Sensors & Motors

Misha G. and Shashank P.

OUTLINE

- DIO Sensors
- Analog Sensors
- PWM
- CAN Devices
- Motors

WHAT IS A SENSOR?

- A sensor is a device that picks up on information within its environment and converts that to an electrical signal to use create some sort of output
  - Several types: ex. Distance, rotation, temperature, force, etc.
- Applications
  - Subsystems: ex. Intake, Elevators, Shooter

ROBORIO

DIO

PWM

Analog

DIO SENSORS

WHAT IS DIO?

- Stands for “Digital Input/Output”
- Connects to the DIO ports on the RIO
- In most cases, digital signal is the voltage in a wire and it is a binary value, either 0 or 1
    - Measured with the 5V difference between power & ground
- Sensors with two states are almost always digital sensors
- 10 DIO ports on the RIO
    - S → signal (white)
    - V → power (red)
    - ⏚ → ground (black)
- Don’t connect your power to ground or vice versa (power light on RIO will turn red)

SWITCHES & BUTTONS

- Buttons and switches are either pressed or not pressed
- Can be used for physical constraints or to detect game pieces

PHOTOGATE/BEAM BREAK

- Sensor with a laser that can be broken
- Two states
- Some example use cases
  - Check height of an elevator
  - See if object has passed through some opening
  - Check if game piece is loaded inside of shooter

PROGRAMMING DIGITAL INPUTS

```java
public class SensorSubsystem implements Subsystem {
  TalonSRX potMotor;
  CANSparkMax switchMotor;
  DigitalInput limitSwitch;
  DigitalInput photoGate;
  AnalogPotentiometer potentiometer;

  // Constructor for our subsytem. In this case its the sensorboard
  public SensorSubsystem(CANSparkMax switchMotor, TalonSRX potMotor, DigitalInput limitSwitch,
    this.switchMotor = switchMotor;
    this.potMotor = potMotor;
    this.limitSwitch = limitSwitch;
    this.photoGate = photoGate;
    this.potentiometer = potentiometer;
  }

// Returns the value of the Photogate using the method .get()
public boolean getPhotoGateValue() {
  return photoGate.get();
}
```

Subsystem:

- Define a DigitalInput and pass it through constructor
- Create getPhotoGateValue() method to return the boolean value
    - Boolean from photoGate.get()

PROGRAMMING DIGITAL INPUTS

Command:

- Call the subsystem methods inside of the command
- Use the boolean values for logic

Constants:

- Put the DIO ID of the photoswitch

Robot Container:

- Define the subsystems and commands

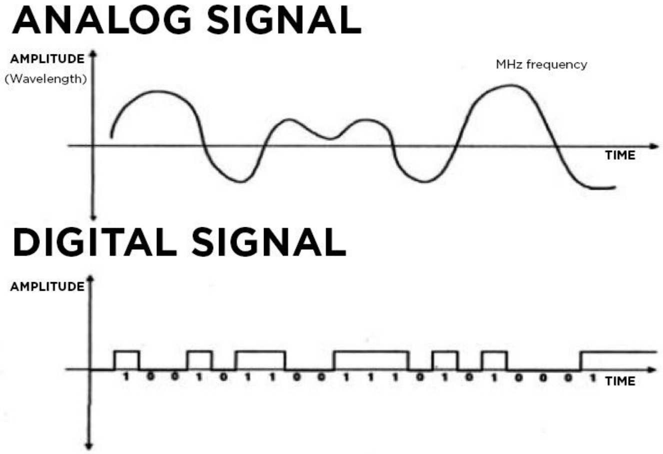
```
@Override
public void execute() {
    if (sensorBoard.getSwitchValue()) {
        sensorBoard.setSwitchMotor(speed:0.3);
    } else {
        sensorBoard.setSwitchMotor(speed:0);
    }

    if (sensorBoard.getPhotoGateValue()) {
        sensorBoard.setPhotoMotor
        (sensorBoard.getPotentiometerValue());
    } else {
        sensorBoard.setPhotoMotor(speed:0);
    }
}
```

ANALOG SENSORS

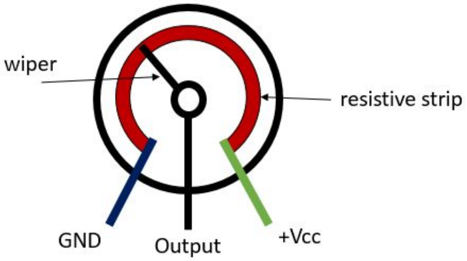
WHAT ARE ANALOG INPUTS?

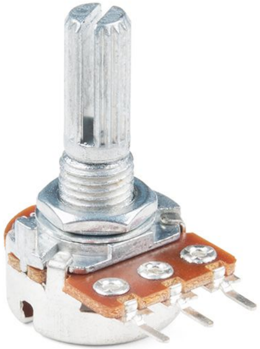
- Used for sensors with values that vary over a range
  - Values range from 0 to 255
- Signal is continuous, but provides different values based on sensor readings
- 4 analog ports on the RIO



POTENTIOMETERS

- Variable resistor
  - Output voltage changes
  - Resistor → component that regulates flow of electrical current
  - Turning the knob changes the resistance between outer two connections
  - Middle connection = “wiper”
- Another way to measure rotation on a robot
- Example usage: adjusting the speed of a motor





PROGRAMMING POTENTIOMETERS

```java
public class SensorSubsystem implements Subsystem {
  TalonSRX potMotor;
  CANSparkMax switchMotor;
  DigitalInput limitSwitch;
  DigitalInput photoGate;
  AnalogPotentiometer potentiometer;

  // Constructor for our subsytem. In this case its the sensorboard
  public SensorSubsystem(CANSparkMax switchMotor, TalonSRX potMotor, DigitalInput limitSwitch,
    this.switchMotor = switchMotor;
    this.potMotor = potMotor;
    this.limitSwitch = limitSwitch;
    this.photoGate = photoGate;
    this.potentiometer = potentiometer;
  }
```

Subsystem:

- Construct AnalogPotentiometer
- A double value is returned through getPotentiometerValue() method
  - Uses potentiometer.get()

PROGRAMMING POTENTIOMETERS

```java
@Override
public void execute() {
    if (sensorBoard.getSwitchValue()) {
        sensorBoard.setSwitchMotor(speed:0.3);
    } else {
        sensorBoard.setSwitchMotor(speed:0);
    }

    if (sensorBoard.getPhotoGateValue()) {
        sensorBoard.setPhotoMotor
        (sensorBoard.getPotentiometerValue());
    } else {
        sensorBoard.setPhotoMotor(speed:0);
    }
}
```

Command:

- Call the subsystem methods inside of the command
- Use the double value for logic

Constants:

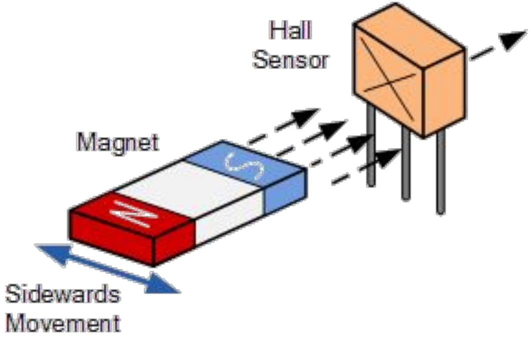
- Put the Analog ID of the potentiometer

Robot Container:

- Define subsystem and command

HALL EFFECT SENSOR

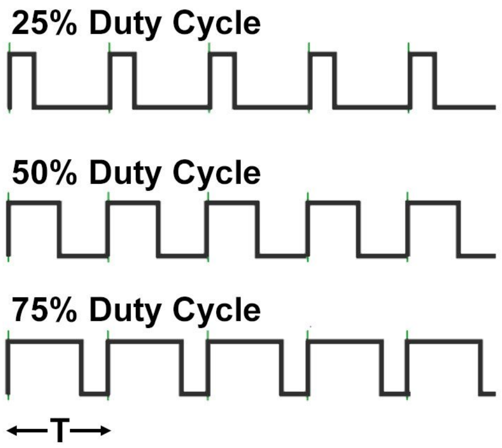
- Similar to a switch but uses the Hall effect to detect magnetism
- Does not require physical touch, can detect through non-magnetic objects
- Can be digital or analog
    - ThriftyBot Hall effect sensor is digital
    - CANCoder is analog



PWM

PWM - Pulse Width Modulation

- This type of data transfer is an output on the RoboRIO
- The RoboRIO cycles the signal line on and off at varying times
- Pulsing power supply on and off at a fixed voltage
- Ex. LED → brightness
- Ex. Motors → speed
- Duty Cycle: current is flowing through the circuit a % of the time
- Ex. if 50% then on time = off time
- Used to get an analog output



SERVO

- Servos use the PWM signal as an angle
- 100% duty cycle is maximum angle
- 0% duty cycle is its minimum angle
- Used for small movements
  - Pushing a basic locking mechanism into place

PWM MOTOR CONTROLLERS

- Motor controllers take the input signal and scale the range
  - RoboRIO Motor
  - 0V to +5V => -12V to +12V
- Common PWM motor controllers include the REV SPARK Max and SPARK Flex

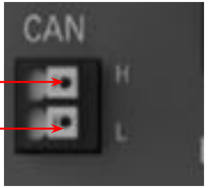
CAN DEVICES

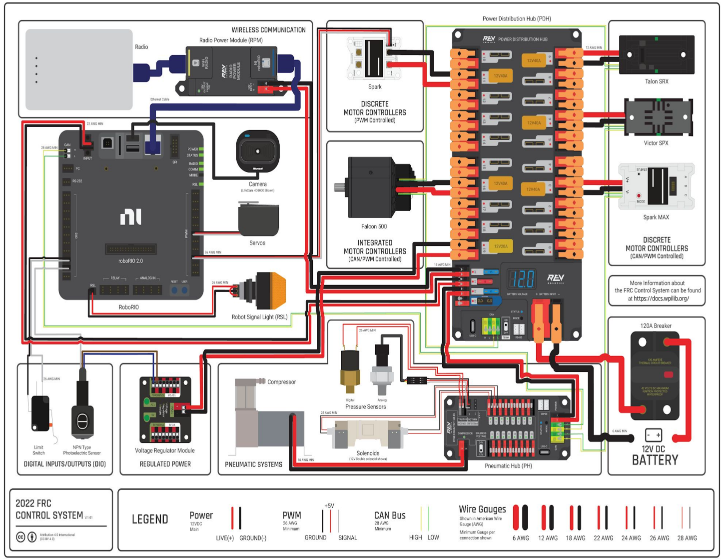
WHAT IS CAN?

A CAN Bus is a chain that allows devices across your control system to communicate with each other

High (Yellow)

Low (Green)





Devices have to be **ID'd** in order to used them

PROS AND CONS OF CAN

PROS

- Modern system, many devices switching to it
- Reduces amount of wiring needed to connect multiple devices
- Communication is quick

CONS

- Networks are fragile, one break can destroy the entire network
- Device firmware and CAN IDs need to be set up properly
- CTRE and REV CAN buses can be incompatible (CAN 2.0 vs CAN FD)

CAN SENSORS

- CAN allows a lot of data to be transmitted to and from multiple devices while minimizing wires
- Examples of CAN sensors:
  - Rotary encoders like the WCP ThroughBore encoder or the CANcoder
  - IMUs like the Pigeon 2.0
  - Distance sensors like the CANrange
- Digital sensors can be connected to the CAN bus through devices like the CTRE CANdi

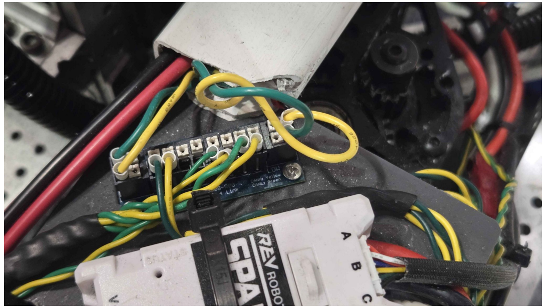
CAN TOPOLOGY COMPARISON

Daisy Chain

- Devices connected in a line
- Most common method
- Recommended by part makers like REV, CTRE and WCP
- If one device fails, can bring down the entire network

Example: CAN Star device for branching

1676 Pascack Pi-oneers



Star

- Devices connected in branches
- Much less common
- Not recommended by part manufacturers due to instabilities and issues
- If one device fails, network can still work with other devices

MOTORS

MOTORS

- Motors provide powerful, instant rotational movement
- Powers everything from shooters to swerve drive
- Most common motors include Kraken x44/x60 and Neo 1/2/Vortex
- Nowadays, most motors are brushless

COMPARING MOTORS

SPEED	TORQUE	PRICE
1. Kraken X44 (7530 RPM) 2. NEO Vortex (6784 RPM) 3. Kraken X60 (6000 RPM) 4. NEO 2.0 (5676 RPM)	1. Kraken X60 (7.09 Nm) 2. Kraken X44 (4.05 Nm) 3. NEO 2.0 (3.75 Nm) 4. NEO Vortex (3.6 Nm)	1. Kraken X44 & Kraken X60 ($218) 2. NEO Vortex ($90) 3. NEO 2.0 ($55)

PROGRAMMING MOTORS

```java
public class SensorSubsystem implements Subsystem {
  TalonSRX potMotor;
  CANSparkMax switchMotor;
  DigitalInput limitSwitch;
  DigitalInput photoGate;
  AnalogPotentiometer potentiometer;

  // Constructor for our subsytem. In this case its the sensorboard
  public SensorSubsystem(CANSparkMax switchMotor, TalonSRX potMotor, DigitalInput limitSwitch,
    this.switchMotor = switchMotor;
    this.potMotor = potMotor;
    this.limitSwitch = limitSwitch;
    this.photoGate = photoGate;
    this.potentiometer = potentiometer;
  }

 // Sets the motors value using the CANSparkMax method .set(double speed)
 public void setSwitchMotor(double val) {
   switchMotor.set(val);
 }
```

Subsystem:

- Construct CANSparkMax Object
- Create method for setting the motor speed
  - Motor speed is set as a percent duty cycle
  - Between -1.0 and 1.0

PROGRAMMING MOTORS

```java
@Override
public void execute() {
    if (sensorBoard.getSwitchValue()) {
        sensorBoard.setSwitchMotor(speed:0.3);
    } else {
        sensorBoard.setSwitchMotor(speed:0);
    }

    if (sensorBoard.getPhotoGateValue()) {
        sensorBoard.setPhotoMotor
        (sensorBoard.getPotentiometerValue());
    } else {
        sensorBoard.setPhotoMotor(speed:0);
    }
}
```

Command:

- Define subsystem methods inside the command
- Use sensor logic to determine when/how to run motor

Constants:

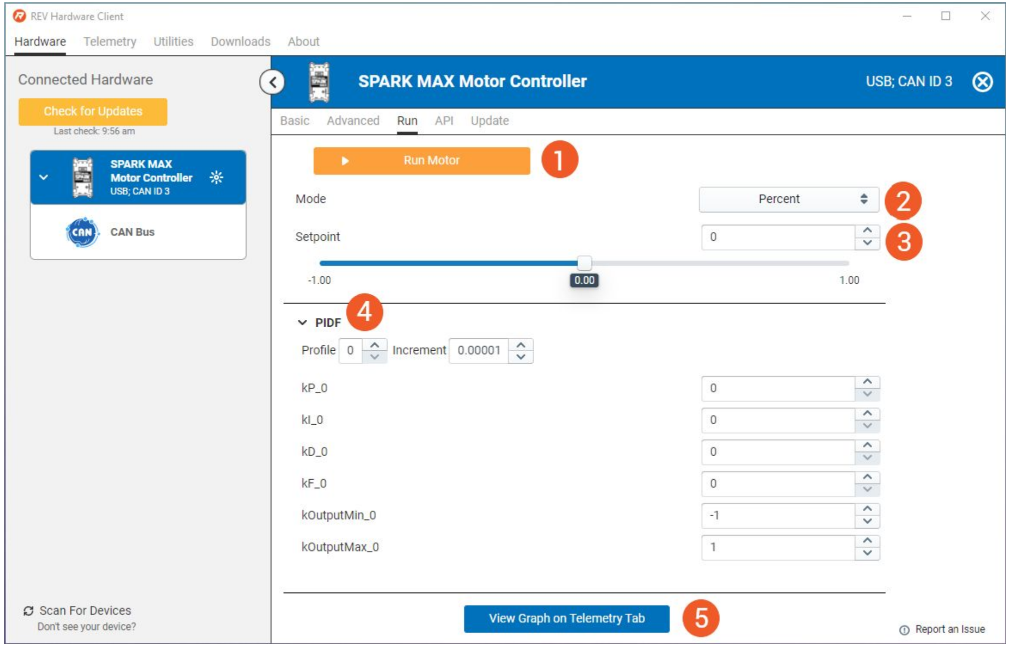
- Define the CAN ID of the motor
    - CAN ID found in REV Hardware Client or Phoenix Tuner, depending on motor controller type

Robot Container:

- Define the subsystem and command

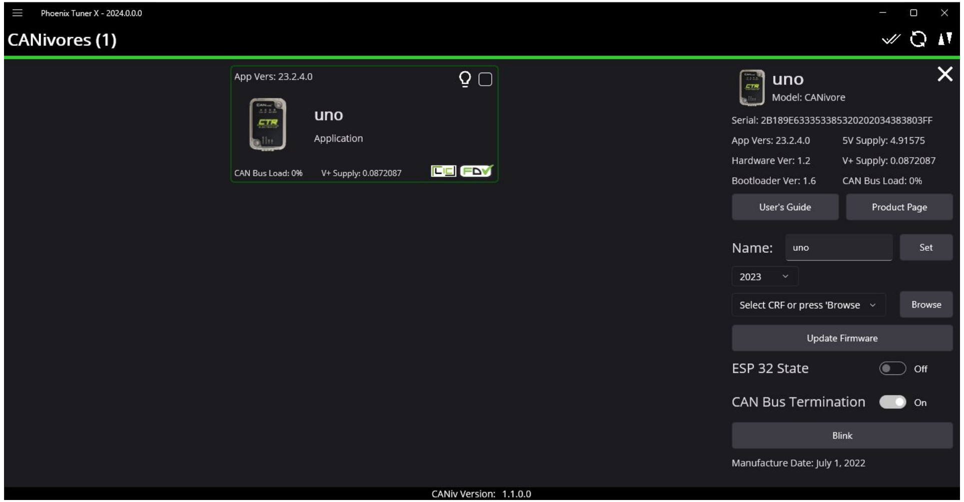
USING REV HARDWARE CLIENT

- The Rev Client can
  - Update Software
  - Set ID for motor controllers
  - Run motors
- Using included USB-C wire, connect to any SPARK Max that is on the CAN Bus
- If robot is on, all controllers is detected
- If robot is off, only 1 controller is detected



USING PHOENIX TUNER X

- Using Phoenix Tuner (X) you can
  - Update Software
  - Set ID’s of CTRE sensors/motor controllers
  - Test motors and sensors
- Connect to the RoboRIO
- All the devices should appear on the CAN Devices tab
- Can configure specific devices individually

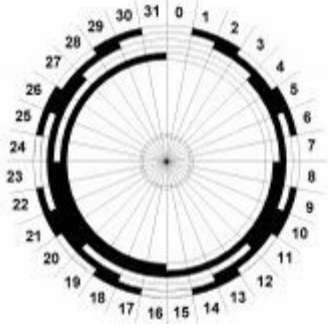


WHAT IS AN ENCODER?

- Communication device that controls the motion of an operating device
    - Helps determine speed/position of a motor
- Encoders convert motion to an electrical signal that can be read and interpreted by the RoboRIO
    - The encoder sends a feedback signal that can be used to determine position, count, speed, or direction
- Ex. rotary encoder
    - Converts angle of a shaft to digital or analog code

TYPES OF ENCODERS

- Relative (Incremental)
  - Shines two lasers through a rotating disc
  - The number of times the beam is broken determines the amount of rotation which is added to a counter
  - Relative encoders are reset every time the robot is turned on
- Absolute
  - A code is read from the disk using the lasers to determine the position
  - Absolute encoders will know their even if the robot is powered off



Absolute

Incremental

PROGRAMMING ABSOLUTE ENCODERS

```
absEncoder = new DutyCycleEncoder(channel:2);
```

Subsystem:

- Construct the DutyCycleEncoder object
- Create a method to return the encoder position as a double

```
public double getPosition() {
    return absEncoder.get();
}
```

Command:

- Pass in the subsystem
- Access position using getPosition(), then utilize it to perform an action

```
@Override
public void execute() {
    if(subsystem.getPosition() > 100) {
        subsystem.stop();
    }
}
```

THANK YOU

https://cougarrobotics.com/

cougar1403@gmail.com

@team1403