Motion Planning and Control in FRC

Jared Russell
Tom Bottiglieri
Austin Schuh

Why are we here?

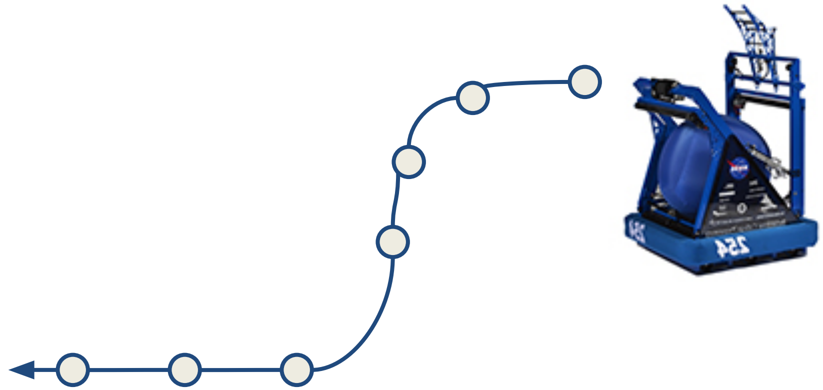
Robots should move less like this

And more like this

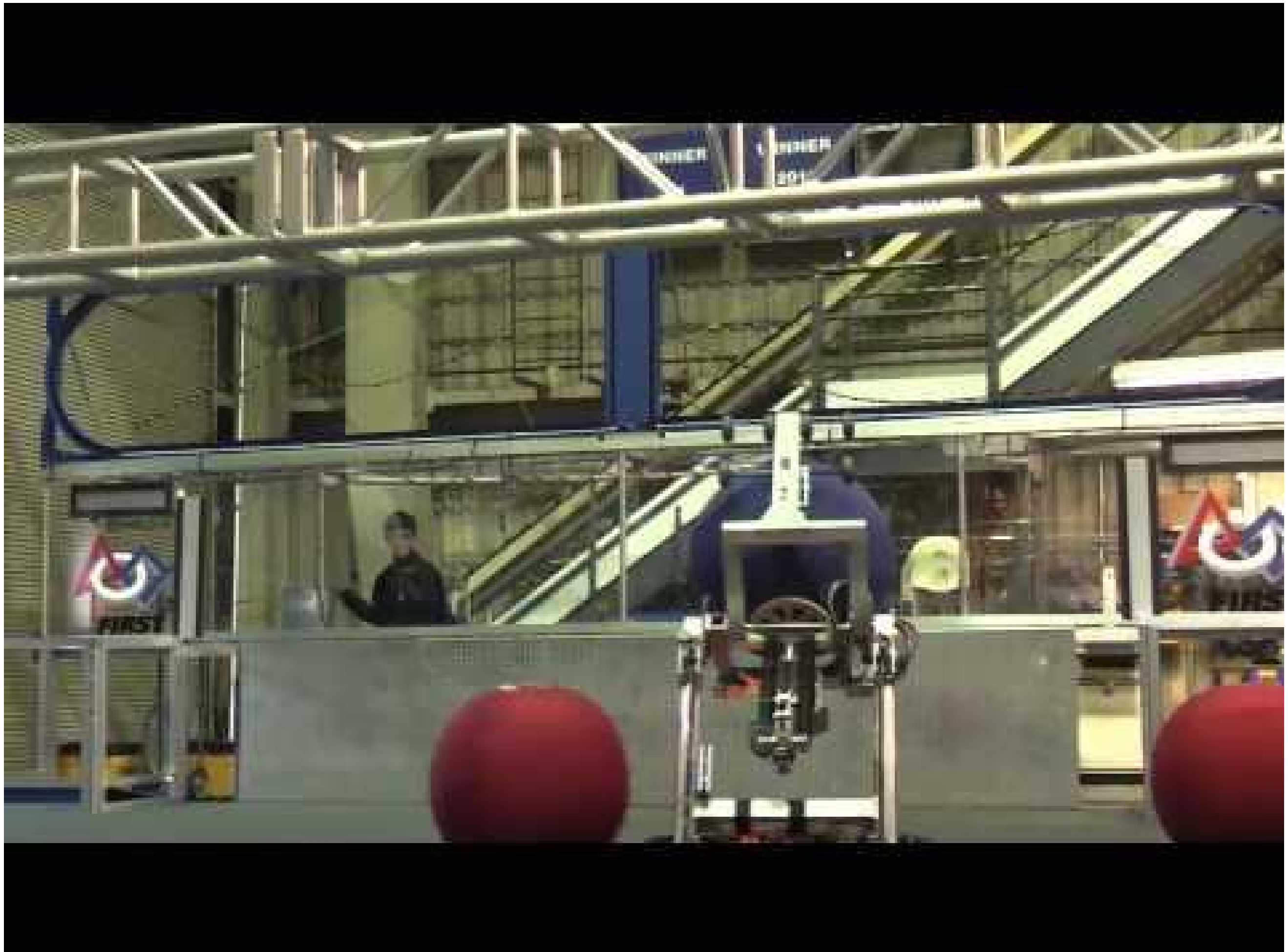
Why are we here?

- **Control systems are fun**
- Software is where most of the hard problems in robotics are
  - Many of the techniques discussed today are common in industry
- Motion control makes your FRC robot better!
- *Motivating examples*...

Go from here to there...



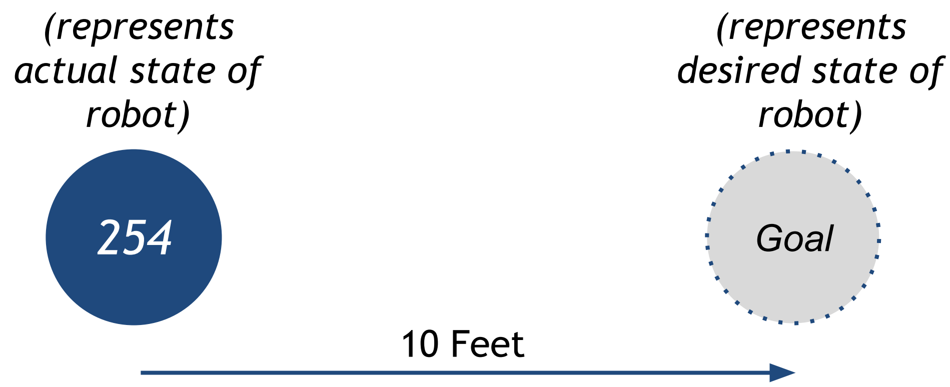
Coordinated motion...



Vision-driven targeting...

How can we do it?

Dead reckoning



Distance = 10 feet
Full Speed = 5 ft/sec

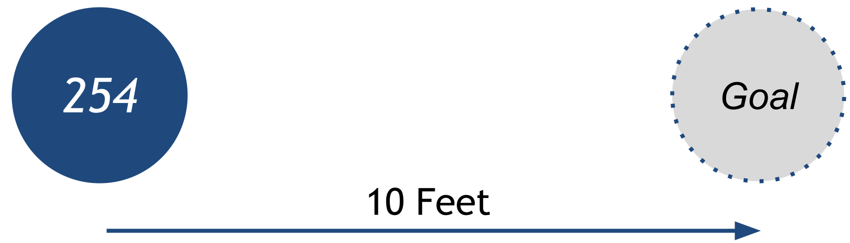
10 feet / 5 (ft / sec) = 2.0 seconds

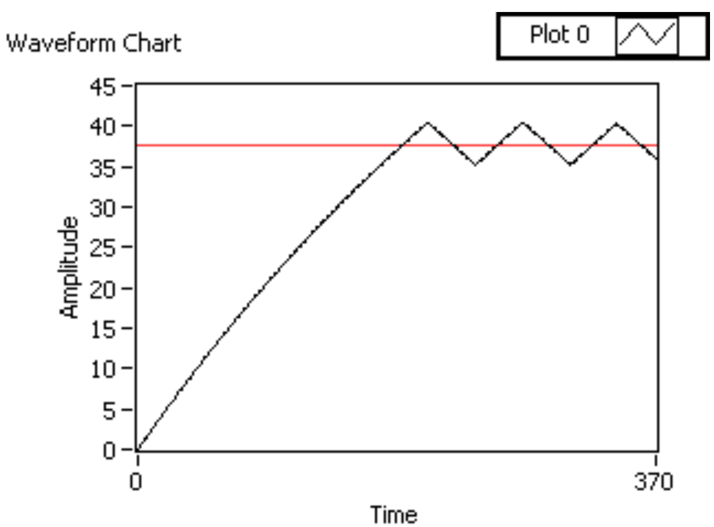
Just drive at full speed for 2.0 seconds and we'll be at the goal!

Let’s try using a sensor!

Dead reckoning

Bang-bang






```
while (true) {
  error = goal_distance - current_distance
  if (error > 0) {
    Drive(1.0);
  } else if (error < 0) {
    Drive(-1.0);
  } else {
    Drive(0.0);
  }
}
```

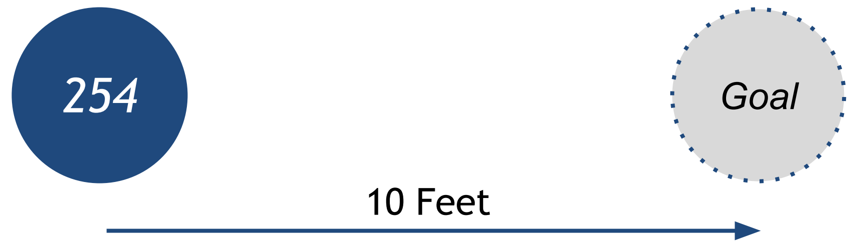
What is wrong with this code?

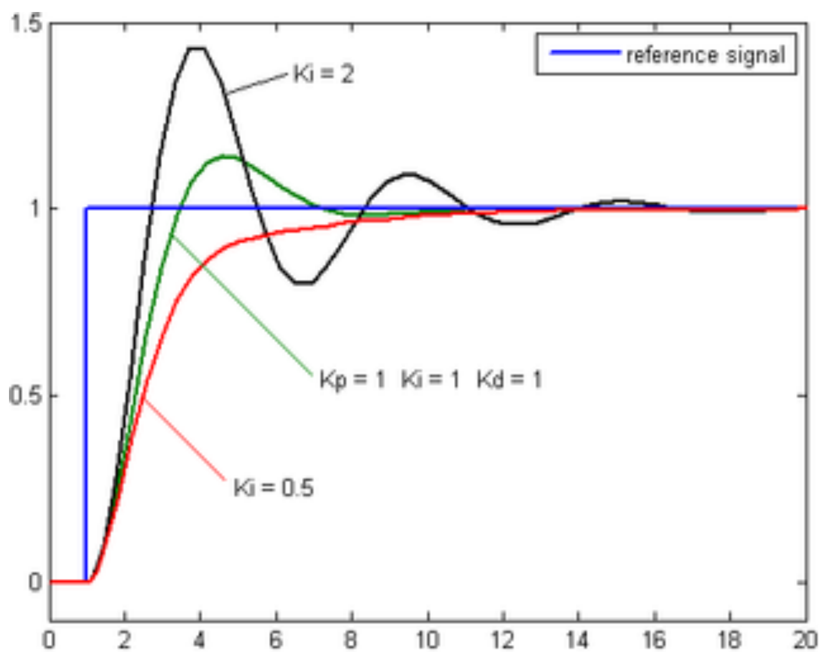
Slow down near the goal...

Dead reckoning

Bang-bang

PID





$$u(t) = \underbrace{K_p e(t)}_{\text{Proportional}} + \underbrace{K_i \int_0^t e(\tau)d\tau}_{\text{Integral}} + \underbrace{K_d \frac{d}{dt} e(t)}_{\text{Derivative}}$$

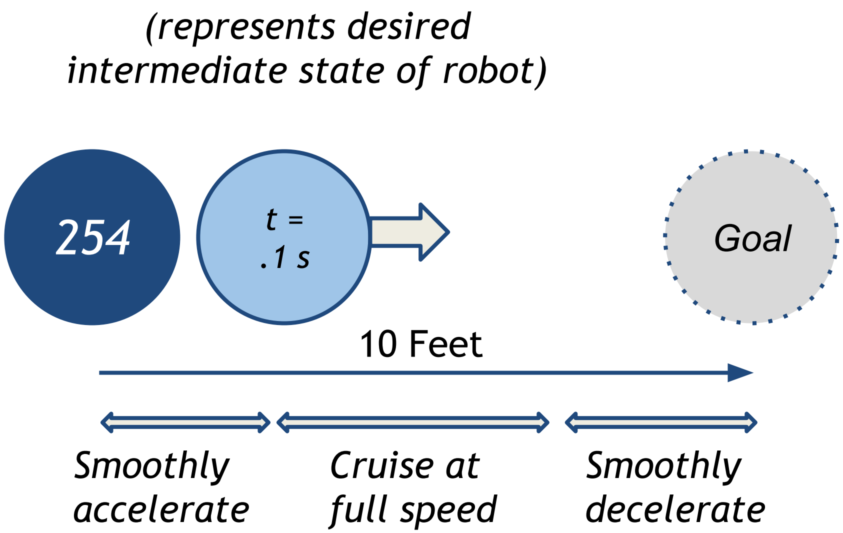
Slow down near the goal

Overcome friction

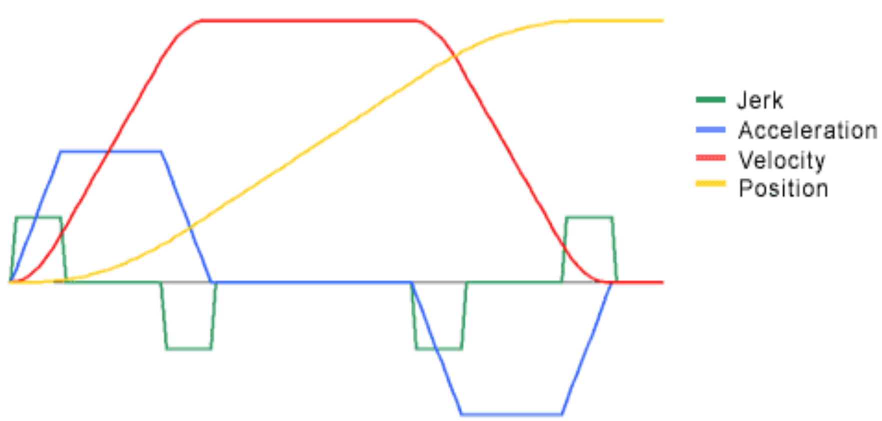
Add damping

Take it one step further...

Dead reckoning
Bang-bang
PID
Motion profiles

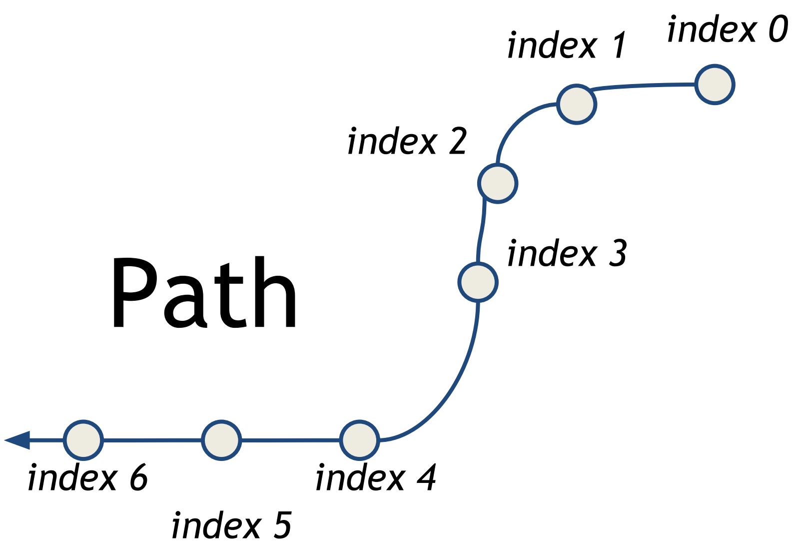


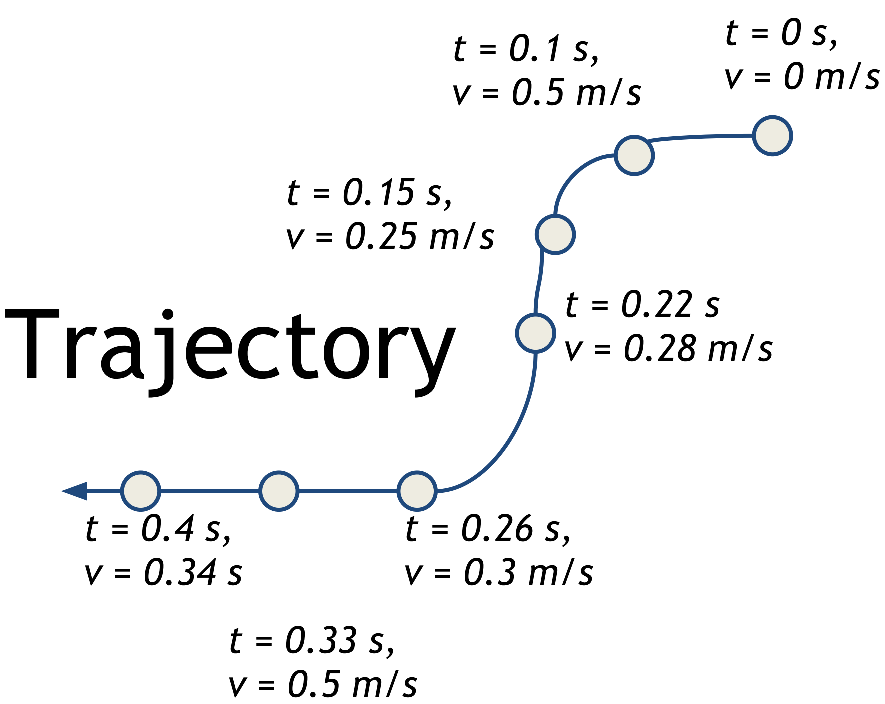
A well-planned profile makes precise control easier!



Some quick definitions...

Paths and Trajectories

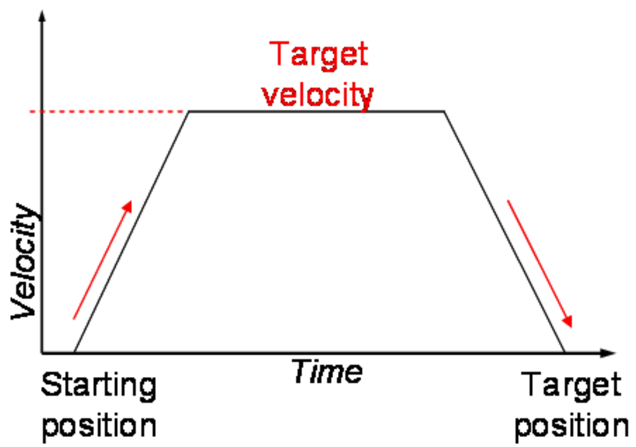


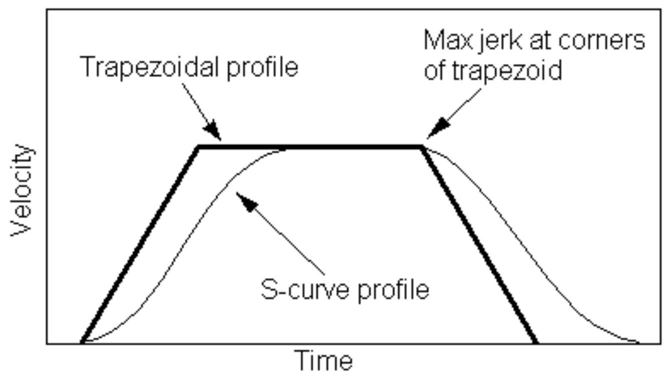


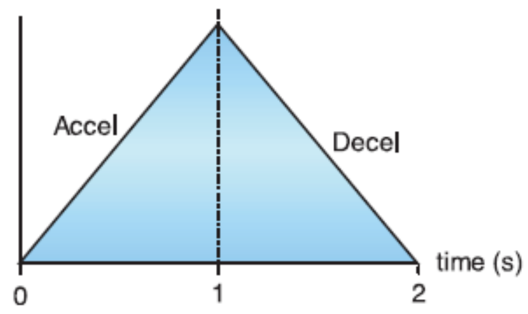
Some quick definitions...

Paths and Trajectories

Motion Profile

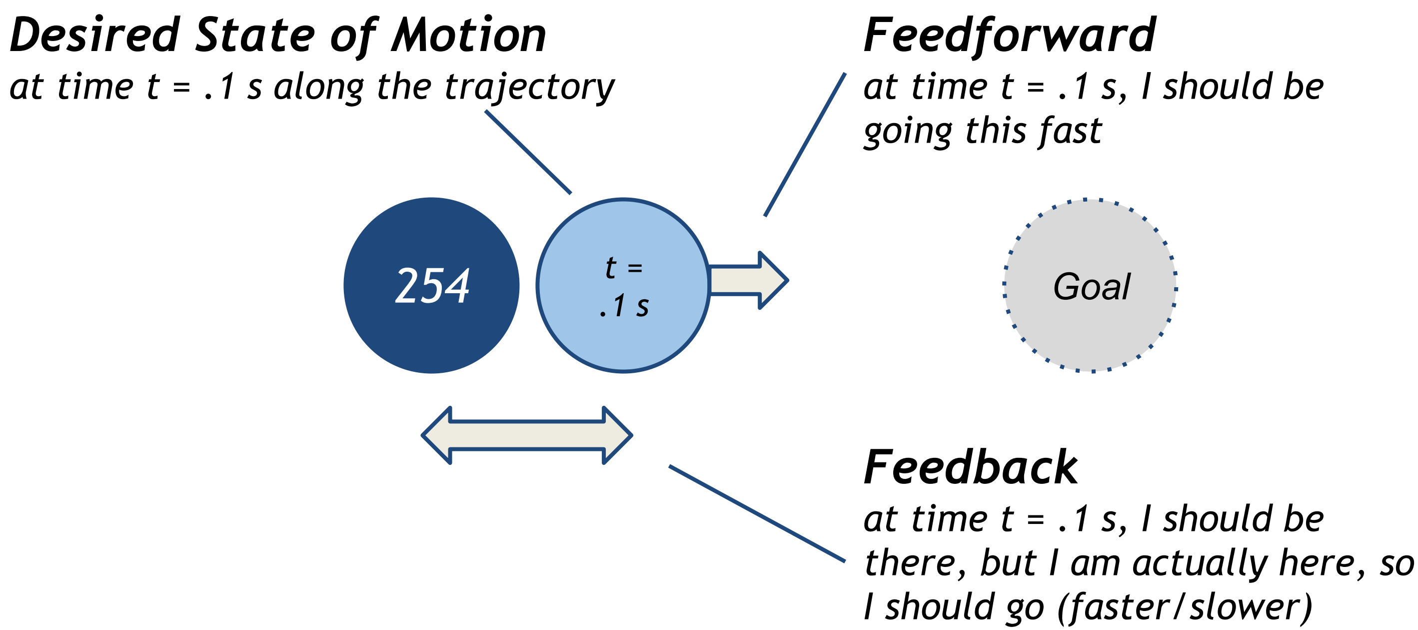






Some quick definitions...

Paths and Trajectories
Motion Profile
Feedforward and Feedback



Some quick definitions...

Paths and Trajectories
Motion Profile
Feedforward and Feedback
Forward and Inverse Kinematics

Forward Kinematics

Joint space → Configuration space

“My left wheel went forward 2 inches, my right wheel went forward 4 inches” → I went forward while turning counter-clockwise

Inverse Kinematics

Configuration space → Joint space

“I want to turn clockwise 90 degrees” → Left wheel must go forward for X inches, Right wheel must go backward for X inches

The Motion Control Process

1. Figure out where you want to go
2. Find a path
3. Find a trajectory
4. Follow the trajectory
   a. Figure out where you should be right now
   b. Feedforward control +
   c. Feedback control

Doing these sequentially is often simpler than doing them simultaneously

Path Planning

- What is the sequence of configurations the mechanism will move through between start and goal?
- The 1D case is usually trivial
- ...but the 2D (or more) case is interesting

Simple Path Planning

- Connect the dots - lots of ways to do it!
- Curve fitting
  - Cubic splines
  - Quintic splines
- Caveats
  - Loops
  - Overfitting
- Spline fit code:
  - [Team 254]
  - [Team 236]

Hermite Specification

2D (Hermite) cubic spline fit

Inputs:
Start (x, y) at s = 0
End (x, y) at s = 1

4 unknowns, 4 equations
Solve!

Equations:

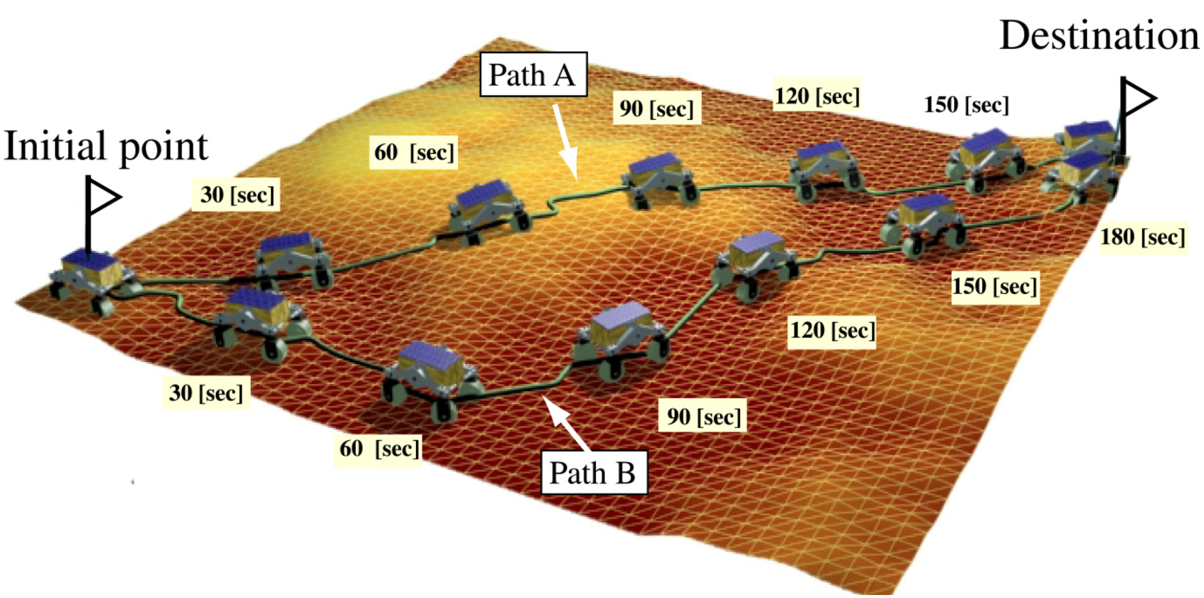
$$x(s) = A_x s^3 + B_x s^2 + C_x s + D_x$$
$$y(s) = A_y s^3 + B_y s^2 + C_y s + D_y$$

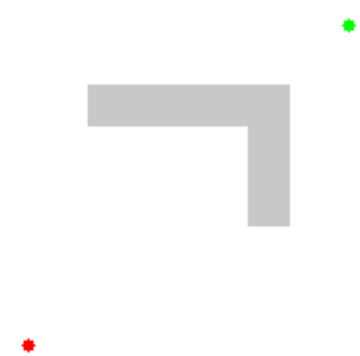
$$dx/ds(s) = 3A_x s^2 + 2B_x s + C_x$$
$$dy/ds(s) = 3A_y s^2 + 2B_y s + C_y$$

If you care about heading, you can choose these such that atan2(dy/ds(s), dx/ds(s)) = heading

Advanced Path Planning

- Active area of research in academia
- Applications in robotics, video games, optimization and routing, etc.





Trajectory Generation

- Ok, so we have a path. How fast should we traverse it? How do we:
  - Move to the goal as quickly as possible?
  - Obey constraints on maximum velocity, acceleration, (jerk)?
  - Know precisely where (and how fast) the mechanism should be moving at all points in time?
  - *Here is one such approach...*

Polynomial method for triangular or trapezoidal velocity profiles

- Remember your kinematics:

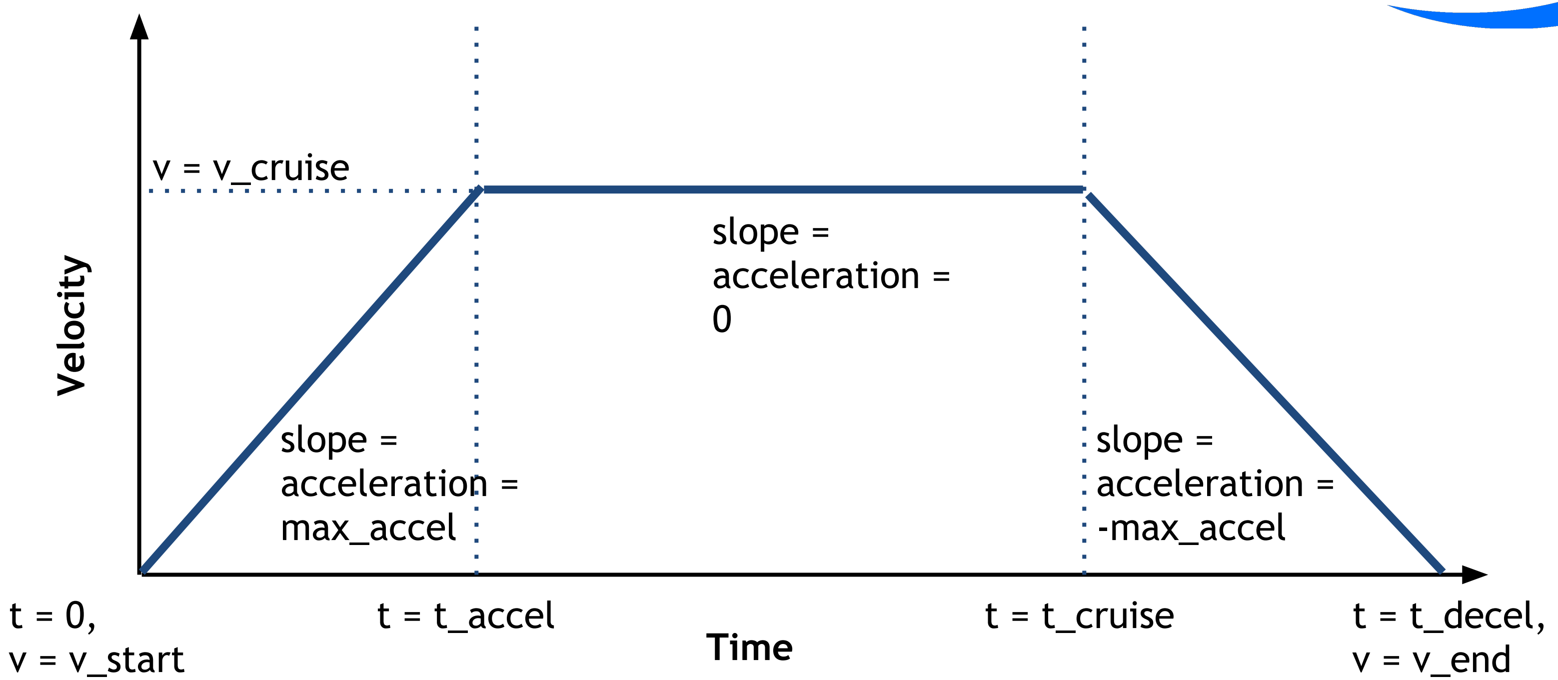
(1) $v_f = v_i + a\,t$

(2) $x_f = x_i + v\,t + \frac{1}{2}\,a\,t^2$

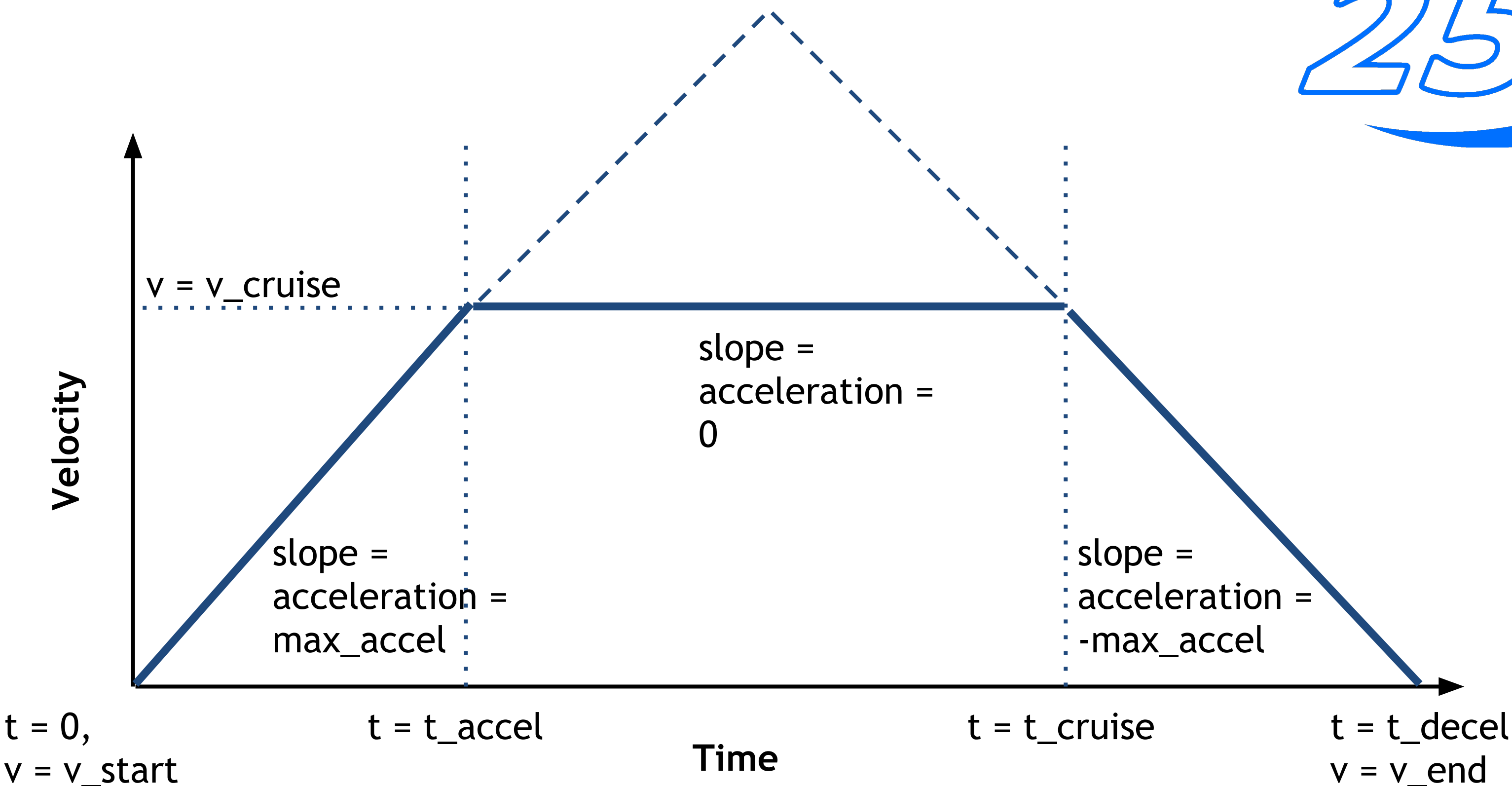
(3) $x_f = x_i + t\,(v_i + v_f)\,/\,2$

(4) $v_f^2 = v_i^2 + 2\,a\,(x_f - x_i)$

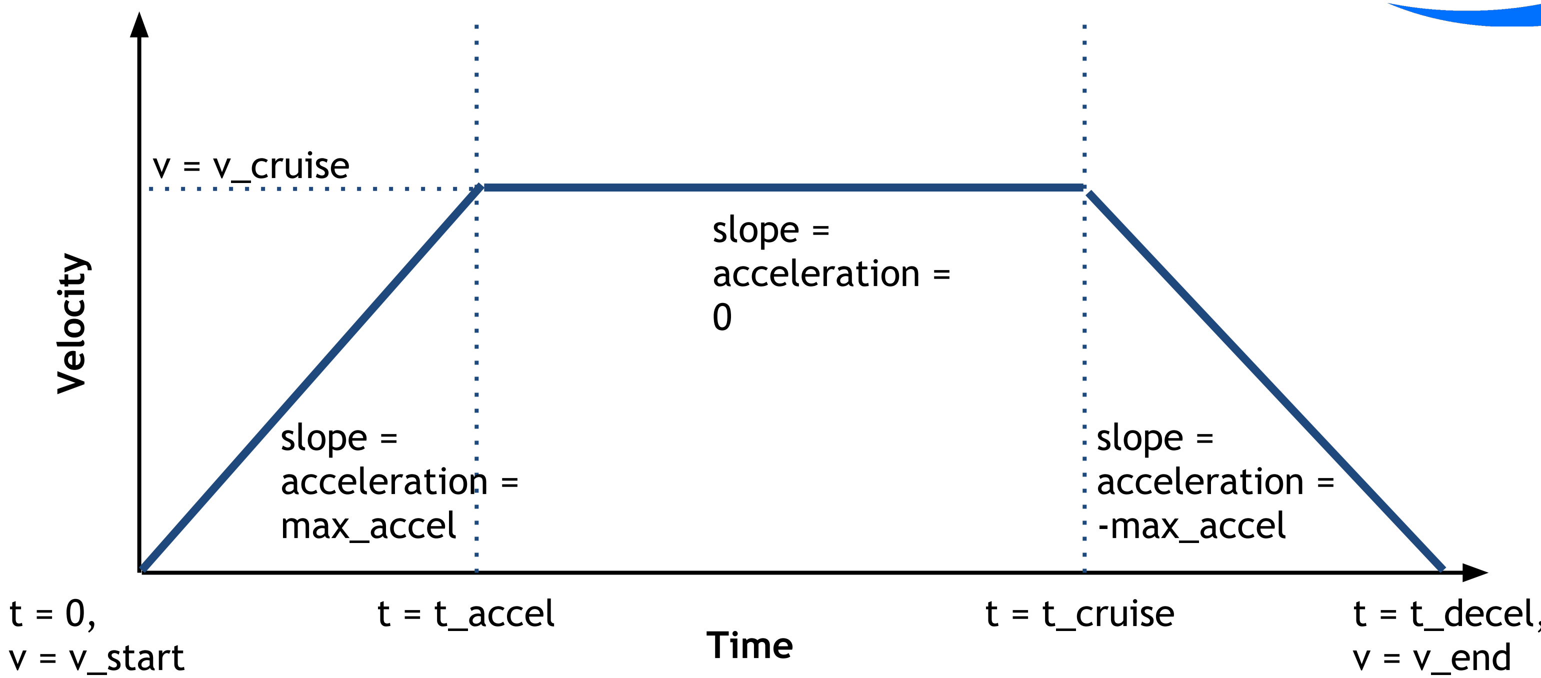
- There are up to three piecewise segments of motion in the motion (acceleration, cruise, deceleration)



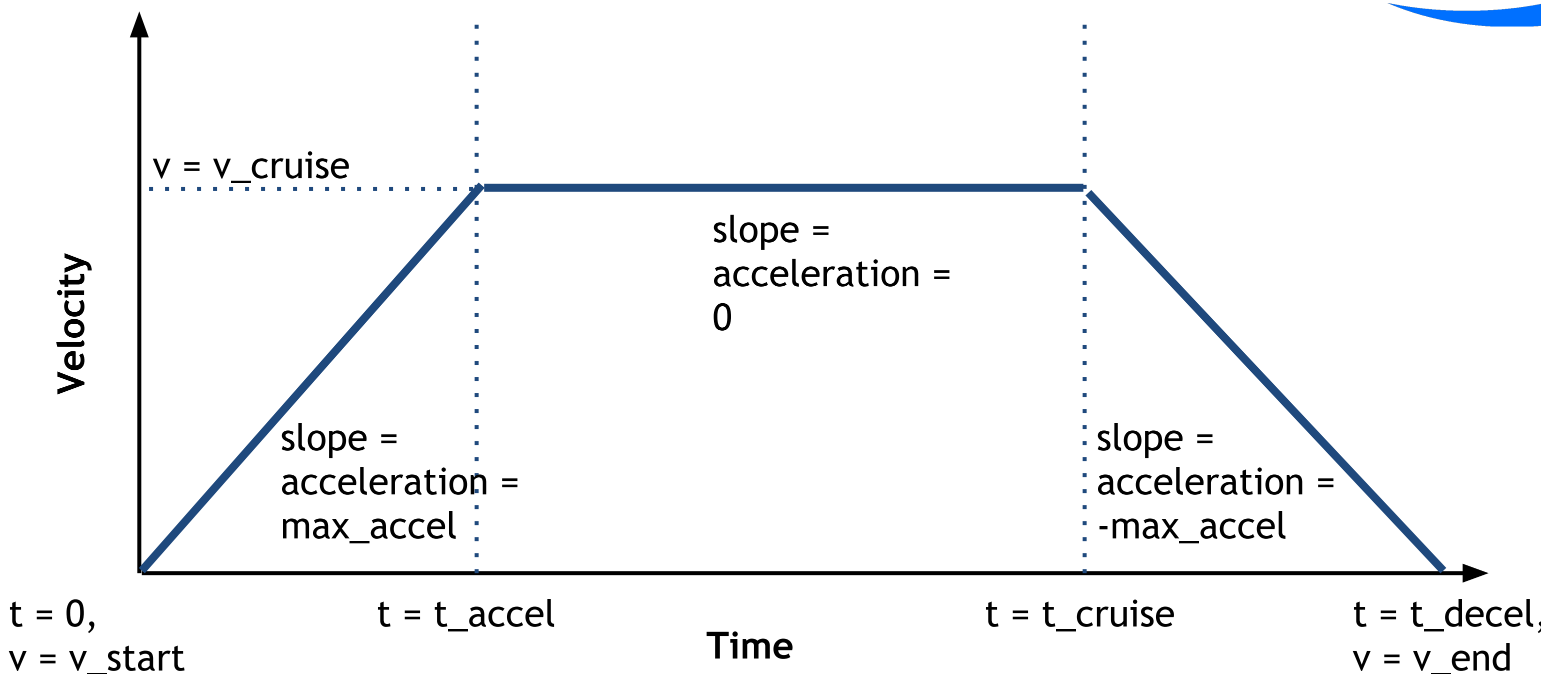
- First step: Find the cruising velocity v_cruise



- First step: Find the cruising velocity v_cruise
  - Pretend there is no velocity limit: How fast could we go? (hint: distance to accel to v_cruise plus distance to decel to v_end must equal total distance)
  - v_cruise = Min(max_velocity, v_cruise)



- Next:
  - figure out how much distance you will cover to accelerate from v_start to v_cruise (hint: $x_f = x_i + v\ t + \frac{1}{2}\ a\ t^2$)
  - ...and the same to decelerate from v_cruise to v_end



- Finally:
  - How long are you at cruise velocity?
  - Total distance = distance during accel + distance during decel + distance during cruise
  - Notice that cruise distance is zero in the case of a triangular profile

The Result

- We now have a function where we can lookup the desired position, velocity, and acceleration of the system for a given time t since the beginning of the motion

Alternative Approaches

- Boxcar Filter Method
  - Inspired by signal processing
  - Use a chain of integrators to model rate constraints
  - Pros:
    - Fast, Doesn’t require floating point math
    - Can deal with limited jerk, snap, etc.
  - Cons:
    - Cannot deal with constraints that change over time
    - Only work for 1D problems
  - This is how the Talon SRX does its motion profiling
  - More information in [this Chief Delphi thread]

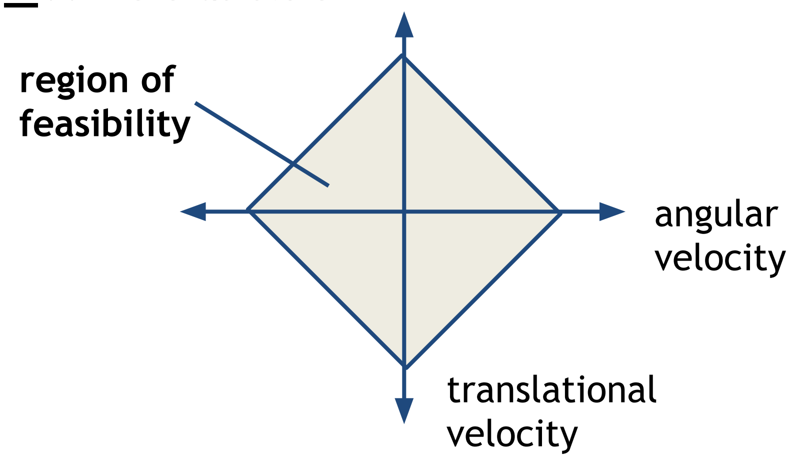
But that is a 1D trajectory…
…what about 2D+?

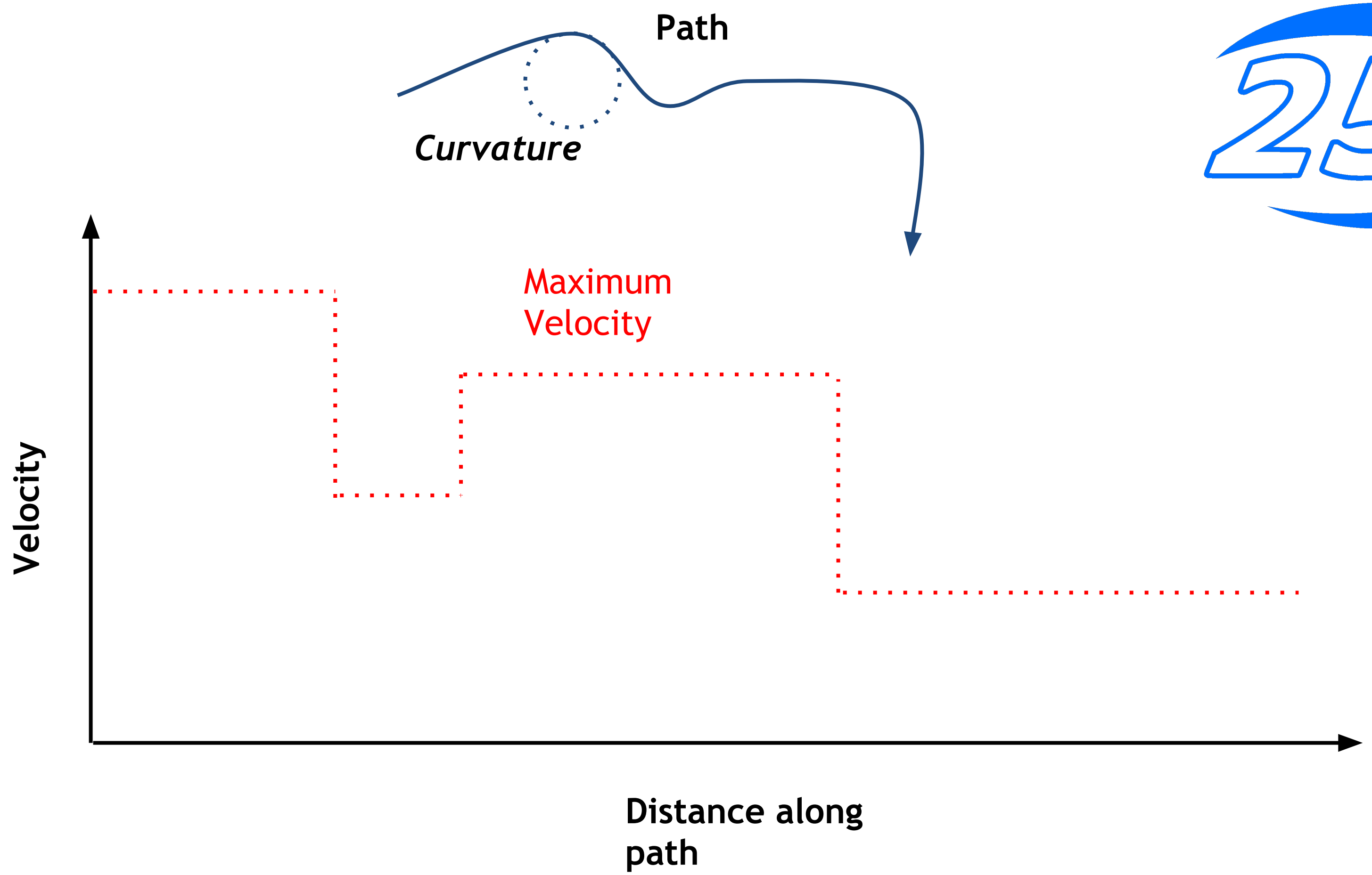
- One simple approach:
  - At time t, figure out the current distance the robot should have covered
  - Figure out the arc length of the spline corresponding to this distance

$$L = \int_{\alpha}^{\beta} \sqrt{\left(\frac{dx}{dt}\right)^2 + \left(\frac{dy}{dt}\right)^2}\, dt$$

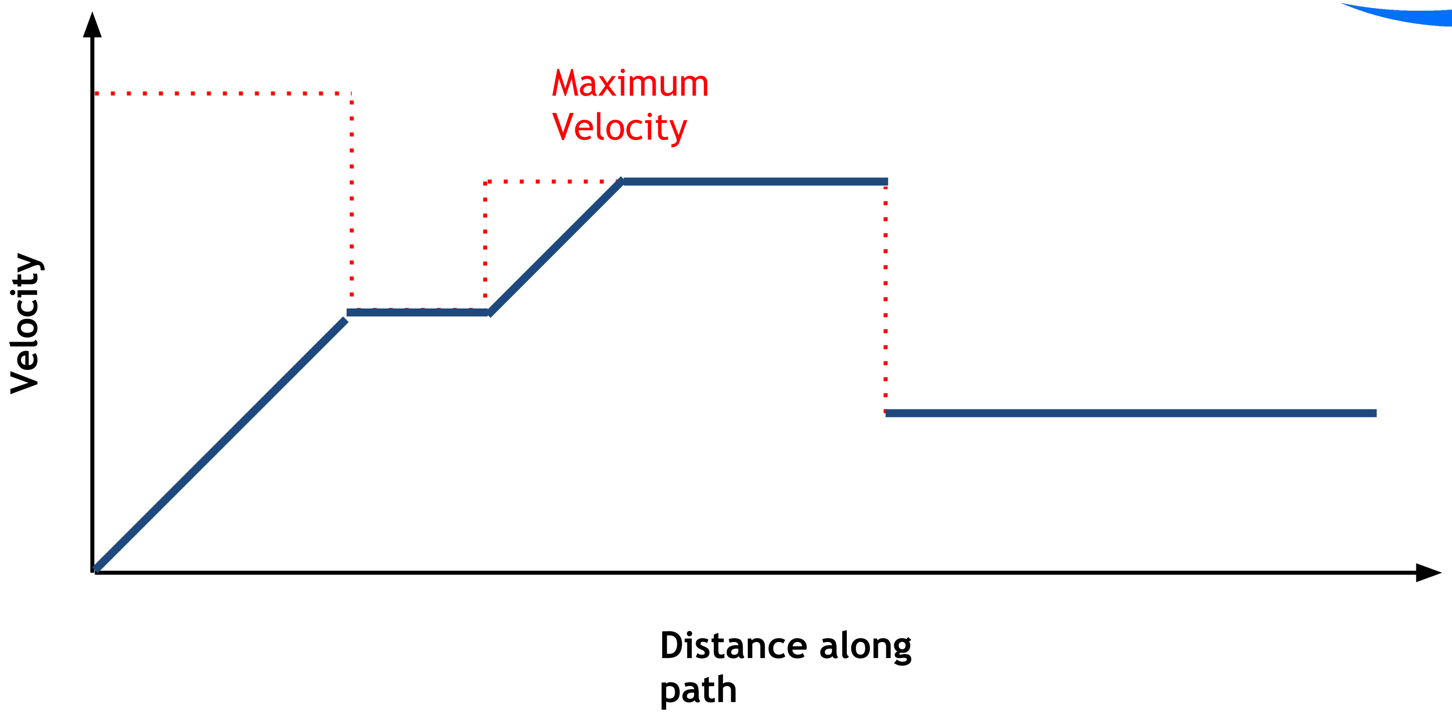
Non-constant Constraints

- A tank drive robot has coupled constraints
  - Can only translate so fast depending on curvature of path (because the outside wheel is the limiting factor)
  - v = (left_wheel_velocity + right_wheel_velocity) / 2
  - w = (right_wheel_velocity - left_wheel_velocity) / width_of_wheelbase

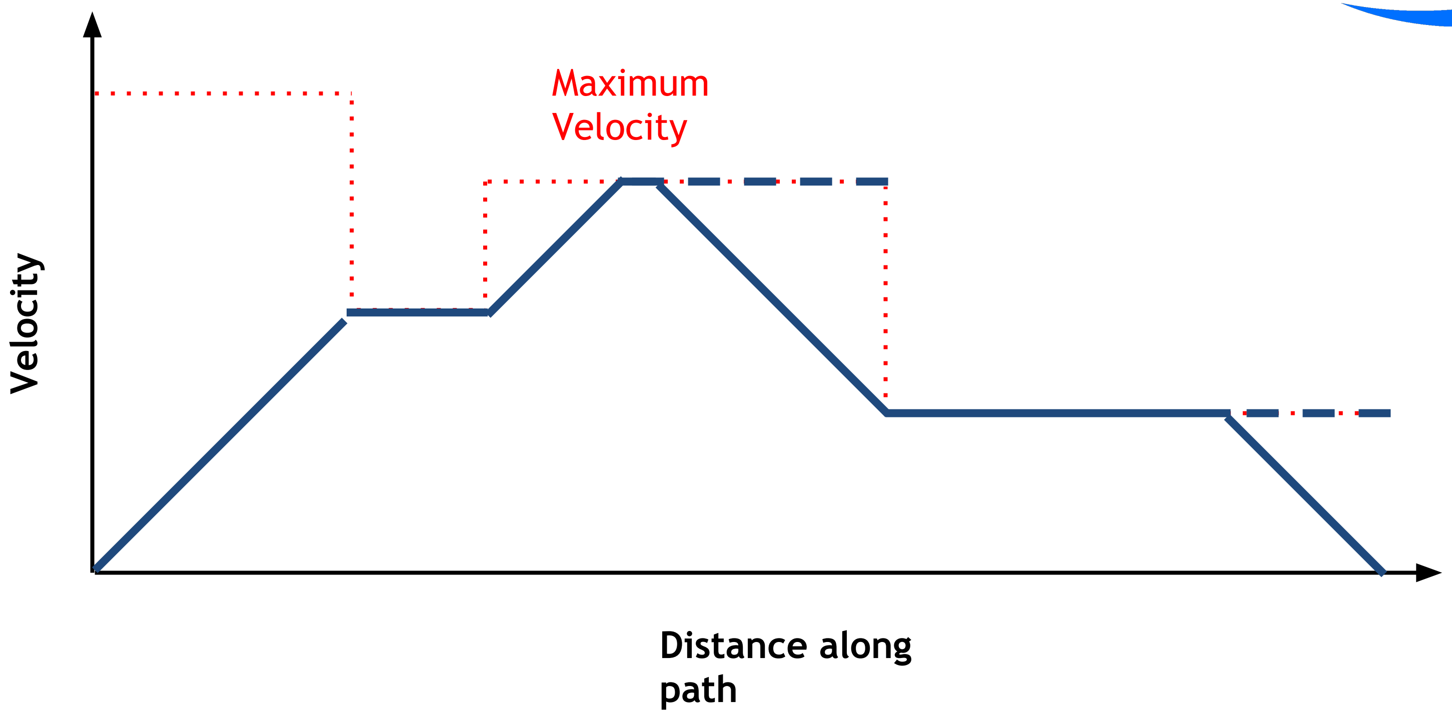




- Solution (Numerical Integration Approach):
    - Rather than computing switching times, divide the path up into discrete samples
    - For each sample, compute the maximum velocity based on curvature or other constraints



- Next, starting at the beginning state, assign a velocity to each point that is Min (max_velocity, max_reachable_velocity)



- One last step: Do the same thing *from end to beginning*

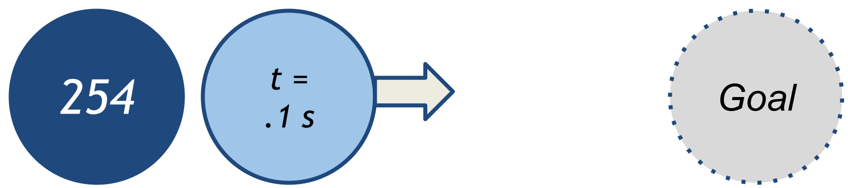
The Motion Control Process

1. Figure out where you want to go
2. Find a path
3. Find a trajectory
4. Follow the trajectory
    a. Figure out where you should be right now
    b. Feedforward control +
    c. Feedback control

Trajectory Following

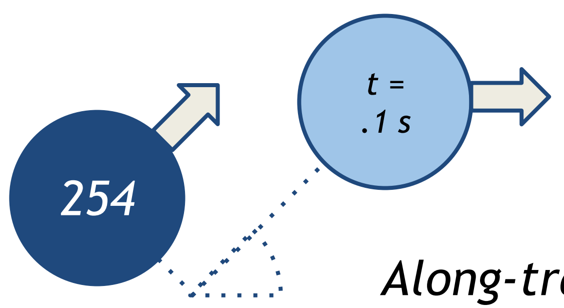
1. Figure out where you should be right now

1D case is easy




```
setpoint = LookUpSetpoint(t)
error = setpoint.pos - actual_position
```

2D navigation case



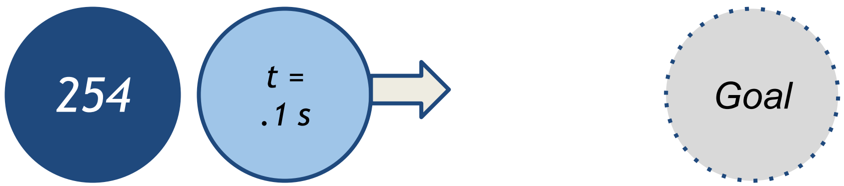
Along-track, cross-track, and heading components to error

Feedforward

- Basic Idea
  - If you have information about your system, tell your controller!
  - Trajectories encode all the information necessary to tell a “perfect” robot to follow them exactly
  - The feedback part will take care of deviations from perfection

Trajectory Following

2. Feedforward control



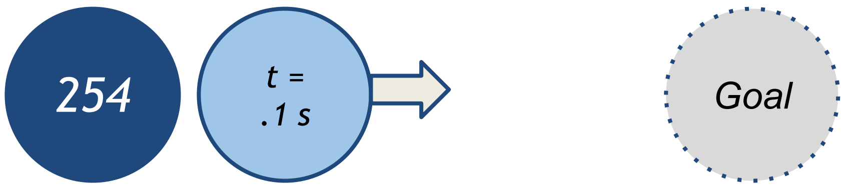

```
setpoint = LookUpSetpoint(t)
SetMotorSpeed(Kv * setpoint.vel)
```

“Kv” is a velocity constant, representing a unit conversion between real-world velocities and motor speeds

Ex. a robot drive is measured to move at 10 ft/sec at full power
The resulting Kv is then 0.1 (so that a velocity setpoint of 1.0 results in full power)

Trajectory Following

2. (Better) Feedforward control



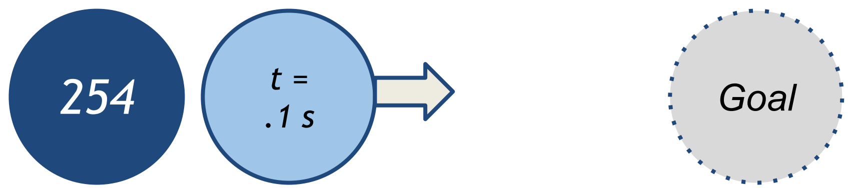

```
setpoint = LookUpSetpoint(t)
SetMotorSpeed(Kv * setpoint.vel +
              Ka * setpoint.acc)
```

"Ka" is an acceleration constant, telling your controller to add a little extra power to accelerate, and a little less to decelerate.

Easiest to tune by first tuning Kv, then experimenting with Ka.

Trajectory Following

3. Feedback control




```
setpoint = LookUpSetpoint(t)
error = setpoint.pos - actual_position
error_deriv = (error - error_last) / dt
SetMotorSpeed(Kv * setpoint.vel +
              Ka * setpoint.acc +
              Kp * error +
              Kd * error_deriv)
error_last = error
```

NOTE: The slides as presented contained an error here; there is no need to subtract setpoint velocity in this formulation, since error_last is relative to the old position setpoint.

These slides are now correct.

Integral usually isn't needed anymore, except sometimes at the very end

$$u(t) = \underbrace{K_p e(t)}_{\text{Proportional}} + \underbrace{K_i \int_0^t e(\tau)\,d\tau}_{\text{Integral}} + \underbrace{K_d \frac{d}{dt} e(t)}_{\text{Derivative}}$$

Tuning: Feedforward first

- Run the system at full speed and record a plot of position vs. time
  - Record maximum velocity
  - Record maximum acceleration (slope)
  - Kv = 1 / maximum velocity
- Create a motion profile with only Kv (all other constants set to 0)
- Adjust Ka until you track reasonably well

Tuning: Add the feedback

- Start with Kp…it can be really high, because errors are so small thanks to feedforward (*we typically track drive or elevator trajectories within an inch or two*)
- The feedforward makes the response less sensitive to Kp than in a pure feedback setup
- Most of the time only Kp is needed
- Add Kd if you aren't satisfied with tracking
  - We only use Kd on turning loops because of how sensitive they are

Best Practices

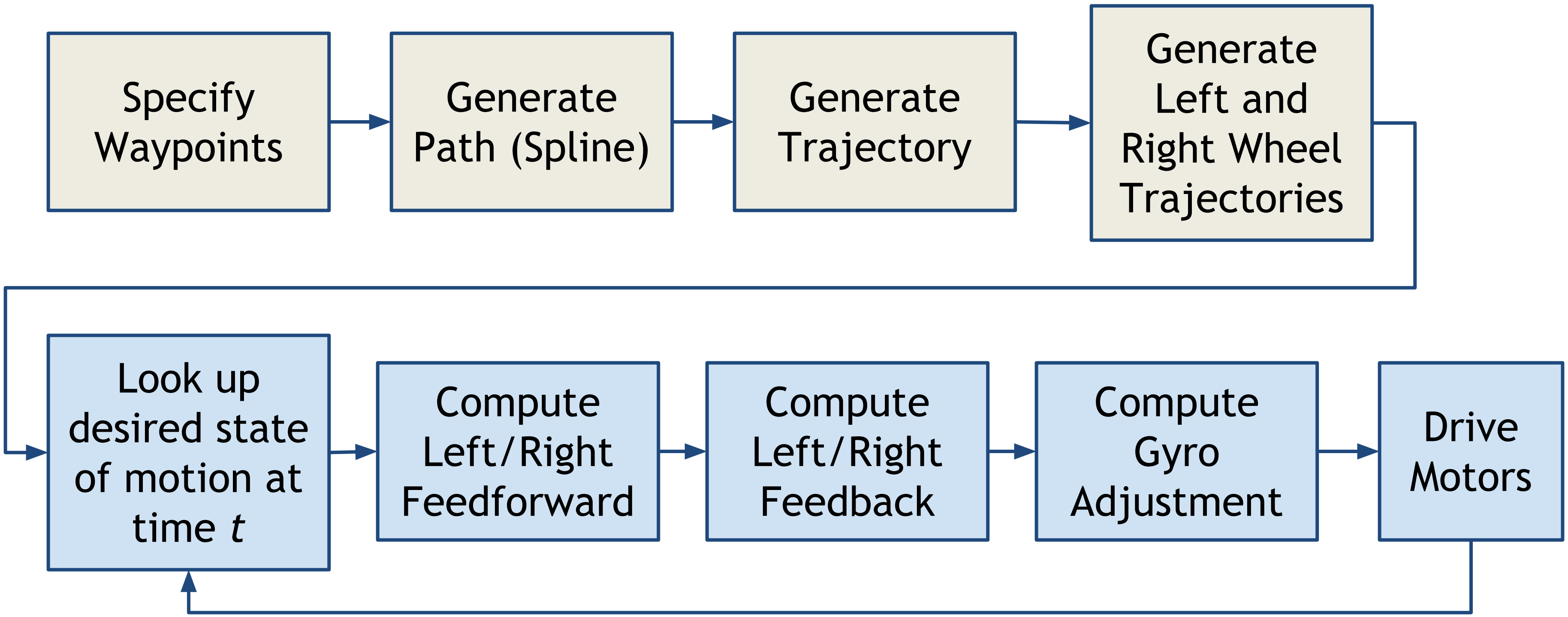
- Do the simplest thing possible!
- Mechanical robustness is important
- Visualization is key
- Good sensing makes everything easier
- Precise timing is important
- Linearity is important
- The Talon SRX can now do a lot of this for you!

Timing

- We run all of our loops in their own thread at 200 Hz (you can give this thread real-time priority in C++ with some hackery)
  - Why 200 Hz?
- 2015 Java Timer code: Uses Thread.sleep()!
- Monitor CPU usage to make sure you aren’t hitting 100% regularly
- **Always use a measured *dt*!**
  - getFPGATimestamp()

Putting it All Together

Simple Spline-following Drivetrain



Gyro adjustment in this case is just a proportional controller on the desired heading

Putting it All Together

Coordinated Motion

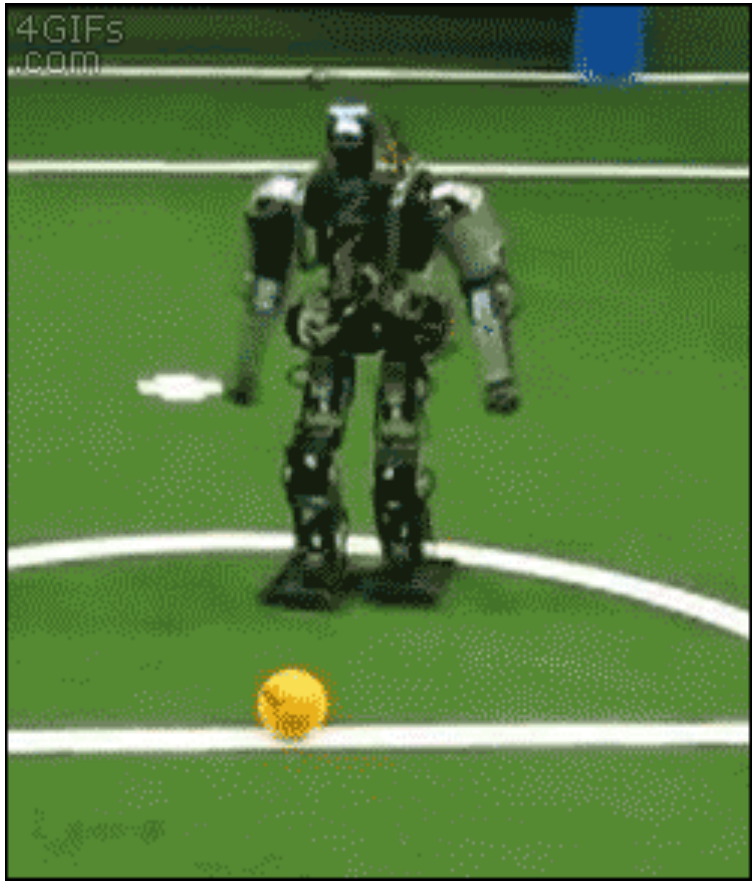
- Generate a separate trajectory for each rigid body
- Use the numerical integration approach to ensure that the trajectories stay synchronized
    - You can do this recursively in the control loop itself!

Putting it All Together

Vision-driven Position Control

- Cheap cameras make poor feedback signals
- Instead, use the camera to generate the goal position, and use separate sensors (gyro, encoders, potentiometer, etc.) to close the loop on position

Questions?



Now get out there and move!