# Lecture 21

## Feedback and Sensors

# Agenda

- Basic PID Feedback Controls
- Ranging Sensors
- Tips for Autonomy

# Using Purchased Devices

- Digikey is having COVID warehouse issues + 5days to order
- For final project, you may use purchased components.
  - Motor Drivers
  - Motors
  - Other microcontrollers
  - Servos
- Before purchasing, be sure about compatibility
  - Does Vcc or Vdd require 3.3V or 5V?  Or more than 5.5V (many motor drivers will not work at 5V)
  - What is the interface? ADC? I2C? SPI? Timer
  - What is the Vih  (logic voltage input high)?  (ESP32 will output 2.6V worst case for Voh)

# Final Project Tentative Schedule

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| April | 4 | 5 **Today** | 6 | 7 | 8 | 9 **Design Review 1** | 10 |
| | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| | 18 | 19 | 20 | 21 | 22 | 23 **Design Review 2** | 24 |
| | 25 | 26 | 27 | 28 Last Class | 29 **Report due** | 30 Reading days | 1 Reading days |
| May | 2 Reaeding days | 3 Reading days | 4 **Grading Evaluation** | 5 **Grading Evaluation** | 6 **Grading Evaluation** | 7 **Grading Evaluation** | 8 **Day 1 Group Stage** |
| | 9 **Day 2 Group Stage** | 10 Final Brackets | 11 Last day Report accepted | 12 | 14 | 15 | 15 |

# 01

# PID Control

# Q1: What are some examples of feedback control you may see around you?

- Thermostat
- Motion control (position)
- ABS
- Cruise control
- Gene regulation

# Some terms

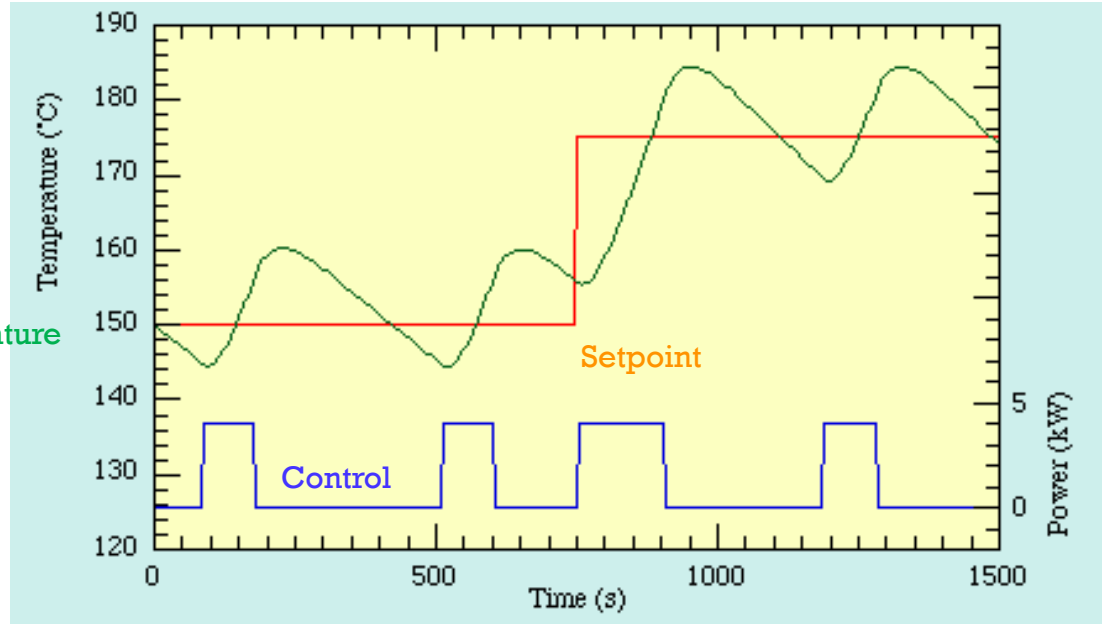
- Sensor
- Actuator
- Controller
- Plant
- Disturbance
- Regulator (set point)
- Step Response

# Balancing a broom

- What is being controlled?
  - What is the **process variable** (what is being sensed)
  - What is the **setpoint**? (goal?)
  - What is the **control variable** (actuation?)
- What is stability?
- What is disturbance?

# On-off control

- Oven




Hysteresis is often added to reduce switching

https://courses.cs.washington.edu/courses/cse466/01au/Lecture/PIDoverview.html

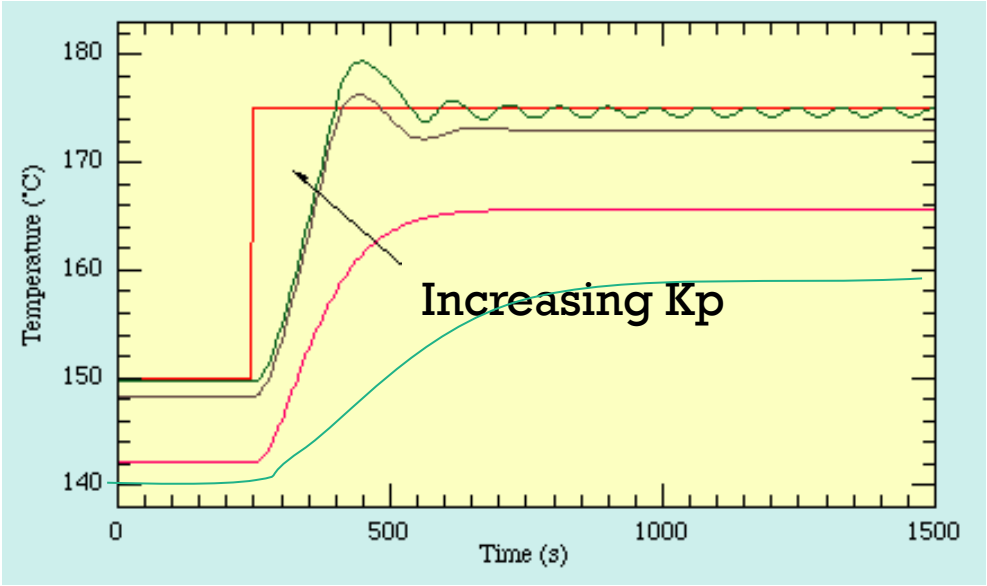
# Sample code

```
int desired;

int controlFunction(int s, int d)      // function that will set actuator values
{  }

void setup()                           //  setup motors and sensors
{  }

void loop() {
  int direction, duty, u, sensor;

  desired = userInput();                 // find desired state
  sensor = analogRead(ANALOGPIN);        // find current state
  u = controlFunction(sensor, desired); // determine control from desired, sensor
  setMotor(abs(u), u/abs(u));            // set the actuator direction and duty
                                         //      direction = u/abs(u);
                                         //      duty cycle = abs(u)

}
```

# Proportional Control (P)

- Simplest (and most common) form of control

Q2: List 3 properties about how well (or badly) the different output values track the desired input (redline)




u = Kp * (Error)

https://courses.cs.washington.edu/courses/cse466/01au/Lecture/PIDoverview.html

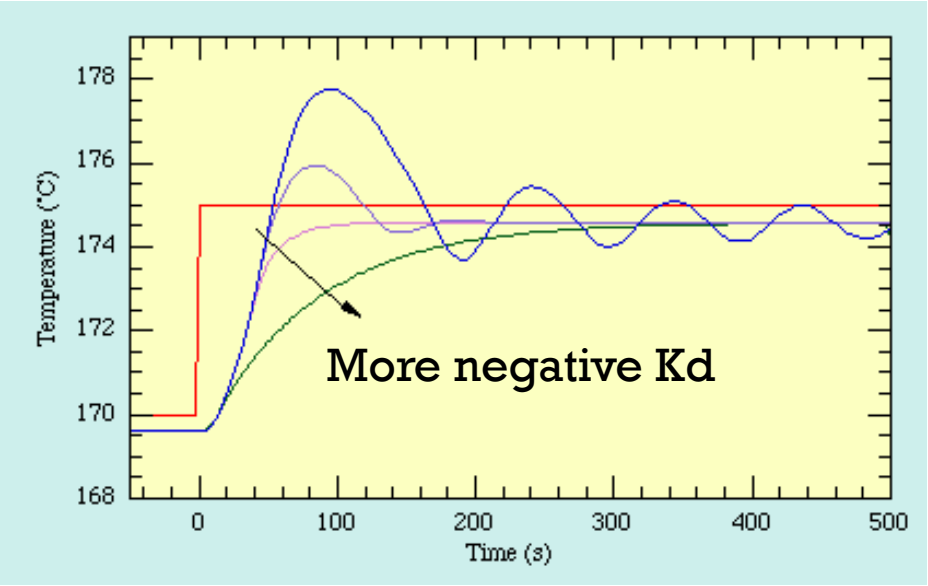
# Sample *controlFunction* for position control

//--modify loop() to call pcontrol instead of *controlFunction*

```
#define KP (2)   /* Proportional gain set here */

int pcontrol(int sensor) {
  int u;

  u = KP * (desired-sensor);

  if (u > 1000) u = 1000;      /* motor limits */
  if (u < -1000) u = -1000;

  return  u;
}
```

# Proportional Derivative Control (PD)

- Negative Derivative adds "damping"
- Gain
  - Too high?
  - Too low?




$$U = Kp * (Error) + Kd * (velocity)$$

https://courses.cs.washington.edu/courses/cse466/01au/Lecture/PIDoverview.html

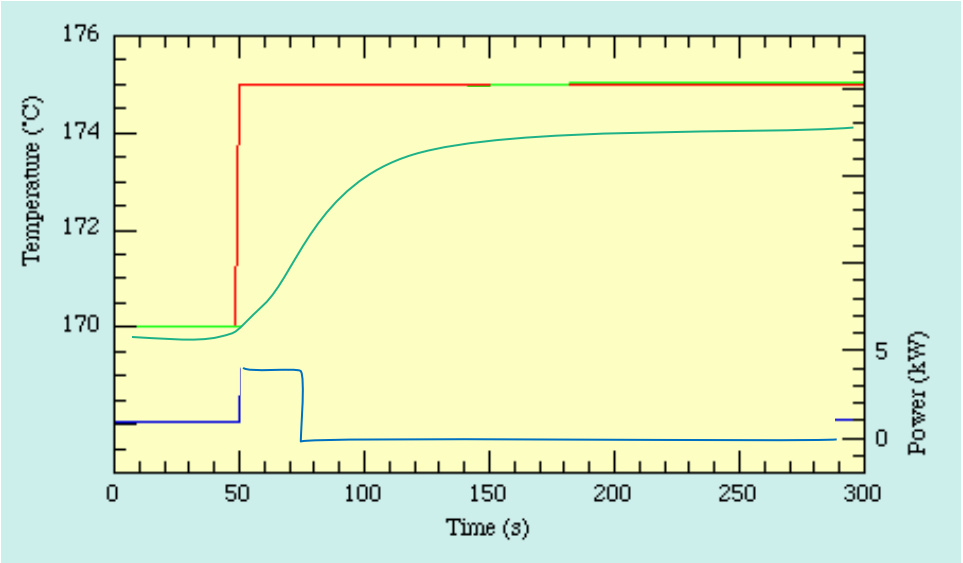
# Sample PD code

//--modify loop() to call pdcontrol instead of *controlFunction*

```
#define KD (-1)

int pdcontrol(int sensor) {
  int u;
  static int oldsensor;
  static int lasttime;

  int velocity = 1000*(sensor - oldsensor)/(millis()-lasttime);
  oldsensor = sensor;
  lasttime = millis();

  u = KP * (desired-sensor) + KD * velocity;
  if (u > 1000) u = 1000;
  if (u < -1000) u = -1000;

  return u;
}
```

# Proportional Integrative Control (PI)

- Integrating control removes steady state error
- Destabilizing




U = Kp * (Error) + Ki * (integrated error)

https://courses.cs.washington.edu/courses/cse466/01au/Lecture/PIDoverview.html

# Sample PI code

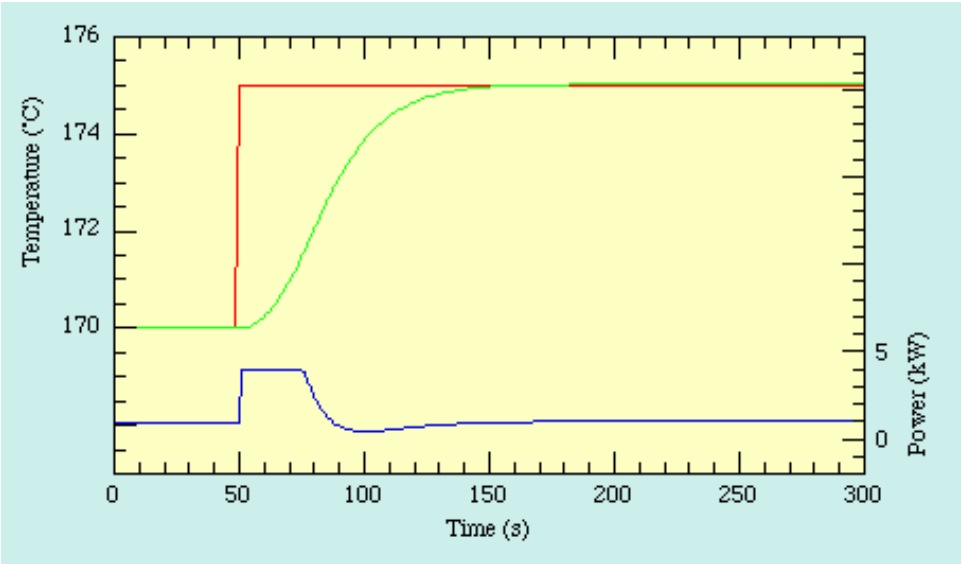
//--modify loop() to call picontrol instead of *controlFunction*

```
#define KI (1/100)

int picontrol(int sensor) {
  int u;
  static int summederror;

  summederror += (desired-sensor);

  u = KP * (desired-sensor) + KI * summederror;

  // optional helps against integrator windup (slightly more stable)
  if (desired-sensor == 0) summederror = 0;

  if (u > 1000) u = 1000;
  if (u < -1000) u = -1000;


  return  u;
}
```

# Proportional Intregrative Derivative (PID)

Add them all together.

- Proportional
- Integral
- Derivative




**U = Kp * (Error) + Kd * (velocity) + Ki * (integrated error)**

https://courses.cs.washington.edu/courses/cse466/01au/Lecture/PIDoverview.html

# Sample PID code

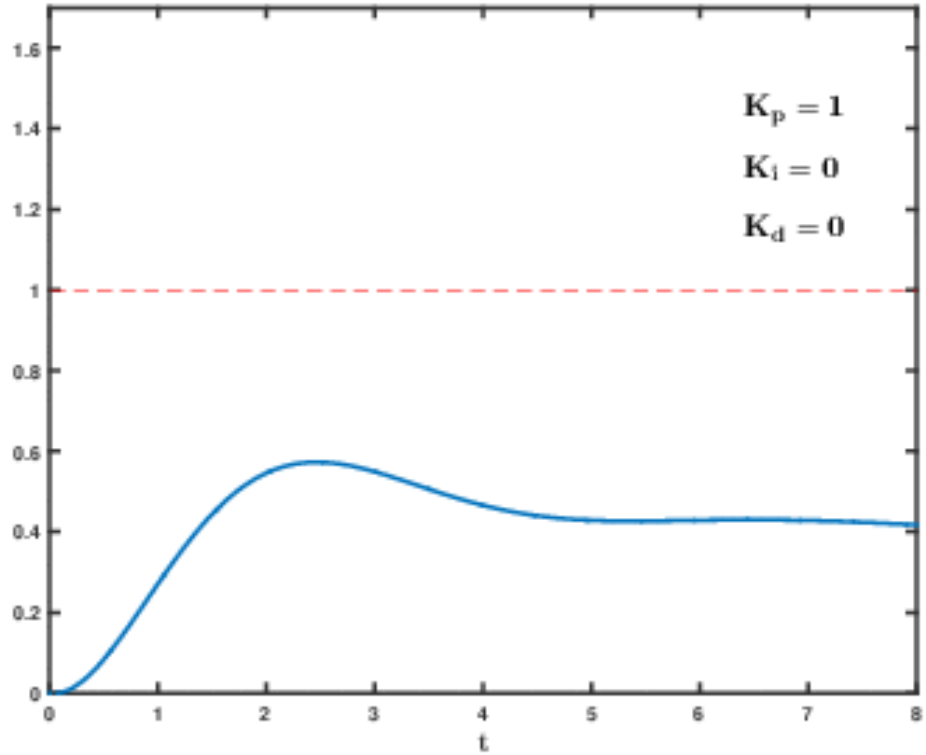
```
//--modify  loop() to call pidcontrol
//  instead of controlFunction
```

```
int pidcontrol(int sensor) {
  int u;
  static int oldsensor;
  int velocity = sensor - oldsensor;
  static int summederror;

  summederror += (desired-sensor);

  u = KP * (desired-sensor) + KD * velocity + KI * summederror;

  if (desired-sensor == 0) summederror = 0;

  if (out > 1000) u = 1000;
  if (out < -1000) u = -1000;

  oldsensor = sensor;

  return  u;
}
```

# Tuning PID controllers

- **Find KP first.** Set KD = 0 , KI = 0, Increase KP until response is fast enough (typically some overshoot).
- Or, increase until constant oscillation occurs, then reduce the gain by 30%.
- **Find KI 2nd**. Slowly increase the integral level until steady state has acceptable error.
- **Find KD last**. Increase KD to reduce overshoot (critically damped if desired, or until speed is set. )

# PID Tuning




https://en.wikipedia.org/wiki/PID_controller

# Other ways of tuning

- Create a simulation with equations of motion measuring important parameters
- Use Ziegler–Nichols for 2nd order system
  - find Kp gain where starts to oscillate,
    - Ku = Kp, then Tu = oscillation period

| Control Type | Kp | Ki | Kd |
|---|---|---|---|
| P | 0.50 Ku | — | — |
| PI | 0.45 Ku | 0.54 Ku/Tu | — |
| PID | 0.60 Ku | 1.2 Ku /Tu | 3 Ku Tu /40 |

# References

- **PID Controller Overview**
  - https://en.wikipedia.org/wiki/PID_controller

- **PID Without a PhD by Tim Wescott**
  - Canvas->files->Reading->pidWithoutAPhd.pdf

# 02

# Ranging Sensors

# Ranging Technology
## – measuring distance to objects

- Three mechanisms for measuring distance (as far as I know)
  - Time of flight
  - Intensity
  - Triangulation
  - Contact sensors (e.g. whisker)
- Forms of energy used
  - Light (e.g., IR proximity sensors)
  - Sound/vibration (ultrasonic transducers)
  - Radio frequency EM (e.g., radar ) – not useful for this class
  - Heat (PIR, or pyro-electric) – human motion detection – not distance.

# Things to think about for quiz at end:

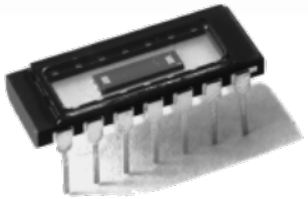
1. You have a robot that is moving in a straight line and you want to accurately estimate when it will hit a wall.
2. You have $5 budget left and you want to sense if something is 2 cm in front of the robot so you can grab it.
3. You have $5 budget left and you want to sense if something is 50cm away to help navigation.
4. You have lots of budget left and want the most reliable way of detecting if a robot is approaching up to 50cm away

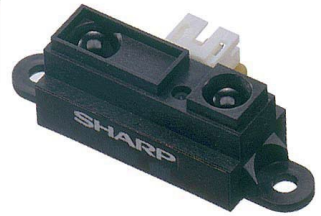
# Ranging Sensors

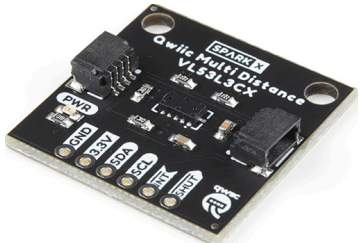
- Ranging sensors determine the distance between objects.
- Typically energy is emitted and reflected energy is measured (intensity, location, time).
- Sometimes energy from environment is used (e.g., cameras)

# Range sensors

- Proximity Ranging
  - ~$1-5 Cheap, short range, inaccurate, single point
- Ultrasonic range sensor
  - ~$4-50 Cheap, unreliable, single point
- Lateral effect photo-diode
  - ~$15-50 Fast, single point
- Triangulation (structure light vision)
  - ~$30-400, typ. Laser line stripe, and video cam.
- Patterned stereo vision (Kinect)
  - ~$200, cheap cameras, need large processing
- Lidar (sick™ hokuyo™ )
  - ~$1000-3000, relatively big expensive
- Time of Flight (phase based) ranger
  - ~$10-15, small, slow, noisy but relatively accurate



# Choosing range sensors

- Cost (can be pennies to thousands of dollars)
- Min/max Range is always critical
- Noise (precision and accuracy)
  - Precise -> consistency in values
  - Accuracy -> correct distance measure
- Sampling speed
  - Probably not critical for final project
  - Can increase precision (by taking many samples and averaging)
- Understand beam width
  - Will increase precision of existance or absence of objects

# IR retro-reflective

- Point ranging
- Intensity based
- Typ short range (~.01 to ~1m) dependent on source light intensity.
- Useful for detecting presence or motion
  - Car in a parking lot
  - Person sitting on a toilet
- Can get rough estimate of absolute range
- Lensing defines active sensed area
- Can be very cheap!

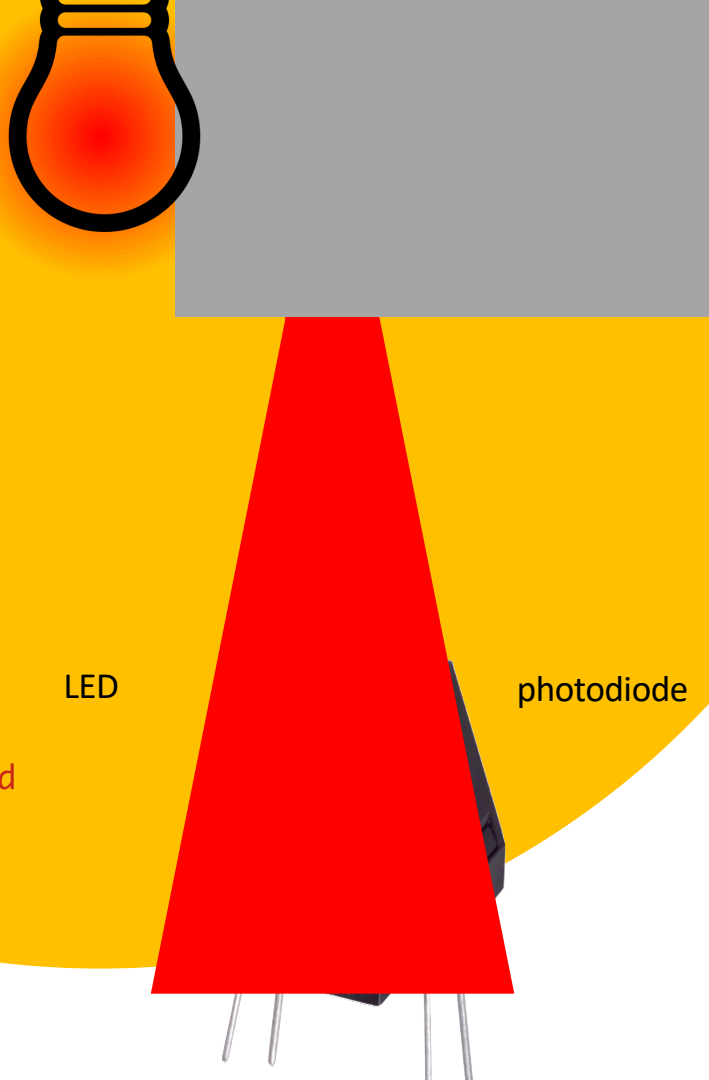
Q3 Can you build your own IR retro-reflector with items in your kit

# Retro-reflection Operation

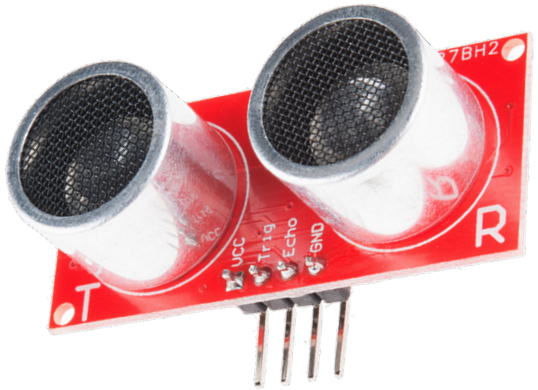
- Intensity of reflection sensed from LED light
  Roughly by inverse square of distance → $P = 1/R^2$
- Ambient light adds sensor offset
- Sensing with LED on and off can remove ambient offset.

Q4 What can we do about compensating for the shift from the ambient light?

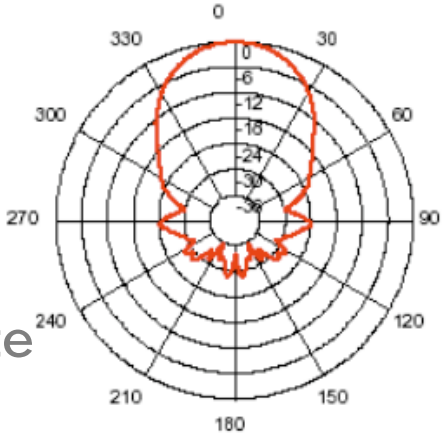
```
int  sensorOutput (){
  int ledsignal, ambient;

  digitalWrite(LEDPIN, HIGH); // turn LED on
  delay(1);                   // LTR4206 needs 10uS to respond
  ledsignal =  analogRead(SensorPin);
  digitalWrite(LEDPIN, LOW); // turn LED off
  delay(1);
  ambient =  analogRead(SensorPin);

  return (ledSignal - ambient);
}
```



# Ultrasonic Ranging

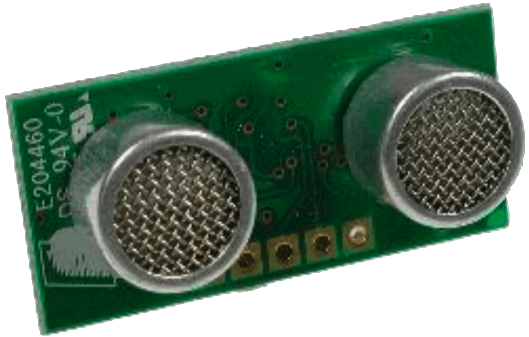



- Point ranging
- Time of flight based
  - speed of sound = 340m/s
  - E.g. 30m takes ~0.1s
- Uses timer to measure distance
- Range typ. (0.03m to 10m)
  - Cannot sense too close - typ transducers ring
  - Gain scheduling for better long range detection
- Sample rate limited by time for ping to dissipate
- Beam width varies
- Chirp typ. slew of freq, for better reflectance



# Example Ultrasonic Range Sensor

- RCWL-1601
- $3.95
- Range: 10cm – 250cm
- Speed: ~10 samples/sec
- Beam width: typ ~75°
- Sample Arduino code on Adafruit

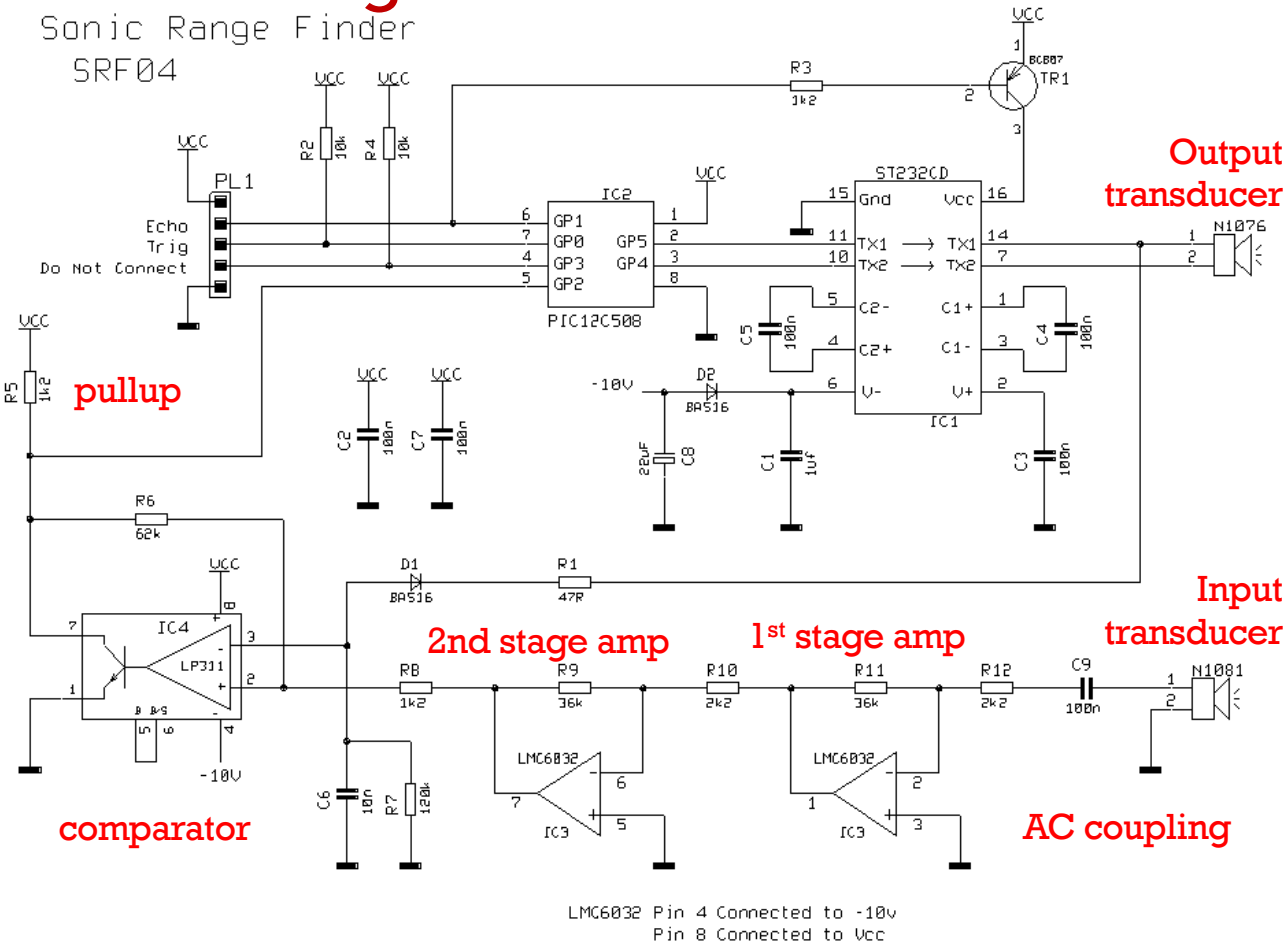


https://www.adafruit.com/product/4007

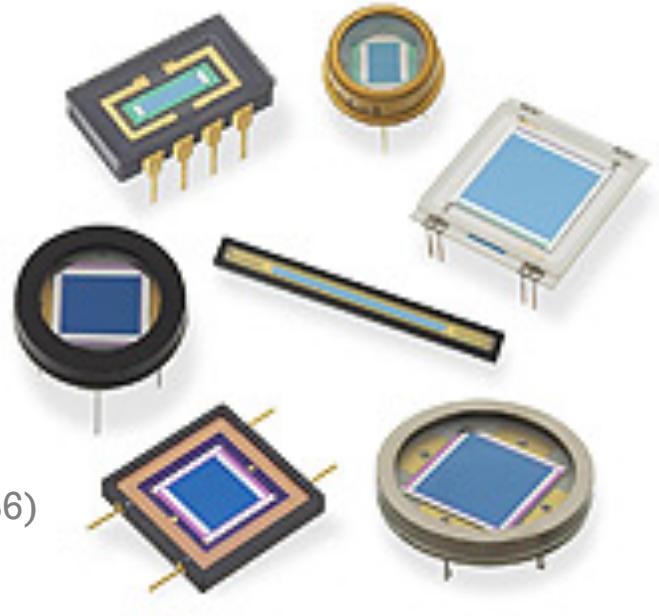
# Example Ultrasonic Range Sensor

Devantech SF04

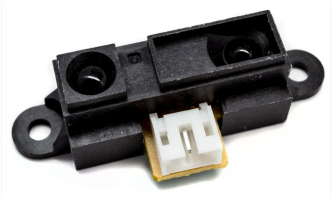
~$18



# PSD: Position Sensing Device
## [aka position sensitive detector]



- Point ranging via Triangulation
- Lateral effect photodiode
  - Returns the centroid of falling light
  - Can be precise (um sometimes nm)
  - Can be fast (typ > 100K samples)
  - Tend to be expensive compared to diodes (cheapest ~$6)
- Applications
  - Alignment motion axis of laser gyro
  - Bore sighting in autocollimator
  - Wheel alignment on cars
- Vendors
  - Hamamatsu
  - On-Trak Photonics Inc
  - Sharp – makes PSD packaged w/lenses

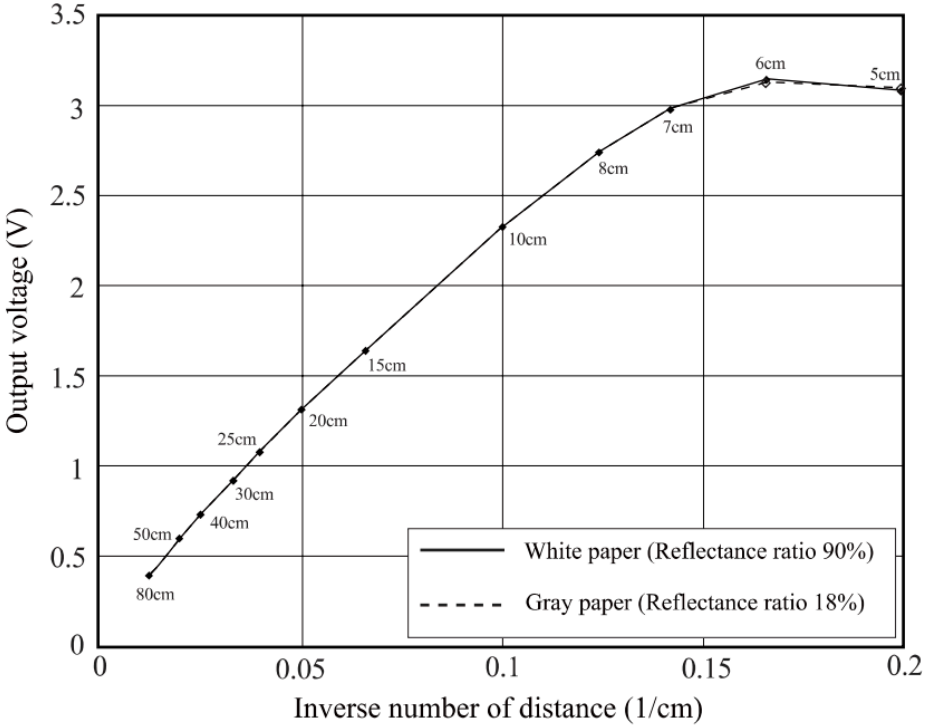


# PSD based sensor units – packaged w/lenses

https://www.adafruit.com/product/164

Sharp GP2Y0A21YK0F

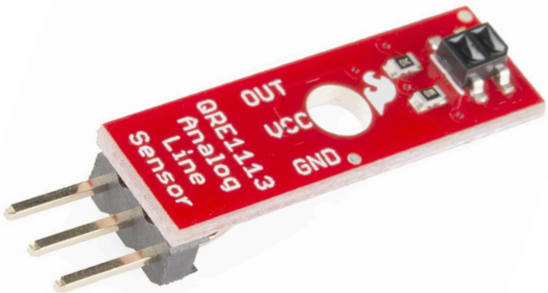
- Range: ~6cm to 80cm
- Speed: ~20 samples/sec
- FOV: 20-40 deg? (guess)
- Uses ADC
- Mostly independent of object color
- $14.95



# Black Line Detector (Redbot line follower)

- Short distance IR emitter/detector pair with integrated resistors. (Just retro-reflectors with  resistors)
- Very simple, Uses ADC
- Range: ideal 0.125” from surface to white/black edge of line.
- ~$2.95, or build your own.
- Probably will work w/3.3V @ Vcc

https://www.sparkfun.com/products/11769



# Flex sensor (whisker)

- Bending sensor changes resistance
- Simple to use with ADC in voltage divider.
- $8.95 to $12.95
- Can use like whisker for wall following
- Range: 2-4"
- Speed: as fast as ADC


https://www.adafruit.com/product/182
https://www.sparkfun.com/products/10264

# Force Sensing Resistor (FSR)

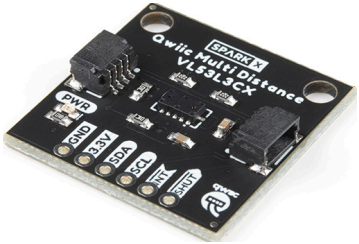
- Push on black pad, resistance changes – use ADC
- Lots of hysteresis (relatively crappy sensor)
- Not very senseitive.
- $6.95,  Some available in GMlab
- Use ADC in voltage divider.

https://www.sparkfun.com/products/9375

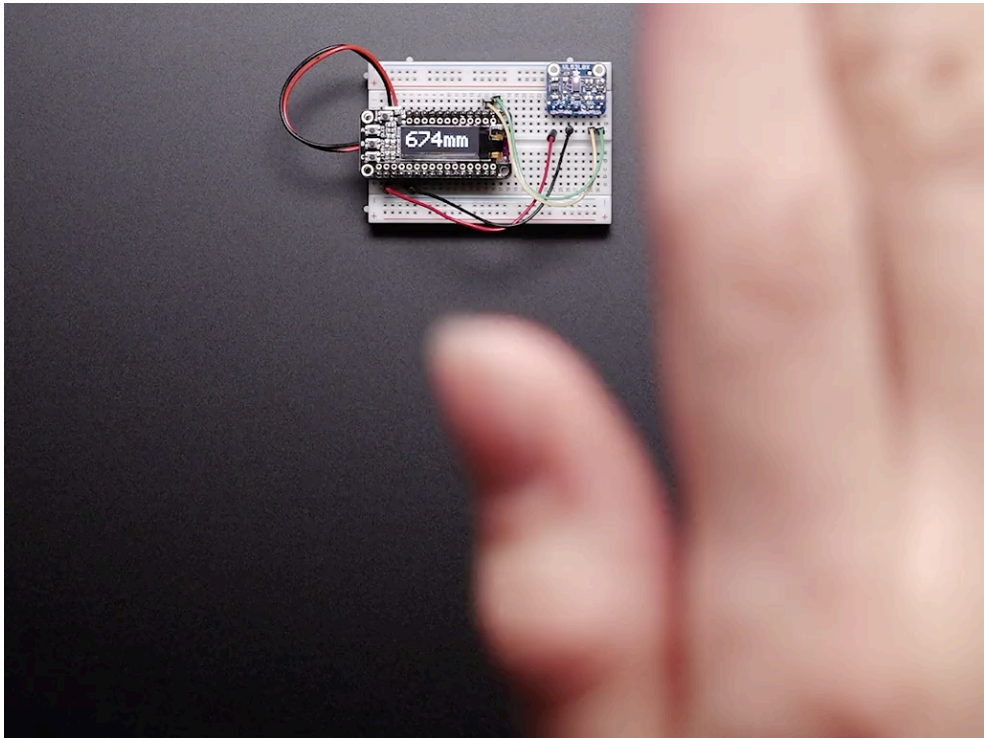
# Time of flight and Lidar sensors

- Light Detection And Ranging (time of flight)
  - Sick
  - Hokuyo
- Functions
  - 2D scanning (line of points)
  - Okay to use outdoors
- Typical specifications
  - Typ range 0.1m – 30m
  - Scan rate ~50Hz
  - Resolution 0.25° @ 3-5mm
  - Price $2k to $6k depending on model
- Historically important to robotics (Solved SLAM)
- Single point sensor (TFMini-S micro LiDAR) $39.95


# TOF Range Finder (VL53L0X)




- Range: 5cm to 120cm
- Speed: ~30 samples/sec
- Beam width: ~25°
- $14.95
- Interface with **I2C**,
- Adafruit has Arduino library.
- Very accurate and linear




https://www.adafruit.com/product/3317

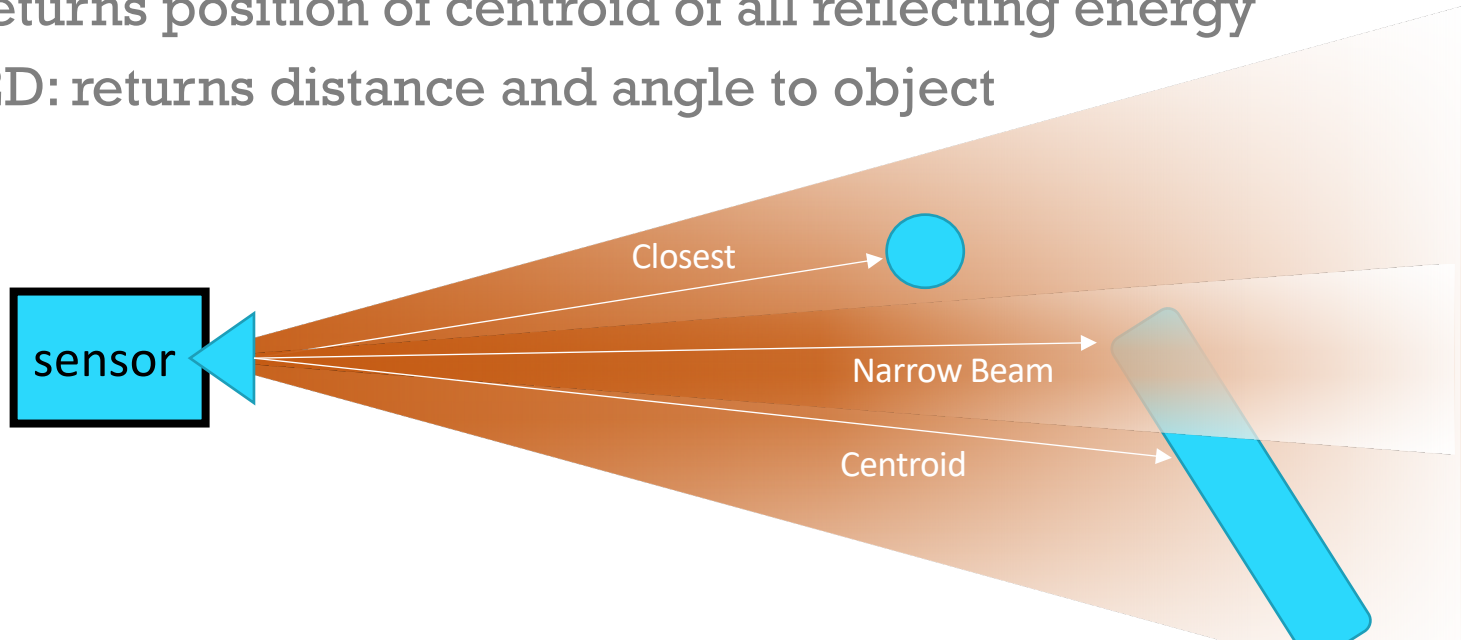
# ZX sensor (2 IR emitter/detectors)

- Z-axis 12” range
- X-axis 6” range
- Speed: 50 samples/sec
- FOV: 2D sensor (centroid+distance)
- $26.50
- **UART or I2C interface**
- 3.3V ok
- Arduino code on sparkfun
- Relative motion is accurate, abs position is not
- Subject to color of object


https://www.sparkfun.com/products/13162

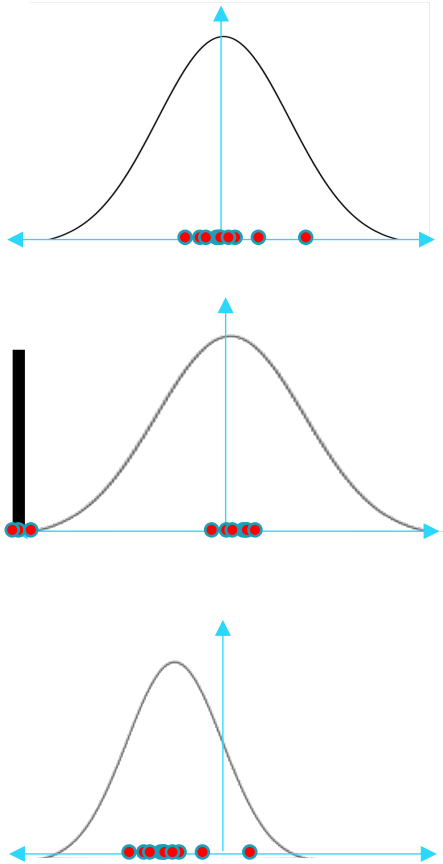
# Beam Width

- Sensing anything in FoV of transmitter – reports distance to closest point that reflects enough energy
  - Smaller FOV gives more precise location
- PSD: returns position of centroid of all reflecting energy
- ZX is 2D: returns distance and angle to object



# Noise characteristics – Precision vs Accuracy

- Variable data - gaussian distribution
  - If center of distribution is accurate, sensor can be accurate but have low precision (take average of many data points to increase accuracy)
  - TOF and PSD
- Unreliable data – bimodal
  - Occasional missed data. False negatives
  - Ultrasonic Rangers
- Distance susceptible to non-distance variables (e.g., object color)
  - Retro-reflectors, ZX sensor



# Summary Range Finders

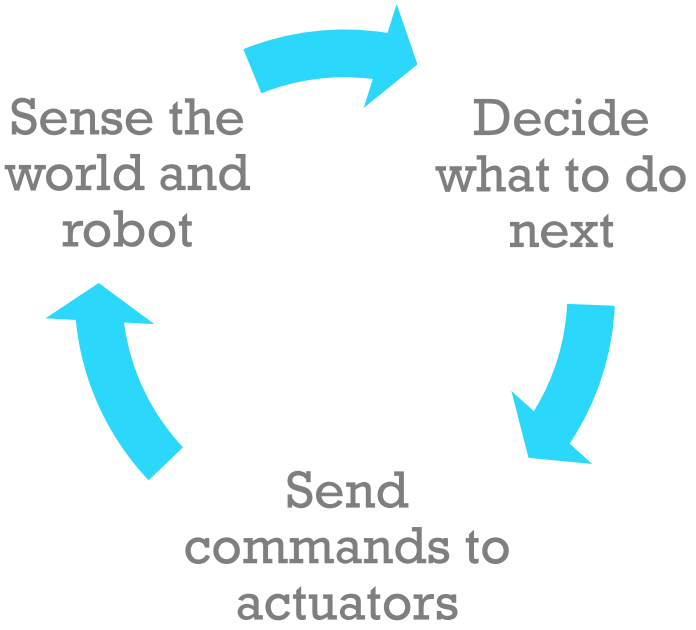
| | Cost | Range | Speed | Beam width | Precision | Accuracy |
|---|---|---|---|---|---|---|
| Retroreflective | < $2 | 0 – 20cm | >500Hz | Custom | Med | Low |
| ZX retroreflective | 26.50 | 0 – 80cm | 50Hz | 2D | Med | Low |
| Ultrasonic (ping) | 3.95 | 2 – 450cm | ~10Hz | ~75° | Low | Med |
| PSD | 14.95 | 10 – 80cm | ~20Hz | Centroid | High | Med |
| TOF | 14.95 | 5 – 120cm | ~10Hz | 25° | Med | High |

# 03

# Tips for autonomy in the final project

# Autonomy: Sensing » Reasoning » Acting

- A similar master loop will likely be in most robots, especially in an autonomous mode
- Exceptions:
  - Pieces done in parallel
  - Open-loop control
  - Modular design (separate loops)



# Example while loop

```
void loop()
{
    […] //Comms section removed for brevity

    if (isADCRead) {
        isADCRead = FALSE;
          adc_read(rawADCCounts);
    }
    localize_wii(&(robot.global));
    target_update(&target, rawADCCounts);

    find_state(&robot, &target);

    solenoid_update(&robot);
    dd_update(&robot);
}
```

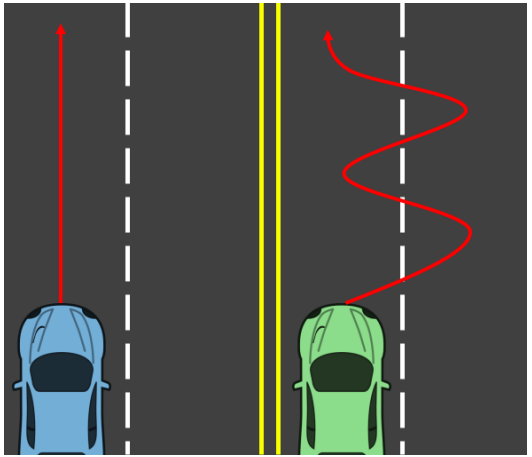
**Sensing**

**Reasoning**

**Acting**

# Going Straight Autonomously

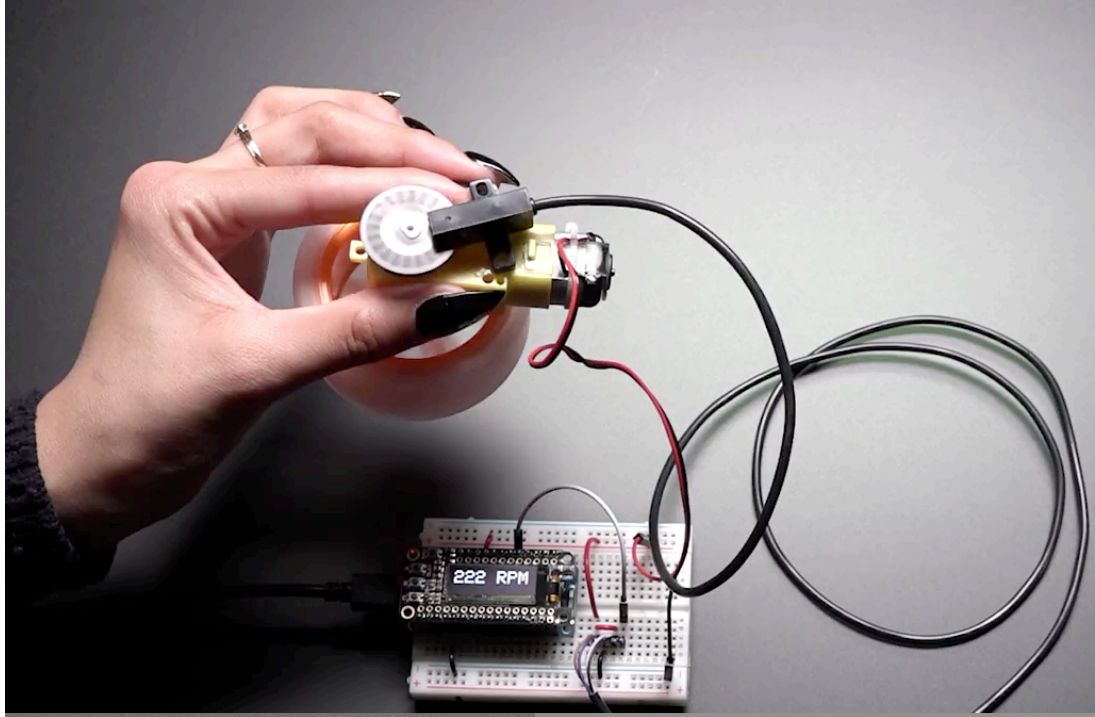
- Well-made systems (high gear ratios) may just work, not that likely
- Closing the loop with sensing
  - Motor encoders can be used with odometry (doesn't consider slippage)
  - If not properly tuned can lead to wiggly lines

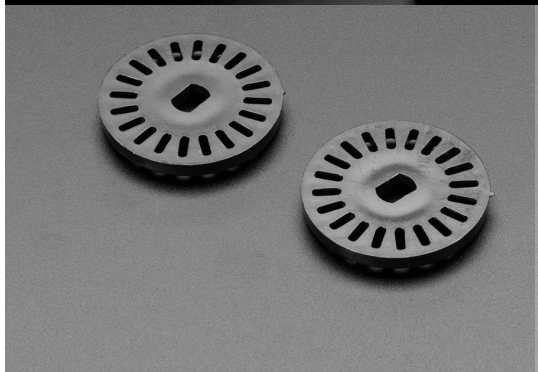


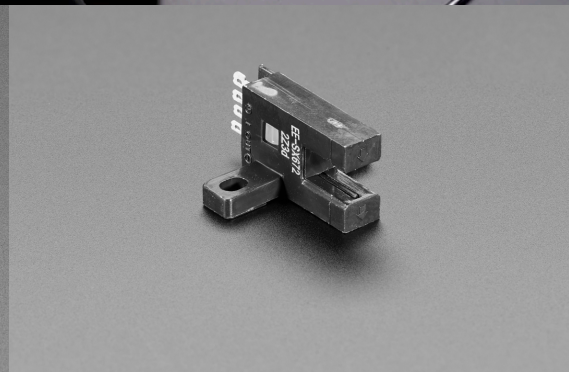
# Going Straight

- Using low cost encoder.
  - Wheel with slots
  - Photo-interrupter
    - LED – photodiode pair
- Creates pulses as wheel rotates.
- Monitor position and velocity

- $2.95 www.adafruit.com/product/3986
- $0.95 www.adafruit.com/product/3782





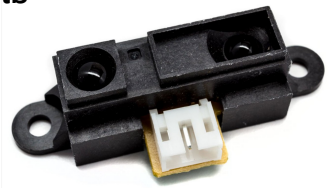
# Possible Blind Strategies

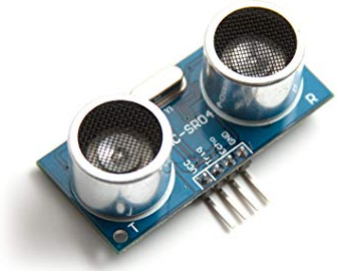
- Wall follow to sweep cans on walls
  - Get to tower, find button, press button
  - May need to detect touch/impact
- Beacon follow to grab beacon
- Lighthouse tracking to return home.
  - Need to turn when bumping into something (like the lighthouse base)

Top half wrapped in foil

Bottom half wrapped in white paper

Soda can and beacon objects





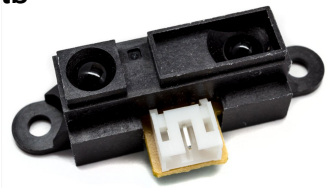
# Possible Sensing/transmission strategies

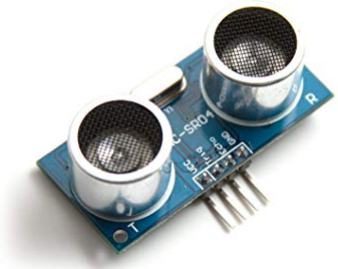
- Use rangers to get an image of field
  - Transmit blob locations through HTTP message back to website
  - (Teaching staff will help write Javascript code for this)
- Transmit position of lighthouse to help with orientation when blind
- Use touch/force sensors to indicate when to grab an object (or robot)

Top half wrapped in foil

Bottom half wrapped in white paper

Soda can and beacon objects





# Summary Quiz

Select which of the following range sensors is the most appropriate for each case:

a) Retroreflective
b) ZX retroreflective
c) Ultrasonic (ping)
d) PSD  Position Sensing Device
e) TOF (Time of Flight)

1. Accurately sensing the time for a robot moving in a straight line to hit a wall.
2. Lowest cost way to sense anything 2 cm infront of me?
3. Lowest cost way to sense anything 50cm in front of me?
4. Most reliable way to sense an object approaching 50cm away

# Sensor Purchase check list

- What voltage power is required? (Am I using Teensy or ESP32)
- What type of interface does it use?
- What is the sensor range (does it fit my use)?
- Where does it apply (does it sense my case)?
- What is the sample speed?
- What is the noise like?
- Does it require other supporting equipment?
- What is the cost?
- What is the size/weight?

# Answer in CHAT

Answer how you feel about each topic below with:

1. I don't understand this topic at all
2. I don't know now, but know what to do to get by
3. I understand some, but expect to get the rest later
4. I understand completely already

A. PID control
B. Ranging sensors
C. What behaviors to use for final project