Agile Catching with Whole-Body MPC and Blackbox Policy Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract: We address a benchmark task in agile robotics: catching objects thrown at high-speed. This is a challenging task that involves tracking, intercepting, and cradling a thrown object with access only to visual observations of the object and the proprioceptive state of the robot, all within a fraction of a second. We present the relative merits of two fundamentally different solution strategies: (i) Model Predictive Control using accelerated constrained trajectory optimization, and (ii) Reinforcement Learning using zeroth-order optimization. We provide insights into various performance trade-offs including sample efficiency, sim-to-real transfer, robustness to distribution shifts, and whole-body multimodality via extensive on-hardware experiments. We conclude with proposals on fusing "classical" and "learning-based" techniques for agile robot control.

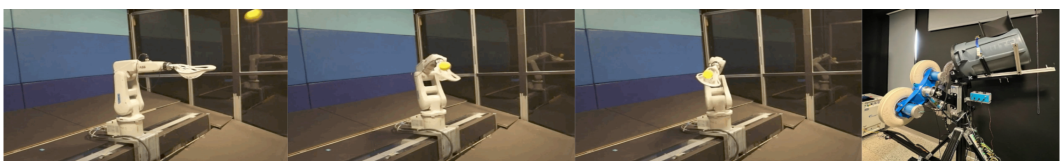

Figure 1: Mobile Manipulator with Lacrosse Head catching a ball within a second. (right) Automatic ball thrower with controllable yaw angles and speed of around 5m/s.

1 Introduction

Chasing a ball in flight and completing a dramatic diving catch is a memorable moment of athleticism - a benchmark of human agility - in several popular sports. In this paper, we consider the task of tracking, intercepting and catching balls moving at high speeds on a mobile manipulator platform (see Figure 1), whose end-effector is equipped with a Lacrosse head. Within a fraction of a second, the robot must start continuously translating visual observations of the ball into feasible whole body motions, controlling both the base and the arm in a coordinated fashion. In the final milliseconds, the control system must be robust to perceptual occlusions while also executing a cradling maneuver to stabilize the catch and prevent bounce-out. The physics of this task can be surprisingly complex: despite its geometric simplicity, a ball in flight can swing and curve in unpredictable ways due to drag and Magnus effects [1]; furthermore, the contact interaction between the ball and the deformable end-effector involves complex soft-body physics which is challenging to model accurately.

In this paper, we study the relative merits of synthesizing high speed visual feedback controllers for this task from two ends of a design spectrum: Model Predictive Control (MPC) [2, 3] representing a "pure control" strategy, and Blackbox policy optimization [4] representing a "pure learning" approach. MPC optimizes robot trajectories in real time in response to state uncertainty - it is nearly "zero-shot" in terms of data requirements and gracefully handles kinematics, dynamics and task-specific constraints, but can be computationally expensive and sensitive to errors in dynamics modeling. On the other hand, policy learning via blackbox or RL (Reinforcement Learning) methods can be extremely data inefficient, but can adapt, in principle, to complex and unknown real world dynamics. Our primary contribution is to provide insights into subtle trade-offs in reaction time, sample efficiency, robustness to distribution shift, and versatility in terms of whole-body multimodal behaviors in a unified experimental evaluation of robot agility. We conclude the paper with proposals to combine the "best of both worlds" in future work.

Submitted to the 6th Conference on Robot Learning (CoRL 2022). Do not distribute.

Related Work: Both classes of techniques have been previously applied to the robotic catching task. Examples of optimization-based control for ball catching include [5, 6, 7, 8, 9]; [10] and [11] present an unified approach subsuming catch point selection, catch configuration computation and path generation in a single, nonlinear optimization problem (also see, [12], [13]). Several papers utilize human demonstration and machine learning for parts of the control stack. [14] probabilistically predict various feasible catching configurations and develop controllers to guide hand–arm motion, which is learned from human demonstration. [15] also learn motion primitives from human demonstration and generate new movements. [16] use bi-level motion planning plus a learning based tracking controller. Some papers aim for soft catching explicitly. [17] extend [14] further, offering a soft catching procedure that is more resilient to imprecisions in controlling the arm and desired time of catch. [18] extend [10] further for enabling soft landing. [8, 19] add heuristics for soft catching, moving the hand along the predicted path of the ball, while decreasing its velocity to allow the dissipation of the impact energy.

2 Problem formulation and proposed solution

We describe the trajectory of the object to be caught by a function F_o, which maps a query time $t \in \mathbb{R}_{\geq 0}$ to the object's position and velocity at time t, i.e., $(p_o(t), v_o(t)) \in \mathbb{R}^{3\times 3}$. Depending on the aerodynamic and inertial properties of the object, F_o may be highly non-trivial. Our knowledge of F_o is encoded via a known $\hat{F}_o$ which maps a query time $t \in \mathbb{R}_{\geq 0}$ and a set of parameters $\theta_o \in \mathbb{R}^d$ to a prediction for the object position and velocity at time t, i.e., $(\hat{p}_o(t;\theta_o), \hat{v}_o(t;\theta_o))$. For this work, we limit our scope to spherical, rigid balls and implement $\hat{F}_o$ via classical Newtonian physics; catching objects with non-trivial aerodynamics and non-uniform shapes is left to future work. However, we only observe the ball position and velocity indirectly via two fixed cameras, and use θ_o to encode our vision system's current position and velocity estimate.

For the robot, we let $q \in \mathbb{R}^7$ denote the joint configuration vector, where $q_1 \in \mathbb{R}$ corresponds to the translational base joint, and $q_{2:7} \in \mathbb{R}^6$ represent the arm joint angles.

2.1 Catching via optimal control

We assume that there exists a lower-level position and/or velocity controller that compensates for the arm's nonlinear manipulator dynamics. Abstracting away the closed-loop behavior of this lower-level control system, we plan for the motion of the arm by assuming second-order integrator dynamics[1] for q, i.e., $\ddot{q}(t) = u_a(t) \in \mathbb{R}^7$.

With this assumption, the optimal catching problem (OCP) can be formalized as a *free-end-time constrained optimal control problem* over the function $u_a(\cdot)$ and catching time t_f:

$$\underset{u_a(\cdot), t_f}{\text{minimize}}\, J(u_a, t_f) := \int_0^{t_f} \left(\lambda + \|u_a(\tau)\|^2\right)\, d\tau + \Psi(q(t_f), \dot{q}(t_f), t_f), \tag{2.1}$$

where $\lambda \in \mathbb{R}_{>0}$ is a weighting constant and $\Psi : \mathbb{R}^7 \times \mathbb{R}^7 \times \mathbb{R}_{\geq 0} \to 0$ is a terminal cost; subject to the second-order integrator dynamics $\ddot{q}(t) = u_a(t)$, and the following constraints:

$$\forall \tau \in [0, t_f],\;\; u_a(\tau) \in [\underline{u}_a, \overline{u}_a],\;\; q(\tau) \in [\underline{q}, \overline{q}],\;\; \dot{q}(\tau) \in [\underline{\dot{q}}, \overline{\dot{q}}], \qquad c(q(t_f), t_f) \geq 0. \tag{2.2}$$

The first three constraints capture limits on the control effort and the joint configurations and velocities. The terminal cost Ψ and endpoint constraint function c capture two desirable properties: (i) $SE(3)$ pose alignment of the lacrosse head with the ball's position and velocity direction at the catching instant, and (ii) minimizing any residual velocity of the lacrosse head perpendicular to the ball's velocity vector. We provide details on these functions within the appendix.

Conversion to Multi-Stage Trajectory Optimization: The OCP is a non-trivial problem which could be solved by leveraging the necessary conditions of optimality for free end-time problems and using boundary-value-problems solvers. However, this would entail optimizing over control, state, and co-state trajectories using dense discretization of the dynamics and inequality constraints (e.g., via collocation). Instead, we simplify the computational burden by optimizing over a restricted class of solutions – a sequence of acceleration and coasting phases, and in the process, convert the problem into a *multi-stage* discrete-time trajectory optimization problem that is subsequently solved

[1] Note that the lower-level control system may have some non-trivial closed-loop response characteristics, including delays. However, these can be pre-compensated for by adjusting the *commanded* $(q, \dot{q})$ setpoints from the planned $(q, \dot{q})$ trajectory.

using a state of the art shooting-based Sequential Quadratic Programming (SQP) solver [20]. We describe this conversion in the appendix.

Asynchronous Implementation: Running concurrently to the catching controller is an estimator that generates updates of the predictor parameters θ_o, necessitating online re-planning. We achieve this via an asynchronous implementation where the optimization problem is continually re-solved in a separate thread, using the latest estimate for θ_o and the current robot state $(q, \dot{q})$. The commanded $(q, \dot{q})$ for the robot's lower-level PD controllers are computed by decoding the most recent stage-wise solution to a continuous-time trajectory, thereby guaranteeing a consistent control rate.

Cradling: Following the intercept of the ball, we use a simple cradling motion primitive, modeled as 2nd-order ODE in q, to slow the lacrosse head and simultaneously rotate the net to point upwards.

2.2 Blackbox Gradient Sensing Optimization

The catching problem can also be formulated as a Partially-Observable Markov Decision Process (POMDP), and solved via Blackbox policy optimization [21, 22, 23]. In this setting, consider a POMDP (S, O, A, R, P) where S is the state space partially observed by O, the observation space, A is the action space, $R : S \times A \mapsto \mathbb{R}$ is the reward function and $P : S \times A \mapsto S$ is the dynamics function. The optimization objective is to learn a parameterized policy $\pi_\theta : O \mapsto A$ that maximizes the expected total episode return, $J(\theta) = \mathbb{E}_{\tau=(\mathbf{s}_0,\mathbf{a}_0,\ldots,\mathbf{s}_T)} \sum_{t=0}^{T} r(\mathbf{s}_t, \pi_\theta(\mathbf{o}_t))$.

Reward function: The reward function is different for training in sim vs. real due to differences in quality of data from each. In both cases we reward the net getting close to the ball during the episode. In sim, we additionally reward orientation alignment before the catch + a stability reward for keeping the ball in the net; in real, we use a flat reward for successful catches (detected by a sensor). Finally, we discourage excessive motion via penalizing position/velocity/acceleration/jerk in sim, and hardware limit violation in real. See Appendix B for more details.

Policy Network: We use a two-tower CNN neural network. The first tower process the historical joint positions represented as an image of size $(n_{\text{hist}}, 7)$, where n_{hist} is the number of past timesteps. The second CNN tower process the predicted ball trajectory represented as an image of size $(n_{\text{pred}}, 6)$, where n_{pred} is the number of predicted timesteps. The output of the two towers is concatenated into a single tensor, which is fed into two fully-connected layers. The final output is then taken as the commanded joint velocities. In total, our policy network has 3255 parameters.

Blackbox Gradient Sensing and Sim-to-Real Finetuning: We apply Blackbox Gradient Sensing (BGS) [24] for optimizing the policy neural network parameters θ. The algorithm optimizes a smoothened version $J_\sigma(\theta)$ of the original total-reward objective $J(\theta)$, given as: $J_\sigma(\theta) = \mathbb{E}_{\delta \sim \mathcal{N}(0,\mathbf{I}_d)}[J(\theta + \sigma\delta)]$, where $\sigma > 0$ controls the precision of the smoothing, and δ is an isotropic random Gaussian vector. We first train in a simulation environment implemented in PyBullet [25]. Once the policy performs well in simulation, we transfer the policy to the real robot and run further BGS finetuning steps using the mechanical thrower.

3 Experiments

We evaluate both our SQP and blackbox (BB) agents in simulation, on the real robot, and also explore performance under various distribution shifts of the thrower. Our SQP agent uses a state of the art SQP solver [20] built on top of trajax [26], a JAX library for differentiable optimal control. For our BB agent, we use a distributed BGS library [4] with policy networks implemented in Tensorflow Keras. The robot used is a combination of an ABB IRB 120T 6-DOF arm mounted on a one-dimensional Festo linear actuator, creating a 7-DOF system. The ball location is determined using a stereo pair of Ximea MQ013CG-ON cameras running with a trained recurrent tracker model.

Error bars: We show catch success for the real robot with error bars which give at least 95% coverage, by using the Clopper–Pearson method to compute binomial confidence intervals.

Inference speed and Reaction time: The BB agent computes a single policy action in time 7.253 ms (std. 0.160 ms), whereas SQP takes 43.046 ms to solve (std. 21.255 ms). Recall that the SQP runs asynchronously, so this solve time does not block the agent; the synchronous part runs in 2.139 ms (std. 0.212 ms). Vision/hardware joint data processing takes about 5 ms. Overall agents are set to synchronously run at 75Hz. The mechanical thrower is 3.9 meters away from the robot and imparts 4.5 m/s horizontal velocity alone; including z-component the speed is ~ 5.5 m/s at catch time.

Simulation to Reality Transfer: Figure 2 highlights the real robot catch performance of both SQP and BB agents. First, we see that while BB performance in sim is mostly monotonically increasing

(Figure 2, left), this does not necessary translate to monotonic improvement on the real hardware (Figure 2, middle). Secondly, we see that SQP suffers less performance degradation compared to BB when transferring to real. Finally, we see that it takes 40 iterations of fine-tuning on real (30 ball throws per iteration) in order for the fine-tuned BB agent to match SQP's real performance (and eventually exceed it). Both methods achieve about 80 to 85% success on mechanical ball throws.

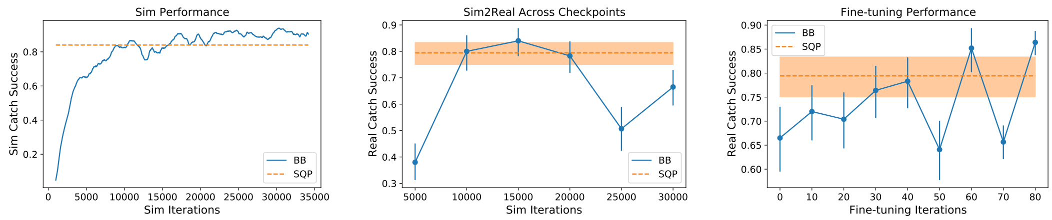

Figure 2: **(Left)** Performance of agents in sim. **(Middle)** Performance of agents on real without fine-tuning. **(Right)** Performance of sim2real transfer after fine-tuning the BB agent starting from the 30k iteration sim checkpoint. Note that each iteration corresponds to 30 mechanical ball throws.

Robustness to Distribution Shifts: Next, we look at the robustness of both agents to out-of-distribution throws. We consider three different distribution shifts: (i) varying the speed of the thrower, (ii) varying the yaw angle of the thrower, and (iii) throwing balls by hand instead of using the mechanical thrower. The first two distribution shifts are plotted in Figure 3. In Figure 3 (left), we see that while BB is reasonably robust to faster throws, its performance significantly degrades for slower throws. This is in contrast to the SQP agent, which moderately degrades in performance for faster throws (most likely due to computational bottlenecks), but is quite robust to slower throws. In Figure 3 (middle), we see that both agents have similar performance across the in-distribution yaw angles, but for out-of-distribution angles SQP maintains its performance better relative to BB.

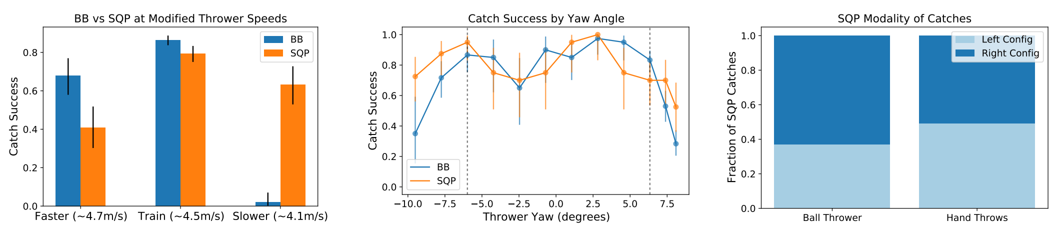

Figure 3: **(Left)** Catch performance as thrower speed varies between faster (~ 4.7 m/s), training (~ 4.5 m/s), and slower (~ 4.1 m/s) throws. **(Middle)** Catch performance as the thrower yaw angle varies from $-9.5°$ to $8°$. Note that the training distribution varies between $-6°$ and $6.3°$ (marked by the dashed vertical black line). **(Right)** Distribution of left and right catches by the SQP agent on both mechanical ball throws and hand throws. Note that the BB agent catches to the right 100% of the time, likely due to the learning bias from the ball throw distribution.

Our last distribution shift involves hand throws (lobs) to the thrower instead of using the mechanical thrower. Using hand throws, the SQP agent has a 68.9% catch success (over 196 throws), whereas the BB agent catch performance degrades to 2.0% (over 150 throws).

Multimodality: In Figure 3 (right), we demonstrate that the SQP agent is able to catch balls in both a left and right pose configuration at fairly even rates matching the bias of the thrower. On the other hand, the BB agent is only able to catch to the right, since the ball thrower distribution is biased (60/40%) towards throwing to the right.

4 Conclusion and future work

While the fine-tuned blackbox agent has the highest catching success performance, the SQP agent is much more robust to distribution shifts in the thrower. To obtain the "best of both", we plan to investigate two different strategies: (i) use BGS to learn the various cost parameters of SQP which we currently tune by hand, and (ii) design a mixed policy which uses SQP to move the end-effector to the ball, and then hands over control to the blackbox agent for final cradling motion. Future extensions include handling multiple non-spherical objects with adaptive dynamics prediction.

References

[1] Aerodynamics of sports balls. *Annual Review of Fluid Mechanics*, 17(1):151–189, 1985.

[2] F. Borrelli, A. Bemporad, and M. Morari. *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.

[3] J. B. Rawlings. Tutorial overview of model predictive control. *IEEE control systems magazine*, 20(3):38–52, 2000.

[4] K. Choromanski, M. Rowland, V. Sindhwani, R. Turner, and A. Weller. Structured evolution with compact architectures for scalable policy optimization. In *International Conference on Machine Learning*, pages 970–978. PMLR, 2018.

[5] H. Yu, D. Guo, H. Yin, A. Chen, K. Xu, Y. Wang, and R. Xiong. Neural motion prediction for in-flight uneven object catching. *ArXiv*, abs/2103.08368, 2021.

[6] U. Frese, B. Bäuml, S. Haidacher, G. Schreiber, I. Schäfer, M. Hähnle, and G. Hirzinger. Off-the-shelf vision for a robotic ball catcher. *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No.01CH37180)*, 3:1623–1629 vol.3, 2001.

[7] J. Kober, M. Glisson, and M. Mistry. Playing catch and juggling with a humanoid robot. *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*, pages 875–881, 2012.

[8] W. Hong and J. Slotine. Experiments in hand-eye coordination using active vision. In *ISER*, 1995.

[9] B. Hove and J. Slotine. Experiments in robotic catching. *1991 American Control Conference*, pages 380–386, 1991.

[10] B. Bäuml, T. Wimböck, and G. Hirzinger. Kinematically optimal catching a flying ball with a hand-arm-system. *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2592–2599, 2010.

[11] R. Lampariello, D. Nguyen-Tuong, C. Castellini, G. Hirzinger, and J. Peters. Trajectory planning for optimal robot catching in real-time. *2011 IEEE International Conference on Robotics and Automation*, pages 3719–3726, 2011.

[12] O. Koç, G. J. Maeda, and J. Peters. Online optimal trajectory generation for robot table tennis. *Robotics Auton. Syst.*, 105:121–137, 2018.

[13] Y. Jia, M. Gardner, and X. Mu. Batting an in-flight object to the target. *The International Journal of Robotics Research*, 38:451 – 485, 2019.

[14] S. Kim, A. Shukla, and A. Billard. Catching objects in flight. *IEEE Transactions on Robotics*, 30:1049–1065, 2014.

[15] M. Riley and C. Atkeson. Robot catching: Towards engaging human-humanoid interaction. *Autonomous Robots*, 12:119–128, 2002.

[16] K. Dong, K. Pereida, F. Shkurti, and A. P. Schoellig. Catch the ball: Accurate high-speed motions for mobile manipulators via inverse dynamics learning. *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6718–6725, 2020.

[17] S. S. M. Salehian, M. Khoramshahi, and A. Billard. A dynamical system approach for softly catching a flying object: Theory and experiment. *IEEE Transactions on Robotics*, 32:462–471, 2016.

[18] B. Bäuml, O. Birbach, T. Wimböck, U. Frese, A. Dietrich, and G. Hirzinger. Catching flying balls with a mobile humanoid: System overview and design considerations. *2011 11th IEEE-RAS International Conference on Humanoid Robots*, pages 513–520, 2011.

[19] V. Lippiello and F. Ruggiero. 3d monocular robotic ball catching with an iterative trajectory estimation refinement. *2012 IEEE International Conference on Robotics and Automation*, pages 3950–3955, 2012.

[20] S. Singh, J.-J. Slotine, and V. Sindhwani. Optimizing trajectories with closed-loop dynamic SQP. In *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022.

[21] K. Choromanski, M. Rowland, V. Sindhwani, R. E. Turner, and A. Weller. Structured Evolution with Compact Architectures for Scalable Policy Optimization. In *Proceedings of the 35th International Conference on Machine Learning*, pages 969–977. PMLR, 2018.

[22] K. Choromanski, A. Pacchiano, J. Parker-Holder, Y. Tang, D. Jain, Y. Yang, A. Iscen, J. Hsu, and V. Sindhwani. Provably robust blackbox optimization for reinforcement learning. In L. P. Kaelbling, D. Kragic, and K. Sugiura, editors, *3rd Annual Conference on Robot Learning, CoRL 2019, Osaka, Japan, October 30 - November 1, 2019, Proceedings*, volume 100 of *Proceedings of Machine Learning Research*, pages 683–696. PMLR, 2019. URL http://proceedings.mlr.press/v100/choromanski20a.html.

[23] H. Mania, A. Guy, and B. Recht. Simple random search provides a competitive approach to reinforcement learning. *NeurIPS*, 2018.

[24] S. Abeyruwan, L. Graesser, D. B. D'Ambrosio, A. Singh, A. Shankar, A. Bewley, D. Jain, K. Choromanski, and P. R. Sanketi. i-sim2real: Reinforcement learning of robotic policies in tight human-robot interaction loops. *arXiv preprint arXiv:2207.06572*, 2022.

[25] E. Coumans and Y. Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2021.

[26] R. Frostig, V. Sindhwani, S. Singh, and S. Tu. trajax: differentiable optimal control on accelerators, 2021. URL http://github.com/google/trajax.

[27] R. Sarkar, D. U. Patil, and I. N. Kar. Characterization of minimum time-fuel optimal control for lti systems. *arXiv preprint arXiv:2102.10831*, 2021.

[28] B. Bäuml, T. Wimböck, and G. Hirzinger. Kinematically optimal catching a flying ball with a hand-arm-system. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2010.

A Catching via Optimal Control

We provide some more details on the formulation of the Optimal Catching Problem. For what follows, let $\texttt{FK} : q \in \mathbb{R}^7 \mapsto \texttt{FK}(q) = (p_h(q), R_h(q)) \in \mathbb{R}^3 \times SO(3)$ denote the forward-kinematics transform that maps the joint configuration vector q to the lacrosse head's $SE(3)$ pose. This transform may be computed in a differentiable manner, e.g., by using a product-of-exponentials method.

Desired Catching Properties. The endpoint catching constraints capture the requirement that the lacrosse head must be positioned and oriented correctly to accept the incoming projectile. In particular, let $(p_o(t_f), v_o(t_f))$ be the true 3D position and velocity of the object at the catching time t_f. Then, we require:

$$\|p_h(q(t_f)) - p_o(t_f)\| \leq \epsilon_p, \quad \text{and} \quad (R_h(q(t_f))e_2)^T \frac{v_o(t_f)}{\|v_o(t_f)\|} \geq \cos \epsilon_r, \tag{A.1}$$

where $\epsilon_p, \epsilon_r \in \mathbb{R}_{>0}$ are prescribed tolerances on the position and angular errors, respectively, and $e_2 = (0, 1, 0)^T$. The second constraint above encourages the local $\hat{y}-$axis on the lacrosse head, which is orthogonal to the net's catching plane, to be aligned with the ball's velocity vector at t_f.

The constraint above is written assuming access to the ball's true 3D position and velocity. However, since we only have access to a prediction of these quantities via the parametric predictor $\hat{F}_o(\cdot; \theta_o)$, we enforce the above constraints w.r.t. the predicted quantities $\hat{p}_o(t_f; \theta_o), \hat{v}_o(t_f; \theta_o)$, making the endpoint catching constraint function $c(q(t_f), t_f; \theta_o)$ parametric in θ_o.

In conjunction with the hard constraints above, the terminal cost Ψ takes the following form:

$$\Psi(q(t_f), \dot{q}(t_f), t_f; \theta_o) := w_p \psi_p(q(t_f), t_f; \theta_o) + w_v \psi_v(q(t_f), \dot{q}(t_f)) \tag{A.2}$$

$$\psi_p(q(t_f), t_f; \theta_o) := \|p_h(q(t_f)) - \hat{p}_o(t_f; \theta_o)\|^2 + \left(1 - (R_h(q(t_f))e_2)^T \frac{\hat{v}_o(t_f; \theta_o)}{\|\hat{v}_o(t_f; \theta_o)\|}\right) \tag{A.3}$$

$$\psi_v(q(t_f), \dot{q}(t_f)) := \left\| v_h(q(t_f), \dot{q}(t_f)) - \begin{bmatrix} 0 \\ v_c \\ 0 \end{bmatrix} \right\|^2, \tag{A.4}$$

where $w_p, w_v \in \mathbb{R}_{\geq 0}$ are constant weights, and $v_h(q(t_f), \dot{q}(t_f)) \in \mathbb{R}^3$ is the lacrosse head local body-frame translational velocity, computed via the Jacobian-vector product $\partial_q p_h(q)\dot{q}$. The constant $v_c \in \mathbb{R}$ is a desired catching *speed*. Thus, the terminal cost Ψ penalizes the catching-time pose errors, as defined within (A.1), as well as the motion of the lacrosse head perpendicular to the ball's velocity vector at the catching instant.

The overall OCP is thus parametric in θ_o, the parameters of the ball's 3D predictor function $\hat{F}_o$, and problem parameters $\{\epsilon_p, \epsilon_r, v_c, w_p, w_v, \lambda\}$.

A.1 Conversion to Multi-Stage Trajectory Optimization

To begin, we assume that the acceleration limits are given by symmetric intervals $[-\ddot{q}_a, \ddot{q}_a]$, where $\ddot{q}_a \in \mathbb{R}^7_{>0}$ is a fixed vector. Then, we can define an $N-$stage discrete-time trajectory optimization problem, where each "stage" is composed of a constant acceleration phase followed by constant cruise phase. Formally, stage-k for $k \in \{0, \ldots, N-1\}$ lasts for $\delta t[k]$ seconds, where $\delta t[k] \in \mathbb{R}_{\geq 0}$. Then, within the acceleration phase of stage-k, joint $i \in \{1, \ldots, 7\}$ accelerates at $\pm \ddot{q}_a$ starting at $(q_i, \dot{q}_i)[k]$ to achieve a net velocity change of $\delta \dot{q}_i[k]$. In the cruise phase, the joint moves at a constant rate of $\dot{q}_i[k] + \delta \dot{q}_i[k]$ for $\delta t[k] - (|\delta \dot{q}_i[k]|/\ddot{q}_{a_i})$ seconds.

We can summarize the stage transition above by defining a composite state $x[k] := (q[k], \dot{q}[k], t[k]) \in \mathbb{R}^{15}$, and control $u[k] := (\delta \dot{q}[k], \delta t[k]) \in \mathbb{R}^8$. Then, the stage-"dynamics" are written as:

$$x[k+1] = \begin{bmatrix} q[k+1] \\ \dot{q}[k+1] \\ t[k] \end{bmatrix} = \begin{bmatrix} q[k] + (\dot{q}[k] + \delta \dot{q}[k])\delta t[k] - (1/2)\Delta^{-1}_{\ddot{q}_a} (\delta \dot{q}[k] \circ |\delta \dot{q}[k]|) \\ \dot{q}[k] + \delta \dot{q}[k] \\ t[k] + \delta t[k] \end{bmatrix} \tag{A.5}$$

where $\circ$ denotes the Hadamard product, and Δ_v is the diagonal matrix form of the vector v.

Let $\boldsymbol{u} := (u[0], \ldots, u[N-1])$. The stage-equivalent discrete-time objective is given as:

$$J(\boldsymbol{u}) = \sum_{k=0}^{N-1} \left(\lambda \delta t[k] + \|\delta \dot{q}[k]\|^2\right) + \Psi(x[N]). \tag{A.6}$$

Remark A.1. *Note that the exact conversion of the integral objective in* (2.1) *to the stage-wise discrete-time objective would result in a stage-cost of the form* $\lambda \delta t[k] + \ddot{q}_a^T |\delta \dot{q}[k]|$. *However, this was found to be numerically less robust than the* $\mathcal{C}^2$ *smooth objective used above.*

The terminal cost and endpoint catching inequality constraints from (A.1) carry over directly, and are applied to $x[N] = (q(t_f), \dot{q}(t_f), t_f)$, where $t_f = \sum_{k=0}^{N-1} \delta t[k]$. We now tackle the limit constraints on $(q, \dot{q}, \ddot{q})$. For acceleration, we require:

$$|\delta \dot{q}[k]| \leq \ddot{q}_a \delta t[k], \quad k = 0, \dots, N-1. \tag{A.7}$$

Since $\dot{q}(t)$ linearly interpolates between the stage-values $\dot{q}[k]$, the velocity limit constraints need only be enforced at the stage values:

$$\underline{\dot{q}} \leq \dot{q}[k] \leq \overline{\dot{q}}, \quad k = 0, \dots, N. \tag{A.8}$$

Finally, to handle the limit constraints on $q(\tau)$ for all $\tau \in [0, t_f]$, we must account for both the parabolic (constant acceleration) and linear (cruise) profiles within each stage. There exist two cases:

- Case 1: $\dot{q}_i[k](\dot{q}_i[k]+\delta\dot{q}_i[k]) \geq 0$. In this case $q_i(\tau)$ interpolates in-between $\{q_i[k], q_i[k+1]\}$ for all $\tau \in [t[k], t[k+1]]$. Thus, we need only apply the limit constraints on the endpoints $q_i[k], q_i[k+1]$.
- Case 2: $\dot{q}_i[k](\dot{q}_i[k] + \delta\dot{q}_i[k]) < 0$. In this case, there is a local max/min for $q_i(\tau)$ within $[t[k], t[k+1]]$ where $\dot{q}_i(\tau) = 0$. Denote this max/min as $\hat{q}_i[k]$. Then, in addition to enforcing the limit constraints at $q_i[k], q_i[k+1]$, we must also enforce the constraint on $\hat{q}_i[k]$. The expression for $\hat{q}_i[k]$ is given by:

$$\hat{q}_i[k] = q_i[k] + \begin{cases} \frac{\dot{q}_i[k]^2}{2\ddot{q}_{a_i}} & \text{if } \dot{q}_i[k] > 0 \\ -\frac{\dot{q}_i[k]^2}{2\ddot{q}_{a_i}} & \text{if } \dot{q}_i[k] < 0 \end{cases}.$$

Given the discrete-time "stage"-dynamics, optimization objective, and constraints, we can use any off-the-shelf constrained discrete-time trajectory optimization solver. In this work, we leverage Dynamic Shooting SQP, introduced in [20].

Remark A.2. *Note that the combination of max/min acceleration and cruise phases within each stage reflects the nature of mixed control-effort/minimum-time optimal control solutions, colloquially characterized as the 'bang-off-bang' strategy. Further, recent work [27] has shown that for LTI systems with a single control input, the optimal solution to a mixed control-effort/minimum-time problem with an endpoint reachability constraint is a sequence of "bang-off" stages. This justifies our use of such a stage-wise reduction of the original continuous-time OCP, and is similar in spirit to previous works on catching using trapezoidal velocity profiles [28].*

B Detailed Reward Functions

We provide detailed descriptions of the reward functions used for training the Blackbox policy and how they are computed below.

Object Position Reward. This reward is based on the closest distance the end-effector comes to the object during the episode. The closest distance is scaled on an exponential curve with a cutoff at $20cm$ scoring 1.0 for any episodes that get closer than this. This reward is used both in sim and real. This reward is useful for the Blackbox Policy to learn to get the net close to the ball.

Object Orientation Reward. This reward is based on the orientation of the net right when the ball gets to within $20cm$ of the net. The score is computed as a dot product of the velocity vector of the object and the axis of the net, scaled between 0 to 1 as a reward. This reward is only used in sim and encourages the policy to point the net in the right direction for a catch.

Object Stability Reward. This reward is based on how stable the object remains after it is close (defined as within $20cm$ of the net). Entering the close criteria and staying there through the end of the episode provides a flat 0.2 reward. The remaining 0.8 part of the stability reward is given by measuring the speed of the ball while it's close for $0.25s$. Each time-step during this duration contributes equally and is scored on an exponential curve based on object speed, capping out at speeds less than $0.2m/s$ scoring full for that timestep. A full score would be keeping the speed less than $0.2m/s$ for the full $0.25s$. This reward is only used in SIM as the precision of ball tracking is difficult when the ball is in the net or obscured.

Object Catch Reward. This reward is only used in real and is measured using a proximity sensor attached close to the net that can reliably detect whether a ball is in the net or not. The ball is declared as caught if the sensor detects a ball continuously in the net for greater than $0.25s$. This provides a flat 0 or 1 reward.

Penalties for exceeding Robot dynamic constraints. In Sim, there are multiple penalty rewards used to ensure the policy learns to operate within the robot constraints such as joint position, velocity, acceleration and jerk limits. The penalty rewards are implemented as a flat 1.0 if the agent actions stay within constraints and reduces to 0 depending on how much it violates them. The reward is reduced depending on how many timesteps and by how much it exceeds them. In Real the hardware produces a fault error code and freezes when movements exceed constraints. So in real the penalty is just scored based on whether or not the hardware encounters the fault code.