Evaluating Approximate Inference Control for Classical Underactuated Systems

Rockey Hester
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA
rockeyh@andrew.cmu.edu

***Abstract*—Modern robotic trajectory optimization problems are often underactuated and high-dimensional, but still demand real time performance. "Robot Trajectory Optimization Using Approximate Inference" by Marc Toussaint introduces AICO, an algorithm for efficiently solving the stochastic optimal control problem using approximate inference. iLQR, iLQG, and AICO are implemented for several dynamic systems. Results for iLQR and AICO are shown for the pendulum swing-up problem. AICO as implemented exhibits incorrect convergence properties, but achieves goal of being faster to compute than iLQR. Future work aims to expand notions of distributed computation and approximate inference to graphical models such as factor graphs and graph neural networks.**

I. Introduction

Real time performance on trajectory optimization and model predictive control is critical for robots that interact with people or an otherwise changing environment. Many of the robots we use in these scenarios have complex dynamics - being either underactuated, high-dimensional, or both. Solving the optimal control and stochastic optimal control problems for such systems exactly and in real time is challenging.

The previous state of the art for solving the stochastic optimal control problem was to first compute an optimal trajectory assuming no noise, and then solve a local linear quadratic Gaussian (LQG) problem to account for the uncertainty in the system. This approach has resulted in algorithms such as iterative linear quadratic Gaussian (iLQG). [1] aims to improve this approach by leveraging approximate inference.

The main contribution of [1] is an approximate inference control algorithm (AICO) that provides a local approximate solution to the stochastic optimal control problem that is fast to compute. AICO defines the maximum likelihood trajectory as coinciding with the optimal trajectory and leverages the close theoretical relationship between stochastic optimal control and inference (as exemplified by the Kalman Filter) to solve the trajectory optimization problem.

A key difference between AICO and previous algorithms like iLQG is that it distributes the computation over the trajectory using a forward-backward message passing algorithm. Instead of solving a Ricatti recursion over the global trajectory, AICO performs iterative updates of local messages. This reproduces the classical solution in the linear case and efficiently generalizes to nonlinear systems and costs.

[1] showed AICO outperforming iLQG on a 39 degree of freedom humanoid robotic reaching problem. While a single "sweep" of AICO requires more computation time than one iteration of iLQG, AICO is able to converge faster overall.

Initially the goal of this work was to implement AICO for some classical underactuated robotic systems and a more complicated robotic system and compare its performance to that of iLQG and iterative linear quadratic regulator (iLQR). The additional robotic system chosen is a set of three two-dimensional double integrators connected by springs. This "drone rope" approximates a team of drones cooperatively manipulating a flexible object. At this time, completed work includes the derivation of the "drone rope" dynamics, successful implementation of iLQR on the pendulum and cartpole swing-up tasks, and a partially successful implementation of AICO on the pendulum swing-up task.

II. Background

A. Optimal Control

First, we introduce the classic discrete optimal control problem.

Minimize cost

$$J(x_0, U) = \sum_{i=0}^{N-1} l(x_i, u_i) + l_f(x_N)$$

Subject to dynamics

$$x_{i+1} = f(x_i, u_i)$$

Rather than, for example, solving for a general state feedback controller, trajectory optimization is a class of methods that attempts to solve the optimal control problem by finding a (x, u) sequence that minimizes the cost function.

There are two primary classes of trajectory optimization algorithms:

- Direct methods, which parameterize both the state and control as decision variables
- Shooting methods, which parameterize only the controls and determine the state sequence from forward simultation

[3]

After an initial trajectory is found, trajectory optimization algorithms are often used in a model predictive control (MPC), or receding horizon control, framework, where the system plans a trajectory over a fixed number of time steps m, takes $n < m$ steps, and then replans for the next m steps given where it ended up.

Two trajectory optimization algorithms commonly used for MPC are iterative linear quadratic regulator (iLQR) [2] and differential dynamic programming (DDP). [3]

Both are shooting methods that consist of a backward pass that computes a local solution to the value function using the Hamilton-Jacobi-Bellman equation

$$V(x) = min_u[l(x,u) + V'(f(x,u))]$$

$$Q(\delta x, \delta u) = l(x + \delta x, u + \delta u) + V'(f(x + \delta x, u + \delta u))$$

$$\delta u^* = argmin_{\delta u} Q(\delta x, \delta u) =$$

[3]

Where

$$K = -Q_{uu}^{-1} Q_{ux}$$

$$k = -Q_{uu}^{-1} Q_u$$

[3]

And a forward pass that evaluates the local solution through forward simulation.

$$\hat{x}_0 = x_0$$

$$\hat{u}_i = u_i + \alpha k_i + K_i(\hat{x}_i - x_i)$$

$$\hat{x}_{i+1} = f(\hat{x}_i, \hat{u}_i)$$

[3]

Where α is a line search parameter.

The primary difference between the two is that DDP uses the 2nd derivatives of the dynamics, while iLQR uses only the 1st derivatives.

$$Q_x = l_x + f_x^T V_x'$$

$$Q_u = l_u + f_u^T V_x'$$

$$Q_{xx} = l_{xx} + f_x^T V_{xx}' f_x + V_x' f_{xx}$$

$$Q_{ux} = l_{ux} + f_u^T V_{xx}' f_u + V_x' f_{ux}$$

$$Q_{uu} = l_{uu} + f_u^T V_{xx}' f_u + V_x' f_{uu}$$

[3]

Computing 2nd derivatives is expensive, so DDP iterations are more expensive and iLQR is generally faster overall, but DDP convergence is quadratic. We use iLQR later in this work as a benchmark; however, both methods are deterministic and we would like to consider problems with noise.

B. Stochastic Optimal Control

Next, we introduce the stochastic optimal control problem

Minimize the cost

$$C(x_{0:T}, u_{0:T}) = \sum_{t=0}^{T} c_t(x_t, u_t)$$

$$c_t(x_t, u_t) = x_t^T R_t x_t - 2 r_t^T x_t + u_t^T H_t u_t$$

[1]

Subject to the non-deterministic dynamics

$$P(x_{t+1}|x_t, u_t) = \mathrm{N}(x_{t+1}|A_t x_t + a_t + B_t u_t, Q_t)$$

[1]

The solution to this problem is the linear quadratic Gaussian (LQG) feedback control law

$$u_t * (x) = -(H + B^T V_{t+1} B)^{-1} B^T (V_{t+1}(Ax + a) - v_{t+1})$$

[1]

With

$$K = A^T V_{t+1}^T (V_{t+1} + B^T H B^{-1})^{-1}$$

[1]

And V_t and v_t are computed backward in time using the Ricatti equation.

$$V_t = R + A^T V_{t+1} A - K V_{t+1} A$$

$$v_t r + A^T (v_{t+1} - V_{t+1} a) - K(v_{t+1} - V_{t+1} a)$$

[1]

The iLQG algorithm [6] applies LQG in the same backward-forward structure as iLQR and DDP apply LQR. These solution methods broadly fall under sequential quadratic programming (SQP), which is a method for solving constrained nonlinear optimization problems where the objective function can be differentiated twice.

The algorithm is initialized with an initial trajectory. Then, a second-order approximation of the cost around the initial trajectory is computed. Next, LQG is applied to find the optimal trajectory and LQR controller for this approximation. This process is repeated until convergence. [1] See Figure 1 for pseudocode.

[1]

iLQG allows us to address stochastic systems, but it is not naturally suited to problems with nonlinear cost functions.

Algorithm 1 iLQG

Input: initial trajectory $x_{0:T}$, convergence rate α, control costs $H_{0:T}$, functions $A_t(x)$, $a_t(x)$, $B_t(x)$, $R_t(x)$, $r_t(x)$
Output: trajectory $x_{0:T}$, LQR $V_{0:T}, v_{0:T}$
repeat
  access $R_T(x_T)$, $r_T(x_T)$
  $V_T \leftarrow R_T$, $v_T \leftarrow r_T$
  for $t = T-1$ **to** 0 **do** *// bwd Ricatti recursion*
    access $A_t(x_t)$, $a_t(x_t)$, $B_t(x_t)$, $R_t(x_t)$, $r_t(x_t)$
    compute V_t and v_t using (7,8)
  end for
  for $t = 0$ **to** $T-1$ **do** *// fwd control recursion*
    compute $u_t^*(x_t)$ using (9)
    $x_{t+1} \leftarrow (1-\alpha)x_{t+1} + \alpha[A_t x_t + a_t + B_t u^*(x_t)]$
  end for
until convergence

Fig. 1. iLQG pseudocode

C. Approximate Inference Control

Finally, we derive the approximate inference control algorithm.

We begin by introducing a binary random variable

$$P(z_t = 1|u_t, x_t) = exp{-}c_t(x_t, u_t)$$

[1]

The log-likelihood of this binary random variable over the entire trajectory is equal to the negative cost over the entire trajectory.

$$logP(z_{0:T} = 1|u_{0:T}, x_{0:T}) = -C_t(x_{0:T}, u_{0:T})$$

[1]

Thus, maximizing the log-likelihood is equivalent to minimizing a cost, so the maximum likelihood trajectory aligns with the classical optimal trajectory in the LQG case. This is not true in the general case, but by linearizing about an approximation point and computing the maximum likelihood state for that linearization point, we can achieve a similar result. This is exactly approximate inference.

We would like to compute the posterior distribution over state trajectories conditioned on $z_t = 1$ for all states.

$$P(x_{0:T}|z_{0:T} = 1)$$

[1]

By our definition this guarantees that the maximum likelihood trajectory is the optimal trajectory in the LQG case. Rather than solving this problem using iLQG, we do exact inference using a forward-backward message passing algorithm. This approach makes it easier to generalize to the nonlinear case.

In message passing, we compute the backward messages as

$$P(z_{t:T} = 1|u_t)$$

$$P(z_{t:T} = 1|u_t, x_t)$$

Then, we use Bayes' Rule and the backward messages to compute the forward messages as

$$P(x_t|u_t, z_{t:T} = 1)$$

For the backward messages we define

$$Q_t(x, u) = logP(z_{t:T} = 1|u_t, x_t)$$

$$V_t(x) = logP(z_{t:T} = 1|u_t)$$

These functions are essentially the probabilistic version of the Bellman equations for dynamic programming, so passing backward messages corresponds to performing Bellman updates in a Markov Decision Process. [5]

The forward messages are then

$$P(x_t|u_t, z_{t:T} = 1) = \frac{P(z_{t:T} = 1|u_t, x_t)P(x_t|u_t)}{P(z_{t:T} = 1|u_t)}$$

$$P(x_t|u_t, z_{t:T} = 1) \propto exp(Q_t(x, u) - V_t(x)$$

By computing these messages for every state along the trajectory, we obtain the maximum likelihood state-control trajectory. [5]

These messages are the forward, backward, and task messages in the LQG process and can be written as the distributions:

$$\mu_{x_{t-1} \to x_t}(x_t) = \mathrm{N}(x_t|s_t, S_t)$$

$$s_t = a_{t-1} + A_{t-1}(S_{t-1}^{-1} + R_{t-1})^{-1}(S_{t-1}^{-1}s_{t-1} + r_{t-1})$$

$$S_t = Q + B_{t-1}H^{-1}B_{t-1}^T + A_{t-1}(S_{t-1}^{-}1 + R_{t-1})^{-1}A_{t-1}^T$$

$$\mu_{x_{t+1} \to x_t}(x_t) = \mathrm{N}(x_t|v_t, V_t)$$

$$v_t = -A_t^{-1}a_t + A_t^{-1}(V_{t+1}^{-1} + R_{t+1})^{-1}(v_{t+1}^{-1}v_{t+1} + r_{t+1})$$

$$V_t = A_t^{-1}[Q + B_t H^{-1}B_t^T + (V_{t+1}^{-1} + R_{t+1})^{-1}]A_t^{-1T}$$

$$\mu_{r_t \to x_t}(x_t) = \mathrm{N}(x_t|r_t, R_t)$$

[1]

The backward messages can be rewritten to correspond exactly to the Ricatti equation. The forward messages are not equivalent to the Ricatti equation, but are necessary to estimate the posterior belief of the state.

Lastly we define $b(x_t)$ as the posterior marginal belief of the distribution of a state along the trajectory

$$b(x_t) = \mu_{x_{t-1} \to x_t}(x_t)\mu_{x_{t+1} \to x_t}(x_t)\mu_{r_t \to x_t}(x_t)$$

[1]

Now we use expectation propagation to iteratively improve the messages and the belief. For each time step in AICO, we choose an approximation point based on the

current belief and linearize the dynamics about that point. Then, we recompute the messages and the belief using this local approximation and repeat until convergence. Once a time step has converged, we proceed to the next time step. Once the entire forward trajectory has converged, we repeat the process backward along the trajectory, then forward, then backward again, until the entire process has converged. This is in contrast to iLQG, which linearizes about and updates the entire trajectory each iteration. [1] See Figure 2 for pseudocode.

Algorithm 2 Approximate inference control (AICO)

Input: start state x_0, control costs $H_{0:T}$, functions $A_t(x)$, $a_t(x)$, $B_t(x)$, $R_t(x)$, $r_t(x)$, convergence rate α, threshold θ
Output: trajectory $x_{0:T}$
initialize $s_0 = x_0$, $S_0^{-1} = 1e10$, $v_{0:T} = 0$, $V_{0:T}^{-1} = 0$, $r_{0:T} = 0$, $R_{0:T} = 0$, $k = 0$
repeat
  for $t = 1 : T$ **do** *// forward sweep*
    update s_t and S_t using (20)
    if $k = 0$ **then**
      $\hat{x}_t \leftarrow s_t$
    else
      $\hat{x}_t \leftarrow (1 - \alpha)\hat{x}_t + \alpha b_t$
    end if
    access $A_t(\hat{x}_t)$, $a_t(\hat{x}_t)$, $B_t(\hat{x}_t)$, $R_t(\hat{x}_t)$, $r_t(\hat{x}_t)$
    update r_t and R_t using (22)
    update v_t and V_t using (21)
    update b_t and B_t using (19)
    if $|\hat{x}_t - b_t|^2 > \theta$ **then**
      $t \leftarrow t - 1$ *// repeat this time slice*
    end if
  end for
  for $t = T - 1 : 0$ **do** *// backward sweep*
    ..same updates as above...
  end for
  $k \leftarrow k + 1$
until convergence

Fig. 2. AICO pseudocode

[1]

III. Technical Approach

We implemented the linear and nonlinear dynamics for the 2D double integrator, augmented pendulum, cartpole, and "drone rope". The standard nonlinear dynamics were used for the first three systems and the "drone rope" was represented as three 2D double integrators connected by springs (governed by $F = -kx$). The pendulum and cartpole had control limits of $|u| < 5$.

All linear dynamics were computed as

$$\dot{x} = \frac{\delta f}{\delta x}|_{(x^*,u^*)} + \frac{\delta f}{\delta u}|_{(x^*,u^*)}$$

Then, converted to discrete time

$$x_{t+1} = x_t + \Delta t \dot{x}_t$$

With

$$\Delta t = 0.1s$$

The 2D double integrator and "drone rope" were tested on a relocation task, and the pendulum and cartpole were tested on the swing-up task.

Initially, we implemented direct collocation, iLQR, iLQG, and AICO in Python for all four systems, along with a dynamics simulator using Euler integration and matplotlib animation. While simulating the dynamics of each system in Python was successful, unfortunately bugs in the trajectory optimization algorithms led to a switch to MATLAB.

We implemented direct collocation, iLQR, iLQG, and AICO in MATLAB for the pendulum, cartpole, and "drone rope". We based the iLQG and AICO implementations directly off of the source code from [1] rather than just on the description of the algorithms within the text. This led us to add many additional features, such as a maximum step size for the belief, a maximum number of times a time step could be reexamined, and a damping term to limit the convergence speed when the new cost is higher than the cost of the previous time step. Likewise in our iLQR we took note of [3] and added a line search to the forward pass as an adaptive step size to choose the optimal k.

iLQR worked successfully on the pendulum and cartpole, and AICO was partially successful on the pendulum. We were unable to get any algorithm to successfully solve the "drone rope" problem, and we were unable to get iLQG to successfully solve any of the dynamic systems.

Results are shown for iLQR and AICO solving the pendulum swing-up problem with various parameter settings. The runtime and solution quality are compared.

IV. Results

Results of AICO on the pendulum swing-up problem are given in I. Results of iLQR on the pendulum swing-up problem are given in II. Figures 3, 4, 5, and 6 give selected state trajectories computed using AICO. Figures 7, 8, 9, and 10 give selected state and control trajectories computed using iLQR. In all figures, the thickest trajectory is the final trajectory returned by the algorithm, the second thickest trajectory is the desired trajectory, and all others are intermediate trajectories.

In AICO, the parameter "MaxStepSize" refers to the largest amount the belief is allowed to change in a single iteration. The parameter "Cov" refers to the uniform covariance matrix governing the process noise in the dynamics. The parameter "Damp" refers to whether or not the system changes its step size when the new cost is greater than the cost from the previous iteration. The parameter "MaxFwdReloc" refers to how many times the system reexamines a single time step in an iteration.

In iLQR, the parameters Q and R are the uniform state and control costs, respectively.

TABLE I
AICO PENDULUM SWING-UP RESULTS

MaxStepSize	*Cov*	*Damp*	*MaxFwdReloc*	*Time(s)*	*Cost*
0.01	1	1	0	0.22	0.040
0.1	1	1	0	0.30	3.960
1	1	1	0	0.20	237.457
0.1	10	1	0	0.18	3.435
0.1	100	1	0	0.30	2.414
0.1	1	0	0	0.69	51.413
0.1	1	1	1	0.67	8.860

TABLE II
ILQR PENDULUM SWING-UP RESULTS

Q	*R*	*Time (s)*	*Cost*
1	1	0.57	0.98255
10	1	0.51	0.96446
10	10	0.47	0.98255
100	10	0.43	0.96446
1	10	0.51	15.567
1	100	0.44	15.629

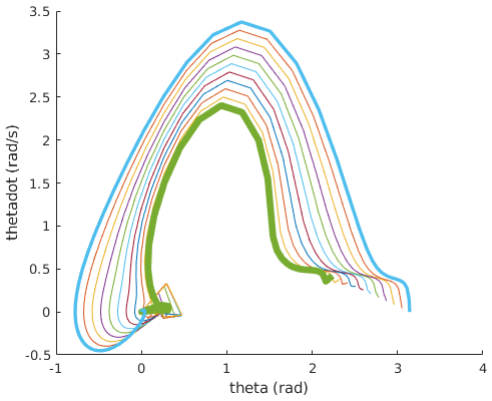

Fig. 3. AICO trajectory for pendulum swing-up with MaxStepSize $=$ 0.1, Cov $=$ 1, Damp $=$ 0, and MaxFwdReloc $=$ 0

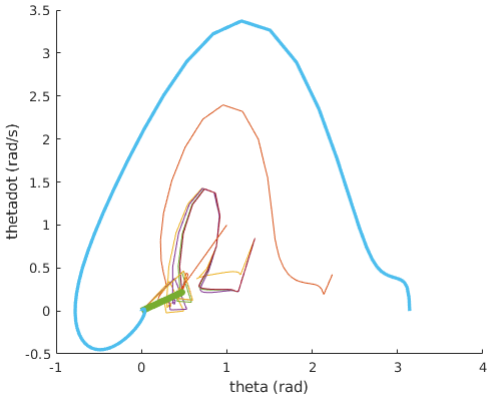

Fig. 4. AICO trajectory for pendulum swing-up with MaxStepSize $=$ 1, Cov $=$ 1, Damp $=$ 1, and MaxFwdReloc $=$ 0

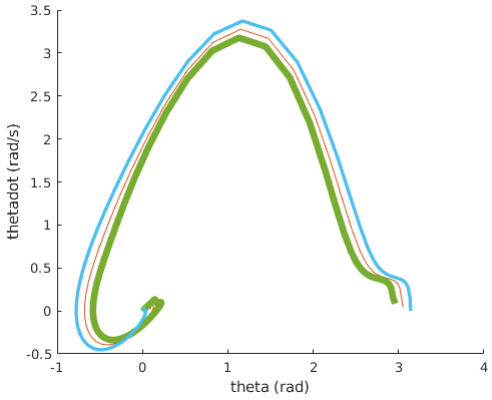

Fig. 5. AICO trajectory for pendulum swing-up with MaxStepSize $=$ 0.1, Cov $=$ 1, Damp $=$ 1, and MaxFwdReloc $=$ 0

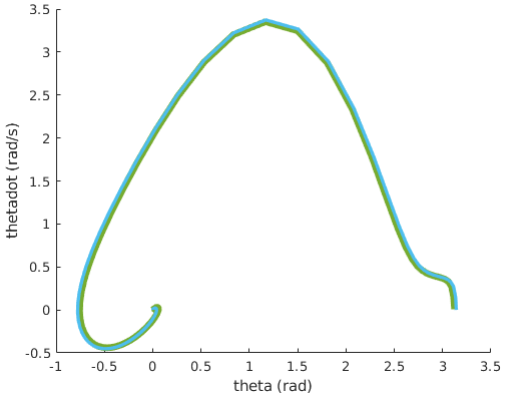

Fig. 6. AICO trajectory for pendulum swing-up with MaxStepSize $=$ 0.01, Cov $=$ 1, Damp $=$ 1, and MaxFwdReloc $=$ 0

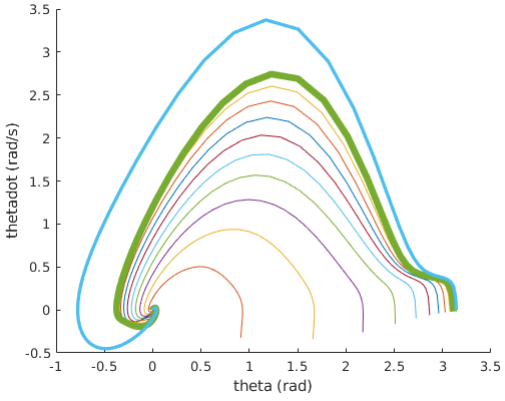

Fig. 7. iLQR state trajectory for pendulum swing-up with $Q = 10$ and $R = 1$

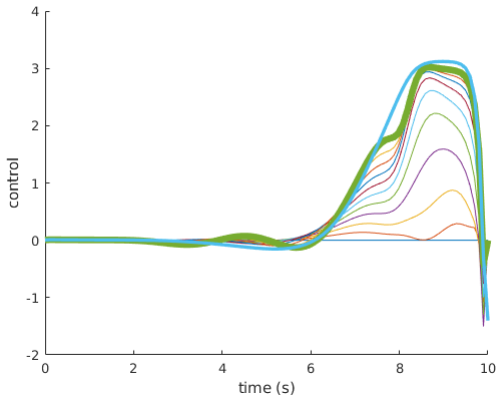

Fig. 8. iLQR control trajectory for pendulum swing-up with $Q = 10$ and $R = 1$

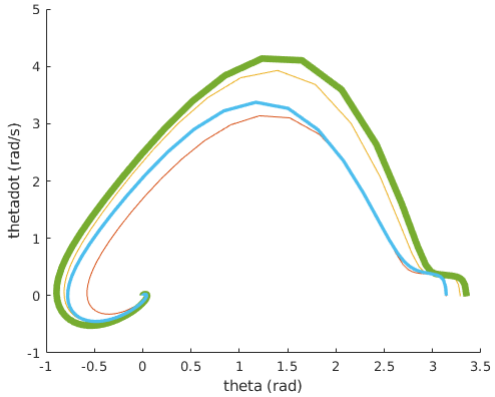

Fig. 9. iLQR state trajectory for pendulum swing-up with $Q = 1$ and $R = 10$

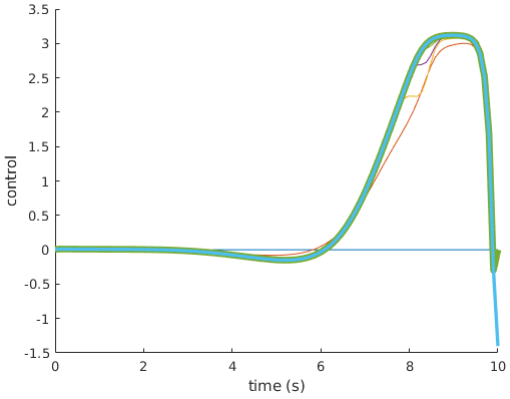

Fig. 10. iLQR control trajectory for pendulum swing-up with $Q = 1$ and $R = 10$

V. Discussion

Overall, AICO computes solutions more quickly as promised, but iLQR returns better solutions (this is an artifact of implementations rather than a theoretical result).

As currently implemented, AICO is initialized with the desired trajectory and attempts to converge to a zero trajectory. It is currently unknown why this is the case, even though the desired trajectory is explicitly incorporated in the cost function and, thus, cost increases as the algorithm converges. However, in this context the results are as expected. Since the algorithm wants to converge to zero, all of the parameters associated with slowing convergence speed will encourage better solutions. These parameters are fewer iterations, a smaller step size, higher variability (covariance), damping, and fewer times reexamining a time step.

Aside from debugging the ”drone rope” linear dynamics and our iLQG and AICO implementations, future work concerns solving the ”drone rope” problem and others using related techniques.

Once the proposed formulation of the ”drone rope” dynamics has been well controlled, we would like to expand the system to include more ”drones” and use more realistic, nonlinear quadrotor dynamics.

The notions of belief propagation, message passing, and distributed computation that are incorporated in AICO are found across robotics and computer science, such as the smoothing problem in simultaneous localization and mapping (SLAM) and relational inductive biases in graph neural networks (GNN). Factor graphs are a revolutionary graphical model used in the SLAM problem that have recently been applied to optimal control and motion planning. [4] Factor Graph LQR has been demonstrated for linear systems, and we would like to solve the optimal control problem for nonlinear dynamic systems like the ”drone rope” using an extension of Factor Graph LQR on a nonlinear factor graph.

Likewise, GNNs encode the structure of the problem in the nodes and edges of a graph and use message passing to learn how nodes and edges interact in general. This is distinct from traditional neural networks, which would have to independently learn parameters describing each node and each edge individually. Systems like the ”drone rope” are naturally represented as a graph, and we would like to use GNNs to learn the dynamics of such systems and then apply approximate inference methods on the learned model in a model predictive control framework.

References

[1] M. Toussaint (2009). Robot trajectory optimization using approximate inference. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 1049–1056.

[2] W. Li and E. Todorov (2004). Iterative Linear Quadratic Regulator Design for Nonlinear Biological Movement Systems. In proceedings of the *1st International Conference on Informatics in Control, Automation and Robotics*, 1: 222-229

[3] Y. Tassa, N. Mansard and E. Todorov, "Control-limited differential dynamic programming," 2014 IEEE International Conference on Robotics and Automation (ICRA), Hong Kong, 2014, pp. 1168-1175, doi: 10.1109/ICRA.2014.6907001.

[4] G. Chen, Y. Zhang, and F. Dellaert,LQR Control using Factor Graphs, https://gtsam.org/2019/11/07/lqr-control.html, 2019.

[5] D. Ghosh, An Introduction to Control as Inference, https://dibyaghosh.com/blog/rl/controlasinference.html, 2018

[6] E. Todorov and Weiwei Li, "A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems," Proceedings of the 2005, American Control Conference, 2005., Portland, OR, USA, 2005, pp. 300-306 vol. 1, doi: 10.1109/ACC.2005.1469949.