Receding-horizon online optimization for dexterous object manipulation

Tom Erez, Svetoslav Kolev, and Emanuel Todorov
University of Washington

Abstract— **We present a system for optimal control of dexterous manipulation. We use numerical optimization to automate the planning of the robot's motion while reasoning about its effects on the object. The user specifies the high-level task (e.g., the desired effect on the object), and the optimization allows the robot to autonomously discover the low-level details of the motion that fulfils this task. Applying this optimization in real time allows the system to adapt the plan to changes in the environment and the task. We demonstrate the effectiveness of the integrated system using a four-finger robot platform manipulating a cylindrical object.**

I. INTRODUCTION

Dexterous manipulation is an exciting open question in robotics: how multiple manipulators (e.g., fingers) can coordinate to effectively manipulate an object [1]. This is a challenge of hardware design – how to build strong and compliant fingers, as well as control — how to generate behavior that can intelligently affect the object in the desired way. The grasping community has dedicated much resources to this question, and several basic abstractions such as force closure [2] have proven instrumental in crafting solutions in various circumstances. However, autonomous real-time discovery of effective manipulation maneuvers in the general case is still an open challenge.

In circumstances where the environment can be designed and carefully controlled (such as an industrial setting), strong assumptions can be made to simplify the problem. The standard practice is for control engineers to craft a motion that can accomplish a specific task, and then use stiff robots to execute it. In contrast, we are interested in circumstances where the robot must interact with a non-static environment, where predefined motions cannot be used. In place of the control engineer making the decisions for the robot, we strive to build autonomous systems that can find effective behavior as the environment, and the task, rapidly change.

We use optimal control to automate the mapping from high-level goals to low-level control: we specify the task in the form of a cost function, and a numerical optimization algorithm is responsible for identifying the low-level details of the motion that accomplishes this task. Re-applying this optimization in real time allows the optimizer to adapt the plan to unexpected changes in the environment.

In our previous work we built a custom physics engine for optimization and control of contact-rich systems [3], designed trajectory optimization algorithms that use this engine to perform real-time optimization [4], and implemented a software framework that supports this core capability [5]. Here we present the first application of this framework to controlling robotic hardware (section III). The setup we present here consists of four 3-DOF Phantom robots, used as fingers that jointly lift a cylindrical object to a desired height and move it about the workspace as it tracks a target position (section V). While this is a simple task that can probably be achieved through a series of hand-crafted solutions, the optimization framework allows us to specify the task in high-level terms (in this case, referring to the desired state of the object), eliminating the need to tailor the controller to the specifics of a particular system and application. In this sense, the integrated system is generalizable, and would work just as well in many different robots and tasks.

Generating the control signal through online receding-horizon optimization is called Model-Predictive Control (MPC). However, as the name suggests, MPC requires an accurate dynamical model of the robot, its environment, and their interaction, and identifying such a model can be difficult and time-consuming. However, many robot platforms come equipped with a low-level, position-based controller, and we can use this controller to execute the optimal plan in the actuated dimensions of the system. If the plan is feasible (i.e., does not violate the constraints of the controller), and correctly predicts the effect on the unactuated dimensions of the system (e.g., the dynamics of the object), the task will be executed successfully despite lacking an accurate dynamical model of the robot. In order to explore the capabilities of such a hybrid system (online optimization + position control), the results we present here employ a low-level position controller we tuned for our hardware platform, which is responsible for executing the online-optimized plan.

Furthermore, many existing robotic platforms have limited contact sensing (or none at all). While we are actively developing a contact sensor for the fingertip of the Phantom, at this phase of the project we are interested to see how well the integrated system can control the object in the absence of contact sensing. This is a challenge since position control is not designed to account for the effects it may have on various objects in the environment. In this paper we propose a method that relies on contact smoothing in order to effectively use position control in the context of a manipulation task (section IV-B).

II. RELATED WORK

A basic challenge in contact-rich domains is the need to reason both about the motion of the system and the forces acting between the system and its environment. One basic and common method to bridge these two domains is

Preprint submitted to 2014 IEEE International Conference on Robotics and Automation. Received September 16, 2013.

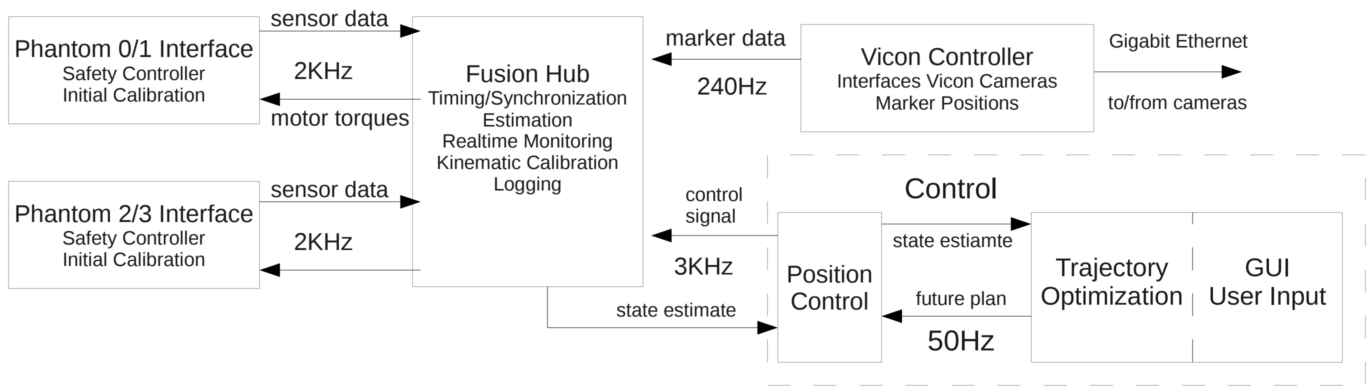


Fig. 1: An overview of the integrated system. The hub computer communicates at 2KHz with the Phantom computers and at 3KHz with the control computer. The effective estimation/control loop of the overall system is around 1KHz.

impedance control: by setting the target position for the robot inside the object, the system is driven to produce force on it. Here we achieve a similar effect by using contact smoothing (section IV-B).

In the literature on articulated robots, contacts are most often considered in the context of *collision avoidance* (e.g., [6], [7]). However, generating and executing effective plans in tasks that require multiple contacts between the environment requires reasoning about the dynamical effects of this physical interaction. The two classic examples for such collision-rich domains are locomotion and dexterous manipulation.

In the study of robot locomotion, two abstractions have emerged as useful in planning effective gaits. The older is Zero-Moment Point (ZMP) [8], which provides an effective heuristic to determine where to place the foot in the next step, given the current state of the system. Another such heuristic can be derived from the Spring-Loaded Inverted Pendulum (SLIP) [9], which accounts for the system's dynamics and can produce running gaits.

In the study of robot manipulation, the fundamental abstraction is that of closure [2]. This is fundamentally a static measure, as it measures how well can the system reject perturbations on the object. While other measures for the quality of a grasp are considered in the literature [10], effective reasoning on the mutual effects of multiple contacts is still considered a hard problem. Optimal control is capable in principle to identify effective behavior without relying on such heuristics. In practice, optimal control may benefit from incorporating these notions as cost terms, as it allow us to identify effective behavior with a shorter planning horizon.

An impressive example of generating manipulation behavior from optimization principles is quadrotor acrobatics [11]; however, this domain affords several simplifications that do not translate directly to robot hands. Using numerical optimization to plan manipulation trajectories in the general case has been considered before mostly in the graphics literature [12], [13], but while this work produces creative and impressive motion, it is currently too slow for real time control of a physical robot.

III. HARDWARE

Our hardware platform consists of four Sensable Technologies Phantom Haptic devices (three Phantom Premium 1.0 and one bigger Phantom Premium 1.5). Although these robots are traditionally used to render virtual forces, it is easy to use them as manipulators since the API allows us to directly set the current to the electric motors actuating the system at a 2KHz control loop. The spatial resolution of the joint encoders is 0.03mm. The maximum torque the motors can produce is estimated to be 0.14Nm, corresponding to 8.5N at the end-effector. The Phantom robot is cable-driven and the gear ratios are very low (approximately 10). These characteristics create a fast, compliant robotic manipulation system.

The phantom API allows up to 2 devices to be controlled by a single computer. Therefore, we use two dedicated computers to transmit the control signals and the joint encoder readings. The custom driver we built has certain built-in safety mechanisms, limiting the maximum joint velocities and preventing strong impacts with the joint limits. These two computers communicate with another computer that fuses the data streams (the phantom data and Vicon motion capture data, see below) and performs basic filtering and estimation. The control signal is generated by a computer that runs the optimization framework, which is the only computationally-intensive part of the system (section IV); here we use a server with two 6-core Intel X5690 processors. We use UDP for our network communications and we are able to maintain about 1KHz control loop over this system setup. See figure 1 for a schematic of the overall system setup.

In order to use the Phantoms for manipulation, we fitted custom fingertips as the end-effectors of the robots. We use 3d-printed L-shaped appendages with silicon-covered tips. These custom fingertips also contain a preliminary version of our force sensor that is under development in our lab. In this paper, however, we are not using these force sensors.

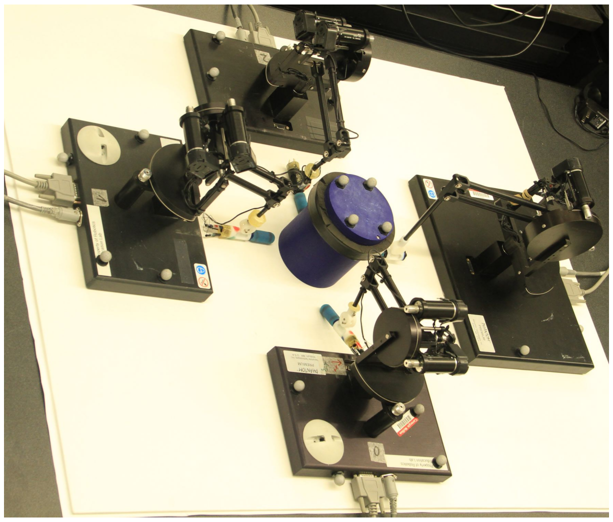


Fig. 2: A photo of the four Phantoms and the object. Note the silicon-covered fingertips and the reflective markers on the top of the object used for estimating the object's configuration.

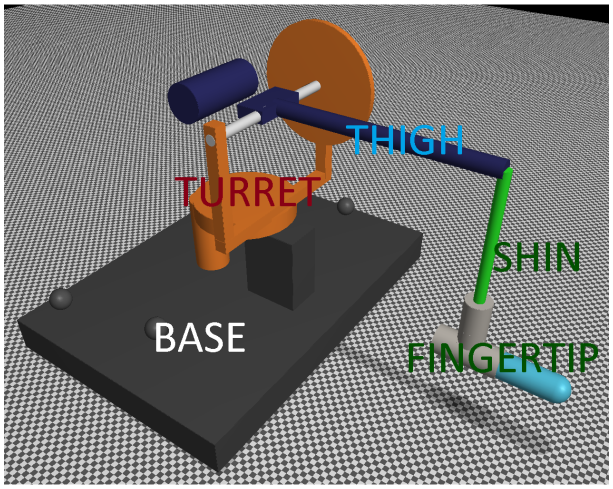


Fig. 3: The MuJoCo model of a Phantom robot. Note that the four-bar linkage coupling the top engine with the distal joint is not modelled. The exact position and orientation of the fingertip is one of the parameters discovered through the calibration process.

The realtime tracking of the manipulated object and the position of the Phantoms' bases is performed by Vicon Motion Capture system consisting of 7 Vicon Bonita VGA cameras running at 240Hz. The output of the Vicon is the position of all the motion capture markers it sees. In order to estimate the position of the object and the Phantoms, we perform object extraction to find the markers that belong to every single object we track, and use the Orthogonal Procrustes Analysis to find the position and orientation of the object being tracked.[1]

A. Kinematic calibration

It is important that the model's kinematics are accurate so that the planner can predict the robot's effect on the object. There are several sources for kinematic inaccuracy in a kinematic model of the Phantom robot: first, the joint encoders are not absolutely correct. The output of the encoders is a regular quadrature-encoded signal, outputting ticks for the direction of motion that are integrated by the Phantom PCI card, and therefore we need to manually identify the nominal position. Additionally, the mapping between the joint encoders and the joint angle is not straightforward — since the system is cable-driven, the thickness of the cable and the tension in the cable affect the ratio between the encoders and the joint angles. Another source of kinematic inaccuracy is our custom fingertip — although we designed it and know its dimensions, the assembly and mounting processes may introduce errors.

In order to eliminate these inaccuracies, we created a kinematic calibration process. First we use a custom-built calibration jig to manually bring the robot to the zero configuration, and set the Phantom drivers to this nominal position. Then we attach a Vicon marker to the tip of the end-effector and move the robots through a predefined trajectory that roughly explores the workspace for one minute. We use this data to optimize for the free parameters mentioned above with a simple gradient descent algorithm. Since we have a fairly-good initialization, this optimization process usually converges within a few iterations. After this calibration process, we estimate out kinematic error to be less than 1mm.

By looking at the correlation between the Vicon reading and the encoder output we can estimate the delay produced by the Vicon system. We find that the best calibration fit is produced when we introduce an offset of 11ms between the Vicon and the encoder data. This information is currently not used in our estimation/control loop, but it suggests that using Vicon for state estimation may become the limiting factor as we move forward to faster manipulation tasks.

Given a desired position and velocity, our low-level controller attempts to track both values using a simple Proportional-Derivative (PD) controller. Tracking a desired velocity is helpful since our we are tracking dynamic trajectories (as opposed to fixed set points). We empirically found proper gain values for this controller, with the coefficient on the position error being 10 times larger than the coefficient on the velocity difference.

Our optimization framework requires a model of the entire system — robot and environment. The Phantom platform employs a link loop structure in order to move the end-link actuator closer to the center of the system so as to reduce the robot's inertia and avoid excessive motor load. Although MuJoCo, our physics engine, can in principle handle equality constraints of this type, we found that their inclusion increases the computational cost with little added value. Therefore, we model the robot as a simple 3-DOF

[1]Even though the Vicon system provides that functionality out of the box, we chose to re-implement it on our side to allow for maximum flexibility and control over the process.

4-link manipulator (compare the robot in figure 2 to the model in figure 3). The kinematic transformation between our model and the Phantom is simple since the model's "shin" angle is simply the sum of the Phantom's "shin" and "thigh" joint encoders. We apply this transformation to the controller's output to obtain the control signal sent to the actuators.

IV. ONLINE OPTIMIZATION FRAMEWORK

Given a robot and a low-level controller, we need to provide it with a plan that realizes the high-level goals while adapting to the immediate state of the robot and its environment. Here, we employ an integrated framework for online optimization built for applications of this type [5]. The system is built around MuJoCo [3], a physics engine designed for real-time optimization and control. Our framework allows us to specify a dynamical model of the robot and the object in XML format, which also includes a section describing the cost function that defines the high-level task (section V). Formally, MuJoCo computes the forward model

$$\mathbf{x}' = \mathbf{f}(\mathbf{x}, \mathbf{u}) \tag{1}$$

given the system's state $\mathbf{x}$ and control signal $\mathbf{u}$, as well as and the cost function $c = \ell(\mathbf{x}, \mathbf{u})$. MuJoCo can also compute the first- and second-order derivatives of these functions if needed (at additional computational cost) for use in the optimization. These derivatives are used by our optimization algorithm to find a sequence of N future controls $\mathbf{U} = \{\mathbf{u}_1, \dots \mathbf{u}_N\}$ such that the future trajectory $\mathbf{X} = \{\mathbf{x}_0, \mathbf{x}_1, \dots \mathbf{x}_N\}$ (computed by integrating the dynamics $\mathbf{f}$ starting from our best estimate of the system's current state $\mathbf{x}_0 = \hat{\mathbf{x}}$) minimizes the cumulative cost

$$C = \sum_{i=1}^{N} \ell(\mathbf{x}_i, \mathbf{u}_i).$$

In order to find the optimal control sequence $\mathbf{U}^*$ we use the iterative-LQG algorithm [14] [4] which relies on the Bellman Principle to approximate the *optimal cost-to-go*:

$$V(\mathbf{x}, i) = \min_{\mathbf{u}} [\ell(\mathbf{x}, \mathbf{u}) + V(\mathbf{f}(\mathbf{x}, \mathbf{u}), i{+}1)]. \tag{2}$$

The algorithm first integrates (1) and computes a first-order expansion of the dynamics $(\partial\mathbf{f}/\partial\mathbf{x}, \partial\mathbf{f}/\partial\mathbf{u})$ and a second-order expansion of the cost around every step of the current trajectory, followed by a *backward pass* which computes a modification $\delta\mathbf{U}$ that brings the trajectory to a local minimum. Algorithm I provides a high-level overview of the iLQG algorithm.

At runtime, the optimizer is constantly sampling the current estimate of the system's state and producing new plans. The most recent solution is used to control the system while the optimizer is computing the next solution; therefore, only the initial part of every plan is ever executed. In order to maximize computational efficiency, we use a receding-horizon scheme — every optimization iteration is initialized with the previous optimum $\mathbf{U}^*$ and the most recent estimate of the system's state $\hat{\mathbf{x}}$.

Algorithm I Trajectory optimization

```
Inputs: The dynamics $\mathbf{f}$, the running and final costs $\ell_i, \ell_N$, the current state $\mathbf{x}_0$ and the warm-start sequence $\mathbf{U}$.
Outputs: A locally-optimal control sequence $\mathbf{U}^*$.
1) Rollout: Integrate $\mathbf{U}$ to get the initial $(\mathbf{x}_i, \mathbf{u}_i)$ trajectory.
2) Derivatives: Compute the derivatives of $\ell$ and $\mathbf{f}$.
3) Backward Pass: Approximate a local 2nd-order solution to (2), obtain a search direction $\Delta\mathbf{U}^*$.
4) Forward Pass: Integrate $\mathbf{U}+\alpha\Delta\mathbf{U}^*$ with several line search parameters $0 < \alpha < 1$ and pick the best one.
```

A. *Planning with approximate physics*

Creating a dynamical model for optimization presents us with a conflict: on the one hand, we want the model to be accurate; on the other hand, we want a numerical stability, a long planning horizon, and quick computation. In light of this conflict, it is reasonable to deliberately use an inaccurate model for planning, and trust that the online re-optimization will mitigate the errors introduced by this model mismatch.

Since many robots are shipped without a dynamical model but with a low-level position controller, in the current phase of this project we wish to explore whether our framework can generate dynamical behavior such as manipulation in the absence of an accurate dynamical model. Therefore, the dynamical parameters of the robots' model are arbitrarily chosen to maximize the numerical stability of the simulator, and do not reflect the physical reality of the robots. Specifically, by adding armature and damping to the joints we gain numerical stability and can take large timesteps when computing tentative plans. Since the computational cost of optimization is proportional to the number of steps in the planning horizon, larger time steps mean the optimizer can plan further into the future.

We assume that since these parameters only affect the dynamical model of the fully-actuated part of the system, the low-level position controller can realize any reasonable plan. In contrast, the dynamical model of the object needs to be accurate — in order to achieve effective manipulation, the planner must account for the dynamics of the object.

B. *Contact smoothing*

Using position control for a task that requires contact is not obvious — unless the contact is made with a predictably-soft surface (e.g., a linear spring), the robot's position is not informative of the contact forces applied once the robot and the object make contact. Note that this is not merely a limitation due to lack of force sensing. Even if we had a force sensor, the data it provides would be related to the control signal rather than the state, and so it is not clear how to incorporate this data into the planning process. One way to solve this limitation is with impedance control, setting a target position that lies inside the object, thereby causing the robot to apply forces on the object. However, using impedance control requires tuning additional parameters beyond the PD gains.

A similar effect can be achieved by planning through contact smoothing, where the optimizer expects the fingers to penetrate into the object upon contact. When using position control on a plan that involves smooth contacts, the target position lies inside the object, and therefore the robot will continue to push into the object even after contact is made. Contact dynamics in MuJoCo are modelled with an additional smoothness term [3]; the value of softness coefficient we used induces a penetration of 10mm when the object is simulated standing on the ground.

V. THE LIFTING TASK

The model used for optimization reflects the results of kinematic calibration (section III-A). The masses of every link of the robot are set to 1kg; in reality, we estimate that the phantom's limbs weigh about 50 grams, plus another 120 grams for the motor, so the dynamical model is off by a factor of 10; however, the low-level position controller is able to compensate for this modelling error. The manipulated object is a rigid cylinder with an added foam lip near its top (figure 2), weighing 200 grams. We embedded four reflective markers on the top lid in a known configuration that is used to track the object in the Vicon marker data. We use a planning horizon of 30 steps, which results in a policy update every 20ms on average (figure 4). Since MuJoCo can stably simulate the system at a timestep of 10ms, this results in a planning horizon of 0.3s.

The task involves bringing the object to a desired position. This is realized by a cost term penalizing the distance between the object's planar position and a planar target:

$$c_1 = \sqrt{\delta_x^2 + \delta_y^2}. \tag{3}$$

In order to maintain the object upright, we include a term penalizing the deviation angle from the vertical:

$$c_2 = \alpha. \tag{4}$$

Finally, we have a cost term for deviation of the object's height from a set value:

$$c_3 = \delta_z. \tag{5}$$

All these terms are only functions of the object's position. In order to ensure that the optimization is aware of the potential interaction between the fingers and the object, we add a term that drives the fingers to desired positions near the object. Given the end-effector position p_j of Phantom j and the position of the corresponding target t_j, which is attached to the frame of the object, we compute the distance $\Delta_j = ||p_j - t_j||_2$ and set:

$$c_t = \sum_{j=1}^{4} \Delta_j. \tag{6}$$

The total cost is a linear combination of these costs, along with a quadratic penalty on the applied torques:

$$\ell(\mathbf{x}, \mathbf{u}) = 0.001\mathbf{u}^\mathsf{T}\mathbf{u} + 0.01c_t + \sum_{k=1}^{3} c_k, \tag{7}$$

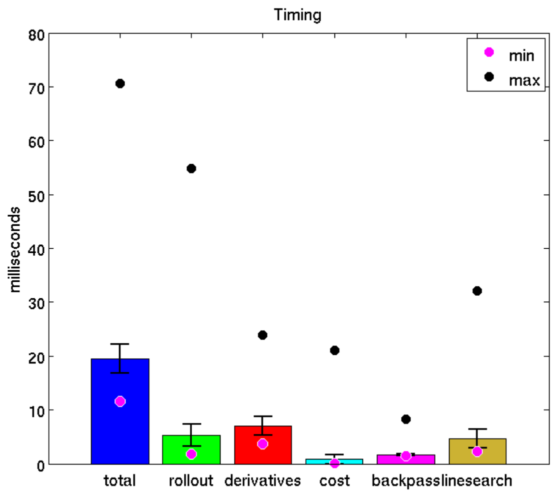
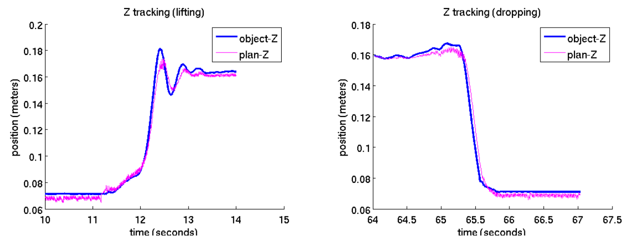


Fig. 4: Computation time for the online optimization. The first column represents the sum of all other columns. The dots show the maximum and minimum times recorded, and the error bars show one standard deviation.



(a) Lifting (b) Dropping

Fig. 5: Lifting and dropping performance. The purple dots indicate the planned position of the object's height, while the solid blue line indicates the actual trajectory.

and the weights reflect the relative importance of the different cost components.

The system's behavior is best illustrated by the movie attached to this submission. Here we quantify some of the behaviors seen in that movie by looking at the response of our system in various scenarios. The metrics we present are the target tracking ability, the mismatch between expectation and reality, as well as computational costs, which we expect to be a limiting factor in future deployments of the system.

Figure 5 shows the performance during lifting and dropping of the object. Some oscillations can be seen when we move the desired height of the object. This is probably due to our using very soft contacts in our planner: the planner produces trajectories in which the fingertip penetrates the object, and during lifting the desired target is higher than the fingertip's contact position with the object, leading to slight overshooting. Figure 6 shows a zoom-in of this overshoot, where the slower policy updates are visible.

Figure 7 shows the ability of our system to track a moving target. The task is to maintain a certain height while only

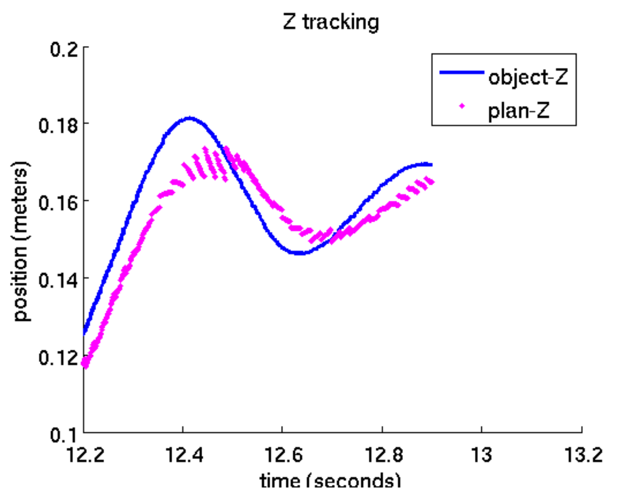


Fig. 6: Zoomed-in comparison of real trajectory versus planned height for the object. The policy updates are clearly seen at approximately 50Hz.

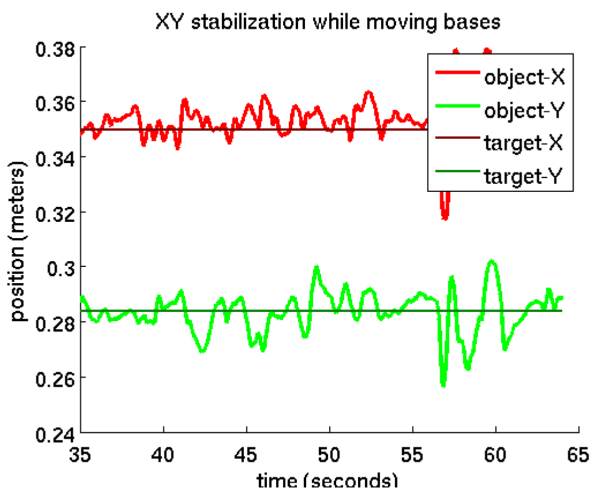


Fig. 8: Object position in the XY plane while the bases are being moved.

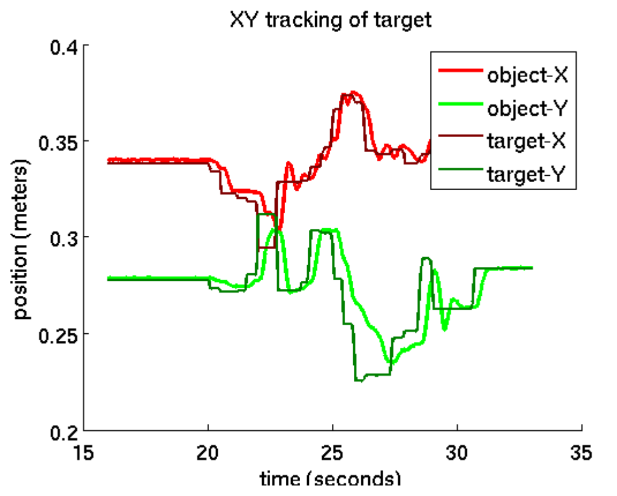


Fig. 7: Object position in the XY plane (light colors) overlayed with desired position (dark colors).

tracking the horizontal position of the target. The object clearly follows the target while exhibiting the same minor oscillations as in the lifting task. Figure 8 shows the response of our system while the bases of the Phantoms are being moved.

As the movie demonstrates, our system exhibits robustness to significant perturbations: we can push the Phantom arms and the object, and even lift and move the bases of multiple Phantoms, and the system recovers and finds new plans that maintains the object's desired state.

VI. Future work

There are several ways to extend the work presented here. Using a simple proportional controller to execute trajectories planned on a model with very soft contacts works well for simple tasks such as lifting and tracking a target, but this strategy would fail on tasks requiring more complex fingertip motions such as a finger-gait rotating the object. In order to apply MPC properly, we intend to identify the dynamical parameters of the Phantom robots and build a dynamically-accurate model. This will allow us to use our optimization framework for direct torque control.

Our dynamic planner relies on computing the contact forces, and even without an accurate dynamical model of the Phantoms, the forces acting on an object can be computed independently of the Phantom's dynamics. Therefore we expect those forces to produce the desired motion when applied to the object. We can then use a model with normal contact softness, and perform impedance control by addiing to the set positions a vector such that if the end-effector tries to follow it it would produce the desired forces. Since we know the contact forces from our planner, it is simply a matter of multiplying by the contact-space Jacobian to convert the forces to joint angle displacements.

Another route forward is to produce a good inverse dynamics model, and use it to compute the torques required to achieve certain accelaration at a given state (position and velocity) of the robot. Since we have a trajectory from our optimizer, we can readily compute the velocity and accelaration. A smaller gain PID controller can be used on top of the inverse dynamics model to correct for the model errors.

Acknowledgements

We thank Joseph Zhe Wu for his help designing the fingertip. This research was funded by DARPA and NSF.

References

[1] A. M. Okamura, N. Smaby, and M. R. Cutkosky, "An overview of dexterous manipulation," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 1. IEEE, 2000, pp. 255–262.

[2] A. Bicchi, "Hands for dexterous manipulation and robust grasping: A difficult road toward simplicity," *Robotics and Automation, IEEE Transactions on*, vol. 16, no. 6, pp. 652–662, 2000.

[3] E. Todorov, T. Erez, and Y. Tassa, “MuJoCo: a physics engine for model-based control,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems, (IROS)*, 2012.

[4] Y. Tassa, T. Erez, and E. Todorov, “Synthesis and stabilization of complex behaviors through online trajectory optimization,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems, (IROS)*, 2012.

[5] t. Erez, K. Lowrey, Y. Tassa, V. Kumar, S. Kolev, and E. Todorov, “An integrated system for real-time model predictive control of humanoid robots,” in *Humanoid Robots (Humanoids), 2013 13th IEEE-RAS International Conference on*, 2013 (accepted).

[6] J. J. Kuffner Jr and S. M. LaValle, “Rrt-connect: An efficient approach to single-query path planning,” in *Robotics and Automation, 2000. Proceedings. ICRA’00. IEEE International Conference on*, vol. 2. IEEE, 2000, pp. 995–1001.

[7] J. Schulman, J. Ho, A. Lee, I. Awwal, H. Bradlow, and P. Abbeel, “Finding locally optimal, collision-free trajectories with sequential convex optimization,” in *Proceedings of Robotics: Science and Systems*, Berlin, Germany, June 2013.

[8] M. Vukobratović and B. Borovac, “Zero-moment pointthirty five years of its life,” *International Journal of Humanoid Robotics*, vol. 1, no. 01, pp. 157–173, 2004.

[9] R. Blickhan, “The spring-mass model for running and hopping,” *Journal of Biomechanics*, vol. 22, pp. 1217–1227, 1989.

[10] R. Suárez Feijóo, J. Cornellà Medrano, and M. Roa Garzón, “Grasp quality measures,” 2012.

[11] M. Hehn and R. D’Andrea, “Real-time trajectory generation for interception maneuvers with quadrocopters,” in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 4979–4984.

[12] I. Mordatch, E. Todorov, and Z. Popović, “Discovery of complex behaviors through contact-invariant optimization,” *ACM Transactions on Graphics (TOG)*, vol. 31, no. 4, p. 43, 2012.

[13] Y. Ye and C. K. Liu, “Synthesis of detailed hand manipulations using contact sampling,” *ACM Transactions on Graphics (TOG)*, vol. 31, no. 4, p. 41, 2012.

[14] E. Todorov and W. Li, “A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems,” in *Proceedings of the 2005, American Control Conference, 2005.*, Portland, OR, USA, 2005, pp. 300–306.

Preprint submitted to 2014 IEEE International Conference on Robotics and Automation. Received September 16, 2013.