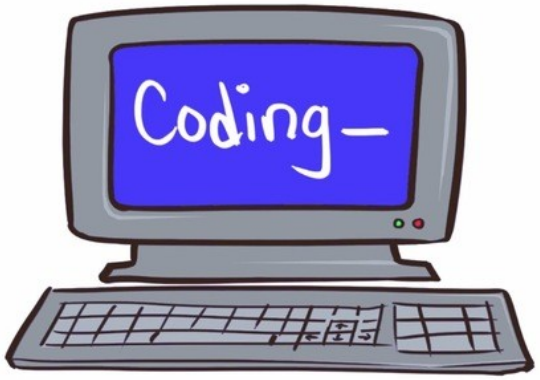
Class 43

EasyOled
&
Quadrature Encoders



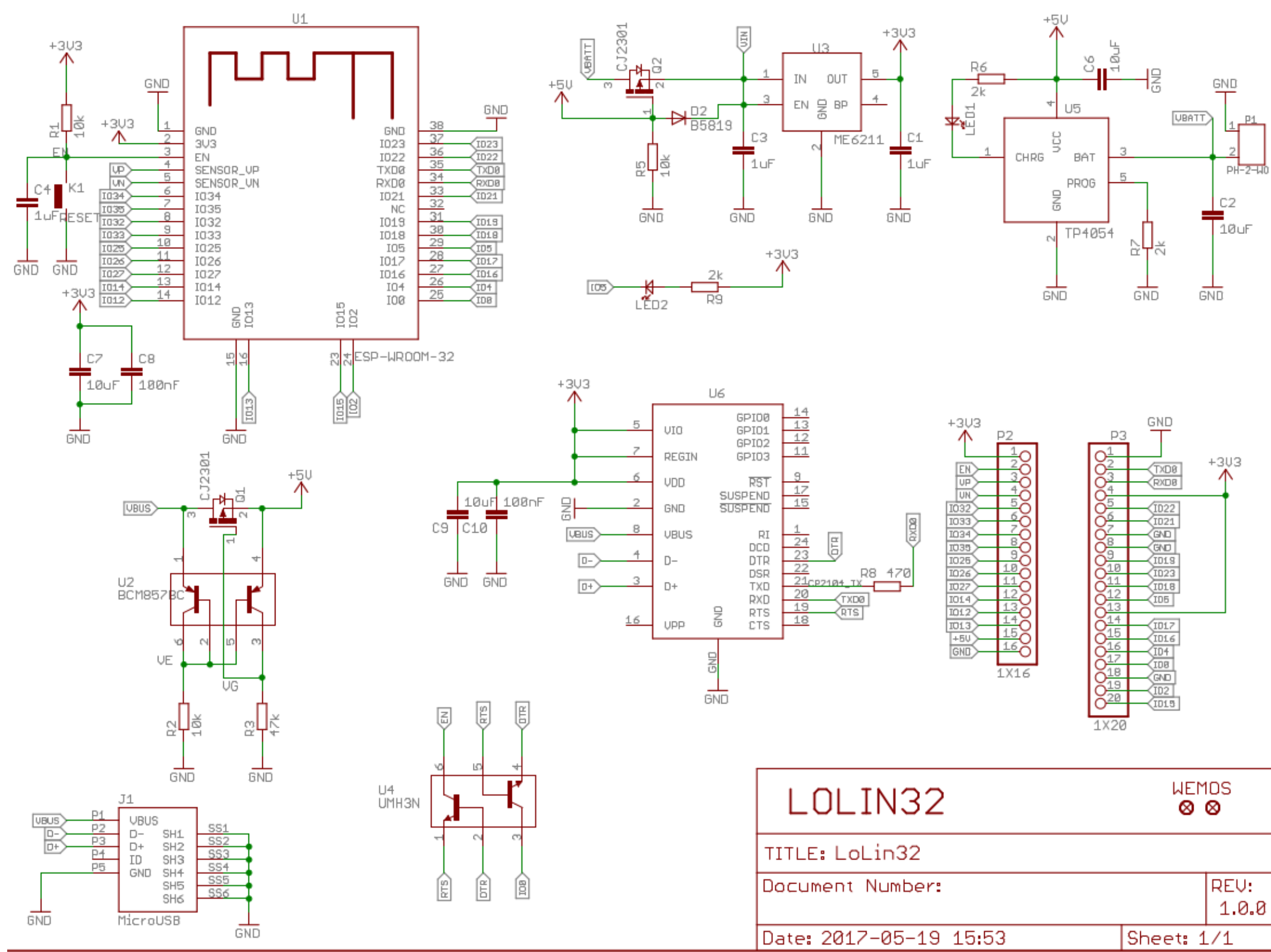
Counting wheel revolutions

3V --
Reset --
A0 #36i --
A3 #39i --
A4 #32 --
A5 #33^ --
A6 #34i --
A7 #35i --
DAC1 A18 #25 --
DAC2 A19 #26 --
A17 #27 --
A16 #14 --
A15 #12 --
A14 #13 --
5V --
Gnd --

GPIO pins marked i are input only. Others in/out

Output pins marked ^ do not support AnalogWrite32 lib (pwm)

-- Gnd
-- #01^ TX (ie USB)
-- #03^ RX (ie USB)
-- 3V
-- #22^ SCL (ie I2C)
-- #21^ SDA (ie I2C)
-- Gnd
-- Gnd
-- #19 MISO
-- #23 MOSI
-- #18 SCK
-- #05 SS (pullup LED_BUILTIN)
-- 3V
-- #17
-- #16
-- #04 A10
-- #00^ A11 (pullup. LO on boot = reflash)
-- Gnd
-- #02 A12 (pulldown?)
-- #15 A13



LOLIN32

WEMOS

TITLE: LoLin32

Document Number:

REV: 1.0.0

Date: 2017-05-19 15:53

Sheet: 1/1

GPIO						lolin32
0	boot	adc11	touch1		pu	spare
1	TX0					(tx0)
2		adc12	touch2		pd	**Spare**
3	RX0					(rx0)
4		adc10	touch0		pd	**butn3**
5					pu with led	(led)
12	HSPI miso	adc15	touch5	TDI	pd	**mot1**
13	HSPI mosi	adc14	touch4	TCK		**mot2**
14	HSPI SCK	adc16	touch6	TMS		**mot3**
15	HSPI ss	adc13	touch3	TDO	pu	**mot4**
16				RX2		**mot5**
17				TX2		**mot6**
18	VSPI sck					**echo2**
19	VSPI miso					**butn1**
21				SDA		**sda**
22				SCL		**scl**
23	VSPI mosi					**butn2**
25		adc18		DAC1		**trig**
26		adc19		DAC2		**trig2**
27		adc17	touch7			butn4
32		adc4	touch9			**svo1**
33		adc5	touch8			**svo3**
34		adc6			in only	**bumper**
35		adc7			in only	**ir rx**
36		adc0		VP	in only	**ldr**
39		adc3		VN	in only	**echo**
EN	reset				pu	

How we will assign those available GPIOs (last column)

Refer to matching info in myconfig.h

SOME OF THESE ARE DIFFERENT FROM EARLIER !

Easyoled

Builds on standard oled library SSD1306 (ie esp8266-oled-ssd1306)
That gave us a “display” object that had zillions of options for displaying.

We now add a new object we will call “oled” that incorporated “display”, but that just offers a few pre-formatted displaying styles.

Easyoled.h in our myconfig library area.

Reason:
Easy consistent display modes to use.

Easyoled

oled.blank();

oled.msgbox(headertext, msg1text, msg2text, msg3text);
up to 3 messages with a header

oled.yell(shortmessage1text [, shortmessage2text]);
one or two words in large font
[Alternative: BOTH arguments may be int number.]

oled.jnl(newjnltext);
journal line is pushed onto message stack from bottom
if text reads blank, whole stack of 4 is blanked

oled.bar(titletext, unitstext, barpercent [, unitsdisplay]);
bar scaling is percent (0 - 100).

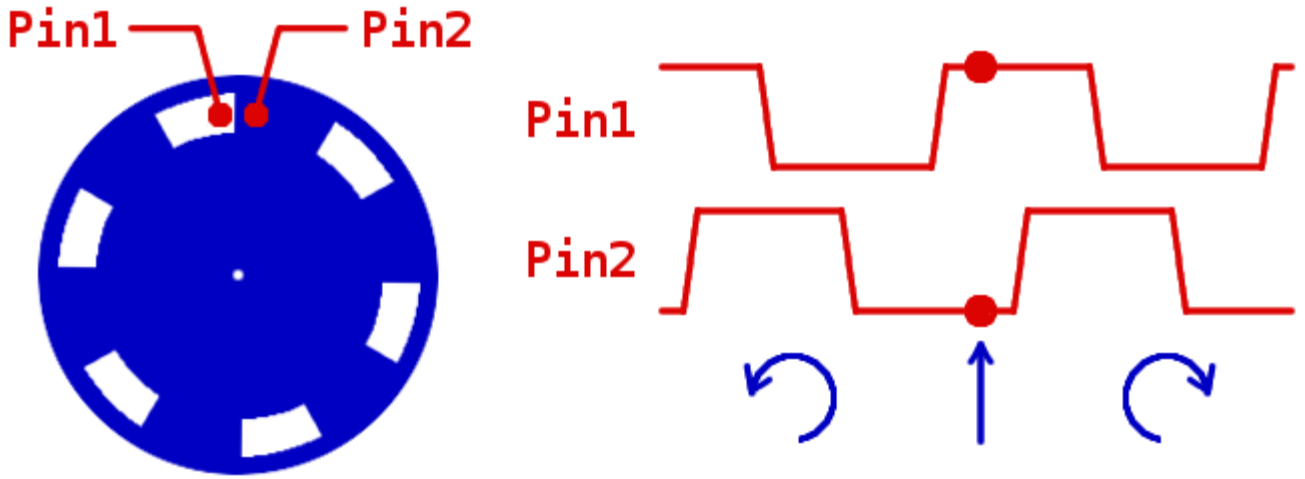
Easyoled

Use easyoleddemo.h

Yell() MsgBox() Bar() and Jnl()
Are the same 4 simple oled patterns we
used earlier in python, javascript and lua.

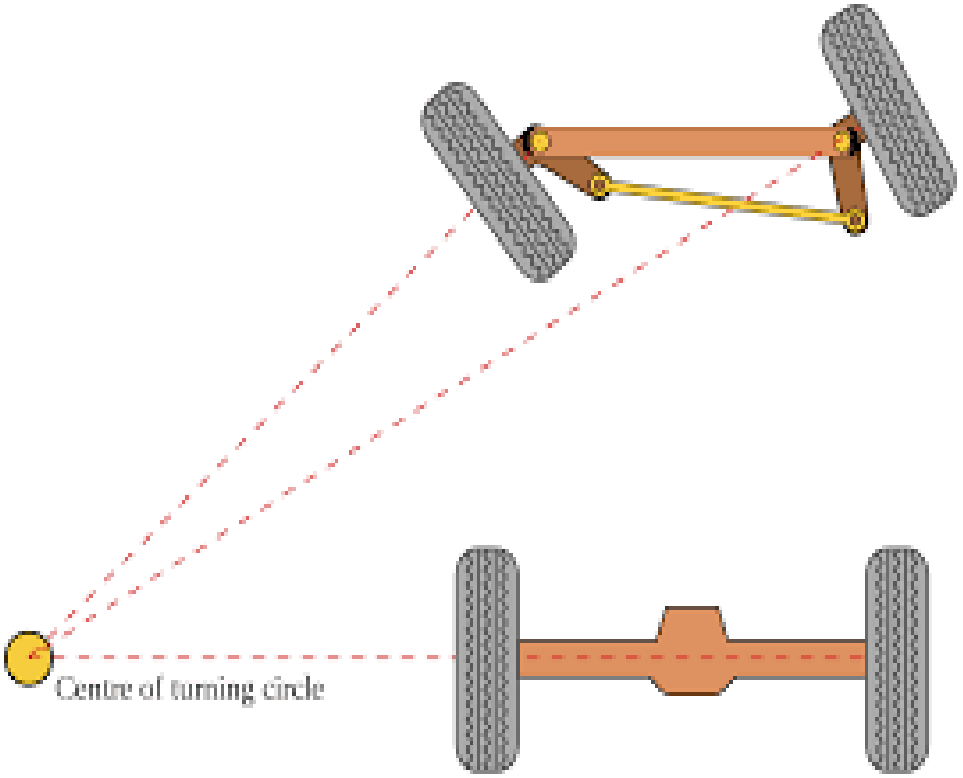
Quad Encoder

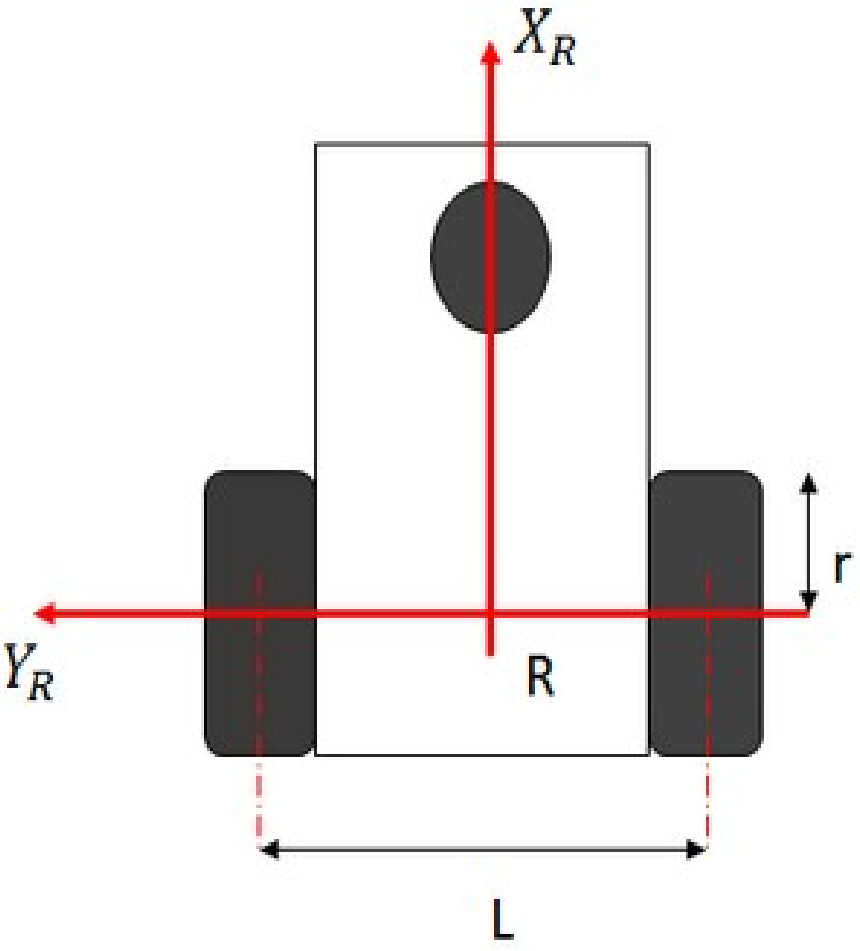
https://www.pjrc.com/teensy/td_libs_Encoder.html



Or http://www.creative-robotics.com/quadrature-intro

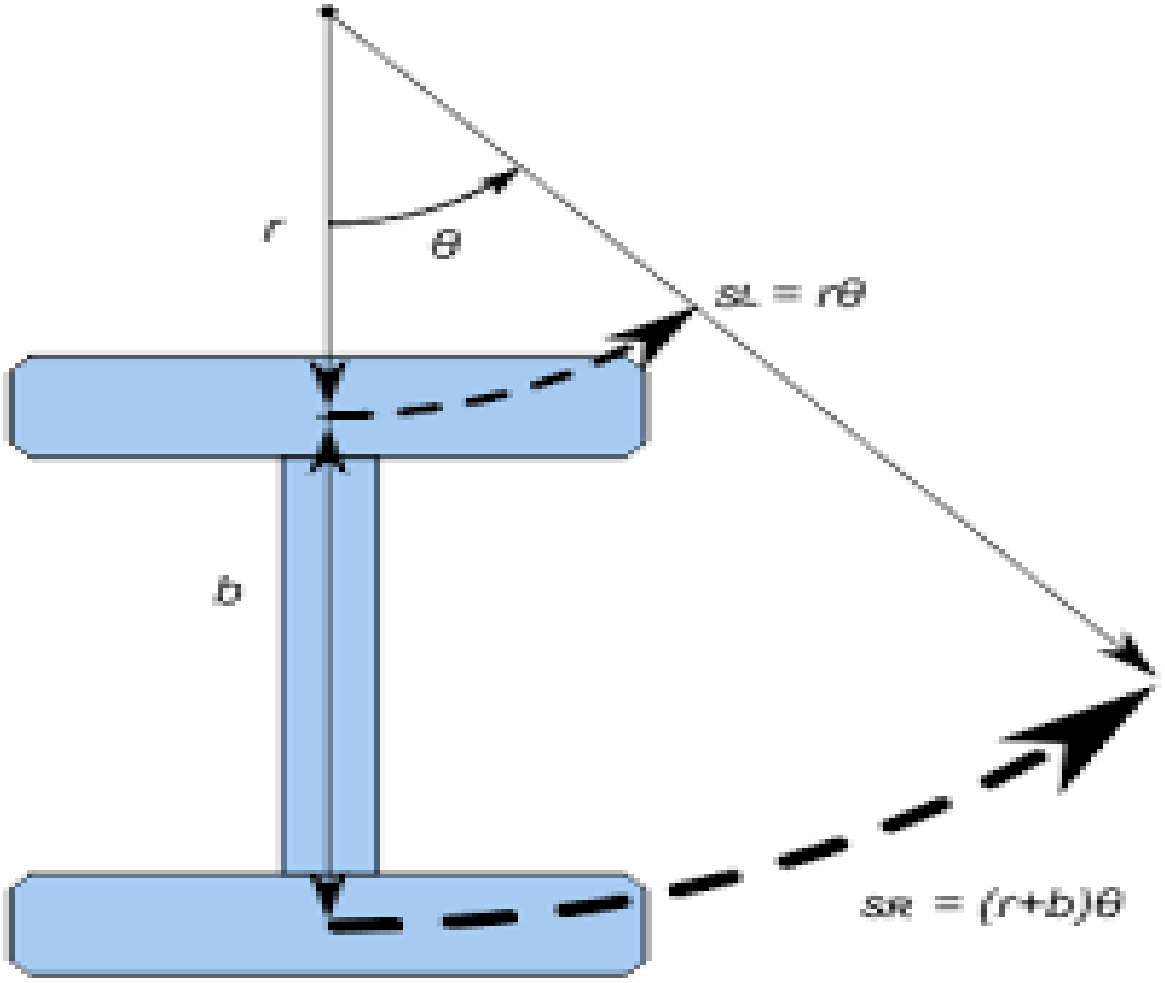
Two Robot Wheel Types





https://en.wikipedia.org/wiki/Ackermann_steering_geometry
https://robotics.stackexchange.com/questions/8911/how-to-find-kinematics-of-differential-drive-caster-robot

If we can count the revolutions (ie distance) of both wheels separately,
We might calculate both distance and new direction



http://ros-developer.com/2017/11/09/ackermann-steering-car-robot-model-with-simulation-in-gazebo/

Connections for today:

BUTTON1 gpio19
BUTTON2 gpio23

BUTTON3 gpio27
BUTTON4 gpio4

Each detector occupies
a “button” gpio.
Supply either 3V or 5V

Wheel encoders

Library:
QuadEncoder:
    1Wcounter.ino   - 1 detector 1 wheel
    1Wquad.ino   - 2 detectors 1 wheel

And also find sketch:
    2Wquad_easyoled.ino

Quad Encoder Matrix

```
{  0, -1,  1,  0 }
{  1,  0,  0, -1 }
{ -1,  0,  0,  1 }
{  0,  1, -1,  0 }
```

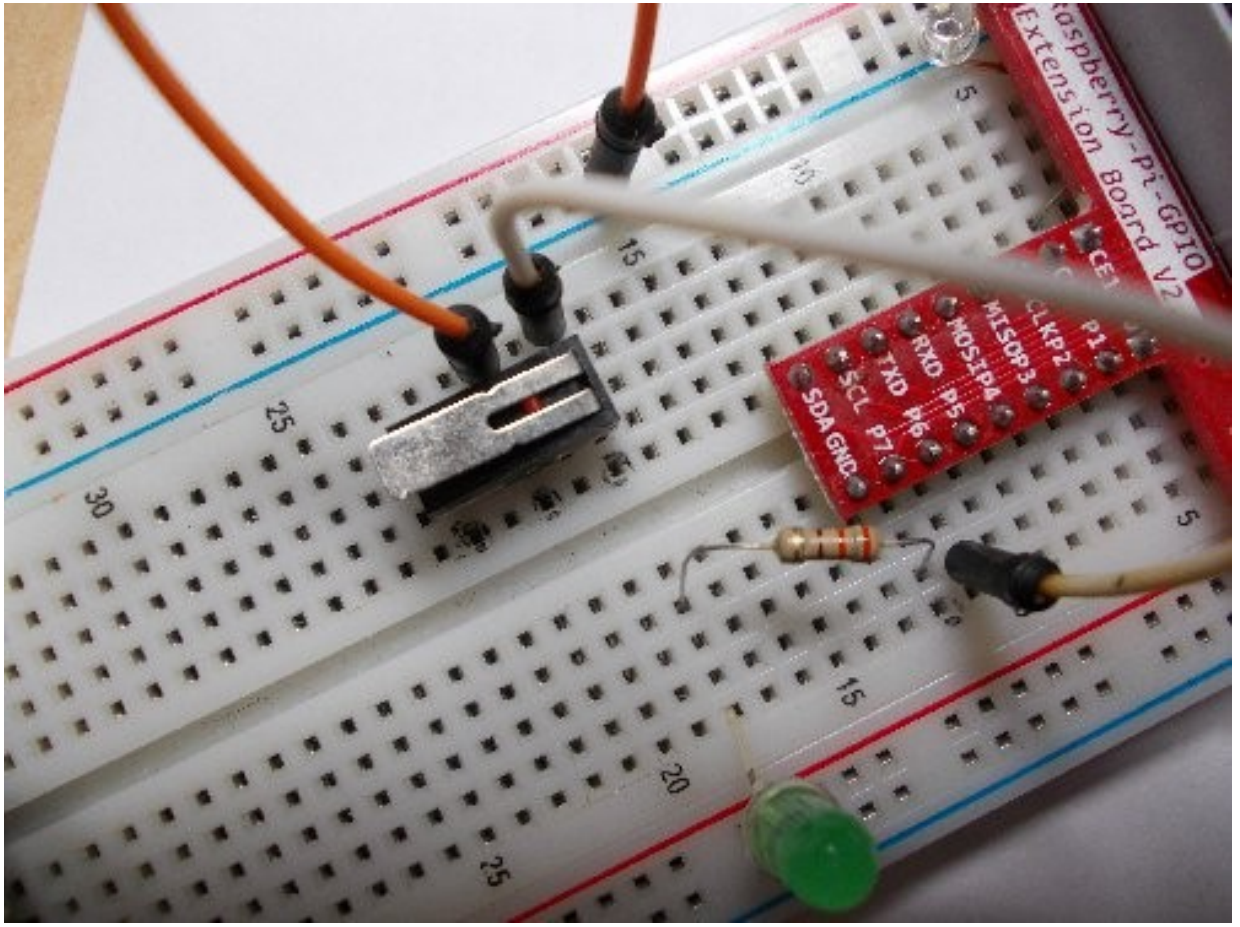
There are 4 possible states of 2 inputs (2 ** 2 = 4)
Depending on **latest "state"** of 2 inputs (4 possibilities),
versus **last known state** (4 possibilities),
We can calculate onto one of those matrix cells (4 x 4),
To tell us to increment or decrement our counter.

Quad or simple?

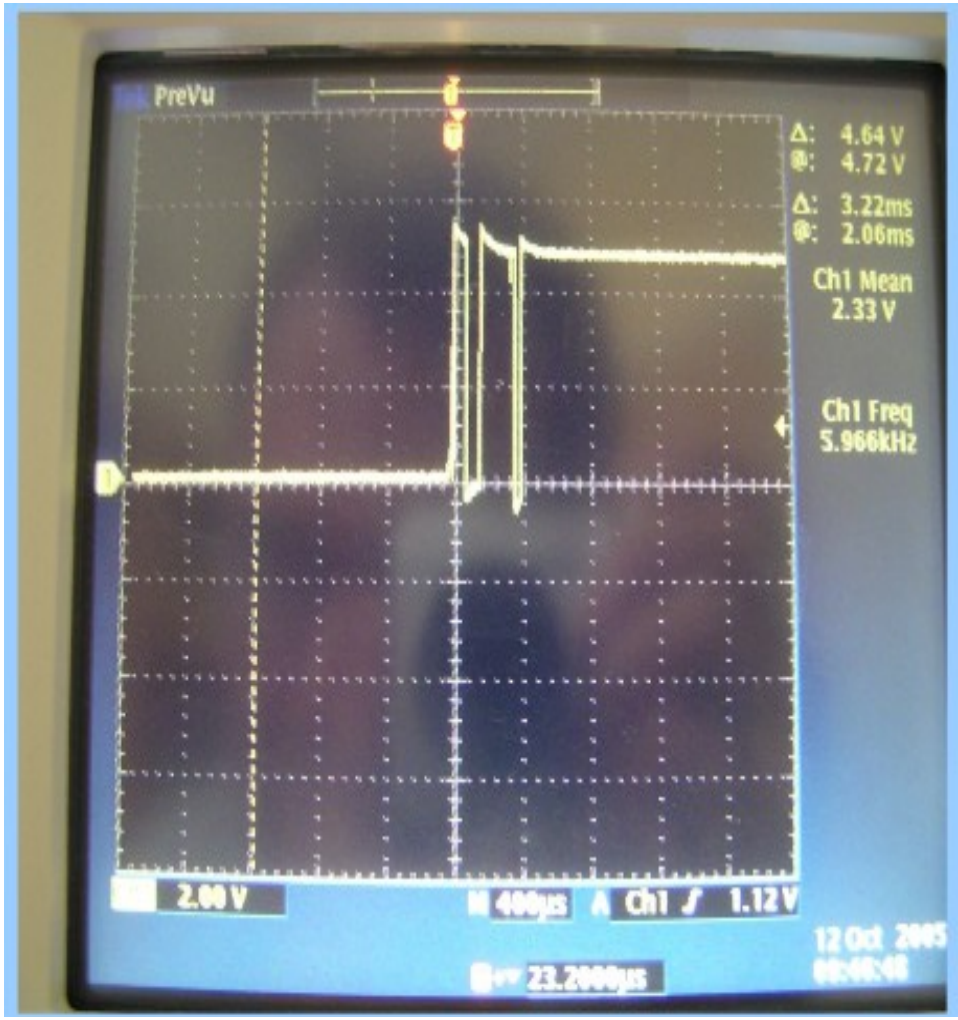
Quadrature Encoder uses 2 inputs (“90 degrees apart”),
… and will reliably count up and down.
Simple encoder/counter uses one input,
… and can only count upwards.

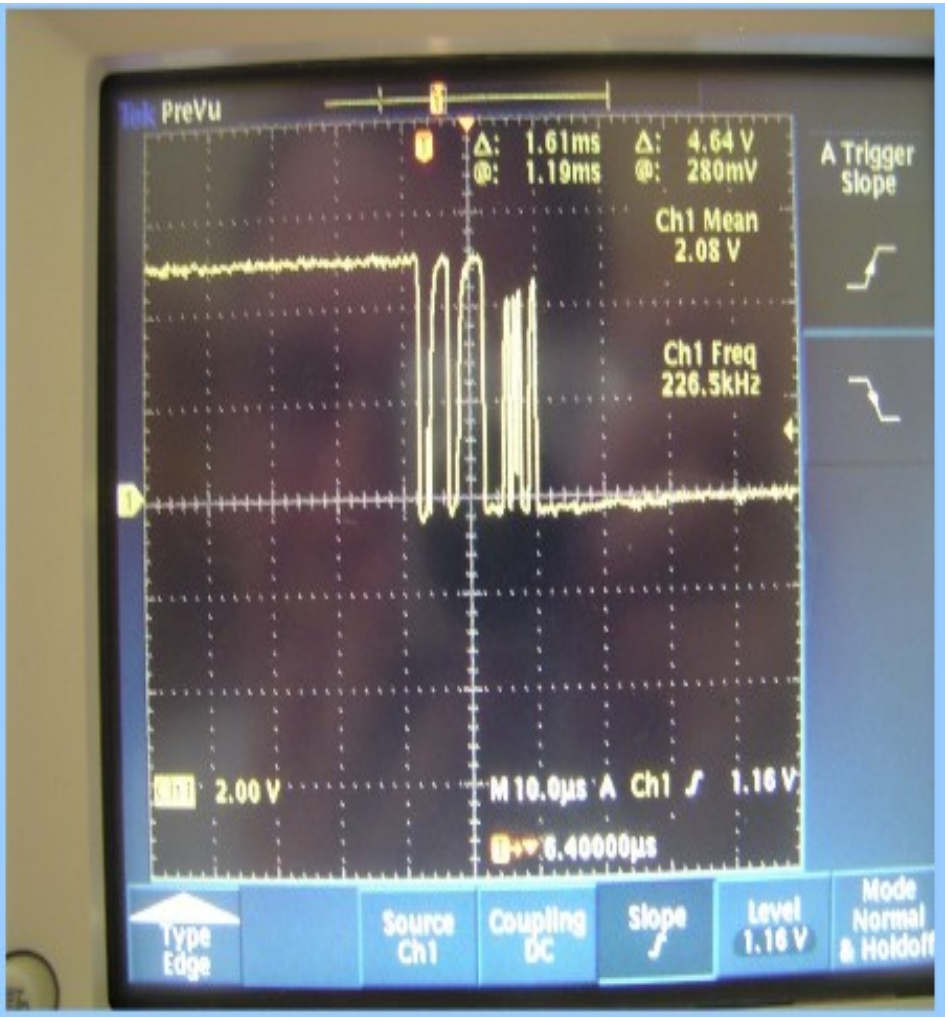
Quad encoder is immune to switch bounce.
Simple counter can count every “bounce” of contacts,
… and then a “debounce” function is useful.

Typical mechanical “limit switch”



Switch Bounce





Switch Bounce

The CPU is usually fast enough to see all the transitions

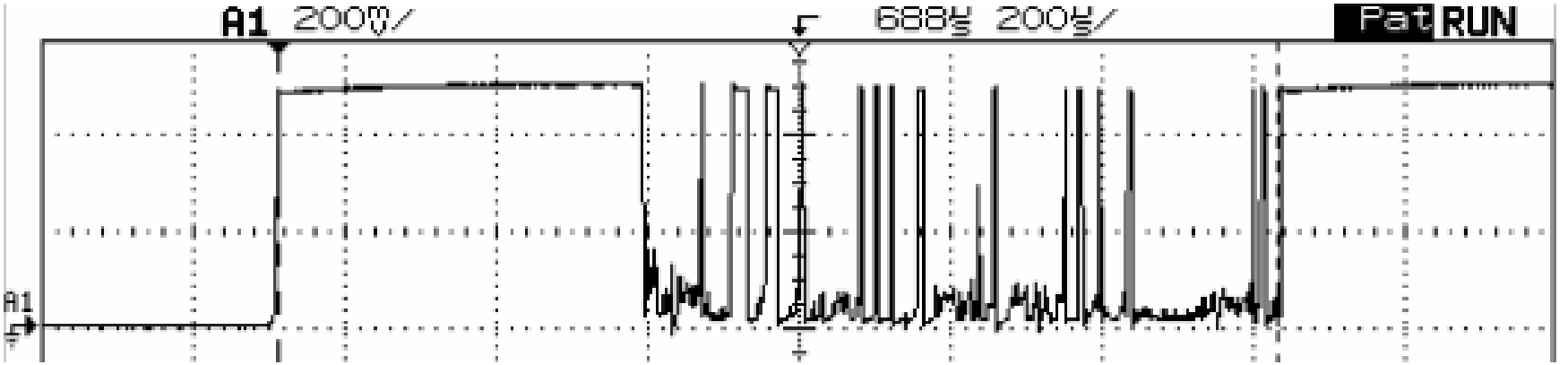
- program acts on multiple transitions instead of a single one

Oscilloscope traces show bounce durations of 10-1000us

- a 100uS bounce could be sampled as high/low 100’s of times
- results are incorrect behavior as seen by user

Characteristics of switch bounce:

- nearly all switches do it
- the duration of bouncing and the period of each bounce varies
- switches of exactly the same type bounce differently
- bounce differs depending on user force and speed
- typical bounce frequency is .1-5ms



This is **one** recorded release of a switch. It was 1300 microsecs (1.3 mSec) before this bouncing stopped!

Concurrency?

The Arduino code model is single-threaded.
Everything that happens is whatever loop() codes.

Delay in loop()

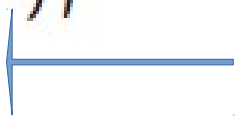
```
void setup() {
    Serial.begin(SERIAL_BAUD);

}

int cntr = 0;

void loop() {
    // ... and call some other regular functions

    Serial.println(cntr++);
    delay (1000);
}
```



Delay in loop()

- Other functions in loop() must WAIT while the delay finishes. (ie everything else freezes.)

This is often quite unacceptable

- loop() should always be free to cycle around at top speed (thousands of loops per second)

With a timer ...

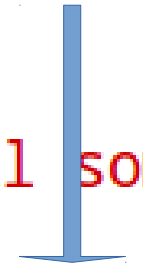
```
#include "ZTimer.h"
ZTimer Timer1;

void setup() {
    Serial.begin(SERIAL_BAUD);
    Timer1.SetWaitTime(100);
    Timer1.ResetTimer(true);
}

int cntr = 0;

void loop() {
    // ... and call some other regular functions

    if(Timer1.CheckTime())  // true once every 100mSec
        Serial.println(cntr++);
}
```



Next week

Avoiding delay loops the “fancy” way:

Intro to multitasking with a RTOS – Realtime OS

Several functions can be executing simultaneously.

ESP32/Arduino code is built upon Free-RTOS.