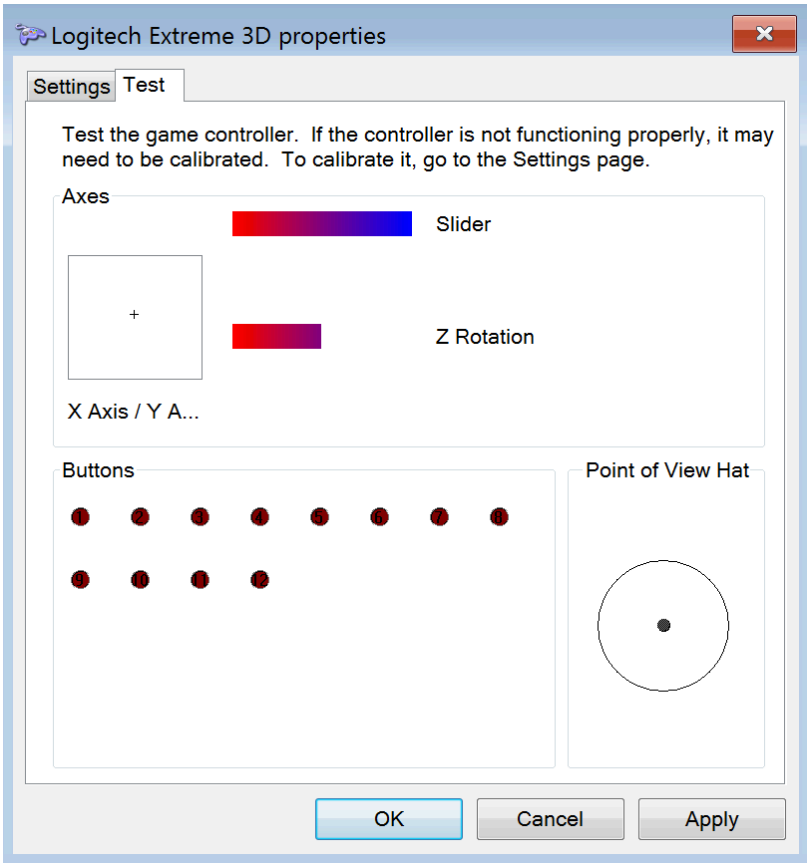
Control System Algorithm

Goal: Use Logitech joystick to control speed and direction of ROV thrusters

There are (at least) 2 key ideas behind how this ROV control system works and how control systems in general should be designed:

1) Define exactly how the system (ROV) should behave based on its inputs (joystick)
2) Understand how properties of the system (thrusters, etc.) should affect the design

These ideas may seem obvious, but they are very important. To start with, we need to understand how our joystick works. The joystick that this system uses is a Logitech 3D Extreme Pro Joystick. It connects to the control laptop via USB. When you plug the joystick in, its drivers should be installed automatically if they haven't been already. To see its inputs, open "Devices and Printers". Right click on the joystick, choose "Game controller settings", then properties. You should see a window that looks like this:



For now, we are only concerned with the X and Y axes of the joystick. The plus symbol represents the current position of the joystick and the square is a boundary for valid positions. If the plus is not centered by default, you should calibrate the joystick under the "Settings" tab. If you push the joystick as far as it can physically go in a certain direction, you should see the plus move towards a side of the square.

The joystick inputs are now being read by the control laptop. To use the joystick within the Python program (ROV_Control_2018_Final.py), we will use a module called "PyGame". The program imports and initializes PyGame, finds every joystick connected to the laptop, and creates an instance named "joystick" to refer to the one we have connected. To fully understand how to use PyGame, refer to its documentation: https://www.pygame.org/docs/

```
#import modules
import pygame
import math as m
import numpy as np
import time
from pymata_aio.pymata3 import PyMata3
from pymata_aio.constants import Constants

#initialize pygame
pygame.init()
pygame.joystick.init()
joystick = pygame.joystick.Joystick(0)
joystick.init()
```

To verify that PyGame is working correctly, run the test program (joysticktest.py). You should see a window that displays all of the joystick's inputs, similar to opening its properties in Windows.

Within the main loop of the control program, the joystick is polled.

```
#main loop
while True:

        #read joystick
        pygame.event.get()

        #store values
        jx = joystick.get_axis(0)
        jy = -joystick.get_axis(1)
```

Each axis value can range from -1 to 1. Pushing the joystick forward gives a raw y-axis value of -1, so the raw value is negated when assigning "jy". This is done so that the variables conform to a typical xy-coordinate system, where coordinates (jx, jy) = (1, 1) would correspond to the top-right corner of the joystick input window.

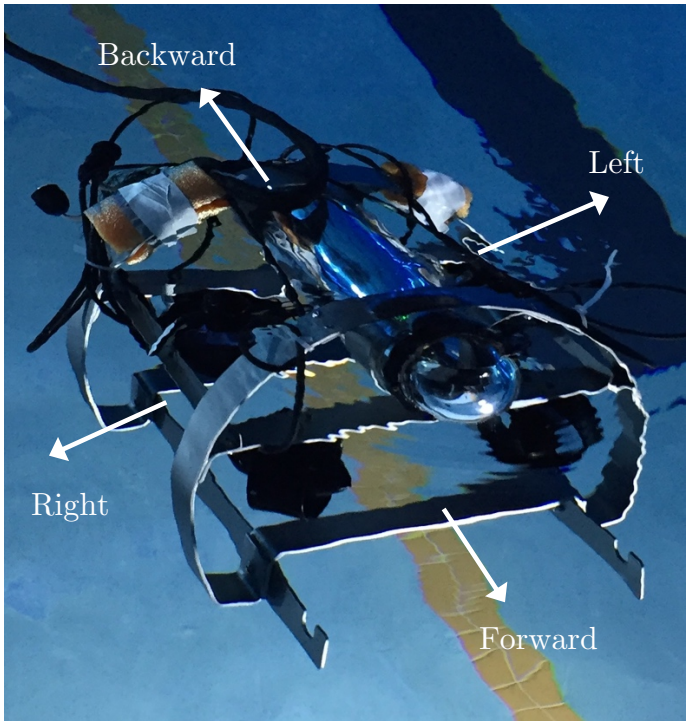
Now that we understand how the joystick works and how to use it in Python, we need to decide how we should use it to control the movement of the ROV (idea 1). Remember that this process relies on a series of design choices (idea 2). After making these choices, we will derive two conditions that the algorithm must satisfy to be considered functional.

Design Choices

Summary

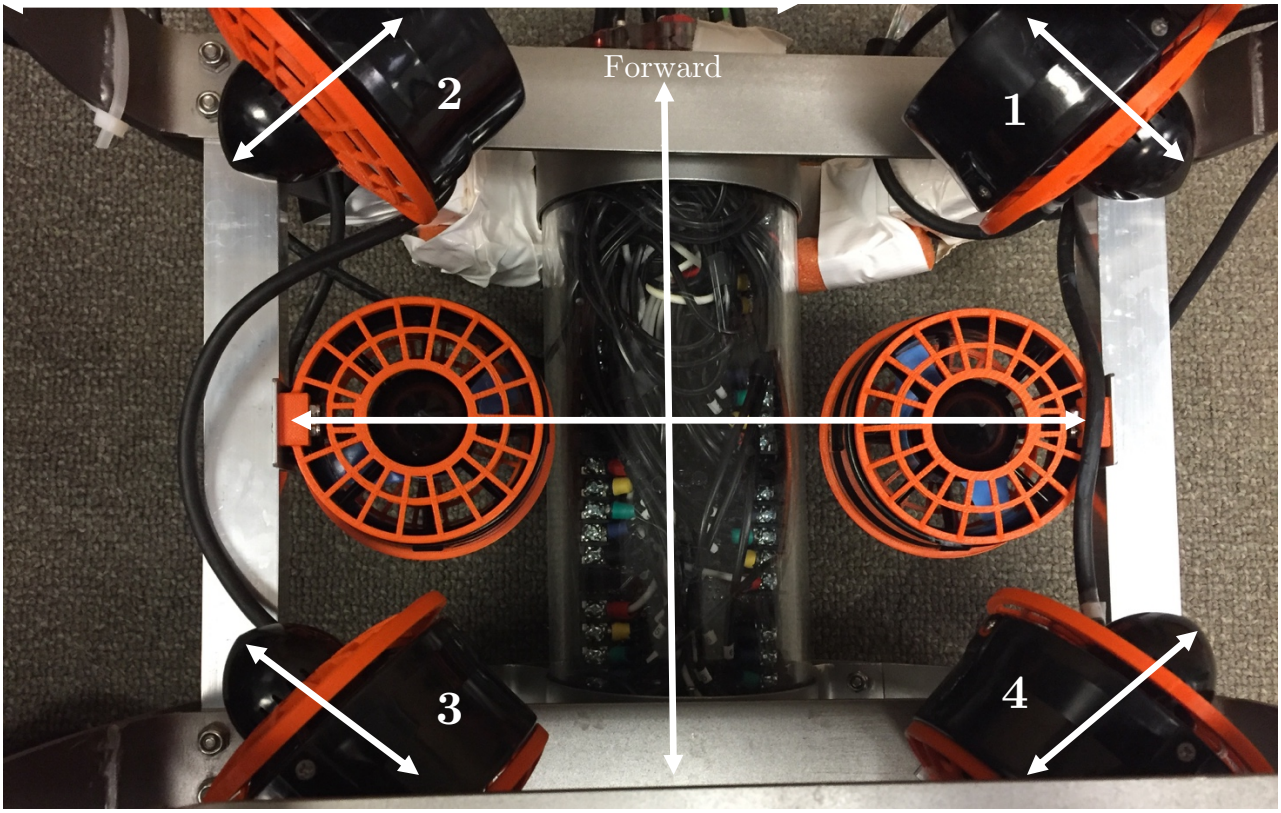
Direction: ROV should move in the same direction as the joystick is pushed
Thrusters: ROV thrusters will be offset 45° from its main directions (forward, backward, left, right)
Speed: ROV speed should be proportional to how "strongly" the joystick is pushed

Direction



Most simply, when we push the joystick forward, we want the ROV to go forward. When we push the joystick right, we want the ROV to go right. But this also applies to any direction in between, up to a full 360° range of motion. To understand how this is possible, we need to think in terms of vectors.

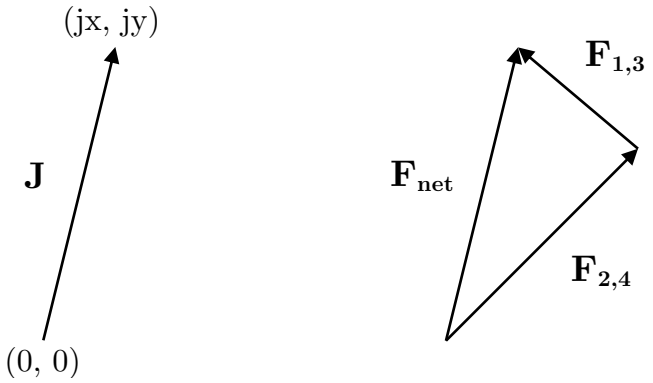
Thrusters



The ROV thrusters can only provide force forward or backward, in the direction of the arrows shown. The forward direction of force for a thruster is opposite to the direction that its nose faces. The forward force vectors provided by thrusters 1 and 2 point forward. The forward force vectors provided by thrusters 3 and 4 point backward. This is another design choice that was made.

It should now be clear that these four thrusters can be divided into pairs. Thrusters 1 and 3 can propel the ROV along the same axis. Thrusters 2 and 4 share a different axis that is offset by 90°. If we want the ROV to move along the 2,4 axis, we simply set these two thrusters to push the ROV in the same direction and leave the other two off.

To move in any direction other than along one of these two axes, both pairs of thrusters must be used. The two pairs will provide force in different directions, causing the net force acting on the ROV to be the sum of these two vectors.



We will also consider the joystick position as a displacement from the origin, and therefore a vector. Our design choice to have the ROV travel in the same direction as the joystick is meant to make control intuitive. To satisfy this choice, we arrive at our first condition: the sum of $\mathbf{F_{1,3}}$ and $\mathbf{F_{2,4}}$ (net force) should lie along **J**. (same angle)

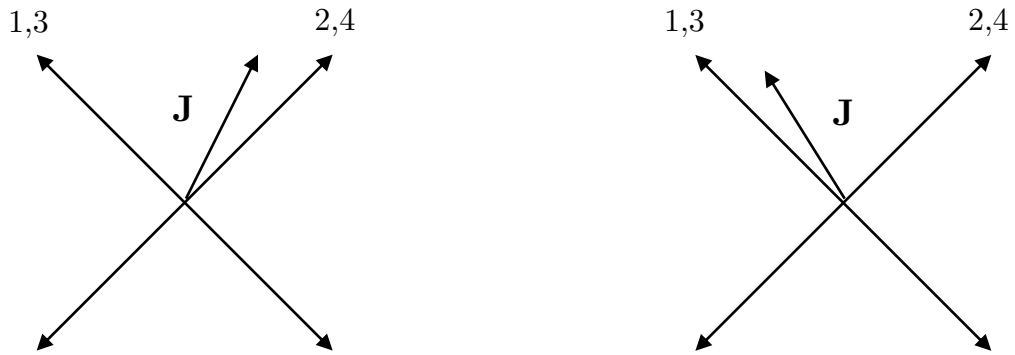
Speed

The ATmega8 microcontrollers on the board that control the thrusters are running tgy firmware, an "open source firmware for ATmega-based brushless ESCs". This firmware accepts a 16-bit throttle value over I^2C, with a decimal range of -32,768 to 32,767. A positive throttle value gives thrust in the forward direction, and a negative throttle value gives thrust in the backward direction.

For simplicity, the throttle values that the program initially calculates will have a range of 0 to 1 and will be scaled and possibly negated later in the program. For example, for a thruster to run at full throttle in the backward direction, the program first finds a value of 1 and later multiplies by it 32,767 (unconditional) and flips its sign (conditional) before sending it as a throttle value over I^2C.

Thruster pairs should always have the same speed. So, the program only needs to calculate two values. If the ROV needs to move in a direction closer to the axis belonging to a certain pair, that pair should provide more force. The initial value for the stronger pair is called "**a**", and the value for the weaker pair is called "**b**". Clearly, the

stronger and weaker pairs change based on direction, so the program conditionally assigns **a** and **b** to each thruster. We will again consider **a** and **b** vectors.



In the case on the left, **J** is closer to the 2,4 axis, so thrusters 2 and 4 provide more force and are assigned to be **a**. In the case on the right, **J** is closer to the 1,3 axis, so thrusters 1 and 3 provide more force and are assigned to be **a**.

So, how do we determine our speeds? Before we actually do any math, we need to define how our system should behave more clearly.

From an intuitive standpoint, if you push the joystick farther in a certain direction, the ROV should start moving faster in that direction. For example, if we push the joystick forward halfway, the thrusters should provide half as much force as if we push it all the way forward. This is really a comparison of ratios. A halfway pressed joystick should result in half throttle.

To put this into terms of **a**, **b**, and **J**, we will consider their magnitudes. The magnitude of the sum of **a** and **b** ($||\mathbf{a} + \mathbf{b}||$) is proportional to the net force acting on the ROV. The magnitude of **J** ($||\mathbf{J}||$) represents how far we have pushed the joystick in a certain direction. So, the ratio of $||\mathbf{a} + \mathbf{b}||$ to the maximum value that it could possibly have should always be equal to the ratio of $||\mathbf{J}||$ to the maximum value that it could possibly have. This is our second condition for a functioning algorithm.

$$\frac{\|\mathbf{a}+\mathbf{b}\|}{\|\mathbf{a}+\mathbf{b}\|_{max}} = \frac{\|\mathbf{J}\|}{\|\mathbf{J}\|_{max}}$$

It's worth noting at this point that creating an algorithm to determine a and b to satisfy this condition is considerably more difficult because of our thruster positioning. In fact, if our thrusters were aligned so that the axes along which they provide force overlapped the axes that we consider forward/backward and left/right movement, we could simply use jx and jy as base thrust values. Just replace $||\mathbf{a} + \mathbf{b}||$ in the equation with $||\mathbf{jx} + \mathbf{jy}||$, and it becomes easy to see how trivial this solution is ($\mathbf{J} = \mathbf{jx} + \mathbf{jy}$ by definition).

This solution won't work for our case. Let's think about how we could find all of the values in the ratio. We can always compute $||\mathbf{J}||$ using the current values for jx and jy as $||\mathbf{J}|| = \sqrt{jx^2 + jy^2}$. However, $||\mathbf{J}||_{\max}$ varies as a function of θ_J, the angle of $\mathbf{J}$ referenced to the positive x-axis.

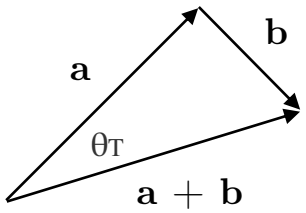
For example, if $\theta_J = 45°$, $||\mathbf{J}||_{\max} = 1.414$ because jx must be equal to jy and they both have a maximum value of 1. However, if $\theta_J = 90°$, $||\mathbf{J}||_{\max} = 1$ because jx must be 0 and jy has a maximum value of 1.

We have an idea for finding the right side of our ratio, but we still have the same problem we originally had: how do we choose **a** and **b**? This might seem circular, but we now have a framework for our solution. Let's come up with an example and think about how we might choose **a** and **b** to satisfy our condition. The process is simpler than you might think.

Let's consider a case for which jx = 1 and jy = 0.5. Whenever either jx or jy is equal to 1, this implies that the joystick is fully pushed in a certain direction. Since jx = 1 in this case, the joystick is pushed as far as it can go at this specific angle. This implies that we want full throttle and that $||\mathbf{J}|| \ / \ ||\mathbf{J}||_{\max} = 1$. **J** is closer to the 2,4 axis than the 1,3 axis, so thrusters 2 and 4 are assigned **a**. To achieve full throttle, it doesn't make sense to set ||**a**|| to anything less than 1. If ||**a**|| was less than 1, we wouldn't really be at full throttle. In a general sense:

$$||\mathbf{a}|| = ||\mathbf{J}|| \ / \ ||\mathbf{J}||_{\max}$$

If we know the value of ||**a**||, there is only a single value of ||**b**|| for which **a** + **b** will point in the right direction.



This should look familiar. Since **a** and **b** are offset by 90°, they form a right triangle with **a** + **b**. We can find the offset between **a** and **a** + **b** as $\theta_T = 45\text{deg} - \theta_J$. Then, we can find ||**b**||:

$$||\mathbf{b}|| = ||\mathbf{a}|| * \tan(\theta_T)$$

In essence, the purpose of the algorithm is simple: compute ||**b**||. The method of computing θ_T is conditional, but this is a general solution.

We can verify this method formally satisfies our second condition.
(skip this part if you don't care about a more formal proof)

Assume 0 < theta_j < 45:

abs(jx) > abs(jy)

J = sqrt(jx^2 + jy^2)
theta_j = arctan(jy/jx)
Jmax = sqrt(1 + (tan(theta_j)^2)) = sqrt(1 + (jy/jx)^2)

a = jx
b = jx * tan(45 - theta_j)
a+b = sqrt(jx^2 + (jx*tan(45 - theta_j))^2)

amax = 1
bmax = tan(45 - theta_j)
(a+b)max = sqrt(1 + (tan(45 - theta_j))^2)

square both sides

(jx^2 + jy^2) / (1 + (jy/jx)^2) = (jx^2(1+tan^2(45 - theta_j))) / (1+tan^2(45 - theta_j))

cancel out

(jx^2 + jy^2) / (1 + (jy/jx)^2) = jx^2

multiply top and bottom of left side by jx^2

jx^2(jx^2 + jy^2) / (jx^2 + jy^2) = jx^2

cancel out

jx^2 = jx^2 -> J/Jmax = (a+b)/(a+b)max