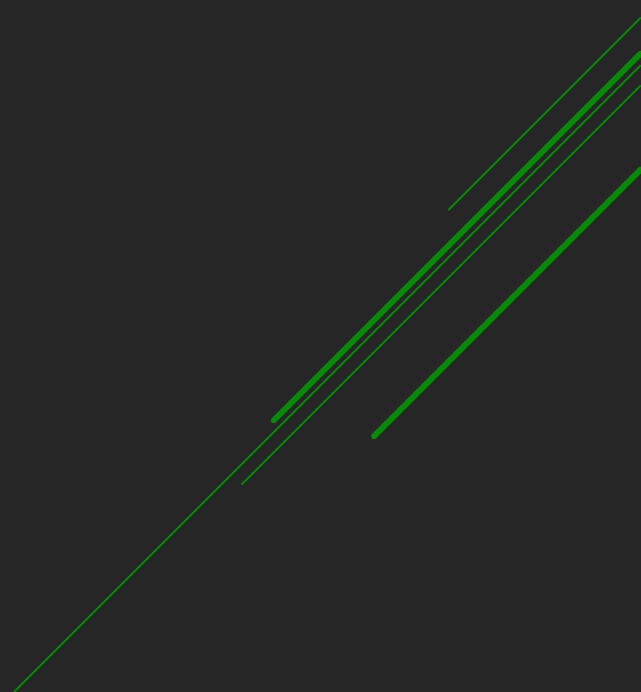


# BASIC THEORY OF MOTION PROFILING IN CODE

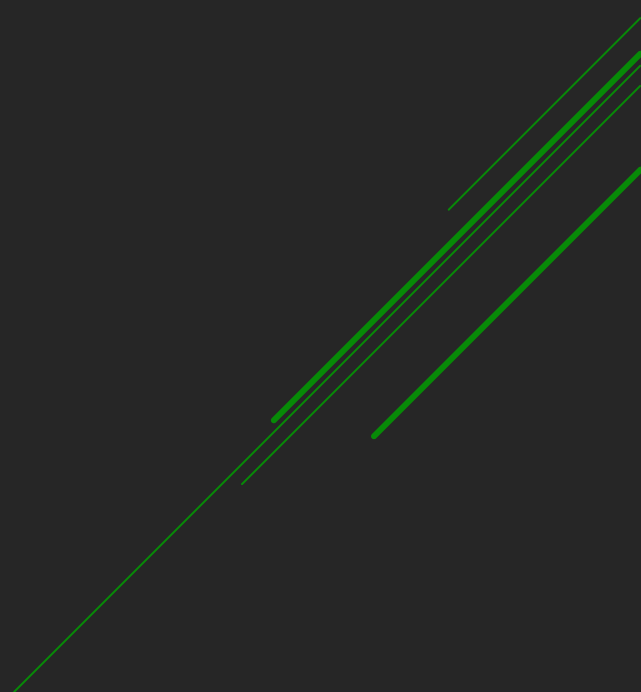
ThunderChickens – FRC Team 217

Several thin, parallel green lines of varying lengths and slopes are positioned on the right side of the slide, extending from the top right towards the bottom right.

# WHAT IS MOTION PROFILING?

- ▶ A motion profile is a graph/function specifying how a component should travel from one position to another
  - ▶ Typically calculated as a velocity graph with the position graph being the integral of the velocity graph
  - ▶ Used to achieve more reliable and better-controlled motion than positional PID
- 
- Three parallel green lines of varying lengths are positioned in the bottom right corner of the slide, slanted upwards from left to right.

# TYPES OF PROFILES

- ▶ Ramp rate
  - ▶ Geometric
    - ▶ Purely geometric
    - ▶ Geometric with PID
  - ▶ Waypoint (trajectories)
- 
- A series of parallel green lines of varying lengths and slopes, located in the bottom right corner of the slide, creating a modern, abstract graphic element.

# RAMP RATE

- ▶ Add to the velocity at a given rate
- ▶  $v = a_t * t$ , or  $dv = a_t * dt$ 
  - ▶  $a_t$  is target acceleration
  - ▶ We typically work in deltas ( $dv$ ,  $dt$ )
- ▶ Used in combination with position PID
  - ▶ Setpoint is final position, measurement is current position
  - ▶ Hand control of velocity over to PID once the ramp rate and PID lines intersect ( $v \geq \text{pid.getOutput()}$ )

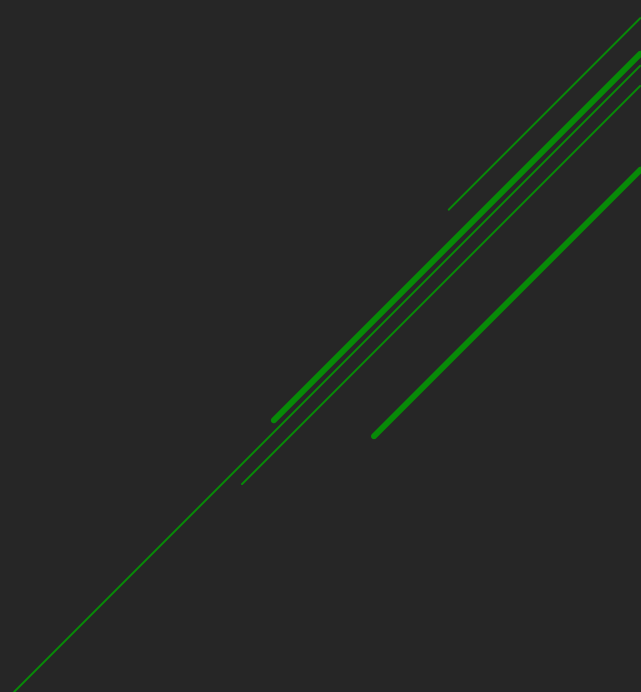
# RAMP RATE

## ▶ Pros:

- ▶ Easy to write
- ▶ Easy to use
- ▶ Motors don't start at full speed

## ▶ Cons:

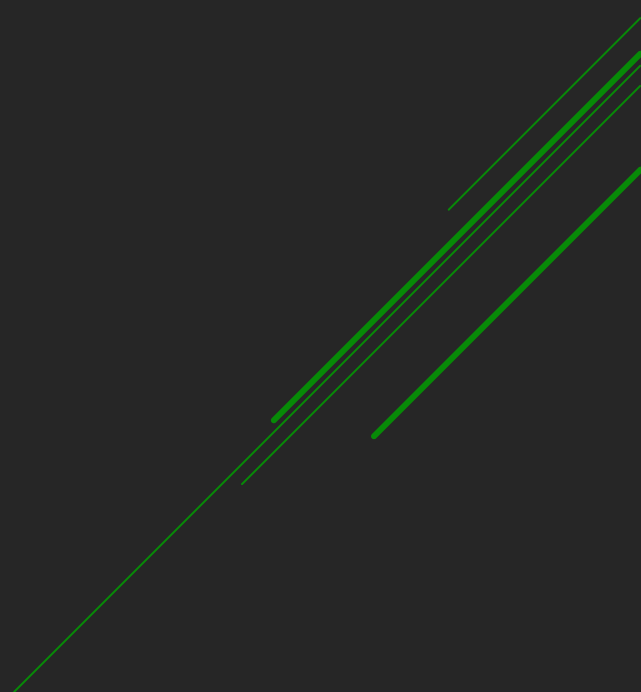
- ▶ Can't apply on the deceleration period
  - ▶ Causes overshoot
- ▶ Still has all the issues of traditional position PID after reaching full speed



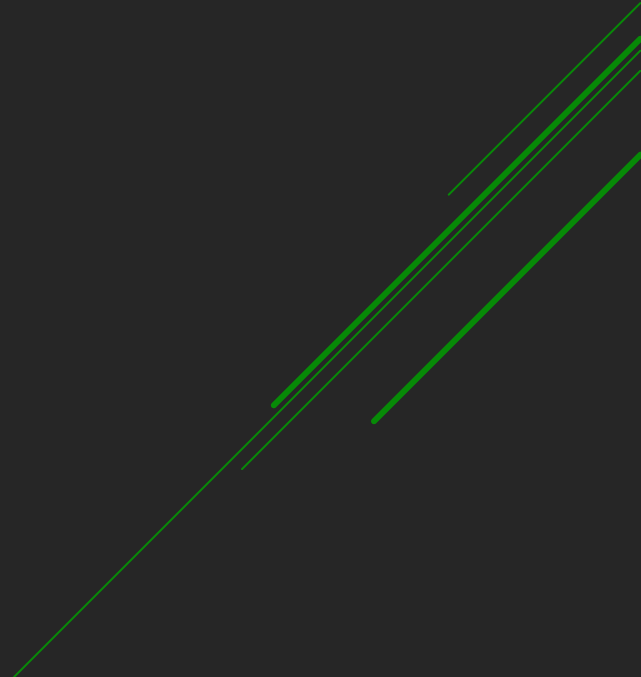
# RAMP RATE – ALTERNATE FORMULA

- ▶ Assume  $v$  is a parameter that does not exceed the max velocity of the motor controller
- ▶  $dv = \text{signum}(a) * (a_t - |a|) * dt$ 
  - ▶  $a$  is current acceleration
  - ▶  $\text{signum}$  returns +1 if input is positive, -1 if negative, 0 if zero
  - ▶ Only runs if  $|a| > a_t$  and if  $a * v > 0$  (speeding up)
- ▶ Logic
  - ▶ Need  $\text{signum}$  to get the sign of  $a$  since  $|a|$  is used, and  $a_t$  is positive
  - ▶  $|a| > a_t$ , accelerating too fast, subtracts the error in  $a$  from  $v$

# GEOMETRIC – PURELY GEOMETRIC

- ▶ Velocity vs Time graph forms a trapezoidal shape
  - ▶ Given target distance, max velocity, and target acceleration, can calculate how long to accelerate and decelerate, and how long to drive full speed in between
  - ▶ Area under graph is position:  $s(t) = \int_0^t v(\tau) d\tau$
  - ▶ Slope of graph is acceleration:  $a(t) = \frac{d}{dt} v(t)$
- 
- A series of parallel green lines of varying lengths and orientations are positioned in the bottom right corner of the slide, creating a modern, abstract graphic element.

# GEOMETRIC – PURELY GEOMETRIC

- ▶ Pros:
    - ▶ Doesn't rely on encoders
  - ▶ Cons:
    - ▶ Relies on a perfect system
    - ▶ More complicated to calculate than ramp rate, but less effective
  - ▶ No good team utilizes this system
- 
- Several thin, parallel green lines of varying lengths and orientations are positioned in the bottom right corner of the slide, creating a decorative graphic element.

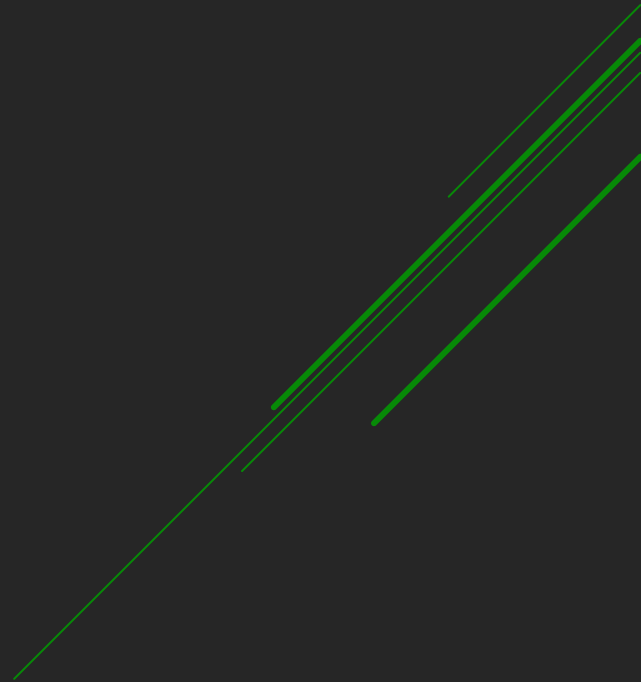


# GEOMETRIC – WITH PID

- ▶ Uses the Purely Geometric curve
- ▶ Using integrals, calculate target velocity and target position at a given time
- ▶ Feedforward with target velocity, feedback from position
  - ▶ Feedforward: “I should be going this fast”
  - ▶ Feedback: “I should be here but am (ahead/behind), so I should go (slower/faster)”
- ▶ Using velocity control mode:
  - ▶ `motor.set(ControlMode.Velocity, vt + pid.getOutput(p, pt))`
    - ▶ v<sub>t</sub> is target velocity at the current time (from profile)
    - ▶ p is current position
    - ▶ p<sub>t</sub> is target position at the current time (from profile)
  - ▶ Configure kF (feedforward) and optionally kP on the motor controller
    - ▶ Get motor velocity (with load) at a given percent output (Recommended: ~75% of max)
    - ▶ For CTRE:  $kF = percOut * \frac{1023}{motorVel}$ , tune kP afterwards as necessary

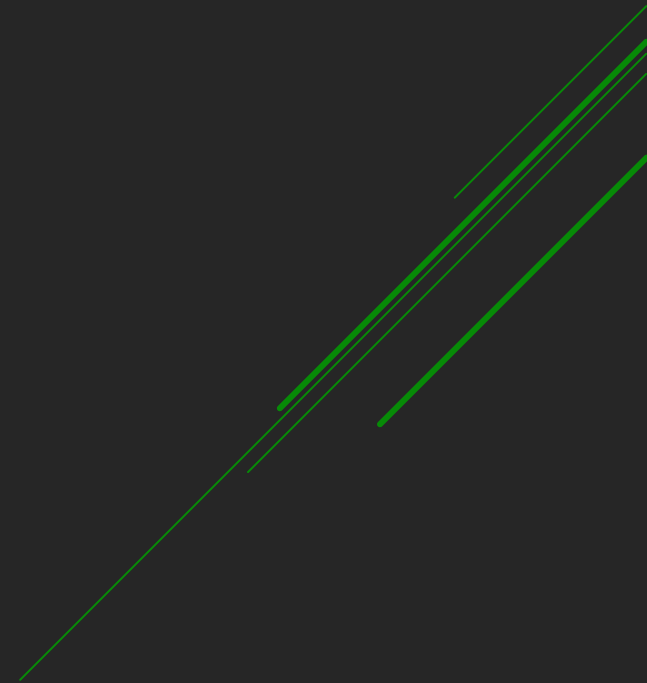
# GEOMETRIC – WITH PID

- ▶ Pros:
  - ▶ Precise
  - ▶ Applies to both acceleration and deceleration
  - ▶ Manages errors in position and velocity
- ▶ Cons:
  - ▶ Harder to program



# WAYPOINT (TRAJECTORIES)

- ▶ Calculate multiple points along a path, their position, and the target velocity at those points using a Hermite spline
- ▶ Same calculations as a Geometric controller with PID



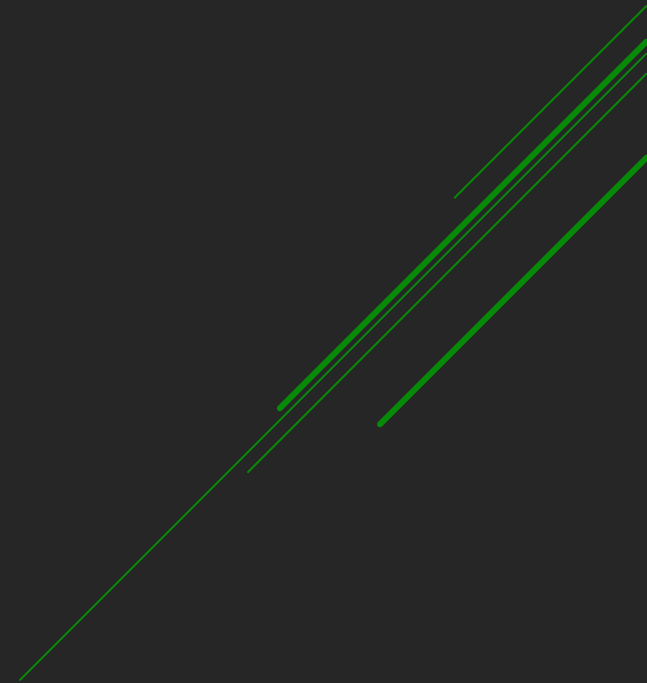
# WAYPOINT (TRAJECTORIES)

- ▶ Pros:

- ▶ Very precise
- ▶ Provides tremendous control over motion
- ▶ Allows for curved trajectories

- ▶ Cons:

- ▶ Very difficult to program, use pre-made libraries/applications to calculate paths
- ▶ Excessive for linear trajectories



# WHEN TO USE WHICH

- ▶ Ramp rate
  - ▶ Want to use standard position PID control (if it ain't broke, don't fix it)
  - ▶ Need to control acceleration and/or jerk in teleop
- ▶ Purely Geometric
  - ▶ Just...don't
- ▶ Geometric with PID
  - ▶ Want to have precise control of linear trajectories or of a single robot component (elevator, arm, etc.)
- ▶ Waypoint
  - ▶ Want to accurately move along curved trajectories (drivebase)

# WHAT WE'RE USING

- ▶ Pre-2020: Pure PID
  - ▶ High current draw from motors
  - ▶ Wheelies at start and (sometimes) end of motion
- ▶ 2020: Ramp rates
  - ▶ Pure PID, except we control the speedup period
  - ▶ Significantly lower current draw
  - ▶ Wheelies (sometimes) at end of motion
- ▶ 2021 Goal: Geometric with PID
  - ▶ Smooth, precise control both speeding up and slowing down
  - ▶ Lowest current draw
  - ▶ No wheelies
- ▶ 2021: Investigate Waypoints
  - ▶ Allows for curved trajectories instead of “drive straight then turn”

