ARMABOT

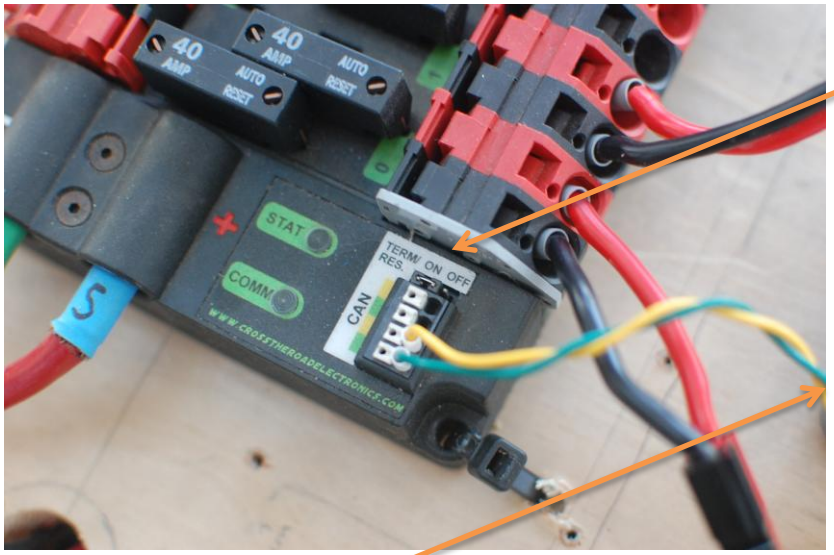
Encoder Technology Tutorial

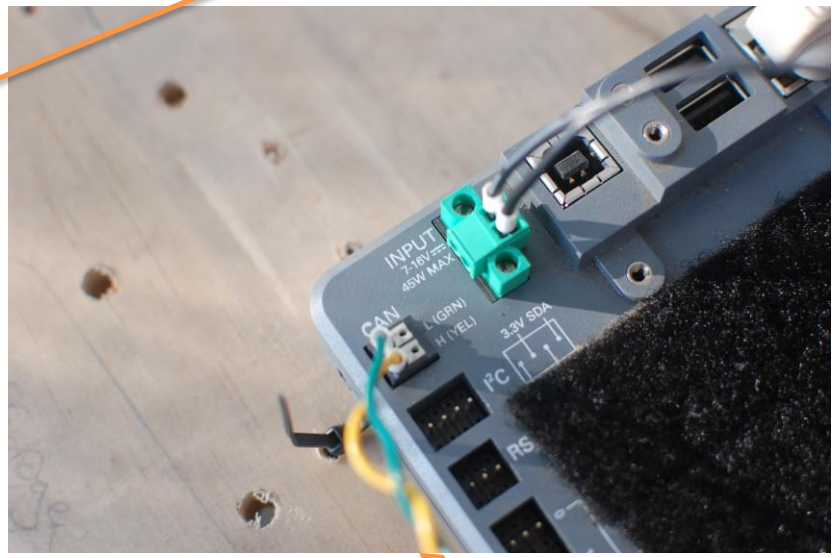
Special thanks to FRC team 5818 for putting together this tutorial

Table of Contents:

Wiring the CAN bus

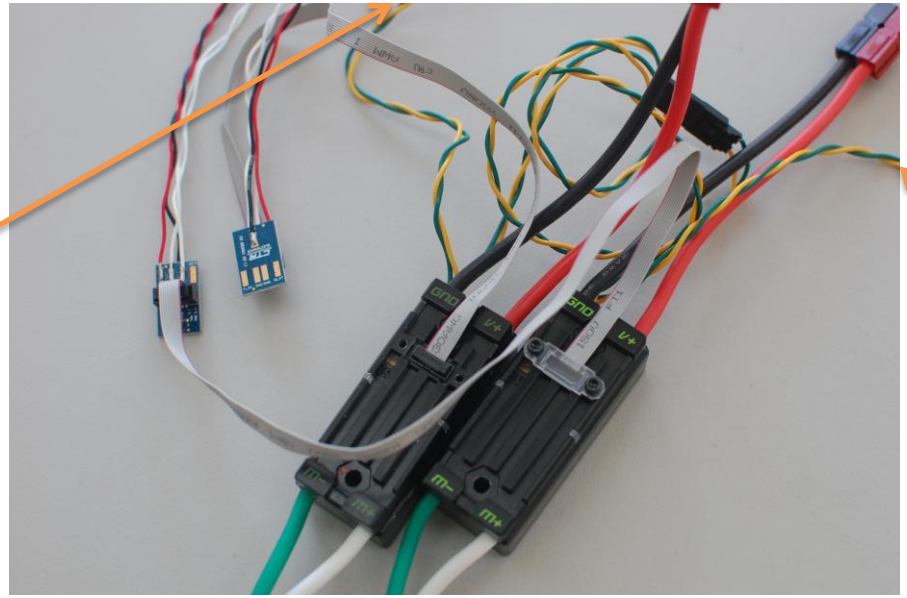
Jumper in “ON” position





To Talon SRX

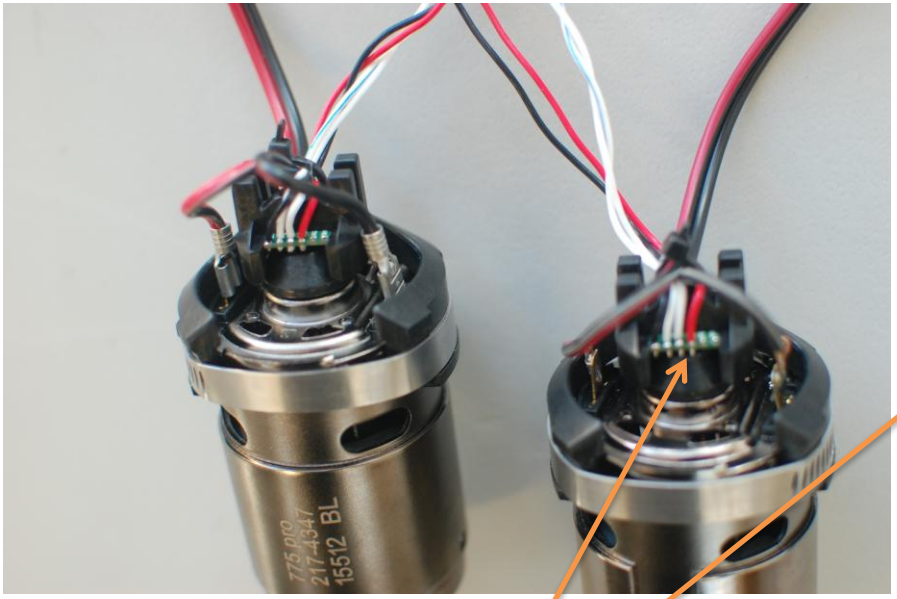
To Talon SRX

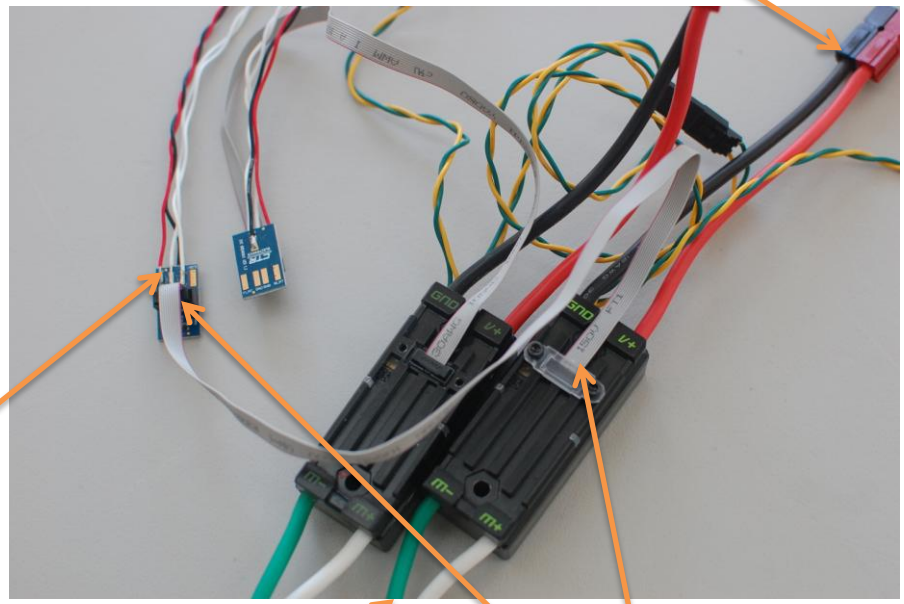


To Power Distribution Panel (PDP)

To RoboRIO

Connecting the Motors, Encoders, and Talon SRXs





To PDP

Solder power (5V), signal, ground

To motor

Attach ribbon cable

*EncoderHolder.java

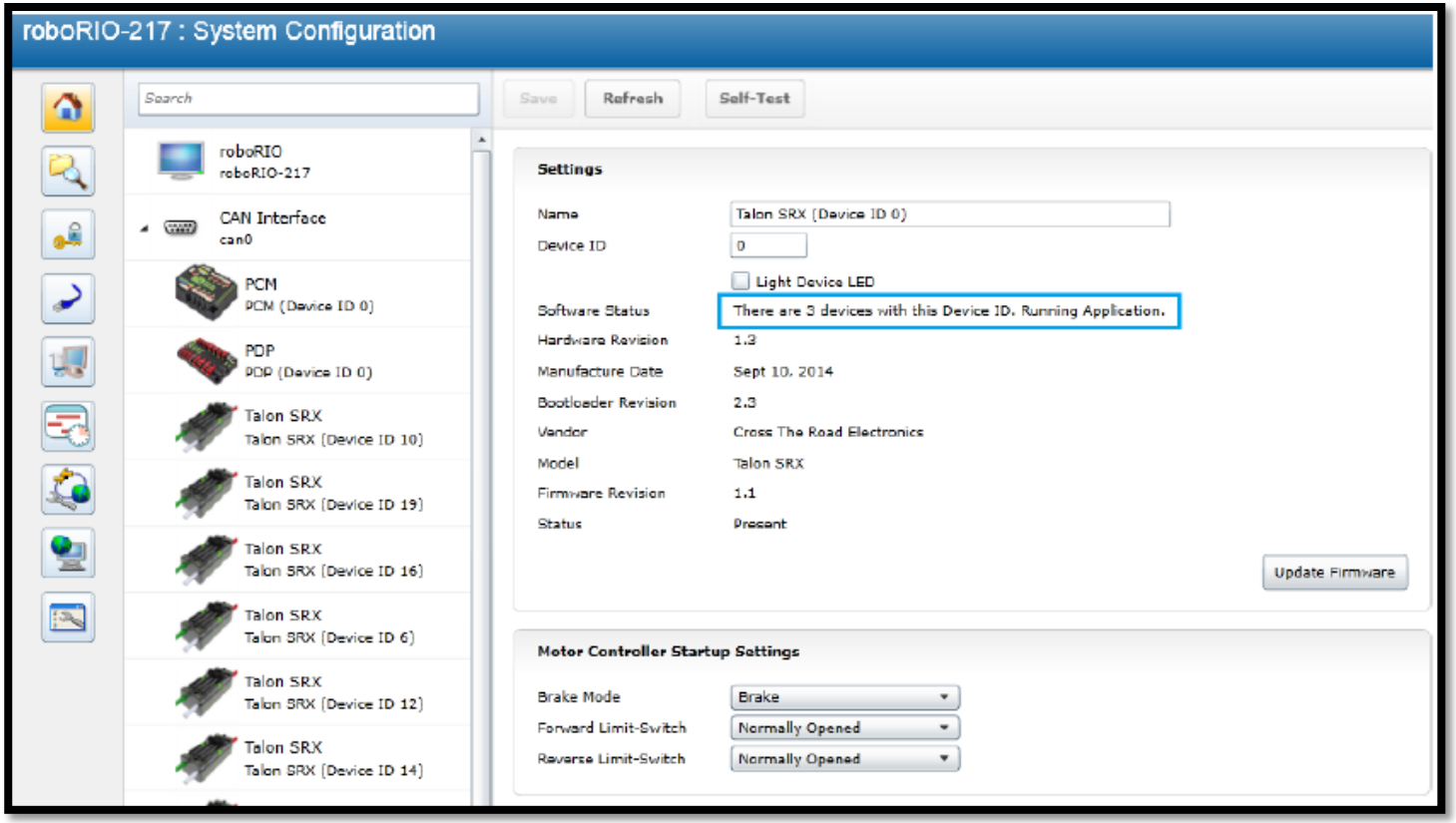
```java
package org.usfirst.frc.team5818.robot.subsystems;

import edu.wpi.first.wpilibj.CANTalon;

public class EncoderHolder extends Subsystem{

    public static final int CAN_ID = 5;
    private CANTalon TalonWithEncoder;
    private PIDController pid;

    //Constants for PID controller
    double kP = 1.2;
    double kI = 0.7;
    double kD = 0.3;

    public EncoderHolder(){
        //Instantiate CANTalon
        TalonWithEncoder= new CANTalon(CAN_ID);
        //Instantiate PID controller
        pid = new PIDController(kP, kI, kD, TalonWithEncoder, TalonWithEncoder);
    }

    public void runPID(double targetPos){
        pid.disable();
        pid.setSetpoint(targetPos);
        pid.enable();
    }

    public void setPower(double pow){
        pid.disable();
        TalonWithEncoder.set(pow);
    }

    public double getEncoderVal(){
        return TalonWithEncoder.getPosition();
    }

    protected void initDefaultCommand() {}

}
```

A simple Java class using a CANTalon and a PID controller. We'll explore the different sections of this class.

```
public EncoderHolder(){
    //Instantiate CANTalon
    TalonWithEncoder= new CANTalon(CAN_ID);
    //Instantiate PID controller
    pid = new PIDController(kP, kI, kD, TalonWithEncoder, TalonWithEncoder);
}
```

Instantiating a CANTalon:

- The CANTalon constructor takes an integer CAN ID
- Note that the CAN ID of your device is **NOT** its port on the PDP
- Finding and Configuring CAN IDs:
  - Enter **roborio-XXXX-frc.local** into a web browser, where XXXX is your team number
  - Use a browser with Silverlight installed (not Chrome)
  - You should see screen a that looks like this:



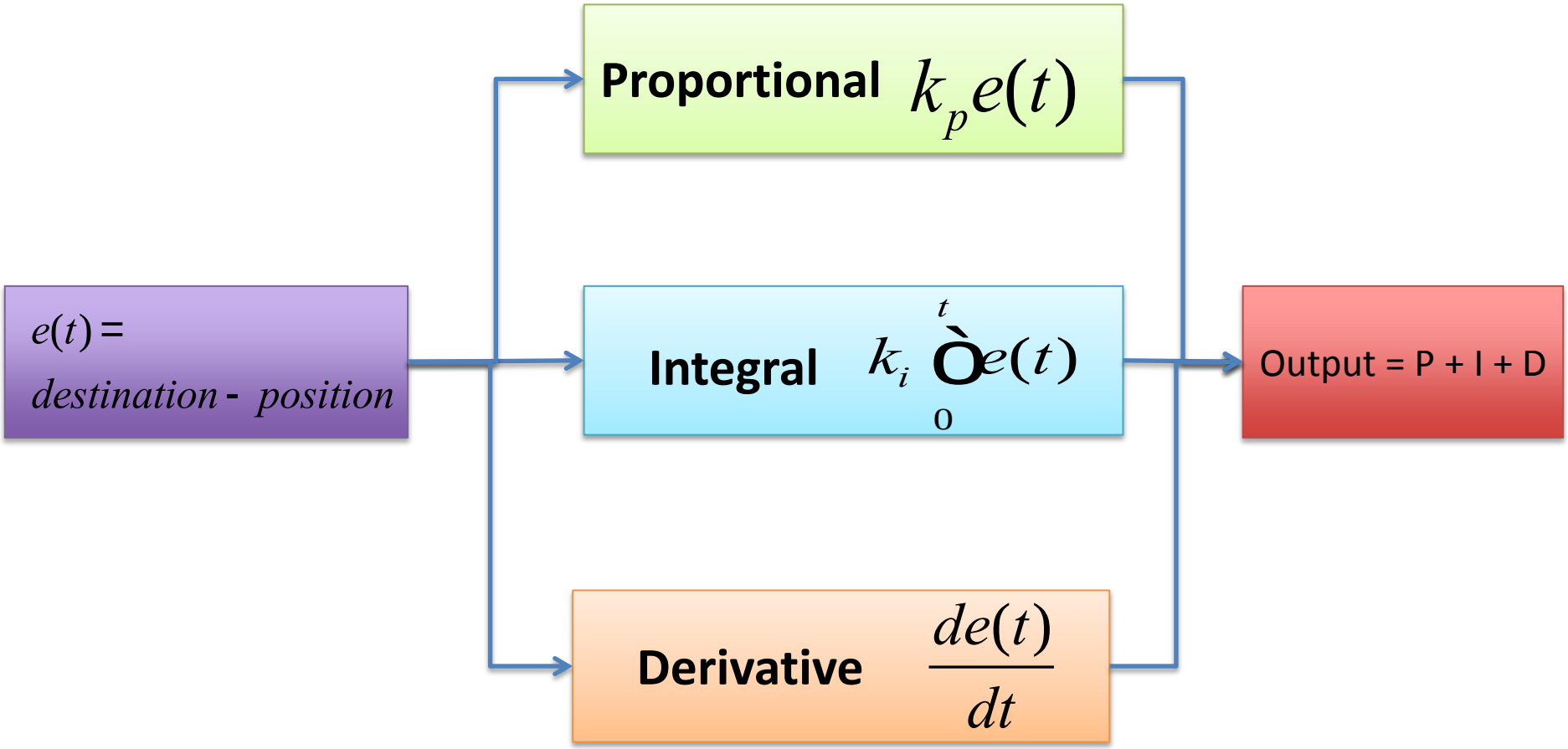
Configuring Talons:

- Click on a device on the left-hand side to configure it
- To identify which physical Talon you are configuring, check “Light device LED” and press “Update Firmware”
- Once you have selected the right Talon, you can change its ID in the “Device ID” field. This is the number passed to the CANTalon Constructor
- From this page, you can also choose whether the Talon is in “Brake Mode”, which means it will resist movement while stationary.

```
31      public void setPower(double pow){
32          pid.disable();
33          TalonWithEncoder.set(pow);
34      }
35
36      public double getEncoderVal(){
37          return TalonWithEncoder.getPosition();
38      }
```

Writing and Reading Values with a CANTalon:

- CANTalons have two simple methods for reading and writing values
- The **set** method takes a power value between -1 and 1 and writes it to the motor
- The **getPosition** method returns the position of the encoder in encoder ticks. Be sure to add a scale and offset to convert it usable units.



PID Controllers:

- A PID controller allows a machine to reach a desired state without overshooting, undershooting, or oscillating
- In a PID controller the "error function" is the distance to the target state
- The Proportional (P) term is directly proportional to the current error
- The Integral (I) term is proportional to the total accumulated error
- The Derivative (D) term is proportional to the current change in error

```
12
13      //Constants for PID controller
14      double kP = 1.2;
15      double kI = 0.7;
16      double kD = 0.3;
17
18      public EncoderHolder(){
19          //Instantiate CANTalon
20          TalonWithEncoder= new CANTalon(CAN_ID);
21          //Instantiate PID controller
22          pid = new PIDController(kP, kI, kD, TalonWithEncoder, TalonWithEncoder);
23      }
```

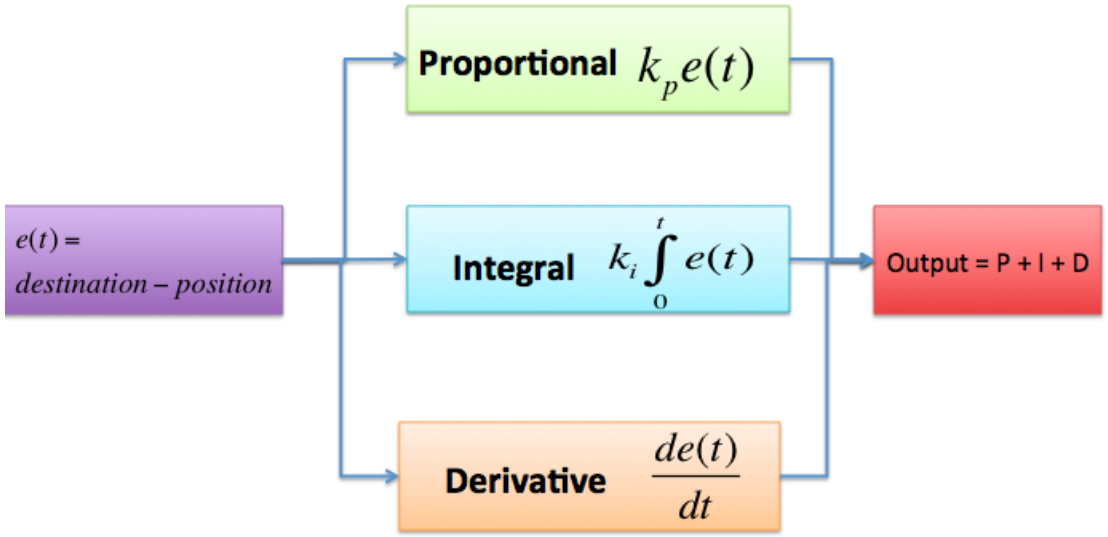
Instantiating a PID Controller:

- The constructor for the PIDController class takes 5 arguments:
  - The K_P, K_I, and K_D Constants
  - A PIDSource from which to get sensor information
  - A PIDOutput that can move the system closer to the target
- A CANTalon in both a PIDSource and a PIDOutput, so it can be passed into either argument of the constructor
- If you want your PID loop to use multiple motors or sensors, create your own PIDSource or PIDOutput by implementing the PIDSource or PIDOutput interface.

```
25⊖     public void runPID(double targetPos){
26          pid.disable();
27          pid.setSetpoint(targetPos);
28          pid.enable();
29      }
```

Running a PID loop:

- Whenever you want to change a parameter of the PIDController or set a power value, it is wise to stop the current PID loop using the **disable** method
- To set a target position for your PIDController, use the **setSetpoint** method. This position must be in the same units as those given by your PIDSource.
- To start the PID loop running, us the **enable** method. After calling this method, the PID loop will keep running in the background until it is told to stop.




```
13      //Constants for PID controller
14      double kP = 1.2;
15      double kI = 0.7;
16      double kD = 0.3;
17
```

Tuning the PID constants:

- For good performance, it is important to choose good values for your K_p, K_i, and K_d constants
- The following is a simple method for loop tuning:
  - Set all the values to 0.
  - Gradually raise K_p until the system moves in the right direction and slightly undershoots
  - Then raise K_i until the system begins to oscillate slightly around the target
  - Finally, raise K_D until the oscillation dampens
- For more information on PID tuning, see : https://en.wikipedia.org/wiki/PID_controller#Loop_tuning