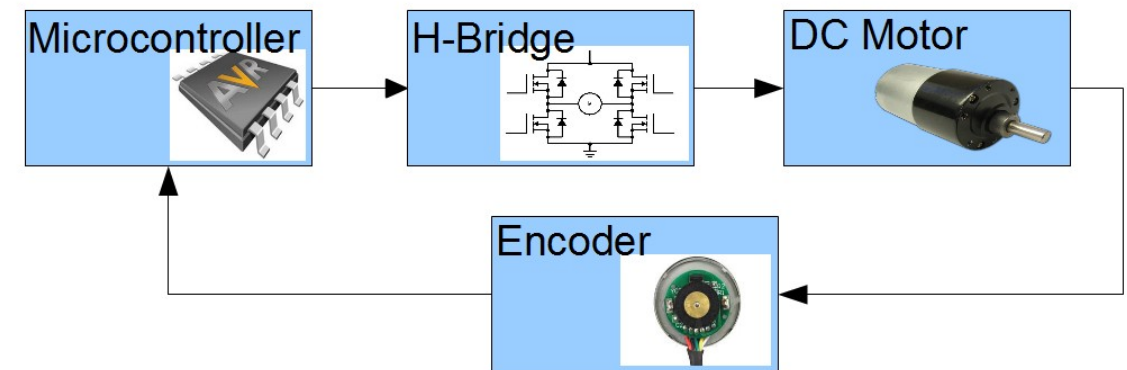
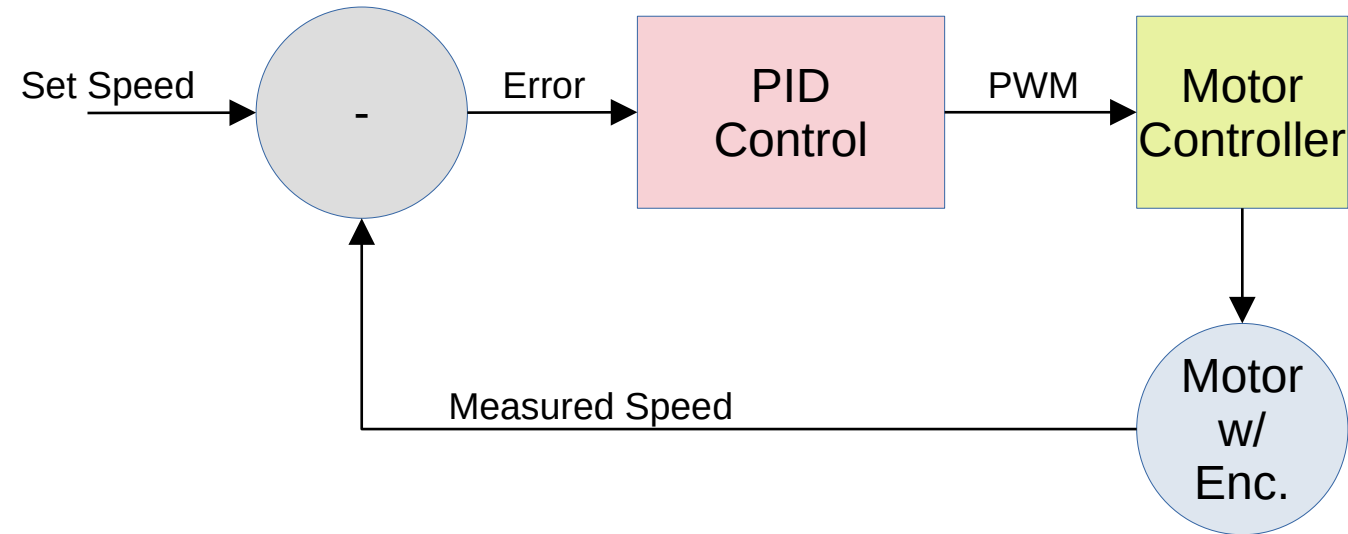


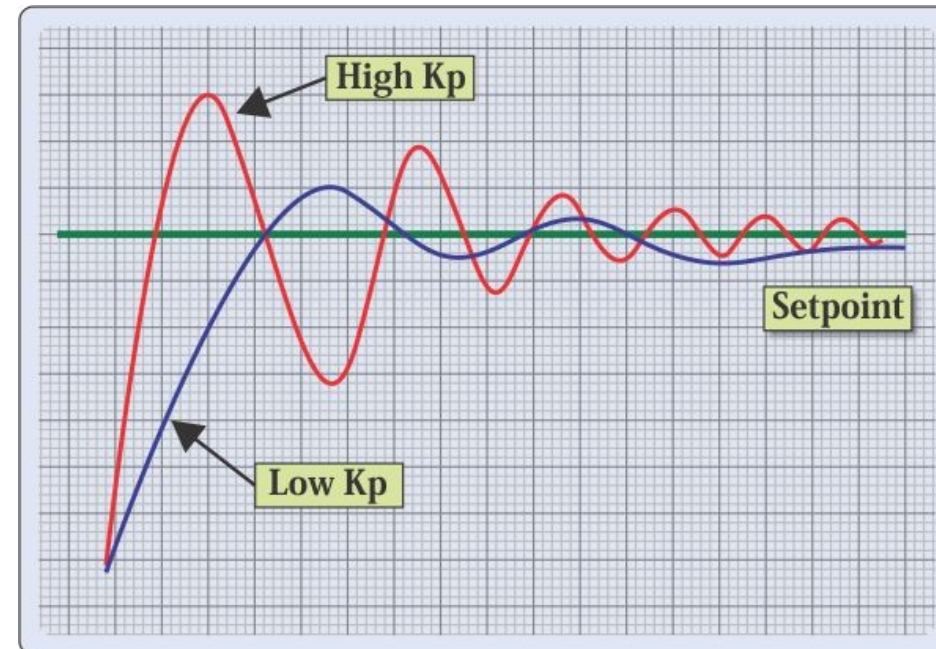
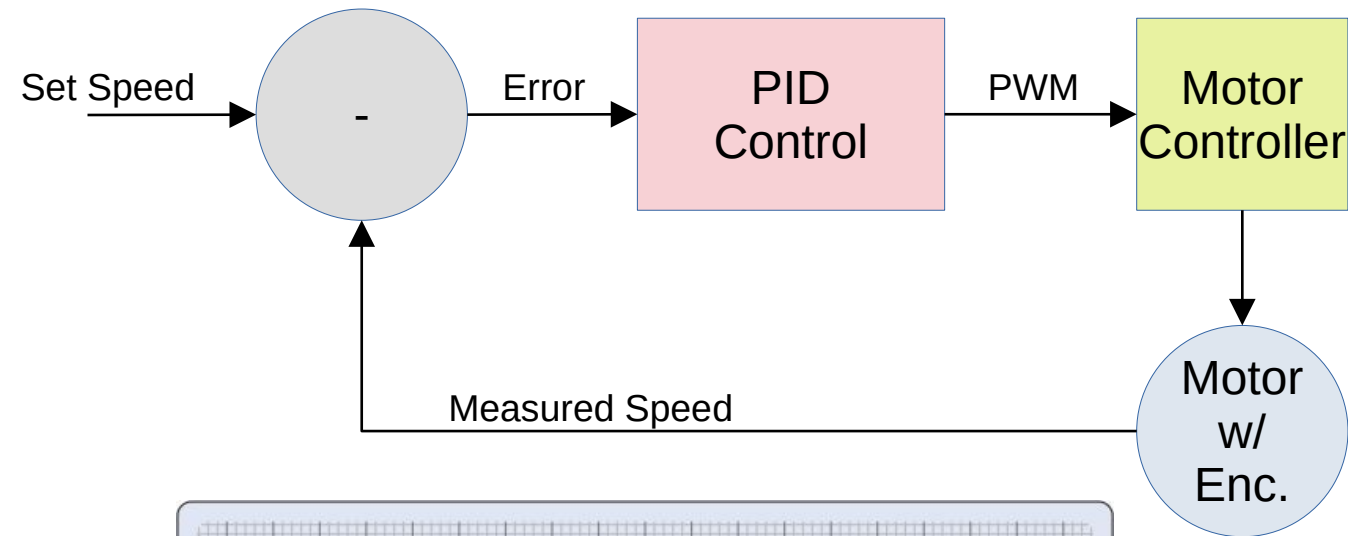
- As you have noticed, it is very hard to keep both wheels moving at the same speed. This is due to manufacturing tolerances of the motors and manufacturing quality. Expensive high quality robots have similar issues.
- One way to get the wheels to operate at the same speeds is to implement a control loop called PID.
- PID is Proportional, Integral, Derivative control.
- An example of a PID control loop is the cruise control for cars. You set the speed and the cruise control maintains it.
- In order for a control loop to work, it needs to measure the speed of the motor. A car has a speedometer. The robot has wheel encoders. Both work in a similar way.



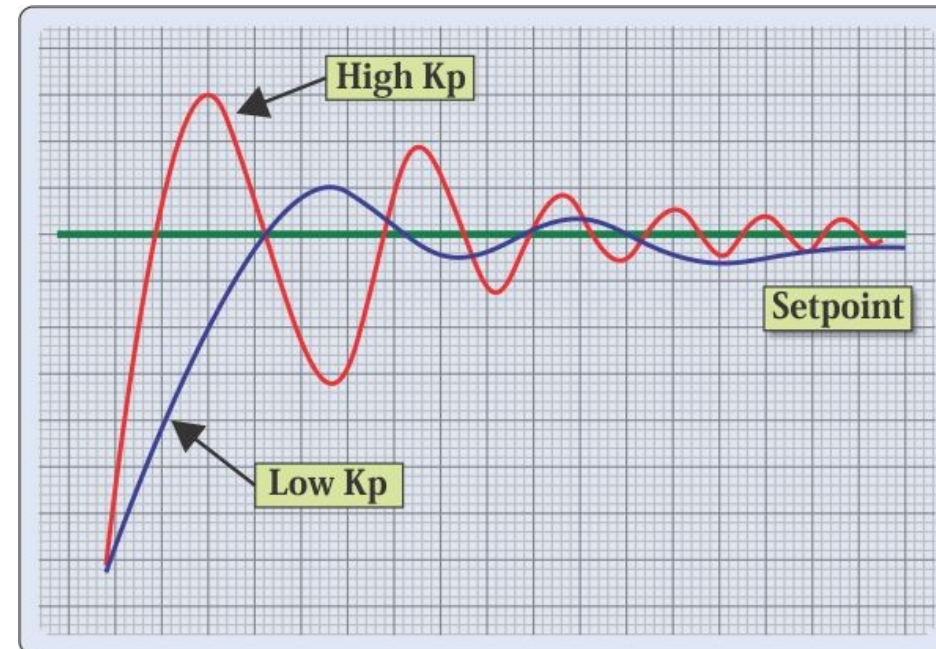
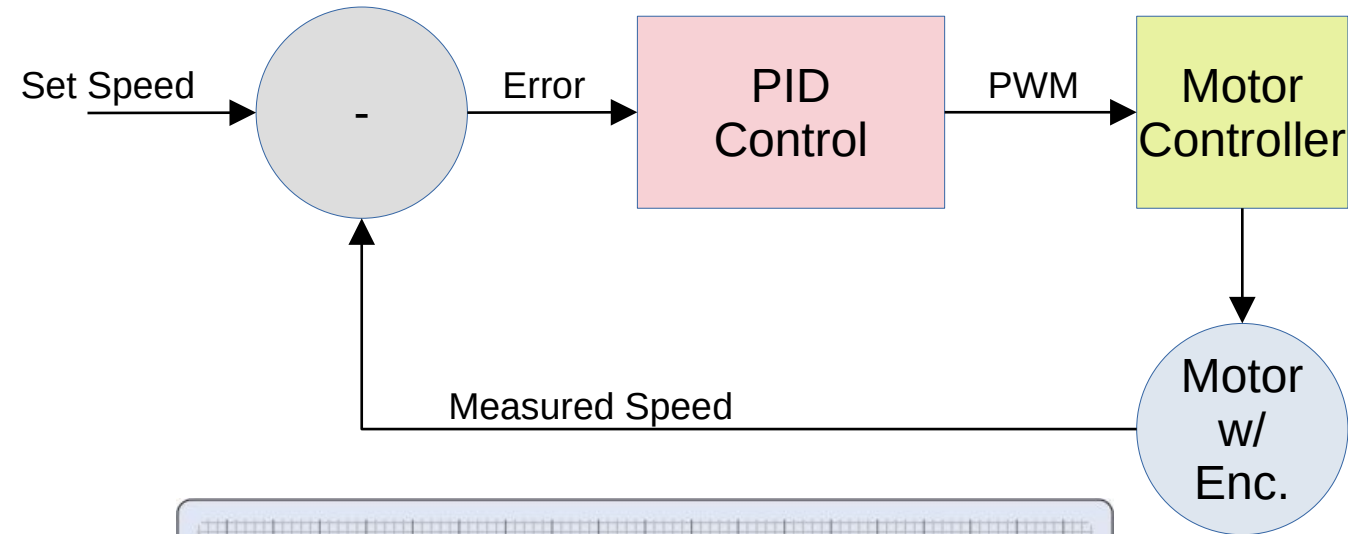
- For a PID controller to work, it requires two inputs, the desired speed and the measured speed. The PID controller takes those inputs and calculates a value to apply to the motor controller as a PWM duty cycle value.
- The way the PWM controller works is that a set speed is specified. In our robot, the speed will be the number of encoder pulses per 20 milliseconds.
- The measured speed is subtracted from the set speed. The result is called the error signal. This error is used by the PID controller to adjust the PWM duty cycle until the measured speed equals the set speed.



- The motor will never stay at the set speed so the PID controller will adjust the PWM duty cycle to try and maintain the set speed.
- The PID controller has three gain settings. These settings control how fast the motors get to the set speed and how accurately it can maintain the speed.
- The plot shows an example of different gain values and how well the speed is controlled. This plot shows the response to the proportional gain value. Too high gain and the motors will speed past beyond the set speed and then go below but eventually get to the right speed.
- We will focus mainly on the proportional gain.



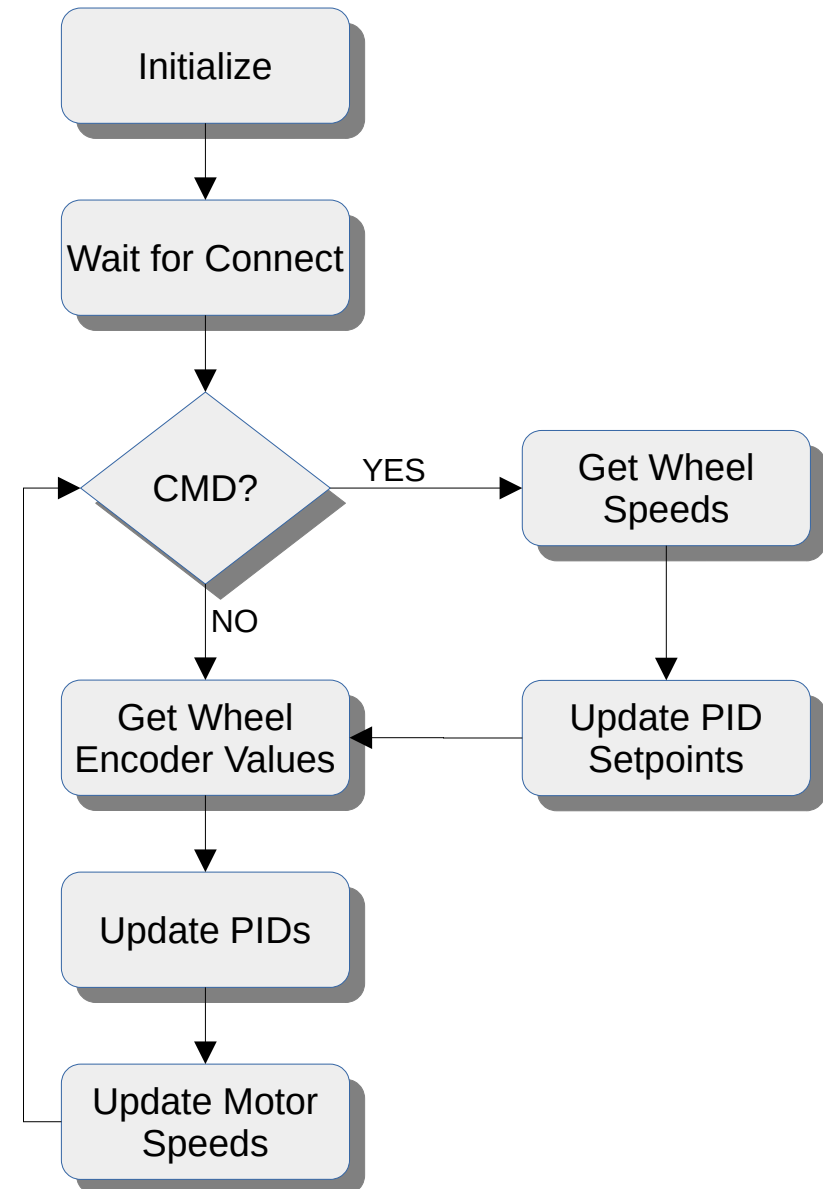
- The three gain control values for the PID controller is
 - K_p – proportional gain
 - K_i – Integral gain
 - K_d – derivative gain
- Since this is a simple robot, we do not need the derivative gain so it will be set to zero.
- The integral gain helps smooth the speed changes and we will set it to 0.02.
- You will experiment with the K_p , proportional gain control.



- Download the file **pid.py** to your computer.
- Find the file **pid.py** and right click on it and select **Upload**.
- The file should now appear in the robot file list.
- The pid module includes a function to create a PID controller and set its parameters:
 - `pid = PID(proportional, integral, derivative, setpoint =0)`
 - The four arguments are the proportional gain, integral gain, derivative gain, and the initial setpoint which is zero.
- To perform a calculation:
 - `output = pid(input)`
 - input will be the encoder value difference from the last value and the current value.
- To set the output limits
 - `pid.output_limits - (lowest,highest)`
 - This keeps the pid output within a specific range such as the range the motor controller can support.

Robot Program Overview

- The flow chart shows how the following robot program will operate.
- First, all needed variables are initialized, the PID controllers are initialized, and the wheel encoder pins and interrupts are initialized.
- Next, the program will wait for the laptop to connect.
- Then the program will go into a loop and first check for a command setting the wheel speeds.
- If there is a command, the PID controller setpoints will be updated.
- If there is not command or after processing the command, the wheel encoder values will be used to calculate the the change in values since the last loop.
- The PID controllers will be updated with the calculated change values and the motors will updated with a new speed setting.



- More details on the loop updating the PID.
- The setpoint is the number of encoder ticks per loop cycle. The loop cycle is about 20 ms.
- The input to the PID is the number of encoders ticks from the last loop cycle.
- The number of encoder increments or ticks is calculated by subtracting the last encoder value from the current encoder value.
- The difference value is the argument to the `pid()` function. The result is the speed to set the motor so the encoder value difference matches the setpoint.

Capturing Wheel Movement

- Open a new file in Thonny and enter the program to the right. This program is multiple pages and formatted as a class. This will help organize and modularize the code. Name it **rover7.py**.
- A class always has an initialization function called **__init__()**. That is two underscore characters on each side. This function is automatically called when the class is instantiated or called.
- As before, the I2C interface and the UGV get instantiated.
- **last_enc[]** is a list that will hold the last encoder values.
- **inpid[]** is a list that will hold the difference between the current encoder and last encoder values.

```
import ugv
import time
from machine import Pin,I2C
import select
import connect
from pid import PID

class UGV():
    def __init__(self):
        i2c = I2C(sda=Pin(4),scl=Pin(5))
        drv = ugv.ugv(i2c)
        last_enc = [0,0,0,0]
        inpid = [0,0,0,0]
        outm = [0,0,0,0]
        pid = []
        for i in range(4):
            pid.append(PID(3.0,0.002,0.0,setpoint=0))
            pid[i].output_limits = (-127,127)
        self.sk,self.addr = connect.connect()
```

Capturing Wheel Movement

- **outm[]** is a list that will hold the output of the PID controller for each wheel.
- pid is created as an empty list.
- The for loop will loop four times and assign each list element a PID control loop. The output limits are also set for each control loop.
- Then the program will wait for the laptop to connect.
- The connect function will return the UDP socket connection and address. They are assigned to variables with self in front so they can be used in future functions that will be added later to this program.

```
import ugv
import time
from machine import Pin,I2C
import select
import connect
from pid import PID

class UGV():
    def __init__(self):
        i2c = I2C(sda=Pin(4),scl=Pin(5))
        drv = ugv.ugv(i2c)
        last_enc = [0,0,0,0]
        inpid = [0,0,0,0]
        outm = [0,0,0,0]
        pid = []
        for i in range(4):
            pid.append(PID(3.0,0.002,0.0,setpoint=0))
            pid[i].output_limits = (-127,127)
        self.sk,self.addr = connect.connect()
```

Capturing Wheel Movement

- Continuing, the while loop is entered.
- In the while loop, the function **select.select()** checks to see if a packet has been received. It will wait .01 seconds. If a packet is received within the .01 seconds, select.select() will return with ready[0] equal to 1.
- If select.select() times out, ready[0] will equal to 0.
- If a packet is received, the program then gets the packet and decodes it. It then splits the packet into a list which should have two elements.
- The setpoints for the PID controllers are set using the values received.
- If any of the values are zero, the pid controller gets reset which will stop the motors.

```
while 1:
    ready = select.select([self.sk],[],[],0.01)
    if ready[0]:
        d,a = self.sk.recvfrom(64)
        print(d)
        d = d.decode().split(' ')
        if len(d) == 2:
            left = float(d[0])
            right = float(d[1])
            pid[0].setpoint = left
            pid[1].setpoint = left
            pid[2].setpoint = right
            pid[3].setpoint = right
            if left == 0:
                pid[0].reset()
                pid[1].reset()
            if right == 0:
                pid[2].reset()
                pid[3].reset()
        enc = drv.get_encoders()
        for i in range(4):
            inpid[i] = enc[i] - last_enc[i]
            last_enc[i] = enc[i]
            outm[i] = pid[i](inpid[i])
        drv.drive(int(outm[0]),int(outm[1]),\
                  int(outm[2]),int(outm[3]))
        time.sleep(0.02)
```

UGV()

Robot Program

Capturing Wheel Movement

- The encoder values are collected and in the for loop, the number of encoder counts is calculated by differencing the last encoder value from the current one.
- last_enc is updated with the current encoder value.
- The pid controller takes the number of encoder counts and calculates the motor speed value.
- After all four are calculated, the motor controller is updated with the new speed values.
- A 20 ms delay is added to give the motors time to change speed and for new encoder values.
- The loop then repeats.

```
while 1:
    ready = select.select([sk],[],[],0.01)
    if ready[0]:
        d,a = sk.recvfrom(64)
        print(d)
        d = d.decode().split(' ')
        if len(d) == 2:
            left = float(d[0])
            right = float(d[1])
            pid[0].setpoint = left
            pid[1].setpoint = left
            pid[2].setpoint = right
            pid[3].setpoint = right
            if left == 0:
                pid[0].reset()
                pid[1].reset()
            if right == 0:
                pid[2].reset()
                pid[3].reset()
        enc = drv.get_encoders()
        for i in range(4):
            inpid[i] = enc[i] - last_enc[i]
            last_enc[i] = enc[i]
            outm[i] = pid[i](inpid[i])
        drv.drive(int(outm[0]),int(outm[1]),\
                  int(outm[2]),int(outm[3]))
        time.sleep(0.02)
```

UGV()

- On the laptop, enter the program into a new file using IDLE. Call it **idle7.py**.
- This program connects to the robot and lets you enter speed values for both wheels.
- The network setup and connection should be familiar now.
- In the while loop, the program waits for an input. It then sends that input to the robot.
- The input should be two numbers separated by a space.
- Start the program on the robot
- Connect the laptop WiFi to the robot.
- Start this program.
- Enter 20 20 and press enter. The robot should start moving. Try other values.

```
import socket

addr = ('192.168.4.1', 10000)
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.sendto(b'start', addr)

while 1:
    print("enter speeds")
    a = input()
    s.sendto(a.encode(), addr)
```

Laptop Program

Capturing Wheel Movement

- Stop the robot and add a print statement.
- Click on the View menu in Thonny and select plotter. This will open a plot.
- Run the robot program again and do not disconnect the USB cable from the robot.
- Hold the robot in one hand and enter 20 20 into the command.py program.
- The plot will show inpid and outm for the front left motor.
- You can try slowing the wheel down a little and you should see outm increase in value as the PID tries to get the motor to operate at the setpoint speed.

```

while 1:
    ready = select.select([self.sk],[],[],0.01)
    if ready[0]:
        d,a = self.sk.recvfrom(64)
        print(d)
        d = d.decode().split(' ')
        if len(d) == 2:
            left = float(d[0])
            right = float(d[1])
            pid[0].setpoint = left
            pid[1].setpoint = left
            pid[2].setpoint = right
            pid[3].setpoint = right
            if left == 0:
                pid[0].reset()
                pid[1].reset()
            if right == 0:
                pid[2].reset()
                pid[3].reset()
        enc = drv.get_encoders()
        for i in range(4):
            inpid[i] = enc[i] - last_enc[i]
            last_enc[i] = enc[i]
            outm[i] = pid[i](inpid[i])
        drv.drive(int(outm[0]),int(outm[1]),\
                  int(outm[2]),int(outm[3]))
        print(indip[0],outm[0])
        time.sleep(0.02)

```

UGV()

Robot Program

- We will continue adding to the robot program.
- The idea is that the robot will receive commands and send telemetry. The laptop will provide a user interface to control the robot and display telemetry. At some point, the laptop will use the telemetry in its own control loop to control the robot.

