a downslope before the wheel rpms close in. The downslope is what we're interested in, as the driven wheel starts accelerating and peak slip is near. Up until here, the worst time losses have already occured, so the faster we reach here, the better it is.

- **RPM/Torque at peak slip:** This is a cruitial thing to high rpm engines that usually don't produce much torque at lower rpms as it's the case with our Formula Student Cars. So it's our intention not to drop past the max torque rpm as it will cause time loss, eventually more than what slip would do.

7 Integration AdvTC with Launch Control

This is FAQ related. Launch Control and Traction Control are independent features of Megasquirt by tradition. Launch Control handles the state when the engine is waiting under Wide Open Throttle conditions for the clutch to engage and disengages after the (clutch) switch is off. The wheels will start spinning as the clutch connects and that's the moment Traction Control takes over. Means each feature handles it's own state which can't be true/active at the same time. Therefore it is not neccessary to do a handshake between AdvTC with Launch Control. At least until now. A future Version of AdvTC may do a handshake with the fuzzy fuel event balancer to actuate traction. A feasible way to increase fuel economy and decrease tire wear as the Moto GP's "long bang" engines.
Basically tweak Launch to minimize the time to close the clutch thus minimize the wear on the clutch. And on the second step tune AdvTC as described in the section before.

8 Integration with other Spark/Advance Controllers

As part of the last release AdvTC disengages during use of other spark/advance control mechanisms, e.g. Boost Control, Launch Control, Flat Shift and Rev Limiter.

9 Outline

As summary, this controller adapts to every surface by means of fuzzy logic (the way computer science looks at controller theory instead of those conventional engineers that still live in the middle ages and claim PID controllers are state of the art). It will estimate, target and hold peak slip condition, where the wheels may spin but the accerleration is optimal. In theory it may even be able to sustain a constant drift (as long as the undriven wheel does not decelerate). This controller is a contrast to all of those conventional piggyback solutions on the market which can't deem up with a fuzzy concept.

10 Author's Note/Legal Things/Disclaimer/Closing

AdvTC is an independent module that I have originally written for Megasquirt 2, intended for our 2009 Formula Student Car. It was my aim to do something revolutionary in Formula Student instead of repeating/copying what other's have done time after time, that there is even the 5th Volume of a book describing it.

AdvTC is considered Open Source, distributed under Gnu Public License as defined by the Free Software Foundation, means you may use it for educational/non-profit purposes, modify/evolve it and what comes out of that must be Open Source as well. There is no warranty for anything!

http://www.gnu.org/licenses/gpl.html

This step is in order to provide AdvTC's features to other free ECUs and prohibit profit-oriented ECU Manufacturers to use it. The LGPL won't do for that purpose.

Caused by a licensing discussion with the MS2Extra Authors, AdvTC is no longer an official mod of MS2Extra. Instead there will be a true open source ECU based on ARM Cortex M3 in the near future. For now, AdvTC will only be available as diff file to the ordinary MS2Extra V3.03r Code found with this tutorial. Means, download their source, push the diff file over it by using a suitable program preferebly Free Software as well, and just compile it for your platform. Debian/Ubuntu Users just install anything found with "m68hc1x" in synaptic, "patch"("man patch" for details) the source with the diff and "make all".

©Dipl.-Ing. Patrick Menschel
Email: Menschel.P@posteo.de