

MOTION MAGIC WITH CTRE MOTOR CONTROLLERS

ThunderChickens – FRC Team 217

[CTRE Phoenix Motion Magic example](#)

[CTRE Phoenix motion profiling example](#)

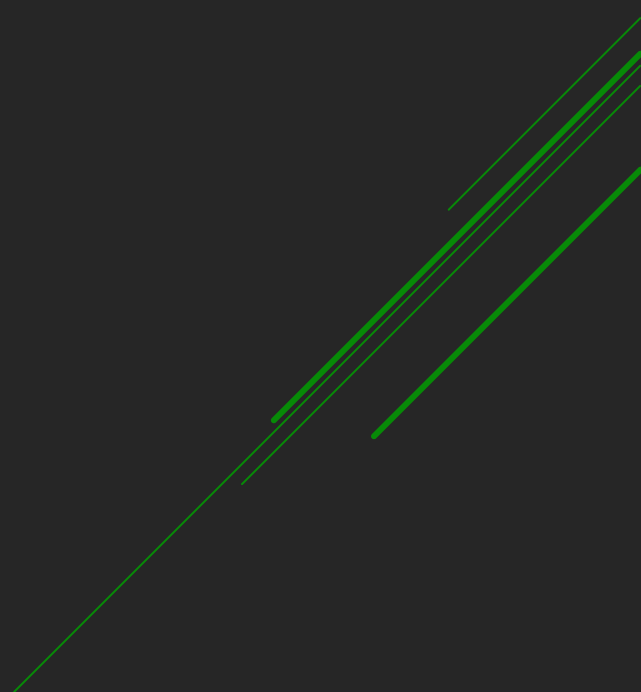
WHAT IS MOTION MAGIC?

- ▶ Drives the motor to a target position
- ▶ A form of geometric motion profiling with PID
 - ▶ Feed forward on velocity target, PID on position target
- ▶ Motion profile calculated as the motor runs



ADVANTAGES OF MOTION MAGIC

- ▶ Simple to use
 - ▶ Pass in PID-F, cruise velocity, target acceleration, and S-curve smoothing (to round the corners of the trapezoidal profile) as configs
 - ▶ Pass in target position when running Motion Magic
- ▶ Does not have to be calculated ahead of time
- ▶ Position PID gives precise control of the behavior of the robot
- ▶ Replaces position PID control
 - ▶ Has all the advantages of motion profiling with the simplicity of position PID control



CONFIGURING MOTION MAGIC

- ▶ Calculating kF:
 - ▶ Get motor velocity (with load) at a given percent output (Recommended: ~75% of max)
 - ▶ $kF = percOut * \frac{1023}{motorVel}$
- ▶ In the example below, the recorded velocity at 100% power was 5100 ticks/100ms

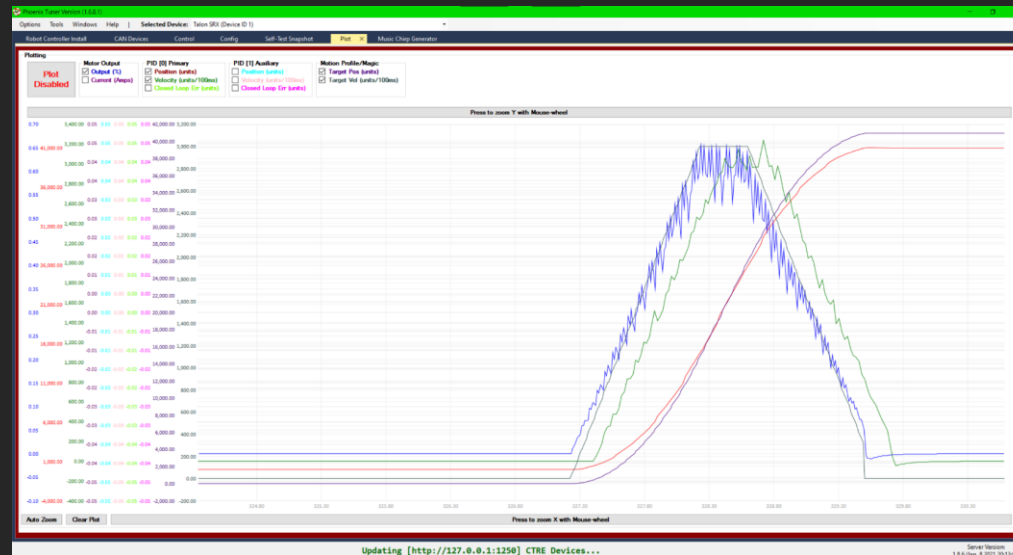
```
/* Set Motion Magic gains in slot0 - see documentation */
_talon.selectProfileSlot(0, 0);
_talon.config_kF(0, 1023 / 5100.0);
_talon.config_kP(0, 0.2);
_talon.config_kI(0, 0);
_talon.config_kD(0, 0);

/* Set acceleration and vcruise velocity - see documentation */
_talon.configMotionCruiseVelocity(3000);
_talon.configMotionAcceleration(3000);
_talon.configMotionSCurveStrength(4);
```

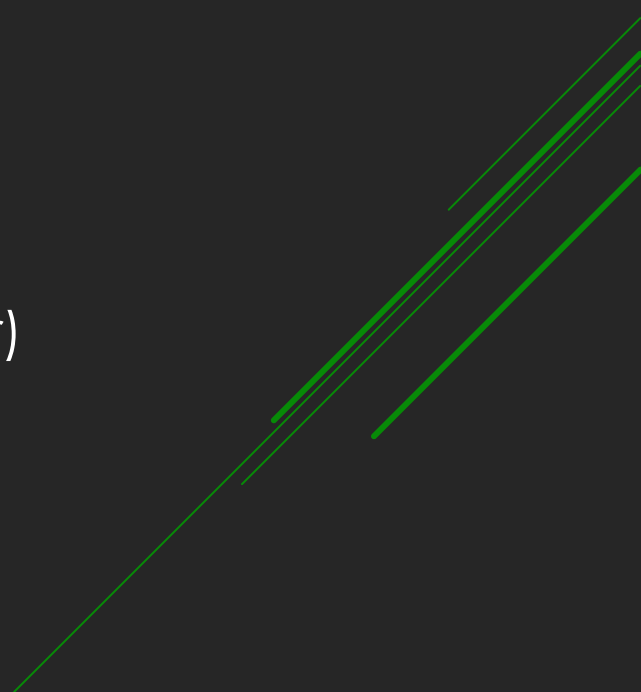
RUNNING MOTION MAGIC

- ▶ Given the target position `targetPos` and Talon SRX/FX `_talon`:
 - ▶ `_talon.set(ControlMode.MotionMagic, targetPos);`
- ▶ You can view percent output, current, target and recorded position and velocity, etc. in the Plot tab of Phoenix Tuner

```
_talon.set(ControlMode.MotionMagic, targetPos);
```



WHAT ABOUT MOTION PROFILING?

- ▶ CTRE offers the ability to run a motion profile in a special motion profile control mode
 - ▶ Feed Forward (kF) applies to the profile target velocity
 - ▶ PID applies to the profile target position
 - ▶ Useful for passing in custom trajectories (such as from PathWeaver)
 - ▶ Requires a pre-calculated motion profile trajectory
 - ▶ Configuring PID-F is identical to Motion Magic
- 
- Several thin, parallel green lines of varying lengths and orientations are positioned on the right side of the slide, extending from the middle to the bottom right corner.

USING MOTION PROFILING

► Configuring a motion profile

1. Configure PID-F (see Motion Magic)
 - Do not configure cruise velocity, acceleration, or S-curve strength
2. Add the points in the motion profile trajectory to a BufferedTrajectoryPointStream

► Running a motion profile

1. Start the motion profile
 - Pass in the BufferedTrajectoryPointStream and the number of points to buffer
2. Run until the motion profile is finished:

```
public boolean isFinished() {  
    return _master.isMotionProfileFinished();  
}
```

```
BufferedTrajectoryPointStream _bufferedStream = new BufferedTrajectoryPointStream();  
boolean forward = true; // set to false to drive in opposite direction of profile (not really needed  
                        // since you can use negative numbers in profile).  
  
TrajectoryPoint point = new TrajectoryPoint(); // temp for for loop, since unused params are initialized  
                                                // automatically, you can alloc just one  
  
/* clear the buffer, in case it was used elsewhere */  
_bufferedStream.Clear();  
  
/* Insert every point into buffer, no limit on size */  
for (int i = 0; i < totalCnt; ++i) {  
  
    double direction = forward ? +1 : -1;  
    double positionRot = profile[i][0];  
    double velocityRPM = profile[i][1];  
    int durationMilliseconds = (int) profile[i][2];  
  
    /* for each point, fill our structure and pass it to API */  
    point.timeDur = durationMilliseconds;  
    point.position = direction * positionRot * Constants.kSensorUnitsPerRotation; // Convert Revolutions to  
                                        // Units  
    point.velocity = direction * velocityRPM * Constants.kSensorUnitsPerRotation / 600.0; // Convert RPM to  
                                        // Units/100ms  
    point.auxiliaryPos = 0;  
    point.auxiliaryVel = 0;  
    point.profileSlotSelect0 = Constants.kPrimaryPIDSlot; /* which set of gains would you like to use [0,3]? */  
    point.profileSlotSelect1 = 0; /* auxiliary PID [0,1], leave zero */  
    point.zeroPos = (i == 0); /* set this to true on the first point */  
    point.isLastPoint = ((i + 1) == totalCnt); /* set this to true on the last point */  
    point.arbFeedFwd = 0; /* you can add a constant offset to add to PID[0] output here */  
  
    _bufferedStream.Write(point);  
}
```

WHEN TO USE WHICH

- ▶ Motion Magic

- ▶ Would normally use standard position PID control or a geometric motion profile
- ▶ Want to have precise control of linear trajectories or of a single robot component (elevator, arm, etc.)

- ▶ Motion Profiling

- ▶ Want to accurately move along curved trajectories (drivebase)

