

# Robotics 311 : How to build robots and make them move

Prof. Elliott Rouse

GSI Yves Nazon MS

Fall 2022



# ROB 311 – Lecture 14

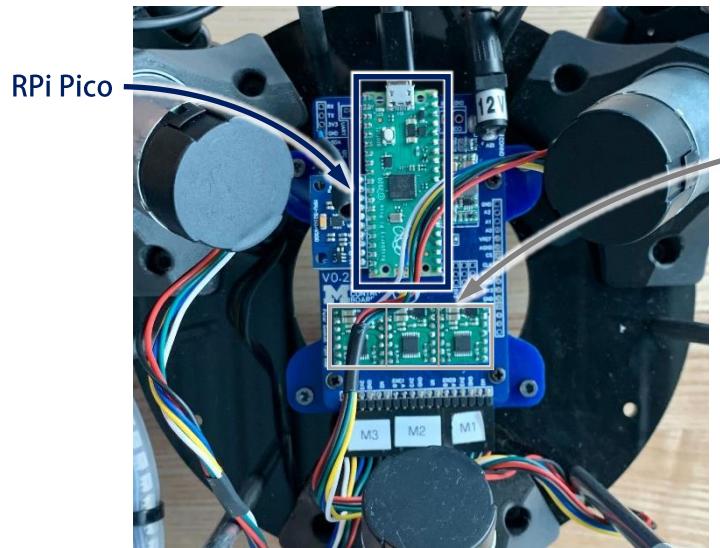
- Today:
  - Review Lab 8
  - Review data acquisition
  - Discuss IMUs
  - Discuss encoders
  - Discuss omni wheels
- Announcements
  - HW 3 due
  - HW 4 will be posted today / soon
  - Midterm exam – 11/8

# Raspberry Pi Pico

- Robotics projects often include the integration of sensing, control, and actuation
- We will use the U-M Robotics Pico board to help with sensing, actuation, and communication
- It uses an RPi Pico—we will use this setup for its convenience and motor drivers, but we could run the system without it
- Your Pico will act as a pass-through—it does not do any computation and instead, collects data, shares with the RPi, and runs the motor drivers



Raspberry Pi Pico  
[\(link\)](#)



Pololu Motor  
Drivers  
(1 per motor)

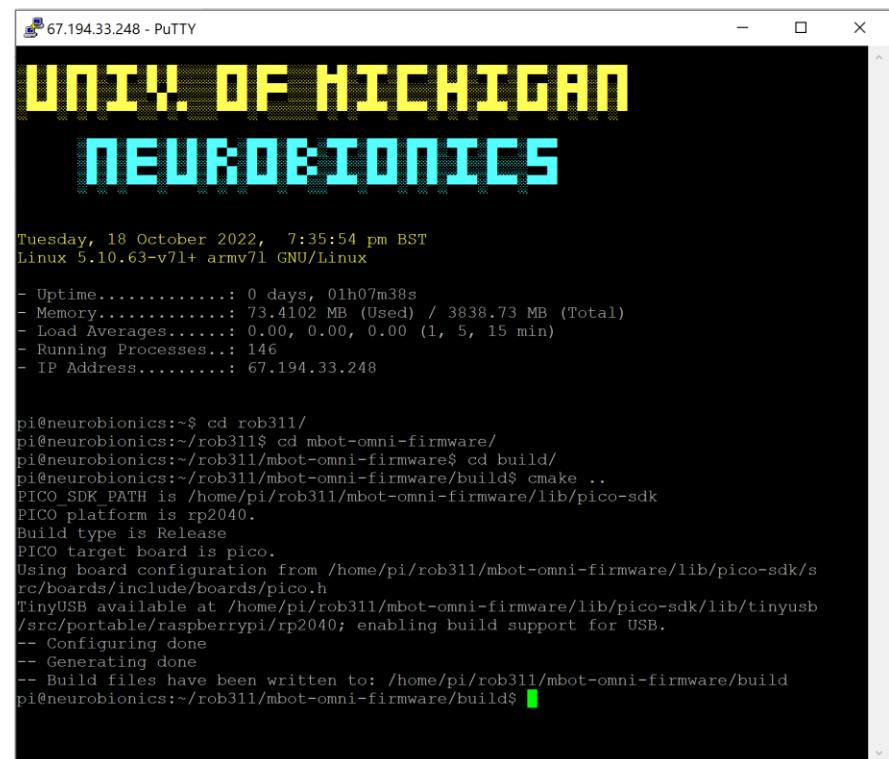


Pololu DRV8874  
[\(link\)](#)

# Flashing the Pico

- Connect to your RPi using its IP address
- Navigate to `~/rob311/mbot-omni-firmware/build`
- If the build directory does not exist, you can make it with `mkdir build`
- Then, run the following commands to compile the firmware
- After compiling the binaries, we will download onto the Pico

```
cd rob311
cd mbot-omni-firmware
cd build
cmake ..
make
```



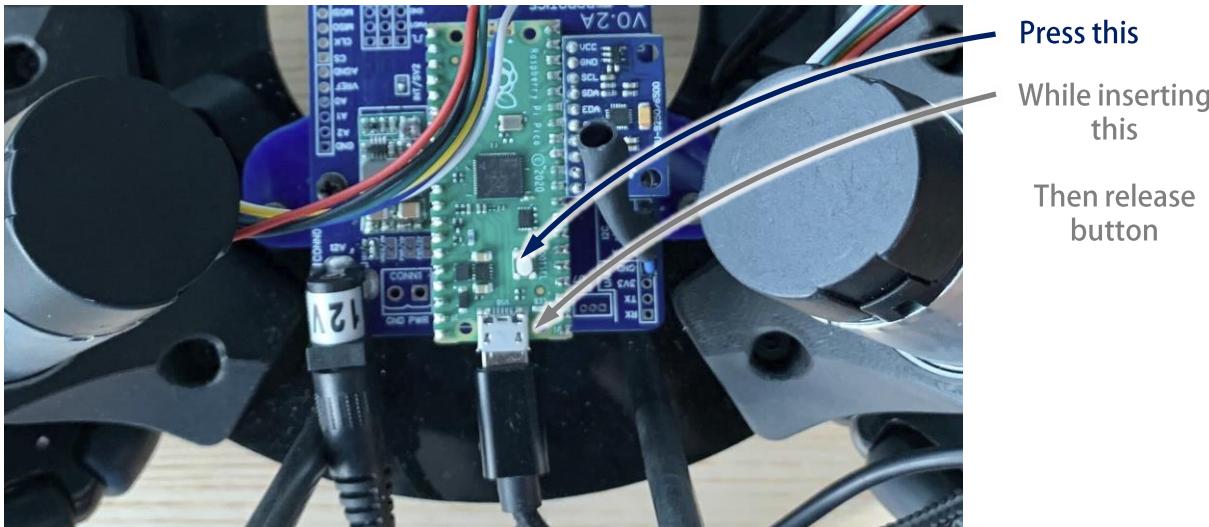
```
67.194.33.248 - PuTTY
UNIV. OF MICHIGAN
NEUROBIONICS
Tuesday, 18 October 2022, 7:35:54 pm BST
Linux 5.10.63-v7l+ armv7l GNU/Linux

- Uptime.....: 0 days, 01h07m38s
- Memory.....: 73.4102 MB (Used) / 3838.73 MB (Total)
- Load Averages...: 0.00, 0.00, 0.00 (1, 5, 15 min)
- Running Processes...: 146
- IP Address.....: 67.194.33.248

pi@neurobionics:~$ cd rob311/
pi@neurobionics:~/rob311$ cd mbot-omni-firmware/
pi@neurobionics:~/rob311/mbot-omni-firmware$ cd build/
pi@neurobionics:~/rob311/mbot-omni-firmware/build$ cmake ..
PICO_SDK_PATH is /home/pi/rob311/mbot-omni-firmware/lib/pico-sdk
PICO platform is rp2040.
Build type is Release
PICO target board is pico.
Using board configuration from /home/pi/rob311/mbot-omni-firmware/lib/pico-sdk/s
rc/boards/include/boards/pico.h
TinyUSB available at /home/pi/rob311/mbot-omni-firmware/lib/pico-sdk/lib/tinyusb
/src/portable/raspberrypi/rp2040; enabling build support for USB.
-- Configuring done
-- Generating done
-- Build files have been written to: /home/pi/rob311/mbot-omni-firmware/build
pi@neurobionics:~/rob311/mbot-omni-firmware/build$
```

# Flashing the Pico

- Before we flash the Pico, we need to tell it that we will be sending it new instructions
- This is done by pressing the white button while the micro USB connector is inserted
- This tells the Pico to bootload the instructions



- Now you can run the picoload script by typing the following in the terminal:  
`./picoload /dev/sda1`
- When completed, it should echo ‘Loaded UF2 !’ without errors

```
pi@neurobionics:~/rob311$ ./picoload /dev/sda1
bballbot.uf2  INDEX.HTM  INFO_UF2.TXT
Loaded UF2!
pi@neurobionics:~/rob311$ █
```

# Troubleshooting

- If `cmake` or `make` is not recognized, we may need to add some packages
- In the terminal, execute the following commands:

```
sudo apt-get update
```

```
sudo apt-get install cmake
```

```
sudo apt-get install gcc-arm-none-eabi
```

- If `picoload` fails, you may need to create a directory

```
cd /mnt
```

```
sudo mkdir pico
```

# Troubleshooting

- If it doesn't work and you've been unplugging / re-plugging in your Pico, your device name may have changed
- Your device can be found by looking in `/dev/` folder (`cd /dev`), then `ls`
- To check, look for `sda1`, `sdb1`, `sdc1`, etc.
- This needs to be the correct device name in `./picoload /dev/sda1`

Needs to  
match `/dev`

```
pi@neurobionics:/dev$ ls
autofs      hwrng      mmcblk0p2   ram6          stdio     tty24    tty42    tty60        vcs4       video10
block       initctl      mmcblk0p1   ram7          stdio     tty25    tty43    tty61        vcs5       video11
bsq         input        net        ram8          stdio     tty26    tty44    tty62        vcs6       video12
btrfs-control  kmsg      null       ram9          stdio     tty27    tty45    tty63        vcsa      video13
bus         log         port       random        stdio     tty28    tty46    tty7         vcsal     video14
cachefiles   loop0     ppp        raw           stdio     tty29    tty47    tty8         vcsa2     video15
cec0        loop1     ptmx      rfkill        stdio     tty30    tty48    tty9         vcsa3     video16
cec1        loop2     pts       rpivid-h264mem stdio     tty31    tty49    ttyAMA0      vcsa4     video18
char        loop3     ram0      rpivid-hevcmem stdio     tty32    tty50    uhid        vcsa5     watchdog
console     loop4     ram1      rpivid-intcmem stdio     tty33    tty51    uinput      vcsa6     watchdog0
cuse        loop5     ram10     rpivid-vp9mem  stdio     tty34    tty52    urandom     vcsu      zero
disk        loop6     ram11     sdb           stdio     tty35    tty53    v4l         vcsu1
dma_heap    loop7     ram12     sdb1         stdio     tty36    tty54    vchiq      vcsu2
dri         loop-control ram13    serial1      stdio     tty37    tty55    vcio       vcsu3
fd          mapper      ram14     sg0          stdio     tty38    tty56    vc-mem     vcsu4
full        media0      ram15     shm          stdio     tty39    tty57    vcs        vcsu5
fuse         medial     ram2      snd          stdio     tty40    tty58    vcs1       vcsu6
gpiochip0   mem        ram3      spidev0.0    stdio     tty41    tty59    vcs2       vga_arbiter
gpiochip1   mmcblk0     ram4      spidev0.1    stdio     tty42    tty6     vcs3       vhci
gpiomem     mmcblk0p1   ram5      stderr       stdio     tty43    tty7
pi@neurobionics:/dev$
```

Re-named `sdb1` by  
unplugging and re-  
plugging in

# Troubleshooting

- If you try to run the sensor read demo and get an error (below), it assigned the serial port to a different number, which we need to update

```
pi@neurobionics:~/rob311/ballbot-omni-app$ python R0B311_sensor_read_demo.py
Trial Number? 0
Traceback (most recent call last):
  File "/usr/local/lib/python3.9/dist-packages/serial/serialposix.py", line 322, in open
    self.fd = os.open(self.portstr, os.O_RDWR | os.O_NOCTTY | os.O_NONBLOCK)
FileNotFoundError: [Errno 2] No such file or directory: '/dev/ttyACM0'

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "/home/pi/rob311/ballbot-omni-app/R0B311_sensor_read_demo.py", line 197, in <module>
    ser_dev = SerialProtocol()
  File "/home/pi/rob311/ballbot-omni-app/MBot/SerialProtocol/protocol.py", line 8, in __init__
    self.serial_dev = serial.Serial(device, baud, parity = parity, stopbits = stopbits, timeout = timeout)
  File "/usr/local/lib/python3.9/dist-packages/serial/serialutil.py", line 244, in __init__
    self.open()
  File "/usr/local/lib/python3.9/dist-packages/serial/serialposix.py", line 325, in open
    raise SerialException(msg.errno, "could not open port {}: {}".format(self._port, msg))
serial.serialutil.SerialException: [Errno 2] could not open port /dev/ttyACM0: [Errno 2] No such file or directory: '/dev/ttyACM0'
pi@neurobionics:~/rob311/ballbot-omni-app$
```

[Errno 2] no such file  
or directory  
/dev/ttyACM0

- To address this, we need to change the port name in  
~/rob311/ballbot-omni-firmware/MBot/SerialProtocol/protocol.py

```
mpu6050.py          protocol.py
File Edit Selection Find Goto Tools Project Preferences Help
1  from time import sleep
2  import serial
3  import numpy as np
4  from threading import Lock
5
6  class SerialProtocol:
7      def __init__(self, device = "/dev/ttyACM1", baud=115200, parity=serial.PARITY_NONE, stopbits=serial.STOPBITS_ONE, bytesize=serial.EIGHTBITS, timeout=0.1):
8          self.serial_dev = serial.Serial(device, baud, parity = parity, stopbits = stopbits, timeout = timeout)
9          self.running = True
10         self.ROS_HEADER_LENGTH = 7
11         self.data_dict = {}
12         self.serializer_dict = {}
13         self.endianness = endian
14
15     def checksum(self, addends):
16         sum = np.sum(addends)
17         return 255 - (sum % 256)
18
19     def get_cur_topic_data(self, topic_id):
20         lock = self.data_dict[topic_id][0]
21         lock.acquire()
22         to_return = np.copy(self.data_dict[topic_id][1])
23         lock.release()
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
```

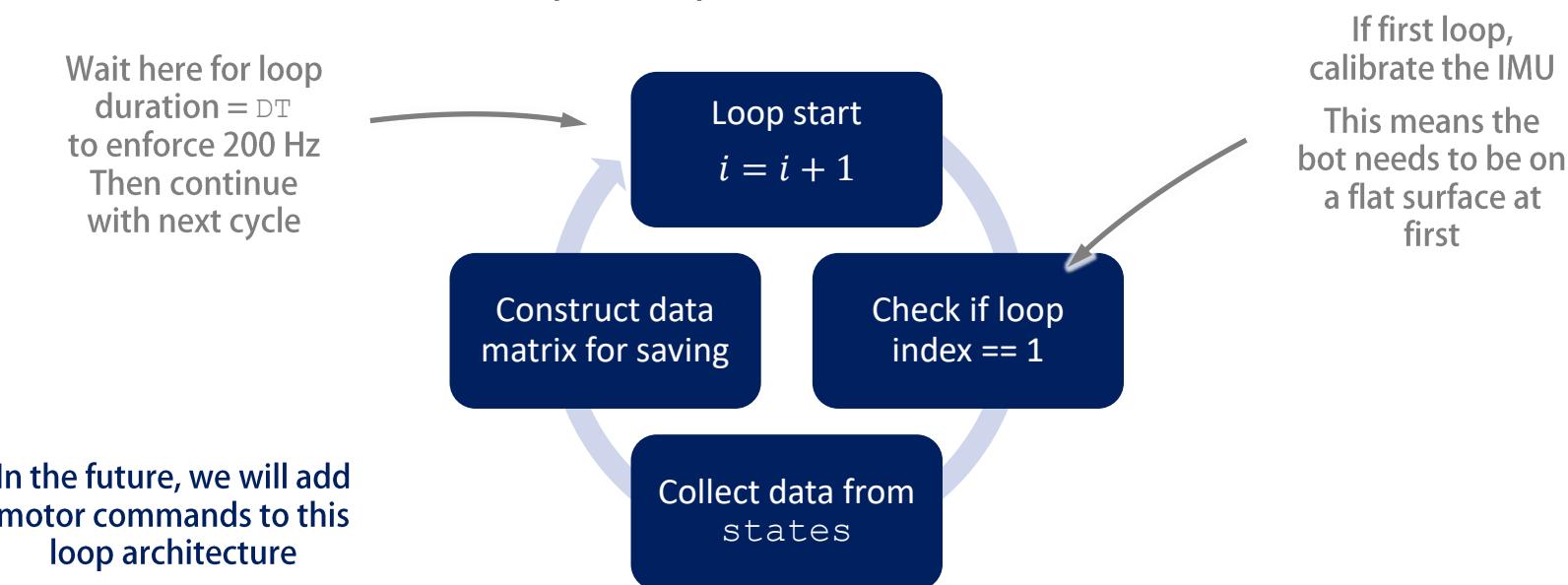
This should be  
changed to  
/dev/ttyACM1

The name should be  
confirmed by  
looking at the port  
names in /dev/

Then reboot your RPi!

# Reading Data From Your RPi

- Now your system is ready to run
- Today, we will read and plot data, but not turn on the motors
- We will read from the IMU and motor encoders
  - The IMU will tell you rotation around the  $x$  and  $y$  axes
    - Units: Radians
  - The motor encoder will tell you the rotation of each motors
    - Units: Radians (I think)



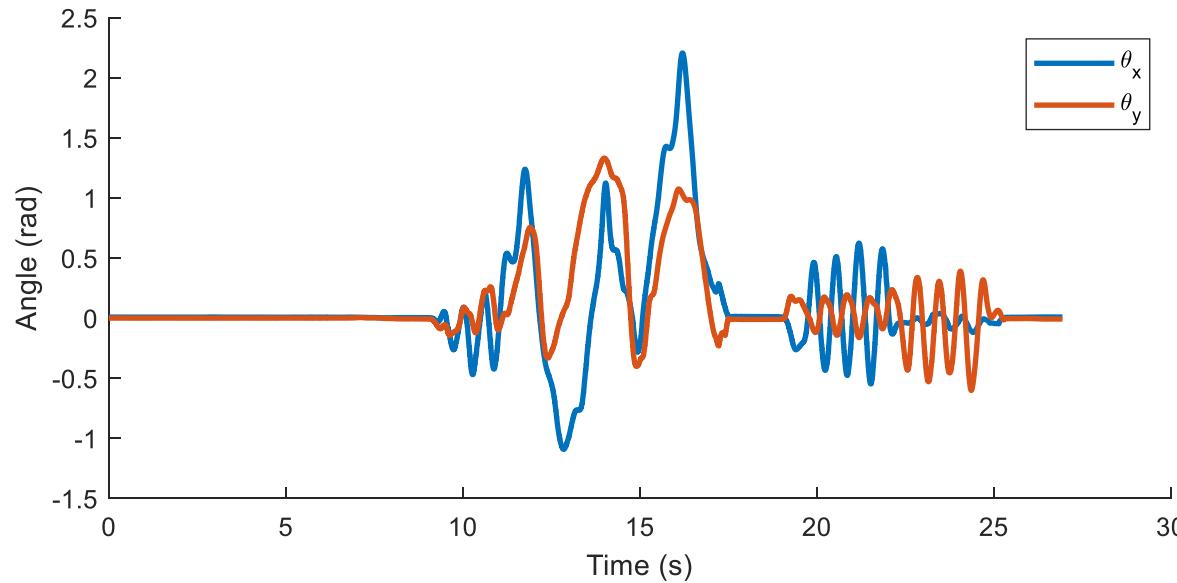
# Reading Data From Your RPi

- To collect IMU data, first download files from Canvas\Labs\Lab 8
- You will need
  - Datalogger.py
  - ROB311\_sensor\_read\_demo.py
  - ROB311\_ball\_bot\_data\_analysis.m
- Add the Python files to the ballbot-omni-app folder on your RPi
- Run the python script using `python ROB311_sensor_read_demo.py`
- It will prompt you for a trial number for the data to be saved
- Calibration must be done while the ball-bot is on a flat surface
- Following calibration, rotate the ball bot around the  $x$  and  $y$  axes
- Use `ctrl+c` to stop the program
- It will save a file named `ROB311_TestX.txt`

This will  
show the trial  
number you  
entered

# Reading Data From Your RPi

- When you're finished, move the saved .txt file from your RPi to your working MATLAB directory.
- Download the m-file from Canvas and move your data to this folder.
- Edit the filename to be sure it matches the name of your .txt file
- Run the m-file and it will plot your  $x$  and  $y$ -axis rotations



# Reading Data From Your RPi

- Place your data from the RPi in the same folder as this MATLAB file
- This enables quick and easy viewing of your data
- We will show a real-time plotter next lab

Edit the filename  
to match

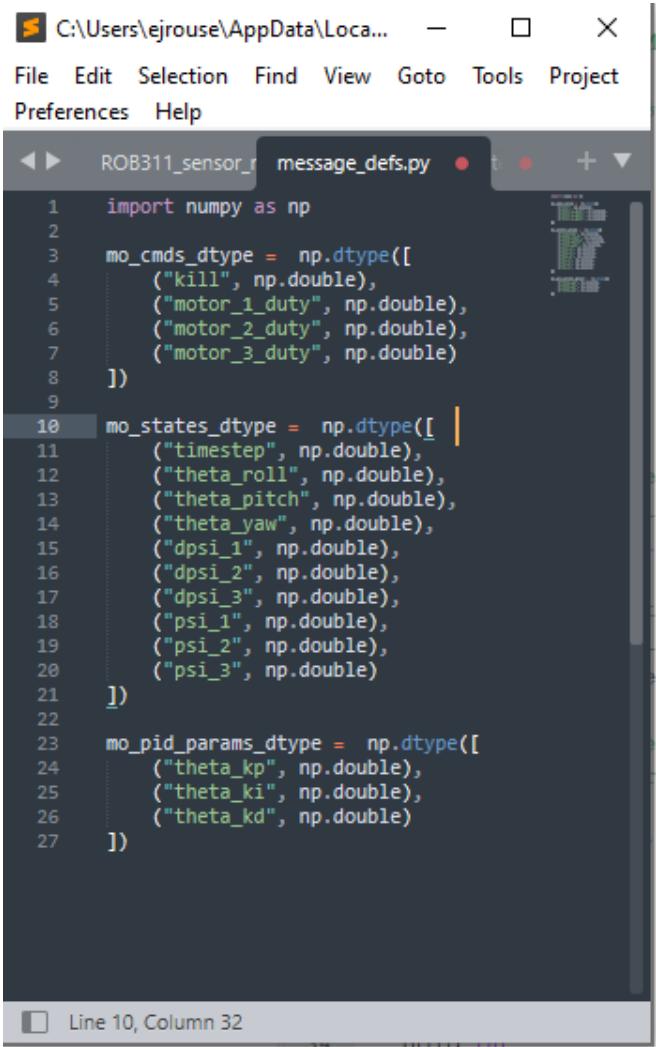
Create data  
vectors (add any  
new variables you  
are saving (in the  
right order)

Plot for  
inspection

```
% ROB 311, Fall 2022
% Data importing / plotting from ball-bot
%
% University of Michigan
%
%%%%%%%%%%%%%
%
clear
clc
close all
%
alpha = pi/4;
beta = pi/2;
Rk = 0.11925;
Rw = 0.04778;
%
% Input file name - this will have to be updated with the name of the trial
% you are analyzing
filename = 'ROB311_Test2.txt';
%
% Use file name to load / create data matrix
eval(['load ' filename ';' ]);
eval(['data = ' erase(filename,'.txt') ';' ]);
%
% Define variables from data - these correspond to the order described in
% the read sensor demo
index = data(:,1);
time = data(:,2);
theta_x = data(:,3);
theta_y = data(:,4);
%
% Plotting
figure
hold on
plot(time, theta_x, 'linewidth',2)
plot(time, theta_y, 'linewidth',2)
legend('\theta_x', '\theta_y')
xlabel('Time (s)')
ylabel('Angle (rad)')
```

# Reading Data From Your RPi

- You have access to many variables collected and sent by the Pico
- These variables are defined in `message_defs.py`
- You can access these variables using the same command we used to get  $x$  and  $y$  axis rotations
- Edit your python script to also save wheel rotations ( $\psi$ )
- Add them to the data matrix to be saved
- Match the formatting / syntax of how the data are received and stored in the data matrix



The screenshot shows a code editor window with the file `message_defs.py` open. The code defines three numpy dtype structures for sensor data:

```
1 import numpy as np
2
3 mo_cmds_dtype = np.dtype([
4     ("kill", np.double),
5     ("motor_1_duty", np.double),
6     ("motor_2_duty", np.double),
7     ("motor_3_duty", np.double)
8 ])
9
10 mo_states_dtype = np.dtype([
11     ("timestep", np.double),
12     ("theta_roll", np.double),
13     ("theta_pitch", np.double),
14     ("theta_yaw", np.double),
15     ("dpsi_1", np.double),
16     ("dpsi_2", np.double),
17     ("dpsi_3", np.double),
18     ("psi_1", np.double),
19     ("psi_2", np.double),
20     ("psi_3", np.double)
21 ])
22
23 mo_pid_params_dtype = np.dtype([
24     ("theta_kp", np.double),
25     ("theta_ki", np.double),
26     ("theta_kd", np.double)
27 ])
```

The status bar at the bottom indicates "Line 10, Column 32".

# Reading Data From Your RPi

Add more variables here

To save variables, add them to this data matrix and remember the order for importing to MATLAB (keep the brackets)

```
# Init serial
serial_read_thread = Thread(target = SerialProtocol.read_loop, args=(ser_dev,), daemon=True)
serial_read_thread.start()

# Local structs
commands = np.zeros(1, dtype=mo_cmds_dtype)[0]
states = np.zeros(1, dtype=mo_states_dtype)[0]

commands['kill'] = 0.0

dpsi = np.zeros((3, 1))

# Time for comms to sync
time.sleep(1.0)

ser_dev.send_topic_data(101, commands)

print('Beginning program!')
i = 0
cal_flag = 0

for t in SoftRealtimeLoop(dt=DT, report=True):
    if i == 0 and cal_flag == 0:
        print('Calibrating...')
        cal_flag = 1
    try:
        states = ser_dev.get_cur_topic_data(121)[0]
    except KeyError as e:
        continue
    if i == 0:
        print('Finished calibration\nStarting loop...')
    i = i + 1
    t_start = time.time()
    t_now = time.time() - t_start

    dpsi[0] = states['dpsi_1']
    dpsi[1] = states['dpsi_2']
    dpsi[2] = states['dpsi_3']

    ser_dev.send_topic_data(101, commands)

    # Define variables for saving / analysis here - below you can create variables from the available states
    theta_x = (states['theta_roll'])
    theta_y = (states['theta_pitch'])

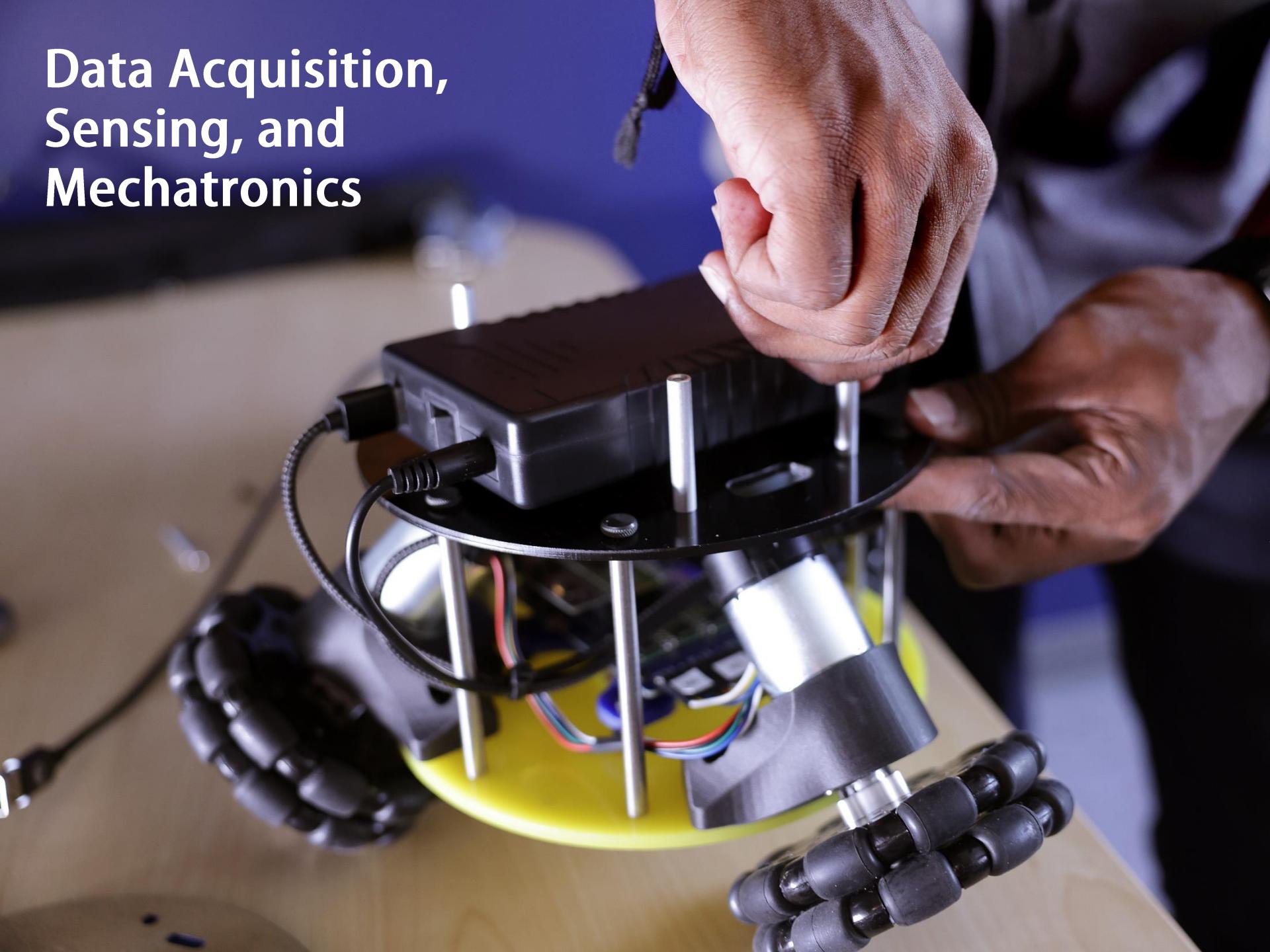
    # Construct the data matrix for saving - you can add more variables by replicating the formula
    data = [i] + [t_now] + [theta_x] + [theta_y]
    dl.appendData(data)

    print("Saving data...")
    dl.writeOut()
    print("Resetting Motor Commands.")
    commands['kill'] = 1.0
    commands['motor_1_duty'] = 0.0
    commands['motor_2_duty'] = 0.0
    commands['motor_3_duty'] = 0.0
    ser_dev.send_topic_data(101, commands)

    time.sleep(DT)

    i = i + 1
```

# Data Acquisition, Sensing, and Mechatronics



# Data Acquisition

- A key aspect of developing a robotic system is sensing the robot state / environment
- For the ball bot, we will want to know
  - Lean angle
  - Ball rotation (obtained from motor rotation)
- In the coming slides, we will learn how use sensors for data acquisition
- Two types of data acquisition
  - Analog: continuous voltages read by a computer and interpreted from the voltage value
  - Digital: information is sent over a communication bus in the form of 1s and 0s that contain the sensor data that has been digitized
- Our ball-bot design only uses digital sensors but we will quickly review analog sensing
  - Previously more common but becoming less common

# Data Acquisition

- Typically, sensors output voltage, which needs to be acquired by a digital computer for analysis and control
- Accomplished using a data acquisition system



- Analog to digital (A2D) converter: digitizes voltage to be read by a computer
- Three key attributes
  - Bits – describes how many ‘bins’ can the voltage be separated into
  - Sample rate (Hz) – how fast the loop runs / A2D converter is sampled
  - Input range (V) – the total range of voltages able to be sampled

$$\text{Resolution} \longrightarrow \frac{\Delta V}{\text{bit}} = \frac{V_{range}}{2^{\text{bits}} - 1}$$

# Analog Data Acquisition

- To calculate the voltage, we sample the A2D system, which will return an integer number of bits
- These bits need to be converted back to voltage
- After converting to volts, it needs to be converted to the original units
  - This is accomplished using the sensors calibration curve or ‘sensitivity’

$$V_{sensor} = V_{min} + \left( b \cdot \frac{V_{range}}{2^{bits} - 1} \right)$$

Number of bits returned from your A2D card

- Example – your 16 bit DAQ system records a value of 11564, with an input range of -10 to 10 V.
- What is the voltage?

$$V = -10 + \left( 11564 \cdot \frac{20}{65535} \right) = -6.47 V$$



USB data acquisition card

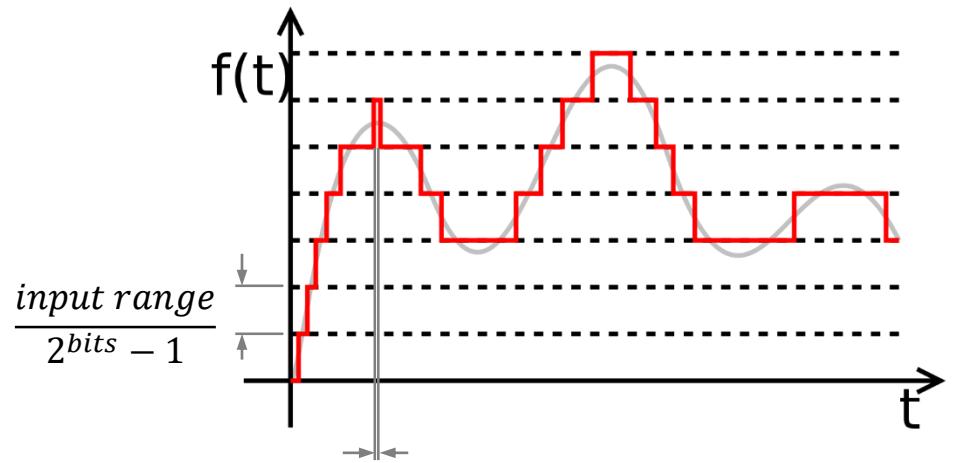
# Analog Data Acquisition

- Signals are sampled periodically, at a frequency governed by the sample rate
- This causes some ‘quantization’ of the analog signal
- The sample rate is a key factor of a data acquisition system
- It governs the frequency content of what can be measured
- Higher sample rates can sense higher frequencies
- Nyquist criterion:

Maximum measurable frequency



$$f_{max} = \frac{1}{2} F_s$$

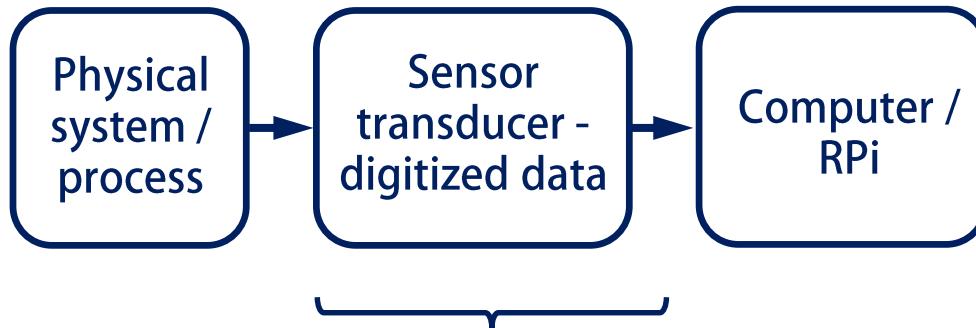


$$\Delta t = \frac{1}{F_s} = \frac{1}{\text{sample rate}}$$

- The highest frequency that can be measured is half the sample rate
- Sample rate / loop rate must be set carefully to make sure the relevant frequencies can be recorded

# Digital Data Acquisition

- Digital sensors are becoming increasingly common
- Data are transmitted using digital communication (USB, I<sup>2</sup>C, etc.)
- No analog voltage – concepts of resolution change for these sensors
  - Governed directly from the sensor, not the A2D card
- Nyquist frequency still applies
- Sampling still applies



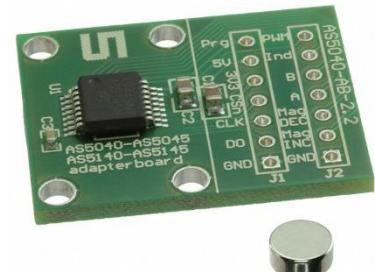
Digital sensors communicate directly with acquisition computer

# Digital Data Acquisition

- Digital sensors are defined by their bit-level resolution
  - For example, 14-bit encoder (rotation sensor)

$$\frac{360^\circ}{bit} = \frac{360^\circ}{2^{14} - 1} = 0.0220$$

- Digital communication busses
  - USB
  - Inter-Integrated Circuit communication ( $I^2C$ )
  - Serial Peripheral Interface (SPI)
  - RS-232 (old school, serial port)
- A program is used to request and interpret information from the sensor
- Information on communication and interpretation is provided in the sensor datasheet
- This program is called a 'driver' or 'API' and is usually written in C or Python



Most common (for now)

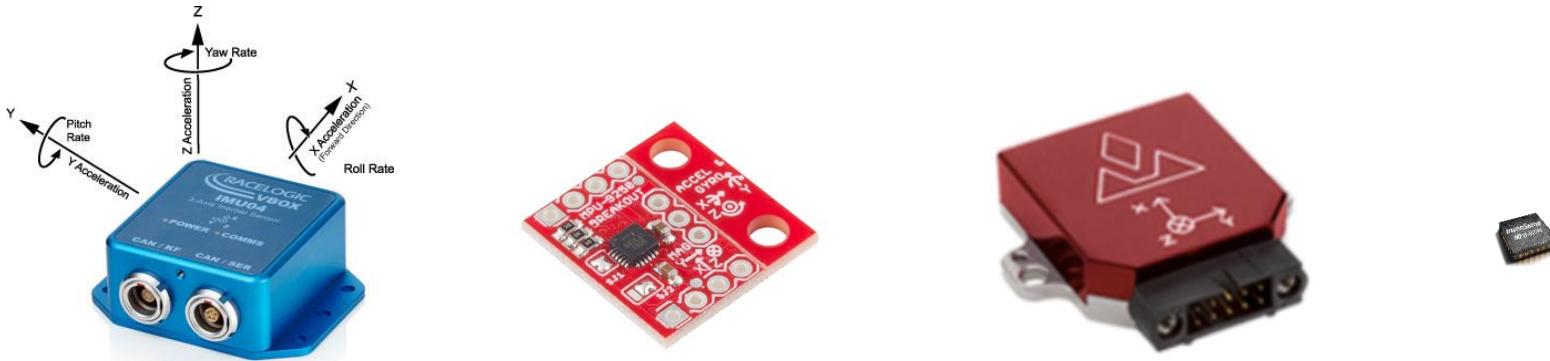
# Sensors

- Many different types of sensors are used in robotic applications
- These sensors convert a physical process into a digital value
- Using sensor characteristics, we can convert the digital value back into physical units
- Common types of sensors:
  - Inertial Measurement Unit – used to tell position / orientation information
    - 3-axis accelerometer
    - 3-axis rate gyroscope
    - 3-axis magnetometer
  - Encoders - used to sense changes in angular displacement
  - Load sensing – force or torque, used in many robotic applications
  - Vision / LIDAR – used for understanding the environment
  - Many others... In this class, we will focus on IMUs and encoders

IMUs are sometimes described by their number of axes or DOFs (e.g. 6-axis or 9-axis)

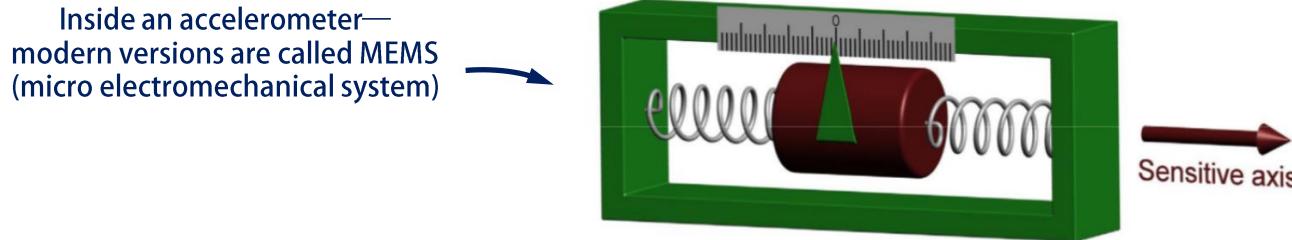
# Inertial Measurement Units

- While there are differences in IMU accuracy and precision, their functions are all relatively similar
- They report  $x$ ,  $y$ , and  $z$  acceleration and  $x$ ,  $y$ ,  $z$  angular velocities
- This information can be fused to obtain global changes to position and orientation—proprietary algorithm within the IMU
  - We use this approach from our IMU, which provides position and orientation of the ball-bot
- Most IMUs can have their accel / velocity ranges scaled, depending on use
  - This provides optimal resolution when scaled correctly



# IMU Mechanism of Action

- Accelerometers measure all external forces, including gravity
- Conceptually, an accelerometer is a spring-mass-damper system



- The force applied is related to the mass, damping, and elasticity of the element
- At equilibrium (zero velocity) the acceleration is  $kx/m$
- Three orthogonal axes are used to obtain omnidirectional measurements
- Mechanical / MEMS accelerometers are low-pass, measuring < 500 Hz
- MEMS gyroscopes operate similarly, with a vibrating mass along the radius which deflects according to angular velocity

- Piezoelectric accelerometers can sense up to 100 kHz!

# IMU Drift

Grade	[mg]	Horizontal Position Error [m]			
		1s	10s	60s	1hr
Navigation	0.025	0.13 mm	12 mm	0.44 m	1.6 km
Tactical	0.3	1.5 mm	150 mm	5.3 m	19 km
Industrial	3	15 mm	1.5 m	53 m	190 km
Automotive	125	620 mm	60 m	2.2 km	7900 km

Bias error—error in MEMS mass—and its effect on integrated position measurement error

Accelerometer Misalignment	[mg]	Horizontal Position Error [m]			
		1s	10s	60s	1hr
0.05°		4.3 mm	0.43 m	15 m	57 km
0.1°		8.6 mm	0.86 m	31 m	110 km
0.5°		43 mm	4.3 m	150 m	570 km
1°		86 mm	8.6 m	310 m	1100 km

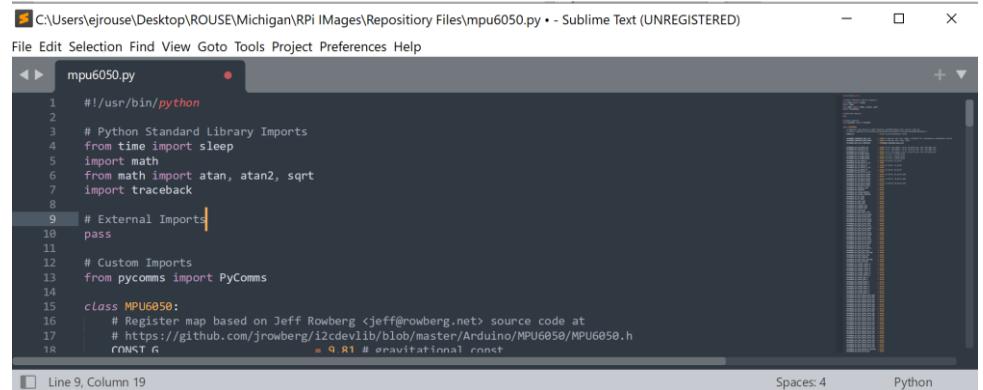
Mounting misalignment error and its effect on integrated position measurement error

Grade	[deg/vhr]	Horizontal Position Error [m]			
		1s	10s	60s	1hr
Navigation	0.002	0.01 mm	0.1 mm	1.3 mm	620 m
Tactical	0.07	0.1 mm	3.2 mm	46 m	22 km
Industrial	3	10 mm	0.23 m	3.3 m	1500 km
Automotive	5	20 mm	0.45 m	6.6 m	3100 km

Gyroscopic random walk (drift) and its effect on integrated position measurement error

# Ball-Bot IMU

- We are using an MPU-6050 knockoff, which was originally created by Invensense
- We use its internal Digital Motion Processor (DMP) which determines the position and orientation for us
  - Why is this hard?
  - We only know linear acceleration and angular velocity—obtaining position / orientation requires integration, which causes drift
  - Complementary / Kalman filters are used to combine the accel / velocity signals inside the IMU
- In our system, the Pico acquires the IMU data—so, interacting with the IMU has been abstracted from you
- A Raspberry Pi can also use this IMU, but it needs a driver / API
- MPU-6050 Python driver / API uploaded to Canvas as an example



```
C:\Users\ejruse\Desktop\ROUSE\Michigan\RPI IMages\Repository Files\mpu6050.py • - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
mpu6050.py
1  #!/usr/bin/python
2
3  # Python Standard Library Imports
4  from time import sleep
5  import math
6  from math import atan, atan2, sqrt
7  import traceback
8
9  # External Imports
10 pass
11
12 # Custom Imports
13 from pycomics import PyComms
14
15 class MPU6050:
16     # Register map based on Jeff Rowberg <jeff@rowberg.net> source code at
17     # https://github.com/jrowberg/i2cdevlib/blob/master/Arduino/MPU6050/MPU6050.h
18     CONST_G = 9.81 # gravitational const

Line 9, Column 19
Spaces: 4
Python
```

# Encoders

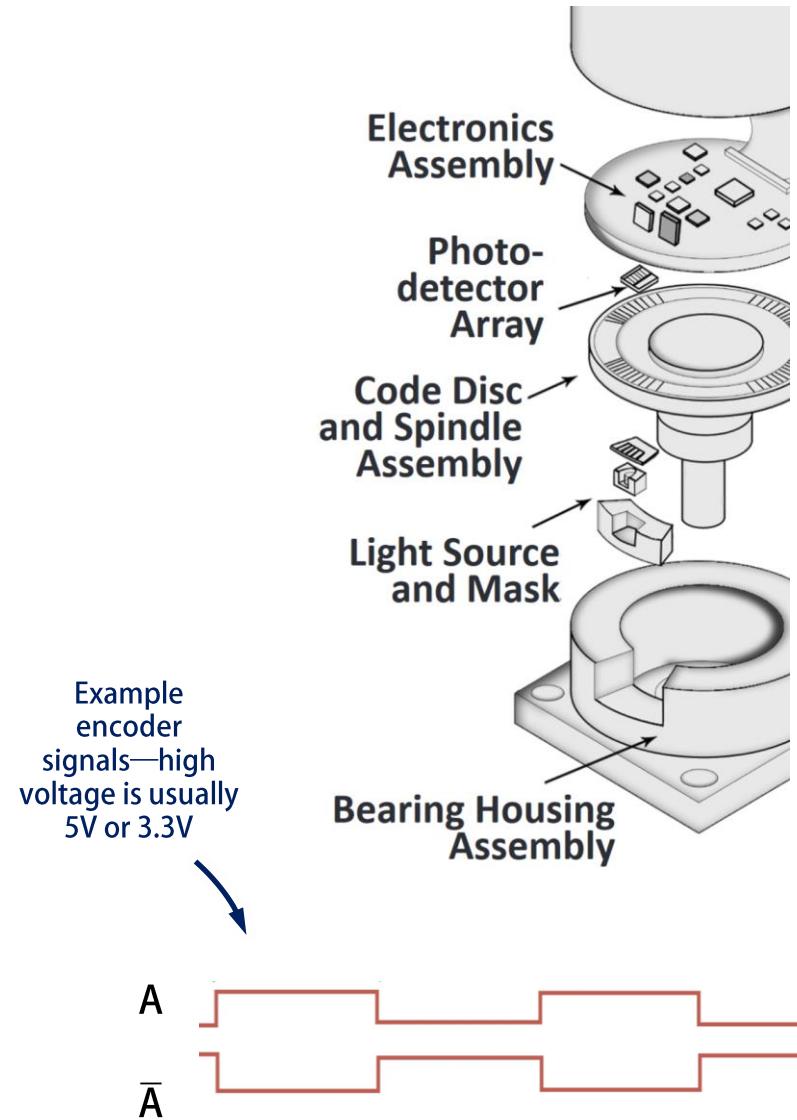
- Encoders are used to measure change in angular position
- Extremely common in robotics and often required for low-level motor control
- Two categories of encoders: relative and absolute
  - Relative: only total traveled distance can be known, but not global orientation
  - Absolute: angle measured relative to global rotation—can measure distance traveled in relation to a fixed global coordinate system
- Many different mechanisms
  - Optical – very common, low cost, and what we're using on our ball-bots
  - Magnetic – very common, low-ish cost, what I use in my research
  - Hall effect – common, low-cost, but very coarse
  - Potentiometers – common, low cost, not robust, analog



We will discuss these

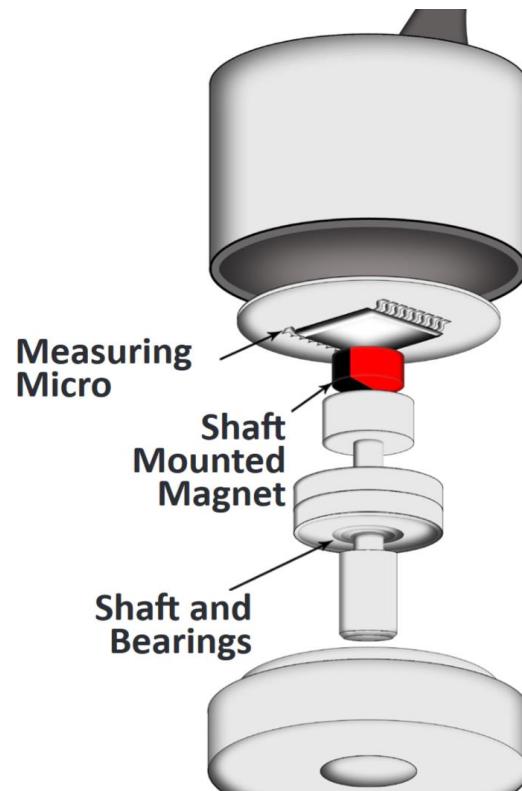
# Optical Encoders

- Optical encoders used interruption of light to detect rotary motion
- A rotating disk inside encodes opaque lines or patterns
- When rotated, a photodetector senses the pulsing light
- This information is then converted and sent out as pulses for the motor driver
- Signals are often sent with the digital opposite, usually noted with a bar
  - Signals A and  $\bar{A}$
- Pros: High resolution, resistant to magnetic interference, low cost
- Cons: Complex / fragile, larger size



# Magnetic Encoders

- Magnetic encoders use the change in local magnetic fields to sense rotation
- Typically based on the Hall-Effect, where a magnetic field across a plate creates a voltage
- Magnet is diametrically magnetized and at a specified location (~1-3 mm away from sensor)
- Outputs can have many forms (sine/cosine, digital, pulses)
- Pros: High resolution, absolute, low cost, simple
- Cons: ~susceptible to magnetic interference
- I use the AMS AS5048 encoder in my designs ([link](#))



AMS AS5048

# Omni Wheels

- Small disks / rollers surround the circumference of an outer wheel
- Rollers roll perpendicular to the turning direction
- Enables full specification of position and orientation of a robot
- Placement on our (ideal) omni wheels on the ball-bot enables no sliding
- Ours have large radial deviation (not smooth)
- Many different versions



- Mechanum wheels use a similar principle—diagonal rollers

