Iterative Linear Quadratic Regulator in C++

Thomas Lowe

March 9, 2016

If you want to move a robot arm or control a vehicle or any machine then a great way to do it is with feedback control. You simply specify the dynamics of your machine with the environment:

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

where x are your degrees of freedom, such as robot arm joint angles and angular velocities. u are your control signals, such as joint torques, and f is the nonlinear dynamics. You then define the running cost of your state and signals: $l(x, u)$ and the final cost for the final state of your machine's trajectory $l_f(x)$:

$$\Sigma_{k=0}^{K} l(x_k, u_k) + l_f(x_K)$$

We then want to automatically find an optimal controller that minimises the cost subject to the dynamics. An Iterative Linear Quadratic Regulator (ILQR) is a useful means of solving this Optimal Feedback Control problem. It generates a feedforward (open loop) signal u_0 and a feedback (closed loop) term:

$$u = u_0 + L(x_0 - x)$$

One could of course have just found an optimal open loop signal u_0 but the feedback term allows perturbations from the assumed dynamics f to be dealt with, and also provides a higher update-rate controller in the case that ILQR is applied intermittently, e.g. in a receding-horizon Model Predictive Control setting. So Optimal Feedback Control is more robust than just Optimal Control.

There are few libraries that do ILQR and particularly not in C++. So I have implemented the algorithm in C++ based off the theory and algorithms by Mitrovic 2010 [1], Tassal IROS 2012 [2], Tassal ICRA 2014 [3] and StudyWolf 2016 [4]. The implementations are very similar but where there are differences I have used macros to allow the user to try the different options. The default is what gave best results for my tests. The parameter names also vary, I have referred to the names that are used in the different papers but I have chosen to use the syntax of standard LQRs as used in wikipedia, this is clearest to me.

I have kept the library very lightweight, it only relies on the Eigen vector library and is implemented just in the header file ILQR.h. You simply derive this ILQR class and override the system dynamics function $f(x, u)$ and the cost functions $l(x, u), l_f(x)$. Internally these are converted into partial derivatives using finite differences, however if you know the partial derivatives of any of these functions analytically then you can override these getDerivatives() functions directly and save CPU cost.

Now we can test the ILQR, I have included acrobot.h which refers to a toy problem described in 1991 by Richard Murray [5], how do you get a double pendulum to swing up and balance vertically when you only control the mid joint? This is a bit like an acrobat on the parallel bars swinging up to a hand-stand. To solve we only need to penalise difference from the desired end state and a squared torque running cost to avoid torque spikes. Try adjusting these parameters in $l()$ and $l_f()$ to see how they effect the results. To keep the library simple I don't animate the acrobot, just print out the trajectory, which you can load in any spreadsheet:

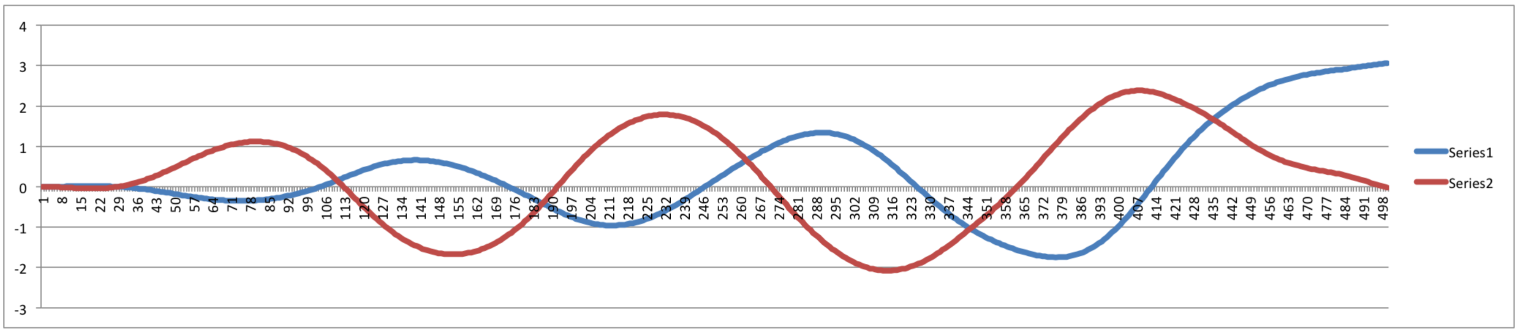

Figure 1: red: actuated joint, blue: unactuated joint

In addition I also include a continuous time version of the solver, which instead interprets the running cost as an integral:

$$\int_{k=0}^{K} l(x_k, u_k) dk + l_f(x_K)$$

This is mainly useful theoretically as it makes a simpler formulation.

References

[1] Stochastic Optimal Control with Learned Dynamics Models Appendix A

[2] Synthesis and Stabilization of Complex Behaviors through Online Trajectory Optimization

[3] Control-Limited Differential Dynamic Programming
- implementation

[4] the iterative linear quadratic regulator method
- implementation

[5] A Case Study in Approximate Linearization: The Acrobot Example
- equations