Encoders and Motor Control Theory

FRC Team 4277

Author: Nate Fatkhiyev
nfatkhiyev@gmail.com

Introduction:

This document serves as a guide and recommendation for the use and implementations of encoders from an analytical point of view. It will cover the mounting, limitations and applications of encoders on an FRC robot without software examples due to the changing nature of FRC programming.

- **What are Encoders?**

  Encoders are sensors used to measure the rotation of a wheel, mechanism or other object's rotation about an axis. Encoders work to serve two purposes.

  The first is to count the number of poles passed of a rotating polarized magnet (magnetic encoder) or the number of pulses on a rotating optical disc (optical encoder). This is used to calculate the distance that an arm or wheel has traveled around an axle.

  The second is to measure the rate of the pulses that pass the sensor. This allows you to calculate the velocity about the axis of the object you are measuring.

- **Why use encoders?**

  Using encoders allows you to better control the movement of your robot's mechanisms and your robot on the field. This is done by closed loop motor control. Closed loop control of a motor is to set the voltage going to the motor based on the feedback you get from a sensor. This allows your robot to run motors more consistently, accurately and sometimes more efficiently. By making your mechanisms run more consistent, accurate or efficient, you can reduce the time to accomplish a certain aspect of the game dramatically.

- **How do I use Closed Loop Control with an Encoder?**

  The two most common uses of Closed Loop Control in FRC are PIDF controllers and Motion Profiling controllers.

  PIDF is a type of motor control function that increases or decreases the voltage provided to a motor based on a user defined setpoint and an analytical error function. The user defined setpoint can be a certain distance, position (mainly relevant in potentiometer use), RPM value or velocity (RPM value and velocity both require the desired RPM but velocity returns a speed to the user not the RPM of the motor). The PIDF function uses four terms to set the voltage of the motor: Proportional gain coefficient, Integral gain coefficient, Derivative gain coefficient and Feed Forward term. These four terms find the error of the measured rotation from the setpoint and and combine it with four determined coefficients to manipulate the voltage going to the motor to match the setpoint.

Motion Profiling is a motor control function that defines the velocity and position of your mechanism over time. Unlike PIDF, there are several functions and models that can be applied to motion profiling that each allows for a different motion profile (shape of velocity vs time or position vs time graph). The importance of motion profiling is that it allows you to limit the acceleration of your robot so that you don't break any part of it. Examples of motion profiling models include: exponential model, linear model, quadratic model, logarithmic model, S - Curve (pathfinding) models, parametric models, Quintic Hermite splines. Some of these models serve to control the acceleration of your motor independently and others allows you to control the acceleration of a motor in relation to another motor to allow for complex paths of movement. For a combination of simplicity and effectiveness, I recommend using a logistic model to define the velocity and position of the robot vs time (will be explained later in this document) for drivetrain motion profiling.

Grayhill 63R256 Encoder:

The Grayhill 63R256 encoder is a pre assembled optical encoder that has a 0.25” input shaft and a 0.5” mounting shaft. It is a quadrature encoder that has 256 pulses per revolution.

Input Voltage: 5V
Max RPM: 5000 RPM
Max Current: 30mA

Using a pre assembled encoder with an input shaft and surface mounting allows for installation without damaging the sensor, increases durability and higher precision.

An encoder like this is better suited for mechanisms with lower RPM that require higher precision. This encoder would be ideal for a drive train.

VersaPlanetary Integrated Encoder (SRX mag encoder):

The SRX mag encoder is a quadrature encoder using a diametrically polarized magnet to count rotation of a shaft. The encoder output is based on a 1024 count per rotation system and is integrated into the Talon SRX programming and control mode.

Input Voltage: 5V
Max RPM 15000 RPM
Max Current: 31mA

This encoder is better suited for mechanisms operating at high rpm and fairly isolated. Measuring the rotation of a flywheel or winch would be good applications for this encoder.

Mounting Encoders

Encoders are extremely sensitive sensors and require you to include their mounting situation in your design. Vibrations, misalignment, backdrive and torque can all ruin the encoder and data coming from it. Be sure to consider these factors when mounting an encoder to a system.

- **Grayhill 63R256 Mounting:**

  Since the Grayhill 63R256 would be primarily used in a drivetrain, the information in this section will be specific to drivetrain mounting but can applied to other applications.

  The best way to mount the Grayhill 63R256 is to use a thin metal bracket to hold the housing of the encoder and a flexible coupling to connect a passive axle to the input shaft. A good flexible and cheap solution is to use surgical tubing clamped with zip ties as a coupling.

- **VersaPlanetary Integrated Encoder Mounting:**

  The VersaPlanetary Integrated Encoder attaches to a VersaPlanetary gearbox after the last stage and before the output. This ensures that you are reading the velocity and position of the output shaft rather than the motor shaft. To use this with an unreduced CIM motor, you can attach it to a 1:1 reduction VersaPlanetary gearbox and attach that to the CIM motor.

  VersaPlanetary Integrated Encoder mounted onto a BAG Motor:

Basic Encoder Functions:

It is important, when using encoders, to be able to manipulate the data in a way that renders the pulse rate or count useful.

- **Distance Traveled**

  If you set up your code to count the number of pulses that the encoder reads, you can easily calculate the distance traveled:

$$d = 2 * \frac{number\ of\ pulses}{pulse\ per\ rotation} * (\text{gear reduction}) * \pi r$$

  Where:

  d = distance
  number of pulses = the counted number of pulses
  pulse per rotation = encoder pulses per rotation constant
  gear reduction = the ratio of the output gear to input gear that the encoder is connected to.
  r = the distance from the axis of rotation

- **Output RPM**

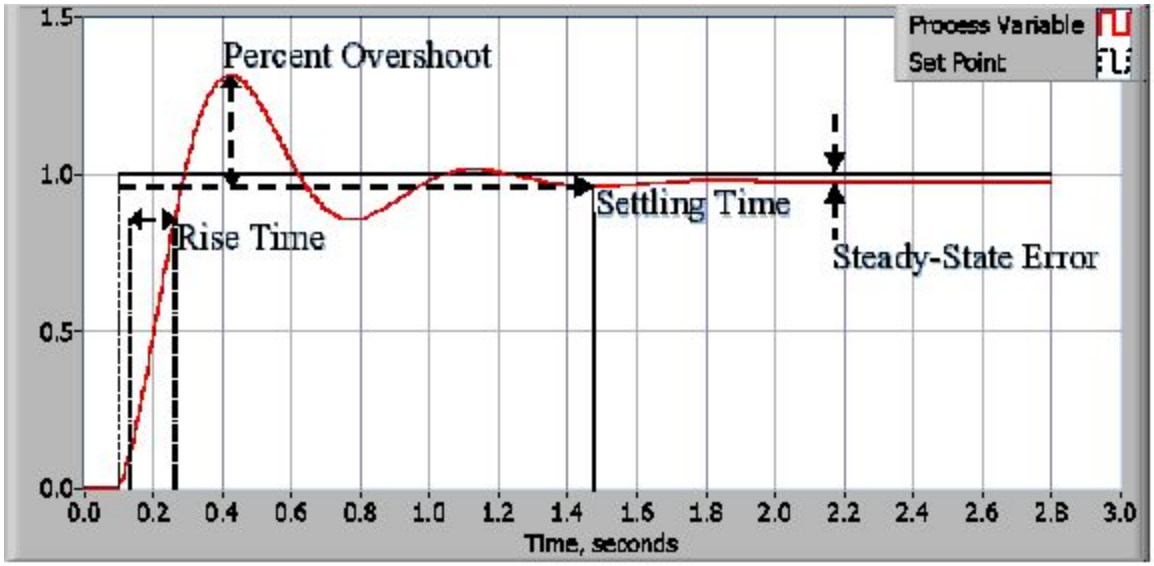
  The RPM of the output shaft of your system can be calculated by this equation:

$$RPM = 60 * (\text{pulse/sec}) / (\text{pulse per rotation})$$

PID

PID is a plant controller that allows you to create a setpoint (distance or RPM) for your system. This means that the motor will automatically return to the setpoint if it is not already there. PID can be used with almost any sensor including: encoders, gyros, ultrasonic, accelerometers and potentiometers. Using PID will increase the functionality of your robot a lot. In 2017 many robots shot balls into the boiler in large streams. PID is what makes this possible. Each time a ball passes through a shooter, it slows the shooter down. Teams that used PID were able to compensate for this in an extremely short time allowing them to shoot balls rapidly. Another example is if a cascading lift gets moved out of position, PID will automatically attempt to return the lift to its setpoint.

PID is not an easy thing to learn. While the concept of PID is not very complex, properly tuning a PID loop requires many complex variables and functions. Given our short build season and the complexity of PID, it is best to tune PID experimentally or using MATLAB.



Setting up PID

- **Is PID Right for Me?**

  The first thing you must do in setting up a PID loop is to identify what you wish to accomplish with PID and if it is the best solution for you. The factors involved in this are:

  - Do you have a determined setpoint?
  - Will there be any disturbances to the system?
  - How big are the disturbances to my system?
  - How fast do I need to compensate for any disturbances?
  - Is my motor capable of correcting these disturbances in my time frame?
  - How accurate does my system need to be?
  - How reliable is my encoder?

  PID controllers are good when you can define a setpoint, need to compensate for disturbance quickly and have an accurate and reliable way of measuring the output of your motor. If your time period is too large, your motor is not very responsive or you have faulty sensors, PID may worsen the performance of your mechanism by causing the motor to oscillate, exceeding the limits of your mechanism or stop the motor all together.

- **Preparing to Tune PID:**

  Before tuning your PID make sure that you have the following items prepared:

  - **100% Working hardware** - if you are confident that your motor controller, motor, encoder and manual control programming are working 100% then you can assume that any issues you experience are software based problems.
  - **Be able to find your setpoint and error** - if you can see your setpoint and actual reading on your dashboard, you can find the effectiveness of your PID loop
  - **Know how big your displacement or error is of your system and how much time you have to fix it** - this ensures that you are able to recognize when your PID loop is working vs when it is not.
  - **Ensure that you motor is in BRAKE mode** - PID will work extremely poorly in coast mode.
  - **Have a fully charged battery** - PID will underperform severely if you do not have a charged battery.

Tuning PID:

Tuning your PID loop means determining the values for Kp (proportional gain), Ki (integral gain), Kd (derivative gain). This is by far the most important and most difficult task in implementing PID. If all of your setup is working properly, PID tuning should not take much effort. It is the only part of the process that must be done on the actual assembly of the mechanism. Given the large number of uncertainties in FRC robots, using a PI (set Kd to 0) controller will be best. Kd is extremely sensitive to noise and can introduce extra interruptions in your PID loop.

- **Ziegler - Nichols Method:**

  Once you have ensured that your hardware works and you know that you can control the motor using software, enable PID (at a known setpoint) with all three gains at 0. The motor should not move.

  Next start increasing Kp slowly. The motor will start moving toward its setpoint. Observe the oscillation of the motor carefully. Continue increasing Kp until the oscillations are stable and consistent. Record the period of oscillation to the best of you ability. Taking multiple readings and averaging them will yield better results.

  Finally adjust Kp and Ki values according to chart:

Control Type	K_P	K_I	K_D
P	$0.50K_c$		
PI	$0.45K_c$	$1.2K_p/T_c$	
PID	$0.60K_c$	$2K_p/T_c$	$K_pT_c/8$

  One you have a stable PID loop, record the gain values and adjust them in small quantities until you have reached your desired result. I encourage you to research PID tuning on your own to gain perspective as to how to change values. The more practice you have tuning PID the more insight you will have on what changes to make.

- **Feed Forward:**

  The feed forward term is the required output to keep our mechanism at its current state. This includes constant velocity of a flywheel and the output required to keep arm or lift in position. This calculation varies greatly on your mechanism and is easy to implement into a tuned PID loop. FIRST provides great resources on this topic and implements it into some PID functions automatically.

MATLAB Tuning:

MATLAB makes PID tuning much easier and gives you significantly more control over your response time and settling time. The downside to MATLAB tuning is that it requires a much more mathematical understanding of motors, PID and rotational dynamics. In order to tune PID in MATLAB, you must have the following information:

- Motor Inertia
- Motor Viscous Damping
- Motor Torque Constant
- Motor Back EMF Constant
- Motor Armature Resistance
- Motor Armature Inductance
- Mass of Wheel
- Radius of Wheel
- Inertia of Wheel
- Estimated Load Viscous Damping
- Output Gearing

Once you have all of these, you can use them in a plant mask and PID controller block and use the PID Tuning tool to adjust the ramp rate and settling rate to meet you needs. MATLAB will then calculate the Kp and Ki values for you to use.

Motion Profiling:

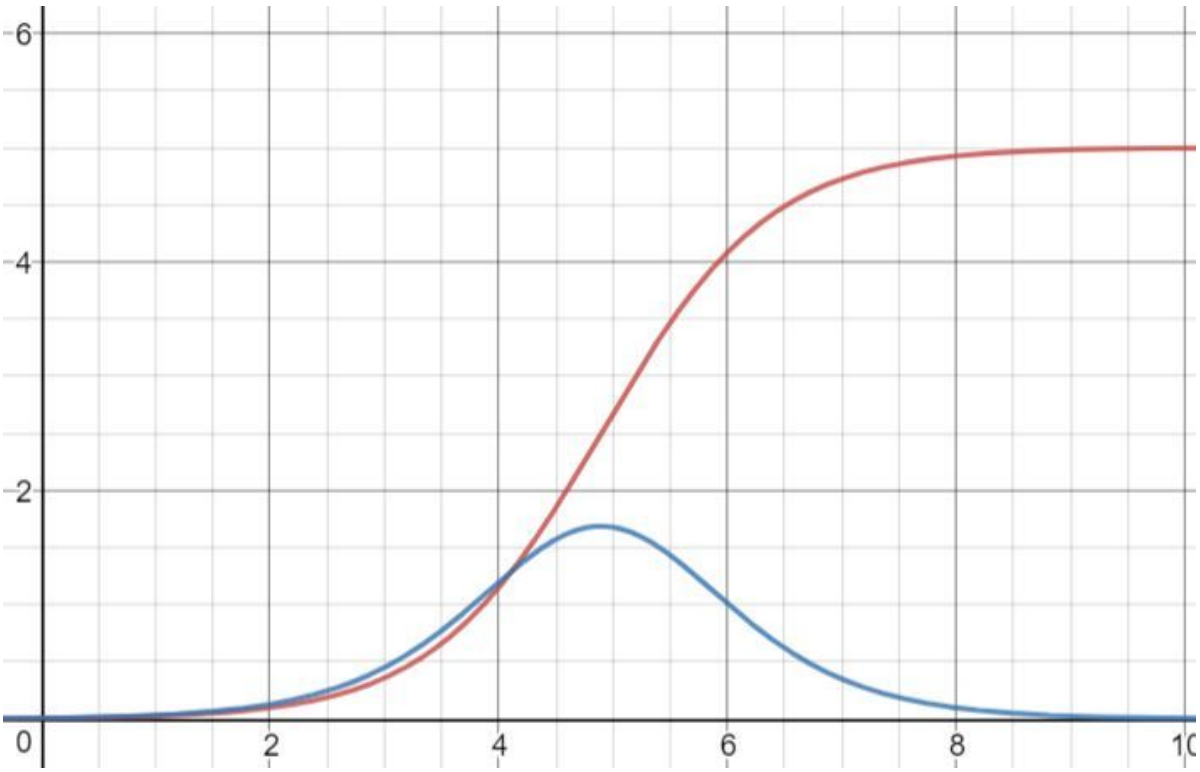
The addition of encoders allows for the ability to measure acceleration, velocity and distance as a function of the rotation rate and pulse count of the encoder. This is important in FRC to facilitate smooth and balanced autonomous movement.

Motion profiling is the change in the robots velocity as a function of time. The need for this feature depends heavily on the design of your robot. A good example of how important motion profiling can be is seen in several 2018 (First Power Up). In 2018 many robots had lifts to allow for cube placement on the center scale. As a result, robots were top heavy and relatively unstable. During several matches, robots would tip over due to rapid acceleration and deceleration. A good way to avoid this in autonomous, and in some tele-op situations, is motion profiling.

Motion Profiling: Logistic Model

One way that motion profiling can be done is with a logistic model controlling the setpoint of a PID loop or the motor output as a function of time. The advantage to using a logistic model over other models is that it gives you control over the entire time interval in one function rather than 3 separate intervals (acceleration, high velocity, deceleration) and has a non-linear jerk.

The following image depicts a logistic model where a robot travels 5 units in 10 seconds:



The Logistic Model:

The function for the velocity in a logistic model is defined as:

$$dx/dt = kx(D-x)$$

Where:

- dx/dt = velocity
- k = an arbitrary constant
- D = target distance
- x = position as a function of time

We can observe that variables a and b (coefficients of 1 and x respectively) is equal to Dk and k respectively.

Now we can solve the differential equation by separation of parts and partial fractions to get the equation for the position of the robot as a function of time:

$$x(t) = (aC)/(bC + e^{(-at)})$$

Where:

- a = Dk
- b = k
- C = constant of integration
- t = change in time

At this point, as long as a/b > motor deadzone velocity, we can solve for C using a, b, and initial velocity. A flaw of the logistic model is that the function is undefined at 0 velocity. This means that your analytical initial velocity cannot equal 0. To fix this, we find the theoretical velocity of the motor at some point within its deadzone to as our initial velocity. By doing this, our motor will not be powered at our setpoint giving us a functional model.

$$C = x(0)/(a-bx(0))$$

Once constant C is determined, we can solve for k by defining our analytical solution as an initial value problem where we define x(0) as in the step before and dx/dy at the maximum velocity we want.

The maximum velocity will occur at the inflection point of the position function which is defined as a/2b. This will give you the position at which maximum velocity is achieved. By substituting this into the velocity function we can solve for k and get the total time elapsed to travel a dictated distance.

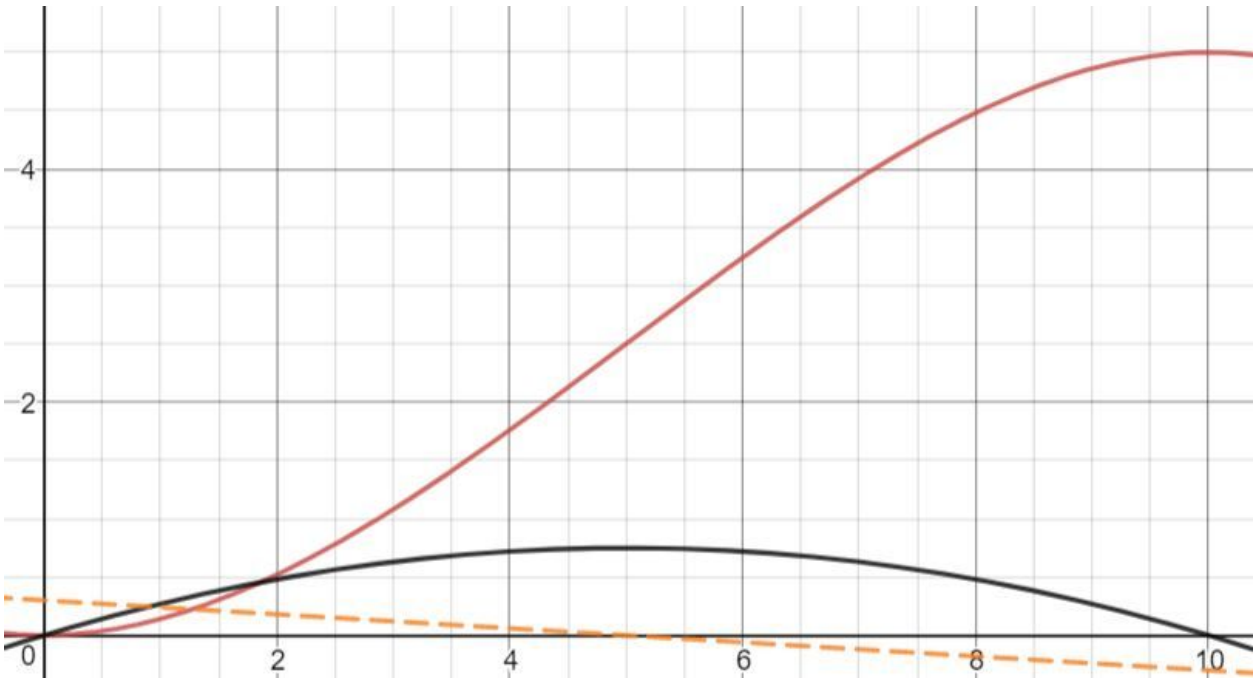
We can also get k by defining the maximum time we would like to spend traveling a dictated distance. Once we have decided on our distance and time, we can graph the problem in terms of k and t and plug in values of k to satisfy our defined time. A good resource for this is desmos.

When using a logistic model, it is important to know the limitations of your robot. The purpose of motion profiling using a logistic model is to limit the robots movements so as to not exceed its limitations. Without knowing its limitations, this model and any other motion profiling function is rendered useless.

Motion Profiling: Cubic Model

Another way that motion profiling can be done is by controlling the motion of your robot with a cubic model. This model has an almost identical functionality to the logistic model. The differences lie in the response of the control model to change. The cubic model, unlike the logistic model, yields a constant acceleration jerk for any situation. If testing shows that a constant acceleration associated with a cubic model best matches the limitation of your robot, it will be a preferable method due to its simplicity in comparison with the logistic model.

The following image depicts a cubic model where a robot travels 5 units in 10 sec:



The Cubic Model:

The cubic model is based off of the following position as a function of time on the interval of [0,T]:

$$x(t) = a + (3/T^2)(b-a)t^2 - (2/T^3)(b-a)t^3$$

Where:

- a = initial position
- b = final position
- T = total time to move from a to b
- t = time elapsed

By differentiating we get a velocity function on the same interval:

$$v(t) = (6/T^2)(b-a)t - (6/T^3)(b-a)t^2$$

Where:

- a = initial position
- b = final position
- T = total time to move from a to b
- t = time elapsed

Differentiating a third time gives us the acceleration as a function of time again on the same interval.

$$a_x(t) = (6/T^2)(b-a) - (12/T^3)(b-a)t$$

Where:

- a = initial position
- b = final position
- T = total time to move from a to b
- t = time elapsed

Implementing this model is simple in comparison with a logistic model. By substituting a, b and T with known values, and t being measured by the robot, a constant acceleration will be suggested based on these results.

A constant acceleration or peak velocity can also be used to define the terms of the function.

Robot Calibration:

Something that is critical in the implementation of a good control system is to know the constraints of the robot. This includes: maximum velocity, maximum acceleration, turning radius, and the transfer function (relationship between controller input and output). It is also important to know the precision that your robot can achieve.

Robot Calibration: Numerical Analysis

One way that the information described above can be attained is through the use of a numerical solution to the set of differential equations known as kinematics. This method tracks the acceleration based on the plant input value. With every cycle of the robot code, a new data point will be collected. This data can be stored in an ArrayList and converted into a CSV file. Once this data has been stored, you can either pull the data from the robot computer and create a regression in excel or run a numerical solution algorithm to correlate the plant input to the velocity of the robot. If you do not want to use closed loop control for your robot positioning system, a good robot calibration allows for open loop control that is more predictable than without.