6 Sequential Action Control

Motivation

Up until this point we have discussed how one might generate a curve of control values $u(t)$ over an entire time horizon, which works well in the simulated systems of the homework. However, if you are interested in applying optimal control on a real system, we have two issues to contend with. The first is that there is virtually no way that our model of the system and the environment will capture the uncertainties and perturbations that the robot will encounter. One might hope that these perturbations are small enough that our open loop solution for $u(t)$ will work, or we can recalculate $u(t)$ based on current sensor(state) feedback. This brings us to our second problem, the *time* it takes to calculate these control curves. On our real system using our iterative methods to find an optimizer over the entire time horizon may not be fast enough for us to recover from a perturbation. In the worst case, our new $u(t)$ will be completely outdated by the time we have found it. To work around this, we can choose to only calculate the open loop control for a very short time horizon into the future or we can insist that our hardware have sufficient computational speed.

Alternatively, we can reframe the problem so that we are only looking for what our next control action should be in the next t_s seconds, where $t_s = \frac{1}{SamplingRate}$. Rather than looking for a curve over some window into the future we can search for a single action to apply in that window. In other words, we will search for a control action defined by a control vector, u_2^*, an application time τ, and an application duration, λ. What we will find is that there is an analytic solution for a schedule of optimal actions $u_2^*(\tau)$, allowing us to perform a simple line search over the scalar τ and choose λ (also scalar) by enforcing a descent condition.

6.1 Hybrid System Dynamics

Suppose we have a system of the form:

$$\dot{x} = f(x(t), u(t)).$$

The system may be nonlinear with respect to the state, but we will restrict ourselves to cases where, f is linear with respect to the control, satisfying control-affine form.

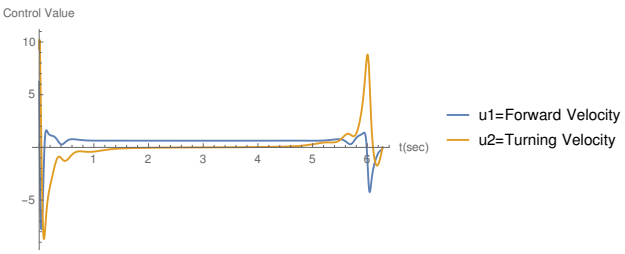
$$f(t, x(t), u(t)) = g(x(t)) + h(x(t))u(t)$$

Given a nominal control, $u = u_1$, the dynamics of the system can be described by $f_1 \triangleq f(x(t), u_1(t))$. Suppose that we want to take a single *short* action—perturbing the nominal dynamics—that improves some cost of the form:

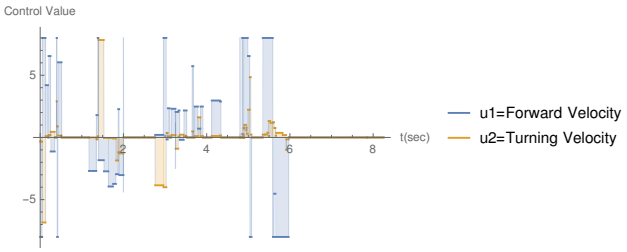
$$J_1 = \int_{t_0}^{t_f} l_1(x(t))dt + m(x(t_f))$$

This will make the control look like,

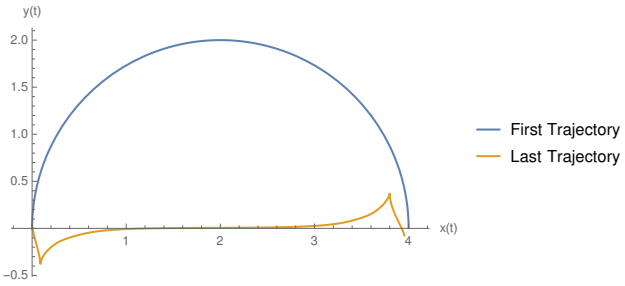
$$u(t) = \begin{cases} u_1(t) & t < \tau,\ t > \tau + \lambda \\ u_2^*(\tau) & \tau \leq t \leq \tau + \lambda \end{cases}$$



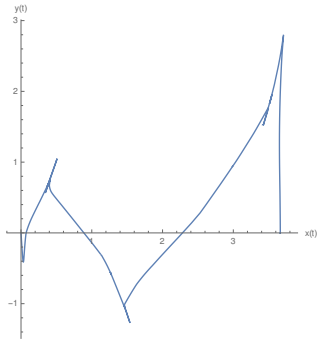
(a) A Control signal for the differential drive vehicle generate by iLQR.



(b) A Control signal for the differential drive vehicle generate by Sequential Action Control(right) and iLQR(left).



(c) A trajectory for the differential drive vehicle generate by iLQR.



(d) A trajectory for the differential drive vehicle generate by Sequential Action Control.

We now have to find the control vector $(u_2^*(\tau))$, the time τ when we want to apply the control, and the (short) control duration λ. The system can now be described by,

$$x(t) = \begin{cases} x(0) + \int_{t_0}^{t} f_1(x(s))ds & t_0 \leq t < \tau \\ x(0) + \int_{t_0}^{\tau} f_1(x(s))ds + \int_{\tau}^{t} f_2(x(s))ds & \tau \leq t \leq \tau + \lambda \\ x(0) + \int_{t_0}^{\tau} f_1(x(s))ds + \int_{\tau}^{\tau+\lambda} f_2(x(s))ds + \int_{\tau+\lambda}^{t} f_1(x(s))ds & t > \tau + \lambda \end{cases},$$

where $f_2 \triangleq f(x(t), u_2^*(\tau))$.

6.2 The Mode Insertion Gradient

Rather than searching for the optimal control vector, application time, and control duration simultaneously, we begin by assuming that τ and f_2 are known. Now we can model the change in cost of a particular (τ, f_2) by taking the deriviation of J_1 with respect to $\lambda \to 0^+$.

$$\frac{dJ_1}{d\lambda^+} = \int_{t_0}^{t_f} \frac{\partial l_1(x(t))}{\partial x(t)} \frac{\partial x(t)}{\partial \lambda^+} dt$$

What does $\frac{\partial x(t)}{\partial \lambda^+}$ look like? Calculating this from the integral equation for $x(t)$ we get:

$$\underbrace{\frac{\partial x(t)}{\partial \lambda^+}}_{z} = \begin{cases} 0 & t_0 \le t < \tau \\ 0 & \tau \le t \le \tau + \lambda \\ \lim_{\lambda \to 0^+} \underbrace{f_2(x(\tau+\lambda)) - f_1(x(\tau+\lambda))}_{z_\tau} + & t > \tau + \lambda \\ \quad \int_\tau^{\tau+\lambda} Df_2(x(s)) \underbrace{\frac{\partial x(t)}{\partial \lambda^+}}_{z} ds + \int_{\tau+\lambda}^t Df_1(x(s)) \underbrace{\frac{\partial x(t)}{\partial \lambda^+}}_{z} ds & \end{cases} \tag{1}$$

Applying the limit $\lambda \to 0^+$, this equation for $\frac{\partial x(t)}{\partial \lambda^+}$ is the integral form of differential equation for an LTV system:

$$\dot{z} = \begin{cases} 0 & t_0 \le t < \tau \\ Az(t) & t \ge \tau \end{cases} \quad z(\tau) = f_2(x(\tau)) - f_1(x(\tau)),\ A = Df_1(x(t)) \tag{2}$$

Now we can write,

$$\frac{dJ_1}{d\lambda^+} = \int_{t_0}^{t_f} \frac{\partial l_1(x(t))}{\partial x(t)} \frac{\partial x(t)}{\partial \lambda^+} dt = \int_\tau^{t_f} \frac{\partial l_1(x(t))}{\partial x(t)} z(t) dt,$$

where z is the solution to the linear differential equation above. Now remember that we haven't actually chosen a specific τ or $u_2^*(\tau)$, so for each choice of these two we would need to solve the differential equation and integrate. Instead let's use the state transition matrix for $z(t)$ in the LTV system in the $\frac{dJ_1}{d\lambda^+}$.

$$\begin{aligned} \int_\tau^{t_f} \frac{\partial l_1(x(t))}{\partial x(t)} z(t) dt &= \int_\tau^{t_f} \frac{\partial l_1(x(t))}{\partial x(t)} \Phi(t,\tau)[f_2(x(\tau)) - f_1(x(\tau))] dt & (3) \\ &= \underbrace{\int_\tau^{t_f} \frac{\partial l_1(x(t))}{\partial x(t)} \Phi(t,\tau) dt}_{\rho^T} [f_2(x(\tau)) - f_1(x(\tau))] & (4) \end{aligned}$$

The second step is possible because $[f_2(x(\tau)) - f_1(x(\tau))]$ does not depend on time. The other term is the convolution integral of a linear affine system running backwards in time. If we call the state variable for this convolution equation $\rho = \int_\tau^{t_f} \Phi^T(t,\tau) \left[\frac{\partial l_1(x(t))}{\partial x(t)}\right]^T dt$, we get that

$$\dot{\rho} = -A^T \rho - \frac{\partial l_1(x(t))}{\partial x(t)}^T$$

The final condition for this differential equation is $\rho(t_f) = 0_{N \times 1}$ if you have neglected the terminal cost $m(x(t_f))$ in your cost J_1 as we did at the start of this derivation. If we had not we would have $\rho(t_f) = \nabla m(x(t_f))$. With this, we get the *mode insertion gradient*

$$\frac{dJ_1}{d\lambda^+}(\tau, u_2^*(\tau)) = \rho^T [f_2(x(\tau)) - f_1(x(\tau))].$$

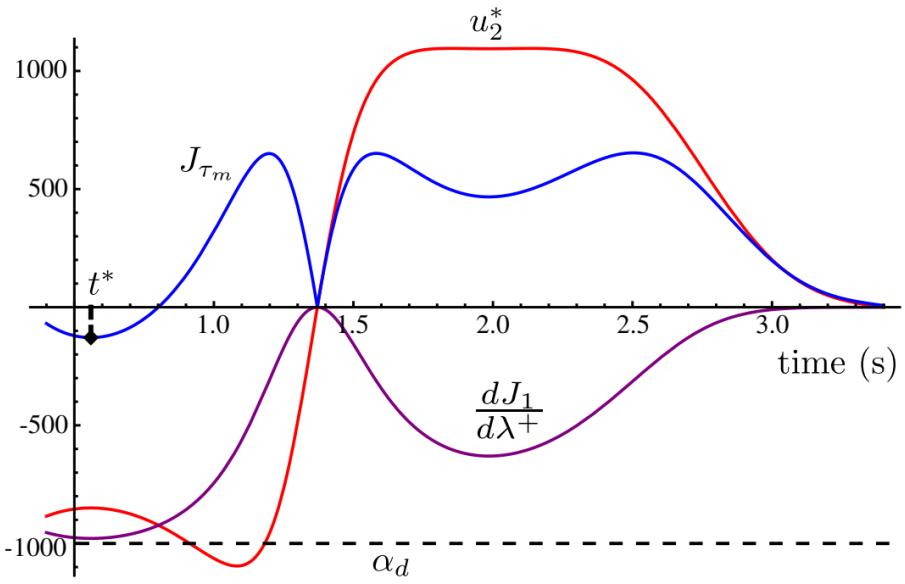


Figure 2: A schedule of optimal actions drives the mode insertion gradient towards α_d. The optimal application time τ is when $\frac{dJ_1}{d\lambda^+}$ is most negative. (From: Ansari & Murphey 2016)

6.3 Schedule of Optimal Actions

You can think of the mode insertion gradient as a measure of the first-order sensitivity of the cost J_1 to a perturbation of the control $u_2^*(\tau)$ at time τ for an infinitesimal duration. We want to choose a control value and time that maximizes the cost sensitivity in the negative direction. We can do this by first finding a schedule of action values $u_2(\tau)$ that minimizes,

$$l_2(\tau, u_2(\tau)) = \frac{1}{2}[\frac{dJ_1}{d\lambda^+}(t, u_2(\tau)) - \alpha_d]^2 + \frac{1}{2}||u_2(\tau)||_R^2,$$

where α_d is the desired sensitivity.

If we evaluate the optimal schedule of actions u_2^* at any time $\tau \in (t_0, t_f)$ that value should minimize l_2, so the entire schedule must also minimize the infinite sum J_2 of the costs associated with each possible value of $u_2^*(\tau)$.

$$J_2 = \int_{t_0}^{t_f} l_2(t, u_2(t))dt$$

In $\frac{dJ_1}{d\lambda^+}$, x depends only on u_1, so unlike the minimization of ζ in Lecture 3, there is no constraint associated with the minimization of J_2. We can now take the directional derivative

and set it equal to zero to find our minimizer. This gives us an *analytic solution* for u_2^*.

$$
\begin{aligned}
DJ_2 \cdot \eta(t) &= \frac{d}{d\epsilon} \int_{t_0}^{t_f} l_2(t, u_2(t) + \epsilon\eta(t))dt|_{\epsilon=0} && (5)\\
&= \int_{t_0}^{t_f} \frac{\partial l_2(t, u_2^*(t))}{\partial u_2(t)} \eta(t)dt = 0 \ \forall \eta && (6)\\
\frac{\partial l_2(t, u_2^*(t))}{\partial u_2(t)} &= 0 && (7)\\
\frac{\partial l_2}{\partial u_2} &= (\rho^T h(x)[u_2^* - u_1] - \alpha_d)\rho^T h(x) + u_2^{*T} R = 0 && (8)\\
u_2^*(\tau) &= (\Lambda + R^T)^{-1}[\Lambda u_1 + h(x)^T \rho \ \alpha_d] && (9)
\end{aligned}
$$

where $\Lambda \triangleq h(x)^T \rho\rho^T h(x)$. This gives use a schedule of actions from which to choose the next control. Although we included a cost on the magnitude of u_2 this doesn't guarantee that the control vector will obey the saturation limits of the system. However, it turns out that control vectors computed from the analytic solution are affine with respect to α_d and linear when $u_1 = 0$, so scaling α_d produces actions that are scaled linearly. This implies that if any component of the control vector violated the saturation limits, we can choose a new α_d to satisfy the constraint. This may produce overly conservative controls when only one element of a multidimensional control vector violates a constraint, but we can also apply scaling element-wise while still producing an action capable of reducing the cost J_1.

6.4 Application Time and Duration

We still need to choose our application time τ and a duration λ. We can use a very small sampling time t_s and choose to just apply the next control based on the schedule at $\tau = t_0 + t_s$ for t_s seconds. Alternatively, we can apply the entire schedule as a descent direction. Finally, we can find τ using an optimization, which we will discuss below.

We now introduce one final cost function to find the time τ at which we can be most effective in applying our single control. Again, we want to minimize $\frac{dJ_1}{d\lambda^+}$ with an additional cost on the control effort and the cost of waiting to apply a control.

$$J_\tau = ||u_2^*(\tau)|| + \frac{dJ_1}{d\lambda^+} + (t - t_0)^\beta, \ \beta > 0$$

Depending on the coding language you are using, there are many ways to find the optimizer. Something like MATLAB's fmincon or Mathematica's Minimize would work here. You can also be creative and create your own minimization function.

Up until this point, we have assumed we are applying control for an infinitely short duration, but assuming we want this to work in the real world we will have to choose a $\lambda > 0$. We do this using a line search with a simple descent condition. Alternatively, you can just choose a very small t_s and apply control

6.5 Algorithm Design Choices

For finite durations, our model of the change in cost is locally described by,

$$\Delta J_1 \approx \frac{dJ_1}{d\lambda^+}(\tau, u_2^*(\tau))\lambda.$$

Our optimization over u_2^* regulates the mode insertion gradient such that,

$$\frac{dJ_1}{d\lambda^+}(\tau, u_2^*(\tau)) \approx \alpha_d$$

Therefore, these two parameters allow us to specify the degree of change provided by each action.

$$\Delta J_1 \approx \alpha_d \lambda$$

A high α_d will lead to aggressive actions by the controller that are often saturated and make dramatic changes in the cost. While any negative number should work, it is often helpful to specify α_d as a feedback law,

$$\alpha_d = \gamma J_{1,init},$$

where $\gamma \in [-15, -1]$ works well. Given a specified α_d and a corresponding $u_2^*(\tau)$ we can choose λ using a line search with a descent condition requiring a minimum ΔJ_{min}, which can be specified as a (negative) percentage of the initial value of J_1.

Finally, β in the optimization of τ is often in the range of $(0.5, 2)$ but any $\beta > 0$ will work.

6.6 SAC Pseudocode

Algorithm 1 Sequential Action Control

Initialize α_d, minimum change in cost ΔJ_{min}, current time t_{curr}, default control duration Δt_{init}, nominal control u_1, scale factor $\omega \in (0,1)$, predictive horizon T, sampling time t_s, the max time for iterative control calculations t_{calc}, and the max backtracking iterations k_{max}.

while $t_{curr} < \infty$ **do**
  $(t_0, t_f) = (t_{curr}, t_{curr} + T)$
  Use feedback to initialize $x_{init} = x(t_0)$
  Simulate (x, ρ) from f_1 for $t \in [t_0, t_f]$
  Compute initial cost $J_{1,init}$
  Specify α_d
  Compute u_2^* from (x, ρ) using Theorem 1:
    $u_2^* = (\Lambda + R^T)^{-1}[\Lambda u_1 + h(x)^T \rho \alpha_d]$
    $\Lambda \triangleq h(x)^T \rho \rho^T h(x)$
  Specify/search for time, $\tau > t_0 + t_{calc}$, to apply u_2^*
  Saturate $u_2^*(\tau)$
  Initialize $k = 0,\ J_{1,new} = \infty$
  while $J_{1,new} - J_{1,init} > \Delta J_{min}$ **and** $k \leq k_{max}$ **do**
    $\lambda = \omega^k \Delta t_{init}$
    $(\tau_0, \tau_f) = (\tau - \frac{\lambda}{2}, \tau + \frac{\lambda}{2})$
    Re-simulate x applying f_2 for $t \in [\tau_0, \tau_f]$
    Compute new cost $J_{1,new}$
    $k = k + 1$
  end while
  $u_1(t) = u_2^*(\tau)\ \forall\ t \in [\tau_0, \tau_f] \cap [t_0 + t_{calc}, t_0 + t_s + t_{calc}]$
  Send updated u_1 to robot
  while $t_{curr} < t_0 + t_s$ **do**
    Wait()
  end while
end while