2011 IEEE International Conference on Robotics and Automation
Shanghai International Conference Center
May 9-13, 2011, Shanghai, China

# First-exit model predictive control of fast discontinuous dynamics: Application to ball bouncing

Paul Kulchenko and Emanuel Todorov

*Abstract*— **We extend model-predictive control so as to make it applicable to robotic tasks such as legged locomotion, hand manipulation and ball bouncing. The online optimal control problem is defined in a first-exit rather than the usual finite-horizon setting. The exit manifold corresponds to changes in contact state. In this way the need for online optimization through dynamic discontinuities is avoided. Instead the effects of discontinuities are incorporated in a final cost which is tuned offline. The new method is demonstrated on the task of 3D ball bouncing. Even though our robot is mechanically limited, it bounces one ball robustly and recovers from a wide range of disturbances, and can also bounce two balls with the same paddle. This is possible due to intelligent responses computed online, without relying on pre-existing plans.**

## I. INTRODUCTION

Numerical optimization is a powerful tool for automatic control. The biggest challenge is the curse of dimensionality: the state space of most interesting robotic systems is too large to construct a control law that generates sensible (let alone optimal) commands in all states. Yet such global control laws are necessary if robots are to achieve rich and versatile behavior in the presence of uncertainty. Model-predictive control (MPC) is remarkably good at avoiding the curse of dimensionality when it works [1]–[3]. The idea is simple: run the optimization in real time as the behavior unfolds, and compute an optimal trajectory up to some time horizon at each step of the control loop. This trajectory always starts at the current (measured or inferred) state, thus optimization for all states is avoided. The initial portion of the trajectory is used to control the system, the clock then advances, and the process is repeated. The unused portion of the trajectory is useful for initializing the (usually iterative) optimizer at the next time step.

The main challenge in applying MPC is the requirement for real-time optimization. Indeed this challenge is so formidable that MPC has rarely been used in robotics. Its typical application is in the control of chemical processes where the dynamics are sufficiently slow. The most impressive robotics application we are aware of is the work [4] on aerobatic helicopter flight. This problem is simpler than the problems studied here in two important ways: the dynamics are smooth, and the control objective can be formalized as tracking a pre-specified trajectory.

This work is supported by the US National Science Foundation.

Paul Kulchenko is with the Department of Computer Science & Engineering, University of Washington paul@kulchenko.com

Emanuel Todorov is with the faculty of the Departments of Applied Mathematics and Computer Science & Engineering, University of Washington todorov@cs.washington.edu

Our goal is to develop MPC methods applicable to discontinuous (and in particular contact) dynamics arising in a variety of robotic tasks: legged locomotion, hand manipulation, ball bouncing. Apart from the above requirement for fast optimization, this is a challenging application domain because numerical optimization does not work well in the presence of discontinuities, and also because such tasks involve significant under-actuation and uncertainty, making pre-specified trajectories rather useless.

This paper makes two contributions. First, we generalize MPC from the usual finite horizon (or receding horizon) setting to a first-exit setting, which allows us to avoid dealing with discontinuities in the online optimization phase. We also generalize our iterative linear quadratic Gaussian (iLQG) algorithm [5] to first-exit settings and use it to handle the online optimization. Second, we describe suitable cost functions and an overall system design which enable us to apply MPC to the problem of ball bouncing. Even though we used a robot with significant mechanical limitations (small workspace and no tilt control), we were able to bounce one ball robustly under a wide range of disturbances, as well as bounce two balls on the same paddle – see attached video.

Ball-bouncing has received considerable attention [6]–[10]. Most of that work has focused on analysis of (usually passive) stability. Indeed one of the more remarkable findings has been that passive stability in the vertical direction requires hitting the ball with negative acceleration, and that humans exploit this strategy [8], in general agreement with the idea that the brain optimizes motor behavior [11]. While we appreciate the elegance of simple control solutions, it seems clear that complex real-world behaviors require rather advanced feedback mechanisms. It remains to be seen what the utility of simple solutions will be once such feedback mechanisms are in place. For example, we found that stabilization in lateral directions is harder than stabilization in the vertical direction, and any controller smart enough to achieve lateral stability also achieves vertical stability. We also observed that if the paddle can only translate but cannot rotate (due to mechanical constraints in the robot), the task becomes much harder for a human – suggesting that future psychophysics studies should perhaps look more closely at paddle orientation in human data. Overall we do not believe that any prior work on ball bouncing comes close to our results in terms of recovering from large unmodeled perturbations. Such recovery (and more generally the ability to invent rich behavior on the fly) is a key selling point of MPC, and ball-bouncing is a domain where it really makes a difference.

978-1-61284-385-8/11/$26.00 ©2011 IEEE

## II. First-exit model predictive control

We describe our general control methodology in this section, and specialize it to ball-bouncing in subsequent sections.

### A. Infinite-horizon and first-exit problems

MPC is normally applied to tasks that continue indefinitely. Thus we formalize the task as an infinite-horizon average-cost stochastic optimal control problem, with dynamics given by the transition probability distribution $p(x'|x,u)$. Here $x$ is the current state, $u$ the current control, and $x'$ the resulting next state. The states and controls can be discrete or continuous. Let $\ell(x,u)$ be the immediate cost for being in state $x$ and choosing control $u$. It is known that the differential[1] optimal cost-to-go function $\widetilde{v}(x)$ satisfies the Bellman equation

$$c + \widetilde{v}(x) = \min_u \left\{ \ell(x,u) + E_{x' \sim p(\cdot|x,u)} \widetilde{v}(x') \right\} \qquad (1)$$

where $c$ is the average cost per step. The solution is unique under suitable ergodicity assumptions.

Now consider a first-exit stochastic optimal control problem with the same dynamics $p$, immediate cost $\widehat{\ell}(x,u)$, and final cost $h(x)$ defined on some subset $\mathcal{T}$ of terminal states. In such problems the total cost-to-go $v$ is finite and satisfies the Bellman equation

$$v(x) = \min_u \left\{ \widehat{\ell}(x,u) + E_{x' \sim p(\cdot|x,u)} v(x') \right\} \qquad (2)$$

for $x \notin \mathcal{T}$, and $v(x) = h(x)$ for $x \in \mathcal{T}$.

We now make a simple but critical observation:

*Lemma 1:* If $h(x) = \widetilde{v}(x)$ for $x \in \mathcal{T}$ and $\widehat{\ell}(x,u) = \ell(x,u) - c$ for all $(x,u)$, then $v(x) = \widetilde{v}(x)$ for all $x$.

This result follows from the fact that when $\widehat{\ell}(x,u) = \ell(x,u) - c$ the two Bellman equations are identical. Note that the cost offset $c$ affects the optimal cost-to-go function but does not affect the optimal control law (i.e. the $u$ that achieves the minimum for each $x$).

Thus we can find the optimal solution to an infinite-horizon average-cost problem by solving a first-exit problem up to some set of terminal states $\mathcal{T}$, and at the terminal states applying a final cost $h$ equal to the differential cost-to-go $\widetilde{v}$ for the infinite-horizon problem. Of course if we knew $\widetilde{v}$ the original problem would already be solved and we would gain nothing from the first-exit reformulation. However, if we only have an approximation to $\widetilde{v}$, choosing greedy actions with respect to $\widetilde{v}$ is likely to be worse than solving the above first-exit problem. This is the spirit of MPC. There is no proof that such a procedure will improve the control law, but in practice it usually does.

There is an important difference between our proposal and the way MPC has been used in the past. Traditionally MPC solves (in real time) a finite-horizon problem rather than a first-exit problem. That is, at each time step it computes an optimal trajectory extending $N$ steps into the future, where $N$ is predefined. The final state of such a trajectory can be any state; therefore the final cost $h$ needs to be defined everywhere. In contrast, our method always computes a trajectory terminating at a state in $\mathcal{T}$, and so our final cost only needs to be defined for $x \in \mathcal{T}$. This is advantageous for two reasons: **i.** guessing/approximating $\widetilde{v}$ is easier if we have to do it at only a subset of all states; **ii.** in the case of contact dynamics, if we define $\mathcal{T}$ as the set of states where contacts occur, then the real-time optimization does not need to deal with contact discontinuities; instead the effects of such discontinuities are incorporated in $h$.

[1]The more traditional total cost-to-go function $v$ is infinite in non-discounted problems, which is why we work with $\widetilde{v}$. Loosely speaking, if $v(x,t)$ is the total cost-to-go at time $t$ for a problem that starts at $t = -\infty$ and ends at $t = 0$, the relation to the differential cost-to-go is $v(x,t) \approx \widetilde{v}(x) - ct$.

### B. Tuning the final cost

One way to specify $h$ is to use domain-specific heuristics. We will see below that in the case of ball-bouncing, our formulation of MPC makes it particularly easy to come up with obvious heuristics – which basically define what is a good way to hit a ball. In other tasks such as walking, the heuristics may define what is a good way to place a foot on the ground.

Another approach is policy gradient that requires a parametric function approximator $h(x;w)$ where $w$ is a real-valued vector. The vector $w$ defines a function $h$, which in turn defines an MPC control law (through some real-time optimization algorithm), which in turn can be evaluated empirically (through sampling) on the original infinite-horizon problem. In this way we can define the average empirical cost $c(w)$ for every possible $w$, and perform gradient descent on it.

This paper presents the implementation of the heuristic approach (which worked surprisingly well for ball-bouncing); the policy gradient based improvement has been tested and discussed in Kulchenko and Todorov [12].

### C. Solving the first-exit problem

Each iteration of iLQG starts with an open-loop control sequence $\overline{\mathbf{u}}^{(i)}(t)$, and a corresponding state trajectory $\overline{\mathbf{x}}^{(i)}(t)$ and includes the following steps (described in more detail in [5]):

1) Build a local LQG approximation around $\overline{\mathbf{x}}^{(i)}, \overline{\mathbf{u}}^{(i)}$.
2) Design a control law for the linearized system.
3) Apply this control law forward in time to the linearized system.
4) Compute the new open-loop controls.
5) Simulate the system to compute new trajectory and cost. For each time step, check if the terminal event condition is satisfied and adjust the number of time steps and the final cost accordingly.
6) End the iteration if the costs for the previous and the new open-loop control sequences are sufficiently close.

The original iLQG algorithm has been modified to handle terminal events in the following way: in the forward pass instead of running for a predefined number of time steps, the algorithm runs until it hits a terminal state. If the terminal state is not hit, then the algorithm behaves exactly like the

original one. Unlike the original algorithm, as the number of steps can now vary between iterations (depending on when exactly the terminal state has been reached), special care needs to be taken not to abort subsequent iterations prematurely. To achieve this, if during the forward pass a terminal state is not hit in the number of steps used in the last backward pass, the sequence of states is extended (and the sequence of controls extended too using values from the initial sequence if necessary) until a terminal state or the limit on the number of states is reached.

## III. Application to ball-bouncing

We have applied this method to the ball juggling system that includes a paddle mounted on a robot and a table tennis ball. The robot moves an effector with the paddle in three dimensions inside a workspace defined by a cylinder (with a center of the cylinder positioned at $[0\ 0\ 0]^T$); the paddle always stays in the horizontal plane. The robot receives a control signal, which is limited by the force that the robot can generate. Let $p_x$, $p_y$, and $p_z$ be positions of the paddle in their respective coordinates; $b_x$, $b_y$, and $b_z$ be positions of one ball and $o_x$, $o_y$, and $o_z$ be positions of the other ball. The state has 18 dimensions: $\mathbf{x} = \left[p_x\ p_y\ p_z\ \dot{p_x}\ \dot{p_y}\ \dot{p_z}\ b_x\ b_y\ b_z\ \dot{b_x}\ \dot{b_y}\ \dot{b_z}\ o_x\ o_y\ o_z\ \dot{o_x}\ \dot{o_y}\ \dot{o_z}\right]^T$ The dynamics are

$$\begin{aligned}
\dot{\mathbf{x}}_{pp} &= \mathbf{x}_{pv} \\
\dot{\mathbf{x}}_{pv} &= \mathbf{u} + [0\ 0\ \text{-}g]^T \\
\dot{\mathbf{x}}_{bp} &= \mathbf{x}_{bv} \\
\dot{\mathbf{x}}_{bv} &= [0\ 0\ \text{-}g]^T - d\|\mathbf{x}_{bv}\|\mathbf{x}_{bv} \\
\dot{\mathbf{x}}_{op} &= \mathbf{x}_{ov} \\
\dot{\mathbf{x}}_{ov} &= [0\ 0\ \text{-}g]^T - d\|\mathbf{x}_{ov}\|\mathbf{x}_{ov}
\end{aligned}$$

where the state variables are $\mathbf{x}_{pp} = [p_x\ p_y\ p_z]^T$, $\mathbf{x}_{pv} = [\dot{p}_x\ \dot{p}_y\ \dot{p}_z]^T$, $\mathbf{x}_{bp} = [b_x\ b_y\ b_z]^T$, $\mathbf{x}_{bv} = \left[\dot{b}_x\ \dot{b}_y\ \dot{b}_z\right]^T$, $\mathbf{x}_{op} = [o_x\ o_y\ o_z]^T$, $\mathbf{x}_{ov} = [\dot{o}_x\ \dot{o}_y\ \dot{o}_z]^T$, $g$ is the acceleration due to gravity, and $d$ is the coefficient calculated based on the drag coefficient and other parameters (see Section Parameter identification for details on how it is calculated). The goal is to juggle the ball given its desired velocity after the contact while keeping the paddle in the workspace (close to the center of the workspace). The control objective is to find the control $\mathbf{u}(t)$ that minimizes the performance index

$$J_0 = h(x(T)) + \sum_{t=1}^{T-1} \ell(x(t), \mathbf{u}(t)) \tag{3}$$

$$\begin{aligned}
\ell(x, \mathbf{u}) = \|\mathbf{u}\|^2 + w_w\|\mathbf{p}_{xy}\|^2 + w_z(p_z - p_{target})^2 \\
+ w_p\|\mathbf{p}_{xy} - \mathbf{b}_{xy}\|^2
\end{aligned} \tag{4}$$

$$\begin{aligned}
h(x) = w_v\|\mathbf{x}_{bv}^{contact} - \mathbf{v}_{target}\|^2 + w_d H\left(\dot{b}_z\right) \\
+ w_p\|\mathbf{p}_{xy} - \mathbf{b}_{xy}\|^2
\end{aligned} \tag{5}$$

where $\mathbf{x}_{bv}^{contact}$ is the ball velocity after the contact, $\mathbf{v}_{target}$ is the target ball velocity after the contact (calculated as described in Sections Target identification for one-ball system and Target identification for two-ball system), $p_{target}$ is the target z coordinate for the paddle, $H$ is a unit step function, $w_v$ is a weight on the velocity error, $w_d$ is the weight on the direction of the velocity at the contact (to penalize hitting ascending rather than descending ball), $w_w$ is the weight on the distance from the middle of the workspace in the xy coordinates, $w_z$ is the weight on the distance from $p_{target}$, and $w_p$ is the weight on the distance from the ball projection on the xy-plane. We used a time step of $10msec$, $T = 1sec$, and set the maximum control torque along each coordinate to be $|u| \leq 50$. As stated before, the final time $T$ is adjusted based on the result of the check for a terminal event[2].

### A. Target identification for one-ball system

To send the incoming ball to its target position and its target height we have to specify the desired return velocity. For one ball system the target position/height are set to specific values[3] and the return velocity is calculated as velocity needed to reach the desired height based on ball's position at contact using the system dynamics as described in the previous section. For the velocity in xy-plane the system calculates the time for the ball to fly up to the target height and then down to the target position and then uses the distance between the ball at the contact and the target position to get the velocity. The drag is not taken into account in the last operation as velocities in xy-plane are small comparing to z-velocity.

### B. Target identification for two-ball system

As noted by Schaal and Atkeson [6], "In order to juggle more than one ball, the balls must travel on distinct trajectories, and they should travel for a rather long time to facilitate the coordination of the other balls." To achieve this in our setup with the hand-crafted cost function, the function that calculates the return velocity has been modified in the following way. In the one-ball configuration the return velocity is calculated based on the desired height and the target position (set to the center of the workspace), whereas in the two-ball configuration the target position is calculated based on where the other ball is expected to intersect $z = 0$ plane. The target position is then set to be on the line that connect the intersection point with the center of the workspace according to the formula: $\mathbf{p}_t = \mathbf{p}_2 - d_{target}\mathbf{p}_2/\|\mathbf{p}_2\|$, where $\mathbf{p}_2$ is the position of the other ball, $\mathbf{d}_{target}$ is a desired distance between the balls, and $\mathbf{p}_t$ is the target position for the ball at the contact. The target height is calculated in such a way as to have one ball at the apex when the other ball is at contact. After the targets are calculated, the same function

[2]The terminal event for this case is defined as the ball hitting the paddle: the z coordinate of the ball being at or below the position of the paddle with the thickness of the paddle and the radius of the ball taken into account.

[3]Both targets can be modified at any time and the system will adapt to the changes; the video submitted with the paper includes a segment where the target height is randomly set to a value between $0.10m$ and $0.60m$ after each contact and the system incorporates that value into the calculations to hit the target.

that calculates the desired return velocity for one ball system is applied.

The advantage of this approach is its simplicity, but one of its drawbacks is that the actual trajectory of the ball is not taken into account. As can be seen from the video submitted with the paper, the balls occasionally collide. One improvement that we are considering to implement is to analyze the trajectories and adjust the target if the projected ball trajectories intersect or come too close.

Notice that all the other parameters of the cost function stayed exactly the same, which allowed for seamless integration of one-ball and two-ball juggling behaviors as demonstrated in the video. This permitted for one of the balls to be removed from or added to the workspace at any time (which also simplified the system setup).

### C. System design

In this section we describe the design of the system we used to run the experiments on. For the robot platform we used the Delta Haptic robot [13] with three degrees of freedom that is capable of high speed motion.

The robot has a regular table tennis paddle mounted on its effector and interacts with regular table tennis balls; both the paddle and the balls follow the ITTF rules[4]. The balls are tracked by high-speed Vicon Bonita cameras[5] with frequency 240Hz and are covered with a reflective tape[6] to enable tracking. The tape has been carefully applied to avoid bulges and gaps to minimize the noise during contact with the paddle. As Reist and D'Andrea [14] observed, "the main source of noise in the measured impact times and locations are stochastic deviations of the impact parameters, i.e. ball roundness and CR"; they also specifically called out table tennis balls as generating too much noise in the horizontal degrees of freedom at contact. In our case both of the aspects–ball roundness and the coefficient of restitution–have been affected by tape application, which introduced more noise for the system to deal with.

Another aspect of the system that posed additional challenges was the limited size of the robot workspace: $0.36m$ in diameter and $0.30m$ in height. Although ITTF rules do not specify the size of the paddle, the majority of them are about $15cm$ across, which is approximately the half of the workspace diameter.

We have not made any modifications to the robot other than attaching a paddle to its effector, but we have made two configuration changes: disabled gravity compensation and raised the velocity threshold to allow two-ball juggling. The robot is capable of applying force of up to $25N$, which, given the mass of the effector with the paddle, roughly translates to $110m/s^2$; however, in all our experiments the acceleration has been limited to $50m/s^2$.

As the ball position is reported in the Vicon coordinates and the paddle position is reported in the robot coordinates, the two coordinate systems need to be merged together to allow for proper calculations of ball positions and desired return velocity. To avoid adding markers to the paddle or limiting the robot to a particular position, we added a simple calibration step to each experimental session. At the beginning of the session the ball is placed on the paddle and the paddle is moved to four different (random) positions. The coordinates reported by the cameras and the robot are recorded and then the conversion matrices for the two coordinate systems are generated based on four sets of coordinates. These conversion matrices are used to map coordinates provided by the cameras to the robot (workspace) coordinates in which all the calculations are done.

### D. Implementation details

All the code for the control loop and the iLQG algorithm has been implemented in MATLAB. The initial implementation of the algorithm was taking up to $100ms$ to get a solution (in simulation) and was too slow to be used inside a $10ms$ control loop. Rewriting some of the components to use vectorization instead of loops somewhat improved performance, but still not sufficiently. After exploring several options we decided to use Embedded MATLAB coder that works with Real-Time Workshop for MATLAB[7] and provides an option of translating MATLAB code to C and then compiling it to native code. To achieve the desired performance we modified the original iLQG algorithm[8] to make it compatible with Embedded MATLAB and could generate a solution in $3ms$ (on average in simulation with one ball), which was sufficiently fast for our purposes. The original algorithm had also been modified to implement terminal events, as described in Section Solving the first-exit problem.

## IV. System identification and modeling

The system has several parameters that affect the model and the cost function used in calculating the optimal solution. Two coefficients used in the model –the coefficient of restitution and the drag–have major impact on the performance of the system and have been estimated from the experimental data. Other parameters–like weights for components of the cost function–have been selected based on results of running one- and two-ball juggling algorithms in simulation.

### A. Parameter identification

The coefficient of restitution ($e_z$) was estimated from the data recorded on the robot based on paddle and ball velocities before and after impact according to the following formula: $e_z = (\dot{b}_z^{after} - \dot{p}_z^{after})/(\dot{p}_z^{before} - \dot{b}_z^{before})$. The results lay in the interval $e_z \in [0.562, 0.700]$. The large range of estimated values is likely to be explained by the impact of the tape applied to the balls. We used $e_z = 0.58$ for most of our experiments as this value produced the least deviation from the target height for the different values of target height tested. The coefficients of restitution for x and y impact

[4]International Table Tennis Federation; http://www.ittf.com/

[5]http://www.vicon.com/products/bonita.html

[6]3M™ Scotchlite™ Reflective Tape 03456C

[7]http://www.mathworks.com/products/rtw/

[8]As described in [5] and based on MATLAB implementation available at http://www.cs.washington.edu/homes/todorov/

directions proved to be difficult to estimate consistently; we hypothesized that this was caused by the effect of the spin, that was difficult to measure and was ignored in the model. These two coefficients of restitution ($e_x$ and $e_y$) have been set to 1 in the experiments.

The second parameter estimated from the data was the $d$ parameter, calculated according to the following formula: $d = c_d \rho A/(2m)$, where $c_d$ is the coefficient of drag for a smooth sphere (0.5), $\rho$ is mass density of the air ($1.229 kg/m^3$ at sea level), A is the reference area (0.0012566 for $40mm$ table tennis ball) and $m$ is the ball mass ($0.0027kg$). As both the ball mass and, more importantly, the coefficient of drag of the ball have been affected by the application of reflective tape, this parameter was estimated from the data based on the best fit with the model that minimizes sum of squared errors. The value that we used in the experiments was 0.4, whereas the value calculated for a regular table tennis ball would be 0.143.

### B. MATLAB simulation

In addition to implementing the core algorithm in MATLAB, we also implemented the simulator that runs the algorithm and also collects and presents the data for analysis (including 3d representation of the paddle and the balls). This simulation was used to estimate performance characteristics under various conditions and optimize the code and also to select weights for the components of the cost function. The next section provides comparison for some of the results in simulation with results on the real robot.

## V. Experimental results

In this section we present and discuss experimental results from the MATLAB simulator and the actual robot system. The focus of these experiments has been on the performance profile of the system as well as on suggesting ways to improving accuracy in controlling the behavior of the system.

### A. Execution time analysis

The simulation environment that we developed supported two simulation modes: one that runs completely in simulation and the other one that gets data from the robot, applies the calculations, and sends control signals to the robot. The only difference with running this on the robot is that the cameras are not being queried and the algorithm is using ball positions as provided by the simulation engine. We completed several runs in the simulation environment and on the real system in one- and two-ball setups and compared results for execution times of the algorithm in each of the conditions[9]. The results are presented in Fig. 1.

As can be seen from the diagrams, the mean execution time on the robot is larger than in simulation; however the mean time for two-ball setup is less than for one-ball setup in simulation. This can be explained by the fact that because of the more frequent contact events, the number of time-steps

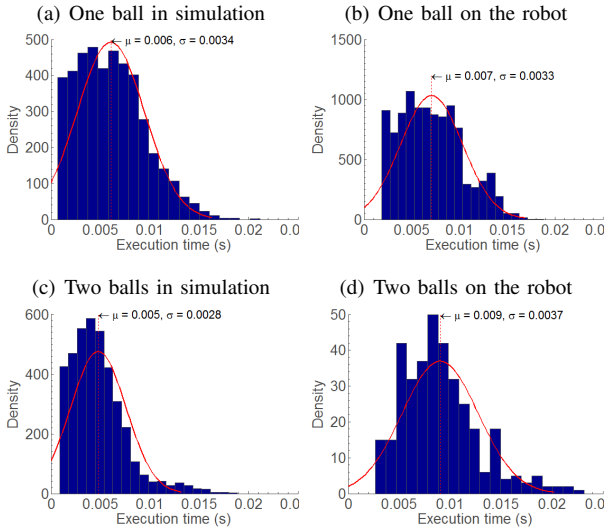


Fig. 1. Execution time histograms and fit Gaussians for one and two ball juggling in simulation and on the real robot.

is approximately twice less, and as the result, there are fewer calculations to be done by the iLQG algorithm.

The diagrams also show that while most of the calculations in all four conditions are done faster than $10ms$, which is the desired duration of the control loop, there is still a large portion of them that take longer than $10ms$. It turned out that the steps that take most time have properties that can be exploited to further reduce execution time without significant impact on the quality of generated solutions. As can be seen in Fig. 2, steps with the longest execution time are the ones that are calculated shortly after the contact between the paddle and the ball[10].

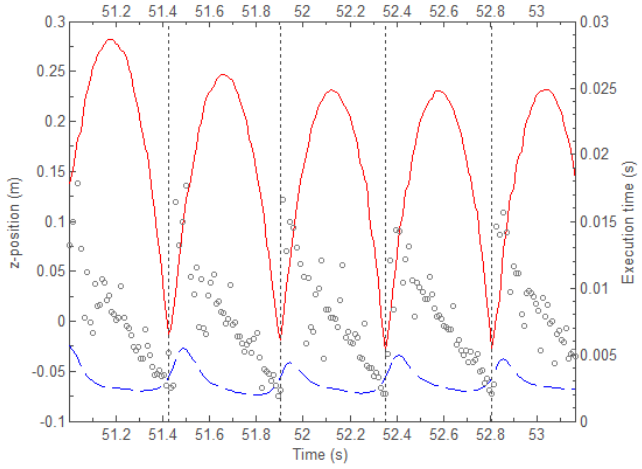


Fig. 2. Positions of paddle (blue, dashed) and ball (red) during several bounces with execution times (gray) of the control loop for each time step.

One way to shorten the execution time is to increase the

[9]All the calculations and experiments were performed on a desktop computer with Intel®Core™2 Duo Processor E8500 3GHz and 4GB RAM with the robot and Vicon cameras connected to the same computer.

[10]The ball and the paddle trajectories do not touch on the diagrams because the paddle position is the position of the effector it is mounted on. There is expected to be approximately $0.0328m$ between the center of the ball ($0.040m$ in diameter) and the effector at contact.

time step for these initial steps, which should improve performance and not adversely affect generated solutions. Another way is to abort the iLQG algorithm if the threshold on the execution time is exceeded even before the convergence is reached. While this may return a sub-optimal solution, this is less critical during the first (after contact) time steps; the solution will be improved during the subsequent steps. As the robot is capable of generating fast movements with relatively large force ($24N$), we have implemented a fail-safe mechanism that ignores the control signal if the cost of the generated solution is too large and sends a signal to the robot that only compensates for gravity, which helps to avoid (potential) damage to the robot and the environment.

### B. Response to perturbations

The investigation of how the system reacts to external perturbations has been one of the most interesting aspects of our research. After the system could robustly bounce one ball[11] we subjected it to various disturbances, which included removing the ball and then throwing it back, holding the paddle or limiting its movement, and pushing or pulling the paddle in various directions. The two graphs in Fig. 3 show z- and x-positions of the paddle and the ball during some of the perturbations.

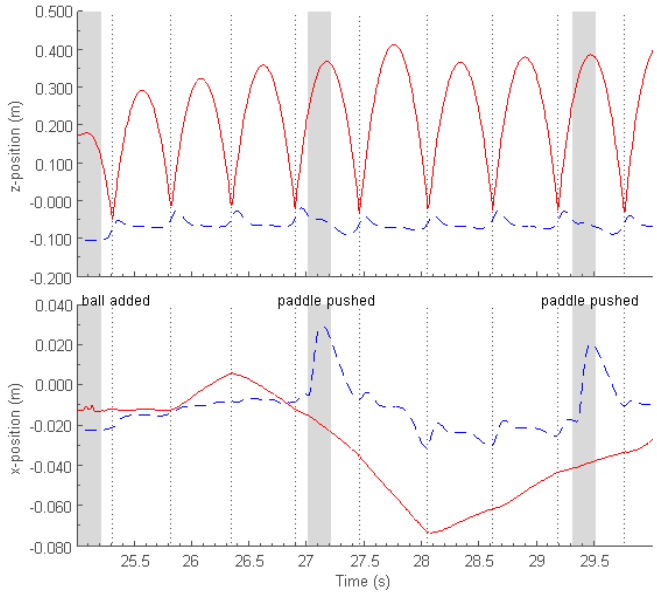


Fig. 3. System response to perturbations. The perturbations are marked with gray lines; the contacts are marked with dotted lines. Top: z-position of the paddle (blue, dashed) and the ball (red). Bottom: x-position of the paddle (blue, dashed) and the ball (red).

The first gray line marks the time when the ball was added (25th second on the graph; this corresponds to 42nd second on the submitted video). The robot juggled the ball for a bit and then the paddle was pushed (27th second on the graph marked with a gray line and 44th second on the video); the system recovered from the push and completed the bounce. Then the paddle was pushed again (29th second on the graph marked with a gray line and 46th second on the video), but the paddle smoothly came back and the system continued juggling. The system demonstrated robust behavior and quick recovery from disturbances and had been fun to play with.

In addition to perturbing the system during one-ball juggling, we also analyzed in detail how the system reacts to changes during two-ball juggling. Fig. 4 shows several events of interest also marked with gray lines. The first event was adding a second ball to the system, which was then bounced along with the first one. Then the system lost one of the balls (around 57th second; marked with a gray line), which happened because two balls were too close to each other[12]. In this case the system hit the ball too slow and the next contact happened with two balls almost at the same time. The system still managed to recover and sent one of the balls high to get the next contact to happen when the ball is close to the apex. The system continued to happily bounce the balls for one more second until one of the balls flew out of the workspace. The cameras lost one of the balls one more time (third gray line), but this didn't have any negative consequences as both of the balls were flying up. As the result of this analysis we implemented a mechanism that checks for this set of conditions and adjusts reported positions when one of the balls is lost, which eliminated this issue from subsequent runs.

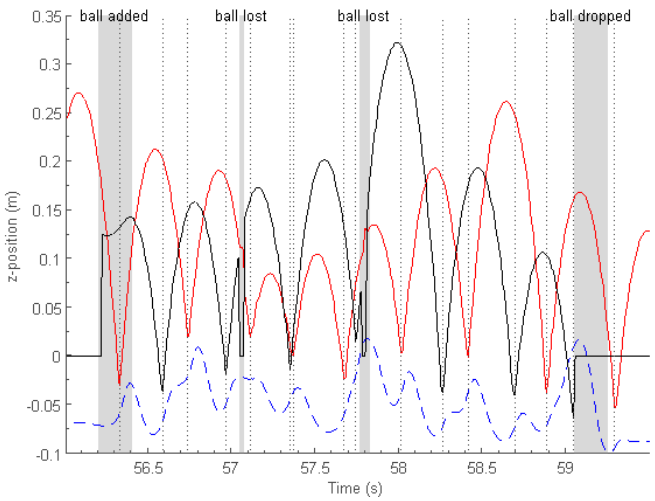


Fig. 4. Two ball bouncing sequence with paddle (blue, dashed) and balls (red and black) z-positions. Events of interest marked with gray background.

### C. Accuracy of reaching target height

To assess the accuracy of reaching the target height we recorded several two-minute sequences of bounces with various target heights; the results are shown in Fig. 5. Even

[11] We recorded several 12 minute runs with 1500+ bounces in each session without any interruptions.

[12] Although the Vicon system proved to be reliable in ball tracking, when two balls get sufficiently close (less than $3/4$ of their diameter), the system tends to recognize and report them as one marker; while the balls pass each other they may still be reported as one marker. This has a double negative effect as not only does one of the balls "disappear" from the system, but the other ball is reported in the wrong position (roughly between the two balls), which affects both position and velocity calculations.

though our analysis of the coefficient of restitution produced different values for different target heights (as described in Section Parameter identification), we used the same value for all these experiments and the system generated good results with rather narrow distributions (except with the target height set to $0.40m$, where the mean is $0.376m$).

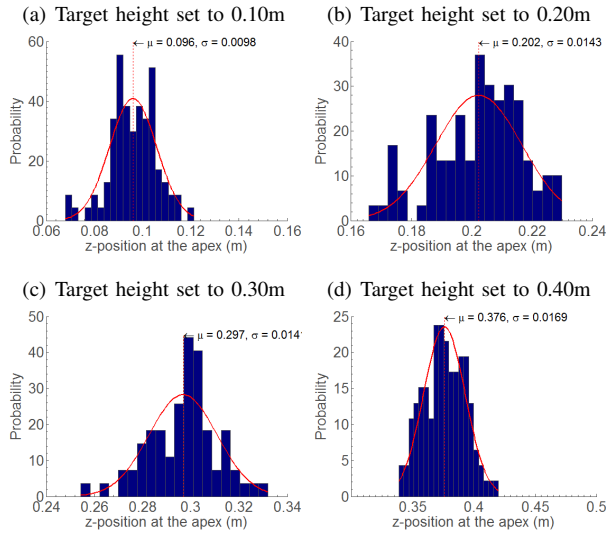


Fig. 5. Accuracy of the ball reaching a target height for various target values.

The accuracy of the target height is an important measurement, especially for bouncing two balls, where the time between bounces is one of the few parameters that can be controlled. This accuracy depends on several factors: the specified coefficient of restitution, paddle and ball quality, the amount of spin (not included in the model), control of the robot to get to the contact point with the right velocity, and other factors. As the desired paddle velocity at contact, which in turn defines the ball velocity after the contact and as such the height it will fly to, is only one of the components in the cost function, the system makes a decision based on a combination of those components and may not necessarily satisfy each of its targets individually (being it a target position or a target height).

### *D. Accuracy of predicting paddle position*

We also analyzed how well the paddle follows the predicted trajectory, which would indicate how well the system can control the robot. Fig. 6 shows the actual trajectory of the paddle during several bounces and the predictions of the algorithm for positions of the paddle in the subsequent time steps before a contact (small black dots mark positions and time steps when the prediction is being made).

There are two things that can be seen from the graph: the robot doesn't exactly follows the trajectory that the algorithms predicts for it, which can be explained by delays in the robot hardware and the fact that sending of the control commands is delayed by the time of the algorithm execution. More specifically, if the vicon/robot data are gathered at time $T$ and the prediction is based on that data, the control signal

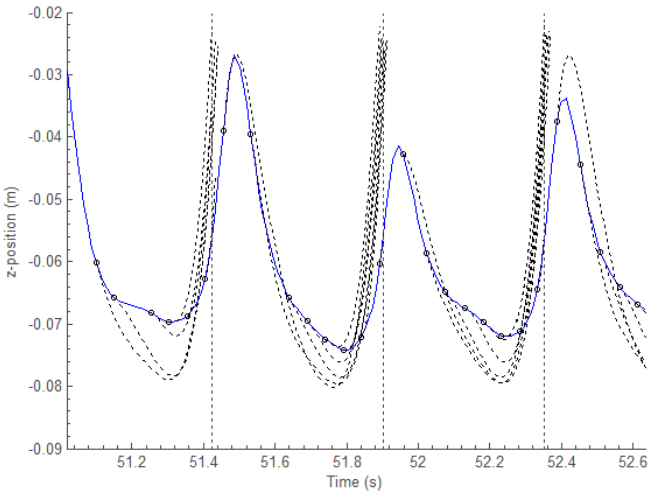


Fig. 6. Actual (solid blue) versus predicted (dashed black) paddle position.

is sent at time $T + \delta$, where $\delta$ is the execution time of the algorithm; it is $7ms$ on average for the system with one ball. Other potential causes for the differences are friction, which is currently not being modeled; some nonlinearities in the robot dynamics which are currently modeled as linear; and mis-estimation of the effective mass.

One thing worth noting in this context is that the modeling errors are significant and yet the control algorithm works remarkably well. In fact it worked so well that we didn't even realize the errors were there until we analyzed and plotted the data. This is a very good property to have because complex robots can rarely be modeled well, so it is important to have control algorithms that are robust to model errors; these errors are different from perturbations (and our algorithm is robust to both). Having said that, one of the directions where we plan to take this work is to get better control of the robot as further detailed in the Conclusions section.

### *E. Accuracy of contact point on paddle*

Another important parameter, especially for the two ball bouncing, is the accuracy of the contact point on the paddle as the two balls need to be separated in both time and space. Fig. 7 shows contact coordinates relative to the paddle as well as their fitted distributions in x and y coordinates. The results show that the contact points have the expected mean in the x coordinate and a small non-zero mean in the y coordinate; we attributed this deviation to the robot platform being slightly misleveled.

## VI. Conclusions

In this work we generalized MPC from the finite horizon setting to a first-exit setting, generalized and extended iLQG algorithm to first-exit setting, described suitable cost functions and overall system design, implemented the system, and analyzed its performance in simulation and on the real robot. We also demonstrated that even though the system is using a robot with significant physical constraints (a small workspace and no tilt control) we were able to robustly bounce one ball,

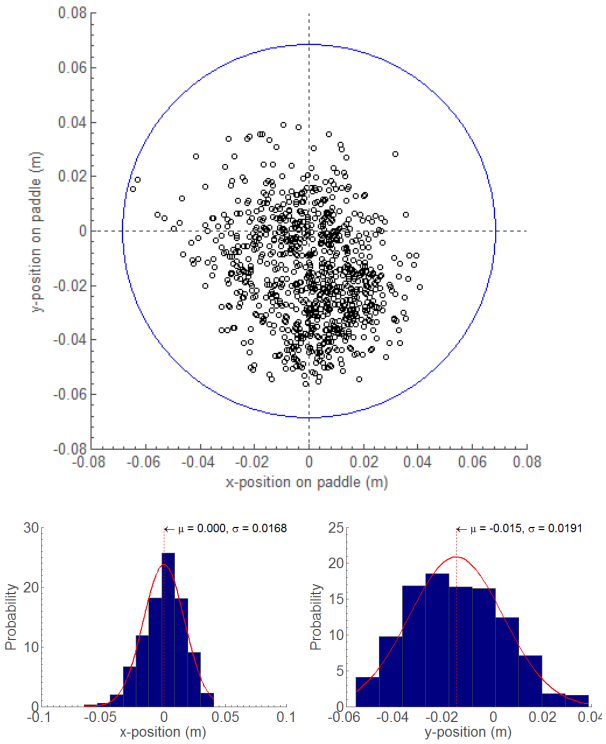


Fig. 7. Contact locations between ball and paddle relative to the center of the paddle. Top: x- and y-positions on the paddle; blue circle approximately shows the paddle size (0.15m in diameter). Bottom: Fitted contact coordinate distributions for x- and y-positions. Contact positions are calculated as the difference between positions of the ball and the center of the paddle. The effect of the discretization (the actual point of contact being somewhere between the measurements) is ignored in this case; we used coordinates from the lowest (in z) measurement.

recover from a wide range of disturbances, and bounce two balls on the same paddle.

We are pursuing extending the work that has been done so far in several directions: **i.** tuning the final cost of the solution using policy gradient and approximate policy iteration as described in more detail in Section Tuning the final cost; **ii.** improving properties of the existing algorithms by speeding up calculations, taking ball trajectories into account when calculating desired target positions, and improving the model by incorporating robot delays and other factors like the horizontal coefficient of restitution, that are currently being ignored.

To minimize modeling errors we plan to develop a more detailed dynamics model of this robot and infer its parameters via system identification, which is likely to increase performance further. Another item for future work is developing a state estimation algorithm for tracking the balls, instead of relying on the instantaneous Vicon data (which proved to be unreliable when the balls come close). This has been partially addressed with the fail-safe mechanism as described in Section Execution time analysis, but we are looking for a more systematic solution.

We also plan to do a comparison with human subjects to explore how the behavior of the robot is different from humans on similar tasks and to bring the cost model closer to the models that humans may be using. We have already performed an experiment with human subjects bouncing ping-pong balls to different heights and with different amounts of perturbation (introduced by making the ball bumpy). This is actually one of the first experimental datasets on mechanically-unconstrained 3D human ball bouncing, and is made possible by our trick of wrapping the ball in reflective tape and treating it as a Vicon marker. We are looking forward to analyzing this data in detail and comparing it to the behavior of the robot. Such comparisons will hopefully suggest additional improvements in our control algorithm.

## VII. Acknowledgements

This work was supported by the US National Science Foundation. Thanks to Yuval Tassa and Alex Simpkins for their help with the project.

## References

[1] M. Morari and J. Lee (1999). Model predictive control: past, present and future. *Computers and Chemical Engineering* 23: 667-682

[2] D. Bertsekas (2005). Dynamic programming and suboptimal control: A survey from ADP to MPC. *European J Control*

[3] M. Diehl, H. Ferreau and N. Haverbeke (2008). Efficient numerical methods for nonlinear MPC and moving horizon estimation. *International Workshop on Assessment and Future Directions of NMPC*

[4] P. Abbeel, A. Coates, M. Quigley and A. Ng (2007) An application of reinforcement learning to aerobatic helicopter flight. In *Advances in Neural Information Processing Systems* 19

[5] E. Todorov and W. Li (2005). A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems. In proceedings of the *American Control Conference 2005*

[6] S. Schaal and C. G. Atkeson (1993). Open Loop Stable Control Strategies for Robot Juggling. In proceedings of the *1993 IEEE international conference on Robotics and Automation*

[7] A. Rizzi and D. Koditschek (1994). Further progress in robot juggling: Solvable mirror laws. In proceedings of the *1994 IEEE international conference on Robotics and Automation*

[8] D. Sternad, M. Duarte, H. Katsumata and S. Schaal (2000). Dynamics of bouncing ball in human performance. *Physical Review E*, vol 63

[9] R. Ronsse, K. Wei and D. Sternad (2010). Optimal control of a hybrid rhythmic-discrete task: The bouncing ball revisited. *Journal of Neurophysiology* 104: 2484-2493

[10] K. Lynch and C. Black (2001). Recurrence, Controllability, and Stabilization of Juggling, *2001 IEEE Trans. on Robotics and Automation*, Vol.17, No.2:113-124.

[11] E. Todorov (2004). Optimality principles in sensorimotor control. *Nature Neuroscience* 7: 907-915

[12] P. Kulchenko and E. Todorov (2011). Policy gradient methods with model predictive control applied to ball bouncing. To appear in *IEEE Adaptive Dynamic Programming and Reinforcement Learning*

[13] S. Grange, F. Conti, P. Helmer, P. Rouiller, C. Baur (2001). The Delta Haptic Device. In proceedings of the *Eurohaptics'01*

[14] P. Reist and R. D'Andrea (2009). Bouncing an Unconstrained Ball in Three Dimensions with a Blind Juggling Robot. In proceedings of the *2009 IEEE international conference on Robotics and Automation*