Agile Locomotion of a Ballbot with Obstacle Avoidance through Trajectory Optimization and Model Predictive Control

T. Kevin Best*

Fig. 1. A photo of the Michigan Robotics Ballbot. Three omniwheels roll on a basketball, allowing the ballbot to move in any direction. This passively unstable system requires active control for balance and locomotion.

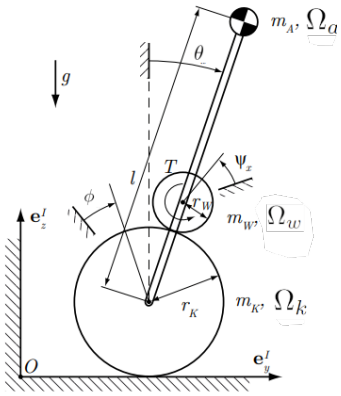

Fig. 2. Planar model of the ballbot, reproduced from [1]. Symbol definitions for inertial and distance variables can be found in Appendix I-A. State variables $q = [\phi, \theta]^\top$ define the rotations of the ball and robot body, respectively. Control torque T acts on the small roller.

***Abstract*— This work presents an optimization-based control scheme for a planar ballbot. An offline multiple shooting trajectory optimization uses a dynamic model of the system to design a minimum-time maneuver between two ballbot poses. Then, a model-predictive controller is used to track the planned trajectories, handle disturbances, and avoid unplanned obstacles. We demonstrate the control approach's efficacy in simulation, discuss its limitations, and suggest considerations for practical deployment to hardware.**

I. INTRODUCTION

The Michigan Robotics Ballbot (Fig. 1) is an underactuated, passively unstable system comprising a basketball and robot body built of three motor-driven omniwheels, control electronics, and batteries. There are two primary control goals for this project: 1) The robot body must remain balanced atop the basketball at all times. 2) The robot subsystem must be able to roll the ball in order to move around the world.

Previous work with this type of robot demonstrated impressive results with a Linear-Quadratic Regulator (LQR) approach [1]. A similar approach was used on a similar system in [2]. However, because the LQR solution is computed offline based on linearized dynamics, the authors resorted to a gain scheduling approach to handle the nonlinearities at different points in the state space. We hypothesize that a controller that considers the full nonlinear dynamic model will produce faster and more agile behaviors.

T. Kevin Best is a doctoral candidate with the Robotics Department, University of Michigan, Ann Arbor, MI 48109. tkbest@umich.edu
*Graduate Student

In this work, we apply a two step control approach: 1) We generate offline solutions through trajectory optimization and multiple shooting for the full nonlinear system. This step creates a plan of torque and state trajectories for the ballbot to follow. 2) We then apply online Model Predictive Control (MPC) to track the offline trajectory. The MPC controller stabilizes around the planned trajectories, rejects disturbances, and modifies the plan to avoid obstacles.

II. BALLBOT DYNAMICS AND SIMULATION

A. Planar Equations of Motion

To simplify the problem, we consider a planar version of the ballbot (Fig. 2). The planar model contains two degrees of freedom $q \in \mathbb{R}^2 = [\phi, \theta]^\top$, where ϕ represents the angle that the ball has rolled and θ represents the angle of the robot body, both relative to the vertical. We define Cartesian coordinate directions e_y and e_z spanning the plane. Note that the horizontal position of the ball $p_{e_1} = \phi r_K$ is easily calculated from the state vector. There is a single torque input $T \in \mathbb{R}$ applied to a wheel that rolls on the ball without slipping.

Similar to the approach presented in [1], we used Lagrangian Mechanics to calculate the system's governing equations. The derivations of the potential and kinetic energies of the ball (V_k, T_k), actuating wheel (V_w, T_w), and robot body (V_a, T_a), as well as the non-potential forces F_{np}, are given in Appendix I-A. Defining the Lagrangian $\mathcal{L} = T_k + T_w + T_a - V_k - V_w - V_a$, we can write the Euler-

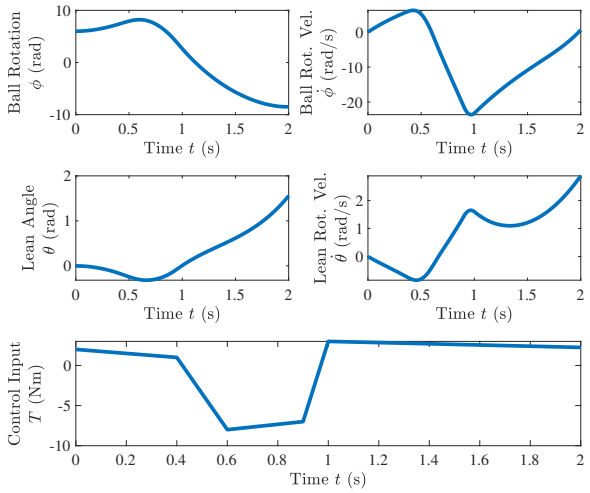

Fig. 3. State and input trajectories for the validation simulation. The random hand-selected piece-wise linear control input produces physically intuitive results, suggesting that the simulation is correct.

Lagrange equation as

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}}\right) - \frac{\partial \mathcal{L}}{\partial q} - F_{np} = 0. \tag{1}$$

The MATLAB Symbolic Math Toolbox (R2022b) was used to solve (1) for the state accelerations $\ddot{q}$ in terms of an inertial matrix $M(q)$ and a coriolis, gravitational, and input function $f(q, \dot{q}, T)$:

$$\ddot{q} = M(q)^{-1} f(q, \dot{q}, T). \tag{2}$$

B. Simulation

A simulation of the ballbot dynamics (2) was implemented in MATLAB using the integration method `ode45`. In the simulation, we assumed a direct measurement of both state variables q and an ideal actuator (no inertia, friction, etc.) acting on the actuating wheel with the torque input T. To validate the simulation, the system's response to a random hand-selected piecewise-linear control input was simulated and the resulting state and velocity trajectories were plotted (Fig. 3). An animation was also constructed, allowing visual validation that the dynamics behaved as expected. The animation video `HelloWorld_Ballbot.mp4` is available in the supplemental material. The simulation code is available on GitHub at `+Ballbot/runSimulation.m`.

Further validation was performed by checking the simulation for conservation of energy. The power put into the system P_{in} is given by the inner product of the non-potential forces F_{np} and the minimal coordinate velocities $\dot{q}$, *i.e.* $P_{in} = F_{np}^{\top}\dot{q}$. Therefore, the energy injected into the system $E(t)$ at a given time t is given by

$$E(t) = \int_0^t (F_{np}(\tau))^{\top} \dot{q}(\tau) d\tau. \tag{3}$$

The system's total energy and the input energy $E(t)$ were calculated throughout the simulation and plotted in Fig. 4.

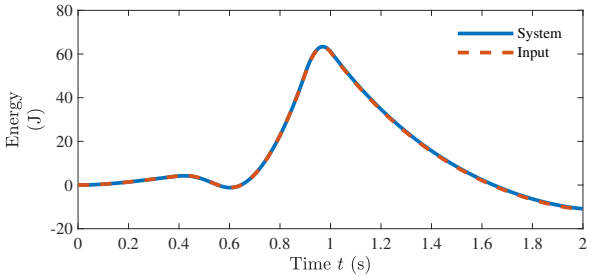

Fig. 4. The total energy trajectory of the system (solid) along with the energy injected via the non-potential forces (dashed). The two curves lie atop one another, illustrating that energy is conserved in the simulation.

The simulation's initial condition was static with $q = [6, 0]^{\top}$. We defined zero potential energy in this configuration as well ($\theta = 0$), allowing the Ballbot's total energy to begin at 0.0 J. As there are no losses in the system, the energy input trajectory matching the total energy trajectory confirms that energy is conserved and that the simulation obeys physics.

III. Optimal Control Methods

Our control objective is to move the ballbot from a current configuration q_0 to a final configuration q_f in a minimum time, subject to system constraints and actuator torque limitations. For more compact notation, let $z = [\phi, \theta, \dot{\phi}, \dot{\theta}]^{\top}$, and define z_0 and z_f as the corresponding initial and final states, respectively. We use trajectory optimization to produce an open-loop control plan and an MPC controller to perform closed-loop control to stabilize around the plan in a feasible manner. If any obstacles prevent the ballbot from executing the pre-planned trajectory, the MPC controller will avoid them.

A. Offline Trajectory Optimization

1) Problem Formulation: For the offline trajectory generation, we implemented a nonlinear multiple-shooting optimization. We define a linear spline of control inputs T over N knot points. The goal of the multiple shooting optimization is to find the control inputs for a minimum time maneuver between the starting and the ending configurations. Thus, the cost function $J_{ms}(x)$ is simply

$$J_{ms}(x) = T_f, \tag{4}$$

where T_f is the duration of the movement. The decision vector $x \in \mathbb{R}^{5N+1}$ contains n torque inputs defining the spline, n angles and velocities for both ϕ and θ, and T_f. Bounds vectors x_{min} and x_{max} are included to ensure that the selected state trajectories are physically feasible, that actuator torque limits are obeyed, and that the maneuver time is positive. We select these constraints to be more strict than the actual limitations of the system, allowing the MPC controller room around the trajectory to modify it as needed. At each knot point, we enforce $-8 \leq \phi \leq 12$ rad, $-0.52 \leq \theta \leq 0.52$ rad, $-75 \leq \dot{\phi} \leq 75$ rad/s, $-10 \leq \dot{\theta} \leq 10$ rad/s, $-10 \leq T \leq 10$ Nm, and $T_f \geq 0.01$ s.

In addition to the decision vector bounds, we add equality constraints on the initial and final states the initial condition and desired final state:

$$C_{task}(x) = 0 = \begin{bmatrix} z - z_0 \\ z - z_f \end{bmatrix}. \tag{5}$$

To ensure that the optimization obeys the ballbot's dynamics (2), we also include defect constraints between the knot points. Each 4×1 sub-vector $C^i_{def}(x)$ of the vector valued function $C_{def}(x) = 0$ is defined $\forall i \in \{1, \ldots, N-1\}$ as

$$C^i_{def}(x) = z(t_{i+1}) - z(t_i) - \int_{t_i}^{t_{i+1}} \dot{z}(t)dt. \tag{6}$$

In total, the trajectory optimization problem is given by

$$\begin{aligned} \arg\min_{x} \quad & J_{ms}(x) \\ \text{subject to} \quad & C_{task}(x) = 0, \\ & C_{def}(x) = 0, \\ & x_{min} \leq x \leq x_{max} \end{aligned} \tag{7}$$

2) Problem Solution: The nonlinear program (7) was solved via `fmincon` in MATLAB using the `sqp` algorithm, which requires an initial guess for the optimal decision vector x_0. After experimentation with different values of x_0, a combination of initial guesses was found that worked well for various problems. We that assumed half the state trajectory was at $z = z_0$ and the other half at $z = z_f$. The initial maneuver time guess was 1.5 s and that the initial torque trajectory was -10 Nm for the first entry and 10 Nm for the final entry with zeros everywhere else. The multiple shooting solution code is available on GitHub at `+Control/optimize_trajectory_MS.m`.

The dynamic simulations required to calculate the defect constraints are computationally intensive. To speed up this process, the evaluation of (6) was implemented in a `parfor` loop, allowing MATLAB to parallelize the simulation of each shot. Even with this improvement, it took on average 90 s to solve (7) with only $N = 11$ knot points on a desktop PC (i7-9700 CPU @ 3.00 GHz). Increasing N to 26 knot points took 975 s on average, which is prohibitively slow for practical use. To remedy this issue, MATLAB Coder was used to compile the MATLAB code into a compiled C++ `mex` file. This increased the speed to 0.8 s for $N = 11$ and 8.9 s for $N = 26$ nodes. With this increase in speed, it is much more reasonable to make a plan "on-the-fly" before starting the maneuver, potentially even on embedded hardware.

3) Example Maneuver: To test the multiple shooting optimization, we tasked it with finding the optimal control trajectory between a stationary pose at $q_0 = [-0.5/r_k, 0]^\top$ and $q_f = [1/r_k, 0]^\top$ in minimum time. Fig. 5 shows the identified optimal control input and state trajectories and Fig. 6 shows still photos taken from an animation. We notice in Fig. 5 that the optimization was able to correctly exploit the non-minimum phase dynamics of the system, driving the state errors to worse values before making them get better. Additionally, we notice that the optimal solution pushes against the torque limits, suggesting that the result is indeed a minimum time solution and that higher torque limits would allow a faster maneuver. An animation video `testMS.mp4` is available in the supplemental media.

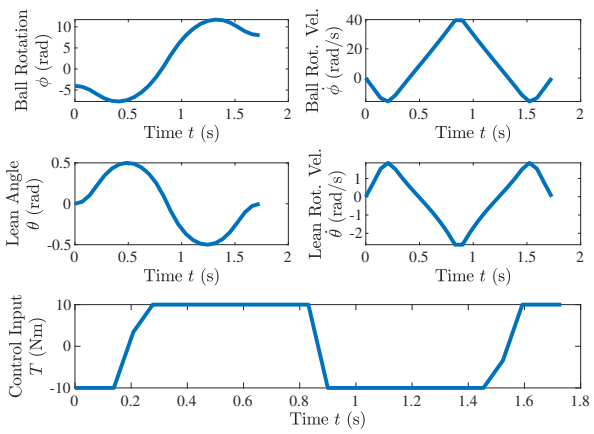

Fig. 5. The optimal control torque trajectory T and states identified by the multiple shooting optimization for moving from a stationary pose at $q_0 = [-0.5/r_k, 0]^\top$ to $q_f = [1/r_k, 0]^\top$ in minimum time.

B. Real-Time Model Predictive Controller

The control trajectory found by multiple shooting is a feed-forward controller rather than a feedback controller, making it unable to accommodate disturbances, model errors, and unforeseen obstacles. Therefore for real-time use, we use online Model-Predictive Control (MPC) to track the planned trajectory via direct collocation.

The MPC controller is run iteratively at a fixed rate where, at each iteration, it calculates an input torque command given a new measurement of the system state. This torque command is selected by allowing an optimization program to select the upcoming state and control trajectories over a prediction horizon t_h, subject to the system dynamics and other system constraints. If we approximate the trajectories via splines at n_k horizon points, the decision vector $x \in \mathbb{R}^{5n_k}$ contains n_k torque values T_N and n_k values for each of the 4 states, denoted in vector form as z_N.

In order to efficiently solve this optimization problem online, we formulate it as a convex quadratic program (QP) with a quadratic cost function and affine constraints. That is, we select x^* as

$$\begin{aligned} \arg\min_{x} \quad & \frac{1}{2} x^\top H x + c^\top x \\ \text{subject to} \quad & Ax \leq b, \\ & A_{eq} x = b_{eq} \end{aligned} \tag{8}$$

where $H \succ 0$ and c are the quadratic and linear terms of the cost function, and A, b, A_{eq}, and b_{eq} define the inequality and equality constraints, respectively. The following sections detail how each of matrices and vectors that define the QP is calculated.

1) Notation: We denote the solution to (8) as x^*. From x^*, we can extract the optimal state trajectories z^*_N and torque commands T^*_N. As MPC is an iterative process, it is useful to

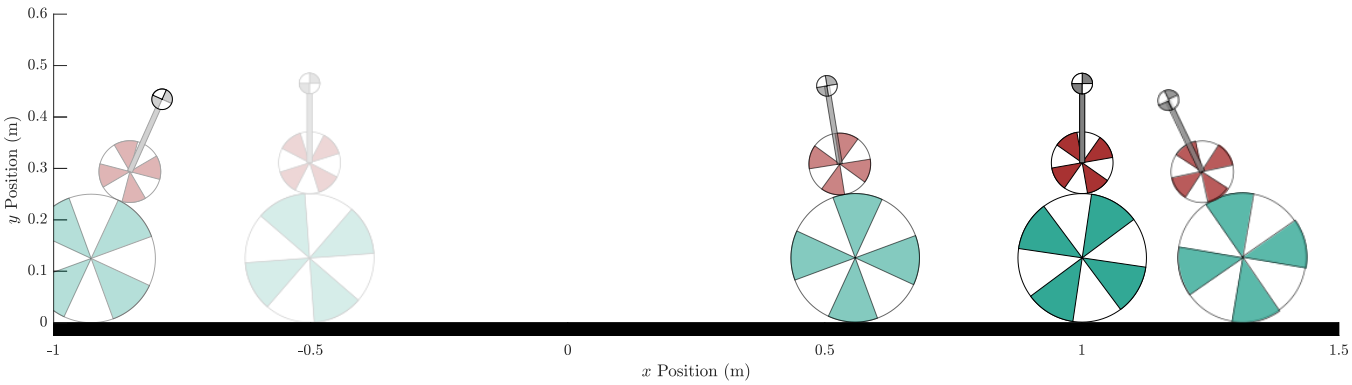

Fig. 6. A timelapse of the identified optimal solution for moving from a stationary pose at $q_0 = [-0.5/r_k, 0]^\top$ to $q_f = [1/r_k, 0]^\top$ in minimum time given by the multiple shooting optimization. Increasing levels of opacity indicate increasing time.

define x^{-*} as the solution to (8) from the previous iteration. Likewise, define z_N^{-*} and T_N^{-*} as the previous iteration's optimal state and control trajectories, respectively extracted from x^{-*}. The time between knot points, Δt, is calculated as $t_h/(n_k - 1)$.

2) Defect Constraints: Defect constraints are included to ensure that the optimization selects state trajectories that obey the system dynamics. In order to write these constraints in the affine form $A_{eq}x = b_{eq}$, we approximate the time-evolution of the system using trapezoidal integration. We enforce these constraints as a vector-valued function $g_{def}(x) = 0$, where we define the sub-vector $g^i_{def}(x)$ for each segment i between knot points as

$$g^i_{def}(x) = z_i - z_{i+i} + \frac{\dot{z}_i + \dot{z}_{i+1}}{2}\Delta t. \tag{9}$$

However, the ballbot dynamics (2) are nonlinear, which we denote in first-order form as $\dot{z} = h(z, T)$. In order to write the dynamics in the form of an affine constraint, we calculate a first-order approximation of the dynamics at a state z^* and control torque T^* as

$$\begin{aligned}\dot{z} \approx & h(z^*, T^*) + A_d|_{z=z^*, T=T^*}(z - z^*) \\ & + B_d|_{z=z^*, T=T^*}(T - T^*),\end{aligned} \tag{10}$$

where

$$A_d = \frac{\partial h(z,T)}{\partial z} \text{ and } B_d = \frac{\partial h(z,T)}{\partial T}. \tag{11}$$

For each of the $i \in \{1, ..., n_k - 1\}$ segments between knot points, we calculate the sub-vector $g^i_{def}(x)$ using (10) and concatenate the results. To make the best possible approximation of the true dynamics, we calculate this approximation about z_N^{-*} and T_N^{-*}, as the previous iteration's solution is likely close to the actual upcoming trajectories. Finally, we convert the vector-valued constraint function $g_{def}(x)$ into standard form as

$$A_{eq} = \frac{\partial g_{def}(x)}{\partial x}, \text{ and } B_{eq} = -g_{def}(0). \tag{12}$$

3) Obstacle Constraints: The MPC controller must modify the planned trajectories to avoid obstacles that impede the ballbot's path. These obstacles, which we approximate as round areas in the sky (see Fig. 8), require the ballbot to "duck" underneath them to avoid a collision. Note that we define a larger than necessary obstacle radius to ensure sufficient clearance around it for the ballbot's center of mass. The buffer zone is shown in a dashed line whereas the true obstacle is shown in black. The Cartesian location of the ballbot's tallest point p_{com} (*i.e.* its center of mass) is a function of its configuration q, given by

$$p_{com}(q) = \begin{bmatrix} \phi r_k + l \sin\theta \\ r_k + l\cos\theta \end{bmatrix}. \tag{13}$$

Recall that the ballbot exists in the plane spanned by coordinate vectors e^y in the horizontal direction and e^z in the vertical direction. Given an obstacle's location p_{obs} and radius r_{obs}, we define a quadratic constraint ceiling e^z_{max} based on the center of mass's horizontal position p^y_{com} as

$$e^z_{max}(q) = h_1(q) = p^z_{obs} - r_{obs} + \beta\left(p^y_{com} - p^y_{obs}\right)^2, \tag{14}$$

where β is a tunable scalar that defines the steepness of the quadratic constraint ceiling. Directly beneath the obstacle, the maximum height is the obstacle height less its radius. On either side of the obstacle, this maximum height increases quadratically. However, in order to implement (14) as an affine inequality constraint, we must linearize it about the expected configuration trajectory given by the previous solution q^{-*}, similar to the defect constraints. At each point along the horizon, this approximation is linear in q, given by

$$e^z_{max}(q) \approx h_1(q^{-*}) + \frac{\partial h_1}{\partial q}\left(q - q^{-*}\right). \tag{15}$$

Next, we similarly write a first order approximation of p^z_{com} using (13) as

$$p^z_{com}(q) \approx p_{com}(q^{-*}) + \frac{\partial p_{com}(q)}{\partial q}(q - q^{-*}). \tag{16}$$

Evaluating these approximations at each horizon point i, we write the inequality constraint $g_{obs}(x) \leq 0$, defined as

$$g^i_{obs}(x) = p^z_{com}(q_i) - e^z_{max}(q_i) \leq 0. \tag{17}$$

4) System Constraints: As in the offline trajectory optimization, we implemented system constraints to ensure feasible state and control trajectories. We select less strict limits than the trajectory optimization to allow the MPC controller room to modify the plan in order to avoid obstacles. Mainly, we allow for much more torque to enable the ducking behavior. At each knot point, we enforce $-24 \leq \phi \leq 24$ rad, $-\pi/4 \leq \theta \leq \pi/4$ rad, $-112.5 \leq \dot{\phi} \leq 112.5$ rad/s, $-15 \leq \dot{\theta} \leq 15$ rad/s, and $-50 \leq T \leq 50$ Nm. We collect these constraints into the vector-valued function $g_{sys}(x) \leq 0$, which is linear in x.

We concatenate the obstacle constraints $g_{obs}(x)$ with the system constraints $g_{sys}(x)$ to get the full inequality constraint function $g_{ineq}(x) \leq 0$. We convert this inequality function into standard form as

$$A = \frac{\partial g_{ineq}(x)}{\partial x}, \text{ and } B = -g_{ineq}(0). \tag{18}$$

5) Cost Function: We select a standard quadratic cost function $J_1(x)$ that penalizes deviation from the planned torque and state trajectories ($\hat{T}(t)$ and $\hat{z}(t)$) over the finite time horizon t_h at a current time t_0 as

$$J_1(x) = \int_{t_0}^{t_0+t_h} \left(\tilde{z}(t)^\top Q \tilde{z}(t) + \tilde{T}(t)^\top R \tilde{T}(t) \right) dt, \tag{19}$$

where

$$\tilde{z}(t) = z(t) - \hat{z}(t), \tag{20}$$

$$\tilde{T}(t) = T(t) - \hat{T}(t), \tag{21}$$

and Q and R are square penalty weighting matrices. The values $Q = \text{diag}([50, 1, 25, 1])$ and $R = 0.05$ produced satisfactory results. Discretizing $\tilde{z}(t)$ and $\tilde{T}(t)$ at each horizon point as $\tilde{z}_N$ and $\tilde{T}_N$ respectively, we can approximate (19) as a quadratic function in x through Euler integration as

$$J_1(x) \approx \left[\tilde{z}_N^\top (I_{n_k} \otimes Q) \tilde{z}_N + \tilde{T}_N^\top (I_{n_k} \otimes R) \tilde{T}_N \right] \Delta t, \tag{22}$$

where I_{n_k} is the $n_k \times n_k$ identity matrix and $\otimes$ denotes the kronecker product.

In addition, we observed that the solution had the tendency to oscillate between two solutions of similar cost from timestep to timestep. We hypothesize that this is due to the nonlinearities of system dynamics and our approximations of them. The end result was an "indecisive controller," which would repeatedly change plans and never make progress towards the goal. To avoid this behavior, we add a second cost term $J_2(x)$ that penalizes changes in the planned state and torque trajectories through an inner product:

$$J_2(x) = \alpha (x - x^{-*})^\top (x - x^{-*}), \tag{23}$$

where $\alpha = 1$ is a tunable weighting parameter. Finally, we convert the total cost function $J(x) = J_1(x) + J_2(x)$ into standard form with $\mathbf{H}(\cdot)$ denoting the Hessian operation as

$$H = \mathbf{H}(J(x)), \quad c^\top = \left. \frac{\partial J(x)}{\partial x} \right|_{x=0}. \tag{24}$$

We note that the sum-of-squares definitions of (22) and (23) ensure that $H \succ 0$ and that the QP is convex with a guaranteed global minimum.

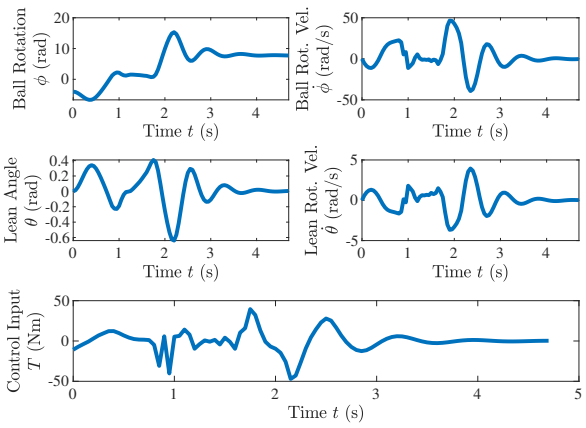

Fig. 7. The optimal control torque trajectory T and states calculated by the MPC controller in real time for moving from a stationary pose at $q_0 = [-0.5/r_k, 0]^\top$ to $q_f = [1/r_k, 0]^\top$ with an obstacle located at $p_{obs} = [0.25, 0.5]^\top$ with $r_{obs} = 0.05$ m.

6) Problem Solution: The constants for program (8) were calculated in MATLAB using the Symbolic Math Toolbox. The QP was solved using MATLAB's `quadprog` function with the convex interior point method. The functions were collected into a MATLAB package and can be run with the `+Control/+MPC/setup.m` and `+Control/+MPC/run.m` functions available on GitHub.

A time horizon of $t_h = 2.5$ s was selected, as shorter time horizons were unable to see far enough into the future to leverage the non-minimum phase dynamics. On the same desktop PC (i7-9700 CPU @ 3.00 GHz), an iteration of the MPC controller using $n_k = 51$ took on average 50 ms, depending on the difficulty of the obstacle. After using MATLAB Coder to create a `mex` file, an iteration of MPC took on average 35 ms. Note that in order to generate the compiled code, the QP solution method was changed to active set, but no difference in solution was observed, which makes sense as the problem is convex. This marginal improvement in computation time for the compiled code suggests that the interior point method is likely more efficient than the active set method for this problem.

7) Example Maneuver with Obstacles: We tested the MPC controller in simulation at 20 Hz. We tasked the MPC controller to follow the planned maneuver from the trajectory optimization (Fig. 5) while avoiding an obstacle at the coordinates $p_{obs} = [0.25, 0.5]^\top$ with $r_{obs} = 0.05$ m. The resulting state trajectories are shown in Fig. 7 and a timelapse of the animation is shown in Fig. 8. The controller starts by following the planned state trajectories until it changes plan due to the obstacle. It stops, reverses so that it can tilt forward, and then proceeds underneath the obstacle. An animation video `testMS_MPC_obstacle.mp4` is available in the supplemental media.

8) Example Maneuver with Actuator Noise: To further test the MPC controller, another simulation was implemented that included scaled normal random noise $\tilde{T} \sim \mathcal{N}(0, 4)$ Nm on the actuator torque to represent a non-ideal actuator.

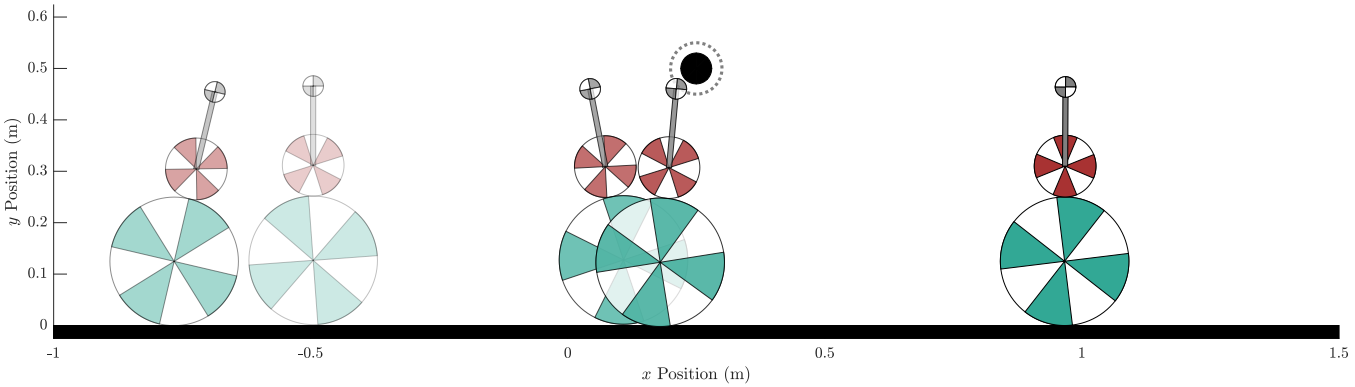

Fig. 8. A timelapse of the ballbot under the MPC controller, moving from $q_0 = [-0.5/r_k, 0]^\top$ to $q_f = [1/r_k, 0]^\top$ with an obstacle located at $p_{obs} = [0.25, 0.5]^\top$ with $r_{obs} = 0.05$ m. The dashed line of the obstacle shows the area that the center of the center of mass must not enter in order to achieve a safety margin. Increasing levels of opacity indicate increasing time.

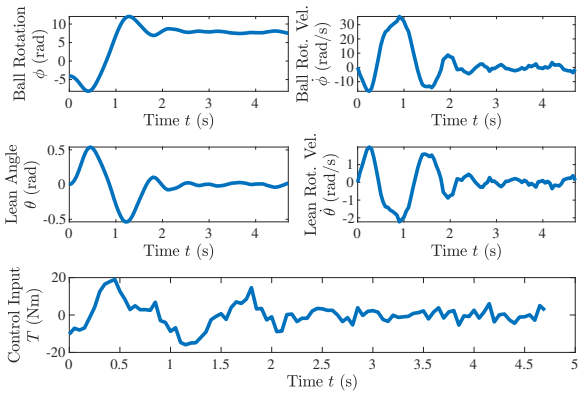

Fig. 9. The optimal control torque trajectory T and states calculated by the MPC controller in real time for moving from a stationary pose at $q_0 = [-0.5/r_k, 0]^\top$ to $q_f = [1/r_k, 0]^\top$ in the presence of actuator noise $T \sim \mathcal{N}(0, 4)$ Nm.

Again, the simulation was run at 20 Hz and the controller attempted to follow the planned trajectory from Fig. 5. Because the MPC controller replans the optimal trajectory online, it was able to handle these disturbances and still regulate to the desired position (Fig. 9). An animation video `testMS_MPC_actuatorNoise.mp4` is available in the supplemental media.

C. Discussion and Limitations

The offline trajectory optimization generated reasonable and agile trajectories for a variety of initial and final conditions. Although the multiple shooting optimization problem is non-convex, the solution was not very sensitive to the initial guess, and the initial guess detailed above produced a solution for all tested cases. It is possible that trying a larger number of initial guesses may yield faster maneuvers.

Perhaps the biggest limitation of the multiple shooting optimization is its computational complexity. Although it only required 8.9 s for 26 nodes, this time would likely be larger on the robot's embedded hardware. While waiting less than 10 seconds before starting a maneuver may be allowable, waiting 30 seconds may not be. Future work should investigate the optimal number of nodes to include in the optimization to balance the computational burden with the quality of the solution on the embedded hardware.

The MPC controller also worked well tracking a variety of plans generated by the multiple shooting optimization. The only times we observed it fail was if the obstacle was too low for it to pass underneath without violating the system constraints, primarily the torque constraint. However, we did notice that the MPC controller would always elect to pass under the obstacle by leaning "forward" instead of "backward." Even in cases where it would have been more efficient to lean backward, the controller would stop the ballbot, have it tip forwards, and then proceed under the obstacle. We hypothesize that the linearization of the obstacle constraints causes this behavior. Future work should investigate alternative constraint formulations that avoid this limitation.

IV. Conclusions and Future Work

This article presented a hierarchical controller for a planar ballbot based on trajectory optimization and model predictive control. The trajectory optimization utilized a multiple shooting approach to generate a plan to move the ballbot from one pose to another based on a simulation of the ballbot's dynamics. Then, a model predictive controller stabilized the planned trajectory, rejected disturbances, and avoided obstacles.

In future work, we will deploy the control strategy to the physical system to test performance. Additionally, the planar model could be extended to the full three-dimensional system and the same control strategy could be tested.

References

[1] P. Fankhauser and C. Gwerder, "Modeling and control of a ballbot," Bachelor's Thesis, ETH Zurich, 2010, online: https://www.research-collection.ethz.ch/bitstream/handle/20.500.11850/154271/eth-7943-01.pdf.

[2] T. Lauwers, G. Kantor, and R. Hollis, "A dynamically stable single-wheeled mobile robot with inverse mouse-ball drive," in *Proceedings IEEE International Conference on Robotics and Automation, ICRA*, 2006, pp. 2884–2889.

APPENDIX I
DETAILED CALCULATIONS

A. Energy Derivations

The following energy derivations are reproduced from [1]. Given an assumption of rolling without slipping, the kinetic energy of the ball T_k is given by

$$T_k = 1/2\left((m_k r_k^2 + \Omega_k)\dot{\phi}^2\right), \tag{25}$$

where m_k, r_k, and Ω_k are the ball's mass, radius, and moment of inertia, respectively. Since the ground is flat, there is no change in potential energy for the ball ($V_k = 0$). The kinetic energy of the actuating wheel T_w is similar, but includes a coupling term:

$$\begin{aligned} T_w = \frac{1}{2} m_w \left(r_k^2\dot{\phi}^2 + 2r_T\cos(\theta)\dot{\theta}r_k\dot{\phi} + (r_T)^2\dot{\theta}^2\right), \\ + \frac{1}{2}\Omega_w\left(\frac{r_k}{r_w}(\dot{\phi}-\dot{\theta}) - \dot{\theta}\right)^2, \end{aligned} \tag{26}$$

where $r_T = r_k + r_w$, and m_w, r_w, and Ω_w are the actuating wheel's mass, radius, and moment of inertia, respectively. The potential energy V_w due to gravity with acceleration g is given by

$$V_w = m_w g r_T \cos\theta. \tag{27}$$

Finally, the kinetic and potential energies, T_a and V_a, of the robot body sitting atop the ball at a center of mass distance l are given by

$$\begin{aligned} T_a =& \frac{1}{2} m_a \left(r_k^2\dot{\phi}^2 + 2l\cos(\theta)\dot{\theta}r_k\dot{\phi} + l^2\dot{\theta}^2\right) \\ &+ \frac{1}{2}\Omega_a\dot{\theta}^2, \end{aligned} \tag{28}$$

$$V_a = m_a g l\cos\theta, \tag{29}$$

where m_a and Ω_a are the robot body's mass and moment of inertia, respectively.

The non-potential forces F_{np} are created by the input torque acting on the actuator wheel coordinate ψ. The configuration of the wheel ψ in terms of the generalized coordinates is

$$\psi = \frac{r_k}{r_w}(\phi - \theta) - \theta. \tag{30}$$

Therefore, the non-potential force expressed in the generalized coordinates is

$$F_{np,1} = \begin{bmatrix} \frac{r_k}{r_w}T \\ -(1 + \frac{r_k}{r_w})T \end{bmatrix}. \tag{31}$$

The reaction torque acts on the θ coordinate of the body, which has opposite sign. Therefore,

$$F_{np,2} = \begin{bmatrix} 0 \\ T \end{bmatrix}. \tag{32}$$

The sum of these two non-potential forces then simplifies to

$$F_{np} = \frac{r_k}{r_w}\begin{bmatrix} T \\ -T \end{bmatrix}. \tag{33}$$