Direct Collocation and Time-varying LQR on a 2D Ballbot

Eric Boehlke

***Abstract*—In this report I present the results of applying the trajectory optimization technique of direct collocation along with the trajectory stabilization technique of the time-varying linear quadratic regulator to a model of a planar ballbot. Using these tools the 2D ballbot can be told to move from one position to another while balancing. The time-varying LQR control system is also able to stabilize the trajectory with a model with different parameters.**

I. Introduction

Trajectory optimization is often used to create paths for robots with large state spaces through areas with many constraints. Direct transcription is a method of solving such a trajectory optimization problem. Direct transcription calculates a path through the robot's state space the satisfies constraint and minimizes a cost function of states and inputs. Direct transcription creates this path by minimizing over a set of points along the path in state and input space. To satisfy the dynamics constraint, the dynamics are integrated using a first-order approximation.

Direct collocation is another technique to solve the trajectory optimization problem and is very similar to direct transcription. However, instead of creating piece-wise linear paths like direct transcription, direct collocation creates piece-wise polynomial paths. Typically the input path $\mathbf{u}(t)$ is a first-order polynomial and the state path $\mathbf{x}(t)$ is third order. Using this structure allows direct collocation to satisfy the dynamics to the third-order by evaluating the dynamics for only the endpoints of each segment. This increases the accuracy of the generated trajectories.

Once a trajectory is found, it needs to be stabilized before it can be successfully run on a real or simulated robot. Time-varying LQR is one such technique to create a controller to stabilize around the trajectory generated by a trajectory optimization problem. Time-varying LQR linearizes the system around each of the points in the path and creates an LQR controller to stabilize the system around each point.

Together direct collocation and time-varying LQR make a powerful combination allowing the creation and execution of complicated paths through state space for underactuated robots. I will be applying these tools to a model of the 2D ball balancing robot called a ballbot. A similar project was performed by ETH Zurich on a 3D ballbot [1].

II. Results

A. 2D Ballbot Model

I used the planar ballbot dynamics described in [2]. However, I changed the state variables such that both θ and ϕ are measured from the vertical position in the clockwise direction. This made it easier to interface with the rest of my simulation. I set all of the constants in the problem including gravity to 1. This means the state vector of the ballbot is defined as

$$x = [\theta, \phi, \dot{\theta}, \dot{\phi}],$$

where θ is the angle the ball has rolled and ϕ is the angle of the head from vertical.

B. Running Direct Collocation

For my test, I created a direct collocation problem that drove the robot near a starting state of vertical and stationary at $\theta = 0$ to a final nearly vertical stationary state around $\theta = 2.5$. The exact lower and upper bound for the initial and final states are the following:

$$[-0.1, -0.1, -0.05, -0.05] \leq x_0 \leq [0.1, 0.1, 0.05, 0.05],$$

$$[2.0, -0.05, -0.4, -0.4] \leq x_f \leq [3.0, 0.05, 0.4, 0.4].$$

I set the torque limit on the actuator sufficiently high that there would be no clipping on the input signal. Finally, I added a constraint that required the head angle, ϕ, to be within 0.5 radians of the vertical position for every point along the trajectory. I set the cost to the square of the input torque to encourage smooth motions. The result was a smooth trajectory moving the ballbot from the stationary position at theta = 0 to the stationary position around $\theta = 2.1$ seen in 1.

C. Simulation without Stabilization

After I created this trajectory, I simulated it on a slightly different 2D ballbot model. I added 50 percent more gravity and 50 percent more drag on the ball. The one other difference between this model and the one used for direct collocation is that the simulated model was always started with $x_0 = [0, 0, 0, 0]$ and not the start point dictated by the trajectory. Without stabilization the planned input trajectory caused the head angle, ϕ, to grow to over 1.5 radians. This means my robot would now be upside-down. The rolling angle, θ, also ended up around 1.5 radians, falling short of the planned 2.1 radians. These results can be seen in 2.

D. Adding TVLQR

To stabilize the trajectory and achieve more accurate tracking on the modified ballbot model, I added a time-varying LQR controller to the system. After a bit of messing with the $\mathbf{Q}$ and $\mathbf{R}$ matrices, I ended up with the following values. These weights concentrate on following the position along the

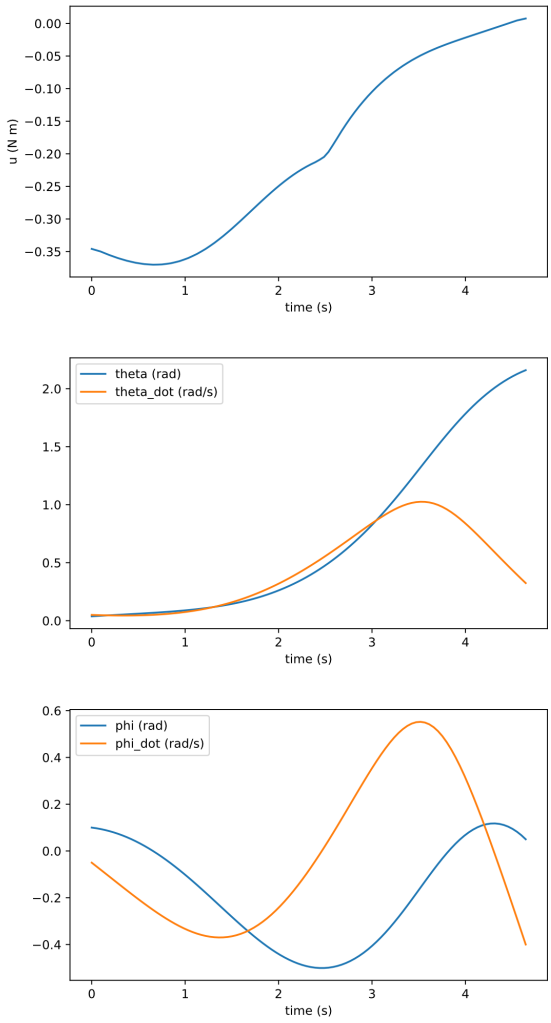

Fig. 1. Trajectory found with direct collocation.

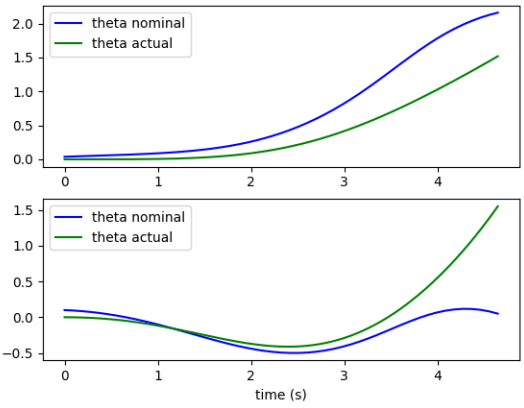

Fig. 2. Comparison between planned trajectory and trajectory followed by the modified ballbot.

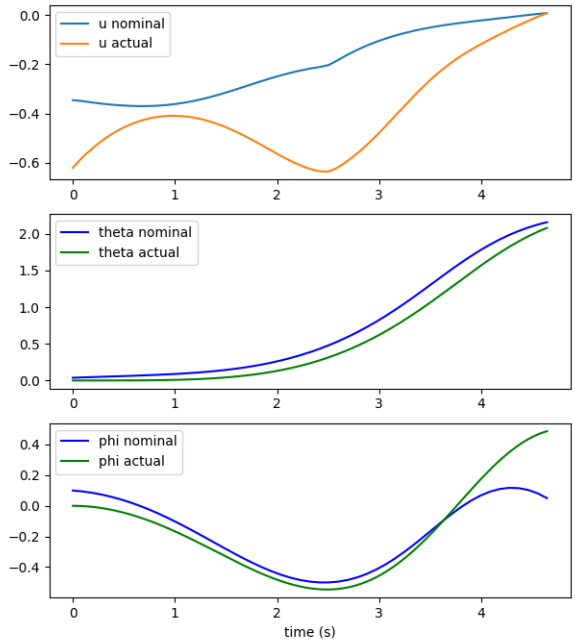

Fig. 3. Comparison between planned trajectory and trajectory followed by the modified ballbot with the TVLQR controller.

trajectory more so than the velocity along the trajectory as I found this to give better results in testing. I also found that if the cost on $\mathbf{R}$ was too low the control input would jump very high to values around 10 which I decided was too high. With this controller, the ballbot does a much better job of following the trajectory as seen in 3. The robot follows the theta trajectory closely. However, ϕ still gets close to .5 radians near the end of the trajectory.

III. Discussion

A. Ballbot Model

The next steps for modeling the system are to change the constants to reflect a real robot and eventually to create a 3D ballbot model. I initially used constants I found in [3], but I had trouble getting the trajectory optimization to give satisfying results. Now that I have a working system, I can slowly add in more complexity.

B. Direct Collocation

Direct collocation gave me a result I was not expecting. In all of the videos I have seen, the ballbot first leans in the direction it wants to move and then pulls the ball under it so it does not fall over. The solution I found was to lean in the opposite direction of motion 4. This could be an error in the ballbot model or it could be due to the constants I used for the model. Many other ballbots have much larger bodies than balls but in this case, all of the constants are set to 1.

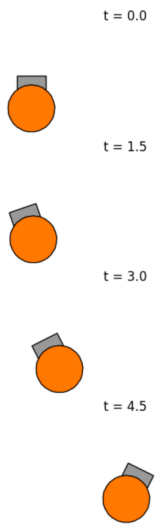

Fig. 4. Frames from the modified ballbot model following the trajectory stabilized with time-varying LQR. Notice how the head leans in the opposite direction of motion.

TABLE I
RUNTIME ANALYSIS

	Average Runtime	
	Direct Collocation	***TVLQR***
Execution Time (s)	0.175	0.225

It was not always possible to find a trajectory from one theta position to another given the constraints. The optimization would also fail sometimes if I tried to constrain the head angle more than 0.5 radians.

The average run time for solving the optimization problem is around 0.175 seconds I. This kind of run time would likely not fast enough to create a model predictive controller, but it is fast enough to run every few seconds to update the trajectory as the robot gathers more information or drifts off course.

C. Time-varying LQR

The TVLQR takes some tuning to get to work well. While my controller does a satisfactory job stabilizing the trajectory it has some problems. It uses nearly twice the torque that the original trajectory required and still diverges from the trajectory in ϕ near the end. To achieve better results there are three possible options.

- I could tune the Q and R matrices until I achieve the desired performance. This is rather tedious but would mean the ballbot could follow long trajectories without falling.
- I could have a smaller difference between the model used for trajectory optimization and the one used for simulation. This would be equivalent to modeling a real-life system better to achieve better results.
- I could decrease the length of the trajectories allowing less time for the ballbot to fall off course. This would be the most computationally expensive option because I would have to recalculate a trajectory and create a new TVLQR controller. I found the average time to recalculate a controller is around 0.225 seconds I which means the whole process to create and stabilize a new trajectory would take around 0.4 seconds. However, it may be slightly less because the trajectories would be shorter.

CONCLUSION

The combination of direct collocation and time-varying LQR is a powerful tool that allows for the creation and execution of trajectories for a robot while respecting constraints, and running in real time. However, finding a trajectory and a controller is not always possible and could take a significant amount of time relative to the time the robot takes to fall down. This means running on a real robot could be dangerous.

MORE INFORMATION

An accompanying video can be found here: https://youtu.be/pz973Jr5Y68. Source code for this project can be found here: https://github.com/ericboehlke/planar_ballbot.

ACKNOWLEDGMENTS

Thank you to the 6.832 course staff for their help with this project. The secret keyword is 'underactuated'.

REFERENCES

[1] D. Pardo, L. Möller, M. Neunert, A. W. Winkler, and J. Buchli, "Evaluating Direct Transcription and Nonlinear Optimization Methods for Robot Motion Planning," IEEE Robotics and Automation Letters, 1(2), pp.946-953. July 2016.

[2] T. B. Lauwers, G. A. Kantor, and R. L. Hollis, "A Dynamically Stable Single-Wheeled Mobile Robot with Inverse Mouse-Ball Drive," IEEE Int'l. Conf. on Robotics and Automation. May 2006.

[3] P. Fankhauser and C. Gwerder, "Modeling and Control of a Ballbot," ETH Zurich. 2010.