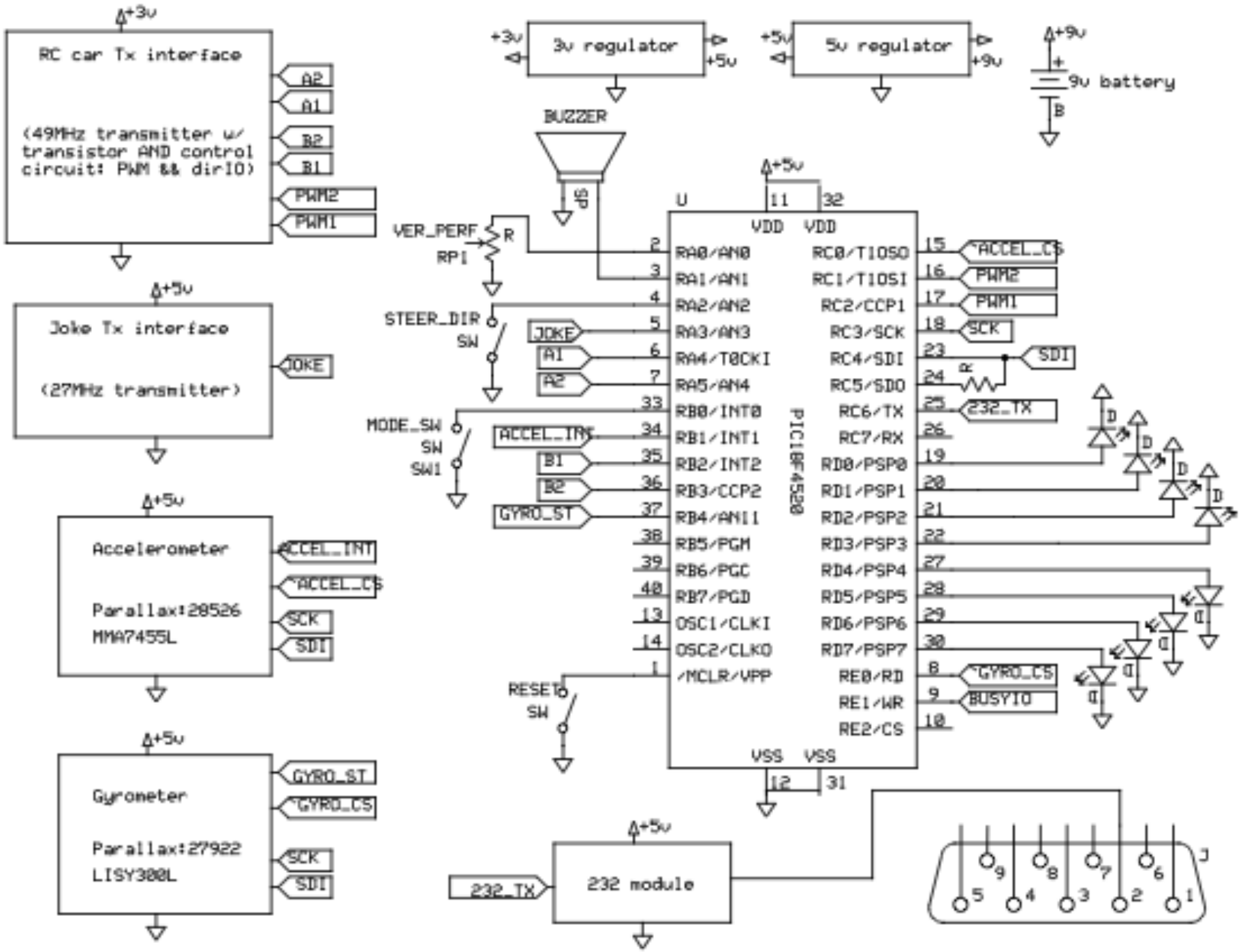
Remote control car “Wii-style” controller design

Replace RC car controller with accelerometer & gyrometer controller
Variable drive & steering
Make interface more intuitive and fun!

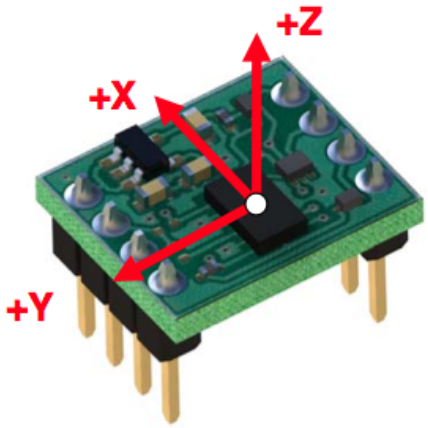
Sean Harre
02/2010

Hardware module overview



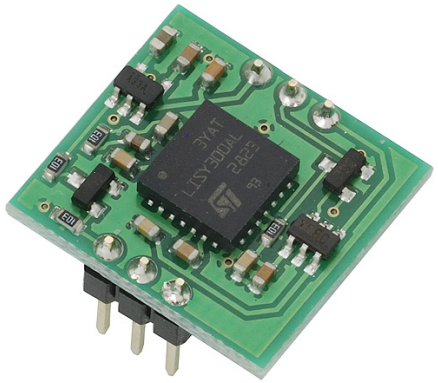
Sensor module detail

- Accelerometer module (Parallax#28526) uses Freescale MMA7455L
- For this project, use 8bit data in ±2g mode
  - Each ADC step = $4g/2^8 \cong 0.016g$/step, thus +1g is 64d
- Accelerometer reports force, integrate to find velocity



	Axis	Range	Output	Power	Voltage	Sampe Rate	Interface	Write offset
Accelerometer	3	+/- 2g	signed 8bit/axis	3mA	2.5-5.5v	125, 250Hz	SPI, I2C	Yes
Gyrometer	1	+/-300deg	unsigned 10bit	5.25mA	3.4-6.5v	88Hz	SPI	No

- Gyrometer module (Parallax#27922) uses ST LISY300AL
- Gyro MEMS analog output at rest = 1.65V
  - Voltage falls when rotated CW around z-axis
- Max MEMS voltage is 3.3V and ADC is 10bits
  - Thus each step $(3.3V/2^{10}) \cong 3.22$mV/step
- Gyro MEMS sensitivity = 3.3mV/°/sec
  - Thus each ADC step ≅ 1°/sec
- Gyro output reports angular velocity



- Both modules support a max of 4MHz SPI
  - 1MHz is used in this demo

Sensor data filters

- As raw sensor data is received, it is added to the average window filter,
  - Buffer is 25 samples deep
    - 25 signed chars for each X,Y,Z, 25 signed integers for gyro
  - Once the raw value circular buffers have been filled, a new sample results in,
    - Oldest sample is removed from the buffer and it's value subtracted from the sum
    - The new sample is written to the buffer and added to the sum
    - Current window average is computed and added to derivative filter
- Derivative window filter,
  - Buffer is 10 samples deep
    - 10 signed chars for X,Y,Z, 10 signed integers for gyro
  - For each new sample from averaged filter,
    - A delta is computed from the previous sample
    - Replace oldest sample with latest
    - Delta average is updated
- The time for a new sample to be pushed to the derivative filter
  - 200ms for accelerometer samples
  - 300ms for gyrometer samples
- Gives a good response time but still provides some basic averaging

Post-process sensor data

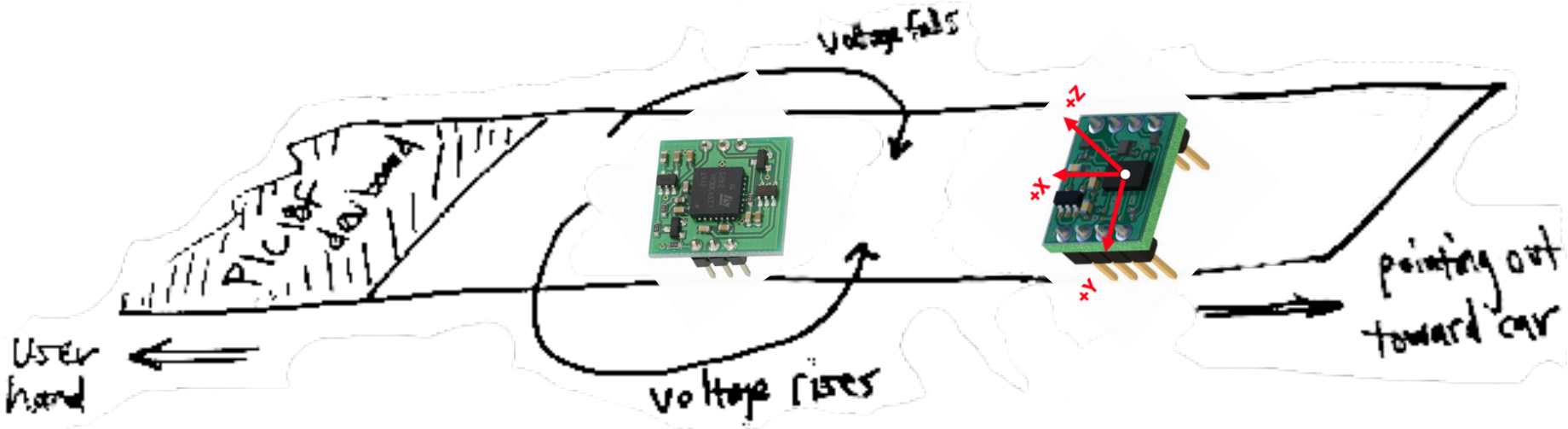
- To determine motion algorithms, RS232 sensor data to PC
- (3) debug modes transmit sensor data as it is received,
  - Raw sensor data (X, Y, Z, and gyro)
  - Averaged data
  - Derivative data
- Data sent as binary packets to reduce RS232 bandwidth,
  - Accel data: ‘A’ [X] [Y] [Z]
  - Gyro data: ‘G’ [gyroH] [gyroL]
- Matlab can read directly from serial port
  - Processes (4) 24ms sample cycles (72bytes) in ~1ms, plot update ~20ms
  - Thus 96ms of sensor data updates the plot window every 115ms
- Accel and gyro sample rates differ,
  - Packets are interleaved and repeat in the sequence shown below

24ms sample cycle (3 accelerometer reads + 2 gyrometer reads = 18bytes)

Timer0 count [4ms]	Time (ms)	accel RDY	gyro RDY
1	4	0	0
2	8	1	0
3	12	0	1
4	16	1	0
5	20	0	0
6	24	1	1
(... pattern repeats...)			

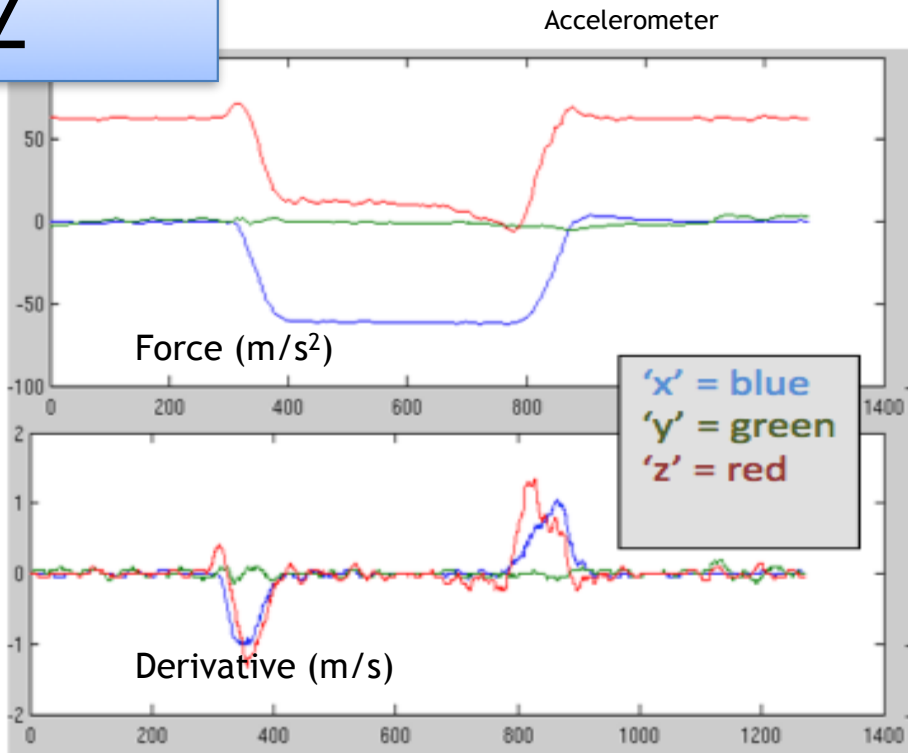
Sensor module orientation

- User holds onto controller at left such that right side is pointed at car

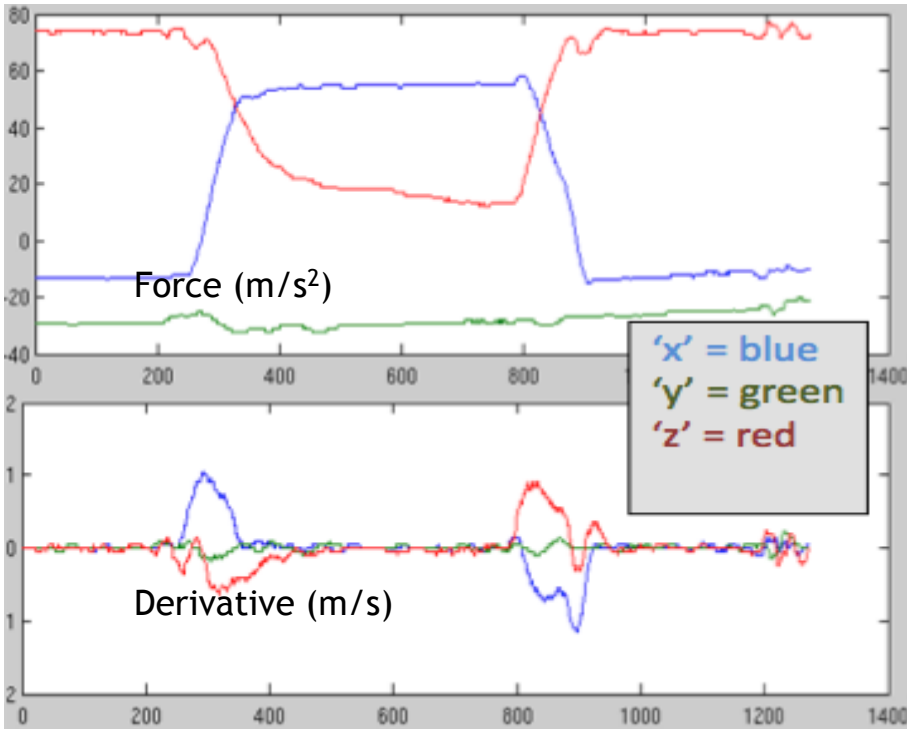


Movement pattern: X & Z

- Holding flat, point controller up, then flat
  - Z-axis force decreases as pointed up
  - Gravity pulls X-axis in negative direction

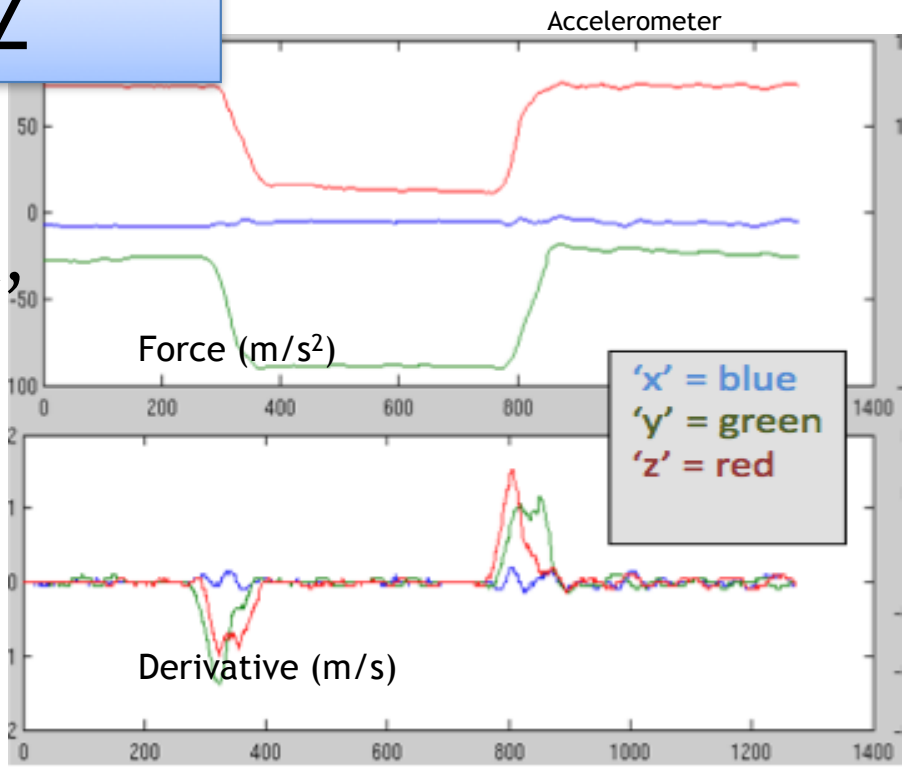


- Holding flat, point controller down, then back to flat
  - Z-axis force decreases as pointed down
  - Gravity pulls X-axis positive

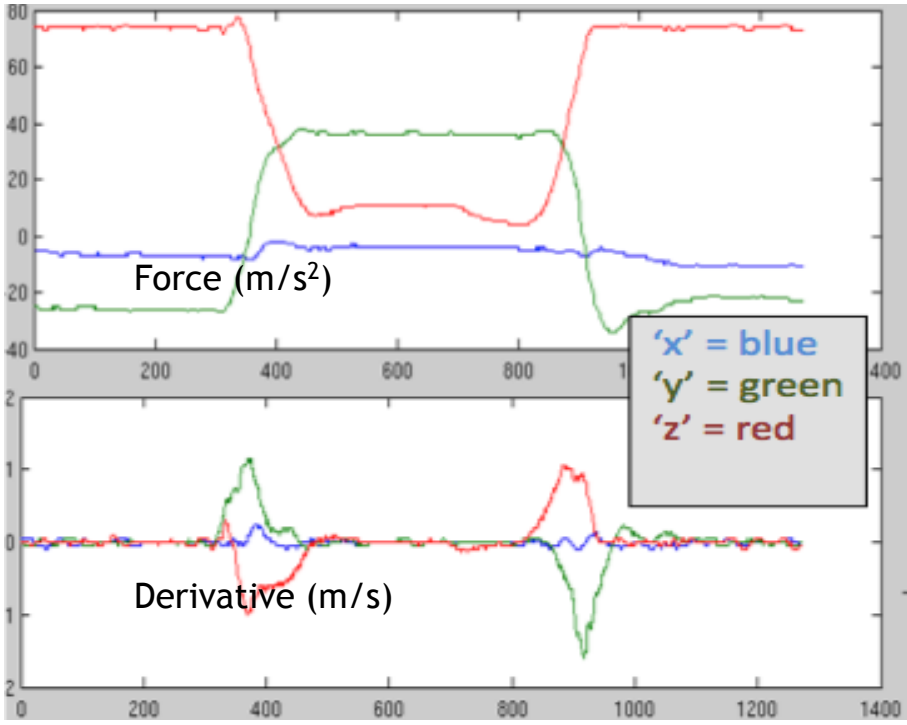


Movement pattern: Y & Z

- Holding flat, roll controller to right, then back to flat
  - Z-axis force decreases
  - Gravity pulls Y-axis in negative direction

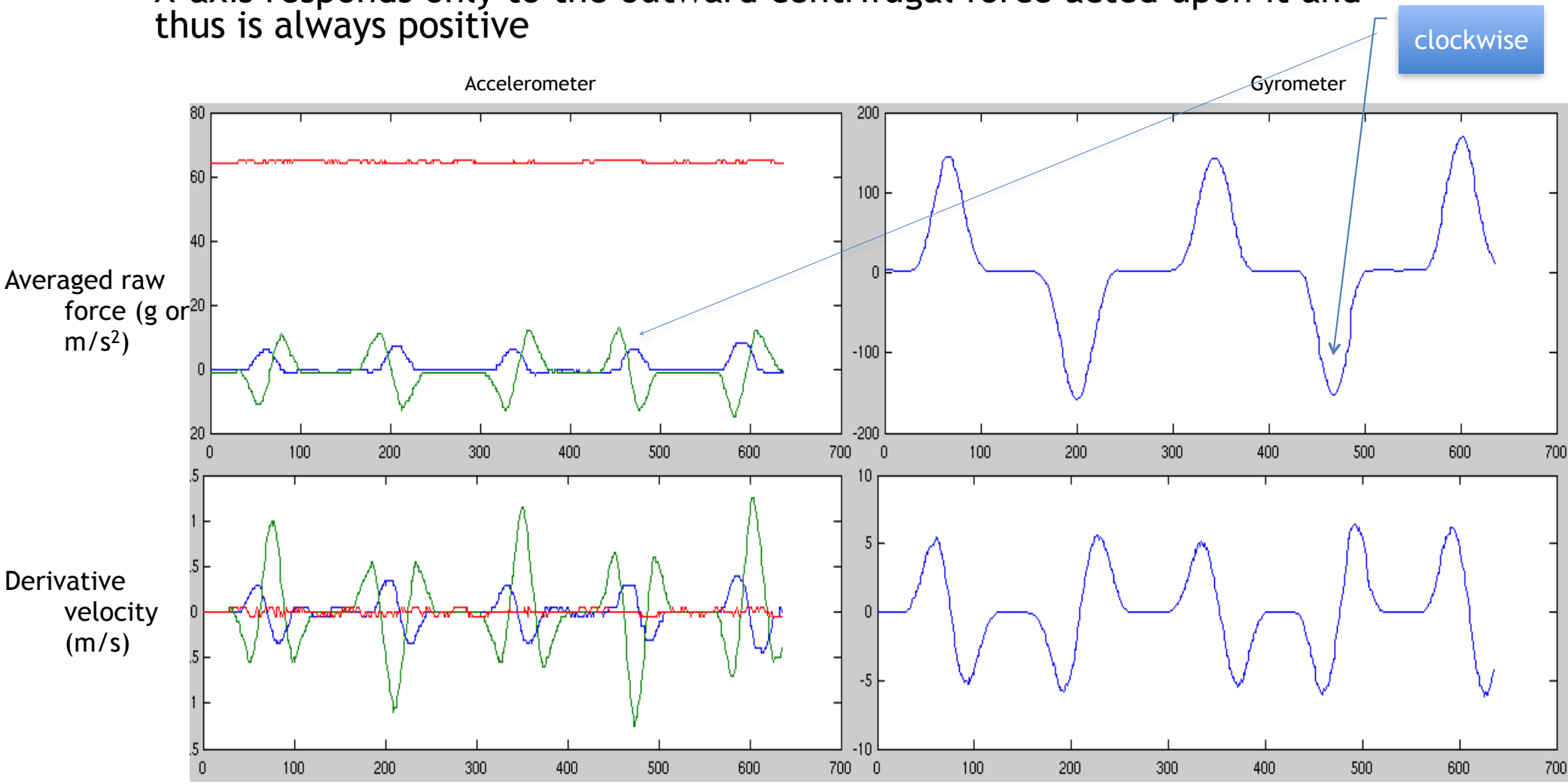


- Holding flat, roll controller to left, then back to flat
  - Z-axis force decreases
  - Gravity pulls Y-axis positive



Movement pattern: X & Y & G

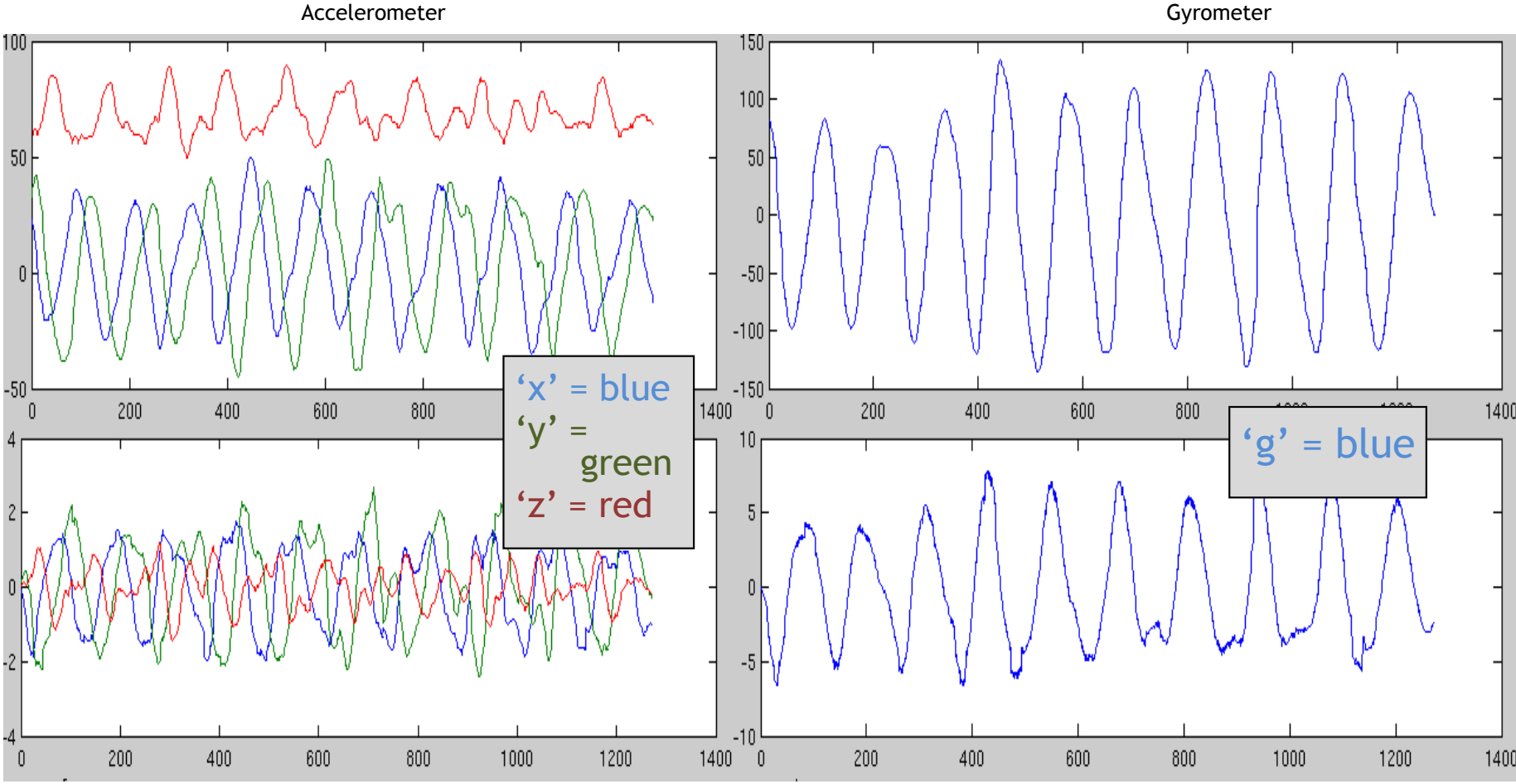
- With controller on flat surface, twist the controller about Z-axis
  - The gyro responds to rotation, clockwise makes gyro voltage fall, counter-CW makes gyro voltage increase
  - Accelerometer Y-axis responds to force on it's axis, first accelerated then decelerated
    - Integrating Y-axis produces (+) half-sine for CW, (-) half-sine for CCW
  - X-axis responds only to the outward centrifugal force acted upon it and thus is always positive



Movement pattern: X & Y & G

Not used in RC car

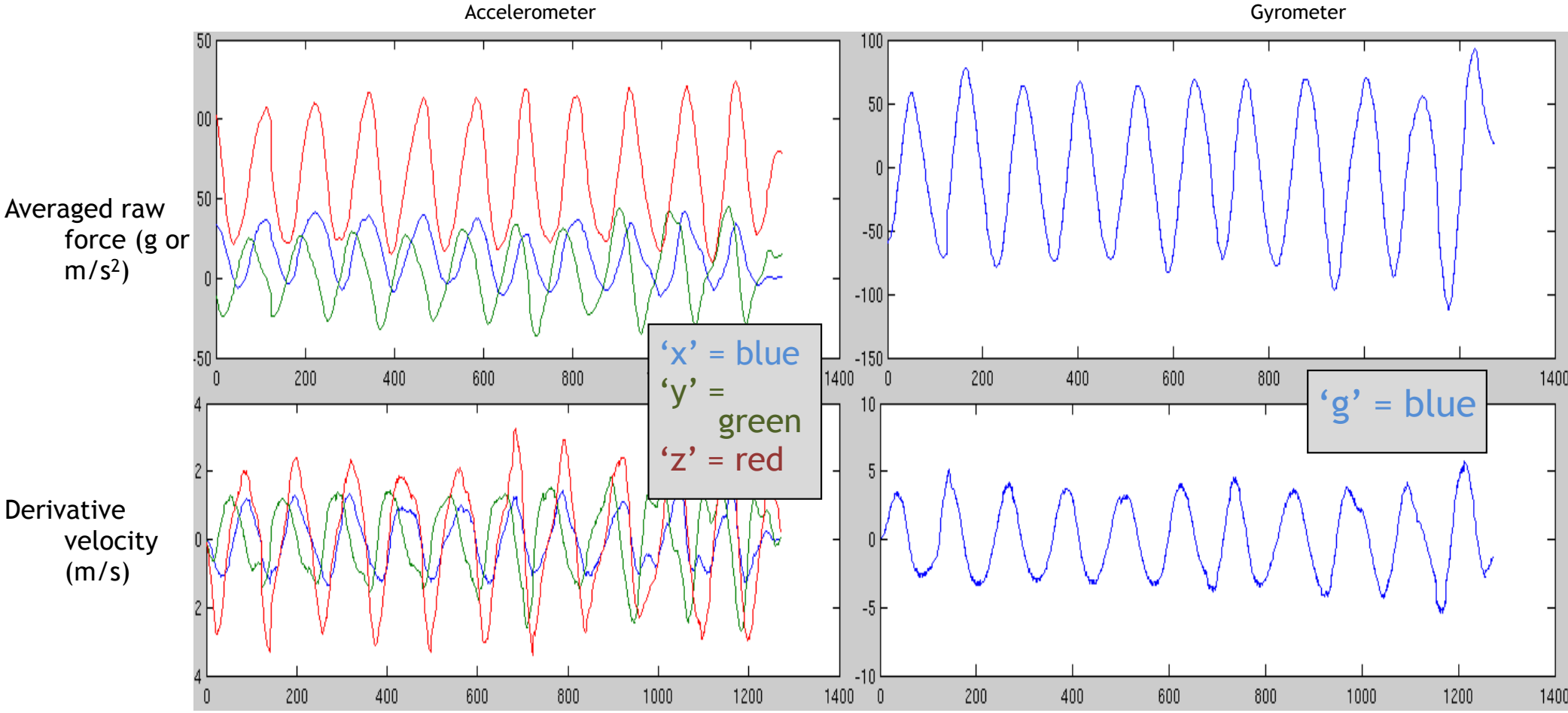
- Hold controller flat with 2 hands and move in large counter-clockwise circle in XY space,
  - Push out to right and circle back around in from the left
  - Gyro shows rotation induced from wrist moving the controller in circle
  - Force on X-axis leads force on Y-axis
    - Pulling towards you gives (+) X values, pushing to right gives (+) Y values
  - Z-axis shows movement because the circling motion was slightly tilted up to one side



Movement pattern: X & Y & G

Not used in RC car

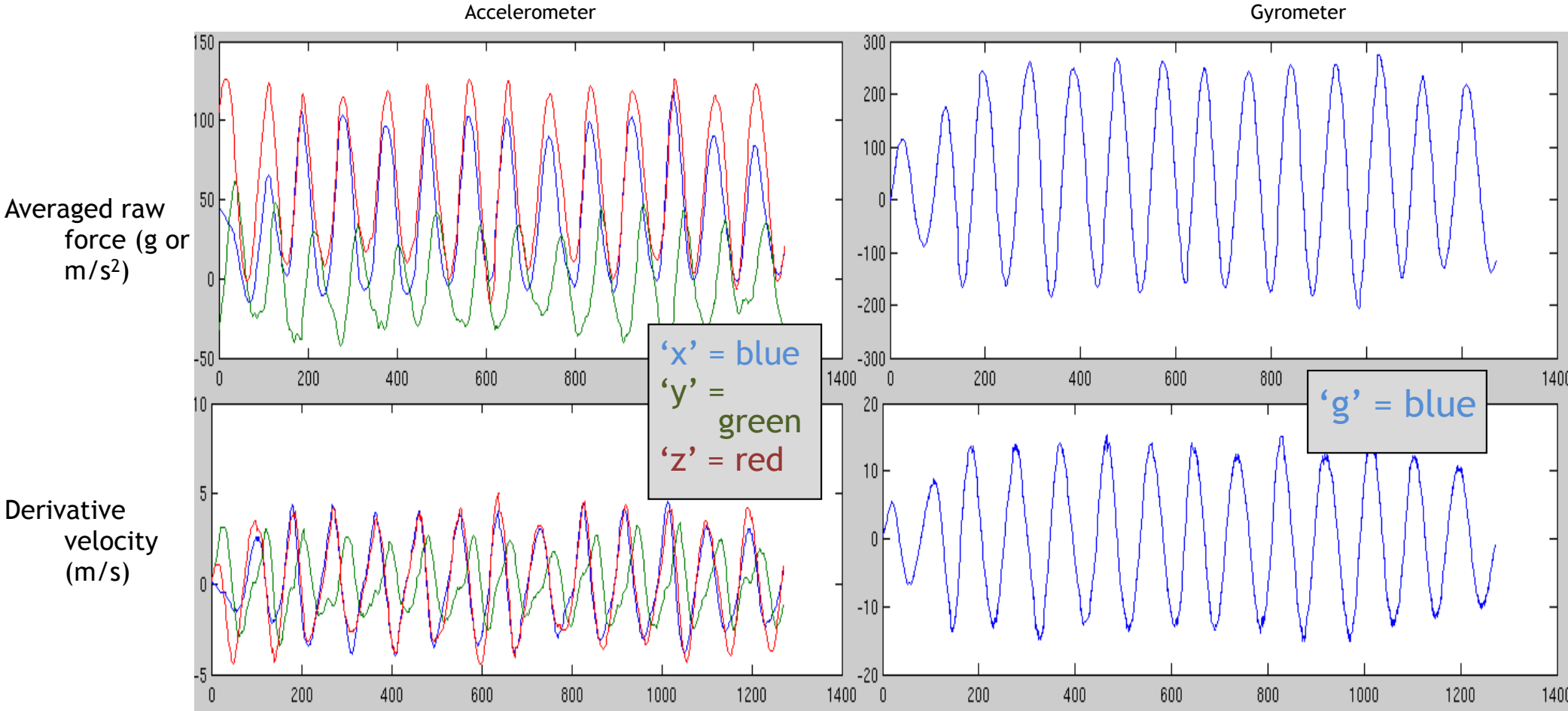
- Hold controller flat with 2 hands and move in large counter-clockwise circle in YZ space
  - Holding flat, move controller up to right above head and rotate back down in a circle to the left
  - Y-axis leads Z-axis since forces are being applied at different times
  - X-axis shows some positive motion since controller was not held perfectly still in X-axis



Last movement pattern: all-axes

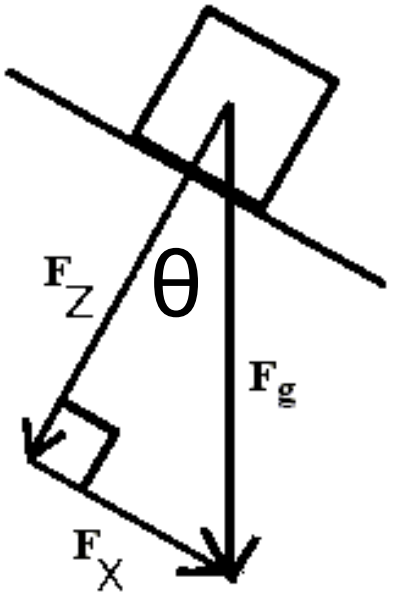
Not used in RC car

- Hold controller flat with 1 hand and rotate counter-clockwise in cone shape in YZ space
  - Holding wrist flat, move wrist counter-clockwise so hand remains stationary and tip of controller makes a cone in YZ space
  - Y-axis still leads Z-axis
  - But now X-axis is in phase with Z-axis as tip is tilted during circle



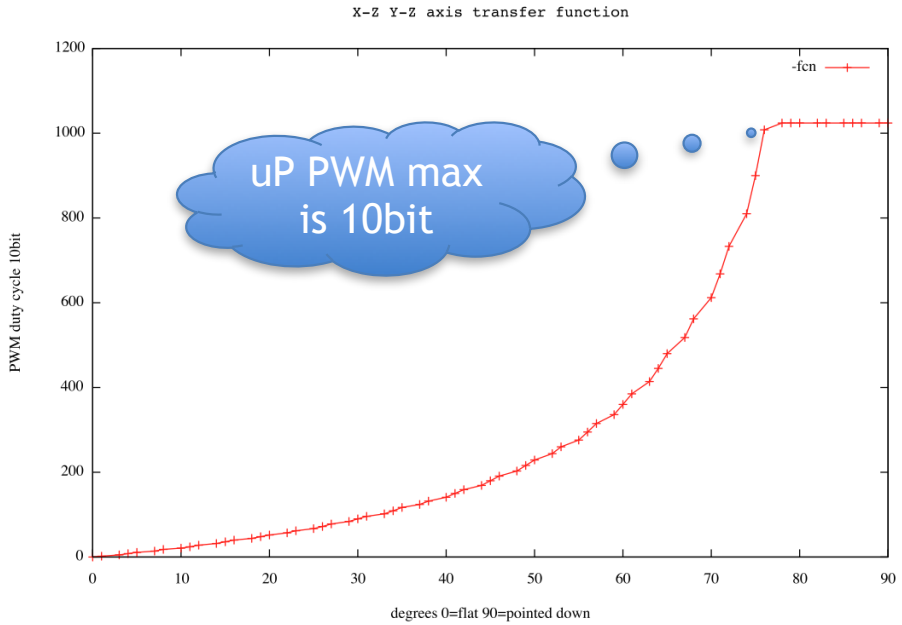
Determining controller tilt

- Accelerometer Z-axis measures 1g down when held flat
- ‘g’ vector can be tracked in 3D as controller moves
  - Tilt forward: Z decreases as X increases
  - Roll left: Z decreases as Y increases
- But sensor forces and gravity appear the same
  - Should only measure ‘g’ vector when derivative values are under some threshold



‘g’ vector algorithm

- Want to reduce processing time and meet timing requirements
  - Can’t use usual ‘arctan’ method to find theta
- Calculate ratios AvgX/AvgZ and AvgY/AvgZ
  - X/Z = 0 when flat, controller down = increasing positive numbers, controller up = increasing negative numbers
  - Y/Z works the same way (0 @ flat, increasing (+) to left, increasing (-) to right)
- Curve is exponential since abs(X|Y) increases as Z decreases
  - When held flat, small controller movements produce minimal affects on RC car
  - Controller needs to be intentionally angled to produce affects on RC car mobility
- This curve is used to translate user movement to PWM drive signals that toggle the forward/backward and left/right buttons



Firmware structures

- The brain structure contains the latest averages and derivatives
  - Averages and derivatives updated every sample
  - X/Z & Y/Z slopes and booleans updated every 24ms sample cycle

- The movement configuration structure contains thresholds and multipliers used to control the car movement

```
typedef struct {
        signed char SYS_AVGx;
        signed char SYS_AVGy;
        signed char SYS_AVGz;
        signed int SYS_AVGg;
        signed char SYS_AVGDERIVx;
        signed char SYS_AVGDERIVy;
        signed char SYS_AVGDERIVz;
        signed int SYS_AVGDERIVg;
        signed int Axz_slope;
        signed int Ayz_slope;
        BOOL motion_any;
        BOOL motion_large;
        BOOL rot_left;
        BOOL rot_right;
        BOOL rot_hard_left;
        BOOL rot_hard_right;
        BOOL sensors_up;
        BOOL just_updated;
        BOOL valid;
} brain;
```

```
typedef struct {
        unsigned char SLOPE_MULTXZ;                    // 2
        unsigned char SLOPE_MULTYZ;                    // 3
        unsigned char AXZ_MOTION_OFF_MIN;              // 4
        unsigned char AYZ_MOTION_OFF_MIN;              // 5
        unsigned char GYRO_ROT_MIN_TURN_STEERING;      // 6
        unsigned char ROT_MULT;                        // 7
        unsigned char GYRO_ROT_HARD_STEER;             // 8
        unsigned char ROT_MULT_HARD;                   // 9
        unsigned char PWM1_MULT;                       // 10
        unsigned char PWM2_MULT;                       // 11
        unsigned char PR2;                             // 12
        unsigned char PWM1_DYN_THRES;                  // 13
        unsigned char PWM2_DYN_THRES;                  // 14
        unsigned char PWM1_DYN_MULT;                   // 15
        unsigned char PWM2_DYN_MULT;                   // 16
} movement_cfg;
```

Firmware ‘g’ vector algorithm

… in `update_brain()`,

```
b.Axz_slope = (b.SYS_AVGz <= 0) ? \
        ((signed int)m.SLOPE_MULTXZ * b.SYS_AVGx) : \
        ((signed int)m.SLOPE_MULTXZ * b.SYS_AVGx) / b.SYS_AVGz;
```

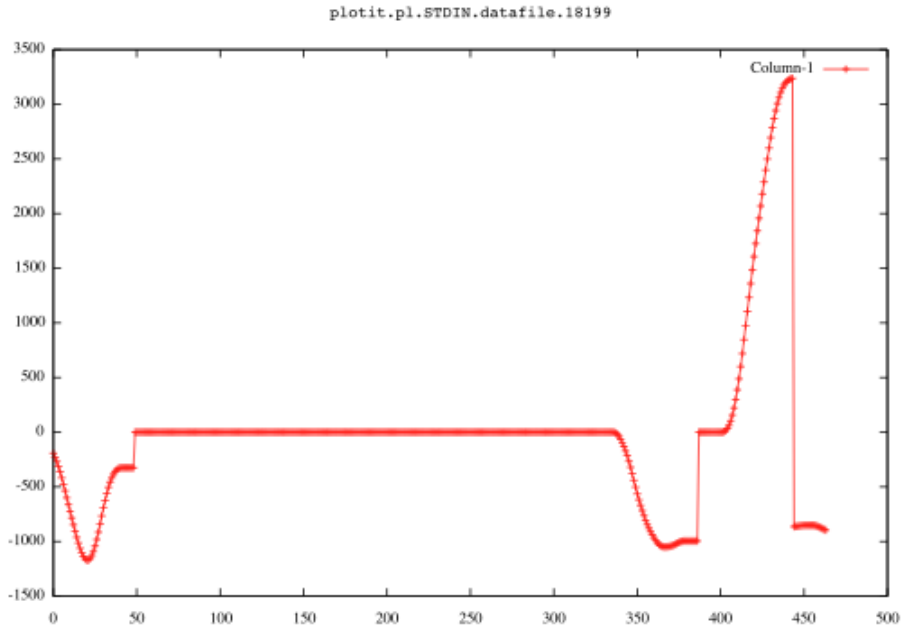
Then in `main_rc_car()` for forward movement, for example,

```
if (tABS(b.Axz_slope) < m.AXZ_MOTION_OFF_MIN) {
    FORWARD_BACKWARD_OFF;
} else if (b.Axz_slope > 0) {
    GO_FORWARD;
    SET_PWM2_DUTY(((unsigned int)tABS(b.Axz_slope)* \
        (unsigned int)m.PWM2_MULT) > PWM2_drive_max ? \
        PWM2_drive_max : \
        (unsigned int)tABS(b.Axz_slope)* \
            (unsigned int)m.PWM2_MULT);
} else {
    GO_BACKWARD;
    …
```

Gyro steering

- Take the integral of gyro average filter for basic position indicator
- Being an integral, error bias can accumulate
  - To reduce the error bias, do two things:
    1. Maintain an integral baseline
       - When integral sum is needed, subtract the baseline value
       - Set baseline value = integral sum after the controller has been held flat and still for 240ms
    2. If the ‘g’ vector and gyro integral sum do not agree, believe ‘g’ vector and set integral sum = ~0

- Gyro directional result is then used to amplify steering multiplier

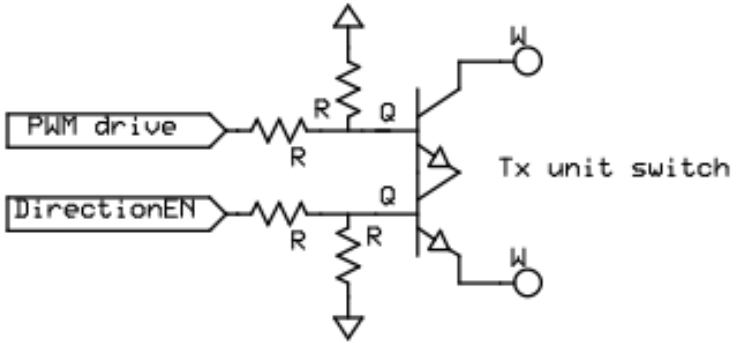


Software flowchart

- “while(1)” main loop cycles in ≈40us
  - Look for switch presses from user
  - Check to see if 4ms Timer0 fired
    - Gyrometer every 12ms, accelerometer every 8ms
  - Calculate “brain” variables if it’s time to do so
  - Check to see if Timer3 fired
    - Used to produce buzzer events and time Joke on-pulses
  - Run “main” code:
    - (DEFAULT) ‘main_rc_car()’: Setting PWM, I/O, etc based on “brain” decisions and movement configuration settings
    - (AT BOOT) ‘main_calib()’ : Takes samples to zero offsets for X, Y, Z, and gyro

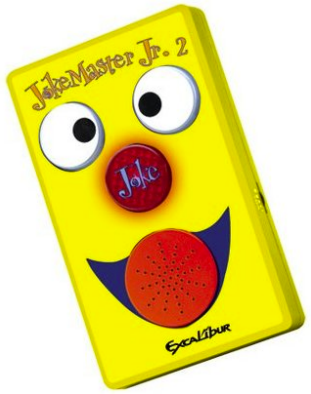
49MHz RC car Tx unit interface

- Resides in controller, driven by uP
- Use 2 transistors (for each Tx switch) as an AND gate to pass PWM signal and directional enable signals
  - PWM1 drives either left or right
  - PWM2 drives either forward or backward
- With controller at rest
  - PWM drive is off
  - Directional enable signals are off



“Joke Master” unit

- Jerking controller triggers the Joke unit in RC car to tell a joke
- A 2nd RC Rx/Tx unit operating at 27MHz is used to control the Joke unit
  - Can use standard H-bridge output as “I/O”
    - Totem pole output, ½Vcc at off state
    - On activation, one side goes to GND, other side to Vcc
  - For joke control, take one side of H-bridge output ([A])
    - Normally, [A] side 1.8v drives transistor into saturation
    - When pressing switch on Tx unit, [A] falls to ground
    - Invert [A] to drive switch on Joke unit
- 27MHz Tx unit is also mounted in controller
  - Needs >200ms pulse to activate remote joke unit



Design analysis

uP idle time & power optimizations

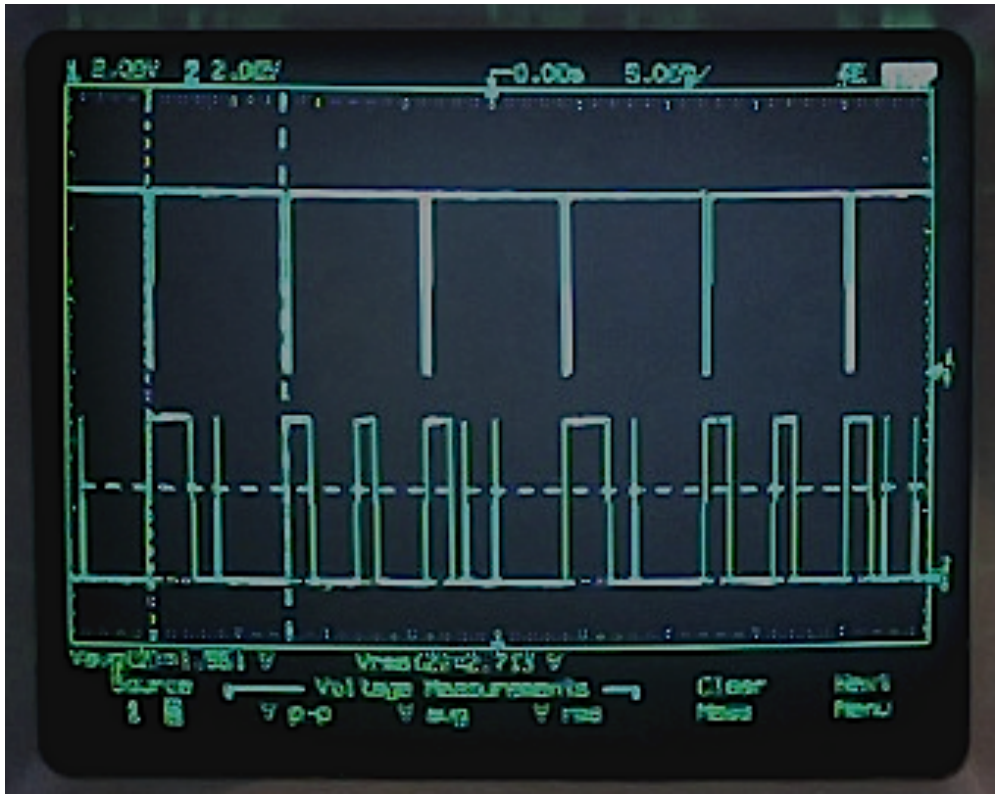
- This code does not sleep, significant power could be saved as processing of samples is fairly fast and most uP time is spent spinning thru main loop
- Scope (5ms/div) shows accel ~CS on top (every 8ms) and sample processing time on bottom
  - Processing time I/O is set on each 4ms timer interrupt and includes reading accel and/or gyro samples, storing the values to both the averaging and derivative arrays, and updating any decision logic

- The sensor sampling sequence can also be seen
  - first ~CS on left is accel & gyro read from the last sequence
  - New sequence at 2nd ~CS is 1st accel, then gyro (no ~CS)
  - Next ~CS is accel, wait 8ms and
  - then accel & gyro sampled at same time
  - Then sequence then repeats
- This cycle is 24ms, roughly half spent in non-processing activities
- Current power usage is ~270mW, which includes two bright LEDs and (3) other LED, and a very power hungry Tx drive unit (low R's in transistor drive),
  - At rest (no Tx unit drive), uses 33mA
  - At any Tx drive = 51mA
  - Max (up and steer) = 63mA

- *Recommendations*
  - Remove LEDs or use 1 dim LED for "on"
  - Sleep, use 4ms timer interrupt to wake up
  - Increase R's in Tx drive

- Power usage could then be reduced to
  - (33mA-[2LED]8*2-[3LED]3*3)+(4*1mA[Txdrive]) ~= 60mW

Timer0 count [4ms]	Time (ms)	accel RDY	gyro RDY
1	4	0	0
2	8	1	0
3	12	0	1
4	16	1	0
5	20	0	0
6	24	1	1
(... pattern repeats...)			



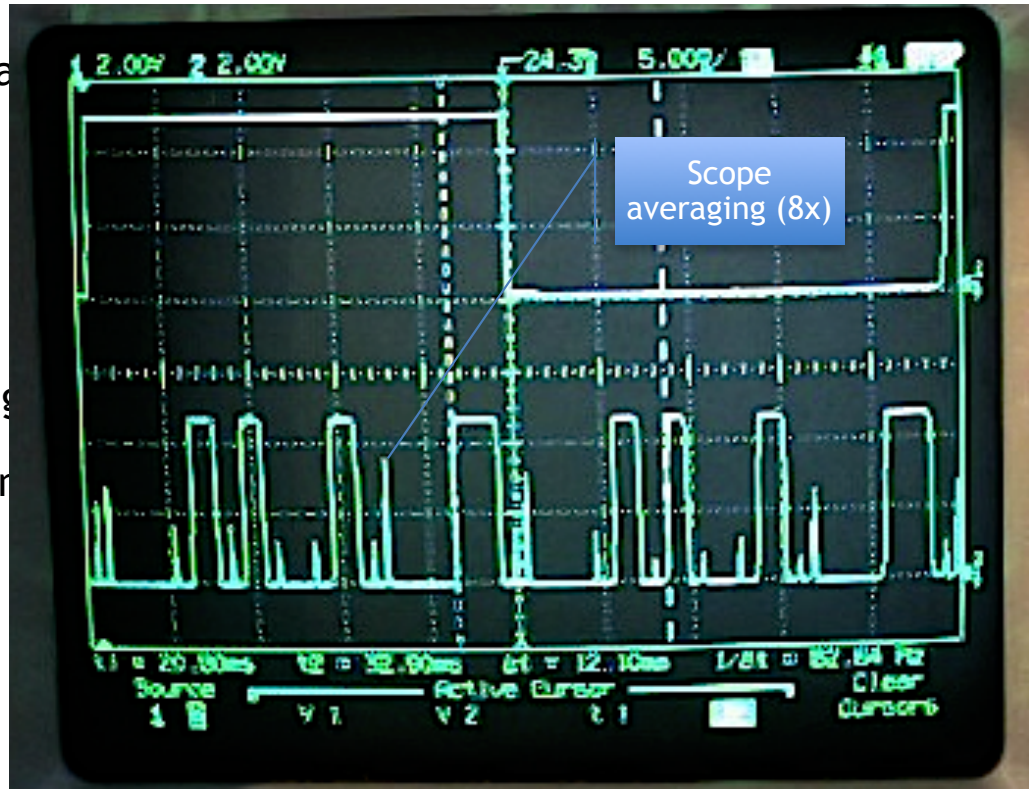
Design bug rework

- The previous slide showed the timing of sensor sampling
  - Accel, gyro, accel, 8ms, accel & gyro at same time
  - @timer0=6, accel and gyro read occur at same time, where accel sample is first, then gyro sample
  - But the accel read time was pushing out the gyro read, this was not being taken into account
  - The result was the gyro being sampled at 13.5ms (@timer0=3) and 10.5ms (@timer0=6)
    - But the max data rate of gyro is 88Hz or 11.4ms, *so the sample at 10.5ms was possibly reading duplicate data*

Timer0 count [4ms]	Time (ms)	accel RDY	gyro RDY
1	4	0	0
2	8[+1ms]	1	0
3	12	0	1
4	16[+1ms]	1	0
5	20	0	0
6	24	1	1
(... pattern repeats...)			

- The rework:
  - On timer0=6, sample gyro first, then accelerometer. The gyro now samples correctly at 12ms
  - But now the accel read @timer0=6 was now forced out 1ms (by the gyro read), giving Δt=7ms to the next read, which was too fast for its 125Hz rate
  - To solve this, in the timer0 interrupt, for timer0=2 || 4, reset the timer for 1ms and set a flag. In 1ms, timer0 fires, clear flag, and take accel sample. Reset timer for 3ms to get back on sync
  - Now the accel samples correctly at 8ms

- Scope trace shows updated waveforms
  - Top trace is IO toggling every 24ms for triggering on sampling sequence
  - Bottom trace is processing time, as described on previous slide
  - Δt shows 12ms for gyro samples

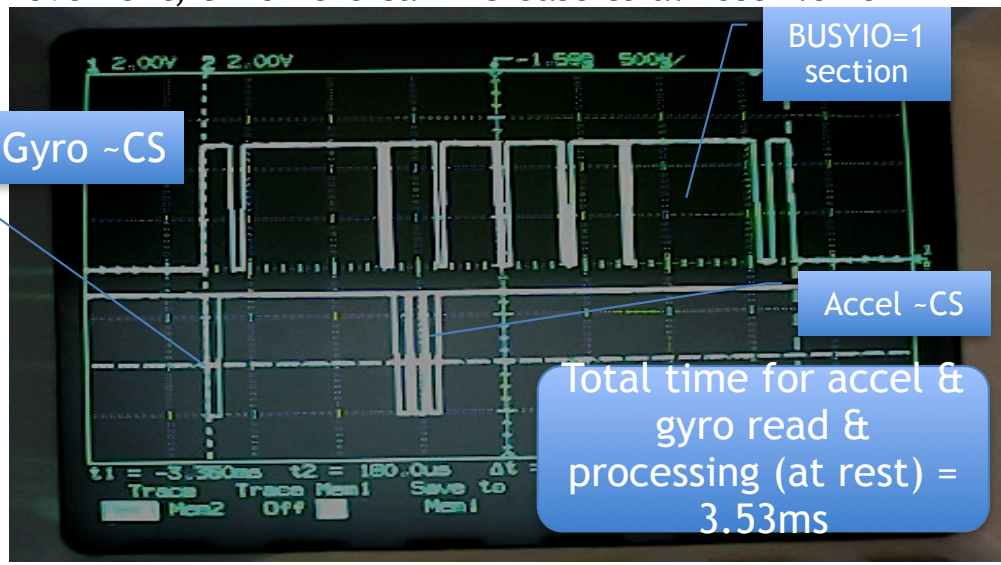


Sensor stream processing

- Sensor samples are processed as they are received
  - The actual measurement time is not that large, at 1MHz SPI, accel measurement takes 72µs (for one) and gyro sample takes 76µs
- The bulk of the time is spent storing the samples into the average and derivative filters, and updating the "brain" variables
  - More specifically, the bulk is spent doing the division to determine avg/deriv average

- The scope trace clearly shows the full processing time for a gyro and accel sample sequence (timer0=6) (each bullet below is one BUSYIO=1 section in trace)
  1. 10bit gyro sample[180us]: (BUSYIO=1&~CS=0[80us]) read as (2) 8bits, (BUSYIO=1&~CS=1[90us]) and stuffed into 16bit int
  2. Gyro storage[920us]: (BUSYIO=0[60us]) sample->AVGbuffer, (BUSYIO=1[830us]) (1)signed long division[750us] for AVG->DERIVbuffer
  3. Accelerometer sample[280us]: each read 90us,
  4. StoreX[390us]: (BUSYIO=0[40us]) sample->AVGbuffer, (BUSYIO=1[350us]) (1) signed int division[328us] for AVG->DERIVbuffer
  5. StoreY[390us]: same as X
  6. StoreZ[390us]: same as X
  7. Update "brain"[770us]: during movement, the time spent in this function can increase out towards 1ms
  8. main_rc_car()[180us]: depending on amount of movement, time here can increase to almost 1.5ms

- Since the time to update the "brain" and run main_rc_car() are variable, *code is not able to meet the sensor timing requirement during movement*,
  - The 24ms cycle time extends out towards 28ms when the controller is being moved and thus more logic to execute

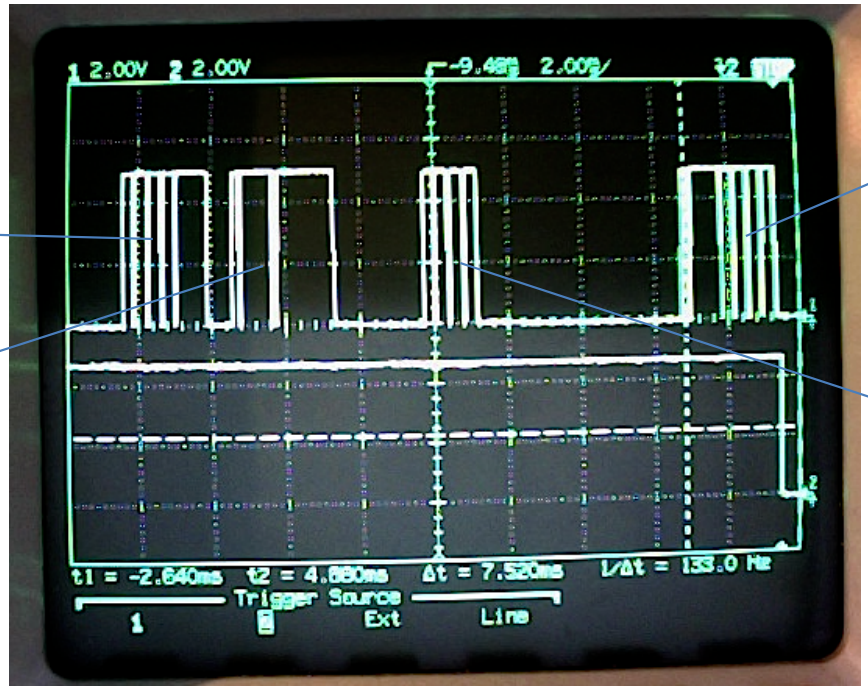


Sensor processing fix

- Instead of updating the “brain” and driving the main_rc_car() logic after each sample, just do this once every 24ms sample sequence
  - The existing ~80Hz RC car update rate will thus be reduced to ~20.8Hz, which provides adequate update
  - If 40Hz update is needed, could add another update_brain() and main_rc_car() in timer0=5
- It doesn’t matter where in the cycle these functions occur, but “brain” must be updated before calling main_rc_car()
- Also, don’t want to do them where another 4ms timer0 might fire, or resetting the timer0 value would be delayed, resulting in the cycle getting behind again

Timer0 count [4ms]	Time (ms)	accel RDY	gyro RDY	update "brain"	main_rc_car()
1	4	0	0	0	0
2	8[+1ms]	1	0	1	0
3	12	0	1	0	1
4	16[+1m	1	0	0	0
5	20	0	0	0	0
6	24	1	1	0	0
(... pattern repeats...)					

- Fix:
  - Update “brain” after the 1st accel read @timer0=2
  - Call main_rc_car() after 1st gyro read @timer0=3
- Now 24ms sampling cycle is maintained during movement



Remaining issues

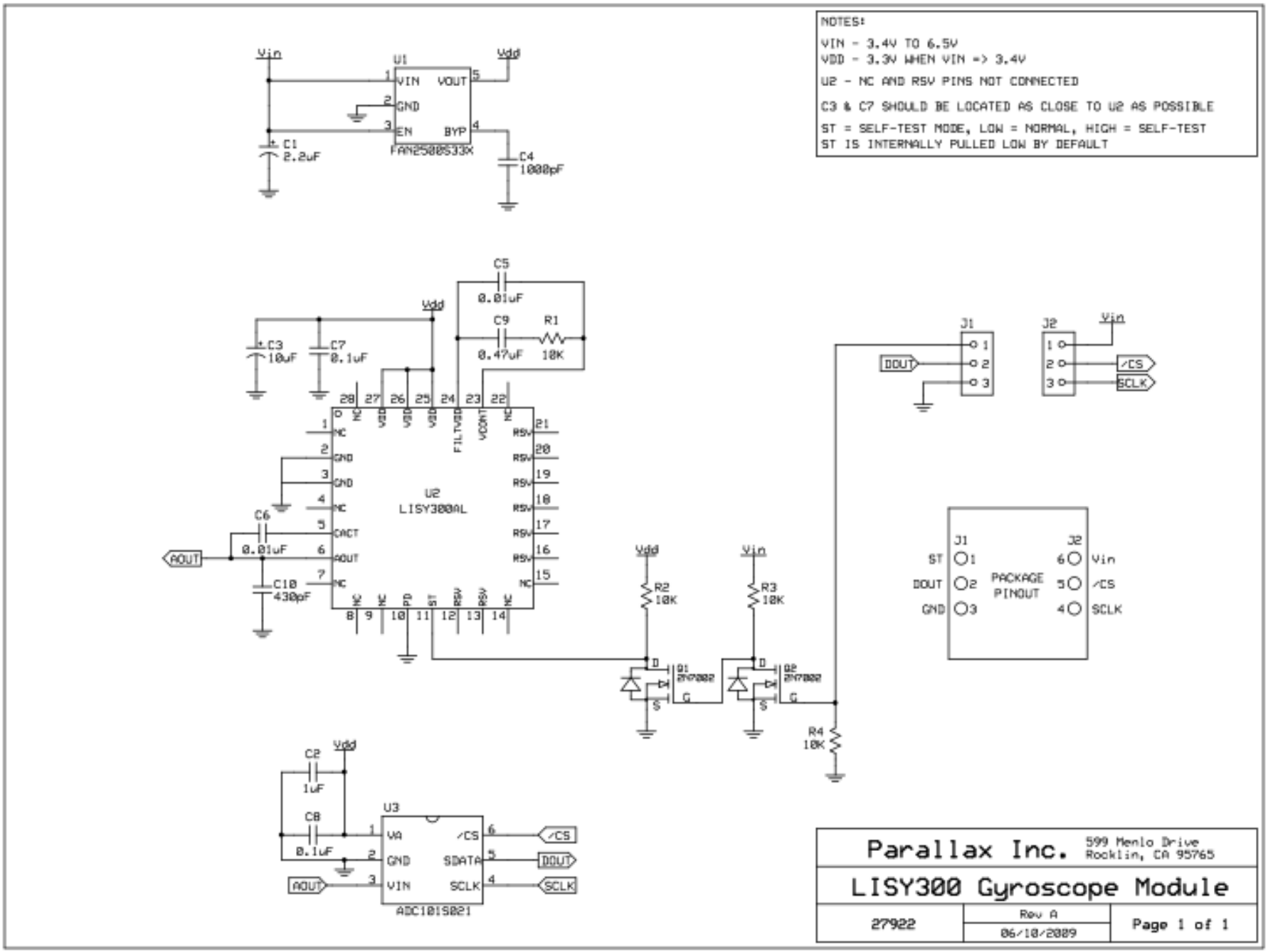
- uP firmware #defines for computing timer times are not correct
  - e.g., can't set buzzer time for less than 200msec -> need to fix prescaler values
- Use 4MHz SPI with both modules
- Neither accel nor gyro calibration is stored, should be placed into EEPROM and only orientation read at boot time. As it is now, needs to be calibrated each time powered on
  - Can't write offsets to gyro and written accelerometer offset values are lost at power cycle
- Accelerometer self-test sometimes is low
  - For applied EM field during self-test, should be deflected to +1g, or 64 ADC steps at ±2g in 8bit mode
  - Sometimes is 57 and as low as 55, over -11%
  - Should accelerometer reads be scaled by 64/57 to account for this? Is the calibration incorrect?
- Joke unit in RC car should be driven from high-side of H-bridge
  - Using low-side has negative consequence of triggering jokes when battery is low

Moving forward

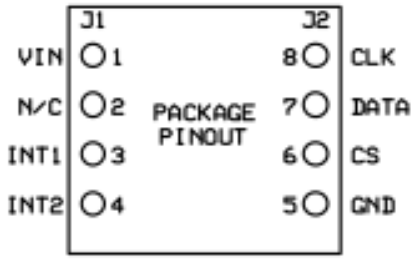
- User movement “gesture” detection needs further study,
  - Is it possible to use “inverse PID” algorithm to find error based on user movement so that control variables can be adjusted based on user corrections
  - ‘g’ vector works well but this is not really interpreting user actions
    - Car cannot be “driven” via hands with natural movement gestures
    - X/Z and Y/Z exponential slopes are too steep, can these be more linear
      - Tried arctan and X/pow(Z,¼), Y/pow(Z,¼) but these take too much processing time
  - Detecting user twist is currently not very accurate and should be correlated with X and Y accelerometer data
- Add button to toggle user control of car,
  - When button is pushed, user controls car, when button is release, car either maintain same settings, turns off, or slowly ramps down PWMs to off over a few seconds
  - Button could also allow relative motions
    - Today, all motion is absolute (flat=stop, up=backwards, etc)
    - Relative motions, for instance, controller pointing down, user presses the button and angles back up to flat. The car reaction is to back up
      - Thus, whenever button is pressed, that combo of X, Y, Z, G is considered ‘flat’ orientation
- Port to helicopter control
  - Requires 5 “channels” which can be accomplished with this design modified

Schematics

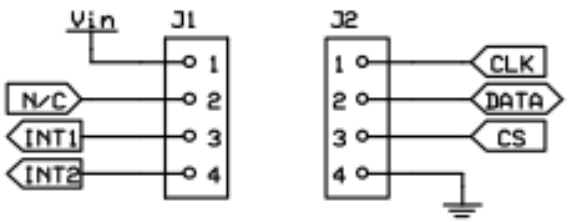
Gyro module

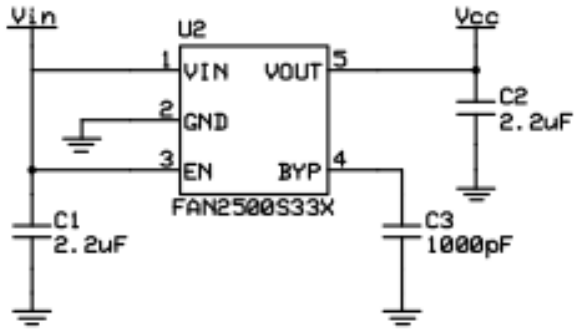

NOTES:
VIN = 3.4V TO 6.5V
VDD = 3.3V WHEN VIN => 3.4V
U2 - NC AND RSV PINS NOT CONNECTED
C3 & C7 SHOULD BE LOCATED AS CLOSE TO U2 AS POSSIBLE
ST = SELF-TEST MODE, LOW = NORMAL, HIGH = SELF-TEST
ST IS INTERNALLY PULLED LOW BY DEFAULT
Vin
Vdd
U1
VIN
VOUT
GND
EN
BYP
FAN2500S33X
C1
2.2uF
C4
1000pF
C5
0.01uF
C9
0.47uF
R1
10K
C3
10uF
C7
0.1uF
U2
LISY300AL
C6
0.01uF
C10
430pF
AOUT
R2
10K
R3
10K
R4
10K
Q1
2N7002
Q2
2N7002
J1
J2
DOUT
/CS
SCLK
ST
GND
PACKAGE
PINOUT
C2
1uF
C8
0.1uF
U3
VA
GND
VIN
SDATA
ADC101S021
Parallax Inc.
599 Menlo Drive
Rocklin, CA 95765
LISY300 Gyroscope Module
27922
Rev A
06/10/2009
Page 1 of 1

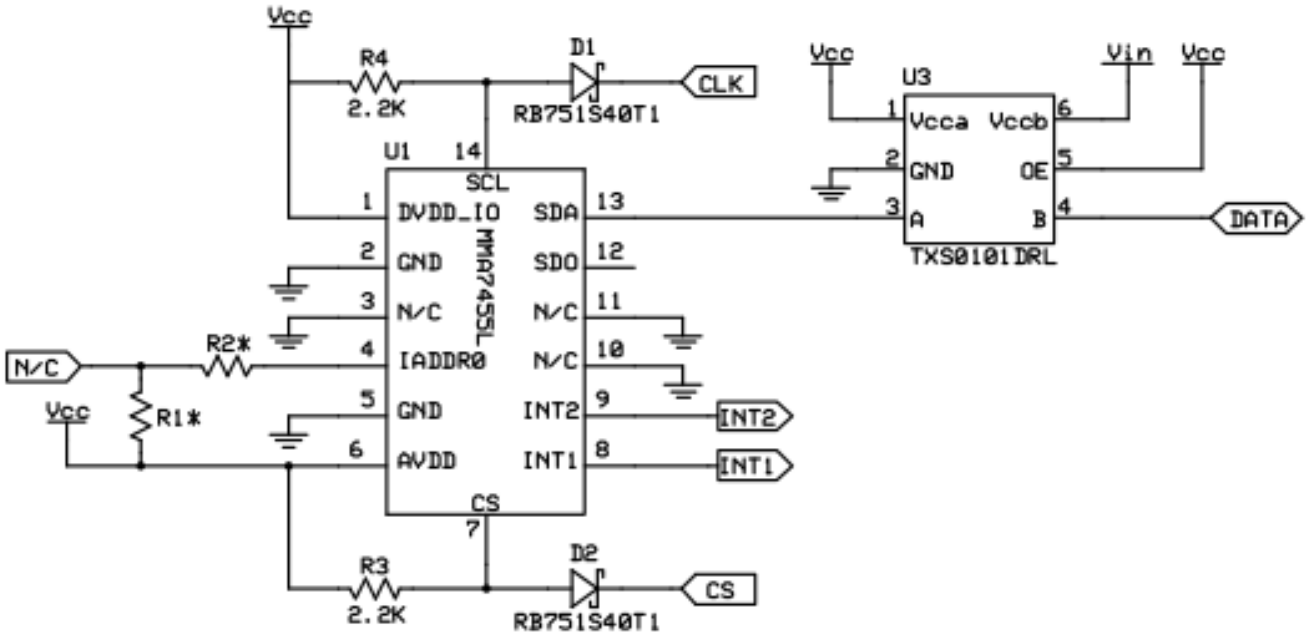
Accel module



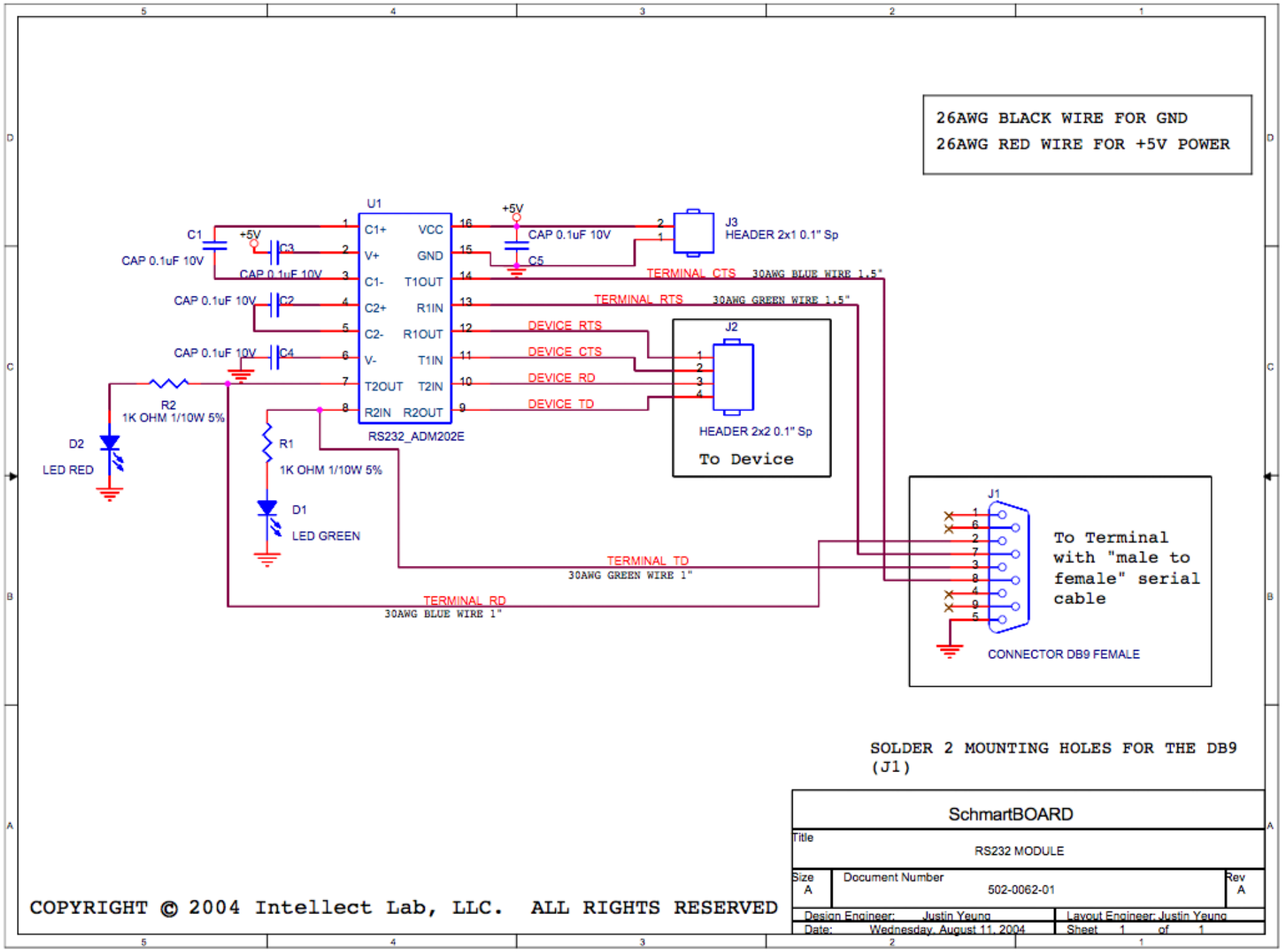
NOTES:
*R1 AND R2 ARE NOT POPULATED. RESERVED FOR FUTURE USE.
PIN 12 OF U1 IS N/C



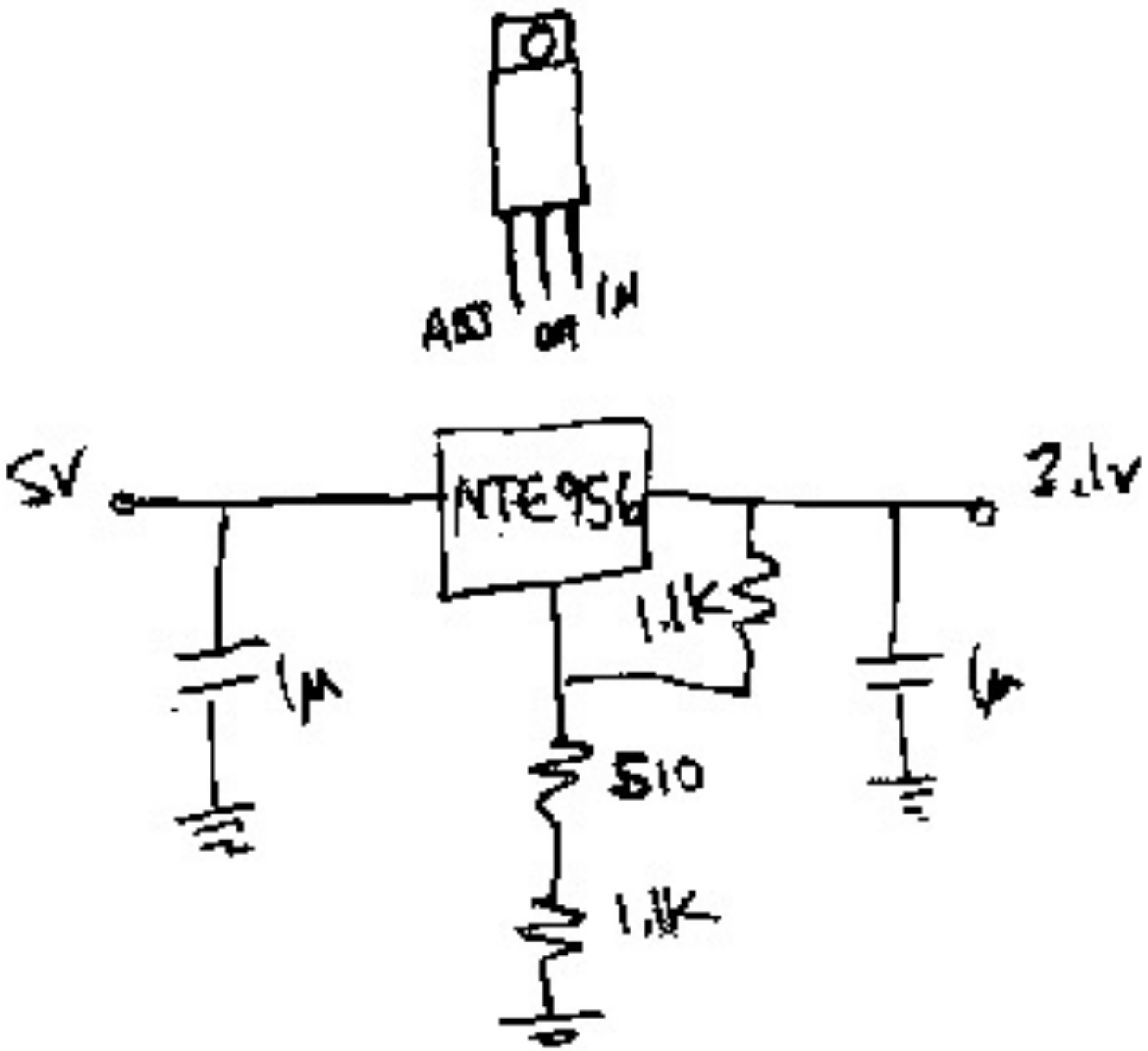




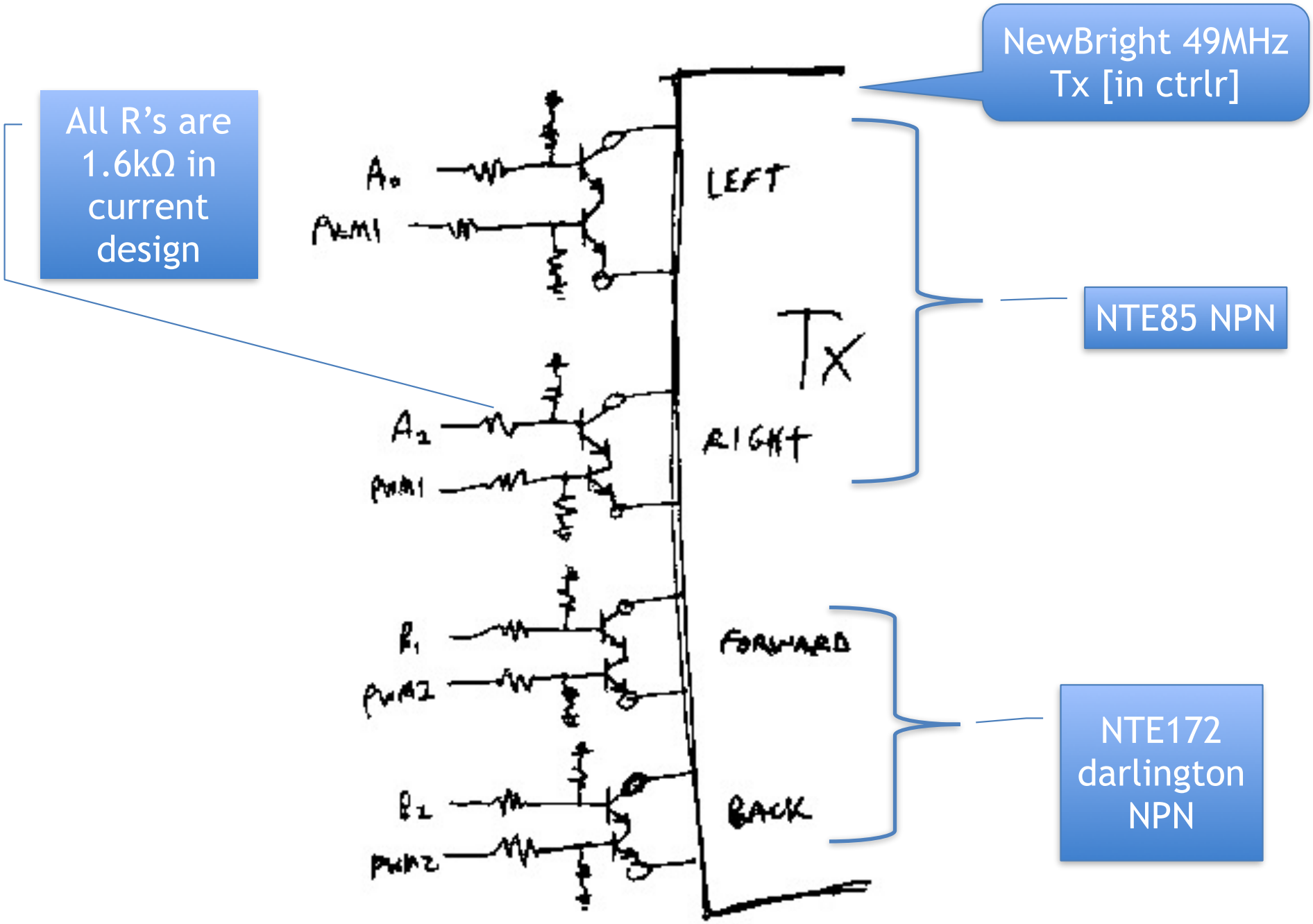
RS232 module



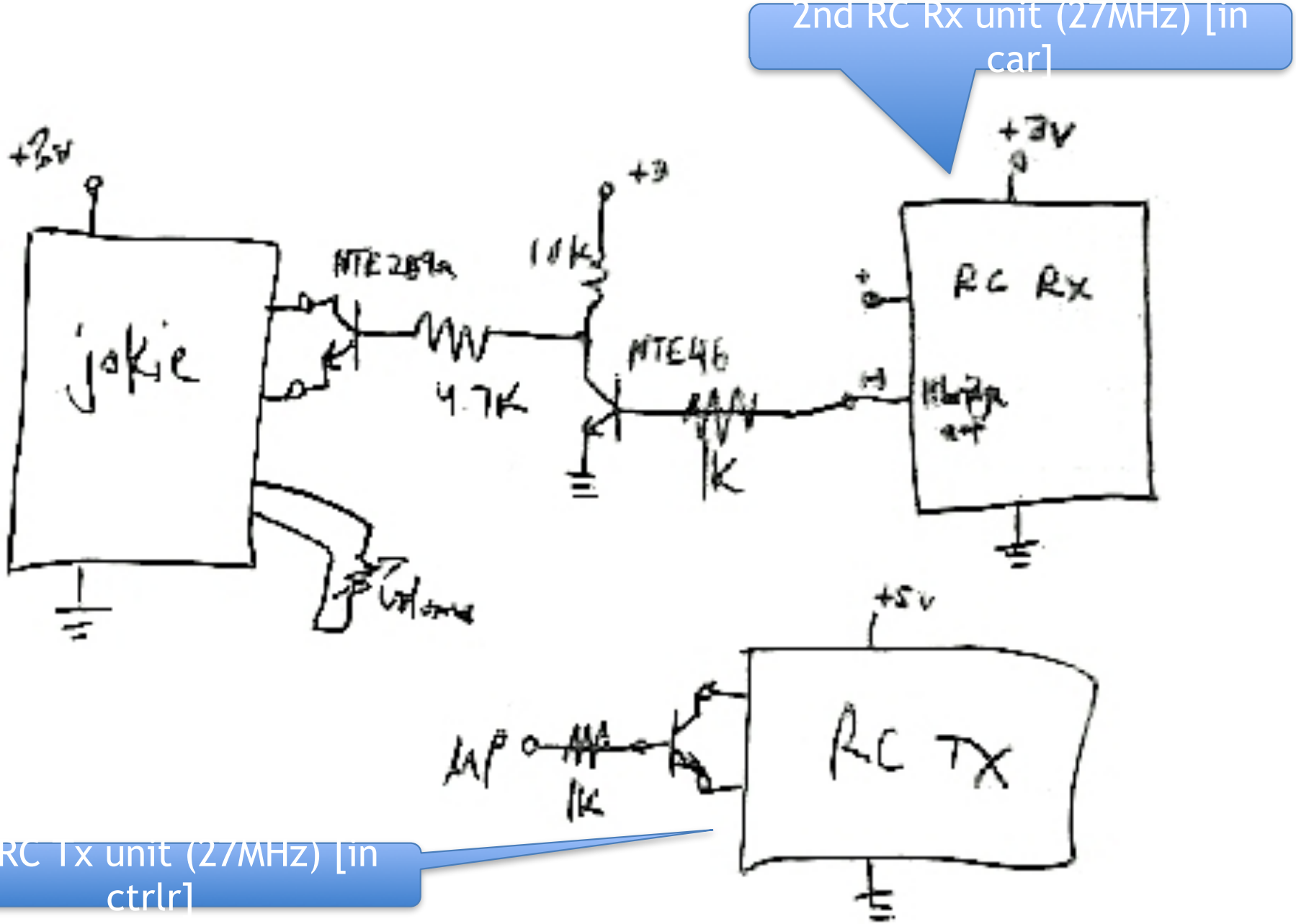
3v regulator for NewBright Tx unit



uP - NewBright Tx unit controller



Joke unit controller



backup

Other considerations

- Use derivatives to determine when device is being touched/moved
  - Useful during calibration - warn user if touched/moved and restart the test
  - Used to tell if user jerks controller suddenly, then the RC car will tell a joke
- User can move the controller at an up/down angle more than they can in the left/right direction,
  - Not comfortable to twist your wrist side/side as much as can done in the up/down direction
- As car speed increases, want to reduce the sensitivity of the ‘y/z’ (left/right) curve
  - If car veers from side to side at increased speed, it will not be able to go fast straight

Tweaking movement configurations

- With so many parameters that control the movement logic, it's difficult to set them correctly
  - Need hands-on experience to make it feel natural
- Created a debug build to perfect the movement to seem as natural as possible and not overly reactive to small movements
  - The 15 control movement variables are transmitted out RS232 in mode 0
  - Pushing the mode button allows selecting each of the control variables and the on-board variable resistor is then used to tweak that value via ADC
  - Pushing the mode button again leaves the affected variable in last state and selects a new variable for modification
  - Once RC car behaves correctly, connect to PC to download the variable stream and update firmware with correct control variables
- Now sets of 15 cfg values can be used to create RC "personalities"
  - Over-reactive/out of control
  - Slow/hesitant
  - Speedy/eager…
- Then during normal driving operation, uP dev.board variable resistor is used to adjust performance from slow to fast speed

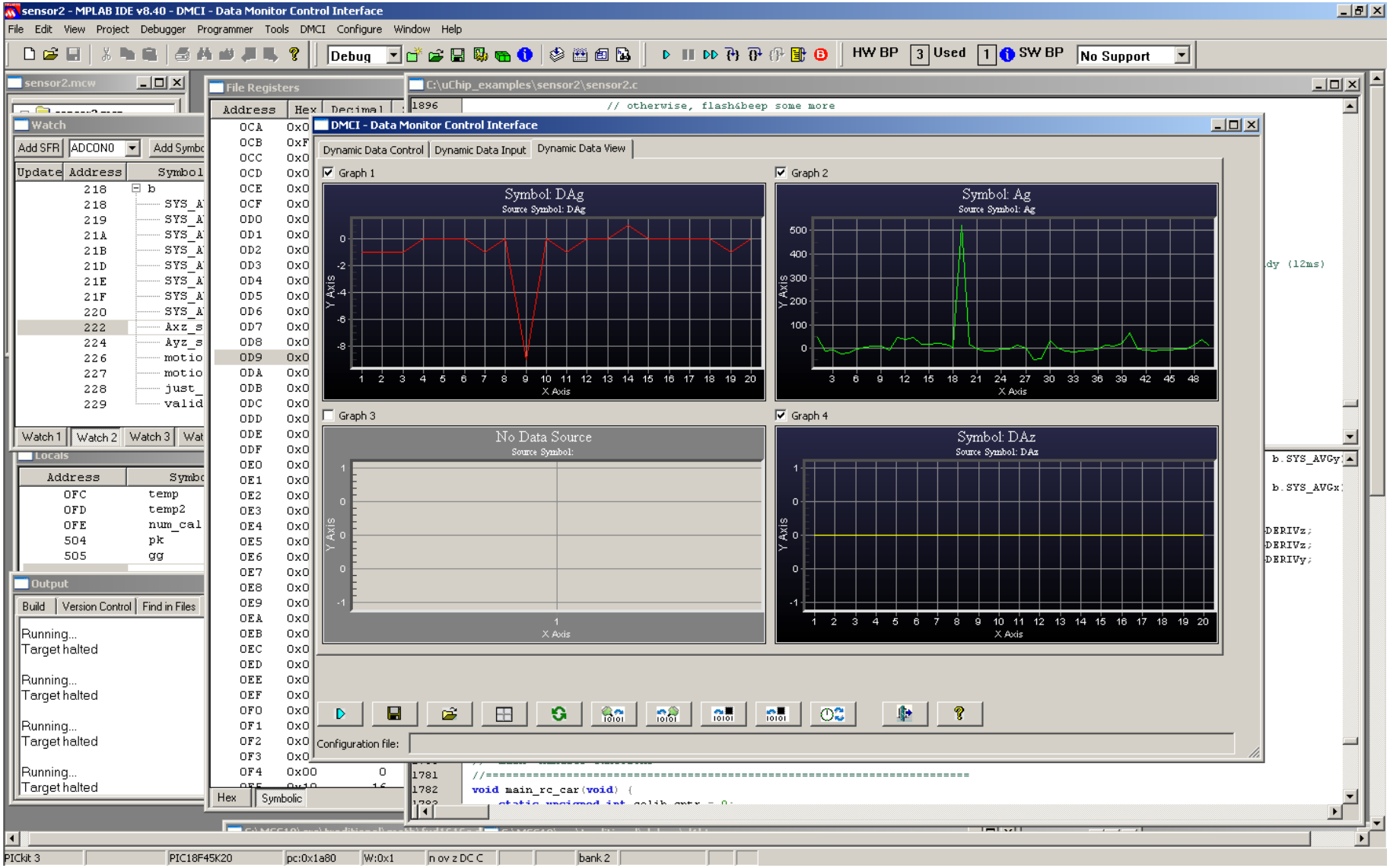
Calibrate & self-tests

- Both accelerometer and gyrometer support self-test mode
  - Setting gyro pin#1(ST) high initiates self-test
  - For accelerometer, set bit#4[ST] in register $16
  - EM field is applied internally to MEMS to affect sensor
  - Firmware reads delta values, die() if not correct
- Accelerometer can store offsets while power is applied
  - Refer to Freescale AN3745 for 0g offset procedure
  - Offset values are lost at power cycle
- Cannot write offsets to the gyrometer, so an “at rest” average value is computed and subtracted from subsequent measurements
- Calibration process iterates 4 times for best result
- Controller works in either ‘sensor up’ or ‘sensor down’ orientations for use as a hand-held, or as a steering wheel
  - Orientation is determined during calibration

- TODO: calibration const -> EEPROM

Plotting of sensor data

- The MPLAB IDE provides “Data Monitor Control Interface” for visualizing data but mainly intended for debugging
  - Provides limited visibility into arrays, data retrieval on breakpoint via PICkit3, limited depth



Post-process sensor data

- To better determine algorithm to detect user motions
  - What is the best way to transmit data to PC?

- Can store sensor data in SRAM and dump via IDE,
  - Using 4 buffers of 250bytes each (1 page), can store 250 samples of each X,Y,Z(@8bit) and 125 samples of gyro(@10bit)
  - Download w/ IDE memory dump, parse with Perl script
  - Time consuming, limited data set (2sec max accel, 1sec max gyro), and not in real time making interpretation of data difficult

- Or can transmit data to PC via RS-232,
  - Fast real-time interface @115kbps with uP @ 4MHz
  - On PC, use kermit for logging data, Perl script to parse
  - For visualization, use generic plot script, LiveGraph, or Matlab

Almost real-time plotting of sensor data

- The sensor data can be plotted in “real-time”
  - Put uP into special debug mode, select one of
    - Raw data, averaged data, averaged derivative data
  - Data sent to PC via serial port
  - Kermit logs incoming data to file
  - Perl script uses File::Tail to parse log file, and outputs CSV file
  - LiveGraph provides real-time ‘tailing’ of a file
  - uP firmware outputs 0x0a(‘LF’) and 0x0c(‘FF’) every ~250 bytes so Perl script sees file is being updated
  - Update time is almost “real-time” but is delayed by several seconds

Note: the data shown here has not been calibrated

To capture sensor data:

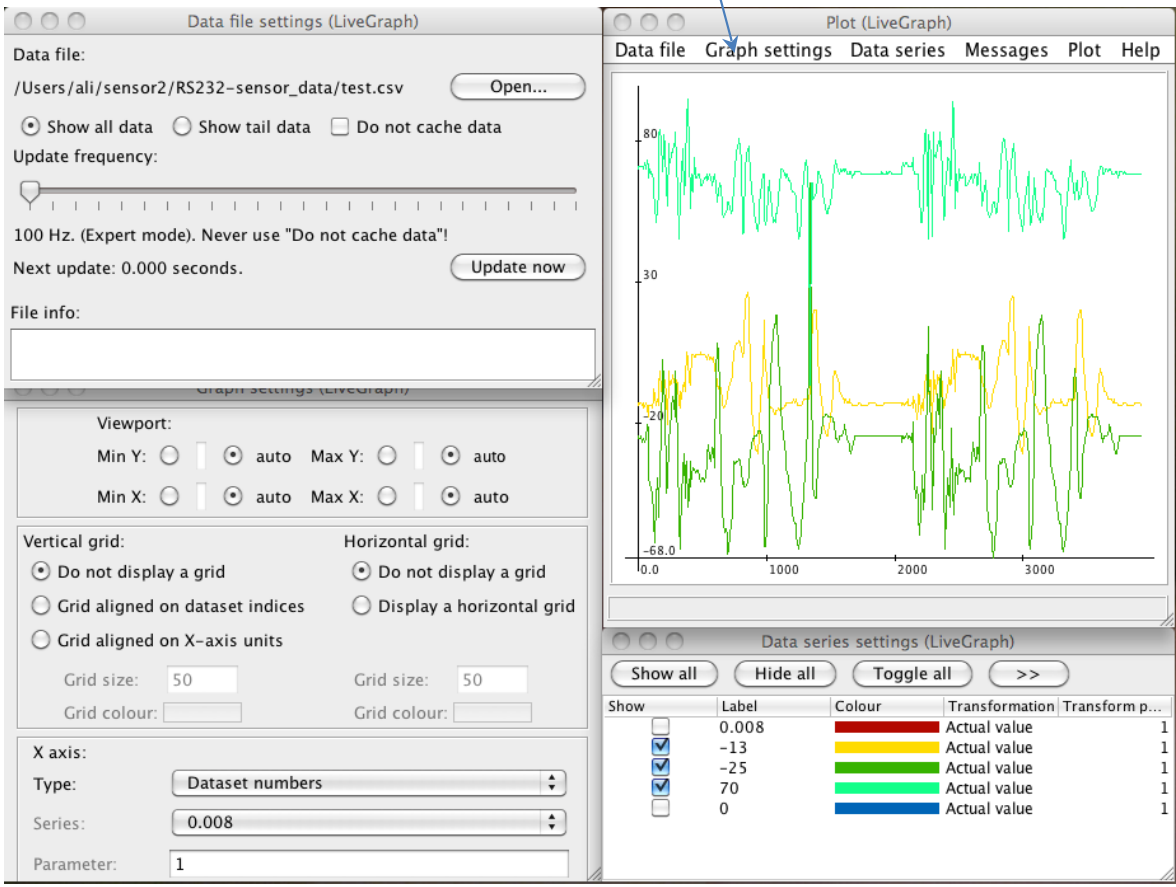
```
$ ./watch.sh -c test.senbin
```

To parse captured sensor data:

```
$ ./watch.sh -p test.senbin
> test.csv
```

• But need to be careful, not all Unix tools can handle this streaming data (waiting for CR?)
• Couldn’t get data piped with,
  - *uniq*
  - *cut*

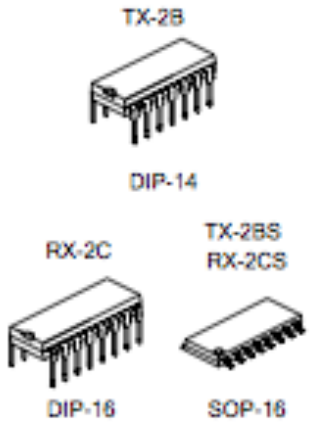
• But still there are plot update delays which make interpreting user motions difficult…



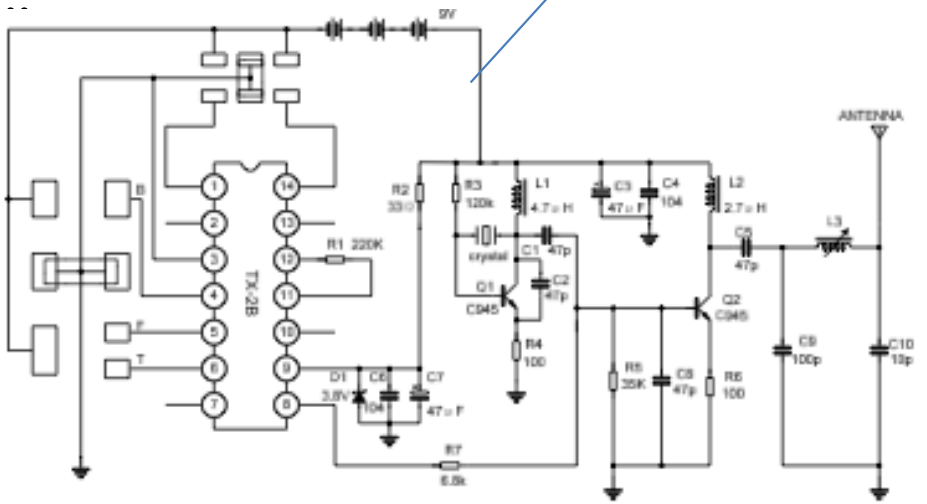
Prior testing w/ 27MHz RC controller

- Some RC cars use low-cost TX-2B/RX-2C chips,
  - With minimal external components, handles radio circuit and provides forward, backward, left, right, turbo signals
  - Operates from 1V – 5V
- Modify transmitter unit to drive PWM to existing on/off switch inputs at varying rates as user moves sensor module controller
- Tested 27MHz RC Tx/Rx units and found about 15Hz update available
  - Time to transmit command code to Rx unit takes ≈ 17ms
- Current RC car unit (New Bright) uses 49MHz
  - Hoping for 2x faster update time…

Silan Microelectronics



Tx unit

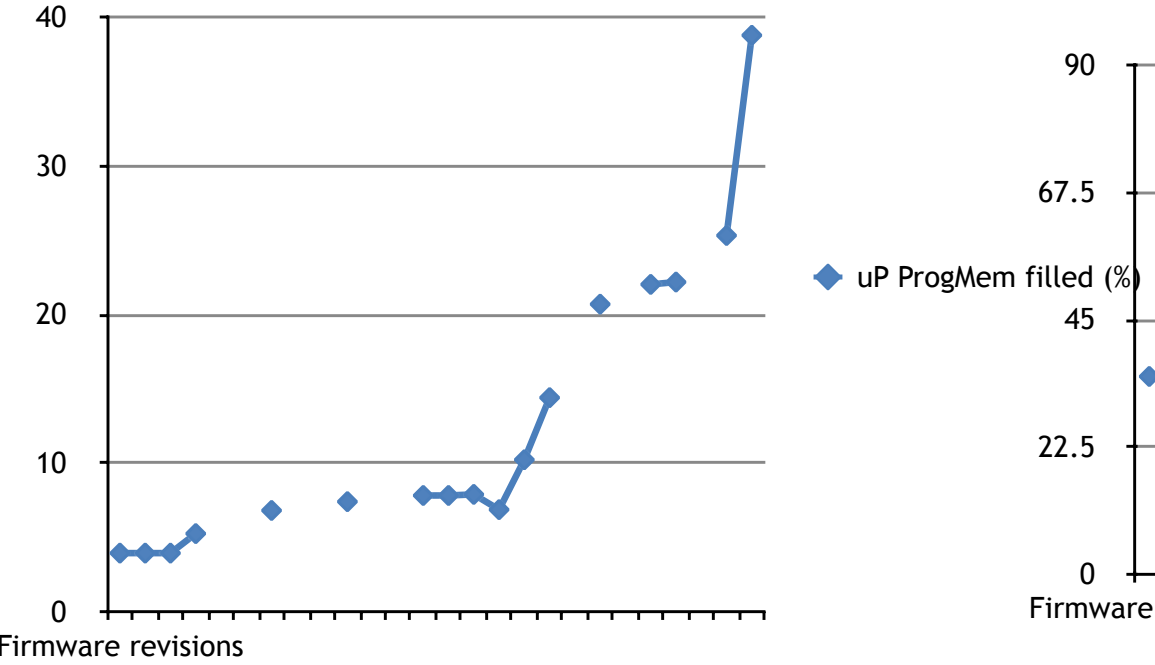


Dev environment

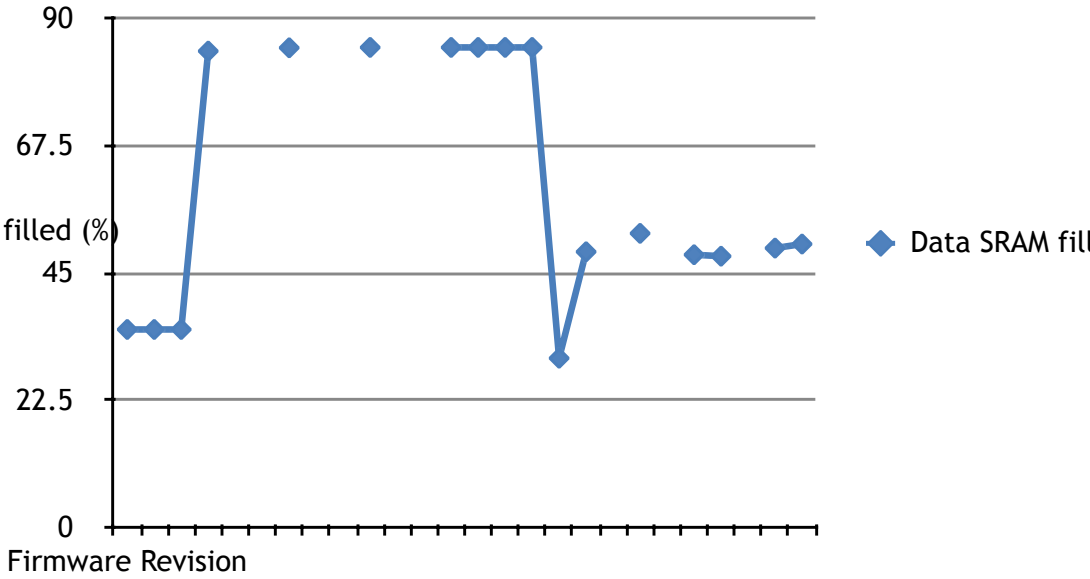
- Mac OS X 10.6.2
  - Bash 3.2.48(1)-release
  - Perl 5.8.9
  - Kermit 8.0.211
  - Live Graph 1.0
- VMware 3.0.1 running Win XP SP3 guest
  - MPLAB 8.40 IDE & MPLAB C18 3.34 compiler
    - mpasmwin.exe v5.34, mplink.exe v4.34, mcc18.exe v3.34
- PICkit3 programmer (USB)
  - PIC18F firmware v1.25.10

Interesting stats...

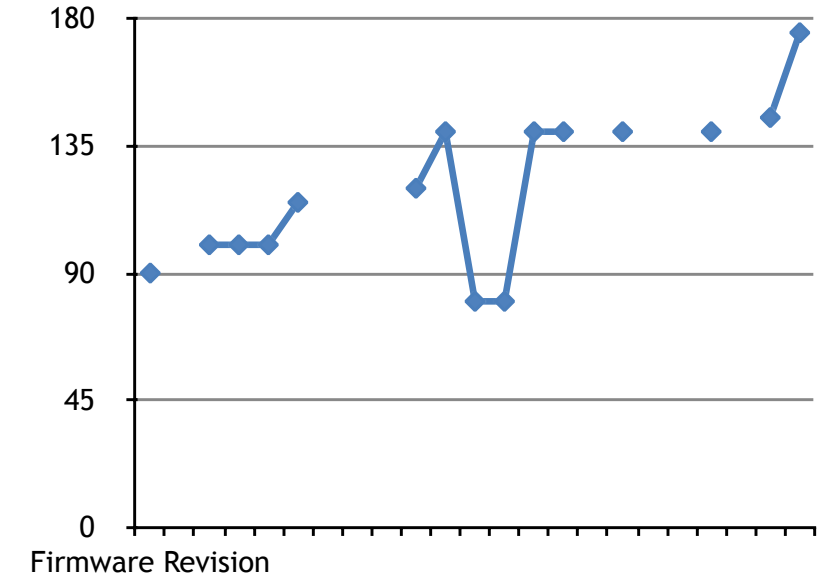
uP ProgMem filled (%)



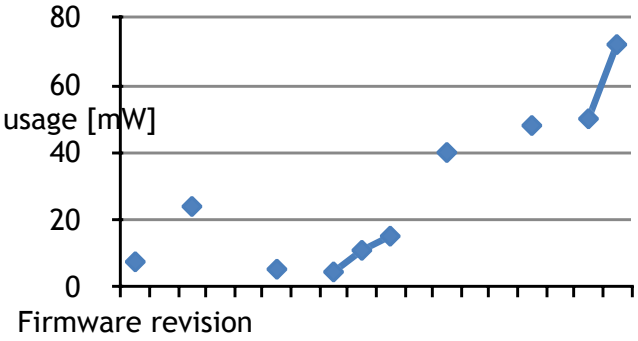
Data SRAM filled (%)



Power usage [mW]



uP busy



Not the same as processing time scope picture shown before. This is AVG voltage/5v (BUSYIO) when doing _anything_ in main(). Very rough estimate