CS 545 Spring 2017 Homework 1 Template Solution

James Preiss

February 20, 2017

a) First-order planner Simulink model

a) i

For the system

$$\dot{x} = \frac{\alpha}{\tau}(x_f - x) \tag{1}$$

we take the Laplace transform, treating x_f as the input, and form the transfer function:

$$\begin{aligned} sX(s) - x_0 &= \frac{\alpha}{\tau}(X_f(s) - X(s)) \\ \left(s + \frac{\alpha}{\tau}\right) X(s) &= \frac{\alpha}{\tau} X_f(s) + x_0 \\ X(s) &= \frac{\frac{\alpha}{\tau} X_f(s) + x_0}{s + \frac{\alpha}{\tau}}. \end{aligned} \tag{2}$$

For the given $\alpha = 1, \tau = 1, x_0 = 0$, this reduces to:

$$X(s) = \frac{1}{s+1} X_f(s) \tag{3}$$

a) ii

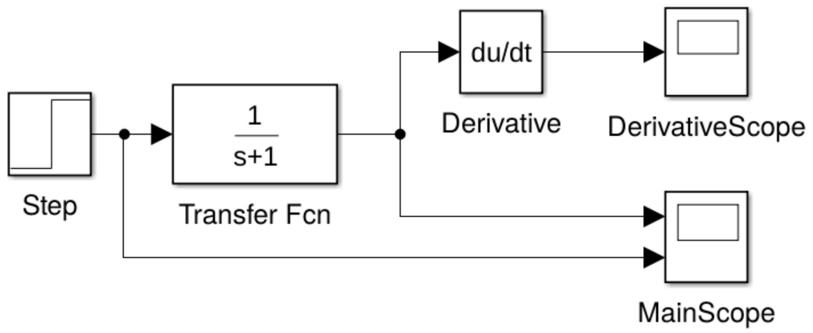


Figure 1: Simulink model implementing the dynamical system of part a).

a) iii

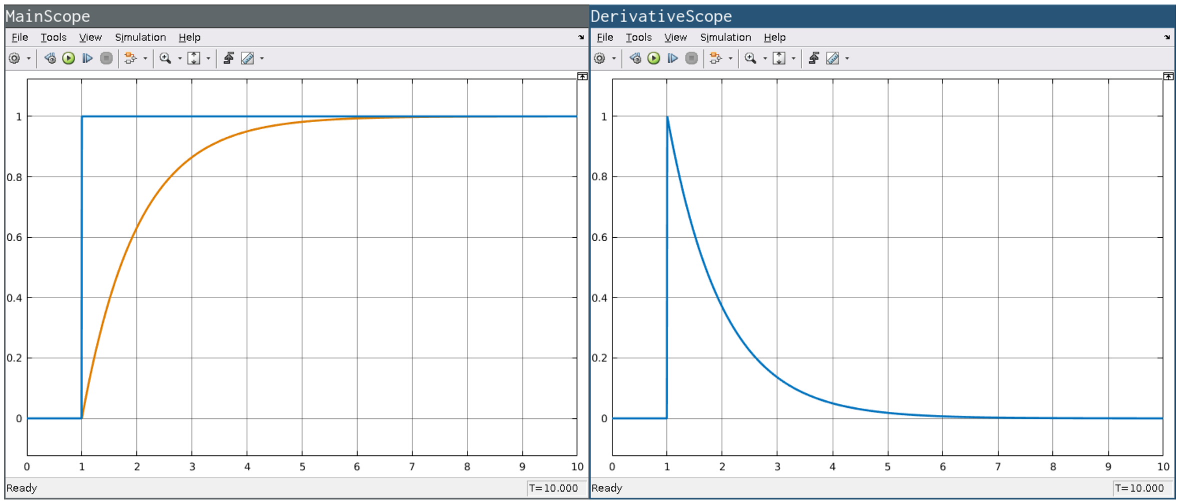


Figure 2: *Left:* Oscilloscope traces of x_f step input (blue) and resulting trajectory of x (orange). *Right:* Oscilloscope trace of system velocity $\dot{x}$.

b) Tuning parameters of first-order planner

b) i α tuning

A value of $\alpha = 7$ makes the system coverge to within 99% of x_f within one second:

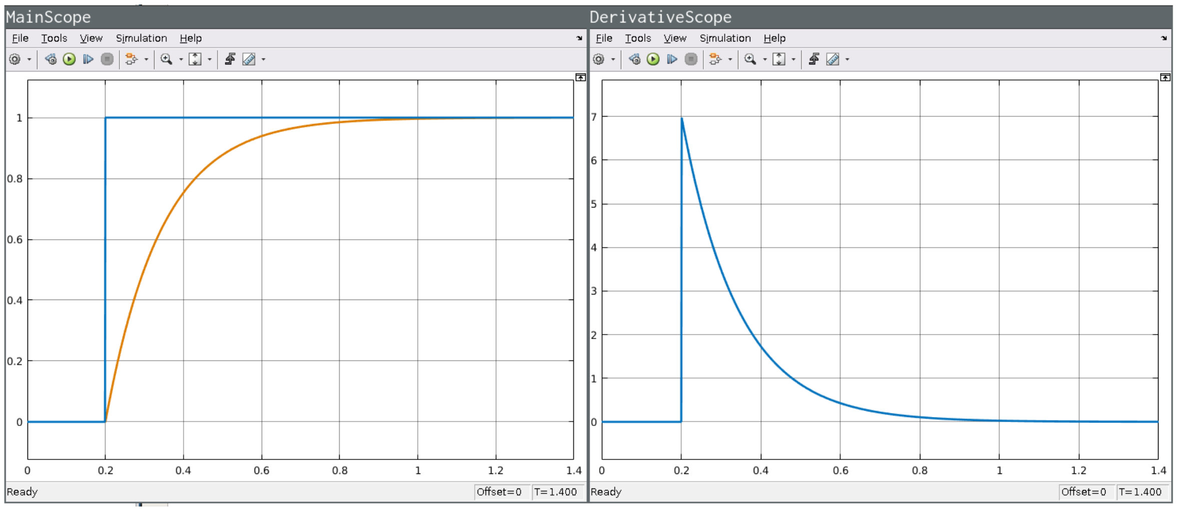


Figure 3: Oscilloscope traces of step input with α parameter tuned to reach goal in one second ($\alpha = 7$).

b) ii τ tuning

As we increase τ, we see that the system still converges to x_f within τ seconds without any changes to α.

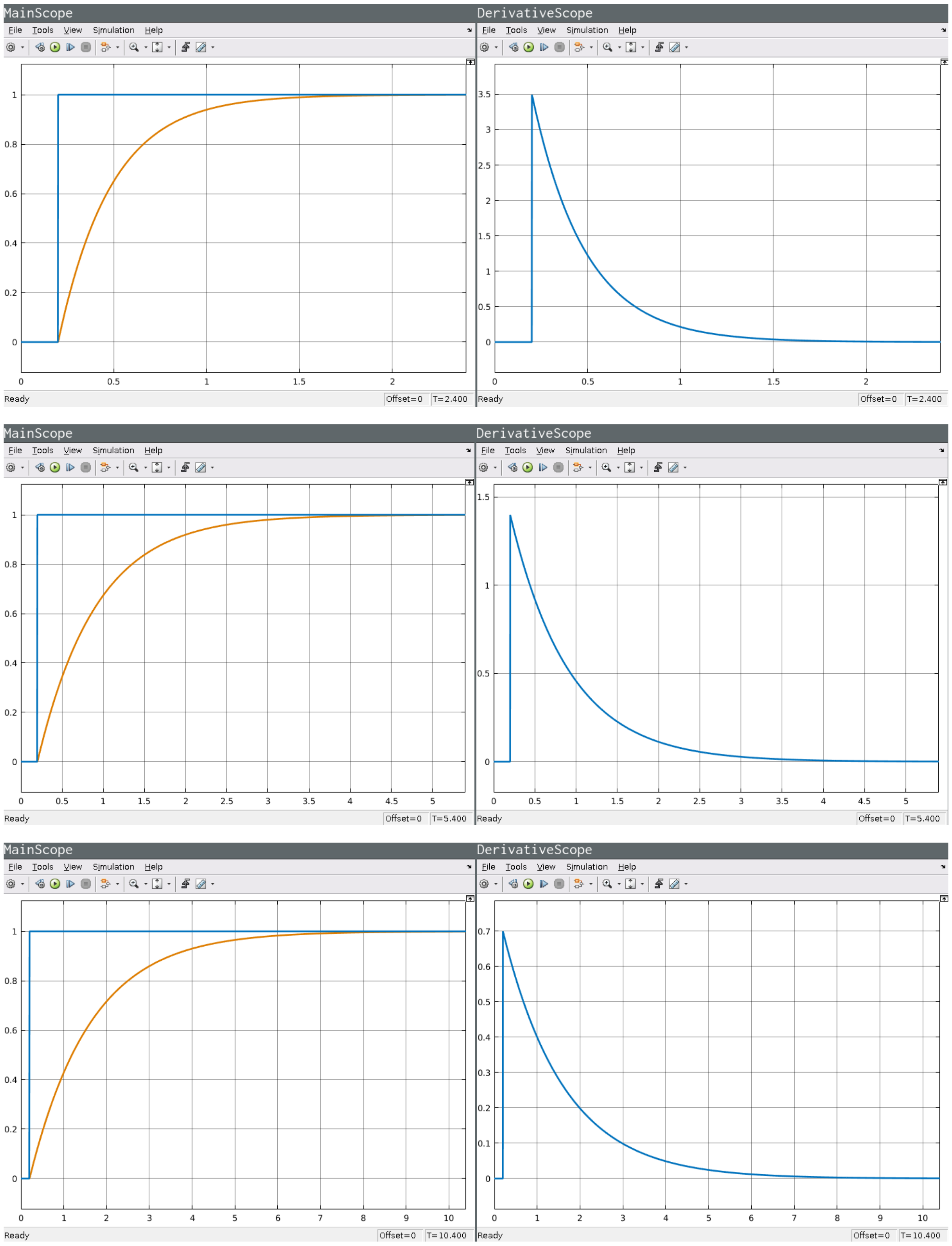


Figure 4: Oscilloscope traces of step input with $\alpha = 7$. *Top:* $\tau = 2$. *Middle:* $\tau = 5$. *Bottom:* $\tau = 10$.

c) Evaluation of first-order planner

Pros:

- Simple to implement
- Low computational cost
- Always converges when $\alpha > 0$ – it is a linear system with equilibrium point $x^* = x_f$ and eigenvalue $-\frac{\alpha}{\tau}$

Cons:

- Initial jump in velocity cannot be achieved on real system - it requires infinite force
- Converges slowly towards end of trajectory
- Requires tuning of α to achieve desired convergence time

d) Euler integration of first-order planner in Matlab

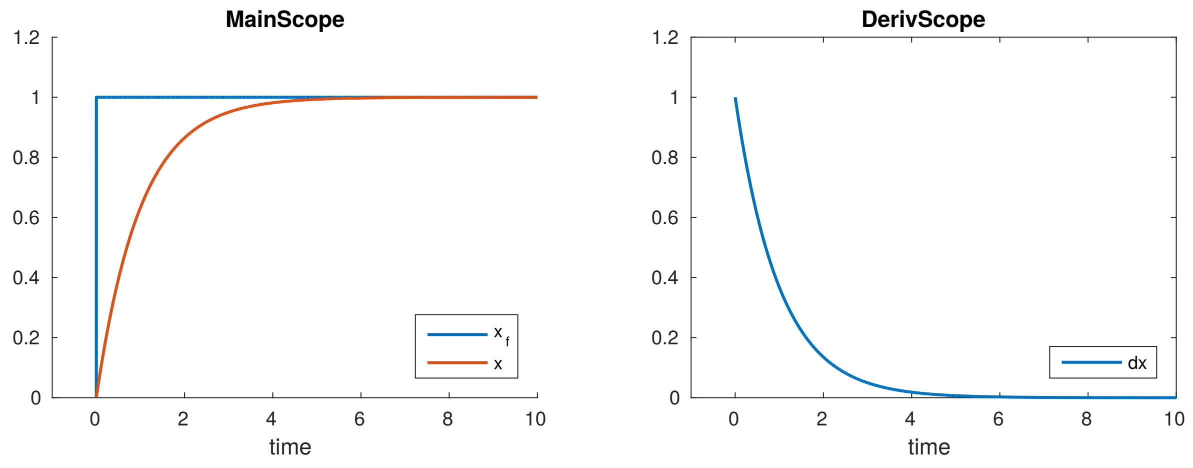


Figure 5: Equivalent of Figure 2 implemented in Matlab with Euler integration.

Code for this section is attached in .zip file as `partd.m`. The result using Euler integration in Matlab appears identical to Simulink's output. However, we never specified which solver Simulink should use to integrate the ODE. Simulink probably used a higher-order solver such as RK4 with better error characteristics than Euler integration.

e) Second-order system

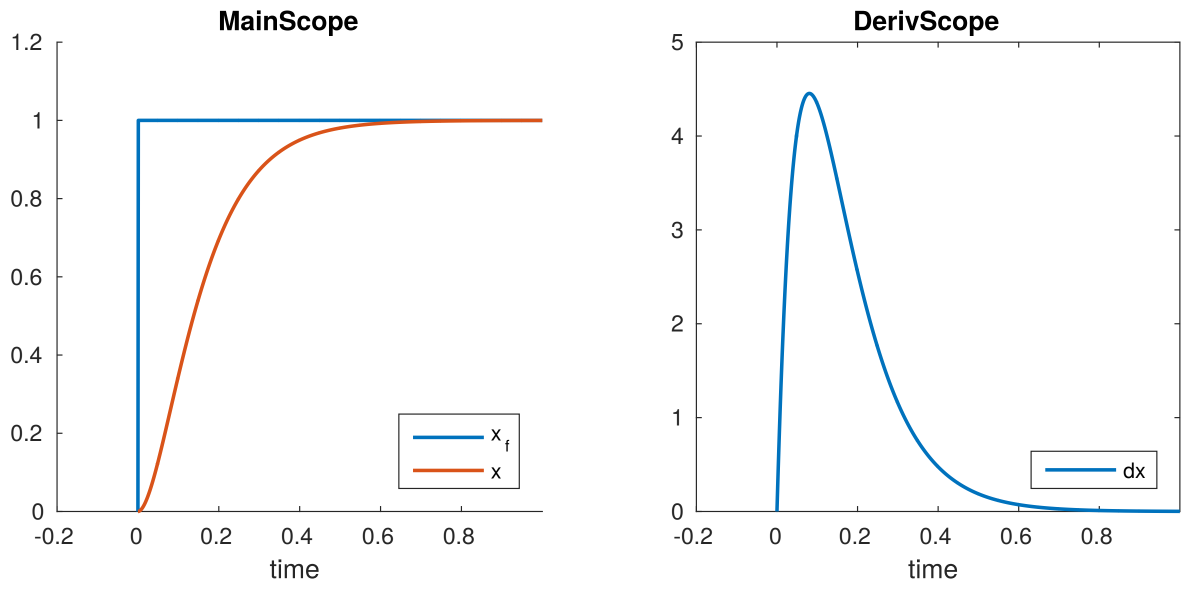


Figure 6: Oscilloscope traces of Second-order dynamical system for part e).

Code for this section is attached in .zip file as `parte.m`. The main difference is that this second-order dynamical system has continuous velocity, so it requires bounded acceleration and is thus more physically plausible than the first-order system of part a). The given parameter values make it reach the goal within one second, but with a lower peak velocity than the $\alpha = 7$ version of the first-order system.

With a second-order system, the error accumulation of Euler integration is much worse. We should use a higher-order ODE integration scheme if we want to simulate the system accurately.

f) Min-jerk spline derivation

The spline equation and its derivatives are:

$$\begin{aligned} x(t) &= c_0 + c_1 t + c_2 t^2 + c_3 t^3 + c_4 t^4 + c_5 t^5 \\ \dot{x}(t) &= c_1 + 2c_2 t + 3c_3 t^2 + 4c_4 t^3 + 5c_5 t^4 \\ \ddot{x}(t) &= 2c_2 + 6c_3 t + 12c_4 t^2 + 20c_5 t^3 \end{aligned}$$

Setting $t = 0$ gives us solutions for the first 3 constants:

$$c_0 = x_0, \quad c_1 = \dot{x}_0, \quad c_2 = \frac{1}{2}\ddot{x}_0$$

Substituting these values and setting $t = \tau$ yields a linear system with 3 equations in 3 unknowns:

$$\begin{bmatrix} \tau^3 & \tau^4 & \tau^5 \\ 3\tau^2 & 4\tau^3 & 5\tau^4 \\ 6\tau & 12\tau^2 & 20\tau^3 \end{bmatrix} \begin{bmatrix} c_3 \\ c_4 \\ c_5 \end{bmatrix} = \begin{bmatrix} x_f - x_0 - \dot{x}_0\tau - \frac{1}{2}\ddot{x}_0\tau^2 \\ \dot{x}_f - \dot{x}_0 - \ddot{x}_0\tau \\ \ddot{x}_f - \ddot{x}_0 \end{bmatrix}$$

To avoid the tedium of doing Gaussian elimination by hand, the Mathematica package was used:

```
a = {{t^3, t^4, t^5}, {3*t^2, 4*t^3, 5*t^4}, {6*t, 12*t^2, 20*t^3}};
b = {xf - x0 - dx0*t - ddx0*t^2/2, dxf - dx0 - ddx0*t, ddxf - ddx0};
LinearSolve[a, b]
```

Yielding the solutions:

$$c_3 = \frac{-3\ddot{x}_0\tau^2 + \ddot{x}_f\tau^2 - 12\dot{x}_0\tau - 8\dot{x}_f\tau - 20x_0 + 20x_f}{2\tau^3}$$
$$c_4 = \frac{3\ddot{x}_0\tau^2 - 2\ddot{x}_f\tau^2 + 16\dot{x}_0\tau + 14\dot{x}_f\tau + 30x_0 - 30x_f}{2\tau^4}$$
$$c_5 = \frac{-\ddot{x}_0\tau^2 + \ddot{x}_f\tau^2 - 6\dot{x}_0\tau - 6\dot{x}_f\tau - 12x_0 + 12x_f}{2\tau^5}$$

g) Matlab implementation of min-jerk spline planner

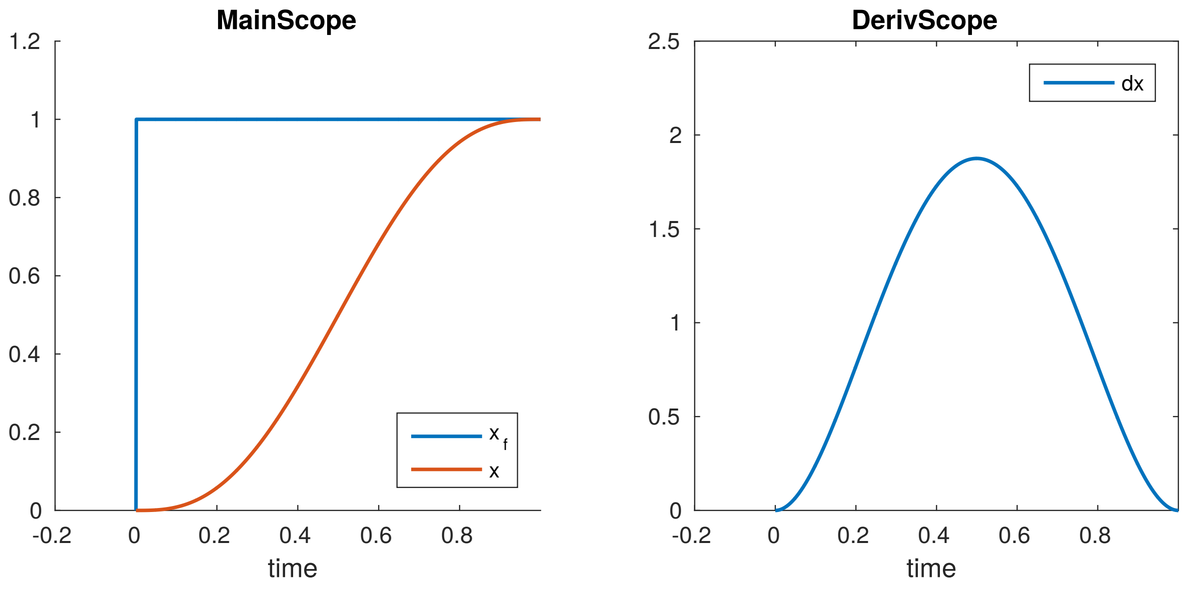


Figure 7: Oscilloscope traces of min-jerk spline planner for part g).

Code for this section is attached in .zip file as `partg.m` [1]. While the dynamical system planners had large velocity spikes at the beginning of the trajectory, the spline trajectory spreads the movement out evenly over the time duration. Like the second-order dynamical planner, the min-jerk planner's velocity profile is continuous, but unlike the second-order planner it is symmetrical. The min-jerk planner also makes it easy to pick the exact time of arrival at the goal and does not require any tuning parameters. However, it requires much more computational effort.

h) Min-jerk planner for reaching task in SL

Code for this section is attached in .zip file as `min_jerk_task.cpp`. Plots of actual vs. desired $\theta, \dot{\theta}, \ddot{\theta}$ for the right arm joints are shown below. There is a significant lag in tracking. This might be fixable by increasing the gain on the low-level joint controllers.

[1] This implementation of the min-jerk spline planner uses Matlab's built-in `polyval` and `polyder` functions to evaluate polynomials and compute their derivatives. The C version for the Nao implements these without using libraries.

The R_HR and R_WR joints had commanded angles of zero, but they deviated slightly. This is because the motors at the stationary joints are still experiencing forces from the rest of the arm, and those motors cannot be perfectly rigid. However, the deviations are very small and probably not an issue in practice.

The lag is most apparent in the acceleration plots. From these plots, it seems like the system might not be using any acceleration feedforward term in the joint angle controllers.

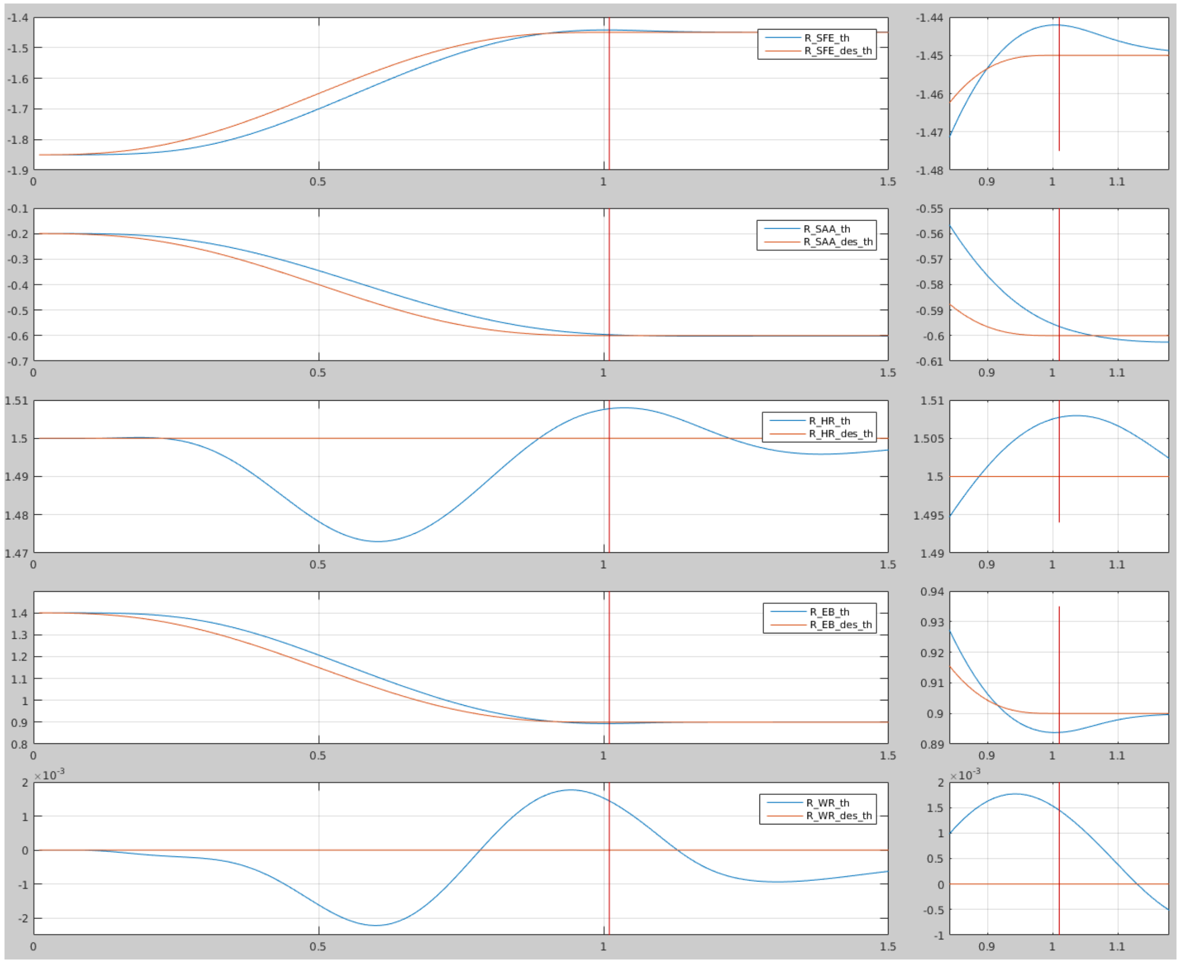


Figure 8: Actual vs. desired joint angles for reaching task with min-jerk spline planner.

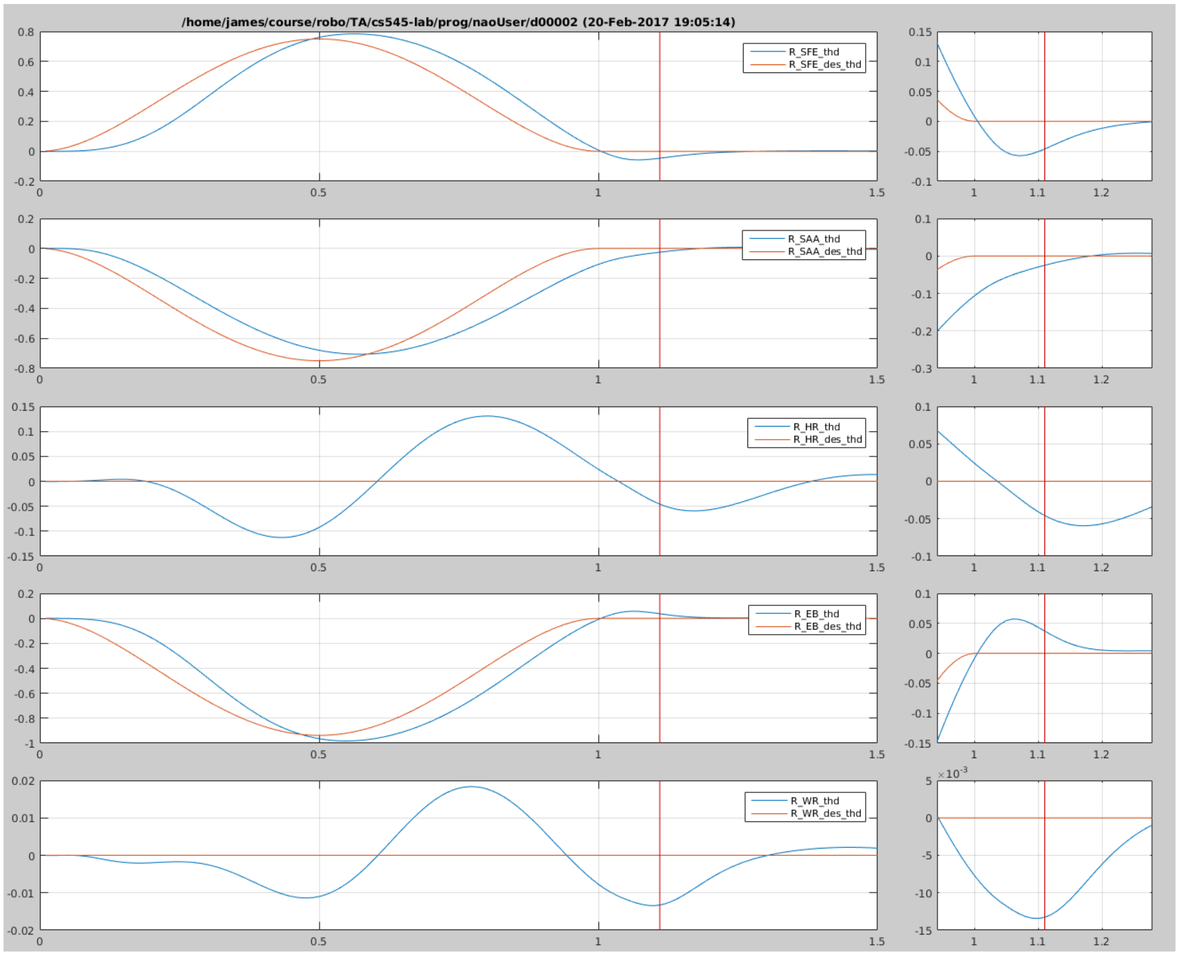


Figure 9: Actual vs. desired joint angular velocities for reaching task with min-jerk spline planner.

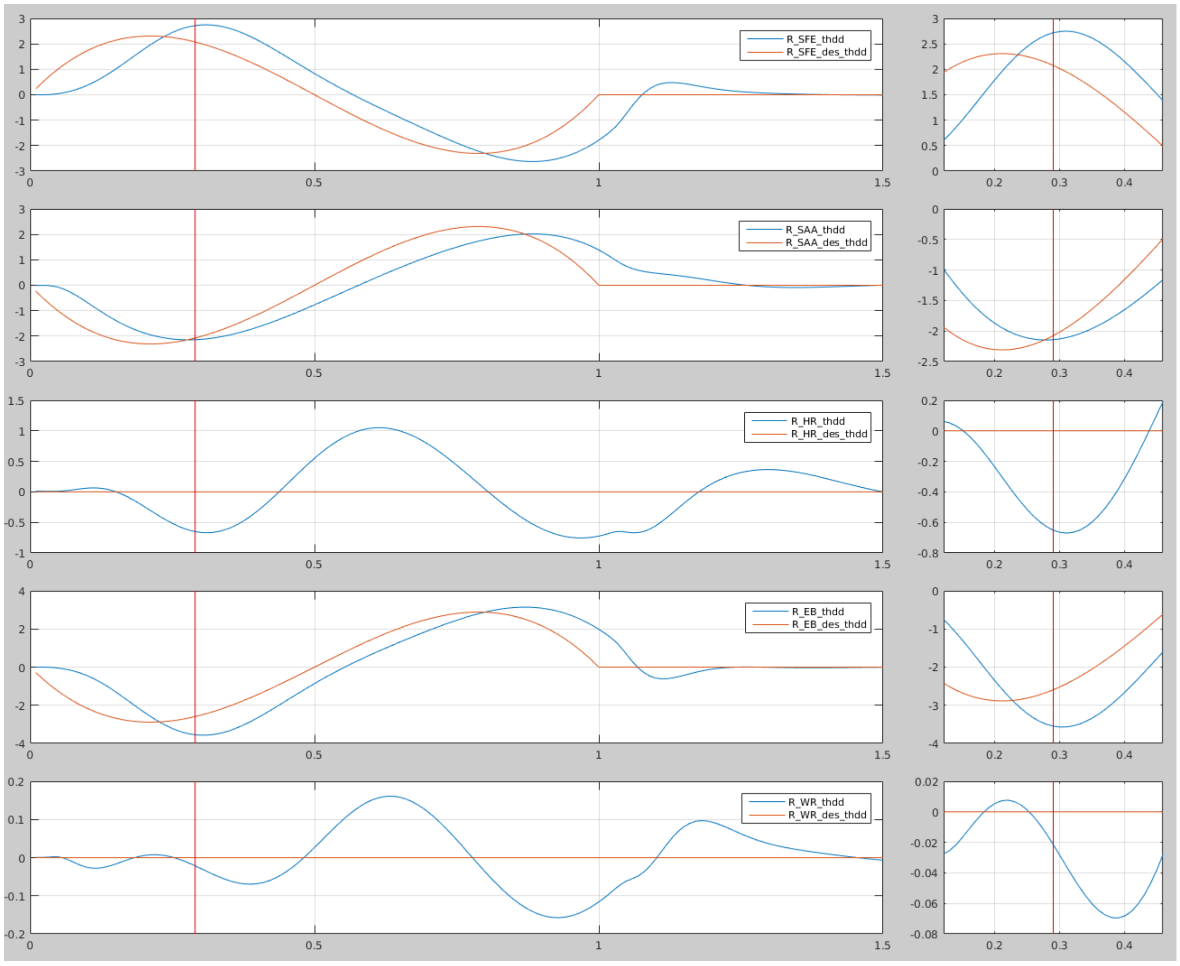


Figure 10: Actual vs. desired joint angular accelerations for reaching task with min-jerk spline planner.

i) Drawing a square with min-jerk planner

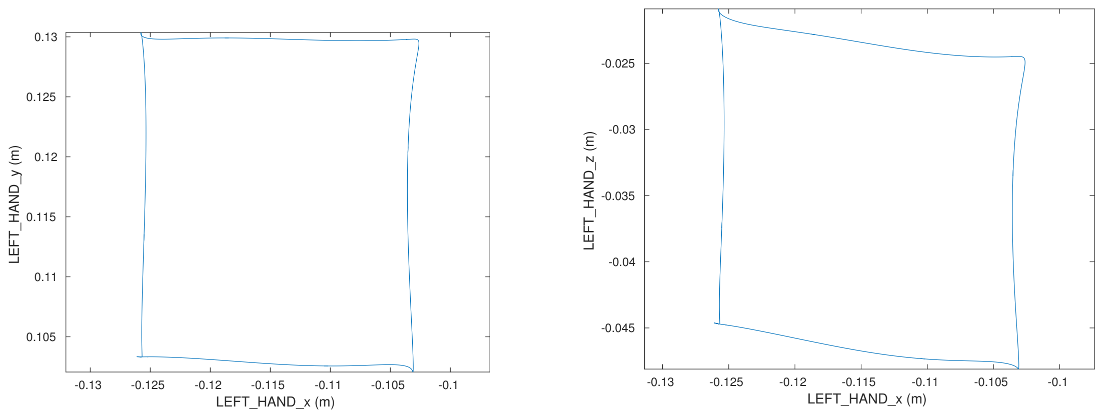


Figure 11: xy (left), xz (right) position phase plots of square movement using min-jerk spline planner.

The phase plots show some problems:

- the x and y side lengths are not equal: The distance from the hand to the shoulder is longer than the distance from the hand to the elbow, so moving $\theta = 0.2$ radians in the shoulder has a bigger effect.
- The lines are not perfectly straight because rotating a joint causes the hand to move in an arc.
- The angles are not 90 degrees when the square is projected into the $x - z$ plane.

But there are some good aspects too:

- Overshoot at the corners is small - this means the plan is generating physically plausible motor accelerations.
- Opposing sides are close to parallel.

In general, these problems are related to planning in joint space; they are not specific to the min-jerk planner. *(Answers vary depending on which joints are used, and how large the angles are.)*

j) Second-order dynamical system planner in SL

Plots of actual vs. desired $\theta, \dot{\theta}, \ddot{\theta}$ for the right arm reaching task using the dynamical system planner are shown below. The tracking is now much worse. The core problem is the big accelerations commanded at the beginning of the trajectory. The motors cannot achieve this acceleration, so the true trajectory falls very far behind the plan.

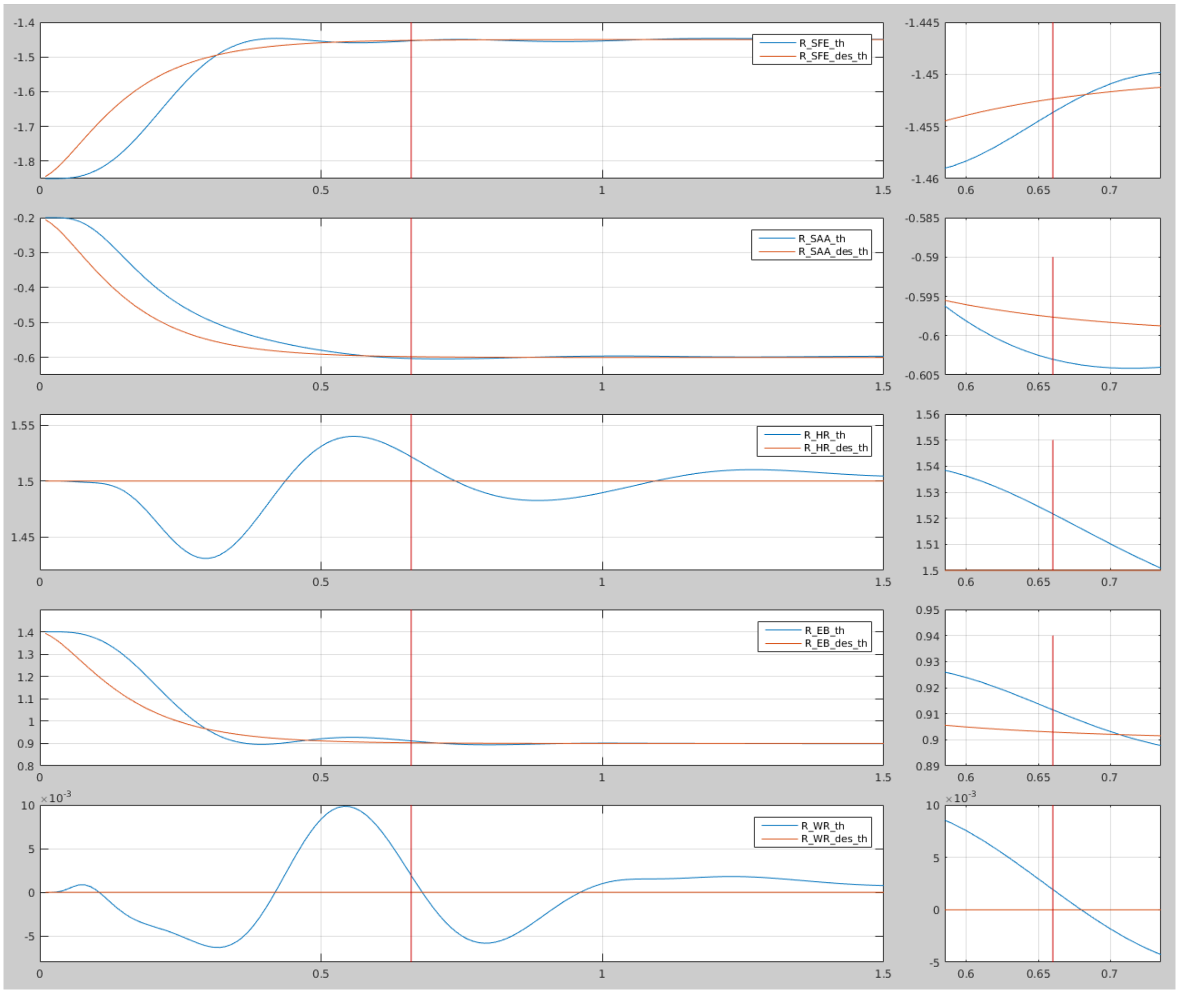


Figure 12: Actual vs. desired joint angles for reaching task with second-order dynamical system planner.

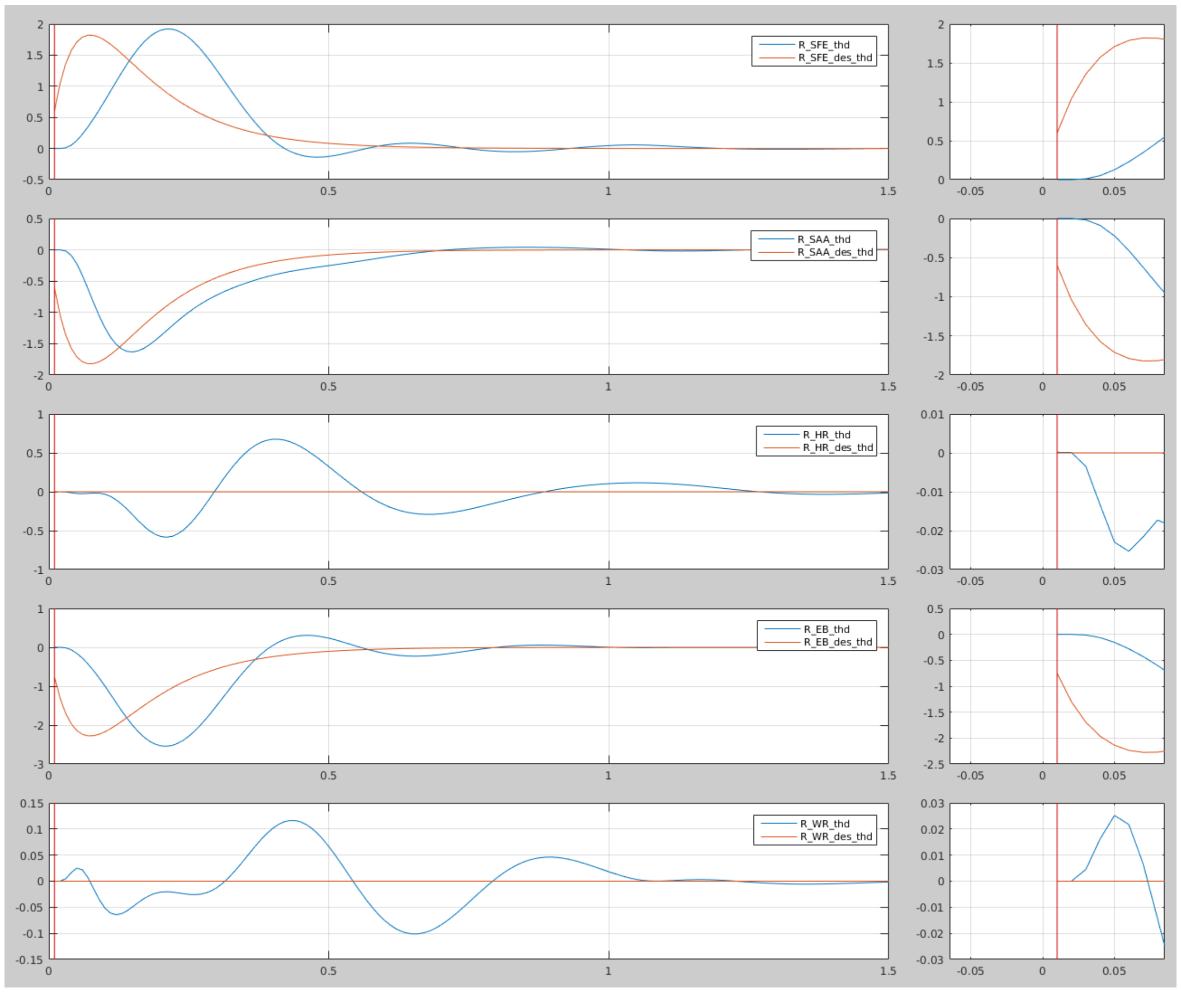


Figure 13: Actual vs. desired joint angular velocities for reaching task with second-order dynamical system planner.

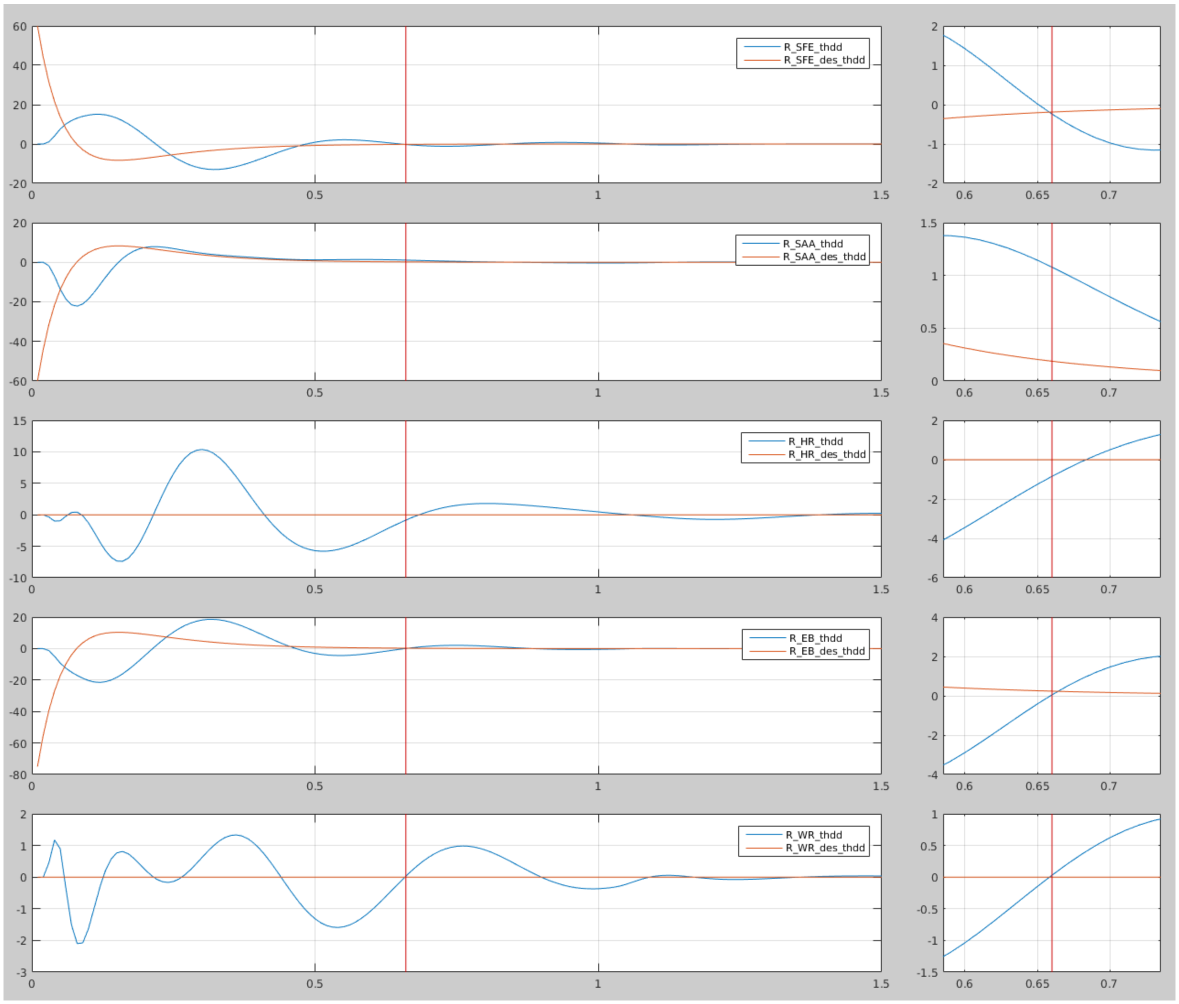
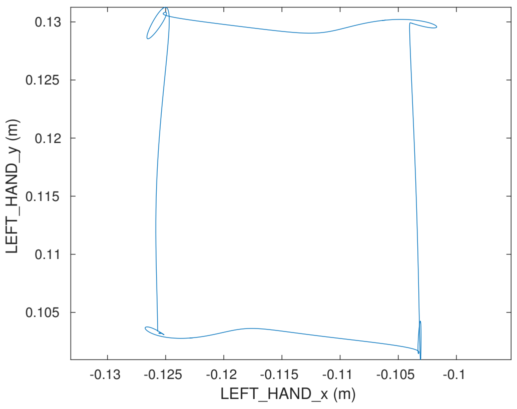


Figure 14: Actual vs. desired joint angular accelerations for reaching task with second-order dynamical system planner.



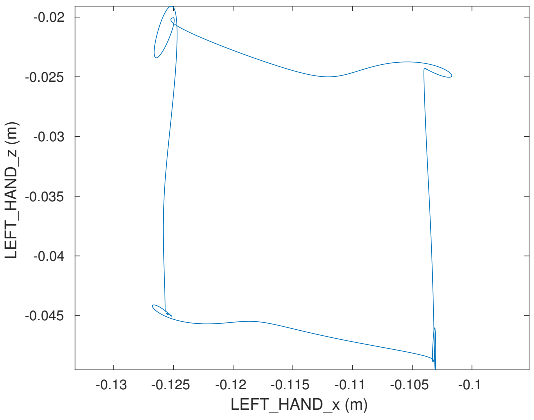


Figure 15: xy (left), xz (right) position phase plots of square movement using second-order dynamical system planner.

The phase plots for the square movement show the same problems with the non-equal side lengths and non-90-degree angles that we saw with the spline planner. However, now the overshoot and ringing at the corners is much worse. This matches the qualitiative impression in the simulator – the movements look much more sudden and jerky, and the movement happens mostly in the first half-second of the one-second intervals. Also, the sides are less straight. These problems are most likely caused by the large velocity spike we saw in part e).

We also see that the movement in R_HR and R_WR, which were supposed to be stationary, is much worse now. This is probably because the faster movements exert larger forces on those joints and the motors cannot produce enough torque to hold the joints rigid.

Overall, this planner appears much worse than the min-jerk spline planner. It could possibly be improved by limiting the commanded accelerations to a hard maximum.

Note: the reaching task, square task, min-jerk planner, and second-order planner are all combined in `min_jerk_task.cpp` using if-*statements.*