Class Period 04 – Sensors

- Class Goals:
  - Teams
  - Raspberry Pi sign-in
  - Learn about "other" sensors
  - Sensors in class
    - Ultrasonic Range Sensors
    - Reflectance Sensors
    - Light-based Distance Sensor
    - Motion Detector
    - Quadrature Encoders
    - IMU ?
    - Others?
  - Arduino data – sensorManagerMultiRate.ino
  - ROS Node to Calibrate them
  - Plot in rqt_plot

- Announcements:
- PREP 05 and PREP 05 Quiz
  - due by next class period
- Class Period 05 – Actuators

Connecting to the Raspberry Pi by VNC

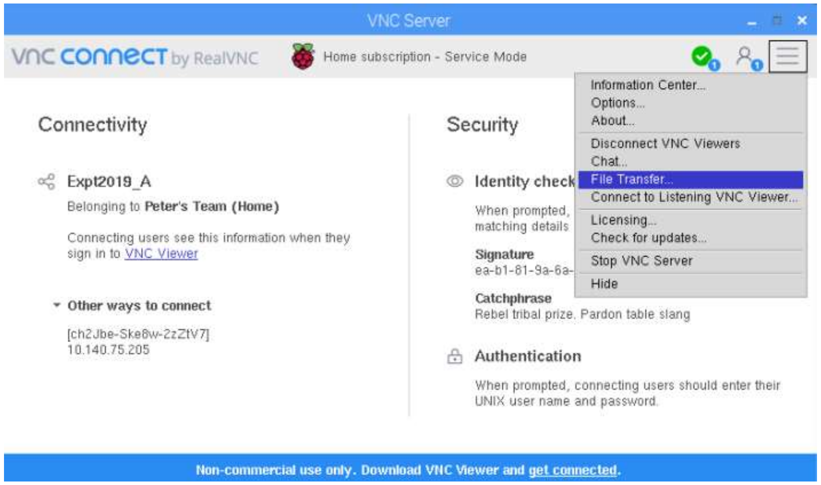
- NOTE: **IP Addresses may change**
- Therefore **sign-in by computer name**.
  - When you signed in on the RasPi, it linked to your account.
  - Therefore, sign in to RealVNC Viewer and the RasPi should appear!

- After daily sign in, get current IP address for file transfers.

- At-home use: see Canvas:
  - Resources, Feedback, Tips and Tricks → Computing Tools → Raspbery Pi → Home Wifi on RasPi

Transferring Files to/from Raspberry Pi

- Several options:
  - WinSCP (recommended)
    - Free download: https://winscp.net



  - RealVNC menu tools
    - Send using VNC icon → File Transfer
    - Fetch using top-edge menu

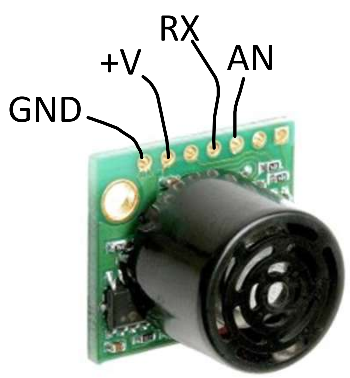
  - Command line SCP through PuTTY

Discussion – Interesting Sensors

- (Talk with teammates)

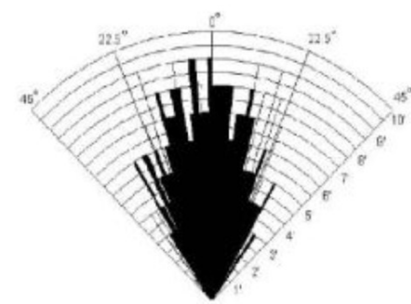
MaxBotix Ultrasonic Range Sensor

- Operation in Class:
  - Connections
    - GND and VX (+5V or 3.3V supply)
    - Signal on Analog port AN (5th pin)
    - * Optional: may need to pull “RX” (4th pin) to “high”
      - * Alternate: may be able to use Digital pins for trigger/echo operation



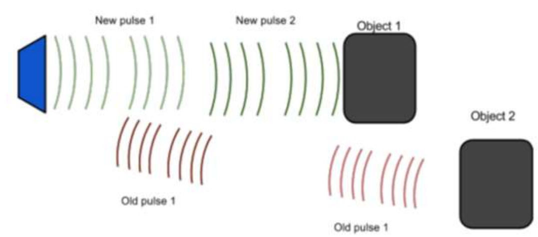
- During Class:
  - Sense different objects
  - Consider strengths/weaknesses

  - Range
  - Field of View

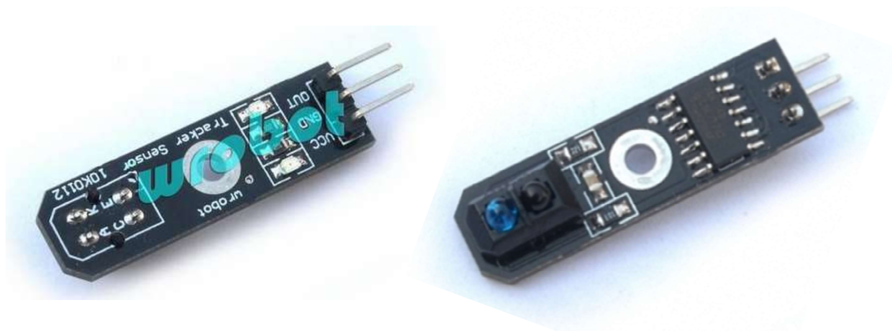


  - Echoes

  - Speed
  - Material Sensed



Light/Dark Sensor

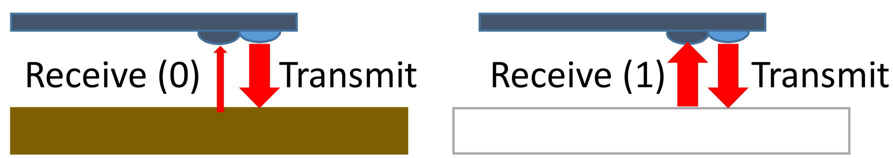


- Operation in Class:
  - Connect to Analog*
    - port A0-A5 (use A0-A2)

* It is really a Digital signal – could theoretically be switched to digital pin.

- Goals for Class:
  - Use 1-2 light/dark sensors
  - Learn response to different surfaces
  - Consider strengths/weaknesses

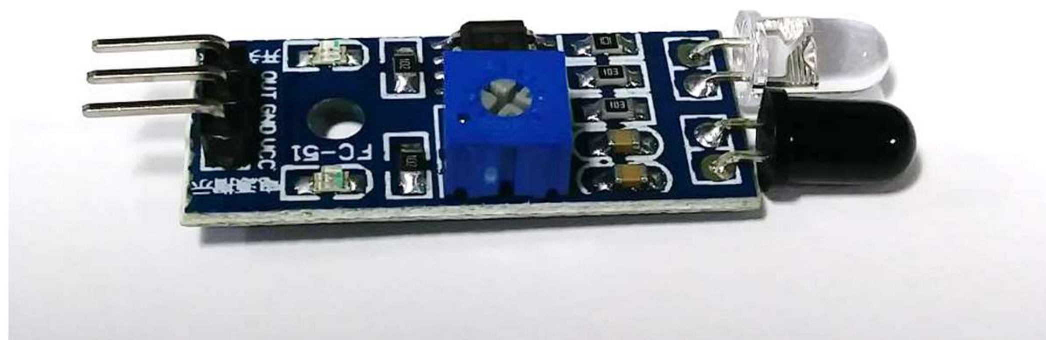
- Range
- Range of Colors/Textures



Reflectance-based Obstacle Detection Sensor

Web search for info!

- http://qqtrading.com.my/ir-infrared-obstacle-detaction-sensor-module-fc-5



Motion Sensor

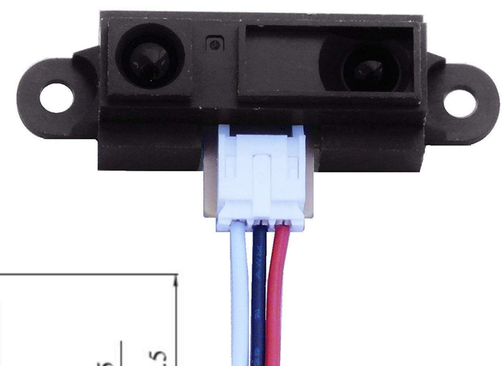
- Web search for info!

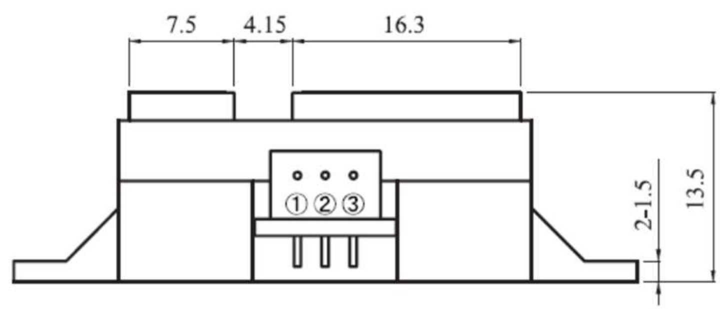
Infrared Distance Sensor

Sharp GP2Y0A21YK0F

- 10-80 cm range
- 4.5-5.5 V supply
- Analog output
  - voltage ~ distance

- https://smile.amazon.com/gp/product/B00IMOSEJA





Connector signal

	signal name
①	Vo
②	GND
③	Vcc

Connector :
Shenglan Technology Co.,Ltd
(JCTC)
12001W90-3P-HF

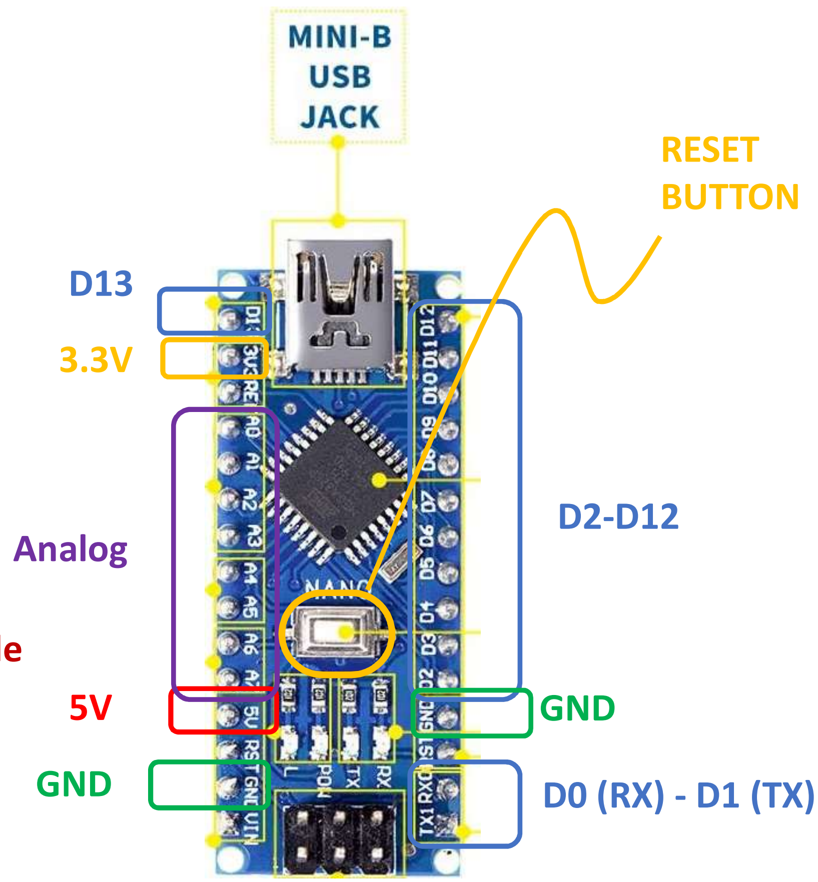
Note:
These have not worked well in the past.
Feel free to try anew.

Arduino Nano Connections:

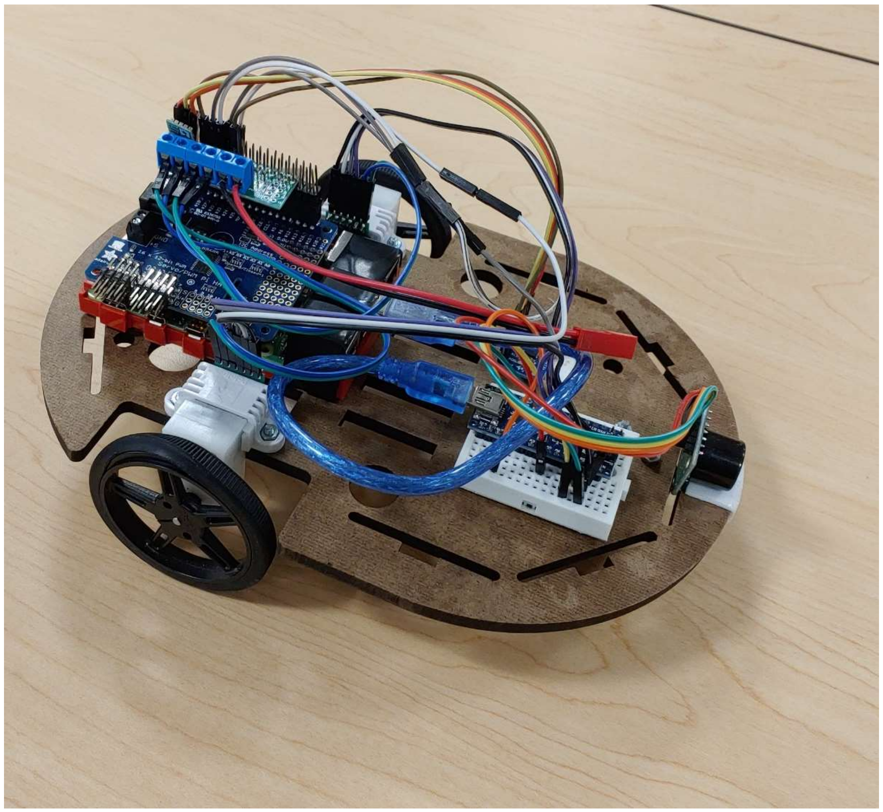
- Arduino Nano
  - ATMega 328P microcontroller
  - Analog Inputs
  - Digital Inputs
  - Timers, Interrupts, …
  - Program with Arduino IDE
  - **Connect to RasPi by USB**
    - name: **/dev/ttyUSB0**

Plug into Breadboard

***Leave extra room on this side to access 3.3V, 5V, GND**



Hardware Connections – Arduino board



- MaxBotix Ultrasound: Analog (any)
  - Gnd – Gnd
  - +V – 5V or 3.3V
  - AN – A0 (or other)
  - maybe RX – 5V or 3.3V
- “Old Ultrasound”:
  - Digital (D2+3; D4+5; D6+7)
  - Pairs: Trigger then Echo
- Light/Dark: Analog (A0-5)
- Others: Analog (A0-5)
- Encoder 0: D8 and D9
- Encoder 1: D10 and D11

Obsolete – but still available if needed

Arduino Nano setup

- **Arduino IDE on RasPi**
  - **From terminal, launch “`arduino`“**
- Choose settings appropriate for the Arduino Nano:
  - Tools/Board → `Arduino Nano`
  - Tools/Processor → `ATMega 328P`
  - Tools/Serial Port → `/dev/ttyUSB0`
- Load an Example file:
  - File/Examples/Basics → **Blink**
- Click “Upload” Button 
- Observe blinking LED!?

Arduino Sensor Program: `sensorManagerMultiRate.ino`

- “sensorManagerMultirate.ino”
- Custom Arduino program*
  - Counts quadrature pulses
  - Measures: Analog, Ultrasonic Range
  - * Streams the data to Raspberry Pi over the USB port (/dev/ttyUSB#)
  - Requires two libraries:
    - NewPing.zip
    - TimerOne.zip
- Transfer all from Canvas into a folder: `/home/pi/Arduino`

- Instructions
  - Launch Arduino
  - Import Libraries (Sketch → Import Library)
    - NewPing.zip
    - TimerOne.zip
  - Load main program in Arduino IDE
    - sensorManagerMultiRate.ino
  - “Upload”

*gratitude to Alex Dawson-Elli

Arduino Sensor Program: **`SensorManager_Multirate.ino`**

- How it Works:
  - Arduino:
    - sensorManagerMultiRate.ino program "polls" all the sensors.
    - Sends data packet e.g. "A0:1023 …" over Serial Port at ~100 Hz.
    - (unit: arbitrary: min. 0, max. 2^{10}-1 = 1023)
  - Raspberry Pi (Python):
    - Receive packet over Serial
    - Decode
    - Calibrate: Volts → Real value?
    - Store data in variables

Testing the Arduino

- Run Arduino
- Start “Serial Monitor”
  - Tools → Serial Monitor
  - Or press Ctrl+Shift+M
- Set “baud rate” to 57600

Example Data:

```
E0:-18300
E1:10
A0:1013
E0:-18328
E1:51
A1:1001
E0:-18356
E1:83
A2:991
E0:-18372
E1:105
A3:980
E0:-18386
E1:131
A4:1023
E0:-18397
E1:155
A5:1023
E0:-18408
E1:170
U1:0
...
```

- E: “Encoder”
  - (Obsolete – now connected to RasPi)
- A: “Analog”
  - Most sensors will put data here
- U: “Ultrasound”
  - (Old, Cheap Ultrasound – if used!)
- If not: troubleshoot (ask for help)

Quick Note: Sensor Streaming from Arduino

- sensormanagerMultiRate.ino
  - Streams serial data:
    - E0, E1, A0, A1, …, A7, U0, …, U2
  - “index out of range” = bad line!
  - Observe the raw data with “screen”
    - `screen /dev/ttyUSB0 57600`
    - To kill it: CTRL+A [then] k
    - Other info: CTRL+A [then] ?

    - */dev/ttyUSB0 = virtual Serial port on USB*
    - ** 57600 baud = bit rate of the data*
    - **If “screen” not available:*
      - `sudo apt-get install screen`

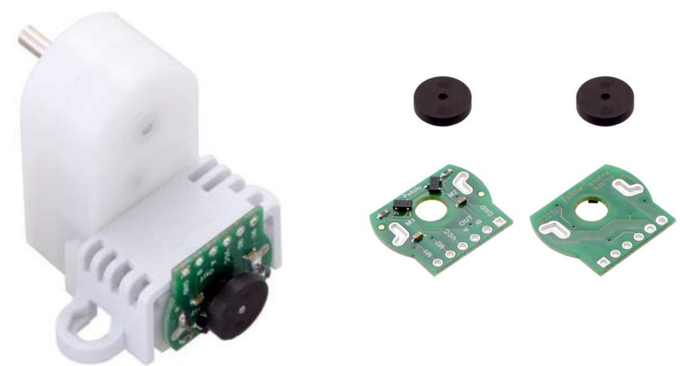
- Example:

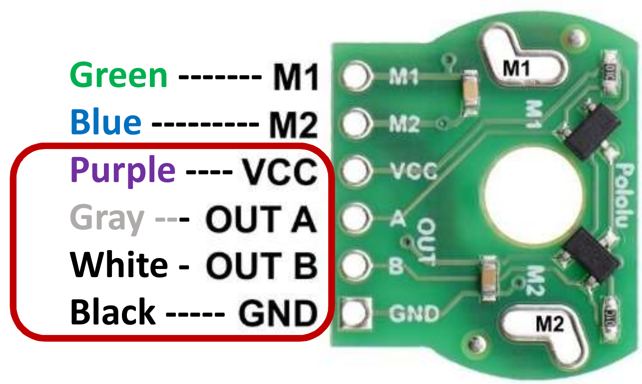
```
E0:0
E1:0
A4:312
E0:0
E1:0
A5:799
E0:0
E1:0
A6:312
…
```

Quadrature Encoder

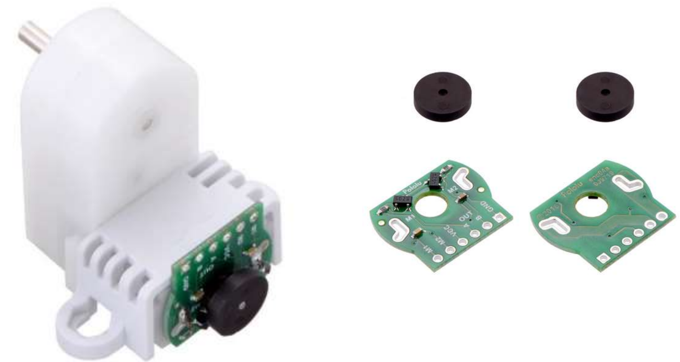
- Operation:
  - Quadrature pulses on lines A and B
  - Runs on 2.7-18V

  - [Pololu 1523]()
  - 12 Counts per Revolution (of the Motor shaft, not the Output shaft)
  - Note: Motor has 120:1 gear ratio

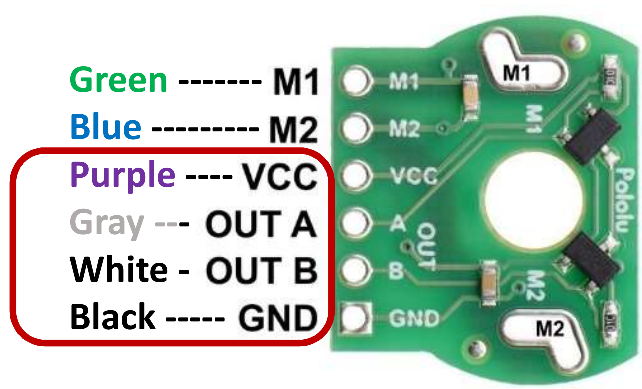




Quadrature Encoder (ctd.)



- Operation:
  - Quadrature pulses on lines A and B
  - Plug into Digital ports on Raspberry Pi:
    - **Left**: Encoder 1: GPIO 17+27
    - **Right**: Encoder 2: GPIO 23+24
- Reading the Encoders
  - A Linux Kernel Module "polls" those pins every time there's an "edge" on any line
  - Updates the encoder count
  - Maintains E1 and E2 counts.
  - Launch the kernel module before using!



Quadrature Encoder (ctd.)

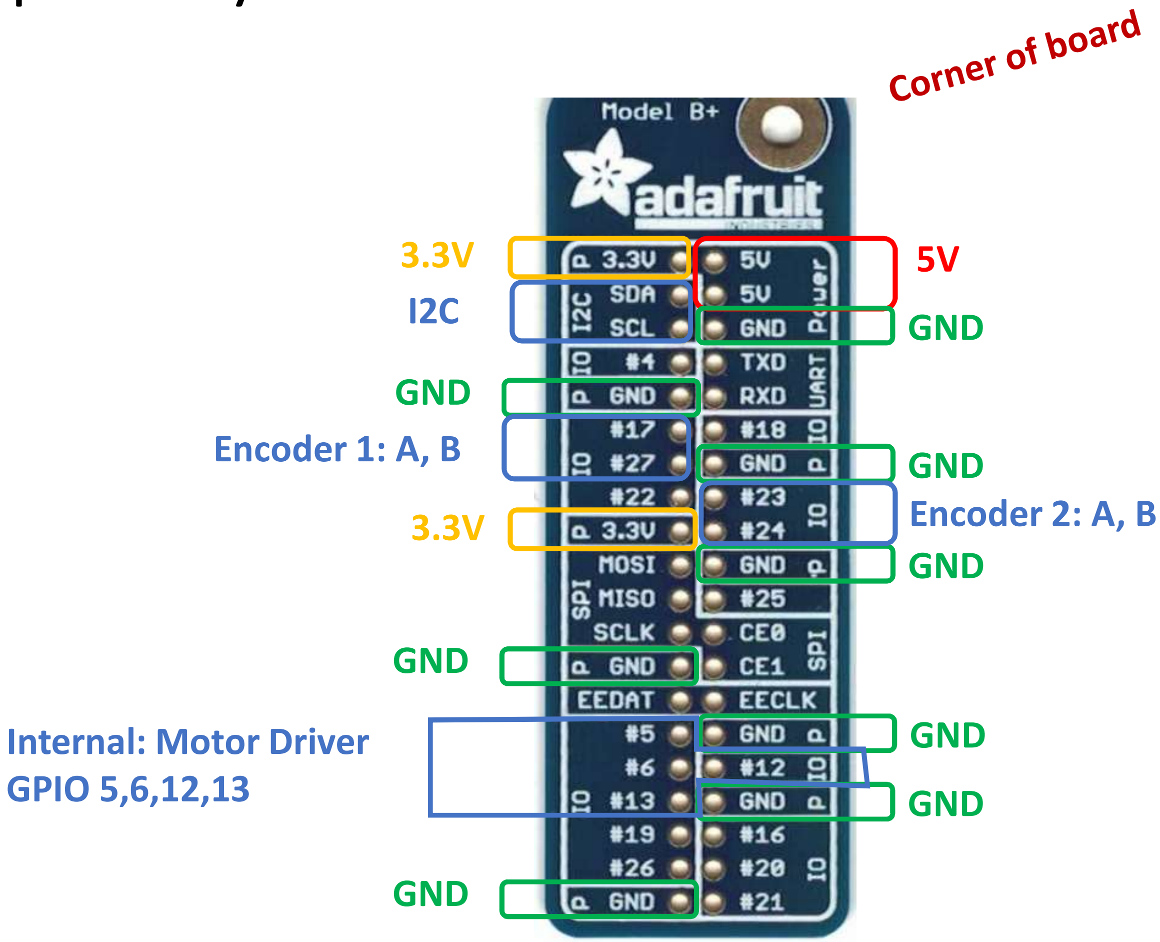
- NOTE: Encoder and Motor Signs

<table>
<tr><td colspan="2" rowspan="2">Example – Depends on each Robot</td><td colspan="2">Encoder Sign</td></tr>
<tr><td>-1</td><td>1</td></tr>
<tr><td rowspan="2">Motor Sign</td><td>-1</td><td>Backward?</td><td>Unstable Backward?</td></tr>
<tr><td>1</td><td>Unstable Forward?</td><td>Correct?</td></tr>
</table>

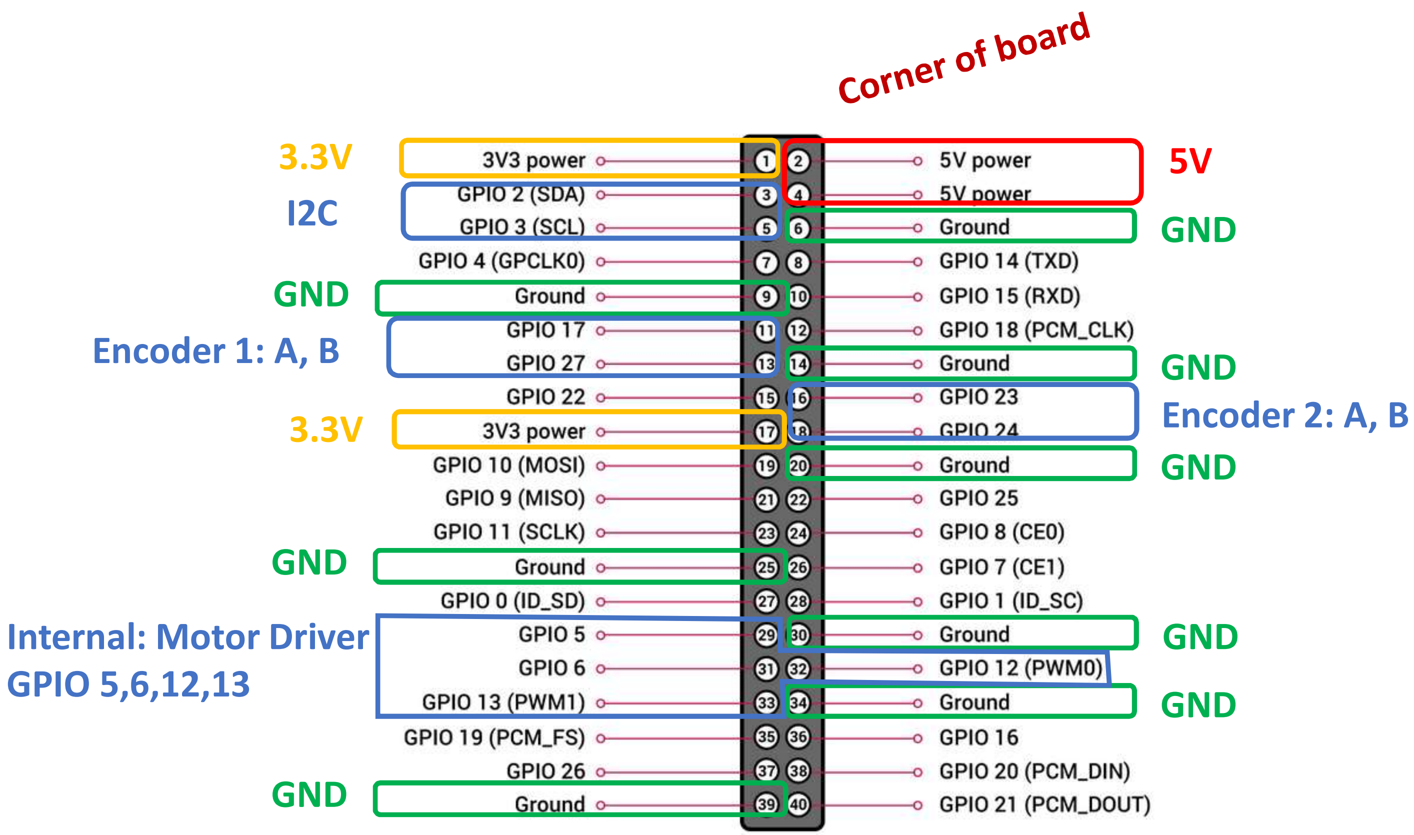
2 values for each = 4 combinations
2 Unstable, 1 Backward, 1 Correct
Depends on wiring and physical placement (Left/Right)

Later use “motor_encoder_test.py” to help troubleshoot

Raspberry Pi Connections:



Raspberry Pi Connections:



Encoder Code – Linux Kernel Driver

- Create a new file and copy/paste code at right: ~/ClonedRepos/encodersetup.sh
- **`nano ~/ClonedRepos/encodersetup.sh`**
  - `Paste in the code →`
- **`cd ~/ClonedRepos`**
- **`Execute:`**
  - **`sudo bash encodersetup.sh`**

encodersetup.sh contents:

```
#! /bin/bash
git clone https://github.com/jtabor/RaspberryPiKernelEncoder.git
sudo chown -R pi RaspberryPiKernelEncoder
apt -y install raspberrypi-kernel-headers
apt -y install --reinstall raspberrypi-bootloader raspberrypi-kernel raspberrypi-kernel-headers
cd RaspberryPiKernelEncoder
sudo -u pi make
sudo make load
```

- To launch each time you start the RasPi:
- **`cd ~/ClonedRepos/RaspberryPiKernelEncoder`**
- **`sudo make load`**

- To test:
- **`cd ~/ClonedRepos/RaspberryPiKernelEncoder`**
- **`python3 readEncodersLoop.py`**

- Then turn the wheels by hand and see what happens!

Tutorial: Encoder Read-and-Publish Node

- Completed file in basic_motors_and_sensors.zip:
  src folder, "**encoder_reading_node.py**"

- Setup:
  - Go into ~/catkin_ws/src
  - Enter or Create pkg: basic_motors_and_sensors
    - Dependencies: rospy, std_msgs, geometry_msgs
  - Create a file "**encoder_reading_node.py**" in there
  - **Copy in "encoders.py"** from ClonedRepos/RaspberryPiKernelEncoder
  - Set PY files "executable"
    - `chmod +x ~/catkin_ws/src/basic_motors_and_sensors/src/*.py`

Arduino Sensors in ROS

- **“basic_sensors_and_motors.zip”**
  - ROS nodes
  - Unzip and upload to catkin_ws/src
  - `cd ~/catkin_ws`
  - `catkin_make`

  - Modify sensors_node.py
    - Publish specific sensor data as ROS topics (Int32 message)
    - Publish topic(s) for only the sensor(s) you are using
  - Modify sensors_processor.py
    - Subscribe to above…
    - Calibrate/process data – to Physical measurement
    - Publish processed data as ROS topic (Float32)

- To view data:
  - `rostopic echo ________`
  - `rosrun rqt_plot rqt_plot`

- Use Different Sensors
- Sense Different Objects
- Use Different Ports
- Check Range, Accuracy, Variability
- Consider strengths/weaknesses

- Make your own calibration
  - Convert raw values to meaningful measurements
  - E.g. use numpy.interp()
  - https://docs.scipy.org/doc/numpy/reference/generated/numpy.interp.html

- Remember to make the files Executable:
- `chmod +x *.py`

Sensors – More Task Options

- Quadrature Encoder
  - Conversion functions
    - Encoder counts → wheel angle → wheel distance traveled
  - Verify angle readings for one revolution
  - Verify distance traveled for *n* revolutions.
- Ultrasonic Sensor
  - Convert Analog reading to Distance to nearest object (meters)
  - Edit function to make this calculation
- Light/Dark sensor
  - Threshold rule (high vs. low)

- ~~Analog Rangefinder~~
  - ~~Convert analog values to Distance~~
  - ~~Write a function to make this calculation~~

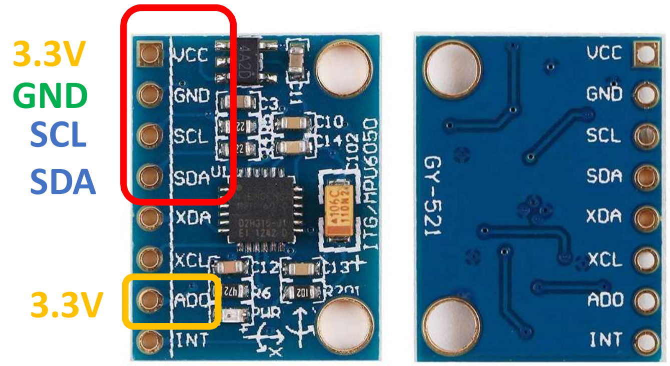
Digital Sensors

- Sensors that have sampling and signal processing onboard, internal.
- They report data through digital communications:
  - **i2c protocol**: “inter-integrated circuit”, “i-squared c”
    - I2C is a “bus” – many devices can be attached to the same set of wires, and the communications specify which one the “master” is “talking with” at any time.
    - Each device has an Address; the Master sends an Address at the start of the communication, and then that device will respond (accept and/or return data)
    - Two wires: SDA (Serial DAta) and SCL (Serial CLock) – sometimes just called “Two Wire”

Inertial Measurement Unit

- Invensense MPU 6050*
  - 3-axis Accelerometer
  - 3-axis Angular Rate Gyroscope
  - * NO magnetometer
  - I2C **digital** communications
    - 2-line communication: SCL, SDA
      - **S**erial **CL**ock, **S**erial **DA**ta
    - Many sensors on one “bus”
      - Every communication is “addressed”
    - MPU 6050 default address is 0x68
      - hexadecimal 0x68 = decimal 104
      - * Conflicts with AlaMode (Real-Time Clock)
    - Connect **AD0** to +3.3V → changes address to **0x69**
      - 3.3V: RasPi pins: **9th pin of the Medial row**
    - See code on Canvas i2c_sensors/imu_mpu6050_node.py
  - See also MPU-9250 and other recent versions

- https://smile.amazon.com/dp/B008BOPN40



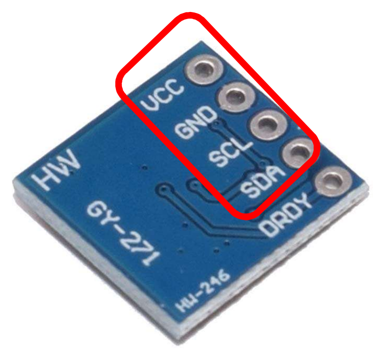
*Breakout module called “GY-521” for unknown reasons

Corner of RasPi

Magnetometer Unit

- Honeywell HMC5883*
  - 3-axis Magnetometer
  - I2C communications
    - Address: 0x1E
      - hexadecimal 0x1E = decimal 30
    - See code on Canvas

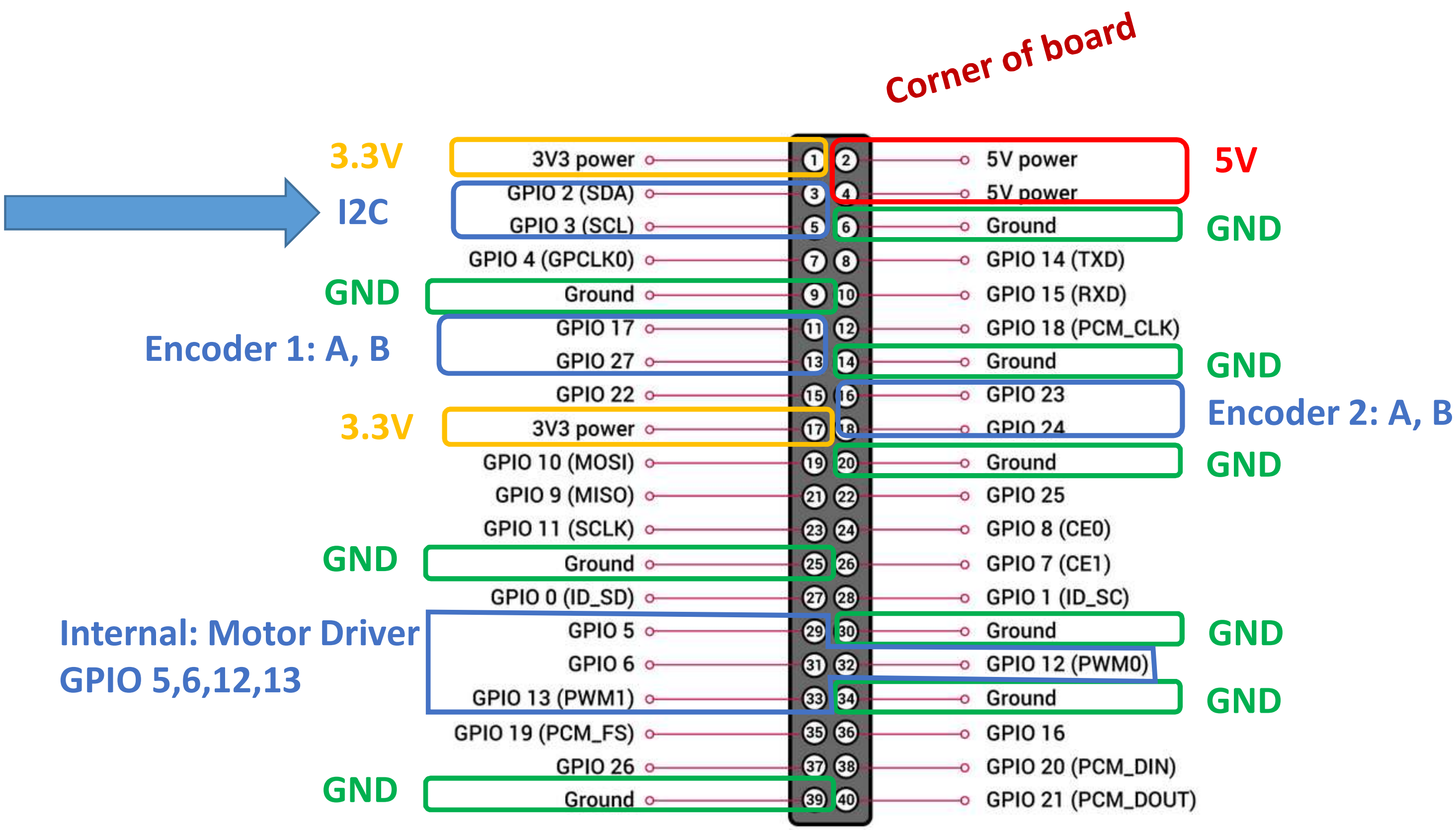
- https://www.amazon.com/dp/B008V9S64E



Corner of RasPi

*Breakout module called "GY-271" for unknown reasons

Raspberry Pi Connections:



ROS Digital Sensors

- ROS digital sensors
  - IMU (MPU 6050)
  - **MPU-6050 IMU.zip** (Example Code)
    - See ROS Node
    - Publishes an IMU topic
      - sensor_msgs.msg: imu
      - https://docs.ros.org/api/sensor_msgs/html/msg/Imu.html
  - Magnetometer HMC-5883
  - **HMC-5883 Magnetometer.zip** (example)

- Instructions:
- Create new package: i2c_sensors
  - `cd ~/catkin_ws/src`
  - `catkin_create_pkg i2c_sensors std_msgs geometry_msgs sensor_msgs rospy roscpp message_generation message_runtime`
- Download contents from Canvas, put in "src" folder of the package
  - ~/catkin_ws/src/i2c_sensors/src/
- Then "make" the workspace:
  - `catkin_make`

Digital Sensors – More Task Options

- IMU (MPU-6050)
  - Determine “spin axis” (gyro data)
    - Make robot spin
    - Read data from IMU gyro
    - Compute a unit vector for this axis!
  - Continually estimate spin angle
    - integrate ang vel about spin axis

- Magnetometer (HMC5883)
  - Determine “North”

Old

Ultrasonic Range Sensor

- Operation in Class:
  - Connections
    - GND and +5V supply (from Analog section)
    - Trigger/Echo:
      - [D2/D3], [D4/D5], or [D6/D7]
  - Arduino:
    - "SensorManager_Multirate.ino" file polls all the US transducers.
    - Sends packet e.g. "U0:1234" over Serial line at ~**20 Hz** (Total).
      - (unit: microseconds)
  - Raspberry Pi (Python):
    - Receive packet over Serial
    - Decode
    - Store data in variables

- During Class:
  - Use 1-3 US Transducers
  - Sense different objects
  - Consider strengths/weaknesses

  - Range
  - Field of View
  - Echoes
  - Speed
  - Material Sensed