# Advanced LabVIEW

http://workshop.frclabviewtutorials.com

# Building A Robust Autonomous

- If your robot:
  - Is easy to code for auto
  - Has the software architected so that auto coding is simple
  - Has driver's that are practiced

  You have a mountain of potential to do well.

TODO: fill this in a little more once we get the game for 2019

Notes:
Every year there has been a way to score points for your alliance by just moving

In the past, auto has been the first or second tie-breaker

A consistent auto is worth more than an if-y end game.

In 2016, (Stronghold) robots got 10 points for crossing a defense and 10 points for scoring in the high goal in auto. A lot of teams that have learned to prioritize Auto got these consistently.

Let's say for the sake of argument that they were only 90% consistent. 90% x 20 points is an expected 18 points per match.

On the other hand, that year a team could climb in the end game for 15 points. A lot of robots had difficulty getting through the match, onto the tower grounds, and climbing. But even if they were 100% consistent at this, that would only be an expected 15 points per match (which is less than 18).

You might say that's not a lot less, and it's not. But in putting in the effort to have a consistent auto and trained drivers, you can be prepared to widen that gap.

Don't get me wrong, top teams will do both and have better than 90% success at it, but in my experience, a lot of teams that could be competitive are not prioritizing auto enough to compete with the stronger teams.

# Ingredients

- Quick, effective mechanisms
- Easy to edit
- Use sensors (closed loop control)
- Appropriate sensors to help your robot account for inconsistencies in field, setup, game pieces, battery level, etc.

Not going to address mechanisms here, this presentation is focused on the software side of things, but I did feel like we should acknowledge that dependency.

For easy to edit, I'm going to present a way to do a file based autonomous so that you don't have to redeploy every time you tweak it.

As well as several example sensors and possible ways that they could be beneficial.

File Based Auto

# EASY TO EDIT

# File Based Auto

- https://www.frclabviewtutorials.com/tutorials/filedAuto/
- Objective: Setup a system where I can quickly tweak a setting and re-run auto and see the difference.

# File Based Auto

- Demo: creating AutoMove control

Open project and right click on {Target …} and select New -> Control

We will want to add multiple fields to this control, so first add a cluster

Then to that cluster add a distance angle and time numeric controls.

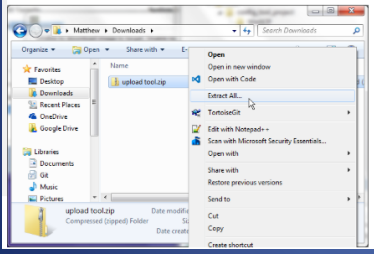
# File Based Auto

- Get the editing/deploying tool

Getting the editing tool

Download autoProfiling.zip

Extract the files to where you want to store them. This should be with the rest of your source because EditMoves.vi uses its location to determine where to store the xml file (which you should include in your source control) and where to get the FTP (File Transfer Protocol) program it uses (WinSCP).



# File Based Auto

- Demo: Add EditMoves.vi to my project under “My Computer” and recallMovesOnRobot.vi under “Team Code” and use in auto

Back in the project explorer, expand and then right click on “My Computer” and select “Add” -> “File . . .”
This will open a file dialogue. Use it to select the EditMoves.vi that you just extracted.

By putting this vi under “My Computer”, you are telling LabVIEW that you want this vi to run locally.
EditMoves.vi is a utility VI that allows for you to put in your own definition of what an autonomous move looks like and build an array of such moves for an auton program. EditMoves will extract xml from the array and store it locally (when save is hit) and put it on the robot for recallMovesOnRobot.vi to load when your autonomous code calls it.
Also add recallMovesOnRobot.vi under the Target (maybe even under “Support Code” or under “Team Code” - it’s up to you).

# Ingredients

- (NA) Quick, effective mechanisms
- (✓) Easy to edit
- Use sensors (closed loop control)
- Appropriate sensors to help your robot account for inconsistencies in field, setup, game pieces, battery level, etc.

Next

Closed loop control through PID

# PID

To close the feedback loop means to bring information from the output of an action back as an input.

# Closed Loop Control

- Open Loop:

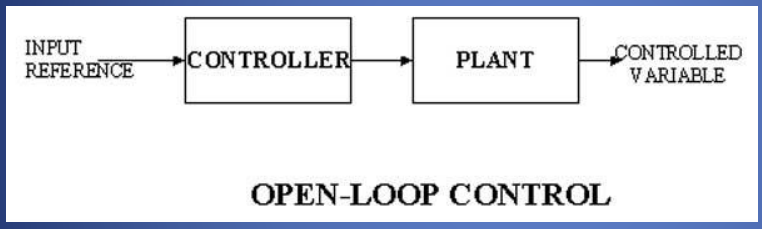



Image from:
https://www.google.com/url?sa=i&source=imgres&cd=&cad=rja&uact=8&ved=2ahUKEwiJubyf2sXfAhVMaq0KHXaFApAQjxx6BAgBEAI&url=http%3A%2F%2Finterviewquestionanswer.com%2Felectrical-engineering%2Fwhat-are-different-types-of-control-systems&psig=AOvVaw0I1yfTz9-ETIWCxb3uyuvJ&ust=1546195772100247

# Closed Loop Control

- Open Loop
- Closed Loop

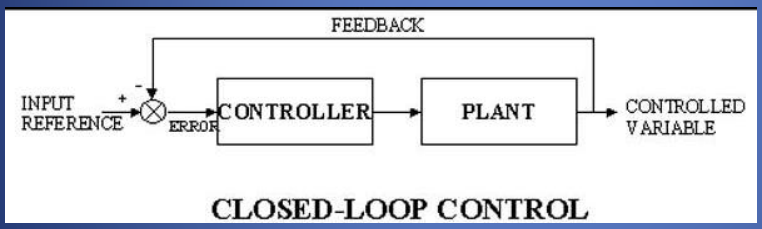



Image from:
https://www.google.com/url?sa=i&source=imgres&cd=&cad=rja&uact=8&ved=2ahUKEwiJubyf2sXfAhVMaq0KHXaFApAQjxx6BAgBEAI&url=http%3A%2F%2Finterviewquestionanswer.com%2Felectrical-engineering%2Fwhat-are-different-types-of-control-systems&psig=AOvVaw0I1yfTz9-ETIWCxb3uyuvJ&ust=1546195772100247

# Closed Loop Control

- Open Loop
- Closed Loop
  - Example

Example:

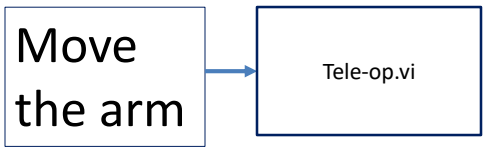
# Closed Loop Control

- Open Loop
- Closed Loop
  - Example


I want to move the arm

# Closed Loop Control

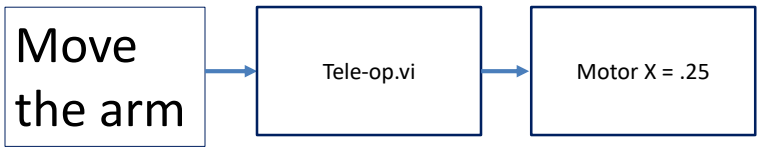
- Open Loop
- Closed Loop
  - Example




So, I tell Teleop to move the arm

# Closed Loop Control

- Open Loop
- Closed Loop
  - Example



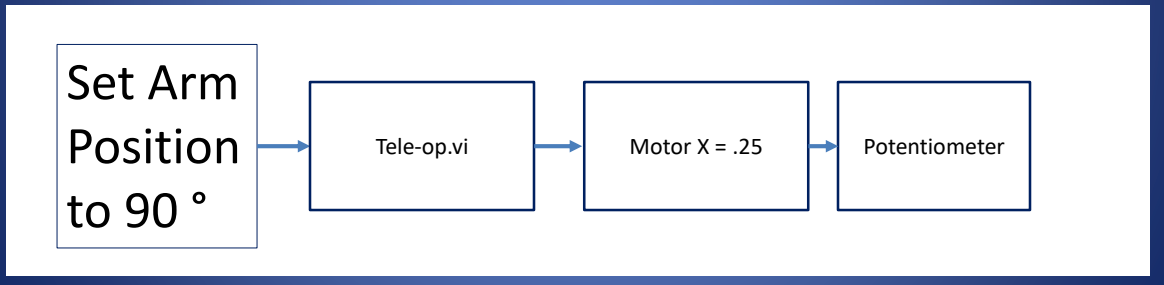

It tells the motor (motor X) to set to 25%

Did that move the arm at all? Did it move to the position I wanted ??

**I don't know at this point**

# Closed Loop Control

- Open Loop
- Closed Loop
  - Example

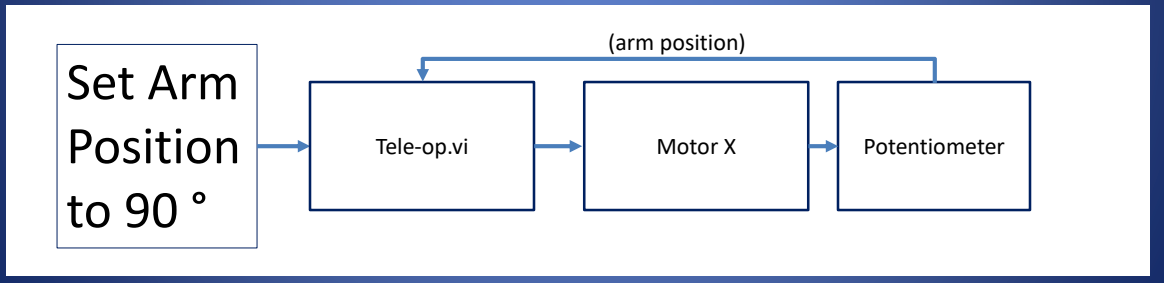



So, let's put a potentiometer on the arm and change our command to move it to 90 °

Now, Teleop still says to set the motor power to 25%, but the motor will affect the potentiometer. We can then read that potentiometer in Tele-op to know what the arm did.

# Closed Loop Control

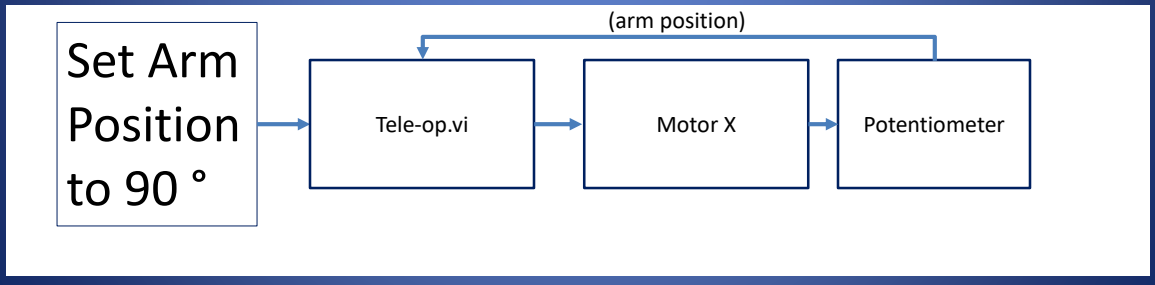
- Open Loop
- Closed Loop
  - Example




Which we would diagram something like this (notice that I removed the value the motor is getting set to because this is now a continuous loop of feedback and the motor value may change over time).

# Closed Loop Control - PID

- PID stand for:
  - Proportional
  - Integral
  - Derivative




PID is a methodology of closed loop feedback/control that allows for fast response times.

# Closed Loop Control - PID

- PID stand for:
  - Proportional
  - Integral
  - Derivative

$$\text{Output} = K_p E(t) + K_i \int E'(t) + K_d E'(t)$$

From Wikipedia: A PID controller continuously calculates an *error value* e(t) as the difference between a desired setpoint (SP) and a measured process variable (PV) and applies a correction based on proportional, integral, and derivative terms (denoted *P*, *I*, and *D* respectively), hence the name.

# PID

- Proportional

https://en.wikipedia.org/wiki/PID_controller
https://www.youtube.com/watch?v=JEpWlTl95Tw
https://www.youtube.com/watch?v=UR0hOmjaHp0
http://robotics.stackexchange.com/questions/167/what-are-good-strategies-for-tuning-pid-loops

# PID

- Proportional
  - Constant multiplied by error (offset)
  - The larger this is, the faster the robot approaches the setpoint (smaller rise time)
  - If too large, the robot will overshoot the target consistently

The proportional

# PID

- Proportional
  - Constant multiplied by error (offset)
  - The larger this is, the faster the robot approaches the setpoint (smaller rise time)
  - If too large, the robot will overshoot the target consistently
- Integral
  - Constant multiplied by integral of all previous error values
  - The larger this is, the less overshoot and settling time (less bounce)
  - If too large, the robot will eventually react to any error violently

Note: a small error long enough eventually produces and integral of significant value (exponentially approaching infinity), causing the robot to jump into action and overshoot.

# PID

- Proportional
  - Constant multiplied by error (offset)
  - The larger this is, the faster the robot approaches the setpoint (smaller rise time)
- Integral
  - Constant multiplied by integral of all previous error values
  - Used to eliminate steady state error (reducing offset after movement)
- Differential
  - The larger this is, the less overshoot and settling time (less bounce)

# PID

- Tuning

# PID

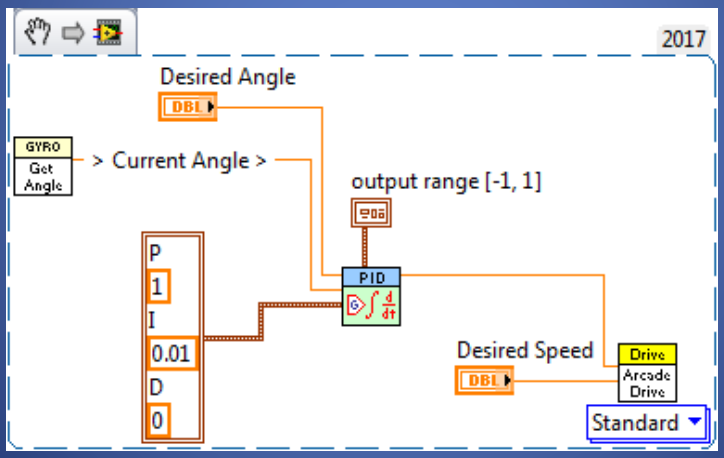
- Tuning
  - Several methods available
    - **Ziegler–Nichols***
    - **Tyreus Luyben**
    - **Cohen–Coon**
    - **Åström-Hägglund**
    - **Manual Tuning***

http://faculty.mercer.edu/jenkins_he/documents/TuningforPIDControllers.pdf#page=6
https://www.youtube.com/watch?v=JEpWlTl95Tw
https://www.youtube.com/watch?v=UR0hOmjaHp0
http://robotics.stackexchange.com/questions/167/what-are-good-strategies-for-tuning-pid-loops
Ziegler-Nichols: http://robotsforroboticists.com/pid-control/
Manual (page 16):
https://docs.google.com/viewer?a=v&pid=sites&srcid=aGFyZGluZy5lZHV8dGVhbS0zOTM3fGd4OjUyNzdiNzRkNjkxNjA3MGM

http://www.ni.com/white-paper/3782/en/

# PID

- Example code




Read the desired angle (a control), and the current angle (from the gyro), bound the output to [-1, 1], us 1 as the the proportional constant and .01 as the integral constant – use the result as the steering on Arcade Drive (in effect, a drive straight).

# Ingredients

- (NA) Quick, effective mechanisms
- (✓) Easy to edit
- (✓) Use sensors (closed loop control)
- Appropriate sensors to help your robot account for inconsistencies in field, setup, game pieces, battery level, etc.

Next, some examples of effective sensors to use.

# SENSORS

# Sensors - Encoder

# Sensors - Encoder

- Places to use encoders:
  - When trying to measure rotational speed
  - Trying to measure rotational distances possibly greater than 8 rotations.
  - Don’t care about starting position

# Sensors - Encoder

- Places to use encoders:
  - When trying to measure rotation speed
  - Trying to measure rotational distances possibly greater than 8 rotations.
  - Don't care about starting position
- Examples:
  - Drive train
  - Fly Wheel/wheeled shooter

Drive-train

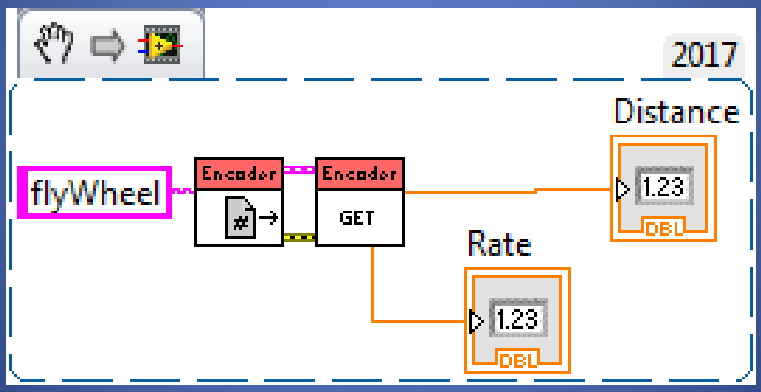
- likely to travel more than 10 rotations in a given match
- Don't care where the wheels are when starting, and even better – the robots starting position is 0 (good)

Fly Wheel

- Don't care what position it starts in
- Likely to rotate more than 10 rotations in a match
- Care is primarily about speed

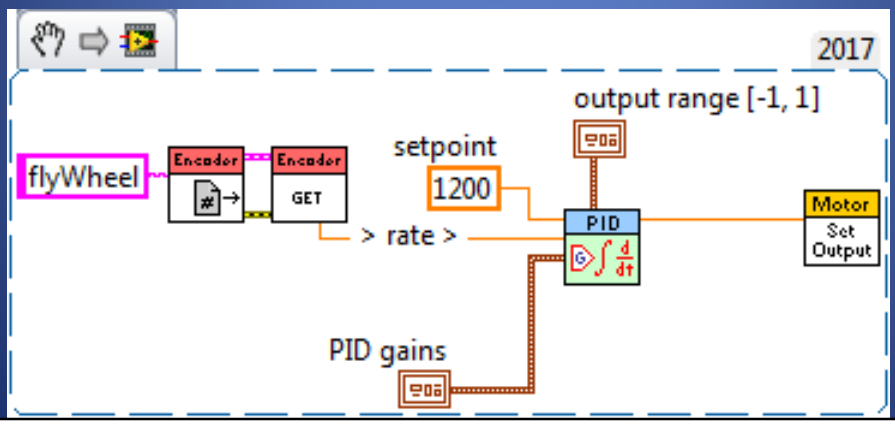
# Sensors - Encoder

- Reading



# Sensors - Encoder

- Control



# Sensors - Potentiometer

TODO: Code

# Sensors - Potentiometer

- Places to use potentiometers:
  - Trying to measure rotational distances less than 8 rotations.
  - Care about starting position – or absolute positions
- Examples:
  - Arm angles
  - Elevator positions

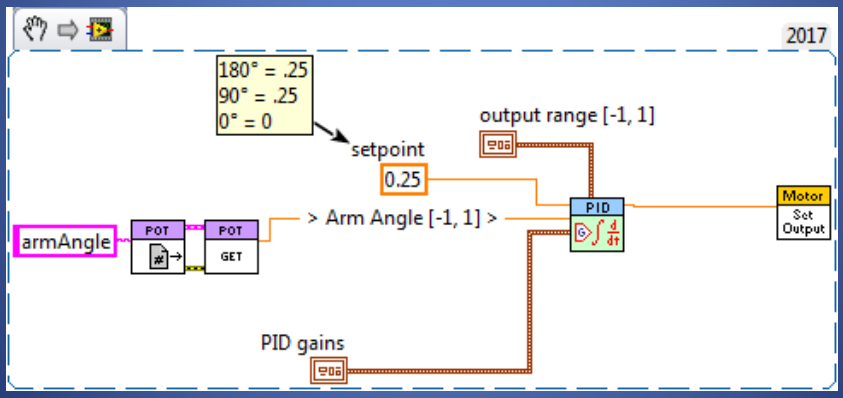
Arm angles:

- Care about absolute position (0 should always mean on the ground regardless of where the arm was when we turn it on).

Same for elevators and similar manipulators

# Sensors - Potentiometer

- Control



# Sensors - Potentiometer

- Note:
  - Easy way to make potentiometer relative to a known point:

https://www.frclabviewtutorials.com/tutorials/sensors/roborio/potentiometer/

One disadvantage that potentiometers have when compared to encoders is now you care very much about the position of the potentiometer when mounting/unmounting it.

I'm not going to go into this today, but there is a documented, simple way to apply a known offset when reading potentiometers – you would then update this offset after doing modifications.

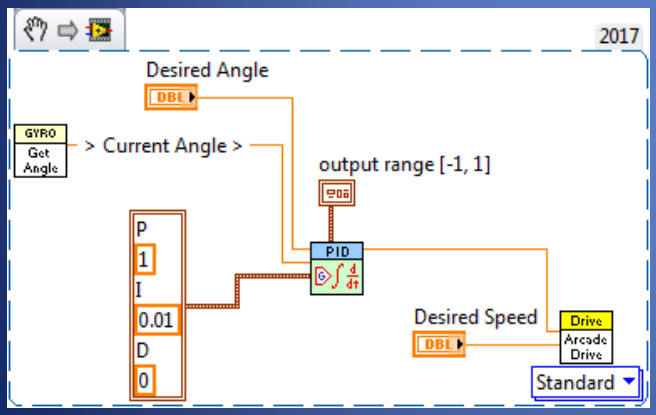
# Sensors - Gyro

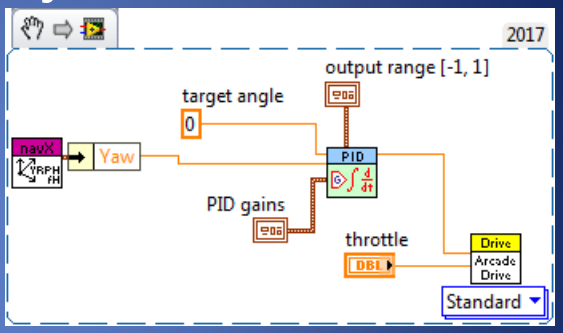
# Sensors - Gyro

- Places to use a Gyro:
  - When trying to drive perfectly straight
  - When trying to turn to specific angles (especially in auto)

# Sensors - Gyro

- Control



# Sensors - Other

- https://www.frclabviewtutorials.com/tutorials/sensors/dashboard/arduino/

# Demo

# Questions