HOWDY BOTS

FRC Team 6377

Don't Break Your Bot

A Guide for Testing Large and Delicate Mechanisms

Table of Contents

Motivation

Control is a big part of robotics, and controlling high-power, robust mechanisms can be a challenging step for any team. For us at 6377, safety when programming our robot is a primary concern for students, mentors, and the robot itself. Therefore, we have implemented tools and processes that enable us to test and iterate over a solid foundation, knowing the system will behave as expected in a consistent way, even before plugging in the motors.

Therefore, in 2018 we adopted a CTRE-based hardware and software configuration (CTRE stands for Cross The Road Electronics, LLC.), in which an ecosystem of Talon SRX and Victor SPX motor controllers play together with our NI LabVIEW-based code to provide precise movement to our robot's systems. These motor controllers, which work directly from the roboRIO's CAN bus, include their own control loops. They allow off-loading of processing from the robot's main controller, keep the control loop within its 20ms timeframe, and can be configured on-the-fly as the code initializes and runs.

This whitepaper is designed to walk you through all of the steps of making your robot use Motion Magic, without "guess and check" that can lead to breaking of mechanisms very quickly. The process that Team 6377 has come to use with success is detailed below, split up into mechanical and programmatic, walking through mechanical considerations how to use Motion Magic mode, and how we approach tuning.

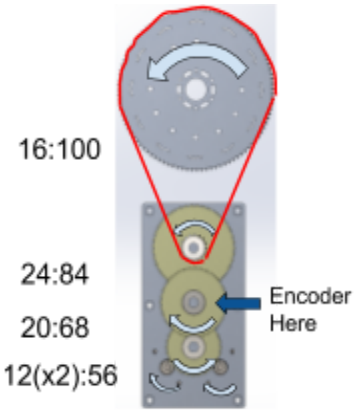
Understand your mechanism

Understanding your mechanism requires a knowledge of each one of its components. The core components of a mechanism usually entail the motor, gears, and the final output (e.g. wheels, arms, or elevators). Understanding the way these parts respond when force is applied, either intentional or not, is important to control the mechanism with high accuracy and precision.

Understanding the way that torque affects your mechanism and how it fails can be useful in determining its weak spots, as well as predicting how it will fail. Building in points of failure, like a chain that is weaker than the rest of the mechanism, can reduce overall damage, make the mechanism easier to fix and limit the amount of gearbox maintenance the team will have to invest when something goes wrong.

For example, our 2019 robot's gearbox consists of 2 mini-CIM motors connected to a custom 3-stage gearbox, with an output that is then further geared down by a chain system before moving the arm. Things to consider while using this gearbox are that, 1) every stage of the gearbox changes direction, 2) the chain system does not, and 3) the encoder is mounted on the second stage of the gearbox.

This knowledge of your system is necessary to prevent damage, especially in high-power systems.



This is the gear stack from our 2019 robot powered by 2 mini cim motors and used to help our robot climb to level 3 of the Hab.

Physical System

To better know your gearbox you can use a modified version of the JVN spreadsheet (Found here) to determine the gear ratios between your input and output shafts. This spreadsheet was first developed by a mentor from team 148 and has been extensively used by the *FIRST* Robotics Competition community. Our version has been modified to calculate P, F, and Motion Magic constants with only a few basic inputs.

To evaluate your physical system, consider the following questions:

- What direction does the motor turn when positive power is applied? Verify behavior by either looking up the specifications sheet or running a separate motor with a flag on the output shaft.

- What is the direction of your gearing mechanism? When positive power is applied to the motor, does the system move in the "positive" direction? In other words, does the motor and the mechanism move in the expected direction? If the system moves in the "negative" direction when positive power is applied, the *motor output* should be inverted.

- Where is your encoder on the gear stack? What direction does it turn? If the encoder moves in the "negative" direction when positive power is applied, the *sensor phase* should be inverted.
  If the encoder is closer to the motor, you can achieve higher encoder resolution, which makes feedback loops and PIDF tuning simpler. However, this approach can run into problems, such as hot motors melting encoders and not accounting for slop at the end of the mechanism. Fortunately, many of these problems are minimal to nonexistent in the majority of FRC applications. On the other hand, running encoders closer to the end effector can account for some slop, but the combination of that slop and lower encoder resolution can cause more problems than it solves when it comes to PIDF tuning. Although there are trade offs with encoder positioning, the howdy bots recommend that you place your encoder as close to the motor as is practical.

- Does your mechanism require the robot to be placed into a specific configuration to set the first value of your encoders during initialization? If so this should be taken into consideration when making testing plans.

- What are the physical limits of the system? What are the safety mechanisms to prevent the robot from breaking itself such as physical stops, limit switches, encoder positions?

Control System

Once you understand the physical characteristics of your robot's system, you need to determine how the mechanism should be controlled. Mechanisms can be controlled directly by the driver (e.g. using a joystick), or indirectly by a command which causes the system to move to a specified target. They also can be controlled by specifying motor output, velocity, or target positions. Calculate a preliminary set of values that will guide your exploration when tuning the mechanism. These values should align with the control algorithm you desire to implement, which will depend on its movement type and how it will be controlled by the code.

For CTRE devices, the options for control include percent Vbus, velocity PIDF, position PIDF, and Motion Magic. This paper will be covering the Motion Magic mode, though many concepts will transfer to other control modes.

Many of the control schemes deal with PIDF control, a control option that uses feedback from an external sensor and uses a PIDF loop. P stands for proportional, and applies more power the greater the error. I stands for Integral, and takes into account how long the mechanism has been away from the target, and adds more power the longer and greater the error. D stands for derivative and provides a damping affect the faster and closer the error decreases. F stands for feed forward, that is a calculated factor and is a starting point for the other factors to work off of. For more information on PIDF Refer to this paper by Tim Winters, (https://frc-pdr.readthedocs.io/en/latest/control/pid_control.html)

- Percent Vbus
    - This is the most basic control system using only the percent of maximum voltage to run to the motor from the motor controller. Useful for things such as intakes and quick tests, or anything that needs no feedback.

- Feedback control schemes.

  All of these control options use feedback from an external sensor and all of them use a PIDF loop.

  - Velocity PIDF

    This uses feedback from a sensor to drive to a specific velocity using a PIDF loop. The system will accelerate to the target as quickly as possible. Good for drivetrains and high precision shooters.
  - Position PIDF

    This uses feedback from a sensor to drive to a specific position using a PIDF loop. The system will accelerate to the target as quickly as possible. Good for small light mechanisms, but is outclassed by Motion Magic.
  - Motion Magic

    This generates a trapezoidal motion profile based on an acceleration and velocity constant, then uses a PIDF loop to stay on the profile. This is one of the best ways to control position in a system, but is more complicated than a position PIDF.

For clarification regarding these refer to the CTRE documentation[1].

Calculate Motion Magic constants

We compute the system control values based on the physical system rather than experimenting with the system. Experimentally determining these values requires time on the robot and in some systems, like our 2019 robot's arm, a P value that makes the system oscillate may cause damage to the system. Our modified JVN is an amalgamation of the original JVN that is able to calculate these control values. The spreadsheet includes input cells which describe the physical system and control system values such as desired rotational speed, and distance to P saturation. Output values are used to fill in values of F, P, Max Velocity, and Max Acceleration.

1. Use Modified JVN to calculate the theoretical maximum for Feed Forward(F). For a rotary mechanism, specify the desired rotational speed and for linear mechanisms, specify the desired linear speed. A good value for this is 60% to 80% of the theoretical maximum speed (loaded).

Rotary Mechanism

	Free Speed (RPM)	Stall Torque (N*m)	Stall Current (Amp)	Free Current (Amp)
Mini CIM	5840	1.41	89.00	3.00

# Motors per Gearbox	Gearbox Efficiency	Arm Load (lbs)	Arm Length (in)
3	65%	110	20

Driving Gear	Driven Gear		Arm Rotational Speed	Arm Time to move 90-degrees
12	56	No Load:	101.0 deg/s	0.89 sec
20	68	Loaded:	74.7 deg/s	1.21 sec
24	84			
16	100			
347.08 : 1	<-- Overall Ratio			

Current Draw per Motor (loaded)	Stall Load
17.56 amps	422.30 lbs

Controlled Rotational Speed (deg/sec)	Encoder (ticks/ Motor rev)	Desired P Saturation Distance (deg)	Desired time to get to controlled speed (sec)
60	4096	10	0.85

Feed Forward	P	Motion Magic Max Velocity	Motion Magic Acceleration
0.034698	0.025905	23694 ticks/100ms	27875 t/100ms/sec

Figure 1. Modified JVN computation for our 2019 robot's arm with motors, gear box, and load. The controlled rotational speed is 60 degrees per second (~80% of loaded theoretical maximum speed). When the arm is 10 degrees off its target, the motors will drive at full power. The arm accelerates to the target speed in .85 seconds.

2. Calculate proportional(P). How far should the final mechanism move before the motor should apply full power? This value will determine the response rate of the mechanism and will do the most to the control of the mechanism.

3. Determine the maximum velocity of the system. This can be computed in the worksheet. When setting max velocity in the control system, set it at least 10 percent below computed loaded velocity to allow headroom for PIDF control.

4. Determine the desired acceleration of the system. This is how quickly the system will reach the maximum velocity. If the system should take 1 second to accelerate to max velocity, then these values will be the same. If acceleration is half of the velocity it will take 2 seconds, if it's double, half a second, etc. a good starting point for most systems is to accelerate to Max velocity over 3-5 seconds. As you gain confidence with the system, you can decrease the time to Max velocity until you see error increasing in the tuner, or you get too much stress on the mechanical components Acceleration is what puts the most stress on, and applies the most force to the mechanism, contrary to just coasting at a constant velocity.

Test Setup with CTRE's Phoenix Tuner

Since we use CTRE motor controllers, we also use CTRE Phoenix Tuner software for testing. This software has many functions that are useful for testing and debugging CTRE devices

Install CTRE software

Phoenix documentation

The Phoenix documentation which is referenced in this whitepaper has a wealth of information about the Phoenix Tuner, Talon SRX, Victor SPX, and the CAN bus. If you have any questions, chances are the answers are contained in these docs.
The Phoenix Tuner is a live tool for configuring and viewing outputs from the Talon SRX and Victor SPX. It can be used for plotting outputs, changing CAN IDs, changing PIDF values on the fly, and more generally, it can be used to change and read parameters on CTRE motor controllers. (Download latest Phoenix framework installer for the most up-to-date Tuner and CTRE firmware)

Connect CAN bus wires and set CAN IDs

Make sure your wires are securely connected and that all of the motor controllers show that they are connected (blinking yellow for the Talon SRX and Victor SPX). This blink code means they are connected. Refer to the motor controller's documentation to find the other blink codes that are configured for your device. If the LEDs are blinking red, it is likely that either there is a loose CAN connection, a CAN wire is plugged in backwards, or a loop has been created on the CAN network. If there is another problem refer to the motor controller's documentation. Additionally, you must set unique IDs for each device. For setting CAN IDs in CTRE devices use these steps.

Set Control System Values

These values can be set in either code or in the Phoenix Tuner (under the config tab). The advantage of using the tuner to edit the constants is that they can be updated in between each test on-the-fly, without redeploying code in between each test; this also reduces the chances for errors caused by code changes. There is a set of constants that are related to the physical system and should not change (i.e. motor output and encoder phase and *invert*). Another set of values are configurable and may change during operation of the robot. (i.e. PIDF values, max velocity, acceleration, etc.)

Robot Test with Motors Unplugged

When running high power systems, the way that the system fails, and the amount of time it takes to fail change dramatically. To minimize risk, our testing procedure is to unplug the motors, and use the indicator lights on the Talons to tell if the behavior is expected.

1. Ensure that the motor controllers are wired up to power, the CAN bus, and encoder(s).
2. Ensure that the motor controller outputs are *unplugged.*
3. Plug in a spare motor to the output of one motor controller and test if the direction is correct relative to the lights on the motor controller. Then repeat for all motors in the system, If one of the motors is wrong verify that the *invert* on that motor controller is set correctly based on the physical system.

    a. Make sure to use the same type of motor, as different motors have different directions of spin (Bag and Cim turn in different directions).
    b. If there is a gear box between the motor and the output, double check that the output will be going in the direction that you expect.
4. Run through the expected actions of the robot and manually move the mechanism; thus causing the encoder to change, verify if the spare motor output speed and direction are as expected.
5. Ensure that the encoder moves in the expected direction relative to how the physical system is moved. When the system is moved in the positive direction, does the encoder move in the positive direction? Verify that the *sensor phase* is set correctly based on the physical system. If the *sensor phase* is inverted, the encoder will read more and more negative as more power is applied, and the power will max almost instantly.
6. Use the Tuner to set a target encoder point.
    a. Manually move the system to the specified target position. Ensure that the motor controller output goes to zero when the system is at the desired location.
    b. Move the system away from the target in the positive direction. The motor controllers should begin to blink red. Move the system away from the target in the negative direction. The motor controllers should begin to blink green. If this is not the case, check that the *sensor phase* and *invert* are set correctly based on the physical system.
    c. Move the system away from the target until the motor controllers turn solid red and solid green in each direction. These are the points where P is at saturation and the motor controller is applying maximum power.
        i. To decrease this range, increase P.
        ii. To increase this range, decrease P.
    d. If the motors drive outside the intended range of motion of the mechanism or way past the expected point, check the *sensor phase*, gear ratios and encoder wires.
        i. If the encoder wires are not plugged in or severed, the encoder position will always stay at zero, leading to infinite movement.
        ii. if the gear ratios are wrong, then the system will travel a different rate proportionally, possibly leading to movement through the robot.

Robot Test with Motors Plugged In

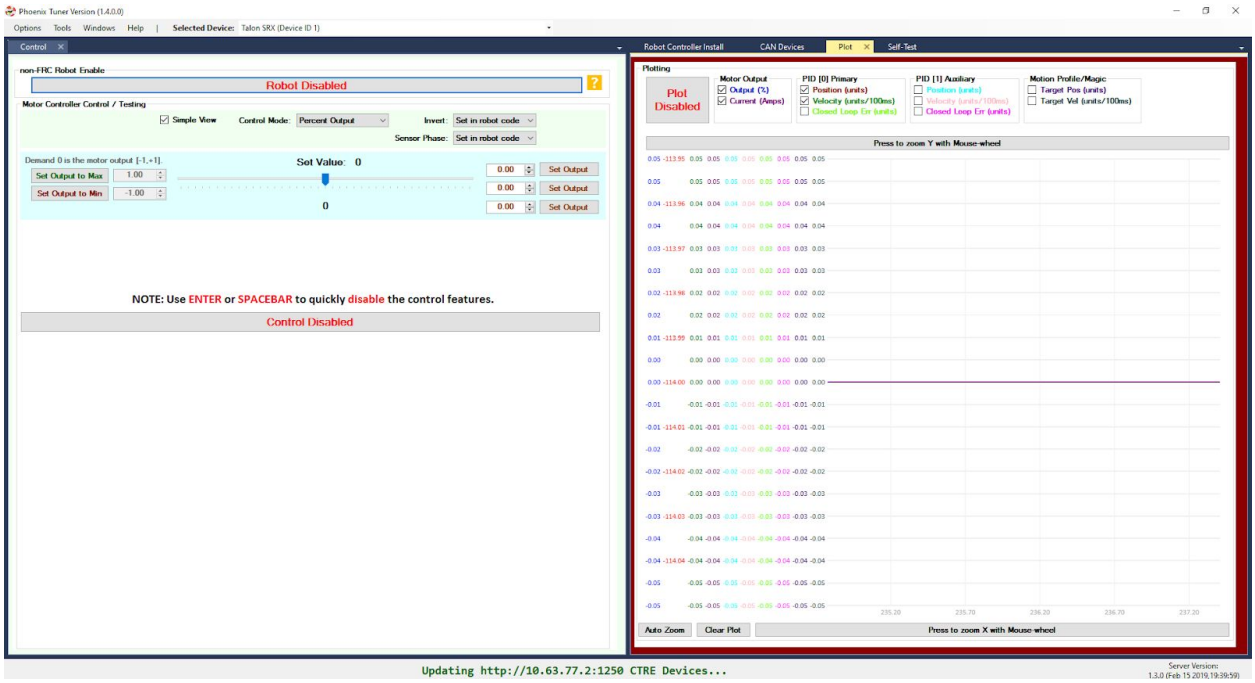
This step is vitally important to ensure the proper functioning of your robot, but is also the step where the most damage can be caused to your mechanism. This is the time in the process where everything needs to be thought through carefully before executing. Make sure the previous step is completed before moving on to this step.

1. Disable the robot before plugging in the motors.
2. Have one of your team members put their hand over the robot's main breaker, while standing in a safe location relative to the mechanism being tested
3. Plug in the motors, look for any form of sparking or smoking. If any smoking, sparking or movement happens, then power off the robot immediately, and recheck your wiring.
4. Once the motors are plugged in, and there have been no sparks, place the mechanism in a mid-position in which there is time to disable it before it runs into anything. For example, an arm rotating more than its expected angle, or an elevator hitting its maximum height.
5. Make sure that the space around the mechanism is clear throughout all the motion path of the mechanism. Enable with a hand over the disable button, as well as the robot's main breaker. If the mechanism doesn't do exactly what is expected, disable the robot and look through your code to find the issue.
    a. If more than one motor are physically linked together, such as through a gearbox make sure to plug in and test them one at a time, to prevent motor burnouts or gear shredding
6. If the robot does as you expect, then use the tuner control tab to make your robot move in small increments. As an added precaution make sure to set the movement amounts (in ticks) to a value that will not come close to the edge of the range of movement.
7. Run the mechanism between a number of points to verify the motion is as you expect. Once proper behavior is achieved, move on to the next section

Tune PIDF and Motion Magic constants

1. Using the Phoenix tuner and the Modified JVN spreadsheet tune these constants:

2. Set up the tuner so that you can edit constants on the robot, and view the



plot at the same time.

3. Run the mechanism while plotting the Percent Output, Position, Velocity, Closed Loop Error, as well as Motion Magic Position and Velocity. Once the movement finishes make sure to disable the plot, as the data will be lost if it goes off the side of the screen.
4. Look at the plot for oscillations that don't show up visually. These will usually appear as spikes in the percent output graph and rapid changes in the closed-loop error. If these show up, then lower the distance to P saturation on the JVN spreadsheet, and update values on the Tuner, until the oscillations decrease. If the oscillations will not go away, consider increasing the D constant in the loop to add a damping effect to reduce oscillations. Another possible cause is max allowable error which can cause spikes in percent output, max allowable error acts as a deadband for the PIDF loop, and if its too high the percent output will jump from zero to a high value and back, very quickly.
5. Look for increasing closed loop error: this can encompass a number of things. If the closed-loop error only increases as the motion magic profile is accelerating, then your acceleration term is too high. Decrease the time to

max acceleration in the JVN. If the error increases continuously over the profile, then your velocity or acceleration is probably too high, if so, decrease controlled speed. If there is a constant error that won't go away, increase P, if oscillations start happening before the error goes away, consider increasing the I constant in the loop. BE VERY CAREFUL WITH I, as it is very quick to cause oscillations; a very small amount goes a very long way.

6. Repeat these processes until your closed loop error is as low as possible (generally if the error is much smaller than the slop in the mechanism, it's good enough), and the plots for position, velocity, and percent output are smooth.
7. If you want to limit stress on your robot, reduce acceleration. Acceleration tends to cause the most problems, but also is a major contributor in achieving the required position promptly.
8. When tuning is finished, make sure to document the constants and put them into the code, because the constants might not persist between robot reboot cycles, or the swapping of a motor controller.

Additional Recommendations

The following are recommendations that we have seen useful and provide additional stability to the mechanism you are tuning.

- Keep your constants saved somewhere off the robot. Consider using a shared document to store them for each of your mechanisms.
- Consider using a configuration file to tune and load the mechanism's constant values in your code and be able to tune them easily.
- Communicate with the drive team. Make sure they understand the physical system and tuning of the robot so that they can recognize and report discrepancies in expected behavior.

Glossary

Invert	changes the direction that the motor controller thinks is positive.
Sensor Phase	changes if the direction that the encoder thinks is positive to the way that the motor controller thinks os positive
Mini-CIM	commonly used motor in FRC
Gearbox Stage	gear set in a gearbox to change output ratio
Chain system	set of sprockets and chain to transfer force from one point to another while possibly changing output ratio
Encoder	a device that is used to measure rotary position, the one referred to in this paper, the CTRE magnetic encoder, uses a magnet mounted in the end of the shaft to tell position
Resolution	as referring an encoder, resolution is number of measurements (ticks) per revolution
Slop	refers to the amount of movement a mechanism can make without moving the motor
Limit Switch	a physical switch placed on the edge of range of motion, when this switch is tripped, motor output moving the mechanism in that direction is disabled
Error/Closed loop error	difference between the target position of a control loop and the actual position of the mechanism as measured by an encoder or another device.

Control Loop	a system used to control a motor to a specific position or velocity
Feedback Loop	a control loop that uses feedback, generally from an encoder or similar device, to optimize control
PIDF	a feedback loop commonly used in many applications that uses 4 different terms to coordinate motion, see page 6 for more information
Tuning	the process of tweeking constants to make the mechanism move in a more optimal way
Motion Magic	a control loop that automatically generates trapezoidal motion profiles from a maximum velocity and acceleration term
Blink codes	a blink code is a series of blinks on an LED that signals something
Howdy JVN	a customized JVN spreadsheet used for calculating PIDF constants and motion magic constants. Can be found at https://howdybots.org/wp-content/uploads/2019/12/HB_PID-JVN-DesignCalc.20191207.xlsx