Pose Control

Alexander Koldy

Forest Hills Robotics League

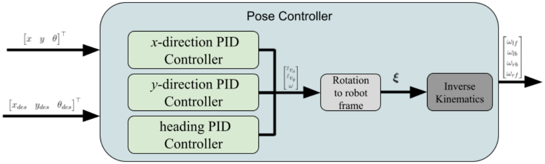
alexanderkoldy.ak@gmail.com

September 18, 2024

Introduction

In this lecture, we will discuss pose control in the context of the *FIRST Tech Challenge* (FTC).

Our **pose controller** will be in charge of moving our robot from one pose to another. We will refer to it as a **mid-level** controller, as its inputs may come from a higher-level controller (such as Pure Pursuit) or a planner; its outputs are sent to a lower-level drivetrain motor controller.


Pose Controller
$[x \;\; y \;\; \theta]^\top$
$[x_{des} \;\; y_{des} \;\; \theta_{des}]^\top$
x-direction PID Controller
y-direction PID Controller
heading PID Controller
$\begin{bmatrix} {}^f v_x \\ {}^f v_y \\ \omega \end{bmatrix}$
Rotation to robot frame
ξ
Inverse Kinematics
$\begin{bmatrix} \omega_{lf} \\ \omega_{lb} \\ \omega_{rb} \\ \omega_{rf} \end{bmatrix}$

Note: this type of control scheme is also sometimes referred to as "PID to point" or "p2p" informally.

Pose control

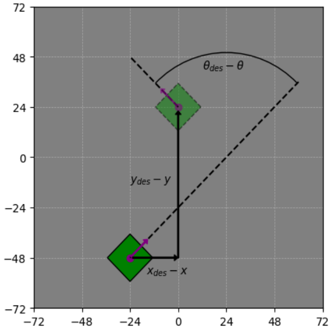
Our inputs to the pose controller are our pose vector (coming from our localizer/estimator):

$$q = \begin{bmatrix} x & y & \theta \end{bmatrix}^\top$$

and our desired pose vector (coming form our planner/higher level controller):

$$q_{des} = \begin{bmatrix} x_{des} & y_{des} & \theta_{des} \end{bmatrix}^\top$$

In the field frame, we can see the error in our position and heading between our current pose and desired pose (faded/dashed green box).



Pose control

We will start by creating three separate PID controllers, one for each dimension of our pose vector. To simplify things, let's write each PID as a function of the current state variable and desired state variables:

$$\begin{aligned} {}^{f}v_x &= \mathrm{PID}_x(x, x_{des}) \\ {}^{f}v_y &= \mathrm{PID}_y(y, y_{des}) \\ \omega &= \mathrm{PID}_\theta(\theta, \theta_{des}) \end{aligned}$$

It is worth noting that each of these PID controllers may have their own set of k_p, k_i and k_d values. Moreover, some of these values may simply be set to 0 depending on tuning and usage. In code, there should be a total of 9 tune-able values.

The output of the PID controllers gives us velocities in the field frame (note that we are choosing this to be our output, i.e., we choose the unit of each gain to be s^{-1}). We will want to convert these velocities to a twist in the robot frame, since our motors are physically on the robot.

Pose control

We apply the rotation $\mathbf{R}_{f \to r}$, rotating our velocity vector (PID output) to the robot frame:

$$\boldsymbol{\xi} = \mathbf{R}_{f \to r} \begin{bmatrix} {}^{f}v_x \\ {}^{f}v_y \\ \omega \end{bmatrix}$$

Now, we must convert these robot velocities into wheel velocities. We apply the inverse kinematics matrix to yield wheel speeds:

$$\mathbf{u} = \mathbf{H}_{ik}\boldsymbol{\xi}$$

where $\mathbf{u}$ is a vector of wheel speeds to be sent to the drivetrain motor controller. We can think of these as "desired" wheel speeds for our feedforward controller to follow!

Tuning

Tuning the pose controller can get difficult as there are technically 9 gains to tune. However, setting all three k_i values to 0 should ease the tuning process. The lower-level drivetrain motor controller should be able to handle friction issues via its k_s term. This should help reduce steady-state error.

Typically, we tune the x and y controllers together as they should have similar, if not the same gains. Mixing in the heading controller will likely lead to strange behavior. Observe which of the three controllers is most overpowering the others and make changes accordingly. A good way to test this is to start at $\begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^\top$ and set a desired pose of $\left[48, 48, \frac{pi}{2}\right]^\top$ (assuming units of inches and radians). From this, we can see if the robot is forcefully turning before moving in the x, y direction or vice versa. Movements in all three dimensions of the pose should happen essentially simulataneously.

Refer to the tuning guide from the PID lecture (ignoring the integral gain if necessary). It is worth noting that k_d might not be necessary. Moreover, we should ensure that the low-level motor controller is tuned properly first.