Electronics
Pulse Width Modulation Sensors

Terry Sturtevant

Wilfrid Laurier University

August 13, 2020

Pulse Width Modulation Sensors

Pulse Width Modulation Sensors

- *Analog* information can be communicated over *digital* signals

Pulse Width Modulation Sensors

- *Analog* information can be communicated over *digital* signals

  This can be done by varying the *width* or *spacing* of digital pulses

Pulse Width Modulation Sensors

- *Analog* information can be communicated over *digital* signals

  This can be done by varying the *width* or *spacing* of digital pulses
- This is called **P**ulse **W**idth **M**odulation, PWM

Pulse Width Modulation Sensors

- *Analog* information can be communicated over *digital* signals

  This can be done by varying the *width* or *spacing* of digital pulses
- This is called **P**ulse **W**idth **M**odulation, PWM
- This document gives a few examples.

Shaft encoders

Shaft encoders

- *Absolute* position sensing

Shaft encoders

- *Absolute* position sensing doesn't use PWM

Shaft encoders

- *Absolute* position sensing doesn't use PWM
- *Incremental rotary* encoding

Shaft encoders

- *Absolute* position sensing doesn't use PWM
- *Incremental rotary* encoding uses PWM

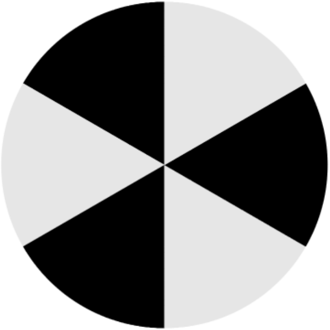
Shaft encoders

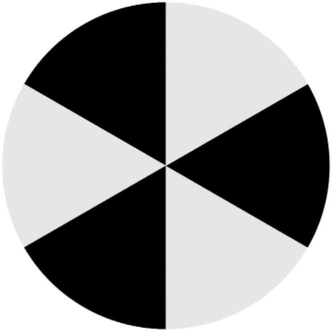
- *Absolute* position sensing

  doesn't use PWM
- *Incremental rotary* encoding

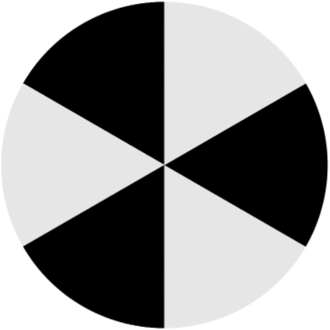
  uses PWM

As long as you know the initial position, you can update if you can sense changes.

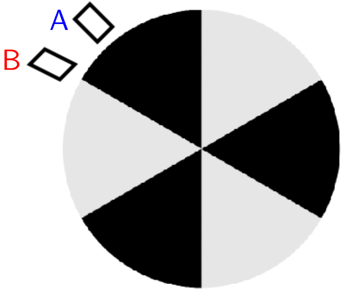




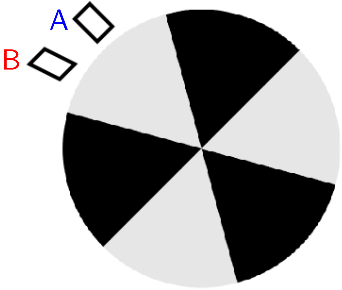
- Shaft encoder wheel



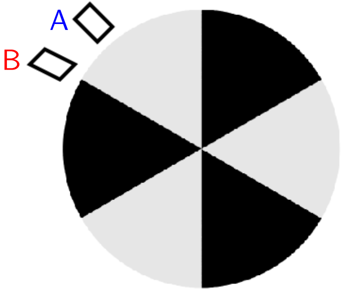
- Shaft encoder wheel
- Two sensors will allow determination of rotation *speed* and *angle*



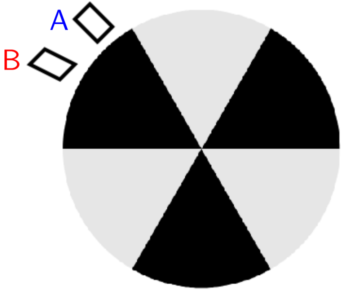
- Clockwise



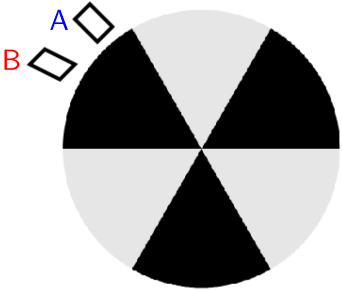
- Clockwise



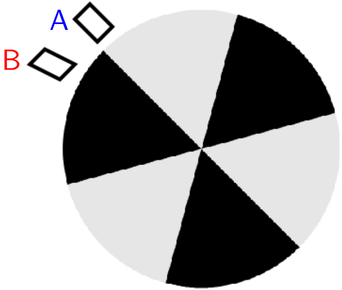
- Clockwise



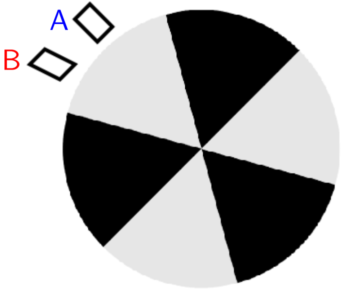
- Clockwise



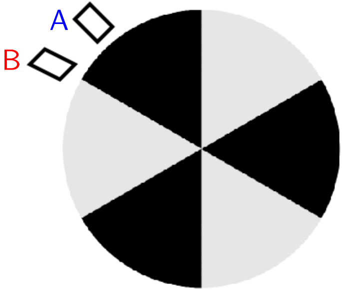
- Counter-clockwise



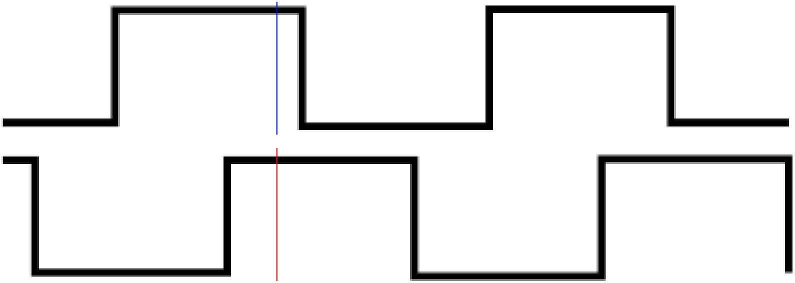
- Counter-clockwise



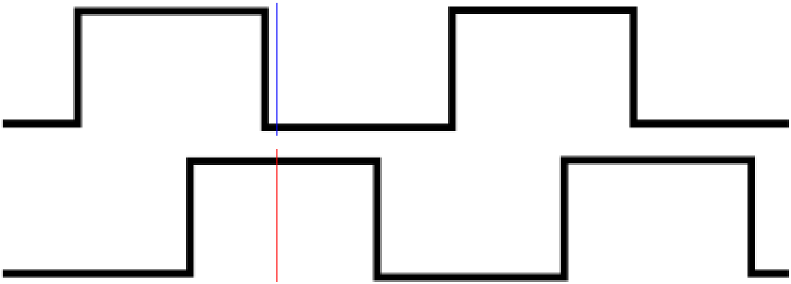
- Counter-clockwise



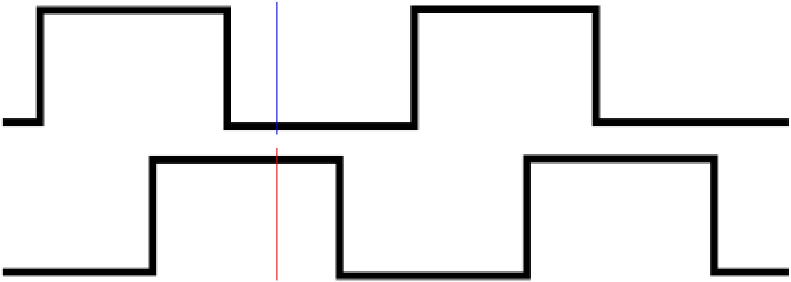
- Counter-clockwise



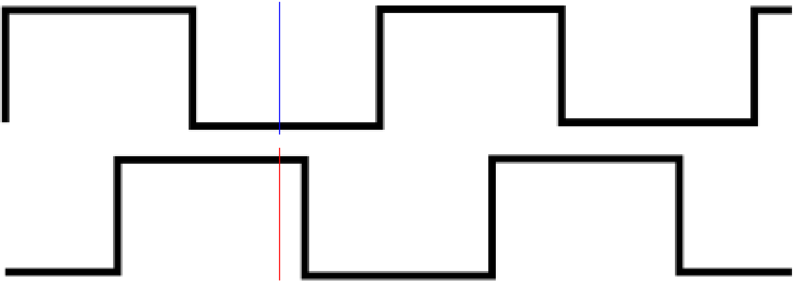
- Shaft encoder timing
- 1
- 1



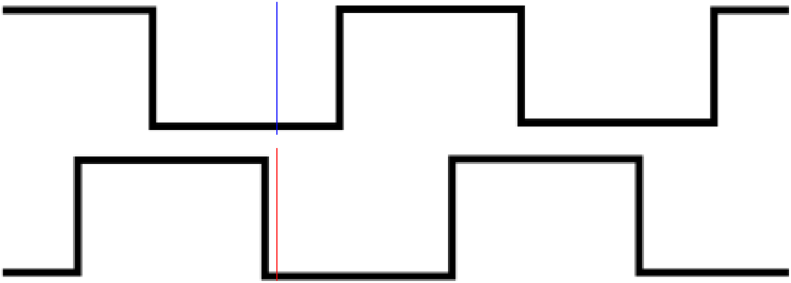
- Shaft encoder timing
- 0
- 1



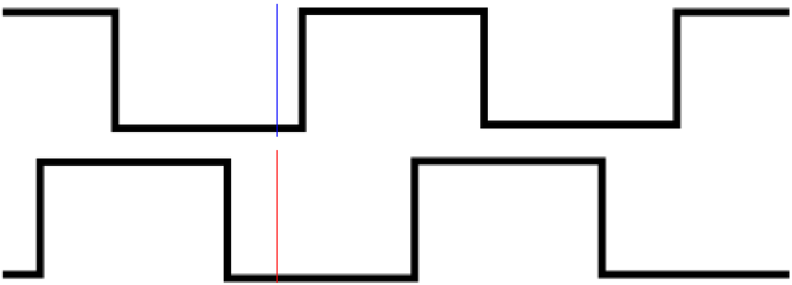
- Shaft encoder timing
- 0
- 1



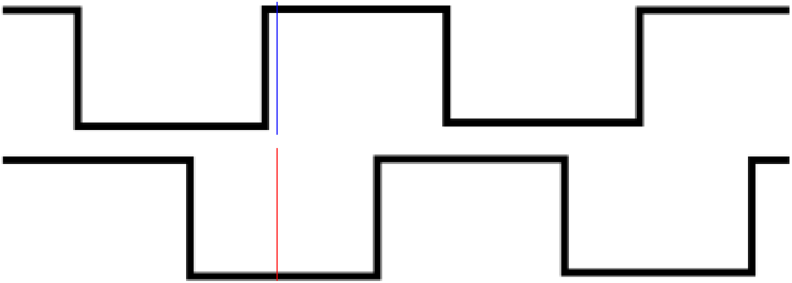
- Shaft encoder timing
- 0
- 1



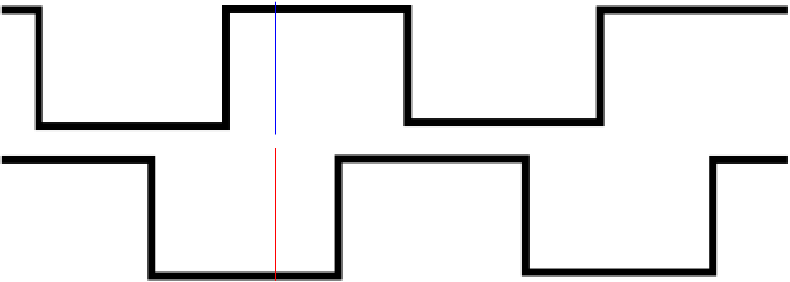
- Shaft encoder timing
- 0
- 0



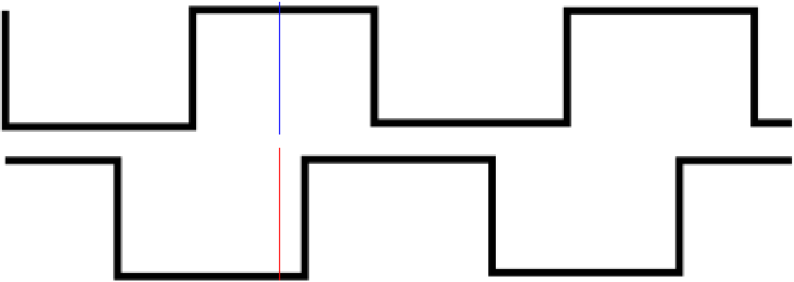
- Shaft encoder timing
- 0
- 0



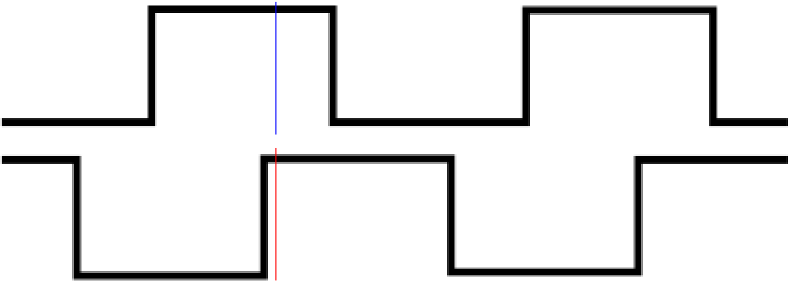
- Shaft encoder timing
- 1
- 0



- Shaft encoder timing
- 1
- 0



- Shaft encoder timing
- 1
- 0



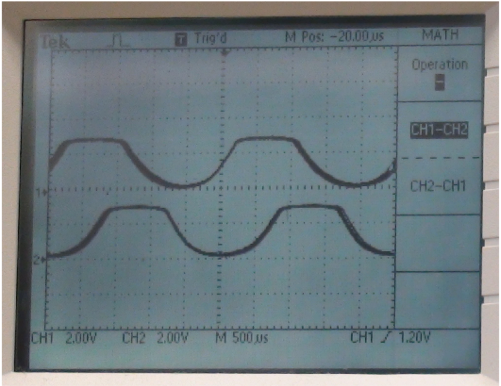
- Shaft encoder timing
- 1
- 1

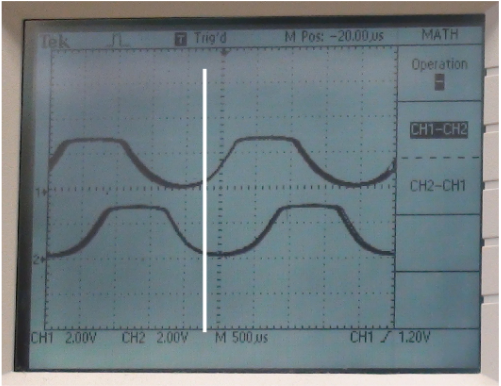
- *Speed* of rotation from frequency of either channel

- *Speed* of rotation from frequency of either channel
- *Angle* of rotation from combination

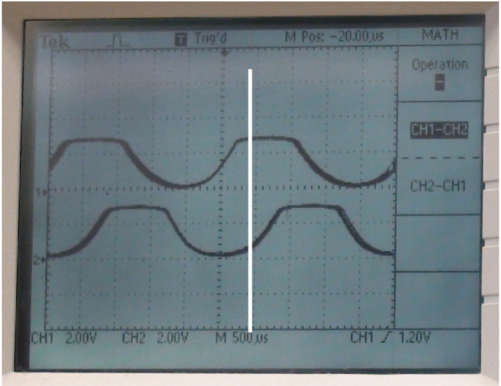
- *Speed* of rotation from frequency of either channel
- *Angle* of rotation from combination

Here's an example from an actual motor.

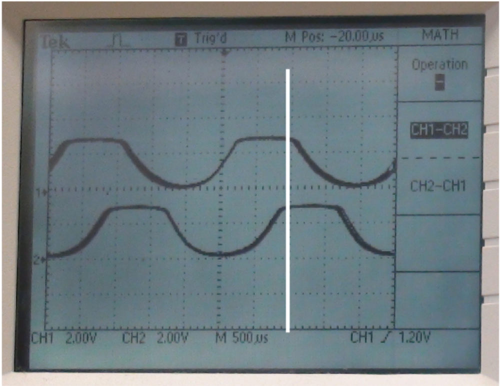

Tek
Trig'd
M Pos: -20.00us
MATH
Operation
CH1-CH2
CH2-CH1
CH1 2.00V
CH2 2.00V
M 500us
CH1 1.20V


Tek
Trig'd
M Pos: -20.00us
MATH
Operation
CH1-CH2
CH2-CH1
CH1 2.00V
CH2 2.00V
M 500us
CH1 1.20V

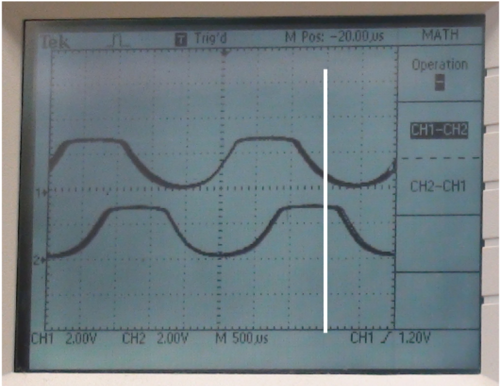
00


Tek
Trig'd
M Pos: -20.00us
MATH
Operation
CH1-CH2
CH2-CH1
CH1 2.00V
CH2 2.00V
M 500us
CH1 1.20V

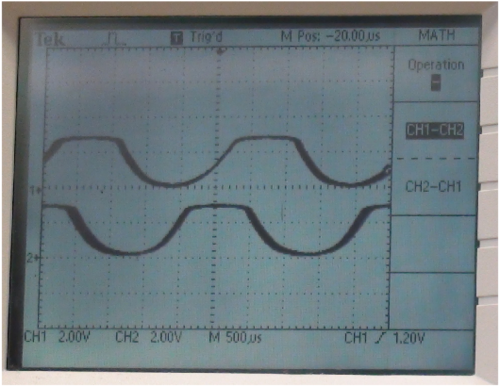
10

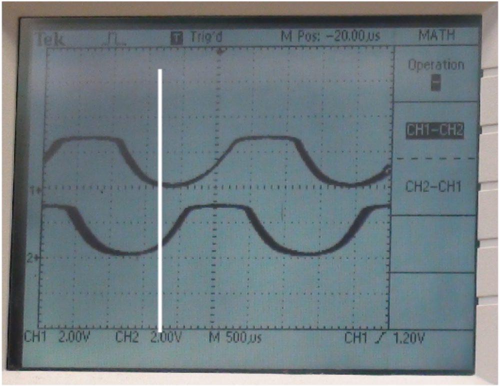

Tek
Trig'd
M Pos: -20.00µs
MATH
Operation
CH1-CH2
CH2-CH1
CH1 2.00V
CH2 2.00V
M 500µs
CH1 ∕ 1.20V

11

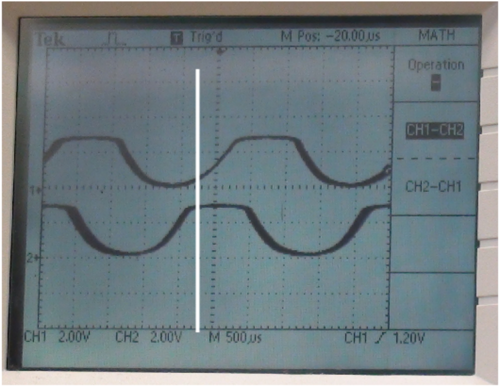

Tek
Trig'd
M Pos: -20.00us
MATH
Operation
CH1-CH2
CH2-CH1
CH1 2.00V
CH2 2.00V
M 500us
CH1 1.20V

01

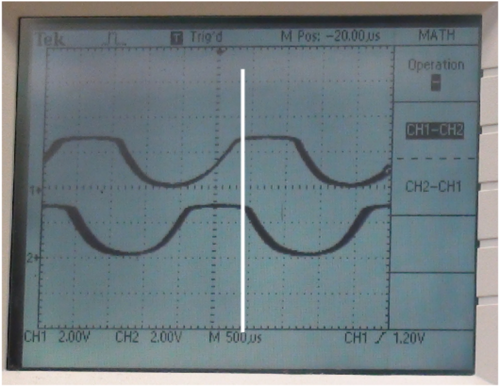

Tek
Trig'd
M Pos: -20.00 us
MATH
Operation
CH1-CH2
CH2-CH1
CH1 2.00V
CH2 2.00V
M 500 us
CH1 1.20V


Tek
Trig'd
M Pos: -20.00µs
MATH
Operation
-
CH1-CH2
CH2-CH1
CH1 2.00V
CH2 2.00V
M 500µs
CH1 ∕ 1.20V

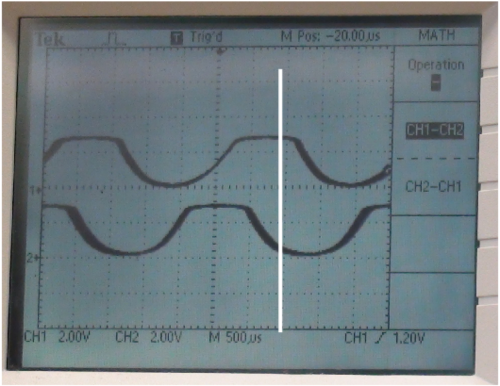
00


Tek
Trig'd
M Pos: -20.00µs
MATH
Operation
CH1-CH2
CH2-CH1
CH1 2.00V
CH2 2.00V
M 500µs
CH1 ⁄ 1.20V

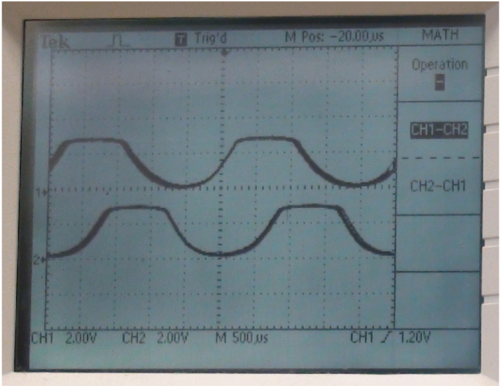
01


Tek
Trig'd
M Pos: -20.00µs
MATH
Operation
-
CH1-CH2
CH2-CH1
CH1 2.00V
CH2 2.00V
M 500µs
CH1 ∕ 1.20V

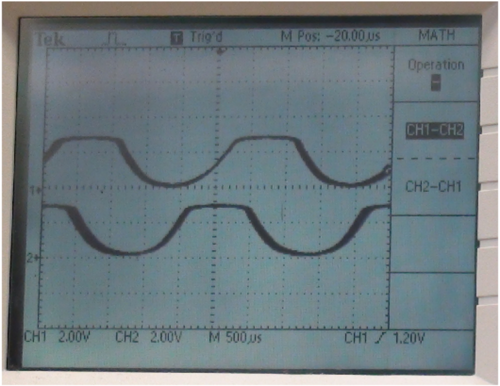
11


Tek
Trig'd
M Pos: -20.00µs
MATH
Operation
-
CH1-CH2
CH2-CH1
CH1 2.00V
CH2 2.00V
M 500µs
CH1 ⁄ 1.20V

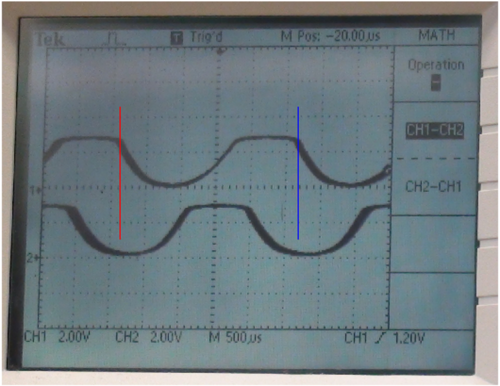
10



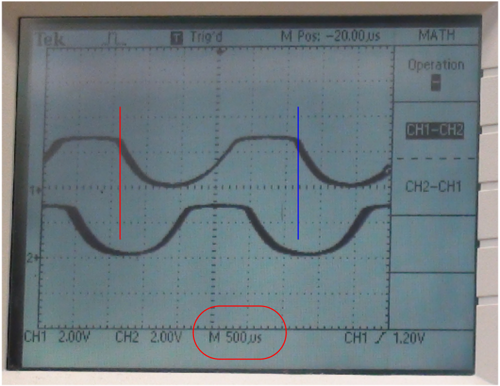
One direction



Other direction

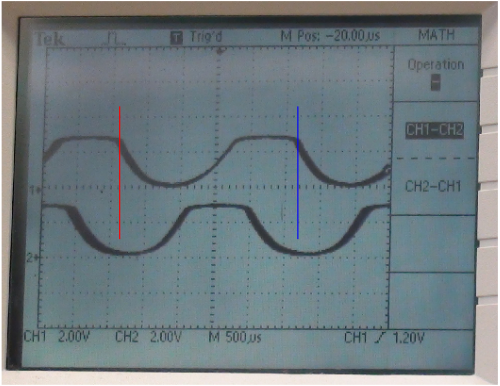


Period is 5 divisions



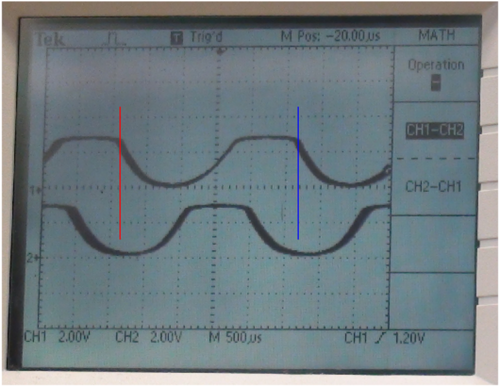
Period is 5 divisions

Time scale is 500μS per division



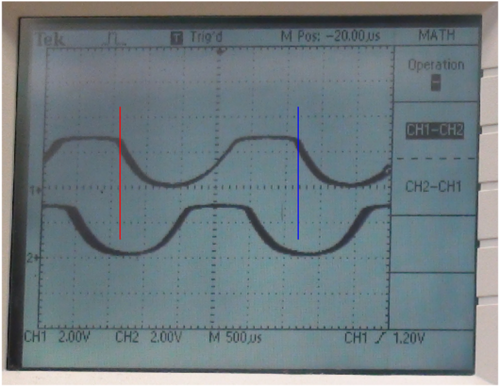
Period is 5 divisions $\rightarrow 2500\mu$S $= 2.5$mS

Time scale is 500μS per division



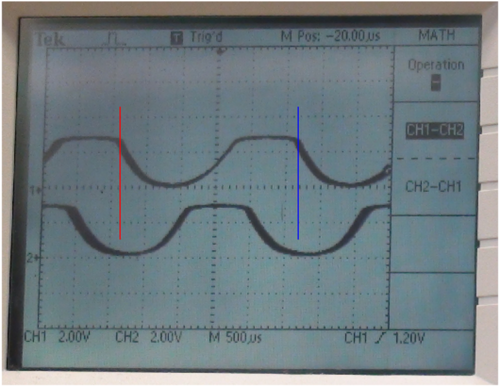
Period is 5 divisions $\rightarrow 2500\mu S = 2.5mS$

30 slots per revolution $\times$ 2.5mS



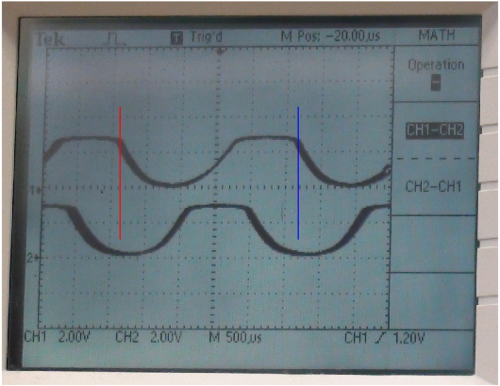
Period is 2.5mS

30 slots per revolution $\times$ 2.5mS $\rightarrow$ 75ms per revolution



Period is 2.5mS $\times$ 30 $\rightarrow$ 75ms per revolution

75/1000 seconds per revolution $\rightarrow$ 13.33 rev./second



Period is 2.5mS $\times$ 30 $\rightarrow$ 75ms per revolution

75/1000 seconds per revolution $\rightarrow$ 13.33 rev./second

$\times$ 60 $\rightarrow$ 800 RPM

Ultrasonic sensors

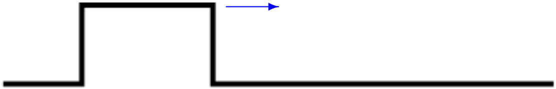
Ultrasonic sensors

- *Transmitter* sends out pulse

Ultrasonic sensors

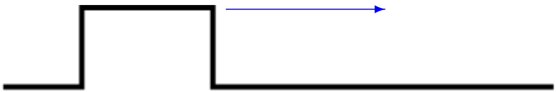
- *Transmitter* sends out pulse
- *Receiver* registers echo

- **transmit**

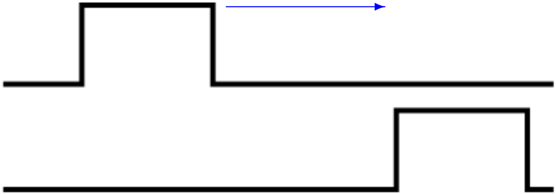


- **transmit**

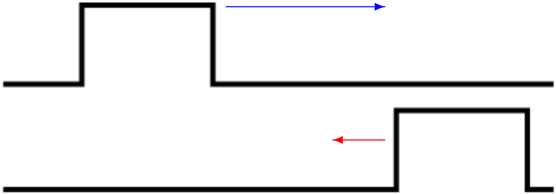
- **transmit**



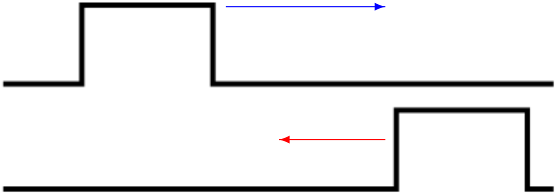
- **transmit**



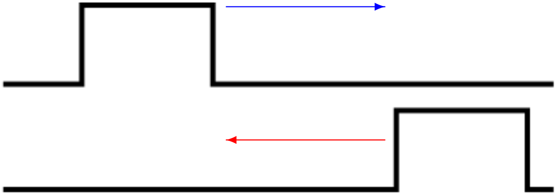
- transmit
- **receive**



- transmit
- **receive**



- transmit
- **receive**



- transmit
- **receive**

- *Time* from beginning of transmit to beginning of receive allows distance to be calculated

- *Time* from beginning of transmit to beginning of receive allows distance to be calculated
- since $2d = vt$

  where v is the speed of sound

- *Time* from beginning of transmit to beginning of receive allows distance to be calculated
- since $2d = vt$

  where v is the speed of sound

Why is it $2d$?

V to F and F to V converters

V to F and F to V converters

- *V to F converters* take in an analog *voltage* and produce a string of digital pulses where the *frequency* is proportional to the input analog voltage

V to F and F to V converters

- *V to F converters* take in an analog *voltage* and produce a string of digital pulses where the *frequency* is proportional to the input analog voltage
- *F to V converters* take in a string of digital pulses and produce an analog *voltage* where the analog voltage is proportional to the input *frequency*

Python PWM Control

Python PWM Control

- **p = GPIO.PWM(*channel*, *frequency*)**
  open channel at given frequency

Python PWM Control

- **p = GPIO.PWM(*channel*, *frequency*)**

  open channel at given frequency
- **p.start(*dc*)**

  start at given duty cycle *(percent)*

Python PWM Control

- **p = GPIO.PWM(*channel*, *frequency*)**

  open channel at given frequency
- **p.start(*dc*)**

  start at given duty cycle *(percent)*
- **p.ChangeFrequency(*freq*)**

  change frequency

Python PWM Control

- **p = GPIO.PWM(*channel*, *frequency*)**

  open channel at given frequency
- **p.start(*dc*)**

  start at given duty cycle *(percent)*
- **p.ChangeFrequency(*freq*)**

  change frequency
- **p.ChangeDutyCycle(*dc*)**

  change duty cycle *(percent)*

Python PWM Control

- **p = GPIO.PWM(*channel*, *frequency*)**

  open channel at given frequency
- **p.start(*dc*)**

  start at given duty cycle *(percent)*
- **p.ChangeFrequency(*freq*)**

  change frequency
- **p.ChangeDutyCycle(*dc*)**

  change duty cycle *(percent)*
- **p.stop()**

  stop PWM

Python PWM sample code

Python PWM sample code

```
import time
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BOARD)
GPIO.setup(12, GPIO.OUT)
p = GPIO.PWM(12, 50)  # chan=12 freq=50Hz
p.start(0)
try:
    while 1:
        for dc in range(0, 101, 5):
            p.ChangeDutyCycle(dc)
            time.sleep(0.1)
except KeyboardInterrupt:
    pass
p.stop()
GPIO.cleanup()
```

Hardware and Software PWM

Hardware and Software PWM

- The Raspberry Pi has 2 built-in PWM channels for **hardware** PWM.

Hardware and Software PWM

- The Raspberry Pi has 2 built-in PWM channels for **hardware** PWM.
- **Software** PWM is available on all GPIO pins.

Hardware and Software PWM

- The Raspberry Pi has 2 built-in PWM channels for **hardware** PWM.
- **Software** PWM is available on all GPIO pins.

  (Hardware PWM *isn't* affected by the Linux scheduler.)

Hardware and Software PWM

- The Raspberry Pi has 2 built-in PWM channels for **hardware** PWM.
- **Software** PWM is available on all GPIO pins.

  (Hardware PWM *isn't* affected by the Linux scheduler.)

  (Software PWM *is* affected by the Linux scheduler.)

Hardware and Software PWM

- The Raspberry Pi has 2 built-in PWM channels for **hardware** PWM.
- **Software** PWM is available on all GPIO pins.

  (Hardware PWM *isn't* affected by the Linux scheduler.)

  (Software PWM *is* affected by the Linux scheduler.)

For applications needing more consistent PWM, use the *hardware* PWM pins.

Raspberry Pi PWM pins

Raspberry Pi PWM pins

- The Raspberry Pi has 2 built-in hardware PWM *channels*.

Raspberry Pi PWM pins

- The Raspberry Pi has 2 built-in hardware PWM *channels*.
- Each *channel* has 2 associated *pins*.

Raspberry Pi PWM pins

- The Raspberry Pi has 2 built-in hardware PWM *channels*.
- Each *channel* has 2 associated *pins*.

  (So only one *pin* from a channel can be used at a time.)

Raspberry Pi PWM pins

- The Raspberry Pi has 2 built-in hardware PWM *channels*.
- Each *channel* has 2 associated *pins*.

  (So only one *pin* from a channel can be used at a time.)

  (Or else they will both have the same signal.)

Raspberry Pi PWM pins

- The Raspberry Pi has 2 built-in hardware PWM *channels*.
- Each *channel* has 2 associated *pins*.

  (So only one *pin* from a channel can be used at a time.)

  (Or else they will both have the same signal.)
- **PWM0** uses (BOARD) 12 and 32.

Raspberry Pi PWM pins

- The Raspberry Pi has 2 built-in hardware PWM *channels*.
- Each *channel* has 2 associated *pins*.

  (So only one *pin* from a channel can be used at a time.)

  (Or else they will both have the same signal.)
- **PWM0** uses (BOARD) 12 and 32.

  i.e. (BCM) GPIO18 and (BCM) GPIO12.

Raspberry Pi PWM pins

- The Raspberry Pi has 2 built-in hardware PWM *channels*.
- Each *channel* has 2 associated *pins*.

  (So only one *pin* from a channel can be used at a time.)

  (Or else they will both have the same signal.)
- **PWM0** uses (BOARD) 12 and 32.

  i.e. (BCM) GPIO18 and (BCM) GPIO12.
- **PWM1** uses (BOARD) 33 and 35.

Raspberry Pi PWM pins

- The Raspberry Pi has 2 built-in hardware PWM *channels*.
- Each *channel* has 2 associated *pins*.

  (So only one *pin* from a channel can be used at a time.)

  (Or else they will both have the same signal.)
- **PWM0** uses (BOARD) 12 and 32.

  i.e. (BCM) GPIO18 and (BCM) GPIO12.
- **PWM1** uses (BOARD) 33 and 35.

  i.e. (BCM) GPIO13 and (BCM) GPIO19.

Raspberry Pi PWM pins

- The Raspberry Pi has 2 built-in hardware PWM *channels*.
- Each *channel* has 2 associated *pins*.

  (So only one *pin* from a channel can be used at a time.)

  (Or else they will both have the same signal.)
- **PWM0** uses (BOARD) 12 and 32.

  i.e. (BCM) GPIO18 and (BCM) GPIO12.
- **PWM1** uses (BOARD) 33 and 35.

  i.e. (BCM) GPIO13 and (BCM) GPIO19.

Note: These pins are shared with the audio subsystem.