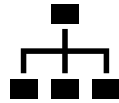


# SCUTTLE Software Guide

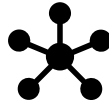
revised 2021.10.25



**Software Architecture**



**Software best practices**



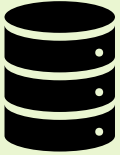
**Sensor Communication**



**Obstacle Avoidance**

# Software Architecture

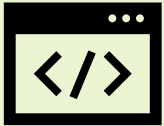
## CONTENTS



This guide covers

- The parts of each software file
- How the programs interact with each other
- How the programs interact with hardware
- Sensor software vs actuator software

## LANGUAGE



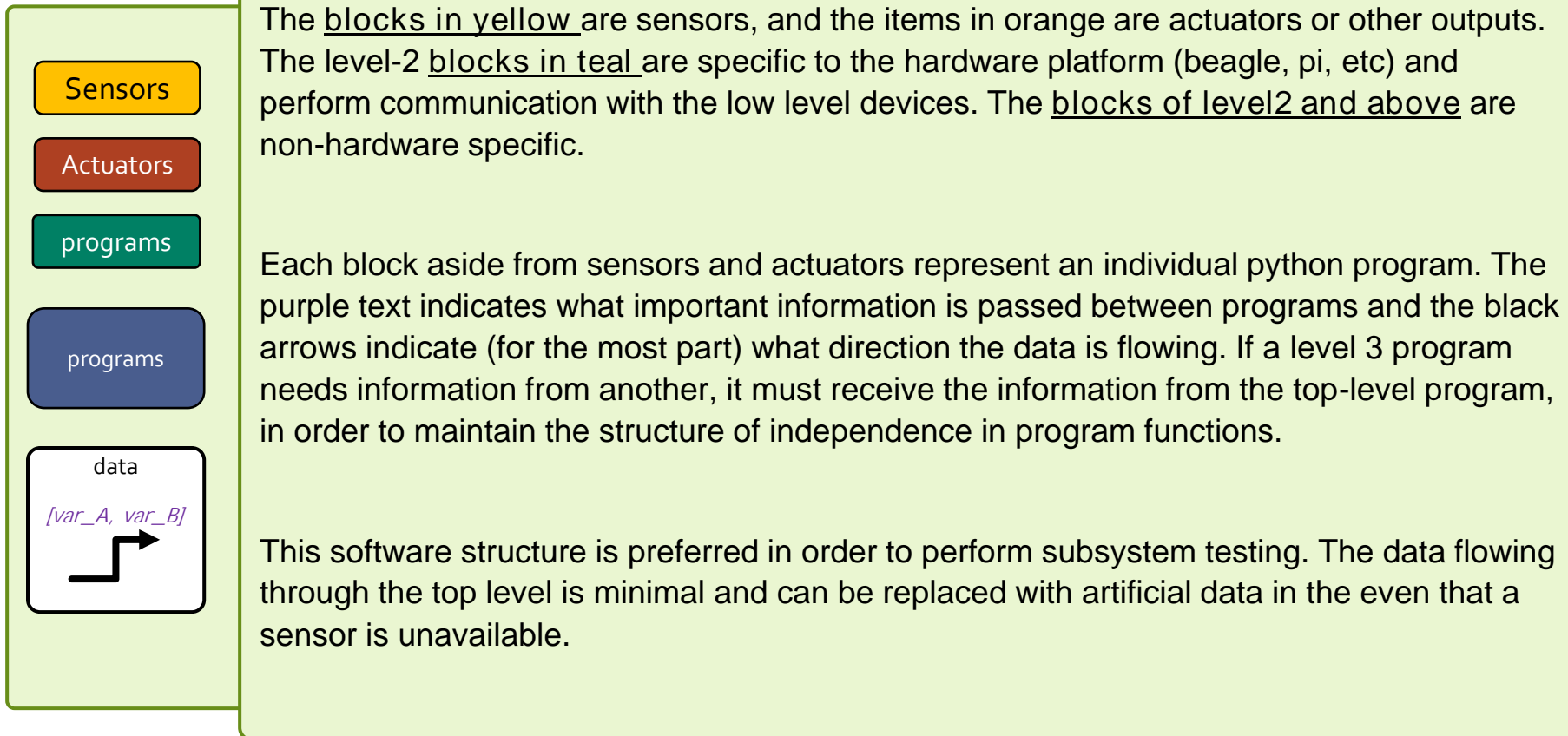
The SCUTTLE robot software has been programmed in Python3 on an embedded Linux platform. Both Beaglebone Blue and Raspberry Pi have been tested successfully. The software has been architected to make a robust starting point for students to create their own autonomous missions. [These slides](#) detail the software architecture.

## FUTURE OUTLOOK

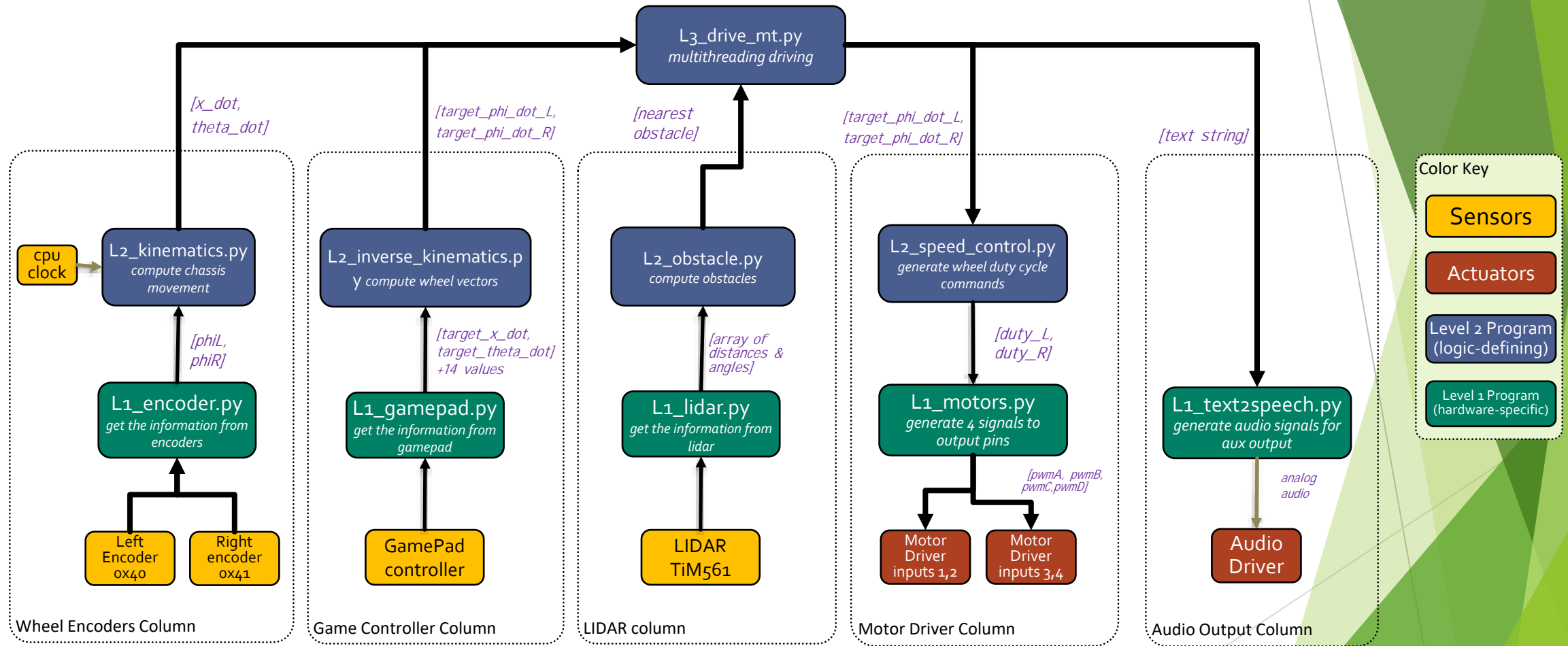


Next Steps: Robotic Operating System 2 (ROS2) is quickly becoming a reliable, versatile software platform for mobile robots. During 2021 the SCUTTLE team will aim to create a new ROS2 version of the software.

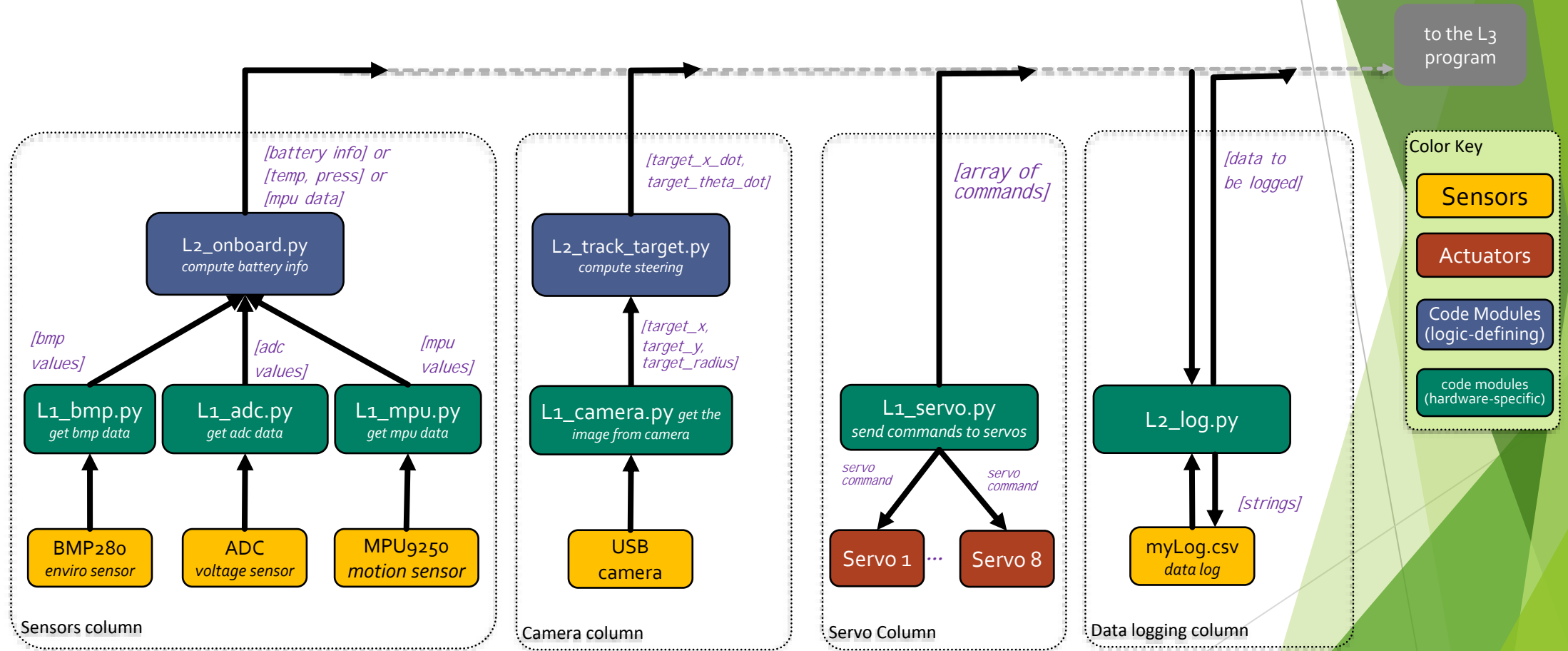
# Software Architecture – Introduction



# Software Architecture - Overview



# Software Architecture – Overview (continued)



# Libraries in use:

## Python importing guidelines:

1. Each file should import the files below it in hierarchy, and not the files above it.
2. Each file may import non-scuttle libraries as needed (import NumPy, import time, etc.)
3. If the Level-1 file has imported an external library, it does not need to be imported by the Level-2 file

## Libraries Utilized:

### BeagleBone Blue Integration:

- [RCPY](#) for communicating with MPU9250 & commanding motor drivers
- [Adafruit GPIO](#) for I2C Communication
- [BMP280](#) for communicating with the onboard bmp280 sensor.

### Raspberry Pi integration:

- [pysicktim](#) for accessing LIDAR data
- [gpiozero](#) for controlling GPIO pins.

### Common Libraries

- [os](#) for making shell commands via python code.
- [time](#) for keeping track of time
- [threading](#) for performing multithreading
- [NumPy](#) for performing math operations
- [Fastlogging](#) for generating log files
- [pygame](#) for accessing gamepad controller data
- [cayenne.client](#) for sending MQTT messages
- [smbus2](#) for accessing i2c bus through python commands

# Libraries Matrix



Lib	Beaglebone Blue	Raspberry Pi 3B+ and 4	Jetson Nano [Under development]
Time	✓	✓	✓
Threading	✓	✓	✓
numpy	✓	✓	✓
pygame	✓	✓	
fastlogging	✓	✓	✓
Cayenne.client	✓	✓	
PySICKtim		✓	
GPIOWZERO		✓	
RCPY	✓		
ADAFRUIT GPIO	✓		
BMP280	✓		

# Outline of an L1 Program

All files follow this outline when possible. The level-1 programs are most suited to this outline.

Explanation of the purpose

Import internal programs (if applicable)

Import external programs (aka libraries). Take actions for initializations of objects or global variables.

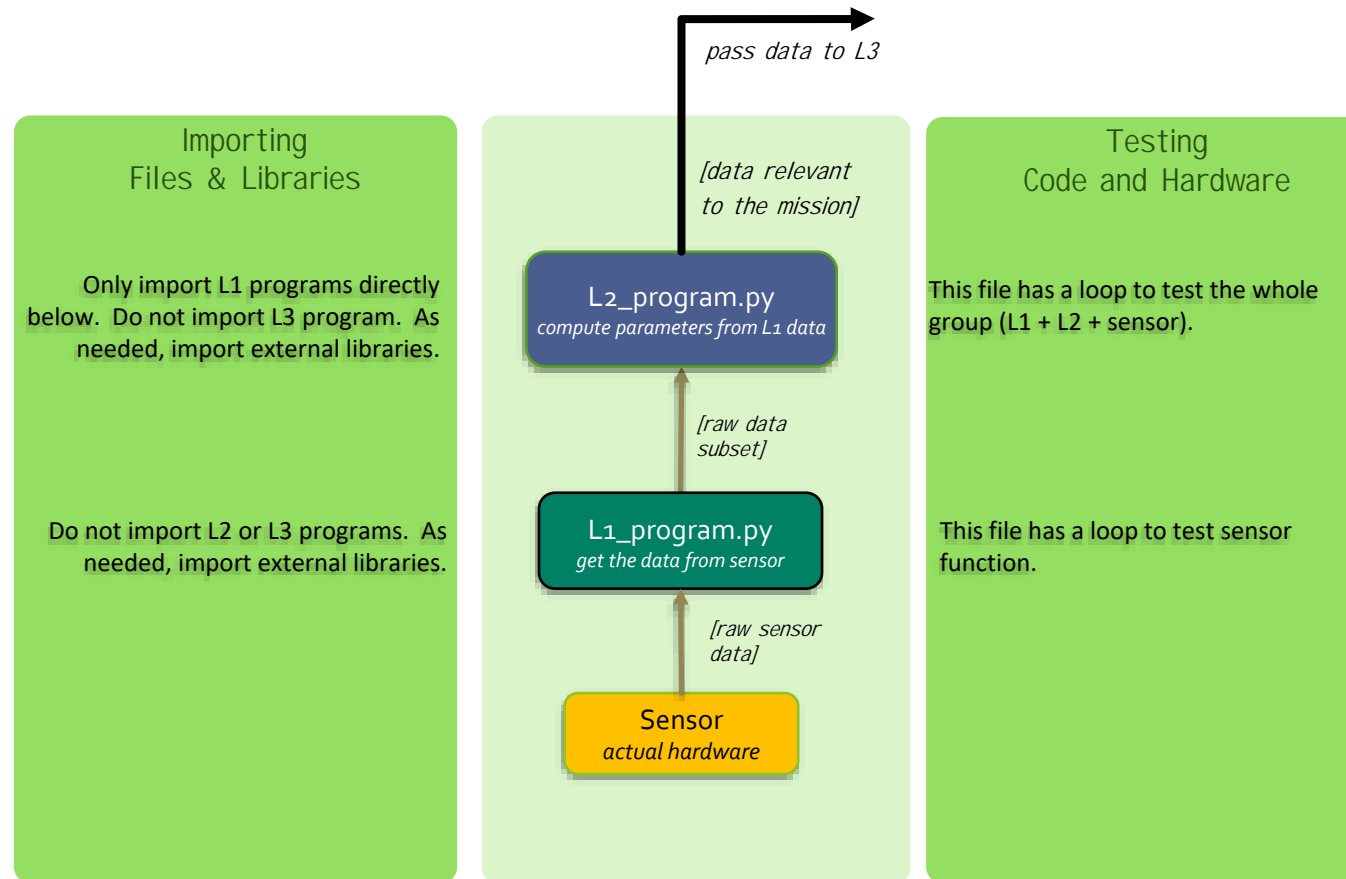
Define functions. In some cases, make functions that combine other functions in sequence.

Offer a simplified, minimal loop for testing the code.

```
1 # This example drives the right and left motors.
2 # Intended for Beagle hardware
3
4 import rcpy
5 import rcpy.motor as motor
6 import time # only necessary if running this program as a loop
7
8 motor_r = 2    # Right Motor
9 motor_l = 1    # Left Motor
10 rcpy.set_state(rcpy.RUNNING)
11
12 #channel refers to left(0) or right(1)
13 def MotorL(speed):
14     motor.set(motor_l, speed)
15
16 def MotorR(speed):
17     motor.set(motor_r, speed)
18
19 # Uncomment this section to run this program as a standalone loop
20 # while rcpy.get_state() != rcpy.EXITING:
21 #
22 #     if rcpy.get_state() == rcpy.RUNNING:
23 #
24 #         MotorL(0.5) # gentle speed for testing program. 0.3 PWM may not spin wheels.
25 #         MotorR(0.5)
26 #         time.sleep(4) # run fwd for 4 seconds
27 #         MotorL(-0.5)
28 #         MotorR(-0.5)
29 #         time.sleep(2) # run reverse for 2 seconds
```

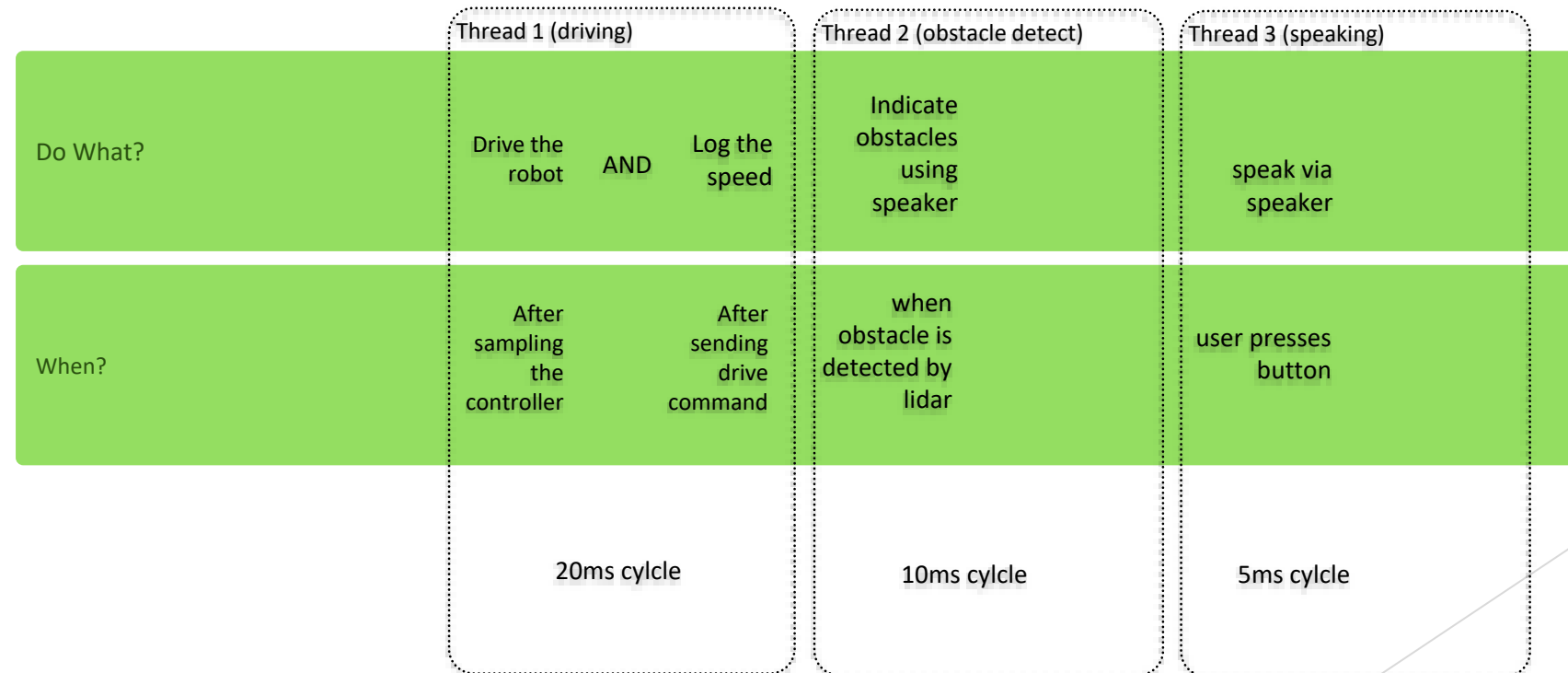


# Guidelines for Levels



# Multi-threading Purpose

- ▶ Threading offers better control over **timing of code execution**.
- ▶ Each thread should contain **actions that are related** and that should be executed within a specific time window.
- ▶ The user should avoid passing data between threads because it reduces robustness. Instead, **call the level 2 program as needed in each thread**, even if you need to communicate with the same device (ie, retrieve gamepad commands for driving and retrieve in parallel for speaking commands)



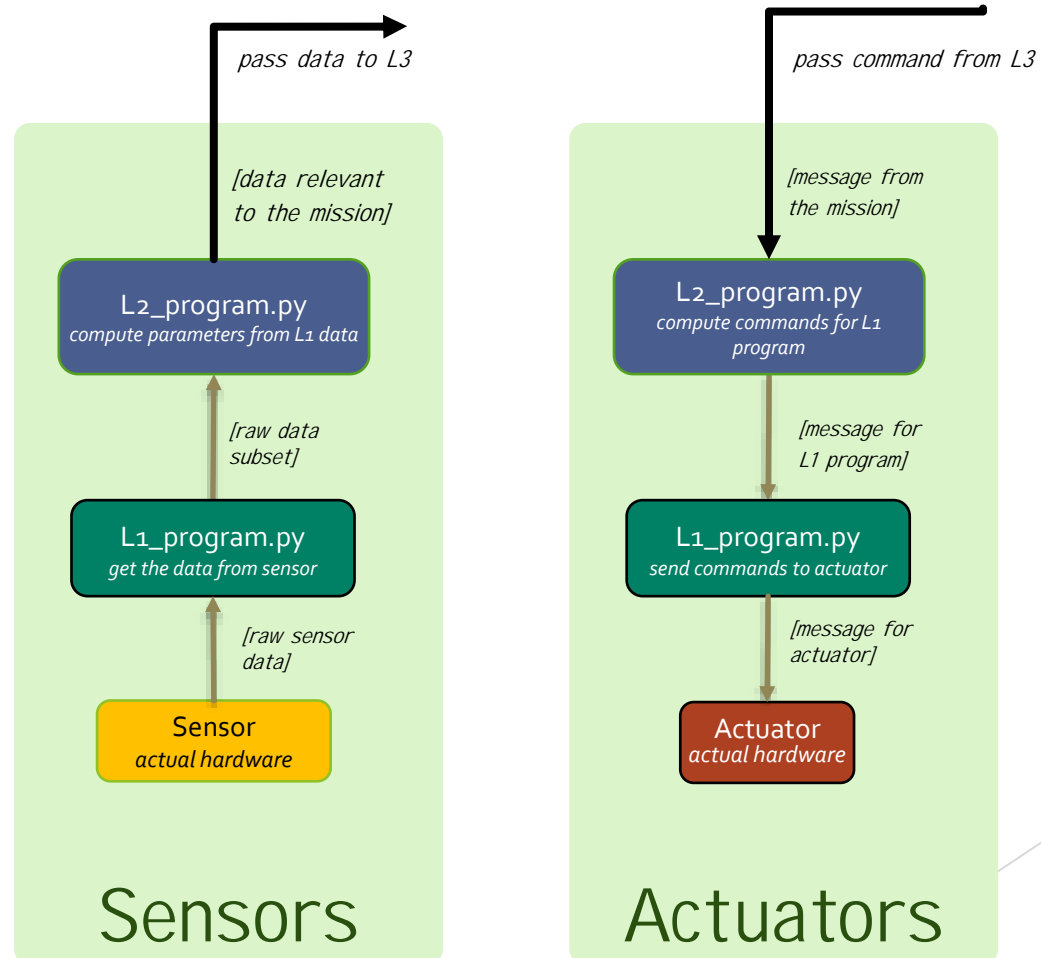
# Software Architecture: Sensors vs Actuators

Sensor and Actuators have the same architecture except for **data direction**.

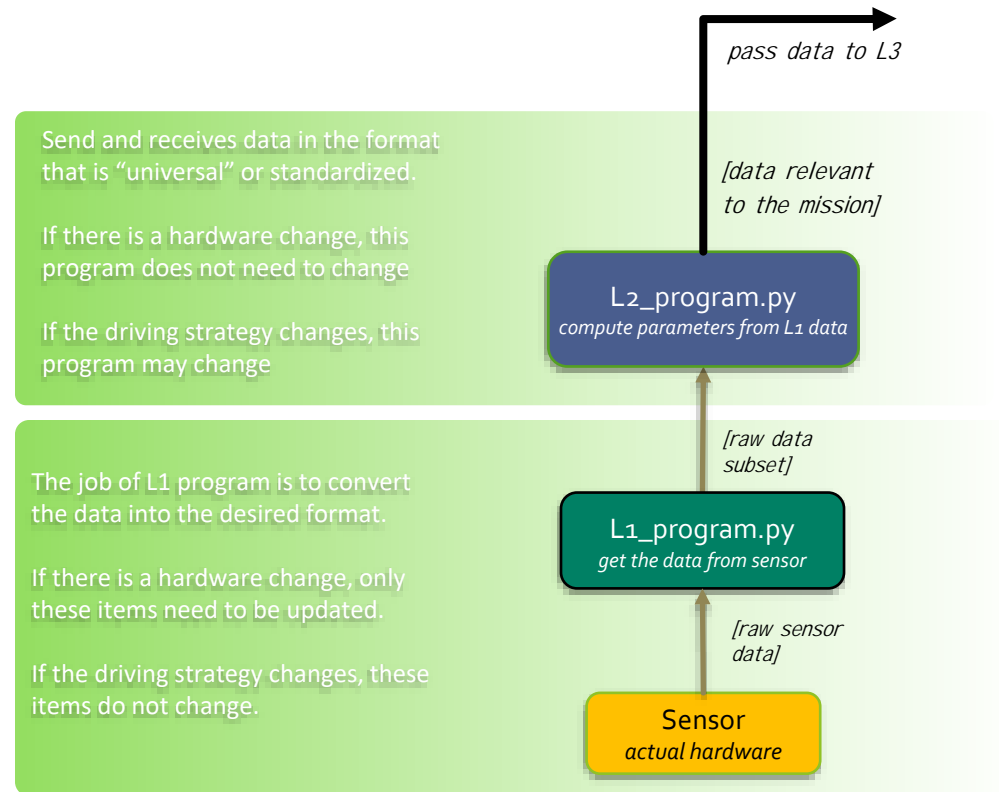
For **sensors**, the data is generated at the hardware and sent UP.

For **actuators**, the data is generated at the top and sent DOWN to hardware.

Some sensors and actuators have feedback and preset commands, so data may flow **both ways**.



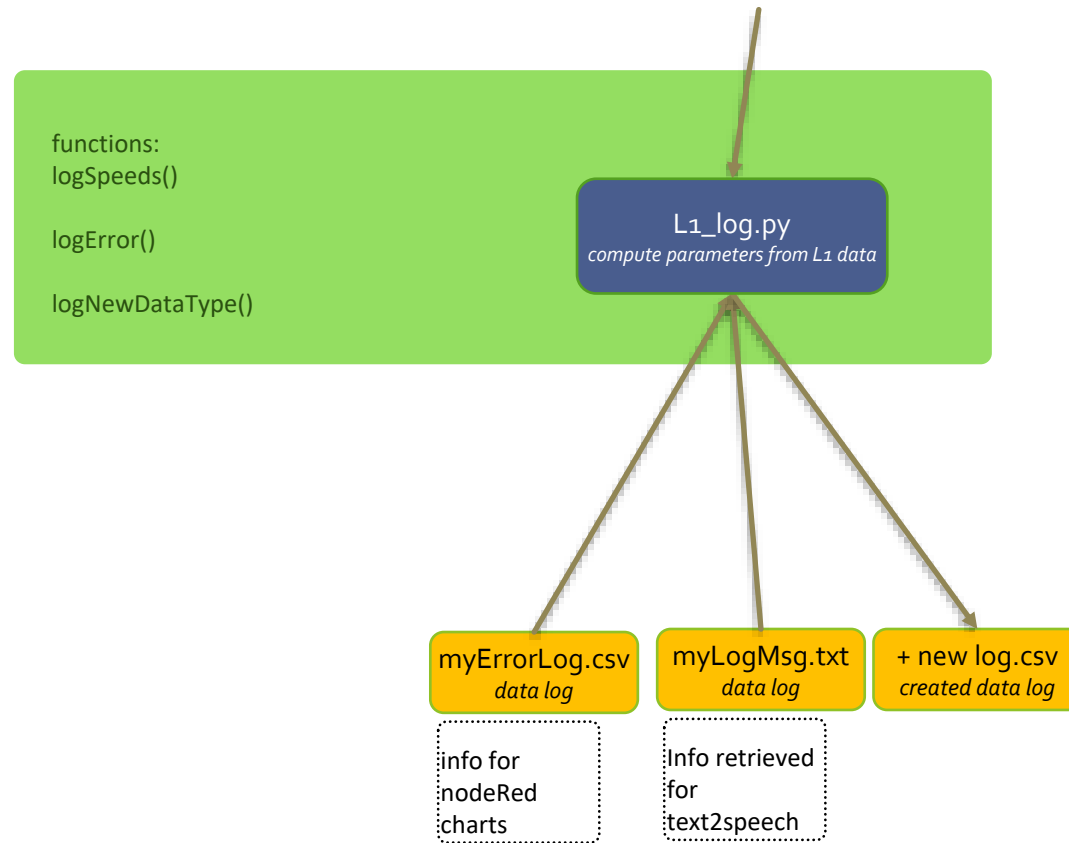
# Software Architecture: Modularity & Robustness



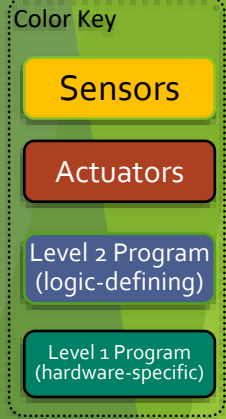
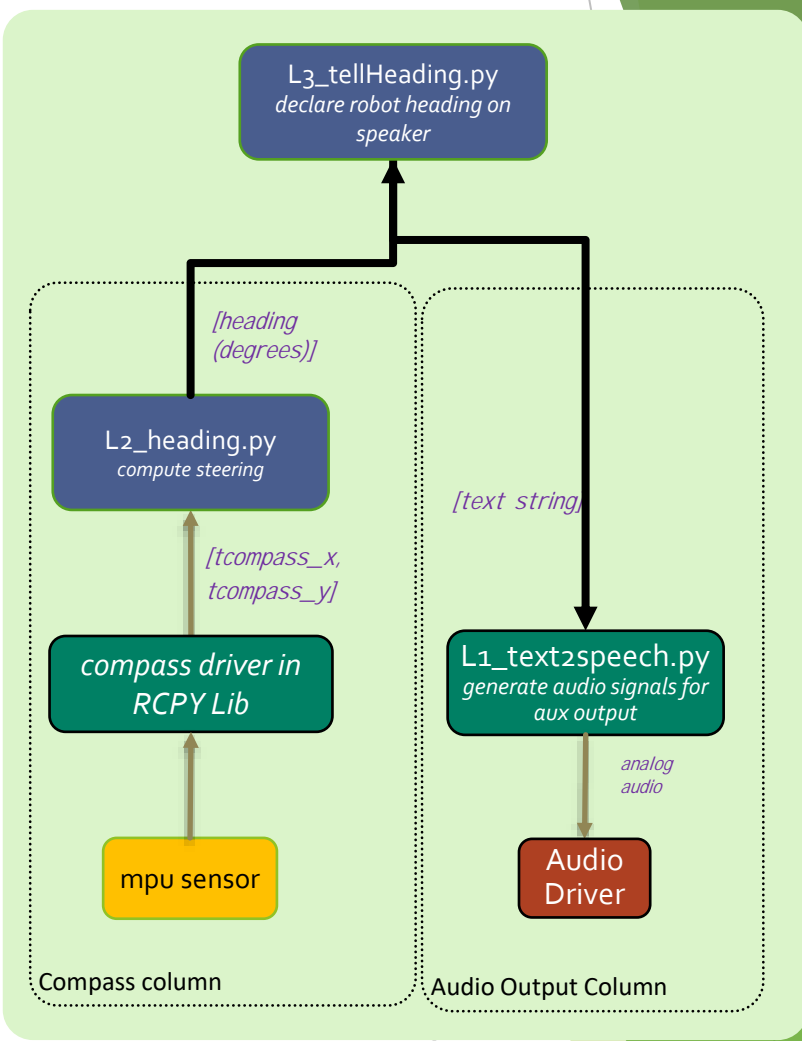
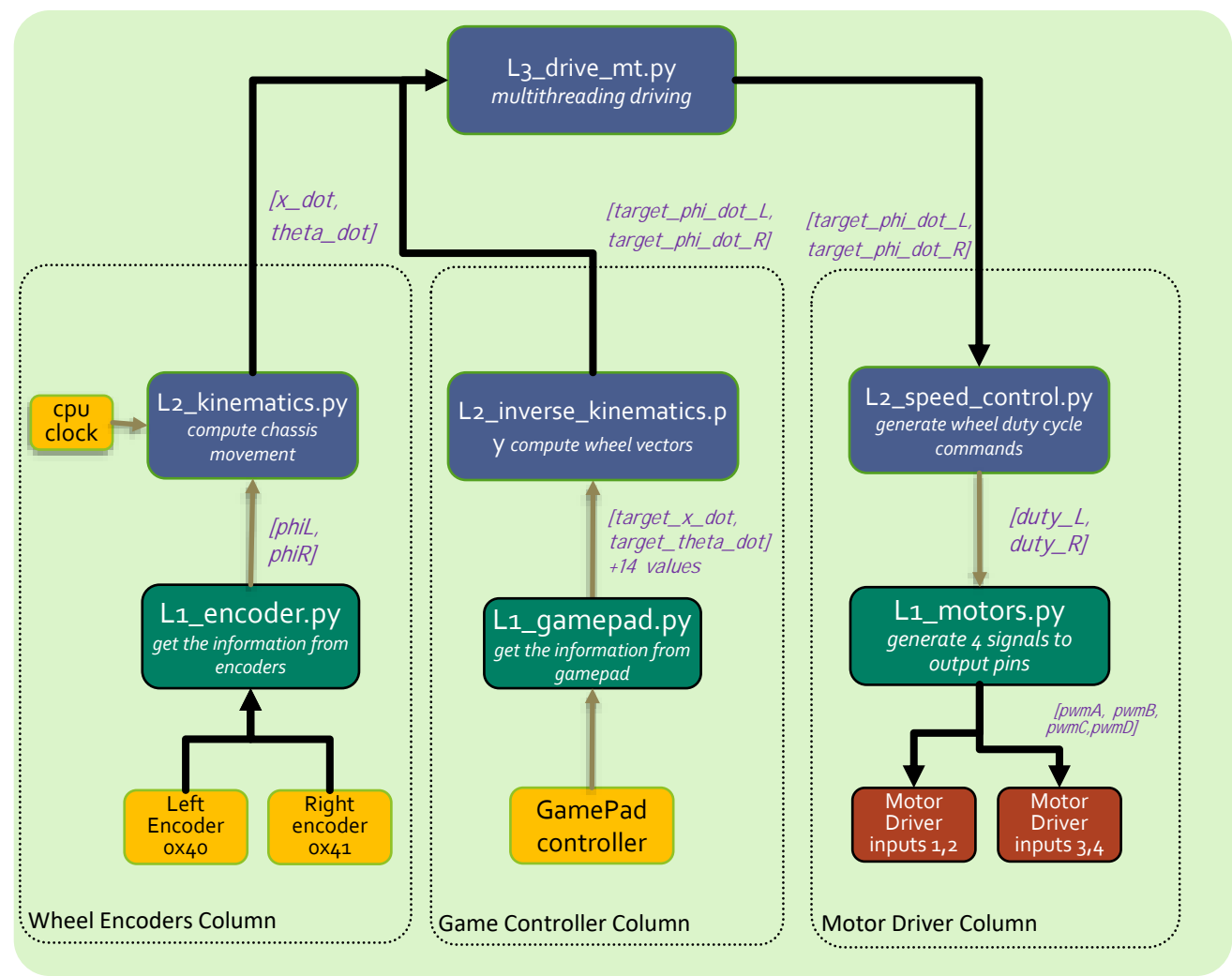
# Level 1: logging

Rather than interacting with hardware, the L1\_log program interacts with other python files. It **acts as a sensor** in that it retrieves recorded data and it **acts as an actuator** in that it can receive data and perform an action with it (store it in a file).

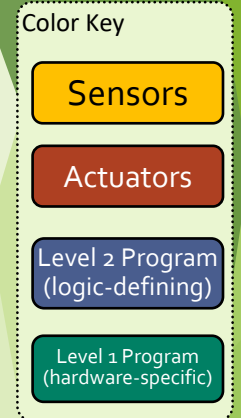
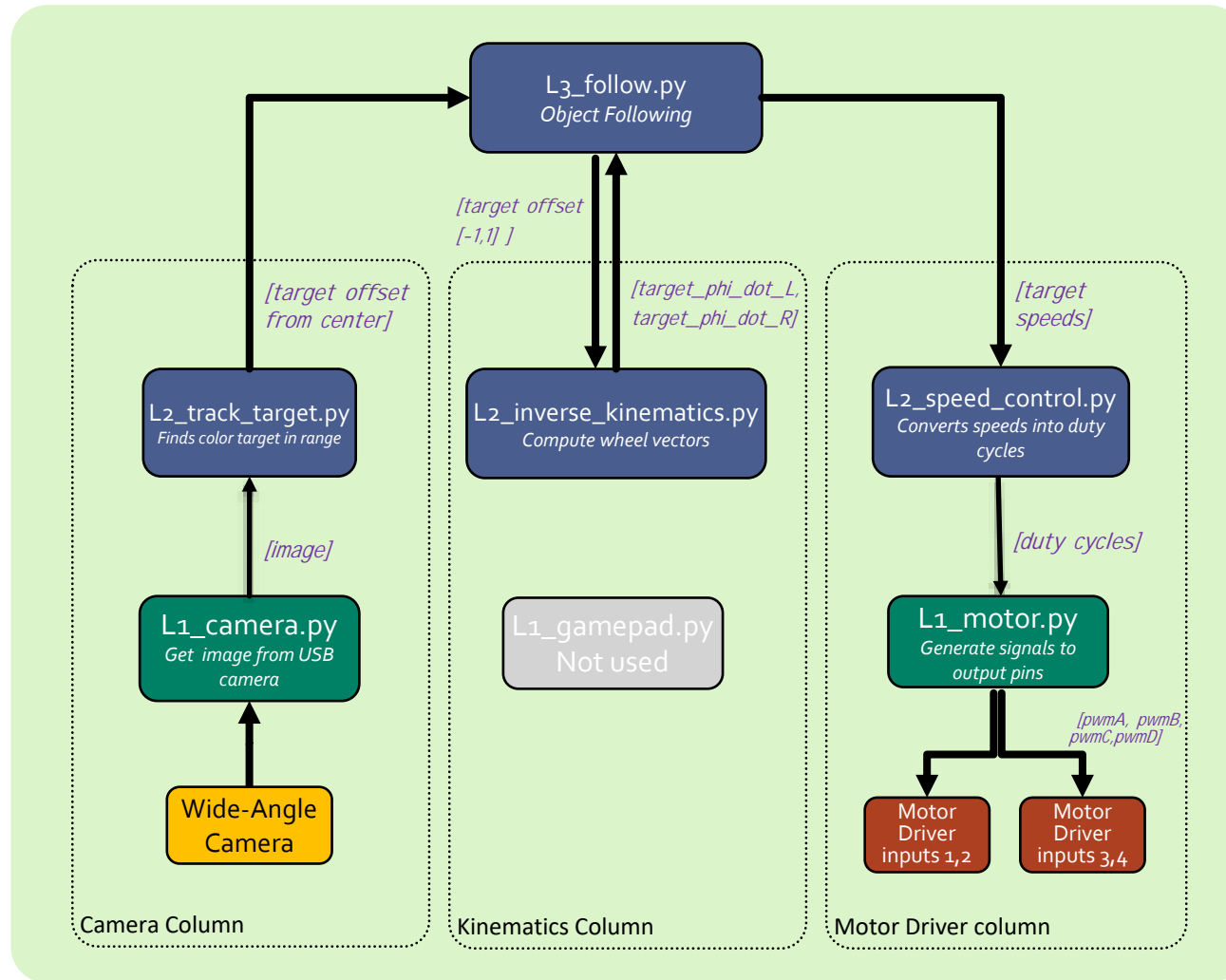
L1\_log.py program was initially designated as level2, but is being set as L1 going forward (2020.11)



# Multithreading example



# Color Tracking Example

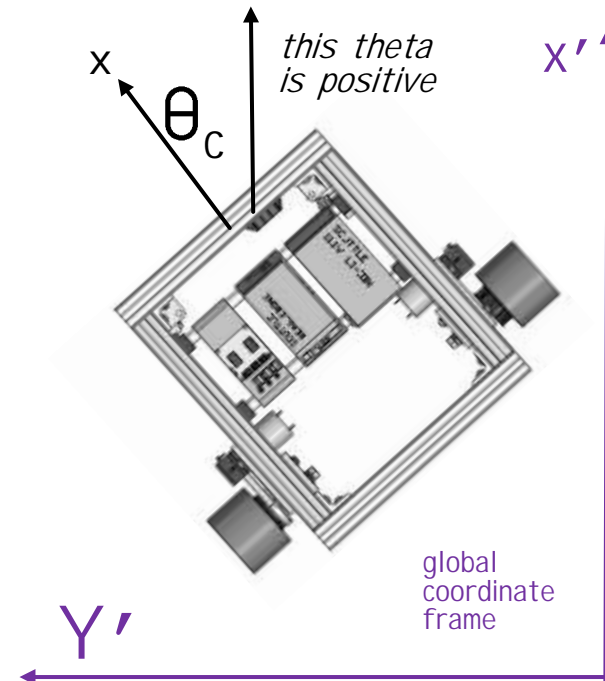


# Absolute Orientation

- ▶ SCUTTLE has a compass for orientation
  - ▶ The compass is nothing but a 3-axis magnetometer
  - ▶ Encoders can provide *relative* orientation
  - ▶ Compass is required for *global* orientation
- ▶ The compass is embedded in the IMU (MPU-9250)
  - ▶ It has 3 sensors oriented in the indicated directions
  - ▶ L1\_mpu.py accesses the magnetometer
  - ▶ Each magnetometer requires calibration



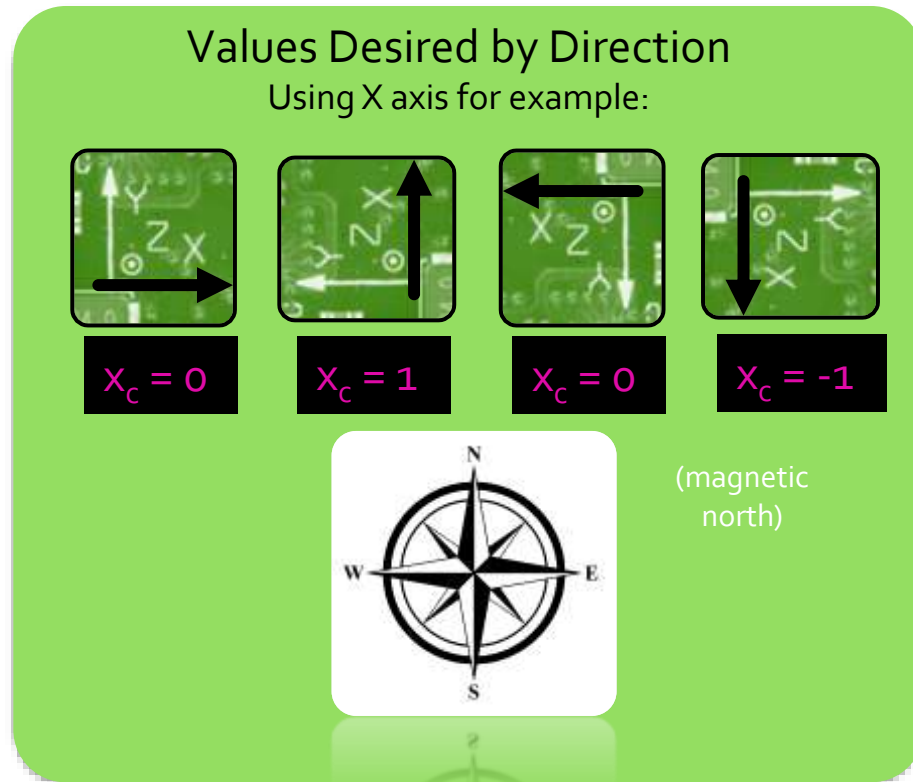
Remember: Theta is defined as scuttle's chassis x-vector minus the global x-vector





# Magnetometer Behavior

- ▶ An axis is at its MAXIMUM when it is **aligned** NORTH
- ▶ The axis is at its MINIMUM when it is **opposing** NORTH
- ▶ After calibration, we can achieve the behavior below



- 1) Discover the maximum and minimum values by rotating sensor in a full circle.

*Permanent magnets influence the sensor, so calibration must be done on the robot, in position near the motors.*

Before Calibration	Min (microtesla)	max (microtesla)
X	-15	38
Y	-22	20

- 2) Using the following equation, re-scale each axis

$$x_{\text{scaled}} = \frac{2(x - x_{\min})}{(x_{\max} - x_{\min})} - (1)$$

After Calibration	Min (ratio to max)	max (ratio to max)
X	-1	1
Y	-1	1

# Determining Absolute Orientation

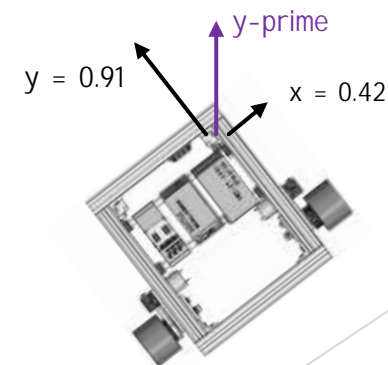
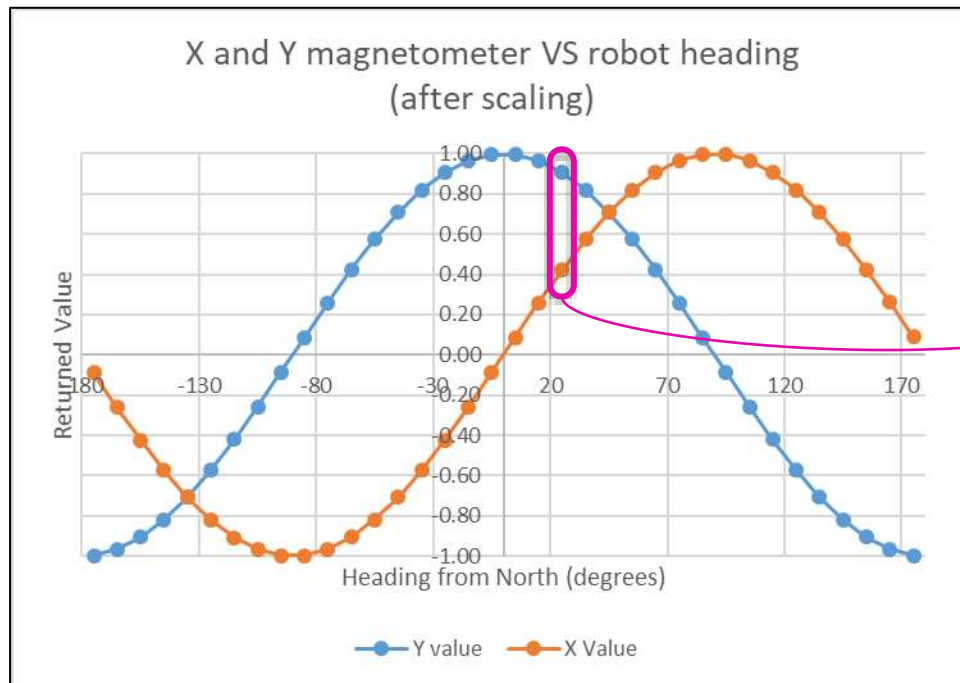
- ▶ X and Y axes are sufficient information to give heading.
  - ▶ Z axis returns zero if scuttle sits flat
- ▶ Theta is defined as rotation of SCUTTLE from the global coordinate frame, or y-prime
  - ▶ positive theta means SCUTTLE is turned left from north
  - ▶ We can define NORTH as the y-axis of the global coordinate frame

Theta is positive when scuttle points west  
Theta is negative when scuttle points east

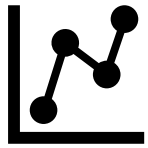
Use  $\arctan2(y, x)$  to return a heading  
 *$\arctan2$  is the "element-wise arc tangent of  $y/x$  choosing the quadrant correctly."*

Example:

$\text{ATAN2}(0.91, 0.42)$  returns 25 degrees



y is pointed strongly north  
X is pointed weakly north  
both axes return positive values



# Speeds Tuning

These are general performance characteristics you can expect when using the standard SCUTTLE hardware:



## Nominal conditions:

Battery: 11.5 volts OC



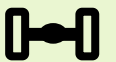
Motors: equipped with standard 200 rpm gearbox



Wheels: 83mm diameter urethane wheels



Pulleys: motor = 15 teeth, wheel = 30 teeth



Wheelbase: 405mm

- ▶  $V_{\max} = 0.4\text{m/s}$  (measured by wheel speed)

$$v = \omega * r$$

- ▶  $\omega_{\max, \text{motor pulley}} = 19.5 \text{ rad/s}$

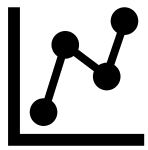
- ▶  $\omega_{\max, \text{wheel}} = 9.75 \text{ rad/s}$

- ▶ With 1 wheel stopped and 1 wheel moving:

$$\dot{\theta} = \frac{v}{L}$$

(where  $L$  = wheelbase)

- ▶  $\dot{\theta}_{\max, \text{chassis}} = 1.98 \text{ rad/s}$  (0.32 turns/sec)



# Speeds Tuning

These are general performance characteristics you can expect when using the standard SCUTTLE hardware:



## Nominal conditions:

Battery: 11.5 volts OC



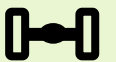
Motors: equipped with standard 200 rpm gearbox



Wheels: 83mm diameter urethane wheels



Pulleys: motor = 15 teeth, wheel = 30 teeth



Wheelbase: 405mm

- ▶  $V_{\max} = 0.4\text{m/s}$  (measured by wheel speed)

$$v = \omega * r$$

- ▶  $\omega_{\max, \text{motor pulley}} = 19.5 \text{ rad/s}$

- ▶  $\omega_{\max, \text{wheel}} = 9.75 \text{ rad/s}$

- ▶ With 1 wheel stopped and 1 wheel moving:

$$\dot{\theta} = \frac{v}{L}$$

(where  $L$  = wheelbase)

- ▶  $\dot{\theta}_{\max, \text{chassis}} = 1.98 \text{ rad/s}$  (0.32 turns/sec)

# SCUTTLE Driving

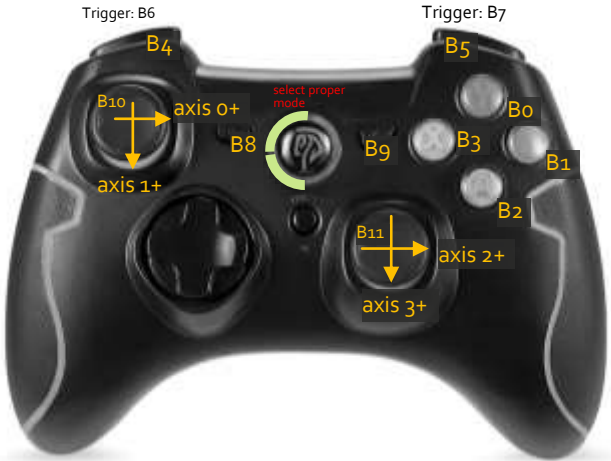
- The left joystick operates the robot wheels
- The forward/backward axis will request a speed
  - (A.K.A movement in  $x$ )
- The left/right axis will request an angular velocity
  - (A.K.A movement in  $\theta$ )



Given a measured max forward velocity of 0.4m/s, the other maximums are calculated.

Move	Theta_dot t (rad/s)	X_dot (m/s)	Phi_dot ot
max	1.99	0.4	9.75

## Gamepad Controls Mapping

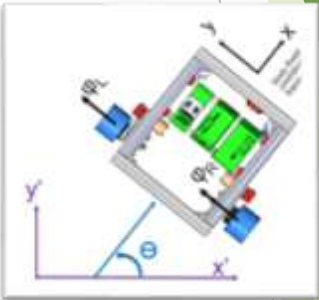
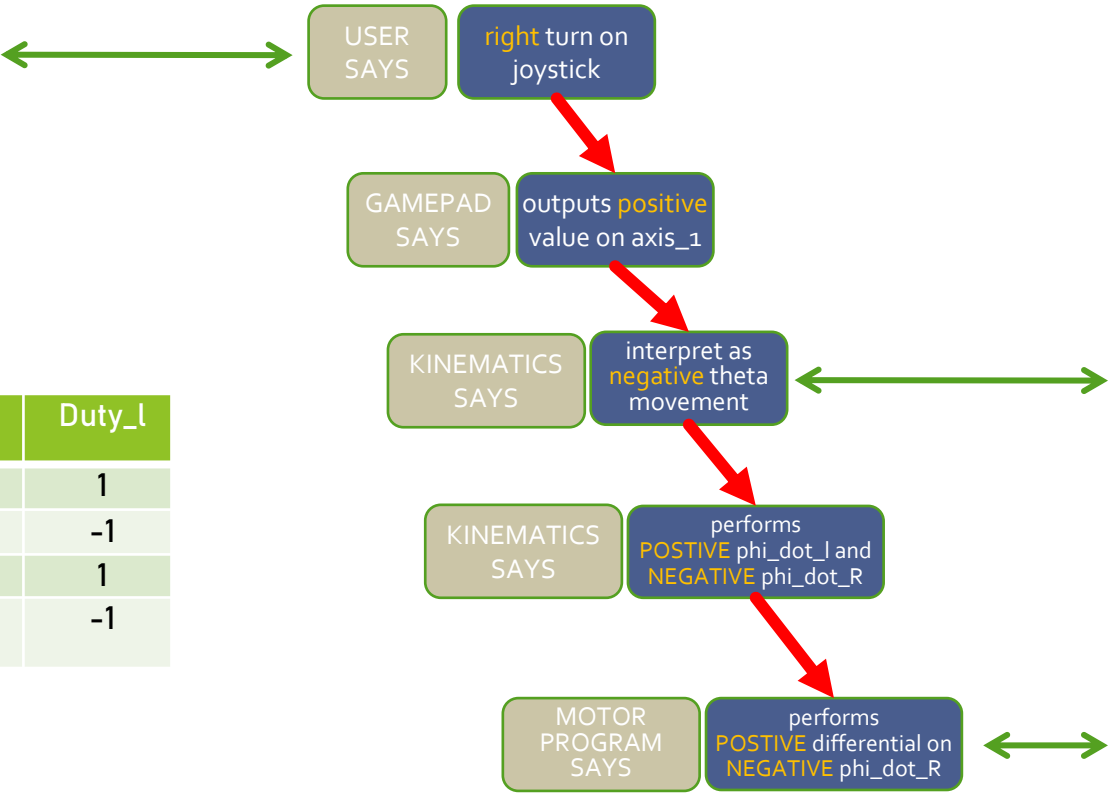


```
axes_status = np.array([axis_0, axis_1, axis_2, axis_3])
button_status = np.array([B0, B1, B2, B3, B4, B5, B6, B7, B8, B9, B10, B11])
```

# SCUTTLE Movement Example

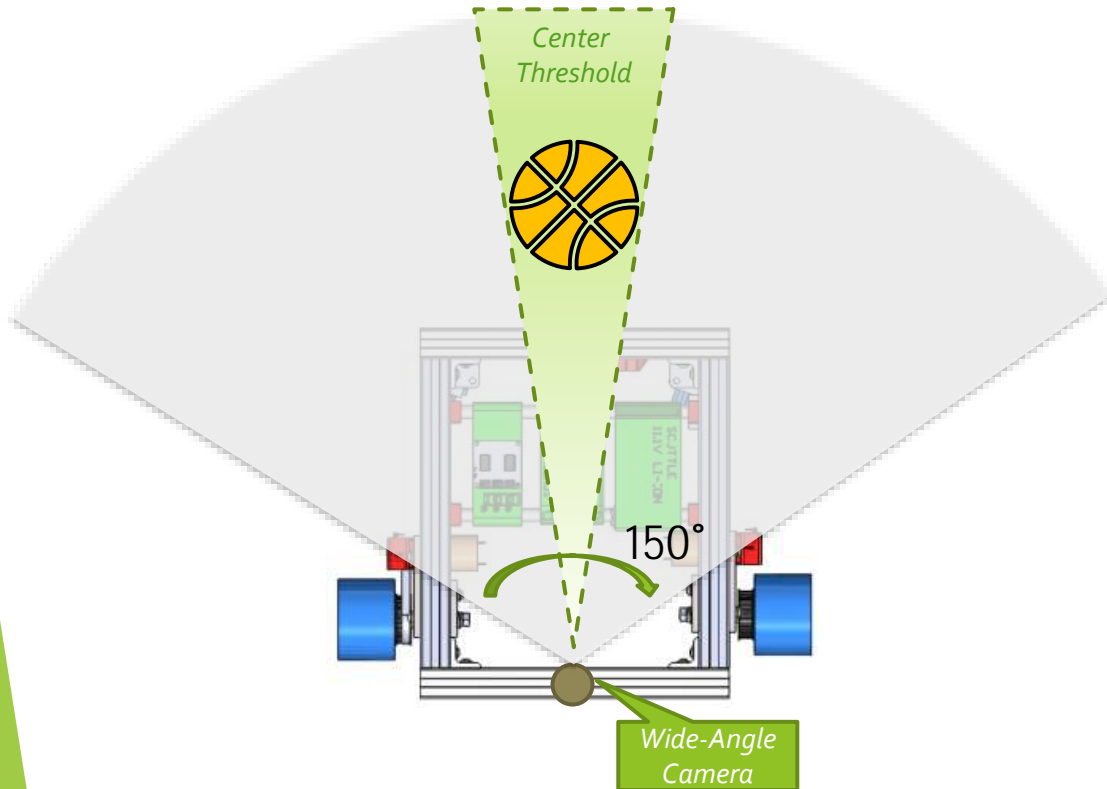


Move	Theta_dot	X_dot	Duty_r	Duty_l
Fwd	0	1	1	1
Rev	0	-1	-1	-1
right	-1	0	-1	1
left	+1	0	1	-1

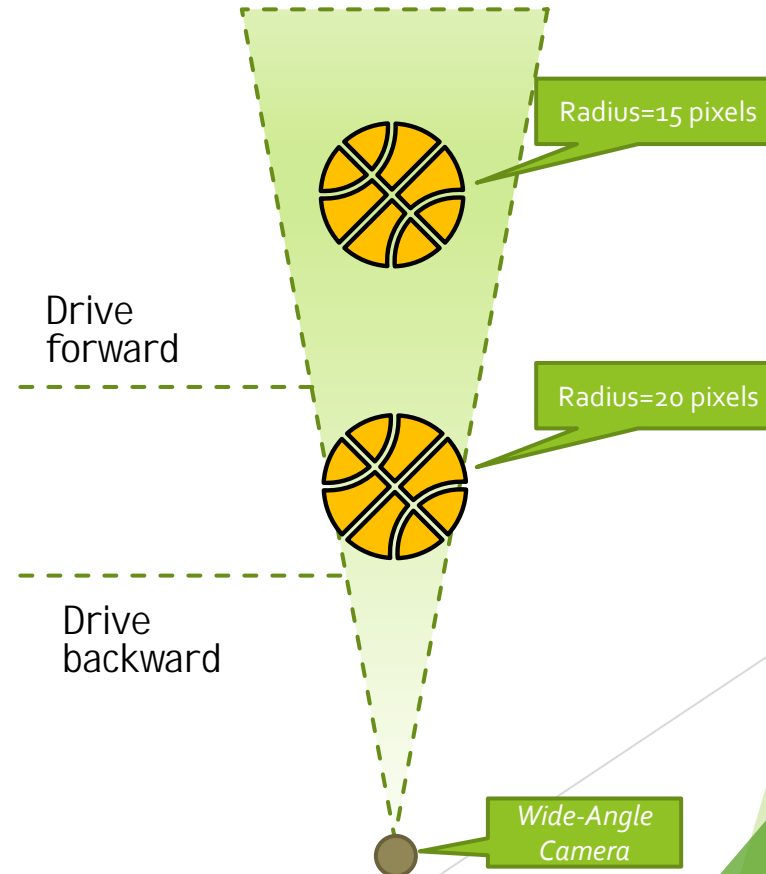


# SCUTTLE Color Tracking

1) SCUTTLE turns until the target is detected within the threshold



2) After the target is in the center range, SCUTTLE drives forward or backwards to reach a target radius, in pixels.



# LIDAR Concept of Operation

## ANATOMY

Lidar systems have a rotating sensor collecting multiple measurements to measure in a 2D plane. (Some have 3D, by other methods).

## METHOD

Lidar emits a beam of light and receives the reflection. distance is based on Time of Flight concept.

## POWER

TiM561 uses about 2.1 watts during operation, mainly due to driving the motor and driving a strong IR emitter diode.

## FAILURE MODES

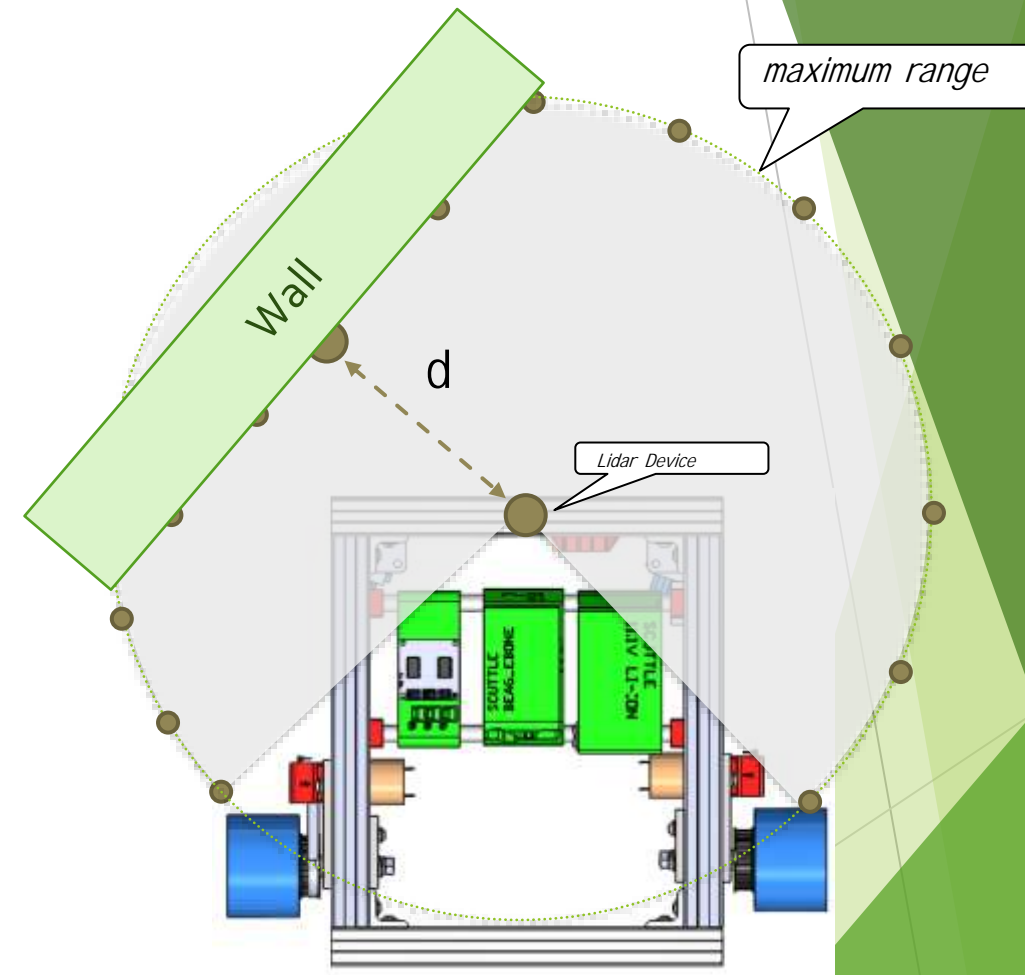
Just like light, a Lidar beam can be absorbed by very dark objects and can be mis-directed by highly reflective objects which are non-perpendicular to the beam.

## DATA QUALITY

The lidar has *variable resolution* in a sense! 0.33 degrees offers 5mm point spacing at a 1m distance, and at 10 meters, 50mm point spacing.

## APPROPRIATE USE

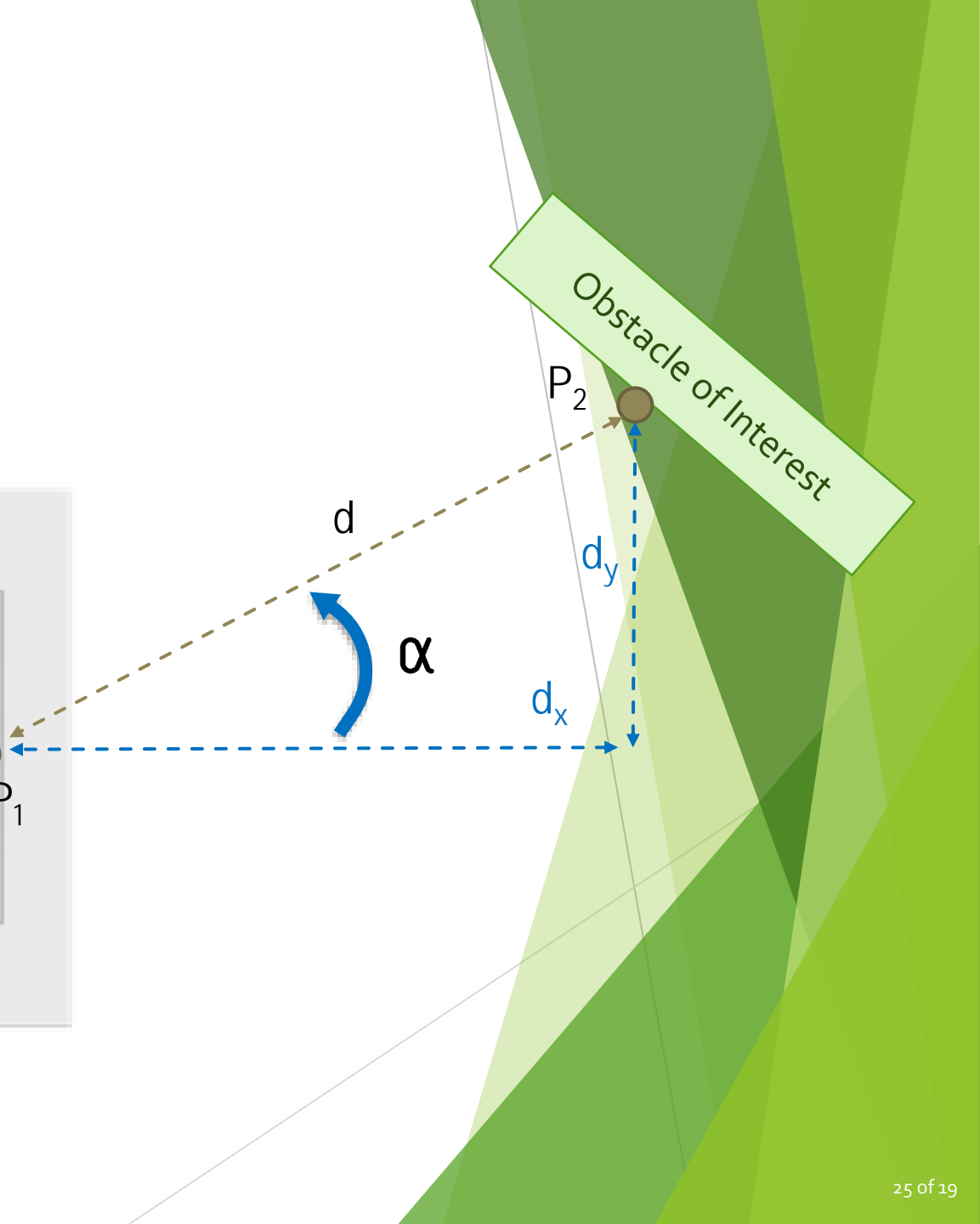
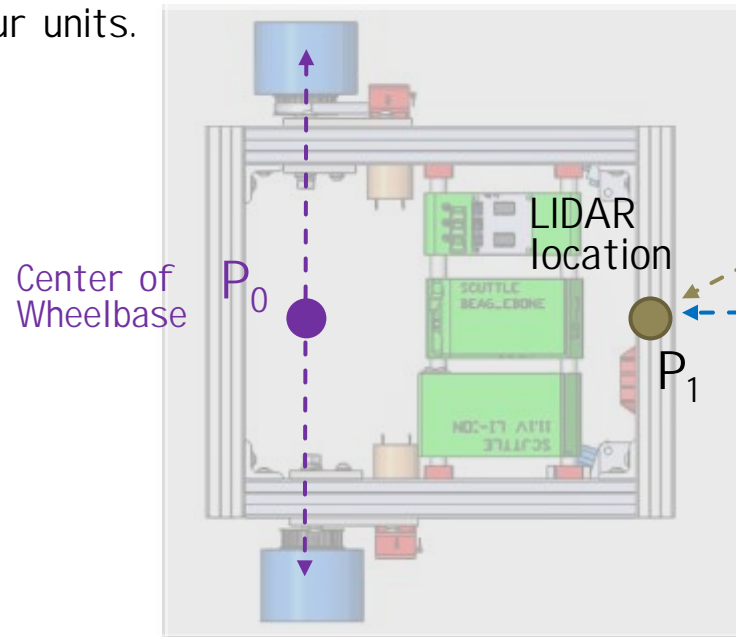
To be successful in using the device, you need to [see the datasheet](#) to understand the parameters of your device.





# LIDAR – measuring a point

- $P_1$  is the location of the lidar.
- The points will be initially measured from lidar and returned as pairs given by:
- $[d \text{ (mm)}, \alpha \text{ (degrees)}]$
- Python's numpy library performs math in radians. It is easy to convert back and forth but you must be aware of your units.



# Software For LIDAR

## Key Points:

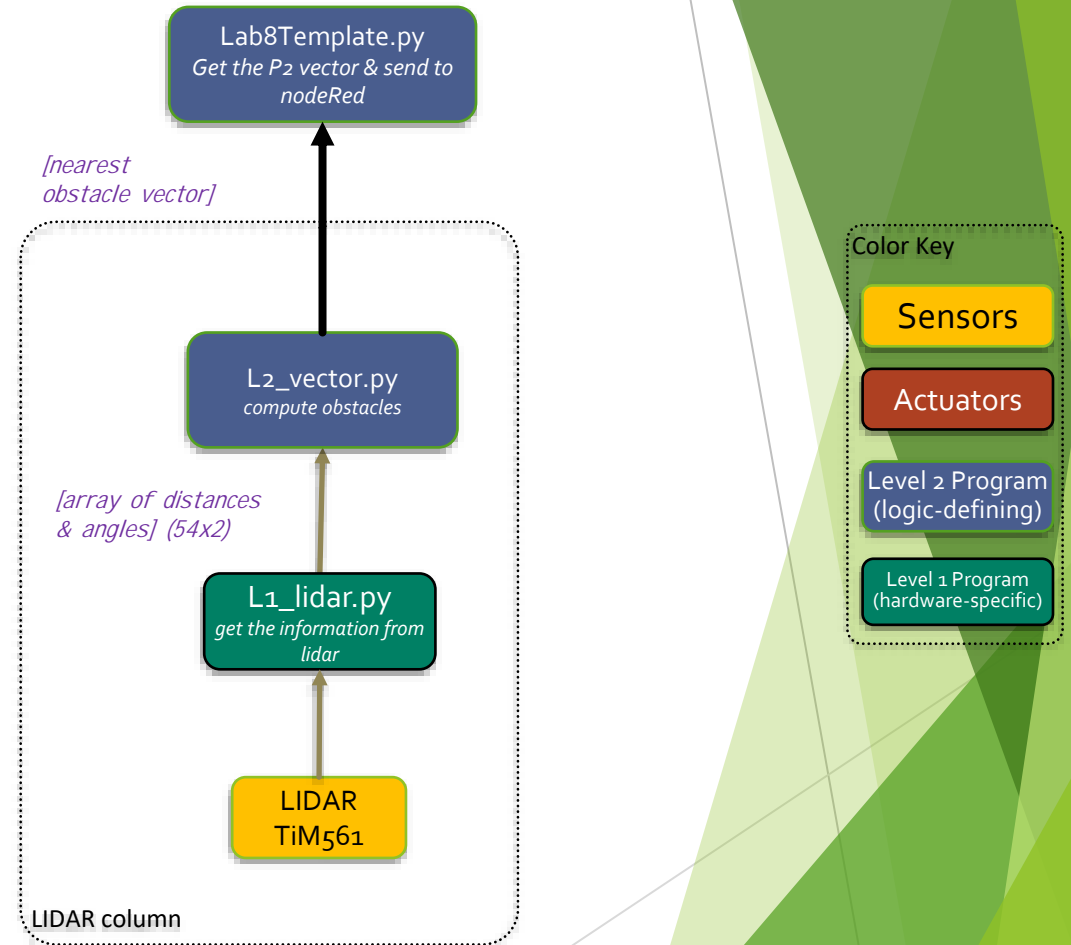
Software is using the numPy library to handle vectors and matrices. numPy computation is faster than raw python and requires proper syntax.

**Lidar scan frequency: 15hz**, so you cannot get new measurements faster than 66ms.

**L1\_lidar.py returns 54 measurements** by default and can return over 800 single points if desired, for more resolution.

**TiM561 LIDAR returns distances in meters.** Distances under 16mm are returned as error codes in case of poor reflection or other problem for a given measurement.

**L2\_vector.py can manipulate measurements**, with functions such as returning the nearest point, combining cartesian vectors, and converting vectors from polar to Cartesian coordinates.

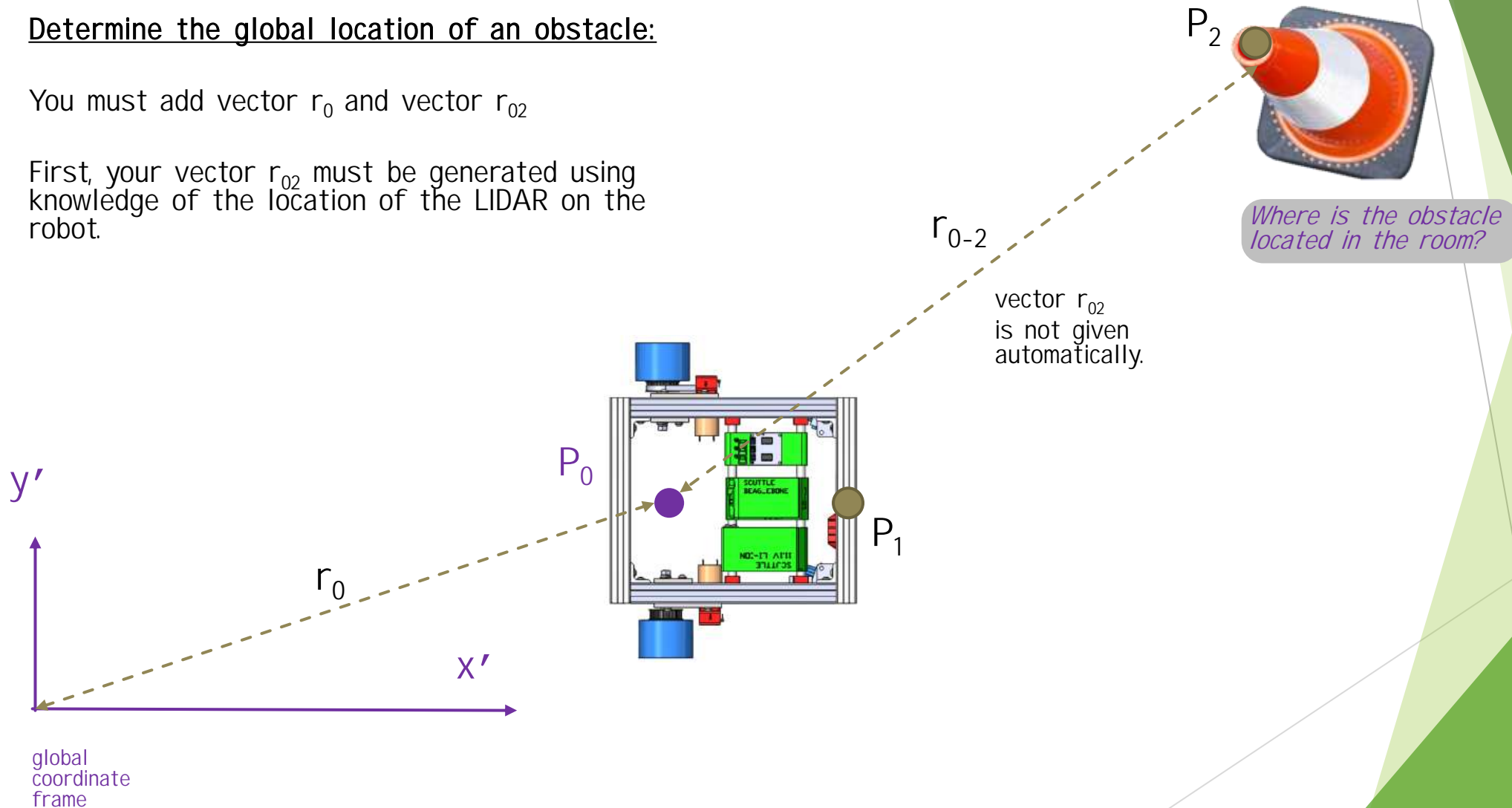


# Global Location of Obstacle

Determine the global location of an obstacle:

You must add vector  $r_0$  and vector  $r_{02}$

First, your vector  $r_{02}$  must be generated using knowledge of the location of the LIDAR on the robot.

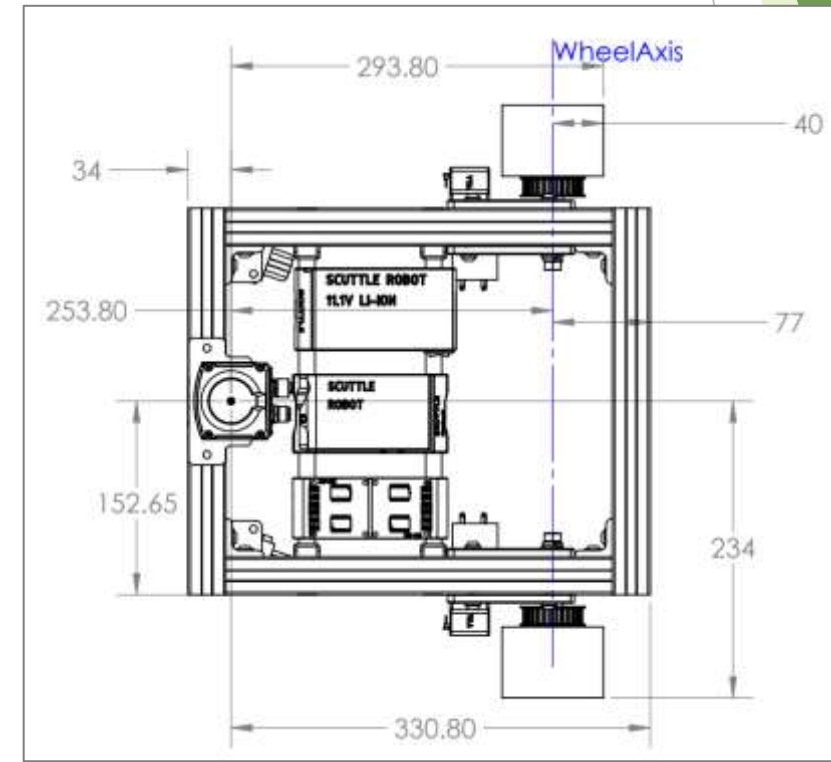
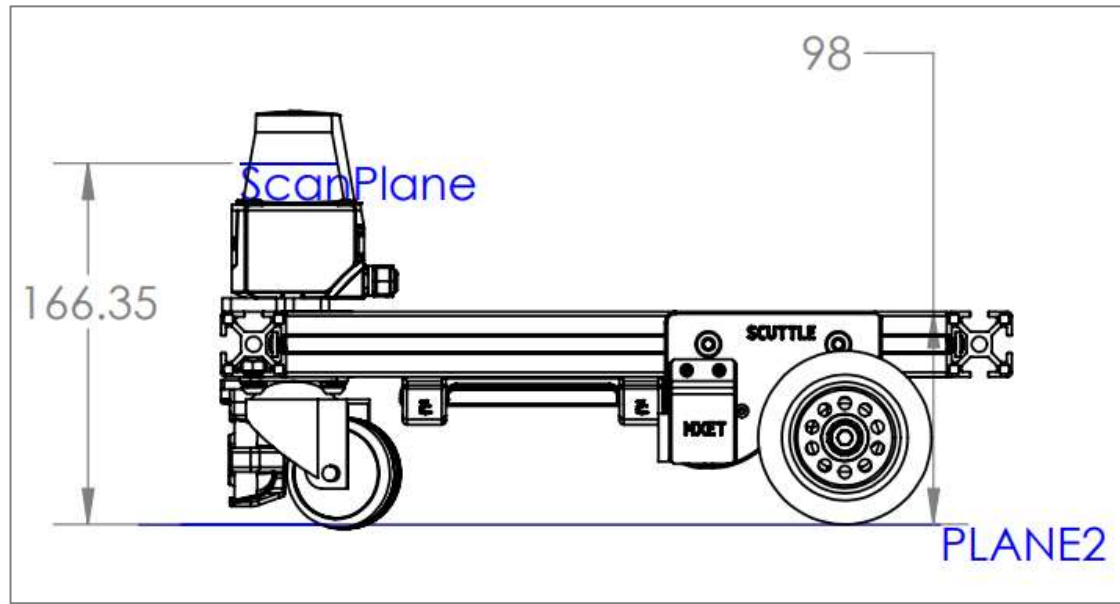


# Global Location of Obstacle

Determine the global location of an obstacle:

Lidar is located at positive 254mm in the x-direction on the robot.

The lidar beam is 166 mm above the floor.

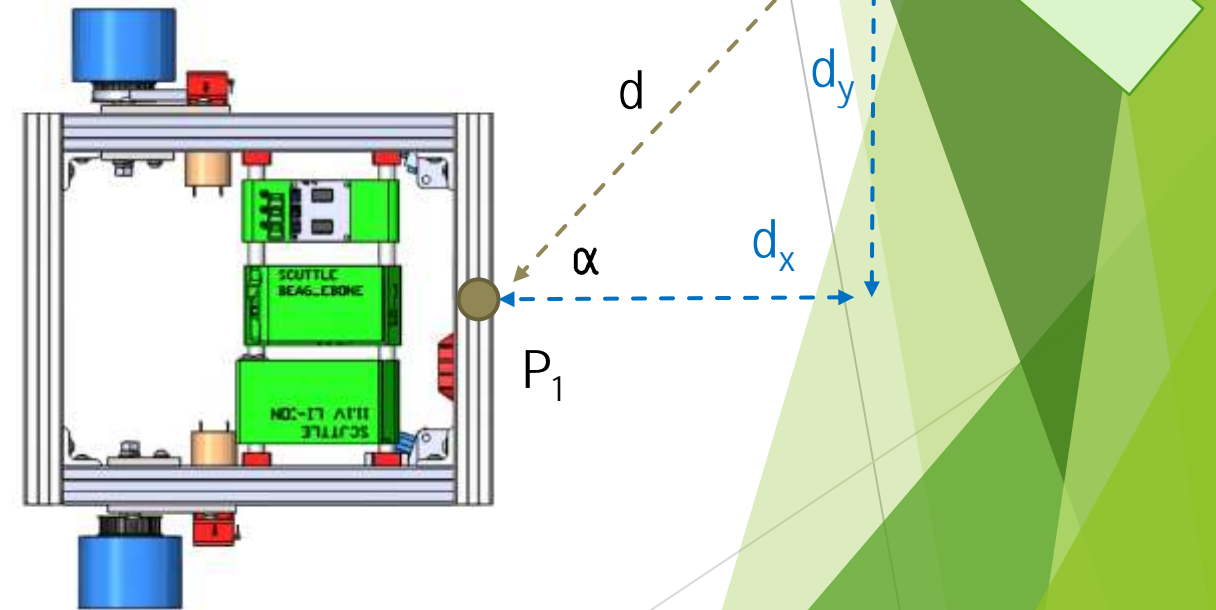


# Obstacle Avoidance by LIDAR

One method to avoid obstacles is to generate an imaginary spring which pushes on your robot and depends on the nearest obstacle.

$D_y$  is the y-component of distance  $d$

$D_x$  is the x-component of distance  $d$



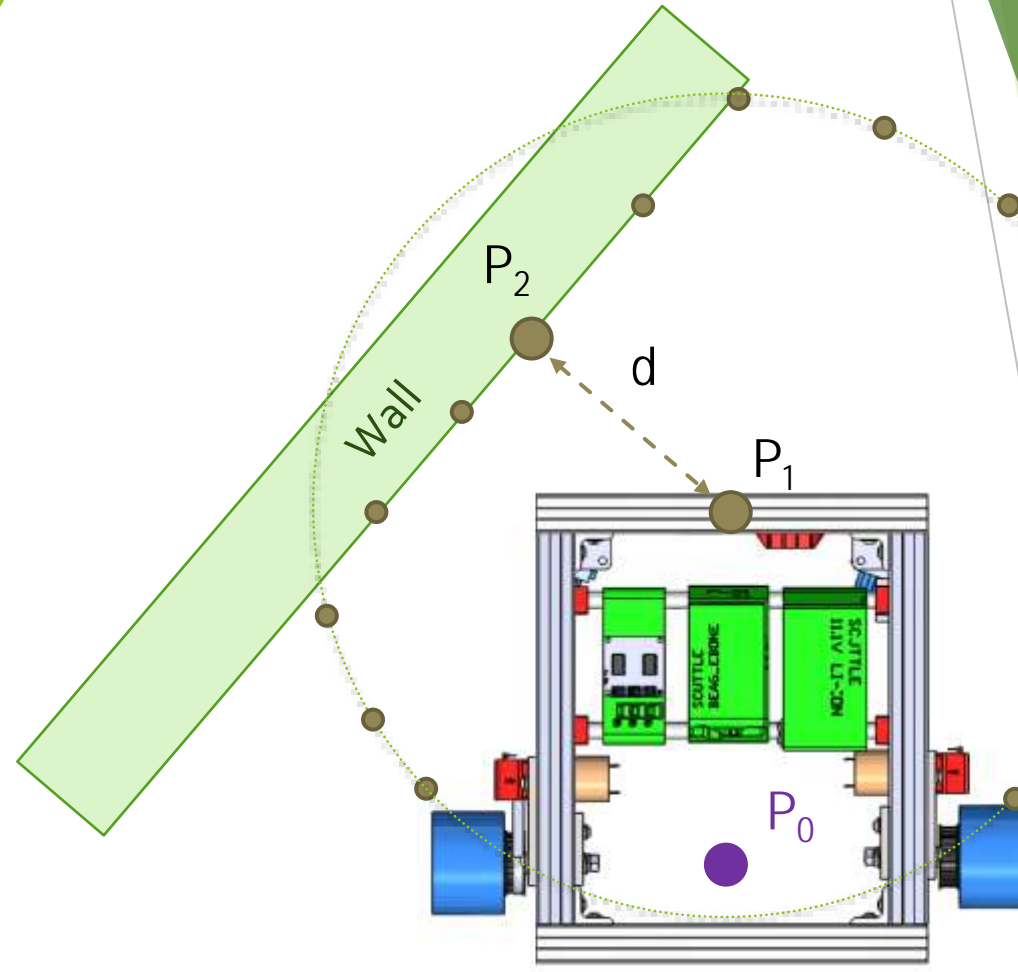
# Obstacle Avoidance by LIDAR

## Strategy:

The obstacle avoidance feature will try to detect the nearest objects to the robot and apply an “invisible force” to prevent the robot from crashing. The force is intended to act like a spring which is anchored to the nearest obstacle and pushes the robot at a point on the body, referred to as  $P_1$ .

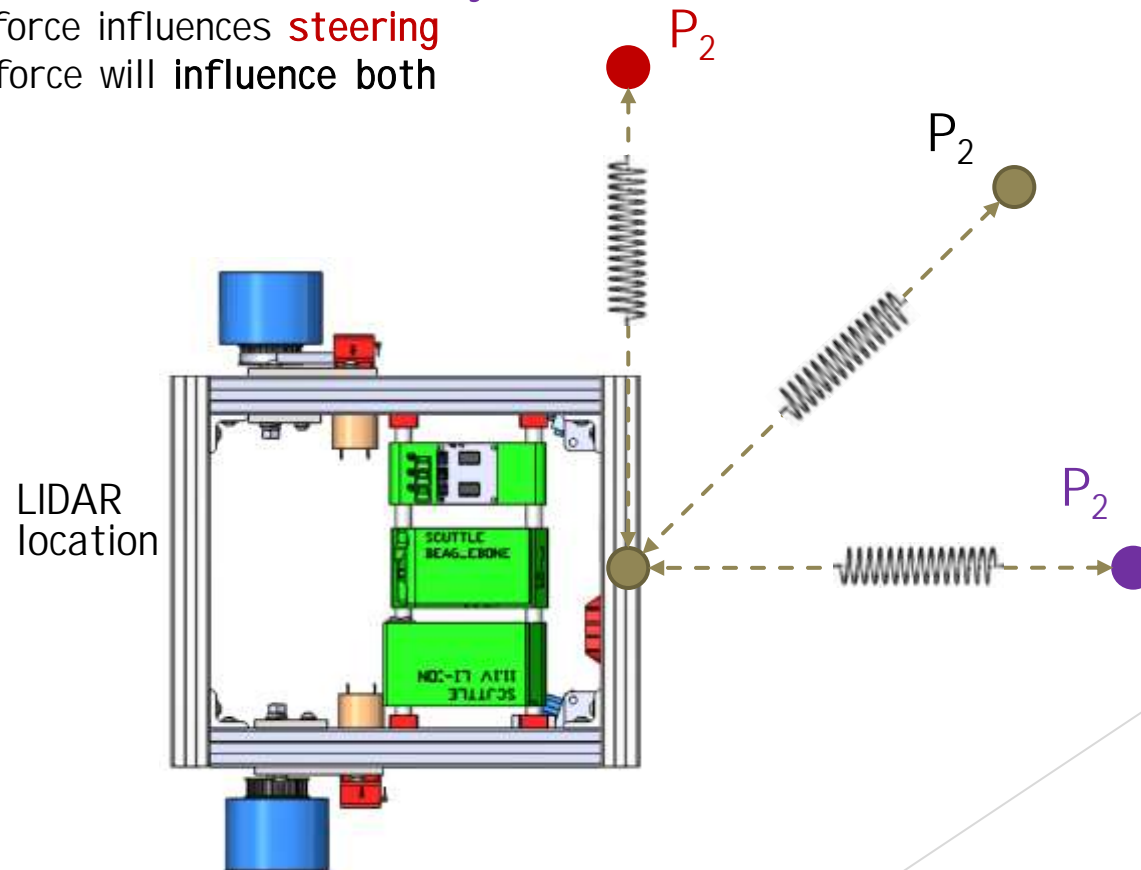
## The obstacle avoidance only deals with the body-fixed frame

- Define  $P_1$  as a point of interest on our robot.
- $P_2$  is assigned to the nearest point detected by the LIDAR scan.
- $d$  is the distance between point 1 and point 2
- We would like to handle all of these variables in:
  - body-fixed frame
  - Cartesian coordinates



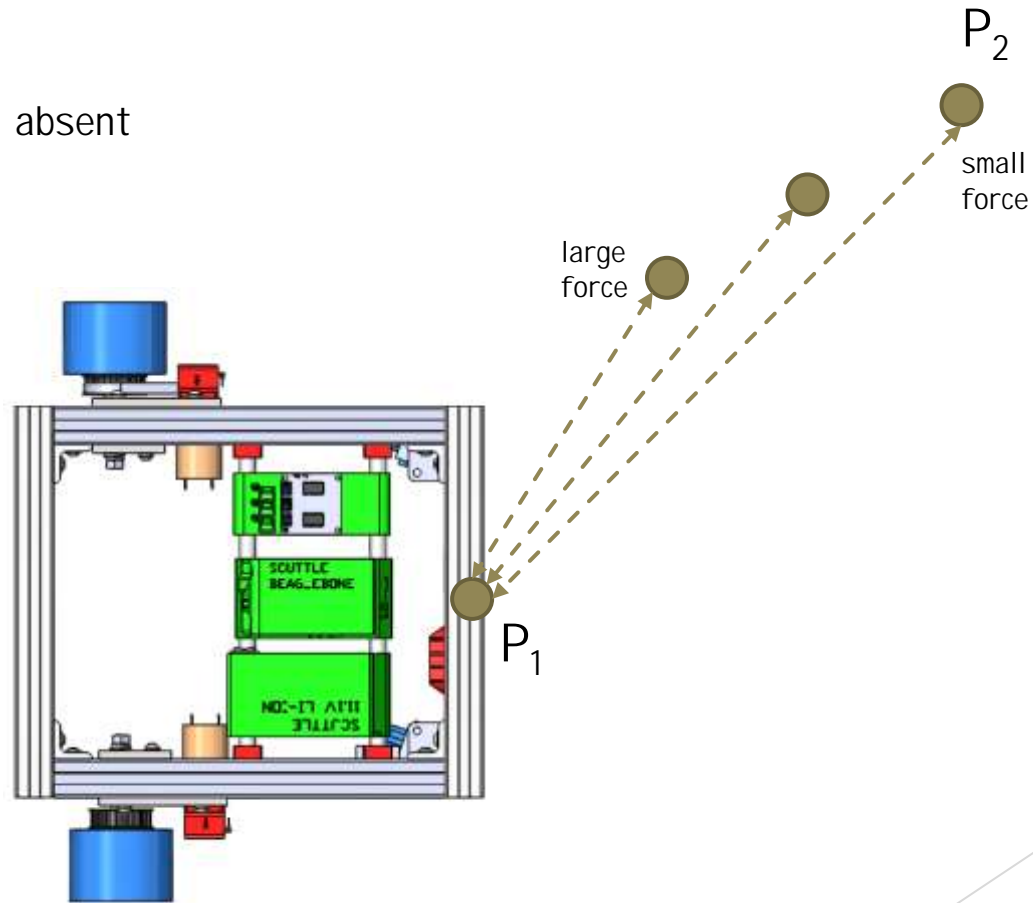
# Obstacle Avoidance – influence on velocity (translational and angular)

- If P2 is detected straight ahead, the force influences **velocity**
- If P2 is detected at the side, the force influences **steering**
- If P2 is detected at an angle, the force will **influence both**



# Obstacle Avoidance – Variable Force

- If  $d$  is large, the force is low
- If  $d$  is small, the force is high
- If  $d$  is larger than  $d_{\max}$ , the force is absent





# Quick Dive – Barometric Pressure

## COLLEGE STATION 4-DAY CHANGE

delta in pressure is 30.09-29.58 "Hg → 0.51" Hg  
delta pressure = 1.73kPa

## STANDARD PRESSURE CALCULATIONS

sea level std pressure: 101.3kPa  
pressure at 1000ft: 97.7kPa  
delta pressure = 3.6kPa  
elevation change represented by 1kPa = 278ft

What the Barometric pressure will tell you:  
1.73kPa change in pressure will represent  
480ft altitude change.

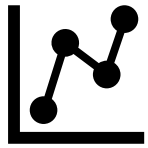
## High & Low Weather Summary for the Past Weeks

	Temperature	Humidity	Pressure
<b>High</b>	93 °F (May 28, 2:53 pm)	97% (May 17, 5:53 am)	30.09 "Hg (May 17, 5:53 am)
<b>Low</b>	65 °F (May 16, 4:53 am)	39% (May 15, 3:53 pm)	29.58 "Hg (May 21, 2:53 am)
<b>Average</b>	80 °F	76%	29.85 "Hg

\* Reported May 15 10:53 am — May 30 10:53 am, Bryan – College Station. Weather by CustomWeather, © 2019

Note: Actual official high and low records may vary slightly from our data, if they occurred in-between our weather recording intervals... More about our weather records

[Historic weather at timeanddate.com](https://timeanddate.com)



# Speed Control

This slide is dedicated to describing the wheel speed measurements and calculation of variables for speed control.



## Nominal conditions:

Battery: 11.5 volts, open circuit



Motors: equipped with standard 200 rpm gearbox



Wheels: 83mm diameter urethane wheels



Pulleys: motor = 15 teeth, wheel = 30 teeth

## Update Shaft positions (take reading)

- Measure position
- Capture time
- 10hz is sufficient



## Get Wheel Increments

- Just math



## Update Phi

- Integrate the wheel positions



## Update PhiDots

- Take derivatives
- Update latest wheelspeeds



## Chassis displacement

- Compute the displacement in theta and x



## Chassis speeds

- Take derivative of last movements w.r.t. change in time since last sample.

# Further Reading

- ▶ [https://en.wikipedia.org/wiki/Holonomic\\_\(robotics\)](https://en.wikipedia.org/wiki/Holonomic_(robotics))
- ▶ Connector types
- ▶ <http://dangerousprototypes.com/blog/2017/06/22/dirty-cables-whats-in-that-pile/>