18-660: Optimization **Final Report**
Samuel Leong Chee Weng ***scleong***
Saral Tayal ***stayal***

1 Introduction

We are entering a new age of robotics, where robots are becoming more complex (e.g. Boston Dynamics), and more popular and commonplace (e.g. Roomba/self-driving cars). This increased complexity and computation has given rise to a new field of robot control: Optimal Control. By leveraging simplified robot dynamics, Optimal Control allows us to create complex robot trajectories and perform very effective feedback control to external disturbances (e.g. gusts of wind for a quadrotor, or a humanoid robot slipping on pavement).

Within Optimal Control, LQR (Linear Quadratic Regulators) are one of the most popular controllers due to their simplicity, robustness guarantees, and ease of implementation on actual hardware. Within the domain of LQR, there are 2 extremely popular controllers: ILQR and TVLQR. ILQR is a simple controller that simply figures out the feedback gain matrix at a robot's linearized equilibrium point. This is extremely simple and memory efficient but only allows for simple trajectories due to the singular linearization at the equilibrium. TVLQR tackles that shortcoming by linearizing at multiple points over an entire trajectory and storing feedback gains at each of these points. While this allows for significantly more complex trajectories, it comes at the cost of high memory usage that scales directly with the linearization resolution and trajectory length. [1]

Our research involves bridging the gap between these 2 popular LQR-based control techniques. We wish to retain the memory efficiency of ILQR while incorporating elements of TVLQR's trajectory-awareness into our optimization problem.

2 Problem Formulation

2.1 Control System

For our simulations, we are modelling the dynamics of the car with a simple non-linear bicycle model, with the following state (x) and control (u)

$$x = \begin{bmatrix} p_x & p_y & \theta & \delta & v \end{bmatrix}^\top \qquad\qquad u = \begin{bmatrix} a & \dot{\delta} \end{bmatrix}^\top$$

where p_x, p_y is the position, θ is the orientation, δ is the steering angle, and v is the velocity. The controls for the bike are acceleration a, and steering angle rate $\dot{\delta}$.

In order to make the system control-affine for easy optimization, we linearize our approximate dynamics model about X_{ref} and U_{ref} to get the following Jacobians for each step $k \in [1, N]$ in the trajectory:

$$A_k = \left.\frac{\partial f}{\partial x}\right|_{x_{ref,k}, u_{ref,k}} \in \mathbb{R}^{5\times 5} \qquad B_k = \left.\frac{\partial f}{\partial u}\right|_{x_{ref,k}, u_{ref,k}} \in \mathbb{R}^{5\times 2}$$

where $f(x, u)$ is our approximate discrete dynamics model, X_{ref} is the optimal trajectory computed offline with approximate dynamics model, and U_{ref} is the optimal controls computed offline with approximate dynamics model.
With this, the system becomes control-affine, where the states are given by

$$x_{k+1} = A_k x_k + B_k u_k$$

and the proportional-derivative (PD) controller gives the control

$$u_k = -\left[P \cdot (x_k - x_{ref,k}) + D(x_k - x_{k-1})\right]$$

where $P, D \in \mathbb{R}^{2\times 5}$ are the proportional and derivative control matrices to be found by our algorithm.

2.2 Optimizing for PD matrices

In order to automatically tune the PD matrices, we optimize for best PD matrices that minimize the quadratic cost of the trajectory generated by the PID controller, as constrained by control-affine system dynamics. Mathematically, we attempt to solve the following problem:

$$\begin{aligned} \min_{P,D} \quad & \text{Cost}(X, X_{ref}) \\ \text{s.t.} \quad & x_1 = x_{ref,1} \\ & x_{k+1} = A_k x_k + B_k u_k \\ & u_k = -\left[P \cdot (x_k - x_{ref,k}) + D(x_k - x_{k-1})\right] f \end{aligned}$$

where X, X_{ref} are the generated and reference trajectories respectively (a direct function of the states x_k), and A_k, B_k are the Jacobians representing the linearized system dynamics as mentioned in Section 2.1 above.

2.2.1 Without Regularization

Our initial formulation without regularization involved a simple quadratic cost.

$$\mathcal{L}_Q = \text{Cost}(X, X_{ref}) = (X - X_{ref})^\top Q\,(X - X_{ref})$$

where $Q \succeq 0$ is a quadratic cost matrix. Since the cost function is quadratic and the constraints are affine, we have a quadratic programming problem, which can be solved efficiently as noted in class.

2.3 With Regularization

However, as noted in Section 3.1.3, the simple formulation results in exploding P and D values. To combat this, we add L1 regularization to the quadratic cost as follows:

$$\mathcal{L}_{Q,reg} = \text{Cost}(X, X_{ref}) = (X - X_{ref})^\top Q\,(X - X_{ref}) + \lambda\left(\|P\|_1 + \|D\|_1\right)$$

where λ is a hyperparameter that affects the amount of regularization in the system. Although the L1 regularization makes the problem no longer quadratic, the problem is still convex and can be solved quite efficiently by proximal gradient descent.

3 Results

3.1 Single trajectory

Saral setup a simulation architecture in Julia with a preset trajectory to control a car with bicycle dynamics, while Samuel worked on the problem formulation and metrics. The code can be found at https://github.com/SaralTayal123/OptimizationFinalProj. We started with a single parabolic trajectory, shown in Figure 1 below.

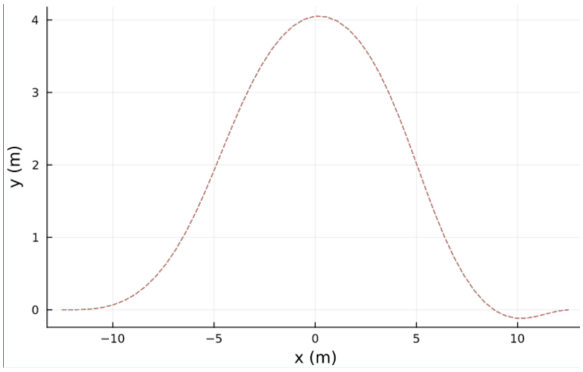

Figure 1: Simple parabolic trajectory for robot to follow

3.1.1 Metrics

Borrowing from the robotics field of odometry [2], two metrics based off the absolute root-mean-square error (RMSE) were included to gauge how well the optimizer performed in trajectory optimization:

- Absolute Position RMSE: RMSE positional difference between calculated and reference trajectory at each point

$$\begin{aligned}\text{RMSE}_{\text{pos}} &= (X - X_{ref})^\top \cdot \text{diag}\begin{bmatrix}1 & 1 & 0 & 0 & 0\end{bmatrix}^\top \cdot (X - X_{ref}) \\ &= \sqrt{\frac{1}{N}\sum_{k=1}^{N}\left[(p_{x,k} - p_{xref,k})^2 + (p_{y,k} - p_{yref,k})^2\right]}\end{aligned}$$

- Absolute Orientation RMSE: RMSE orientational difference between calculated and reference trajectory at each point

$$\begin{aligned}\text{RMSE}_{\text{orient}} &= (X - X_{ref})^\top \cdot \text{diag}\begin{bmatrix}0 & 0 & 1 & 0 & 0\end{bmatrix}^\top \cdot (X - X_{ref}) \\ &= \sqrt{\frac{1}{N}\sum_{k=1}^{N}(\theta_k - \theta_{ref,k})^2}\end{aligned}$$

We also included the quadratic trajectory cost without regularization outlined in Section as a metric. It is mainly useful for debugging and comparing which trajectories were the hardest to optimize from a cost standpoint.

3.1.2 Baseline: No Control Input

We ran the tracker with no controller input (i.e. $P = D = 0$) as a baseline. The generated trajectory is shown in Figure 2 below.

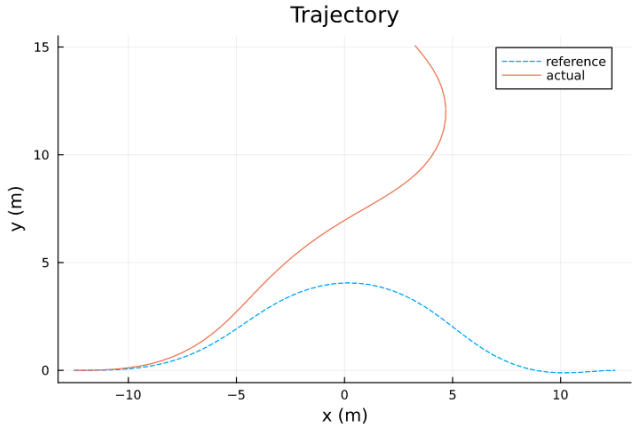

Figure 2: Generated trajectory without control ($P = D = 0$)

3.1.3 Without Regularization

With the simple non-regularized quadratic cost, we found that our P/D matrices had very large values, and caused the optimizer to fail completely, with infinite values for all metrics. See Section 4.1 for a discussion on potential failure reasons.

$$P = \begin{bmatrix} -25.48 & -43.88 & 228.95 & 904.98 & 1151.69 \\ -1.81 & -4.19 & 29.26 & -70.38 & -26.2 \end{bmatrix}$$

$$D = \begin{bmatrix} -130.93 & -71.37 & -1142.36 & 22.02 & 2424.28 \\ -9.39 & 10.48 & -19.01 & 3.23 & 186.85 \end{bmatrix}$$

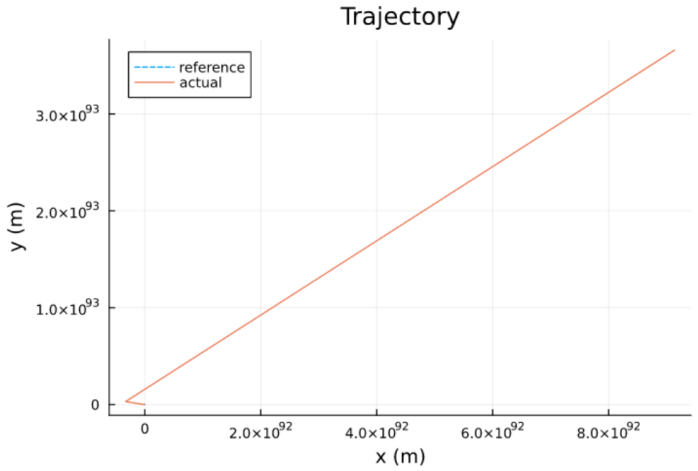

Figure 3: Calculated trajectory with no regularization

3.1.4 With Regularization

As can be seen in Figure 4 below, regularization improves the generated trajectory signficantly and results in sparse P,D matrices with relatively small values, as expected from L1 regularization.

$$P = \begin{bmatrix} -0.79 & 0.11 & 0 & 0 & 0 \\ -0.22 & -0.08 & 0 & 0 & 0 \end{bmatrix} \qquad D = \begin{bmatrix} -0.42 & 0 & 0 & 0 & 0 \\ 0.2 & 0 & 0 & 0 & 0 \end{bmatrix}$$

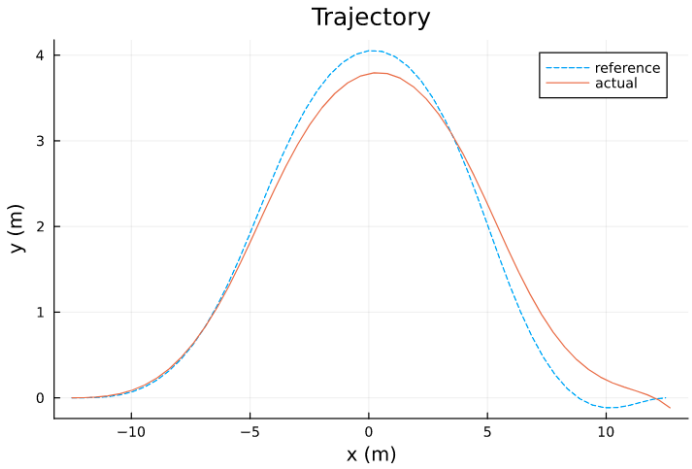

Figure 4: Generated trajectory with regularization

3.1.5 Metrics Comparison

The metrics for the above cases are shown in Table 1 below.

Table 1: Comparison of Metrics (rounded to 3 d.p.)

	Baseline	With Reg.	*Without Reg.*
RMSE_{pos}	7.491	0.290	∞
$\text{RMSE}_{\text{orient}}$	0.976	0.060	∞
$\mathcal{L}_Q$	314554.656	54.695	∞

As evident from Table 1 above, the generated PD matrix using our algorithm (with regularization) clearly improved all metrics. The metrics for the optimization without regularization were all reported to be infinity, and we attempt to discuss why in Section 4.1 below.

4 Discussion

4.1 Potential Reasons why Non-Regularized Quadratic Cost Failed

4.2 Effect of Regularization

5 Learning Objectives

In this project, we originally set out to learn more about optimal control and how optimization can be applied to the field. This project was a indeed good opportunity (especially for Samuel who was more new to the field) to learn more about optimal control theory, particularly dynamics and trajectory optimization. A bigger part of our learning process involved trying to formulate our optimization problem, using what we learnt in class. Since dynamics-aware and trajectory-aware control problems can be quite complicated, it was challenging to do so in a way that was intuitive, while staying convex and easily solvable. Originally, we had wanted to autotune Q/R matrices for optimal control, but quickly found that the formulation was too complex; eventually, we simplified the problem by focusing on PID control and autotuning (optimizing for) P/D matrices. Still, formulating the optimization problem from scratch was not exactly non-trivial, but we used the techniques learnt in class to do it. In particular, we first tried a quadratic cost function with linear constraints to formulate a simple quadratic programming problem, and when that cause the values to explode, we turned to L1 regularization, which worked very well.

References

[1] R. Tedrake, *Underactuated Robotics*. 2023.

[2] Z. Zhang and D. Scaramuzza, "A tutorial on quantitative trajectory evaluation for visual-inertial odometry," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 7244–7251, IEEE, 2018.