Intro to Motion Planning in FRC

Ron Rossbach
1218 Vulcan Robotics
2607 Robovikings

10-Dec-2017

Background

- **What is motion planning?**

Motion planning (also known as the navigation problem or the piano mover's problem) is a term used in robotics for the process of breaking down a desired movement task into discrete **motions** that satisfy movement constraints and possibly optimize some aspect of the movement.

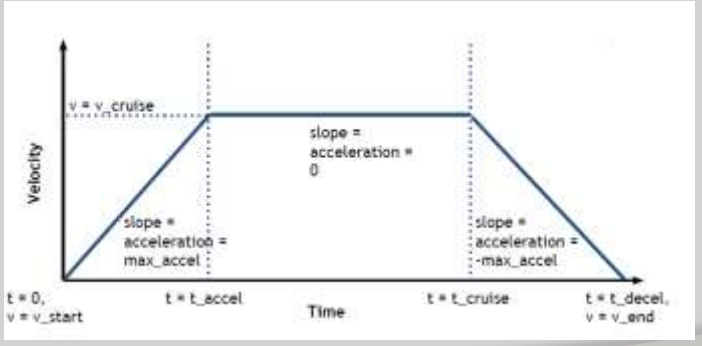
Motion planning - Wikipedia
https://en.wikipedia.org/wiki/Motion_planning

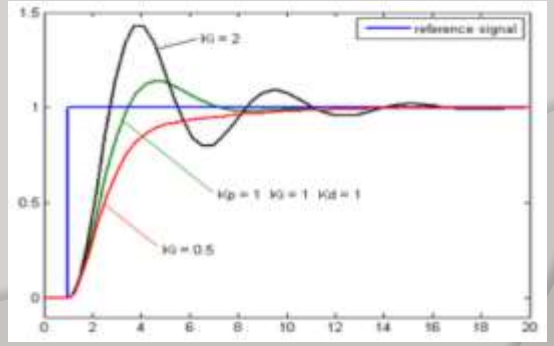
- **More simply: how can we move a mechanism smoothly and precisely to a desired destination?**
  - **"Mechanism" may be a drive train, an arm, an elevator, etc**
- **Sometimes also called motion profiling**

Background

- **Why do motion planning?**
  - **Much more accurate and repeatable than the alternatives:**
    - **Dead-reckoning: drive X speed T seconds then stop**
    - **Start/stop: drive X speed until encoder reads C counts, then stop**
    - **PID alone: overshoot and/or jerky motion**
  - **Allows complex paths (e.g. driving a path with curves)**

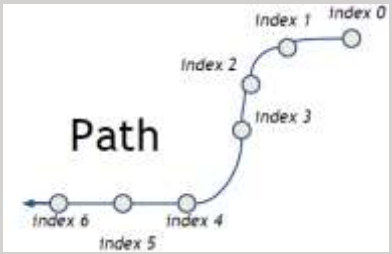
- **Motion planning + PID provides smooth acceleration, cruise, and smooth deceleration so you can arrive precisely at destination without overshoot/oscillation**





Basic Process

- **Determine your destination**
  - Could be pre-planned….like an autonomous routine
  - Or could be determined by other input….like vision!



"Drive through these points"

"Rotate left 20 degrees"

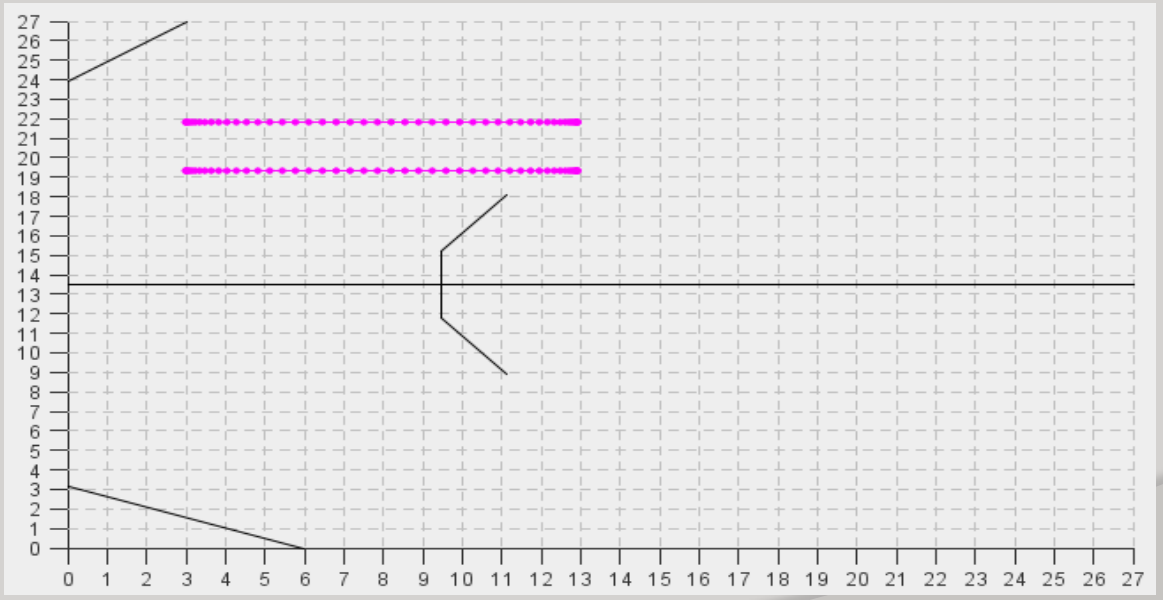
- **Calculate a path & trajectory**
  - Sequence of velocity/position points that will get you to your destination
- **Execute! (Follow the trajectory)**
  - Requires good control (PID, pursuit, etc)

Basic Process

- 1D and 2D motion profiling
  - 1D: movement in one dimension (e.g. arm, elevator, turret…)
  - 2D: two-dimensional movement (e.g. robot on the field)
    - Q: we want our robot to drive 5ft straight ahead; use a 1D or 2D profile?
- Robot drive bases: differential or swerve
  - Mecanum difficult because the wheels are inherently slippery
  - Therefore odometry (measuring how far we travel) is challenging for mecanum – except when going straight forwards/backwards
    - Other than straight forwards/backwards, there's not a simple relationship between encoder ticks and distance travelled with mecanum like there is with differential drive
- We'll use driving as the examples in the rest of this presentation, but the ideas work for other mechanisms

Determine Destination

- **Simple case: drive 10ft forward**
  - **Our desired path is the line between two points: (0,0) and (10,0)**
    - **The starting point and destination point**
  - **1D motion profile, since we don't need to turn – this means the wheels will always be moving at same speeds**



Calculate trajectory

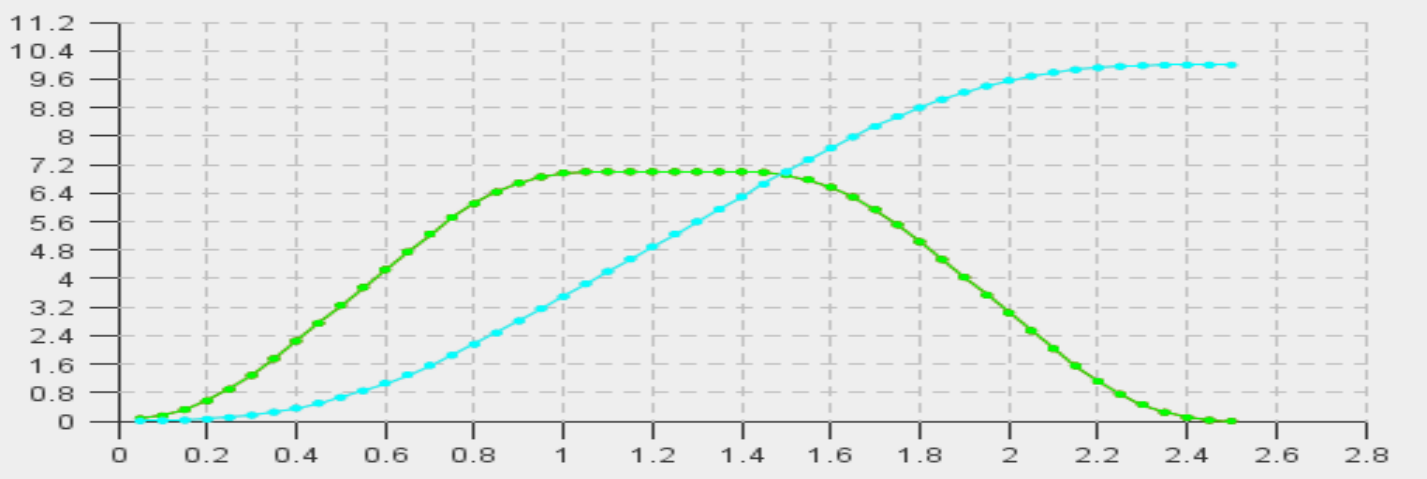
- Variety of tools to calculate trajectory
  - 1D: CTR Motion Profile Generator spreadsheet
    - https://github.com/CrossTheRoadElec/FRC-Examples/raw/master/Motion%20Profile%20Generator.xlsx
  - 1D or 2D: Team 254 (Cheesy Poofs) library
    - https://github.com/Team254/TrajectoryLib
  - Need to specify
    - Max velocity
    - Acceleration (how fast you get to max velocity)
    - Jerk (how fast the acceleration changes)

```
TrajectoryGenerator.Config cheesyConfig = new TrajectoryGenerator.Config();
cheesyConfig.dt = .05;              // the time in seconds between each generated segment
cheesyConfig.max_acc = 10.0;        // maximum acceleration for the trajectory, ft/s
cheesyConfig.max_jerk = 30.0;       // maximum jerk (derivative of acceleration), ft/s
cheesyConfig.max_vel = 7.0;         // maximum velocity you want the robot to reach for this trajectory, ft/s

double trackWidth = 29.872 / 12.0; // wheelbase width in ft
```

Calculate trajectory

```
p.addWaypoint(new WaypointSequence.Waypoint(0.0, 0.0, 0.0));
p.addWaypoint(new WaypointSequence.Waypoint(10.0, 0.0, 0.0));
```

- The resulting trajectory is a sequence of velocity & position
  - At time *t*, we want to be traveling at velocity *v* to position *p*
  - Adjust the parameters if needed to obtain a smooth trajectory



Execute!

- **Now, follow the trajectory!**
  - At each time interval *t,* drive the velocity/position indicated
- **This requires:**
  - Well-tuned velocity/position control of the mechanism
    - PIDF (PID with feedforward), adaptive pursuit, etc
    - A fast PIDF loop (like on the Talon SRX) with direct encoder input helps
      - But PIDF on the roboRIO with PWM motor controllers and DIO encoders also works
  - Robust mechanism
    - If your mechanism has lots of “slop”, belts skip, etc you won’t be able to follow accurately
  - Motion Profiling won’t fully compensate for mechanical issues
- **Software options:**
  - Robot program on roboRIO (requires threading)
  - Motion Magic mode in TalonSRX (for 1D profiles)
  - Motion Profile mode in TalonSRX

Execute!

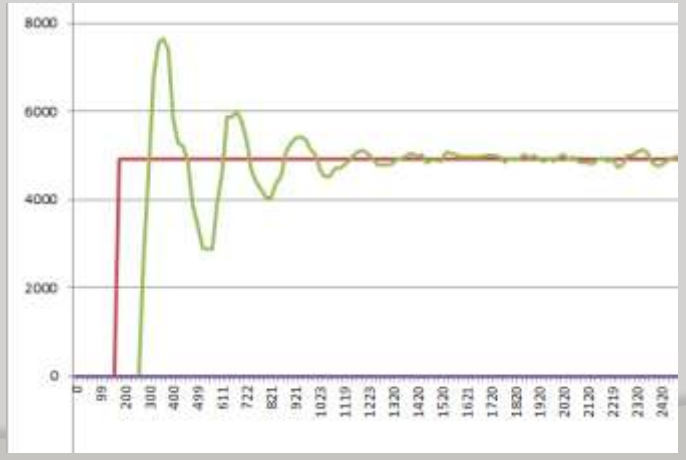
- Transfer trajectory (txt file) to roboRIO
- Read file in your roboRIO code
  - TrajectoryLib does this for you in Java
- Send the velocity commands to the drivetrain

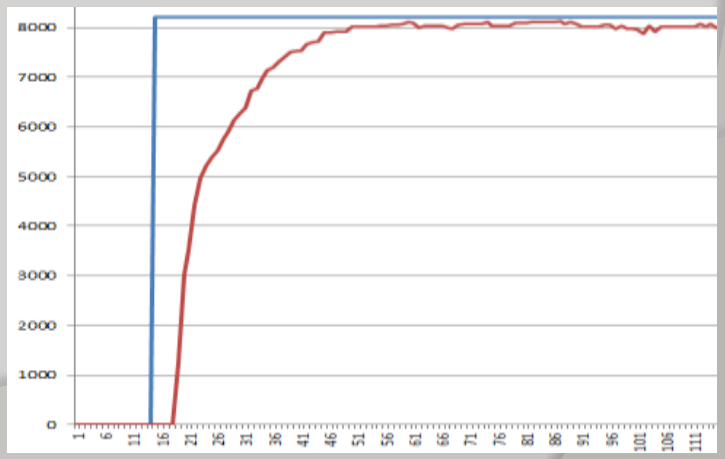
```
public void run() {
    if (firstTime) {
        firstTime = false;
        startTime = System.currentTimeMillis();
        running = true;
        done = false;
        leftMotors.enablePID(true,true);
        rightMotors.enablePID(true,true);
    }
    step = (System.currentTimeMillis() - startTime) / (long)(dtSeconds * 1000);
    //System.out.print("step: " + step);
    try {
        if (interrupt.get() == true) throw new Exception("Interrupting profile");
        if (runBACKWARDS){
            leftMotors.set(Constants.feetPerSecondToRPM(rightVelPts.get((int)step).vel));
            rightMotors.set(Constants.feetPerSecondToRPM(-leftVelPts.get((int)step).vel));

        } else {
            leftMotors.set(-Constants.feetPerSecondToRPM(leftVelPts.get((int)step).vel));
            rightMotors.set(Constants.feetPerSecondToRPM(rightVelPts.get((int)step).vel));
        }
    } catch (Exception e) {
        pointExecutor.stop();
        running = false;
        done = true;
        leftMotors.disablePID();
        rightMotors.disablePID();
        if (runBACKWARDS) runBACKWARDS = false;
    }
```

Execute!

- Some guides for PID tuning:
  - Talon SRX Motion Profile Reference Manual
    - (especially section 6.2)
  - Talon SRX Software Reference Manual
    - (especially section 12.4)
  - Tuning PID constants over a range (ChiefDelphi thread)

- While tuning, graph your setpoint and actual so you can visualize what you're doing

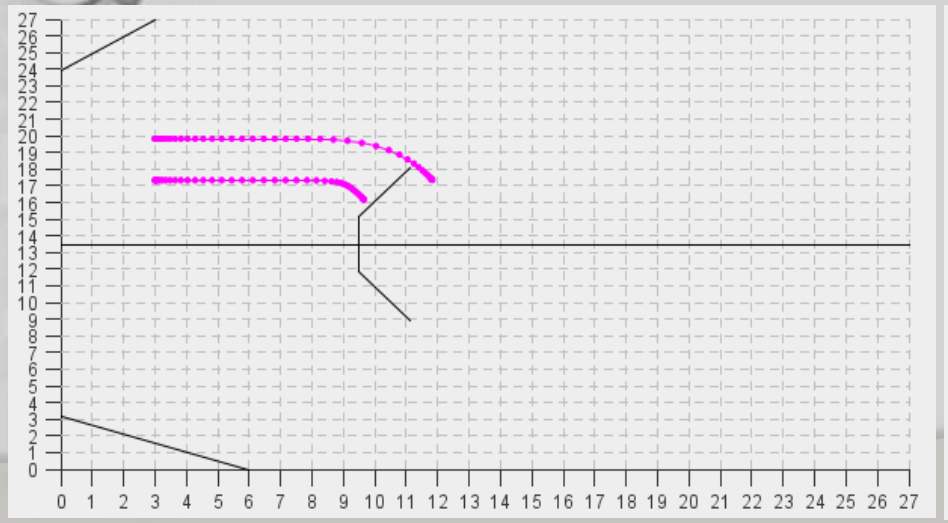


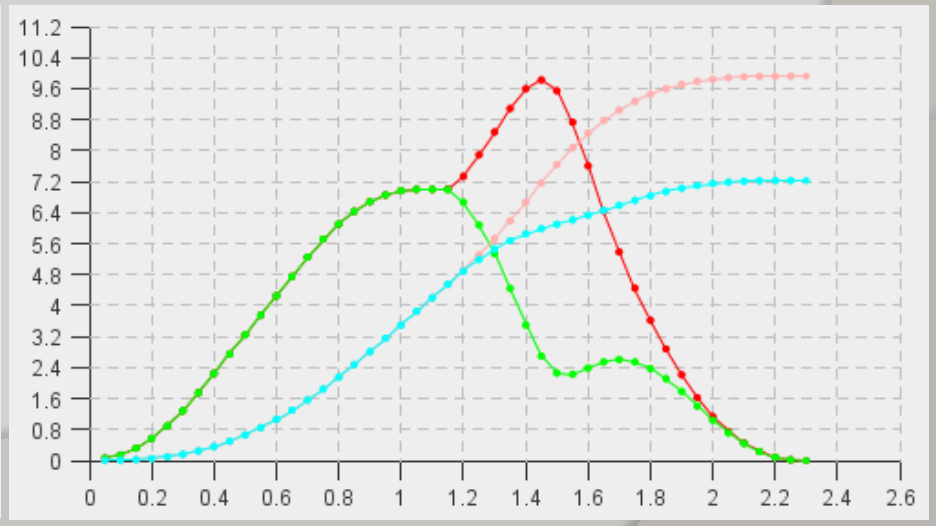


Other examples

- **Rotate in place….or rotate a turret….or move an arm**
  - **Also a 1D profile….for turning robot, just negate one side**
- **Drive a specific path on the field during auton**
  - **2D profiles….deliver gear to side peg….get fuel from hopper and shoot…etc**

```
WaypointSequence p = new WaypointSequence(10);
p.addWaypoint(new WaypointSequence.Waypoint(0.0, 0.0, 0.0));
p.addWaypoint(new WaypointSequence.Waypoint(4.5, 0.0, 0.0));
p.addWaypoint(new WaypointSequence.Waypoint(7.8, -1.8, 5.2));
```





Additional Resources

- Team 2607 github
  - SmoothPathPlanner: plan and visualize auton modes
    - uses 254's TrajectoryLib to calculate paths, and displays them on the screen using 2168's FalconPlot library
  - FRC2017-robotCode: STEAMWorks robot code (Java)
    - RobovikingDriveTrainProfileFollower: class to follow trajectory
    - Embedded Jetty webserver to display motor controller graphs during tuning
- ChiefDelphi has several good threads on motion planning and control
- Cross The Road Electronics documentation on TalonSRX modes
- Team 254's presentations at Worlds in 2016 and 2017

Questions etc

Feel free to e-mail with any questions!

Have fun!

rrossbach@midatlanticrobotics.org