Robotics 311 : How to build robots and make them move

Prof. Elliott Rouse

GSI Yves Nazon MS

Fall 2022

Photo credit: Brenda Ahearn / U-M CoE

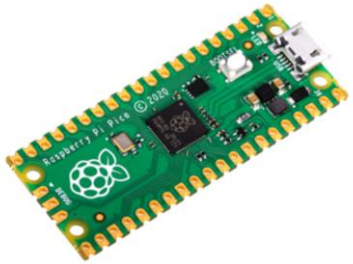
ROB 311 – Lecture 14

- Today:
  - Review Lab 8
  - Review data acquisition
  - Discuss IMUs
  - Discuss encoders
  - Discuss omni wheels

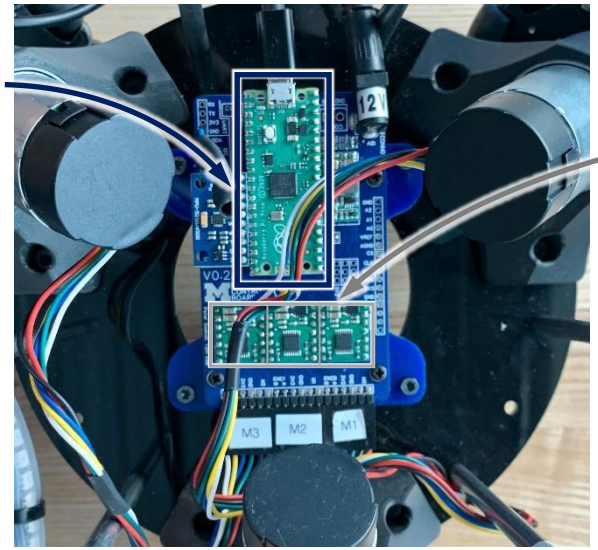
- Announcements
  - HW 3 due
  - HW 4 will be posted today / soon
  - Midterm exam – 11/8

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)

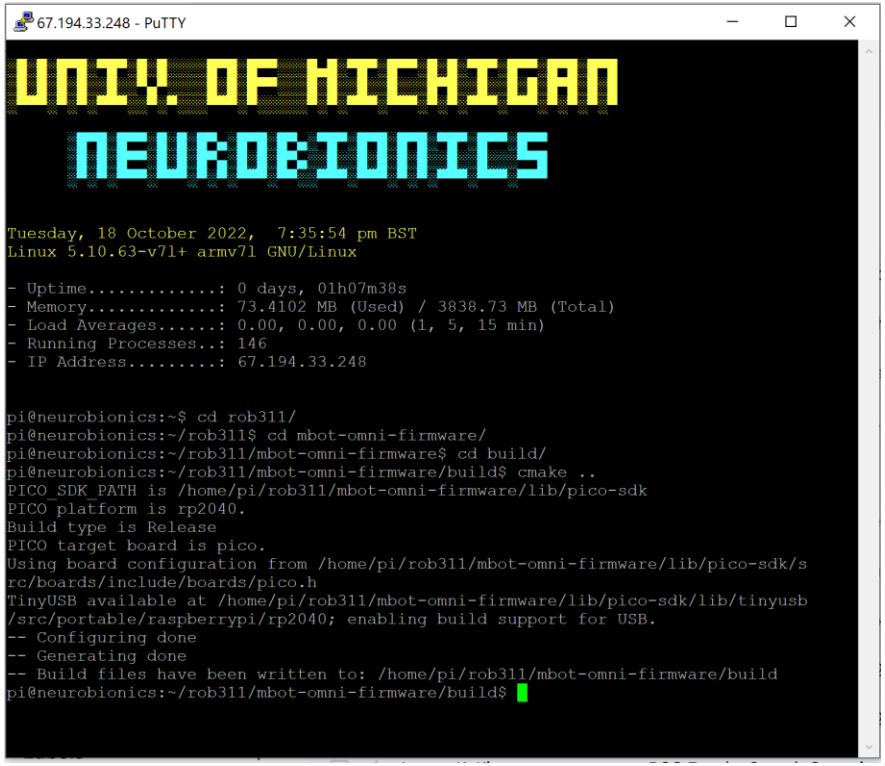


Pololu DRV8874
(link)

Flashing the Pico

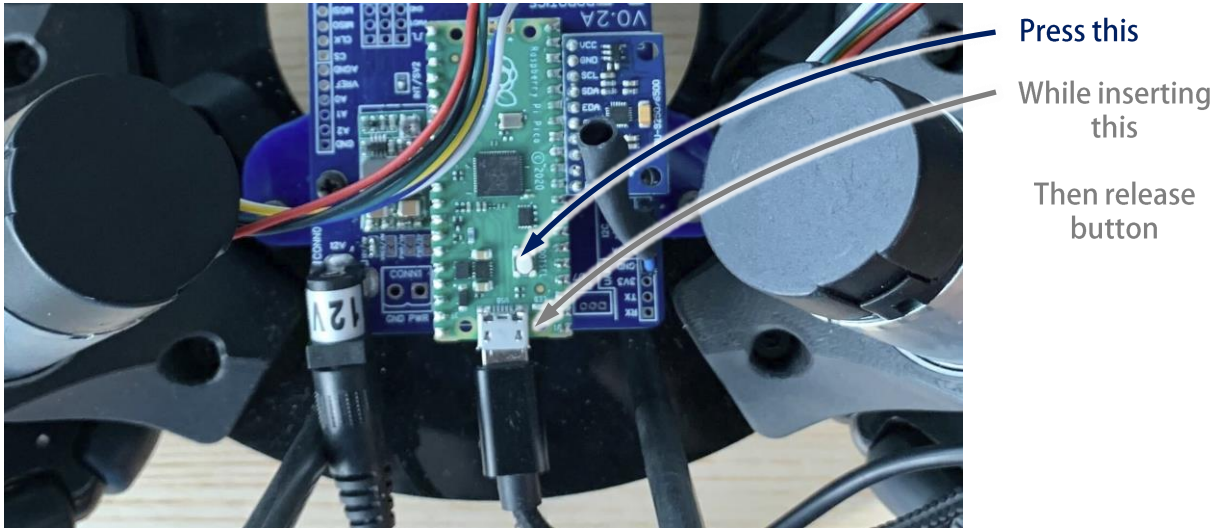
- Connect to your RPi using its IP address
- Navigate to `~/rob311/mbot-omni-firmware/build`
- If the build directory does not exist, you can make it with `mkdir build`
- Then, run the following commands to compile the firmware
- After compiling the binaries, we will download onto the Pico

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



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

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
```

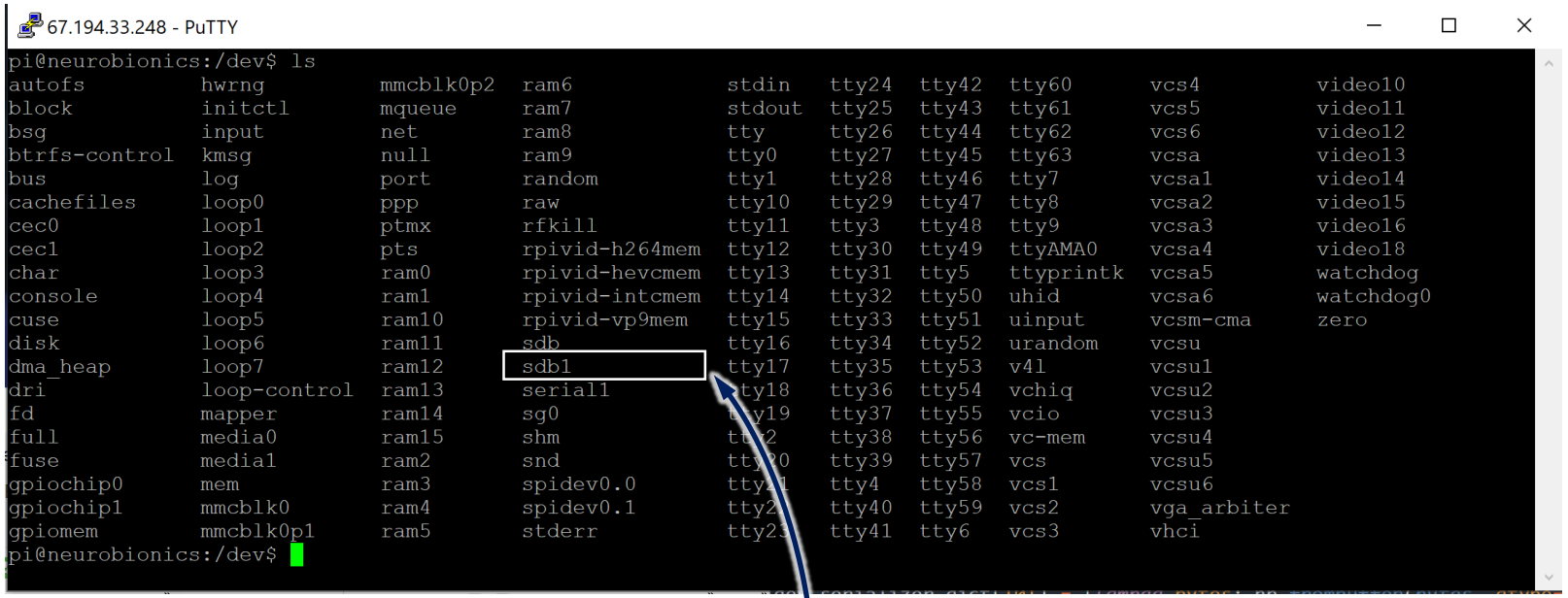
- If picoload fails, you may need to create a directory

```
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



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

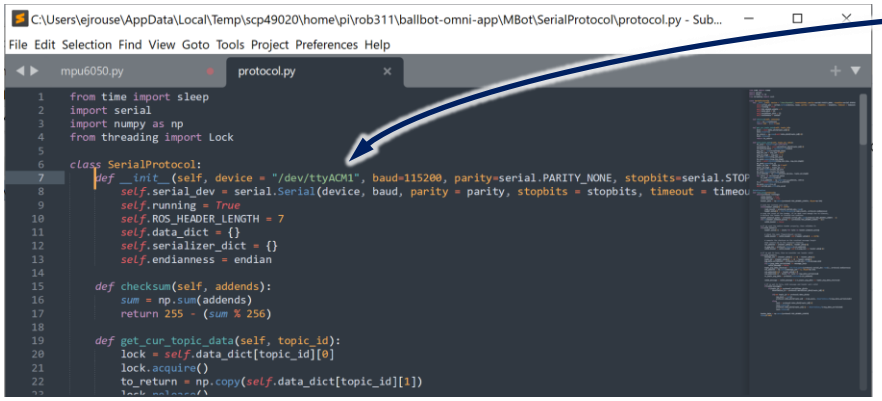
Troubleshooting

- 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 ROB311_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/ROB311_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: '/de
v/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`



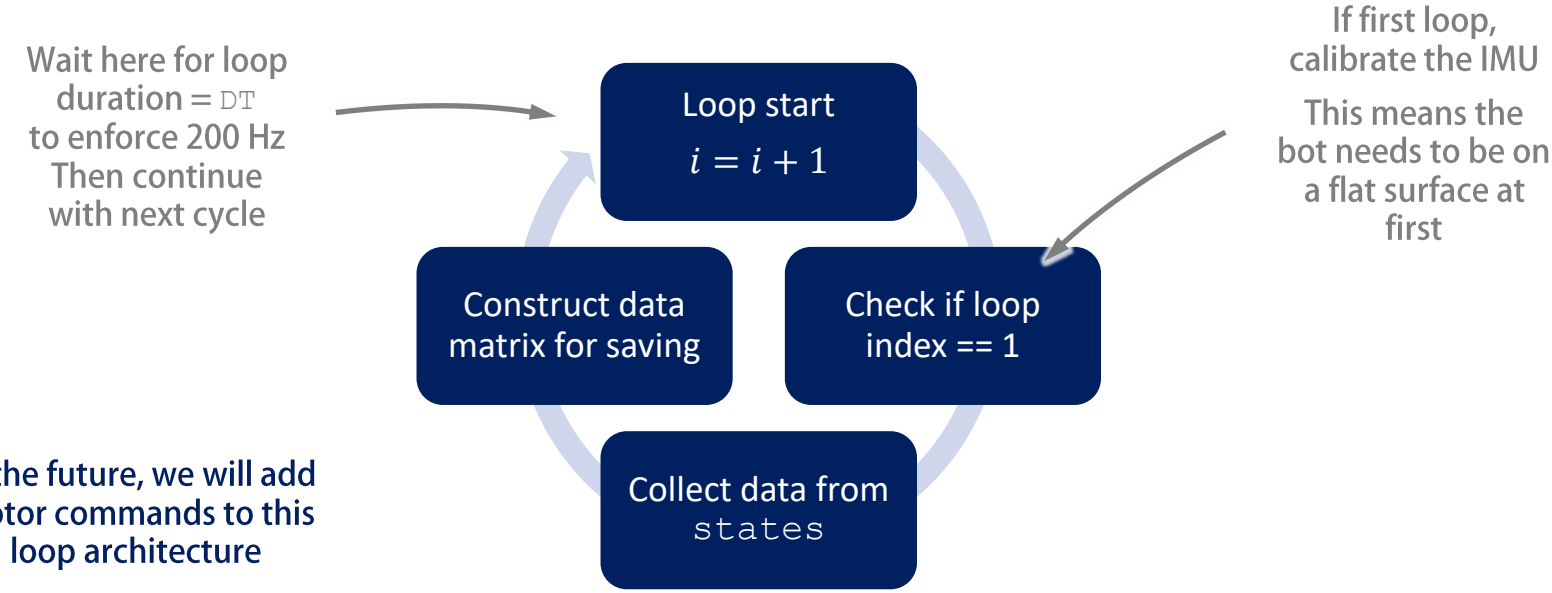
This should be changed to /dev/ttyACM1

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)



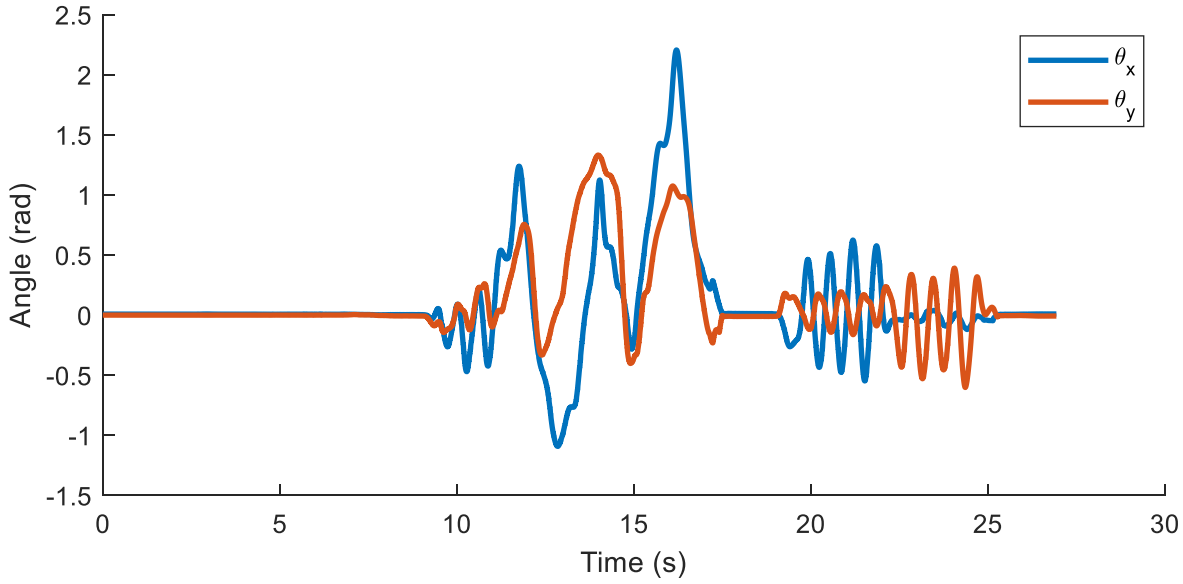
In the future, we will add motor commands to this loop architecture

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`

Reading Data From Your RPi

- When you're finished, moved 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

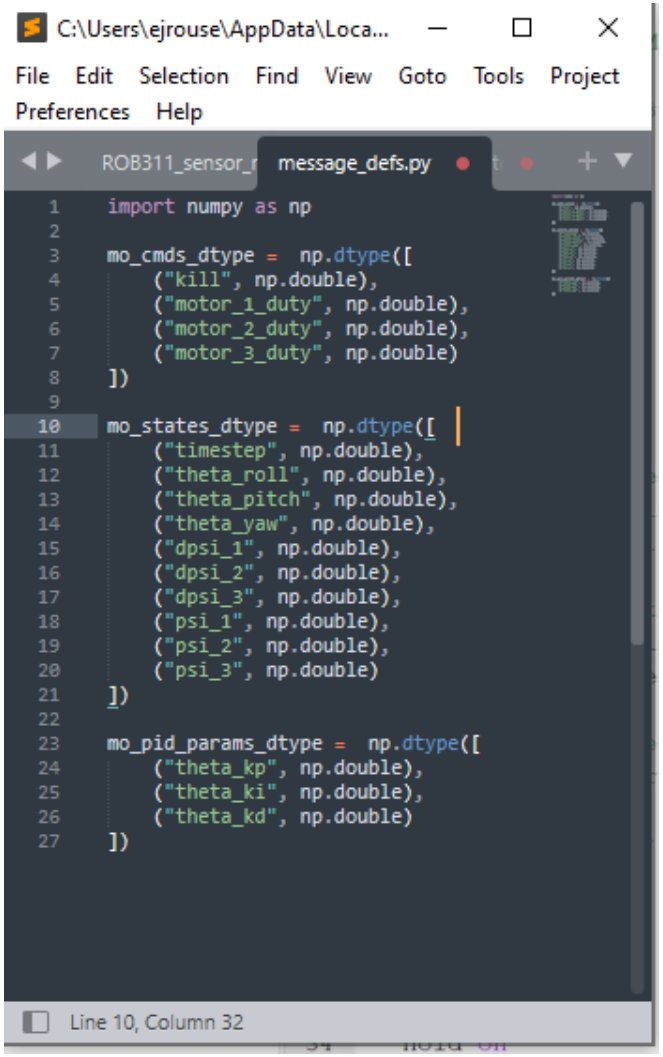
Editor - C:\Users\ejrouse\Desktop\ROUSE\Michigan\ROB 311\Labs\Lab 8\ROB311_ball_bot_data_analysis.m

ROB311_torque_conversion_student.m | ROB311_ball_bot_data_analysis.m

```
% 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 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

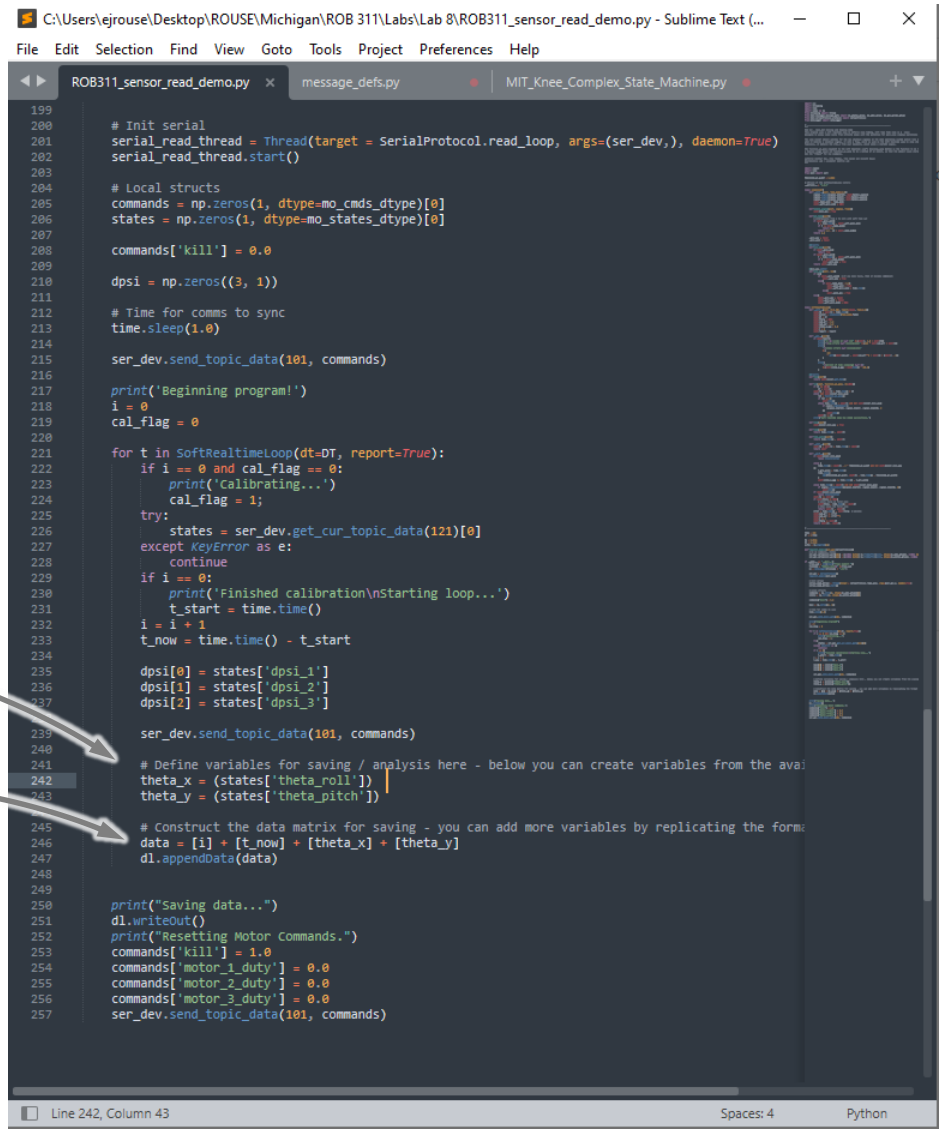
- You have access to many variables collected and sent by the Pico
- These variables are defined in `message_defs.py`
- You can access these variables using the same command we used to get x and y axis rotations
- Edit your python script to also save wheel rotations (`psi`)
- Add them to the data matrix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



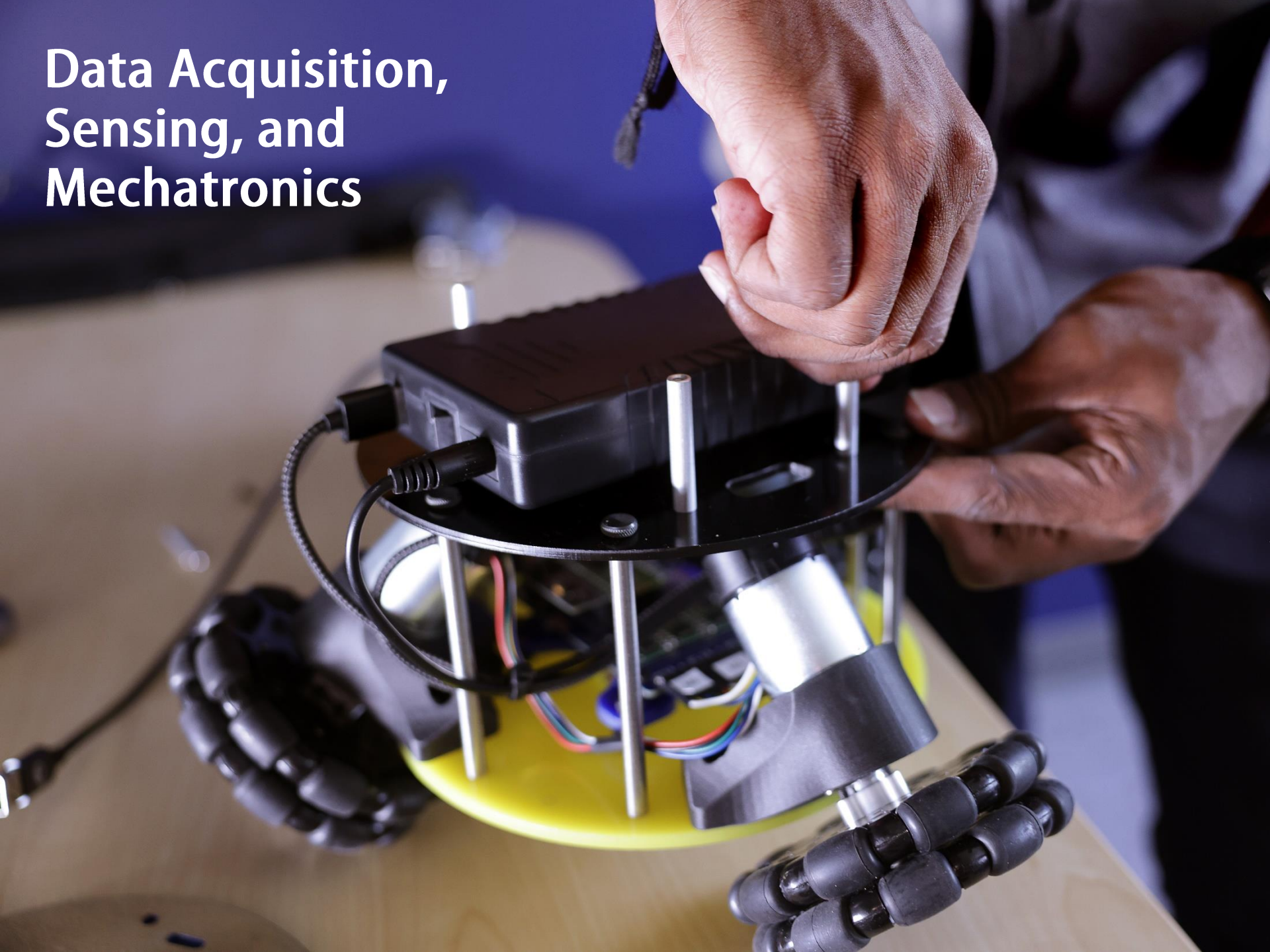
Reading Data From Your RPi

Add more variables here

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



Data Acquisition, Sensing, and Mechatronics

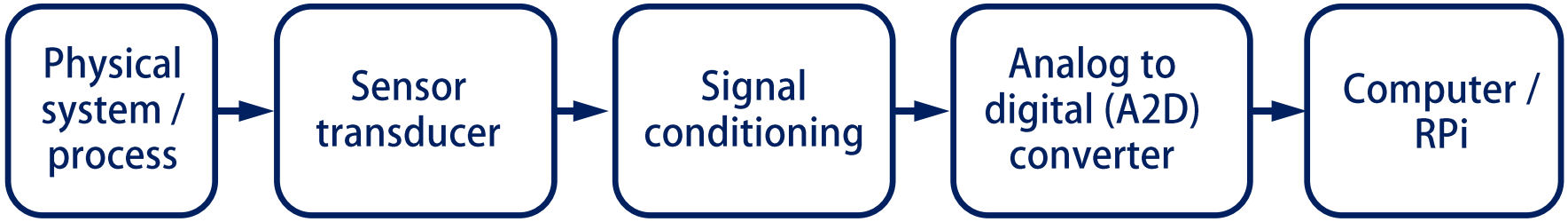


Data Acquisition

- 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

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

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

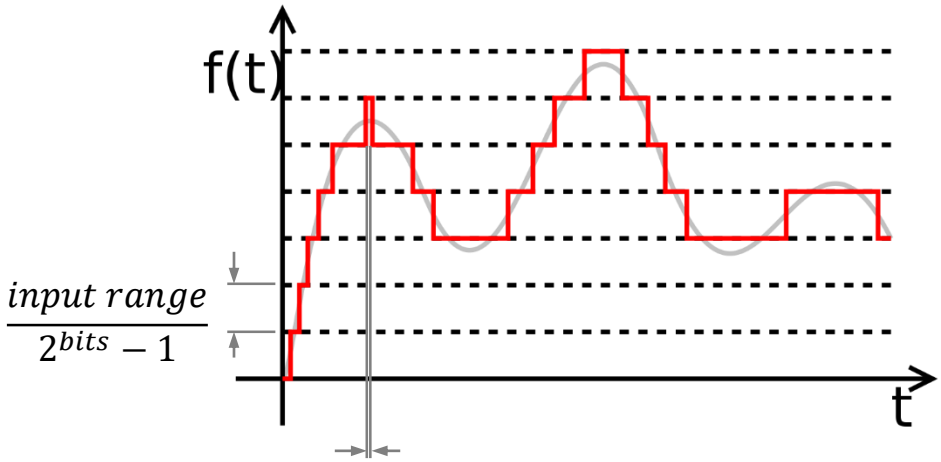
- Example – your 16 bit DAQ system records a value of 11564, with an input range of -10 to 10 V.
- What is the voltage?

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

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:



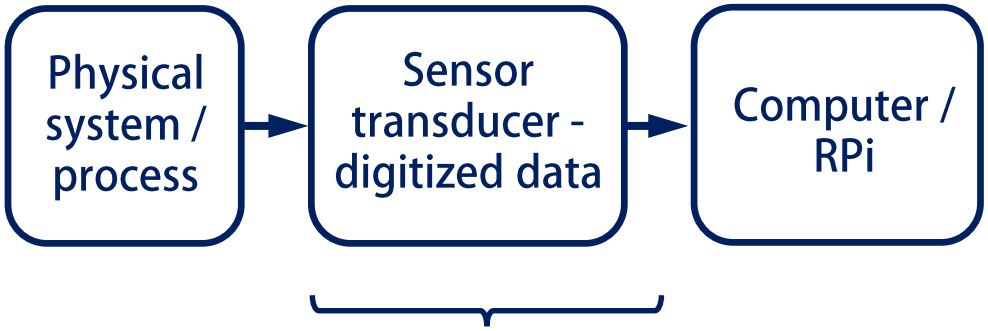
Maximum measurable frequency → $f_{max} = \frac{1}{2} F_s$

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

- 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^2C, 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)

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

- Digital communication busses
  - USB
  - Inter-Integrated Circuit communication (I^2C)
  - Serial Peripheral Interface (SPI)
  - RS-232 (old school, serial port)

} Most common (for now)

- 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

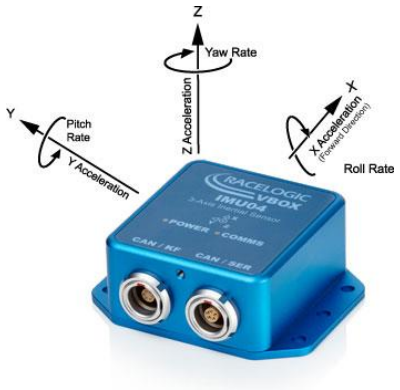
Sensors

- 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

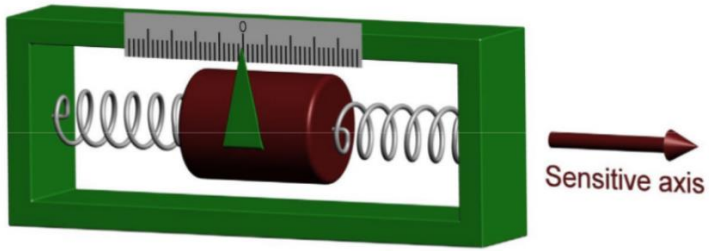
- While there are differences in IMU accuracy and precision, their functions are all relatively similar
- They report x, y, and z acceleration and x, y, z angular velocities
- This information can be fused to obtain global changes to position and orientation—proprietary algorithm within the IMU
  - We 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

Inside an accelerometer—
modern versions are called MEMS
(micro electromechanical 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	Accelerometer Bias Error [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

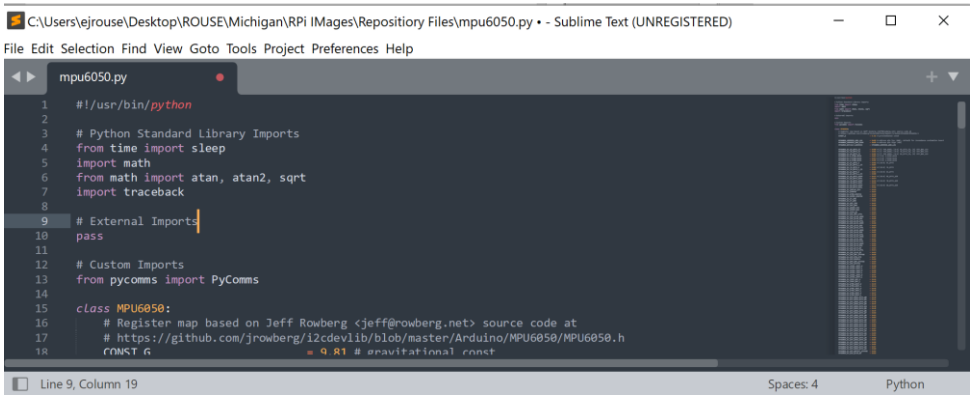
Mounting misalignment error and its effect on integrated position measurement error

Grade	Gyro Angle Random Walk (ARW) [deg/√hr]		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



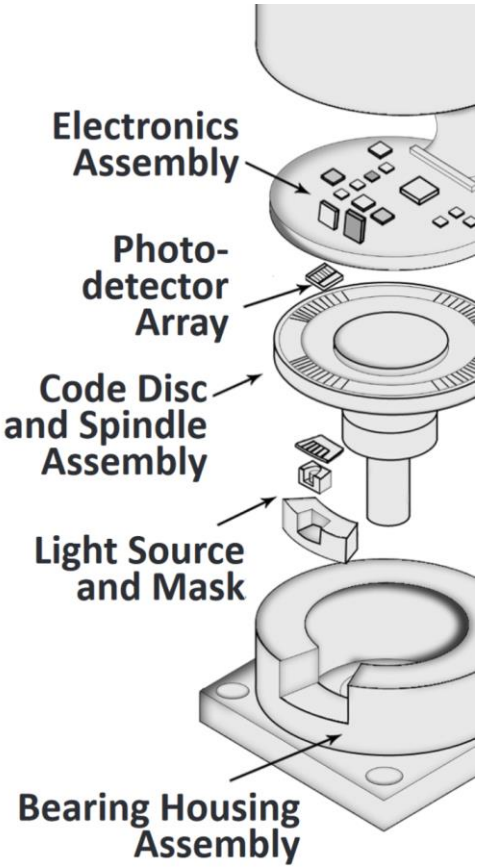
Encoders

- 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

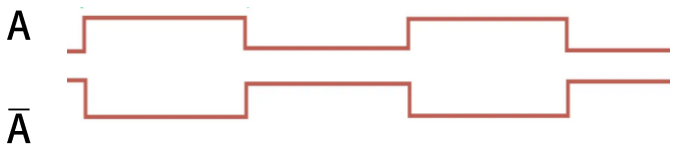
We will discuss these

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 $\overline{A}$
- Pros: High resolution, resistant to magnetic interference, low cost
- Cons: Complex / fragile, larger size

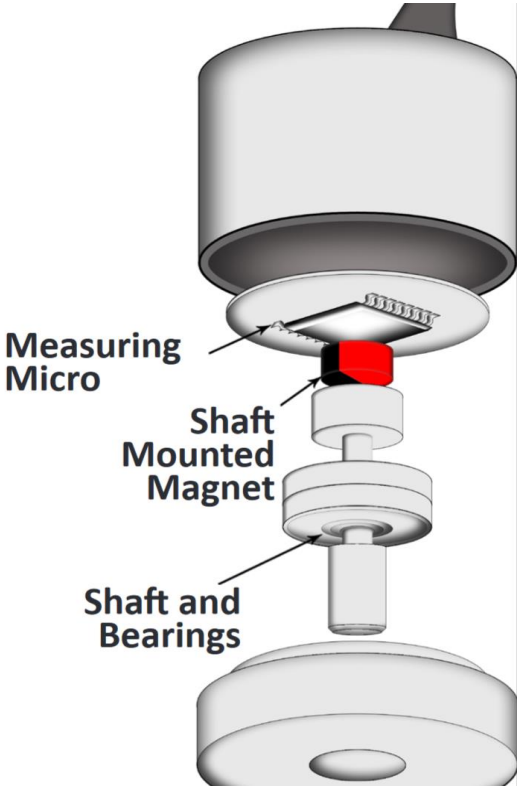


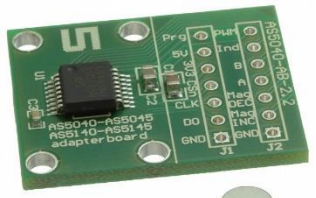
Example encoder signals—high voltage is usually 5V or 3.3V



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

Rolling
Stationary