High lean angle trajectory optimization for ball balancing robots

Mihir Shevgaonkar

***Abstract*—Trajectory optimization is effective in high lean angle ballbot control scenarios. The technique can be used to generate circular or nearly circular orbits, make sharp turns, and avoid obstacles. It can make effective control decisions under lean angle and actuation limits. It is superior to LQR under high lean angle stabilization, and may also have a slight edge for low lean angle stabilization. Future work might involve increasing solve speed for use in a model predictive controller.**

I. Introduction

Ballbots are land robots that locomote using a single spherical wheel. They are statically unstable so they require a controller to balance. They are capable of accelerating in any direction on the ground, and must lean in the direction of acceleration. Because of their balancing nature they can be made tall and thin, unlike statically stable robots of similar mass.

Linear control is effective in ballbot stabilization under small lean angles such as $5°$. The small lean angle assumption has often been used in the literature for ballbot control analysis [1] [2]. However, linear control may be less effective under high lean angles for two reasons. One reason is that linear control will mis-estimate the control forces required for stabilization to an equilibrium point from high lean angles. The other is that the linearized model ignores the cross term dynamics introduced by the lean angles.

The purpose of this paper is to investigate the use of trajectory optimization for ballbot control under high lean angles. A trajectory optimization workflow was constructed and subjected to various dynamic conditions. Part of the work focused on comparing head-to-head the performance of LQR and trajectory optimization in stabilization. The other part focused on optimizing ”canonical” high lean angle trajectories such as circular orbits.

II. Modeling and simulation setup

A. State and input spaces

The ballbot is represented as a base with two degrees of freedom on the ground (the ball), and a rigid mass connected to the base by a universal joint (the body). The ball is actuated directly by two forces acting along the x and y axes, and the universal joint is passive.

Analysis was initially attempted using spherical coordinates because the orthogonality of the coordinates makes them easier to think about. However, linearization is not possible because it extends the loss of heading in the upright state to the entire state space. In contrast, Euler angles are elegantly decoupled by linearization. Thus, all analysis in this report was performed using lean angles represented by Euler angles about the global y and local $-x$ axes, in that order.

The state vector and control input used in this analysis are shown in (1), where γ is rotation about global y and θ is rotation about local $-x$.

$$\boldsymbol{x} = \begin{bmatrix} x \\ y \\ \gamma \\ \theta \\ \dot{x} \\ \dot{y} \\ \dot{\gamma} \\ \dot{\theta} \end{bmatrix}, \boldsymbol{u} = \begin{bmatrix} F_x \\ F_y \end{bmatrix} \tag{1}$$

Figure 1 illustrates the degrees of freedom of the robot. The ball can move along the x and y axes, and the body first rotates about the global y axis, then about the local $-x$ axis.

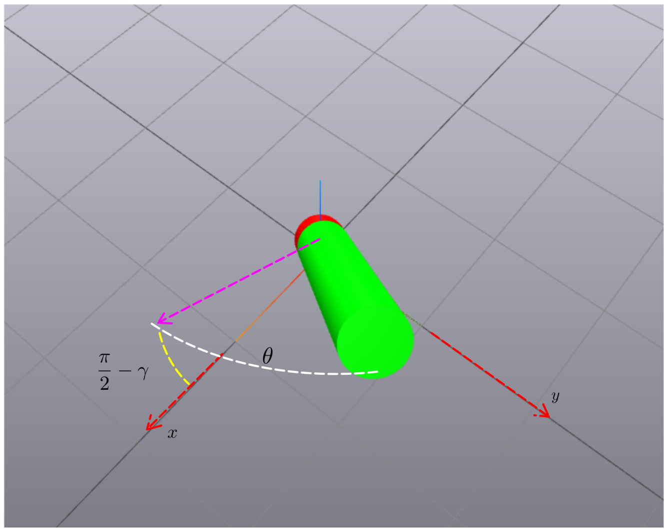


Fig. 1: Degrees of freedom of ballbot

B. Nonlinear equations of motion

I derived the full manipulator equations in spherical coordinates before doing any programming. Because the robot could be specified in a URDF file and passed into `MultibodyPlant`, working directly with the equations of motion was not required. I stopped using spherical coordinates after running into the singularity, so I will not include the equations of motion for brevity.

C. URDF specification

The structure of the URDF file is summarized in the bullet point list below.

- World W
- Massless link A
- Prismatic joint J between A and W along x-axis
- Actuator on joint J
- Ball link B with nonzero mass
- Prismatic joint K between B and A along y-axis
- Actuator on joint K
- Massless link C
- Revolute joint L from B to C about global y-axis
- Body link D with nonzero mass
- Revolute joint M from C to D about local -x-axis

D. Trajectory Optimization

Direct collocation was used to optimize trajectories for three different scenarios. Time-varying LQR was used to stabilize those trajectories.

1) Optimization 1: Initial and Final Condition: This optimization requires that the trajectory takes the robot from an initial condition x_0 to a final condition x_f. Matrices Q_0 and Q_1 are diagonal matrices for imposing costs on "ground distance from destination" and "distance from zero", respectively. Matrix Q_0 is zero except for the first two diagonal terms, and Q_1 is zero except for the last six diagonal terms. h is the timestep, N is the number of timesteps, and S is the cost on final time. The initial trajectory is set to a straight line from x_0 to x_f.

$$\begin{aligned}
\min_{\boldsymbol{u}[\cdot],\boldsymbol{x}[\cdot],h} \quad & \sum_{n=0}^{N-1} \big((\boldsymbol{x}[n]-\boldsymbol{x}_f)^T Q_0 (\boldsymbol{x}[n]-\boldsymbol{x}_f) + \boldsymbol{x}[n]^T Q_1 \boldsymbol{x}[n] \\
& + \boldsymbol{u}[n]^T R \boldsymbol{u}[n] + S(hN)\big) \\
\text{s.t.} \quad & \dot{\boldsymbol{x}}(t_{c,n}) = f(\boldsymbol{x}(t_{c,n}), \boldsymbol{u}(t_{c,n})) \\
& \boldsymbol{x}[0] = x_0 \\
& \boldsymbol{x}[N] = x_f \\
& \max(\boldsymbol{u}) \le u_{max} \\
& |\gamma| \le l_{max} \\
& |\theta| \le l_{max}
\end{aligned} \tag{2}$$

2) Optimization 2: Circular Orbit: This optimization generates a circular orbit. The decision variable r is the orbit radius. The ground speed v is passed into the program. The initial x is zero and the initial y is positive to constrain the initial ground position to the positive y-axis. The initial and final states are set to equal each other, and the ground speed is set to be a constant. Q is all zeros except for x and y which imposes a cost on r. The initial trajectory is set to a square along the ground with constant ground velocity in the tangent direction at each corner.

$$\begin{aligned}
\min_{\boldsymbol{u}[\cdot],\boldsymbol{x}[\cdot],h,r} \quad & \sum_{n=0}^{N-1} \big(\boldsymbol{x}[n]^T Q \boldsymbol{x}[n] + \boldsymbol{u}[n]^T R \boldsymbol{u}[n] + S(hN)\big) \\
\text{s.t.} \quad & \dot{\boldsymbol{x}}(t_{c,n}) = f(\boldsymbol{x}(t_{c,n}), \boldsymbol{u}(t_{c,n})) \\
& x[0] = 0 \\
& y[0] \ge 0 \\
& \boldsymbol{x}[0] = \boldsymbol{x}[N] \\
& x^2 + y^2 = r \\
& \dot{x}^2 + \dot{y}^2 = v^2 \\
& \max(\boldsymbol{u}) \le u_{max}
\end{aligned} \tag{3}$$

3) Optimization 3: Obstacle Avoidance: This optimization is identical to Optimization 1, except there is an additional constraint that the ground position must never come within a distance d of any of the obstacles o_i.

$$\begin{aligned}
\min_{\boldsymbol{u}[\cdot],\boldsymbol{x}[\cdot],h} \quad & \sum_{n=0}^{N-1} \big((\boldsymbol{x}[n]-\boldsymbol{x}_f)^T Q_0 (\boldsymbol{x}[n]-\boldsymbol{x}_f) + \boldsymbol{x}[n]^T Q_1 \boldsymbol{x}[n] \\
& + \boldsymbol{u}[n]^T R \boldsymbol{u}[n] + S(hN)\big) \\
\text{s.t.} \quad & \dot{\boldsymbol{x}}(t_{c,n}) = f(\boldsymbol{x}(t_{c,n}), \boldsymbol{u}(t_{c,n})) \\
& \boldsymbol{x}[0] = x_0 \\
& \boldsymbol{x}[N] = x_f \\
& \forall o_i, (x - o_{x1})^2 + (y - o_{y1})^2 \ge d \\
& \max(\boldsymbol{u}) \le u_{max} \\
& |\gamma| \le l_{max} \\
& |\theta| \le l_{max}
\end{aligned} \tag{4}$$

E. Linearization

The model must be linearized to generate an LQR controller. The A and B matrices obtained for the specific dimensions used in my URDF are shown in (5) and (6).

$$A = \begin{bmatrix}
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & -35.072 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -35.072 & 0 & 0 & 0 & 0 \\
0 & 0 & 60.403 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 60.403 & 0 & 0 & 0 & 0
\end{bmatrix} \tag{5}$$

$$B = \begin{bmatrix}
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0.295 & 0 \\
0 & 0.295 \\
-0.397 & 0 \\
0 & -0.397
\end{bmatrix} \tag{6}$$

The sparsity of these matrices makes it easy to extract the resulting set of linear equations for any linearized ballbot. The values of the constants for the dimensions specified in my URDF are $a = 35.072$, $b = 60.403$, $c = 0.295$, $d = 0.397$.

$$\ddot{x} = -a\gamma + cF_x \tag{7}$$

$$\ddot{y} = -a\theta + cF_y \tag{8}$$

$$\ddot{\gamma} = b\gamma - dF_x \tag{9}$$

$$\ddot{\theta} = b\theta - dF_y \tag{10}$$

The equations (7) through (10) show that the (x,γ) and (y,θ) degrees of freedom are completely decoupled from one another.

III. Results and Discussion

In this section various control cases will be analyzed with the help of plots generated during simulation. Each set of plots includes all state variables (except the derivatives of lean angles) over time, the control inputs over time, and the phase portrait of the ball's position on the x-y plane.

A. Stabilization to Origin

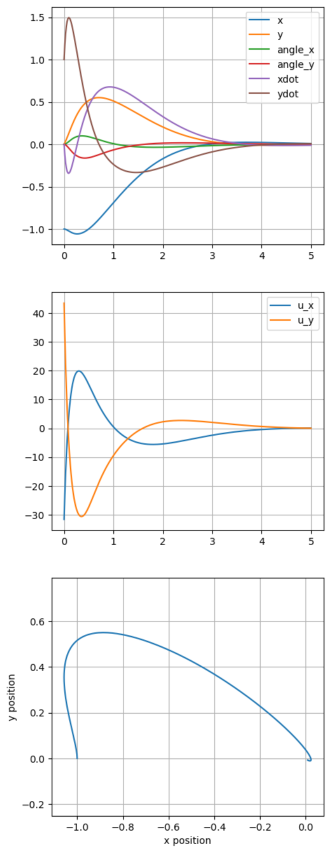
The advantages of trajectory optimization over LQR are clear even in simple stabilization tasks. The sets of initial conditions ($x = -1$, $\dot{y} = 1$) and ($x = -1$, $\dot{y} = 5$) where all other variables are zero are stabilized in Figure 2 and Figure 3 respectively. The cost weighting matrices used for LQR and Direct Collocation are the same, though the direct collocation carries with it a final time cost.

In the case with an initial velocity of 1, the control inputs, and state trajectories look almost identical, though LQR still takes much longer to settle than the trajectory optimization. In the case with an initial velocity of 5, the trajectory optimization is able to follow a state trajectory of a similar shape. However, the LQR controlled system experiences huge oscillations. The lean angle induced by the LQR controller is about 60 degrees, whereas the trajectory optimization only leans up to about 40 degrees. In this scenario trajectory optimization clearly performs better than LQR for high lean angles.

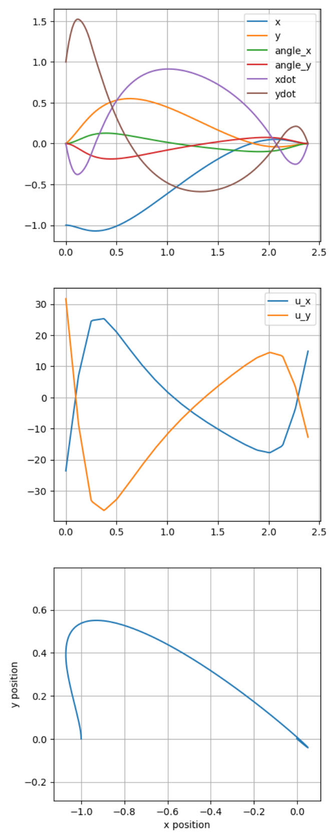
B. Initial State to Final State - 90 degree turns

From this case onward it is impossible to use standard LQR because the final state is not an equilibrium point. Analysis of the trajectory optimizations will have to be done without the reference point of another controller. The initial and final ground positions for these cases are $(0, -1)$ and $(1, 0)$. The trajectories are to have the same speed for the initial and final states. Two cases were tested: one with a boundary speed of 3 and the other with a boundary speed of 10.

The trajectory in Figure 4a takes the turn directly, whereas that in Figure 4b takes a 270 degree turn to take the 90 degree turn. This is likely because of state and input limits. The control inputs are clipped at the limit of 350 and one of the lean angles is clipped at the limit of 1 in 4b. This demonstrates that trajectory optimization can be highly intelligent under these constraints, even generating a trajectory that intersects itself.

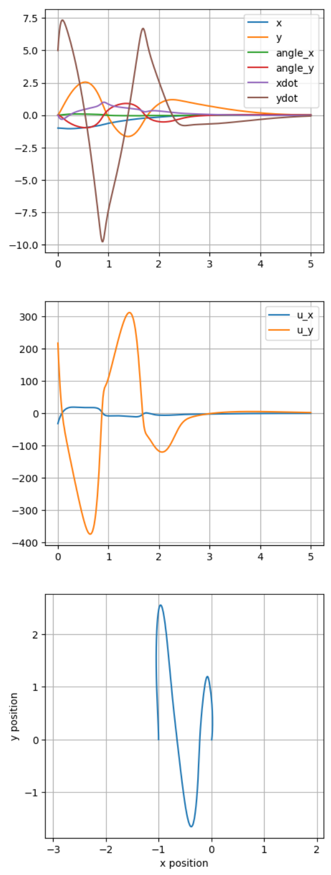


(a) LQR

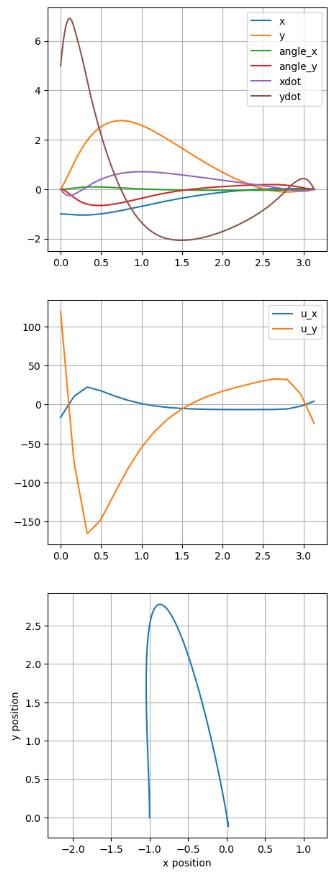


(b) Direct Collocation

Fig. 2: Stabilization of $v_0 = 1$

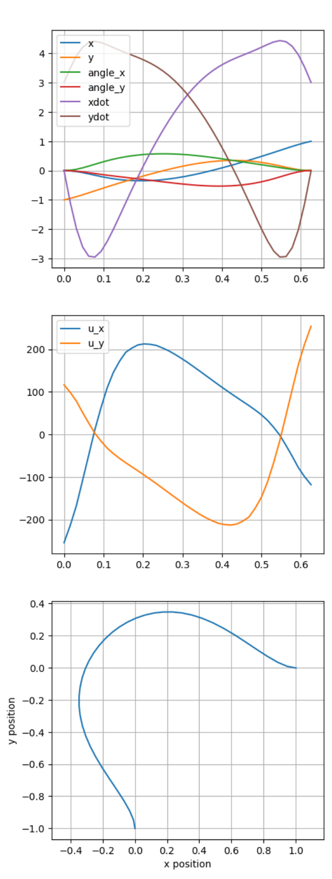


(a) LQR

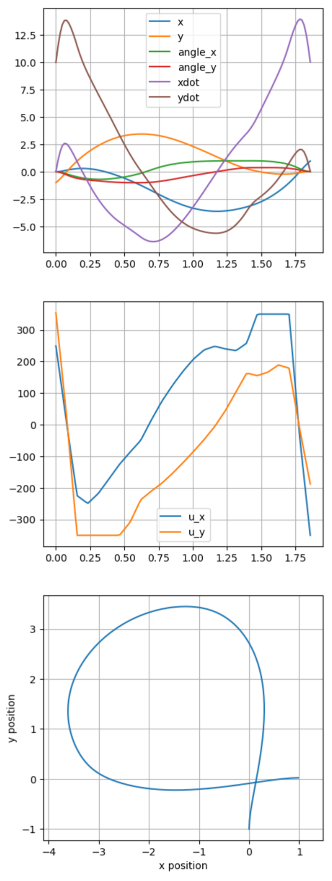


(b) Direct Collocation

Fig. 3: Stabilization of $v_0 = 5$

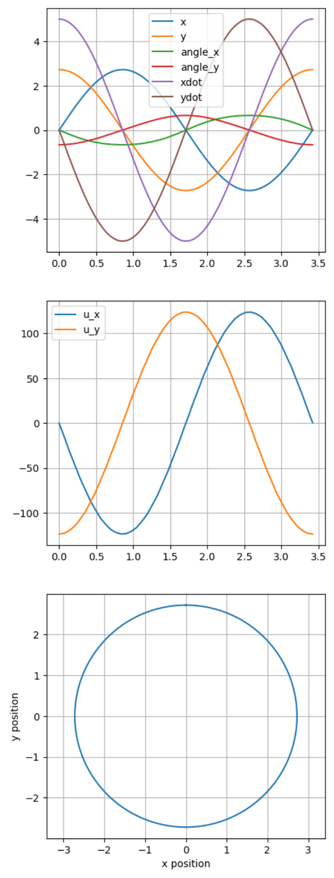


(a) $v_0, v_f = 3$

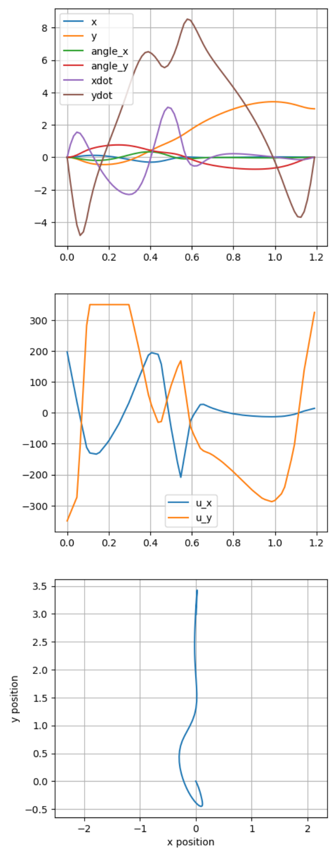


(b) $v_0, v_f = 10$

Fig. 4: Optimization of 90 degree turns



(a) Circular orbit



(b) Obstacle avoidance

Fig. 5

C. Circular Orbit

Trajectory optimization is capable of generating perfect circular orbit trajectories with lean angles of almost 45 degrees, as shown in Figure 5a.

D. Obstacle Avoidance

Obstacle avoidance situations might cause high lean angles for two reasons. One is that the robot may need to quickly burst around obstacles. The other is that the obstacle may physically conflict with the robot, requiring it to take evasive high lean angle maneuvers. Only the first scenario is explored here in Figure 5b. Trajectory optimization is used to quickly weave through obstacles and immediately come to rest afterwards.

IV. Conclusions and Next Steps

Trajectory optimization appears to be an effective technique for ballbot control in a variety of high lean angle scenarios. It can generate circular orbits, intelligently take turns, and move around obstacles. It slightly outperforms LQR for stabilization even for small lean angles and proves far more effective for large lean angles. It is able to harness the elegance of the ballbot platform in a way that linear control cannot.

A logical next step for this line of work might be to implement a Model Predictive Controller that can be used as a feedback controller.

References

[1] M. Shomin and R. Hollis, ”Differentially flat trajectory generation for a dynamically stable mobile robot,” 2013 IEEE International Conference on Robotics and Automation, Karlsruhe, Germany, 2013, pp. 4467-4472, doi: 10.1109/ICRA.2013.6631211.

[2] M. Shomin and R. Hollis, ”Fast, dynamic trajectory planning for a dynamically stable mobile robot,” 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, Chicago, IL, USA, 2014, pp. 3636-3641, doi: 10.1109/IROS.2014.6943072.