Robotics
Exercise 5

Marc Toussaint
Machine Learning & Robotics lab, U Stuttgart
Universitätsstraße 38, 70569 Stuttgart, Germany

November 7, 2014

In the lecture we discussed PD force control on a 1D point mass, which leads to oscillatory behavior for high K_p and damped behavior for high K_d. Slide 03:15 replaces the parameters K_p, K_d by two other, more intuitive parameters, λ and ξ: λ roughly denotes the time (or time steps) until the goal is reached, and ξ whether it is reached agressively ($\xi > 1$, which overshoots a bit) or by exponential decay ($\xi \leq 1$). Use this to solve the following exercise.

1 PD force control on a 1D mass point

a) Implement the system equation for a 1D point mass with mass $m = 3.456$. That is, implement the Euler intergration (see below) of the system dynamics that computes x_{t+1} given x_t and u_t in each iteration, where $x_t = (q_t, \dot{q}_t)$.

There is no need for the robot simulator code—implement it directly in plain C++ or another language.

Assume a step time of $\tau = 0.01$sec. Generate a trajectory from the start position $q_0 = 0$ and velocity $\dot{q}_0 = 0$ that approaches the goal position $q^* = 1$ with high precision within about 1 second using PD force control. Find 3 different parameter sets for K_p and K_d to get oscillatory, overdamped and critical damped behaviors. Plot the point trajectory (e.g. using the routine `gnuplot(arr& q); MT::wait();` .)

b) Repeat for time horizon $t = 2$sec and $t = 5$sec. How should the values of K_p and K_d change when we have more time?

c) Implement a PID controller (including the integral (stationary error) term). How does the solution behave with only K_i turned on ($K_p = K_d = 0$); how with K_i and K_d non-zero?

Note on Euler Integration: See also http://en.wikipedia.org/wiki/Euler_method. Given a differential equation $\dot{x} = f(x, u)$, Euler integration numerically "simulates" this equation forward by iterating

$$x_{t+1} = x_t + \tau f(x, u)$$

for small time steps τ.

2 Following a reference trajectory with PD control

We have the same point mass as above. But now the goal position q^* changes over time: we have a reference trajectory

$$q^*(t) = \cos(t)$$

How can one use a PD controller to let the point mass robustly and *precisely* follow this reference trajectory?

Simulate the problem as above, but at every full second apply a randomized impuls (=instantaneous change in velocity) on the point mass. That is, simply set $\dot{q} \leftarrow \dot{q} + \xi$ for $\xi \sim \mathcal{N}(0, .1)$. (In code: `double xi = 0.1 * rnd.gauss();`) Generate oscillatory, overdamped and critical recovery behaviors.