Physically consistent state estimation and system identification for contacts

Svetoslav Kolev and Emanuel Todorov

Abstract— **Successful model based control relies heavily on proper system identification and accurate state estimation. We present a framework for solving these problems in the context of robotic control applications. We are particularly interested in robotic manipulation tasks, which are especially hard due to the non-linear nature of contact phenomena.**

We developed a solution that solves both the problems of estimation and system identification jointly. We show that these two problems are difficult to solve separately in the presence of discontinuous phenomena such as contacts. The problem is posed as a joint optimization across both trajectory and model parameters and solved via Newton's method. We present several challenges we encountered while modeling contacts and performing state estimation and propose solutions within the MuJoCo physics engine.

We present experimental results performed on our manipulation system consisting of 3-DOF Phantom Haptic Devices, turned into finger manipulators. Cross-validation between different datasets, as well as leave-one-out cross-validation show that our method is robust and is able to accurately explain sensory data.

I. Introduction

Accurate models and state estimation are crucial components of model-based robot controllers. Such controllers usually utilize a planner with which the control algorithm imagines a future and then optimizes the plan to best satisfy user specifications. There are two critical requirements for this strategy to work: 1) We need to have an accurate estimate of our current position, otherwise the resulting plan would not make any sense and 2) we need to have an accurate physical model so that we are confident that the plan is feasible and will indeed be realized by our robot.

Model predictive control (MPC) is a particular implementation of that idea which is able to cope with slight estimation and modeling inaccuracies. The idea is that deviations from the plan accumulate slowly and smoothly; if we replan fast enough then the controller will still achieve the goal. MPC has been applied to contact rich tasks such as humanoid robot walking [1], where despite it being in simulation an MPC approach is still needed because the simulation model is different than the one used for planning. Model based control has even been applied to dexterous hand manipulation [2] where the planner has to reason about multiple acyclic contact events.

Still, applying model based controllers to real-world contact rich problems has been an elusive task. Even if we had perfect contact information, many tasks would still be difficult. For example, in manipulation tasks friction is of great importance, where not only the contact state matters but the friction force and normal forces are what allows one to hold an object against gravity, or change its configuration. This problem manifests itself both in the planner, which has to plan those contact forces, as well as in the estimator that needs to reason about them. Due to their nature, contacts introduce very sharp nonlinearities in the dynamical model, which makes the control and estimation tasks much harder. Even slight error in state information will cause incorrect contact state which will result in significant misprediction when used by a controller.

Robots are usually equipped with plenty of sensors, including force, tactile and motor torque sensors in an attempt to cope with those challenges. Similarly a human hand is covered with tactile sensors and our muscles are equipped with tendons which have force sensors embedded in them. Our brains are able to fuse this information into an intuitive state estimate. We are also able to predict the effects of our muscle actions on the objects we grasp. Similarly we aim to enable robots to do the same in what we call physically consistent state estimation. Physical consistency means our estimator is able to explain all the observed sensory data with its dynamical model of the world.

The standard procedure for system identification of a robot is to collect data while it is doing a task and then optimize over certain model parameters so as to maximize the predictive accuracy of the model. For example, if we have access to an inverse dynamic simulator, given a state trajectory it can compute the motor torques that must have produced that behavior, which we can compare with the collected data and determine success. Or with a forward simulator we can judge by its ability to accurately simulate certain time interval of the future. That strategy works well for smooth systems, but it fails as soon as we encounter contacts as shown in Figure 1. For example, trying to estimate the coefficient of friction between an end effector and a surface would be impossible if we had a distance of even $1mm$ between them in our state estimate (the typical error of motion capture system like Vicon and a robot like a Phantom). This is the case because estimating friction requires knowledge of the normal force, which in physics simulators is greatly dependent on the interpenetration (or distance in the case of remote contacts) between the two objects. Therefore it is essential to consider both problems of estimation and system identification together.

The main contribution of this paper is a joint system

Authors are with the Department of Computer Science & Engineering, University of Washington, WA 98195, USA
e-mail: swetko@cs.washington.edu, todorov@cs.washington.edu
This work was supported by the US National Science Foundation

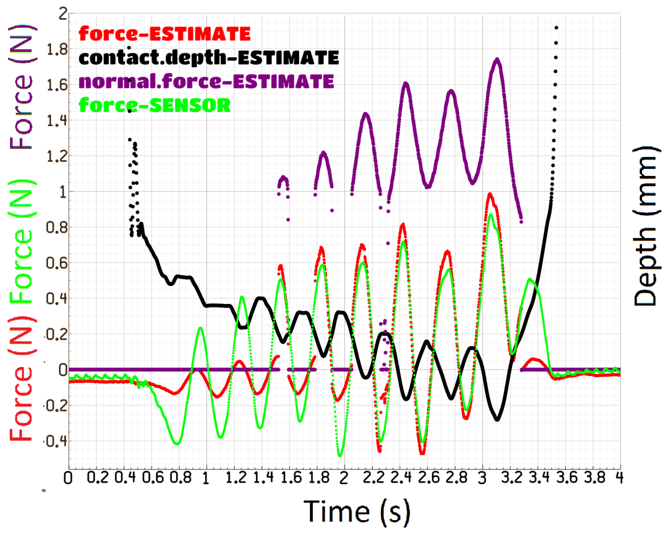

Fig. 1: Example of system identification failure trying to estimate the contact parameters between a robotic finger and a 3D printed object. The finger is sliding on the surface applying various amounts of pressure. The black graph shows the distance between the end effector and the stationary object, estimated purely from motion capture and joint encoder inputs. When the finger is too far from the object (more than $0.2mm$ away) the physics simulator predicts zero normal force (purple graph), making the estimated sensor reading (red graph) very different than the true sensor reading (green graph). The fluctuations of the red graph when not in contact are caused by the simulated sensor measuring the inertial forces of the moving end-effector.

identification and estimation framework that optimizes over both a trajectory and dynamical model with the goal of achieving high consistency. To the best of our knowledge this is the first paper to present such a combination while using a full-featured physics simulator, MuJoCo [3]. Furthermore we propose solutions to associated challenges such as tangential contact deformations and remote contact sensing.

This paper is organized as follows. In section II we discuss previous approaches to system identification and estimation. In section III we outline the mathematical formulation of the optimization algorithm. After that in section IV we detail some of the challenges in solving the posed optimization problem while in section V we detail some of the contact modelling challenges we encountered and their solutions. In section VI we describe the hardware setup and data collection. Section VII presents the results. We conclude in section VIII and discuss possible venues for future work.

II. Related Work

In our previous work [4] we developed an extension to Extended Kalman Filter (EKF) that had two important differences: Instead of keeping the state of the system for the last step only, we kept a fixed length interval and instead of a single Gauss Newton step we allowed for a varying number, with the usual lag/accuracy tradeoff. Looking at multiple steps in the past provides the estimator with richer context which can be used to arrive at a more physically consistent estimate. In this work we extend the algorithm to handle system identification at the same time as estimation.

Zhang et al. also considered the problem of object tracking and identification of contact parameters together by combining a visual datastream and tactile feedback [5]. However, they used a 2D physics engine and also their estimation is done via Particle Filtering which proved too slow for realtime, while our gradient based approach has been successfully applied to realtime applications [4]. Wan et al. attempt to solve the problems of state estimation, parameter estimation (system identification) and dual estimation (combination of the two) using an Unscented Kalman Filter (UKF) [6]. However, their results are not on robotic systems, but on artificial data, and it is not clear whether robotic system parameters can be estimated in an UKF manner, without considering the whole trajectory.

The SLAM (Simultaneous Localization and Mapping) problem is very similar to ours [7] [8]. Mapping corresponds to system identification (learning about the environment) and localization to state estimation. Instead of mapping the tangible aspects of the environment we are modeling dynamical properties of the environment and also the state of a robot consist of many DOFs, whereas in SLAM the state is usually 2D or 3D. SLAM is solved via landmark tracking techniques [7] as well as by dense vision approaches [8].

Vision based systems for state estimation have been very popular recently. Tanner et al. developed DART that uses a depth camera to track articulated bodies [9] and extended it by incorporating tactile information to enable tracking of objects even when they are partially occluded [10]. They use dense camera information which enables gradient approaches and consider contacts and interpenetrations carefully but do not use a full dynamical model and only consider a few predefined states. Particle filtering is another technique applied to estimation in contact rich tasks by Koval et al. [11], as well as by Chalon et al. [12] who also combined vision and tactile sensing. Koval et al. optimized the number of particles required by placing them on the contact manifold but still the number of particles required grows exponentially as the number of contact points increases. While vision methods provide good estimates based only on internal sensors they cannot provide the required accuracy that is needed by a physics simulator for planning, because for stiff contacts even small inaccuracies have great effect on the dynamics. We view this approach as complimentary to ours. For example, a vision based system can replace our motion capture system, as it is expected that robots should be able to operate in arbitrary environments that are not equipped with motion capture systems.

With regards to contact modeling, Gilardi et al. provide an overview of the various approaches and attributes to consider when implementing contact models [13]. Real contacts are rather complex and no existing physics simulator captures all aspects. We mention how we solve some of the arising problems in section V. Drumwright et al. survey the various implementations of multibody systems with contact [14]. Verscheure et al. identify contact dynamics with a stiff robot and using extremely precise measurement tools and do not use a general purpose physics model [15]. It is not clear how easily a general contact model can be incorporated into a physics engine, which is why we think it is important to perform the system identification within the framework of a physics engine.

The physics simulator that we use, MuJoCo [3], is a general purpose physics engine and allows simulation of robotic systems with multiple joints as well as handles contacts between bodies, equality constraints and tendons. MuJoCo relies on a new formulation of the physics of contact described in [3]. The computation of contact forces in forward dynamics reduces to a convex quadratic program, while in inverse dynamics the contact forces are computed analytically. This model also allows soft contacts, and has a rich parametrization making it suitable for system identification. Using a physics simulator such as MuJoCo bridges the gap towards control as MuJoCo has been successfully applied in trajectory optimization for control [2] [1].

III. Problem Formulation

Currently, the mode of operation of our framework is as follows. The robot moves around, either via teleoperation, or following a predefined trajectory, possibly interacting with the manipulated object. We record sensory data including motion capture data, an IMU, a 6-DOF force sensor at the end-effector, as well as joint angles and torques. We also construct a model in MuJoCo that mirrors the realworld scene. That model includes parameters such as masses, inertias, friction coefficients, contact softness, etc. We then ask the question: What is the robot/object trajectory and the set of model parameters that best explains the sensory data? We pose that question as an optimization problem as follows.

We model the trajectory as a sequence of states:

$$\mathbf{Q} = \{q_1, q_2, \dots, q_n\}$$

where n is the number of timesteps we consider. Each state is a vector $q_i = (\theta_1, \dots, \theta_{k'}, x, y, z, q_w, q_x, q_y, q_z)$, with $\theta_1, \dots, \theta_j$ representing joint angles and $x, y, z, q_w, q_x, q_y, q_z$ being the cartesian position and quaternion orientation of the manipulated object. We denote by j the number of joint angles and by $k' = j + 7$ the number of state variables. We do not consider velocities or accelerations separately but compute them via finite differencing from position data:

$$v_i = \frac{q_i - q_{i-1}}{h}$$

$$a_i = \frac{v_{i+1} - v_i}{h} = \frac{q_{i+1} + q_{i-1} - 2q_i}{h^2}$$

When computing the difference of quaternions we convert it to a 3D rotational velocity. Therefore the size of the velocity vector is k, same as the degrees of freedom of our system and one less than the number of state variables. The number of optimization variables is therefore nk.

The inputs to the optimization problem are the sensor readings and control signals for all timesteps:

$$\mathbf{S} = \{s_1, s_2, \dots, s_n\}$$

and

$$\mathbf{U} = \{u_1, u_2, \dots, u_n\}$$

Each sensor vector consists of l elements $s_i = (s_i^1, s_i^2, \dots, s_i^l)$, which includes joint angle sensors, motion capture position and orientation, the IMU data as well as the force sensor readings.

At the core of the optimization lies the physics simulator MuJoCo. It is used to predict accelerations or torques, in forward dynamics and inverse dynamics mode respectively. In forward dynamics mode the simulator computes system accelerations for a given timestep, given position, velocity and control signal for each DOF. In inverse dynamics mode, given position, velocity and acceleration at a given timestep it computes the generalized forces for each DOF, including the unactuated ones, required to produce the given motion. Since we compare the predicted forces to the control signal it is convenient that they have the same dimension so we define the control signal for the unactuated DOFs as 0.

The basic dynamics formulas are

$$(\hat{a}_i, \hat{s}_i) = \text{fwd}(q_i, v_i, u_i)$$

and

$$(\hat{\tau}_i, \hat{s}_i) = \text{inv}(q_i, v_i, a_i)$$

In both cases $\hat{s}_i$ is the predicted sensor output given the state and control signal, produced using MuJoCo via a generative sensor model.

Since we are doing system identification, we also optimize over a vector of c system parameters: $\mathbf{P} = (p_1, p_2, \dots, p_c)$, and generally we also have lower and upper bounds for these parameters: $p_i \in [l_i, u_i]$. The system parameters that we consider are detailed in section III-B.

With this setup we can formulate the following optimization problems:

- Problem formulation using inverse dynamics

$$\min_{\mathbf{P},\mathbf{Q}} \sum_{i=1\dots n} \|\hat{a}_i - a_i\|^{*_3} + \sum_{i=1\dots n} \|\hat{s}_i - s_i\|^{*_2}$$

- Problem formulation using forward dynamics

$$\min_{\mathbf{P},\mathbf{Q}} \sum_{i=1\dots n} \|\hat{\tau}_i - u_i\|^{*_1} + \sum_{i=1\dots n} \|\hat{s}_i - s_i\|^{*_2}$$

This optimization is done offline with the idea that the estimator part can be run at realtime once a suitable model is identified, as done in [4].

A. Cost terms

The difference between the two formulations is the cost term related to accelerations, in the case of forward dynamics, which is substituted to a cost term related to torques in the case of inverse dynamics. The norms used are denoted by $*_i$, meaning they are not just quadratic norms but can be arbitrary convex function. We call each element of the vector $\hat{s_i} - s_i$ a residual and generally we compute the norm of a residual vector r with the formula $\|r\| = \sum_i w_i f_i(r^i)$, where w_i is a weight and f_i can be any smooth convex function. For example we used the standard $f(r) = r^2$ or

$f(r) = e^{r^2} - 1$, which behaves similarly to the quadratic close to 0 but acts more like a barrier function further away. The details of specifying norms and residuals are outlined in [1]. Table I shows the set of cost terms that we used to generate our results.

TABLE I: Cost terms for forward dynamics. Acceleration terms are the difference between acceleration predicted from the physics engine and acceleration computed via finite differencing from the estimated trajectory. Sensor terms are deviations from sensor readings.

Term	*Units*	*Cost Term*
Force sensor	N	r^2
Torque sensor	Nm	$10^2 r^2$
Joint acceleration	$\frac{rad}{s^2}$	$10^{-5} r^2$
Joint position sensor	rad	$20 * 0.02^2 e^{\frac{1}{2}\frac{r^2}{0.02^2}}$
Object acceleration	$\frac{m}{s^2}$	$10^{-2} r^2$
Object rotational acceleration	$\frac{rad}{s^2}$	$10^{-4} r^2$
Object position Vicon sensor	m	$10 * 0.001^2 (e^{\frac{r^2}{0.001^2}} - 1)$
Object orientation Vicon sensor	rad	$10 * 0.005^2 (e^{\frac{r^2}{0.005^2}} - 1)$
Gyro sensor	$\frac{rad}{s}$	$10^{-1} r^2$

B. Physics parameters

The physics parameters that we optimize for fall in 3 categories. This first category is kinematic parameters. For example, in our datasets we have the object hanging on a string. This configuration introduces an inequality constraint (limiting the length of the string) in our system and MuJoCo models that with the same machinery as contacts, since an inequality constraint is very similar to frictionless contact. Therefore the string forces are greatly dependent on the position of its anchor point as well as the object. Similar to state information we can use a motion capture system to estimate the anchor position but we will have small inaccuracies again. Therefore we include the anchor position in our set of model parameters to optimize over. When alternating trajectory optimization and model optimization, we noticed very slow convergence in the case of hanging object. The reason is that given an inaccurate anchor position, the best trajectory is one that is shifted by the same amount as the error of the anchor, and given offset trajectory, the best anchor position is similarly shifted. Our joint optimization approach avoids this issue by taking optimization step with both sets of variables together.

The second category are intrinsic dynamic parameters of our robots. Those are the parameters that govern its interaction-free motion and include the positions of center of mass of all links, their masses and inertia matrices, as well as joint friction and damping. That is done in a separate process because those parameters are best inferred with a contact free behaviour.

The most important category is the parameters related to contacts. Many of the contacts encountered in manipulation tasks are with soft rubber-covered fingers. There are hard contacts as well, such as an object sitting on a table. In order to accommodate these different cases MuJoCo supports contact softness. When objects collide they experience each other's inertia. One way MuJoCo simulates softness is by scaling the apparent inertia depending on distance or penetration between the objects. In this paper the apparent mass scale factor increases quadratically with penetration. Another set of parameters is the more familiar spring parameters, namely damping and stiffness. Overall these parameters result in a position dependent mass on a spring damper system for contacts. The details of the MuJoCo contact model are explained in [3]. Of course the other important parameter is the friction which determines the shape of the friction cone. These are all parameters that we are interested in optimizing, with the option of them being different between different pairs of objects.

IV. Optimization Algorithm

We solve the optimization problem with Newton's method. It is a second order numerical optimization method, meaning it requires both gradient and a Hessian with which we iterate the solution until convergence. The method is summarized as follows: $x^i \leftarrow x^{i-1} - \alpha \mathcal{H}^{-1} g$. The derivatives are computed via finite differencing, as our physics simulator is not yet capable of producing derivatives of the dynamics. Since our optimization variables are non-homogeneous (model parameters and state variables), the computations follow different paths.

Since a point in our trajectory only affects the dynamics of the neighboring timesteps, computing the gradient is not very computationally expensive. It only takes $3kn + n$ dynamics evaluations to evaluate the cost and the gradient for a given trajectory. Again, since the effect of changing a trajectory point is only local, the trajectory Hessian is band diagonal and is also relatively cheap to compute via finite differencing. It will take additional $\frac{9k(k+1)}{2}n$ dynamics evaluations to compute it exactly. However, we do not actually use the real Hessian as it is not positive definite, which would make the use of Newton method infeasible. Instead we use an approximation of the Hessian: $\mathcal{H} \approx J^T J$, where $J_{ij} = \frac{\partial r_i}{\partial x_j}$, with r_i being residual number i and x_j being the j^{th} optimization variable. Computing J also allows us to compute the gradient by $g = Jr$. Computing J costs us $3nk$ dynamics evaluations and computing r is done by evaluating the dynamics for each timestep, hence the total number of $3nk + n$ evaluations for both the gradient and the approximated Hessian.

Unlike a state variable, a change in a model parameter, in general, will change the dynamics for all timesteps. Therefore evaluating the parameters gradient and approximated Hessian costs us cn, where c is the number of parameters. Computing the part of the approximated Hessian between the trajectory variable and the model parameters comes at no additional cost, given we have the residuals from the trajectory perturbations and model perturbations.

A typical trajectory of 20 seconds at 200Hz with 14 state variable yields an optimization problem with 56000 variables. Computing $\mathcal{H}^{-1} g$ directly is impossible. Therefore we use Cholesky decomposition and backsubstitution.

Fortunately the approximated Hessian is rather sparse with a band diagonal structure and there are fast algorithms for computing Cholesky decomposition and performing the backsubstitution. When we add c parameters to the optimization, the Hessian gets expanded with c dense rows and columns. Fast algorithms exist even in those cases and, overall, the computation of $\mathcal{H}^{-1}g$ takes less than 10% of the total execution time of our algorithm.

Our framework is flexible and it allows us to optimize only model parameters, only trajectory, or both at the same time. There are cases when trajectory optimization, which is computationally heavier, is not necessary (e.g. system identification of smooth dynamics), as well as cases when system identification does not make sense (e.g. running the system in realtime). Similarly to Wu et al. [16] we tried an EM approach to solve the combined problem, but noticed very slow progress in certain cases and moved to a full joint optimization.

When optimizing over just model parameters we found the Matlab Optimization Toolbox to be sufficient. However it did not scale well to hundreds of thousands of variable in trajectory optimization. Therefore we used minFunc [17] which provides a wide selection of algorithms and options but we found that Basic Newton method with Wolfe Line Search performs best for our problems. Still, even minFunc had trouble with computing $\mathcal{H}^{-1}g$, because, due to its huge size, $\mathcal{H}$, is numerically close to singular and the Cholesky decomposition algorithm fails, in which case minFunc was defaulting to the extremely slow eignevalue decomposition. We modified that part of the algorithm by adding some small constant to the diagonal of the Hessian in a Levenberg - Marquardt fashion so that it is clearly positive definite: $\mathcal{H} \leftarrow \mathcal{H} + \mu I \quad s.t. \quad \mathcal{H} \succ 0$.

V. Modelling

When defining the optimization problem we emphasize consistency over accuracy because accuracy is not well defined in the presence of modeling errors, and no physics engine can simulate the world with absolute accuracy. For example, a common way of representing robots is with joints and links, represented as rigid bodies. However, when forces are applied on such a structure long robotic links bend under the load. Then they no longer satisfy the rigid body hypothesis and in turn the joint angles lose their meaning as predictors of end-effector position via forward kinematics. Such non-rigidities are usually small but also much greater than the noise floor of joint-angle sensors. Another example is soft-body contacts. Like human fingers, robotics fingers are usually covered with soft rubbery material. Short of using Finite Element Methods there is no clear definition of an end-effector position. In this section we detail specific modelling choices that proved helpful in solving the system identification and estimation problems.

A. Remote Contacts

One feature that MuJoCo supports is contact forces between object that are not interpenetrating. As we mentioned

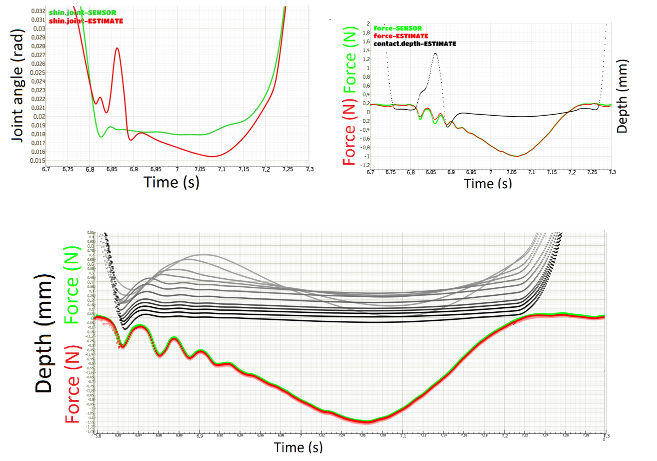

Fig. 2: The top figures show the result of running our estimation algorithm on a sample run consisting of the robot tapping an object. The remote contact distance is set at $0.1mm$. Clearly seen are artifacts in the estimates of the joint angle (top-left figure) and the resulting penetration error (black graph on the top-right figure). This is a local minima of the optimization. Still, the sensor output was predicted well (red graph) matching the raw sensor output (green graph). The bottom figure shows the result of successive optimizations done with different contact sensing distances, varying from $2mm$ (gray graph) to $0.1mm$ (black graph). Each optimization was seeded with the result of the previous, avoiding the local minima as a result.

in section III the apparent mass that the colliding bodies feel during contact depends on their interpenetration but we can extend that outside the boundaries of the objects so that forces are felt at distance greater than 0. That remote distance is another parameter that can be varied. This can be best thought of as an invisible soft pillow that cushions the impact between two bodies. This feature allows the trajectory optimization algorithm to see gradients and orient itself better in the almost discrete search space. By setting this parameter to be relatively large in the beginning we can guide the optimization process in its initial stages and then progressively decrease to 0 it to make realistic estimates. An example of our use of this idea is shown in figure 2. Also, that idea can help with control for hand manipulation [18].

B. Springy end effector

While the contact softening mechanisms in MuJoCo allows us to cope with rigid body violations in the normal direction of the contact, they still cannot explain tangential deformations. For example, when holding a heavy object the friction force would deform a human's skin in tangential direction, similar to what happens with the soft silicon rubber of the end effector. In order to cope with that we introduced extra joints for the end effector, therefore making the end effector non-rigidly attached to kinematic structure of the robot. The joints have spring-dampers attached to them that always aim to return them to nominal position. The idea is that surface deformations will be explained by load and displacement in those virtual springs. Naturally the

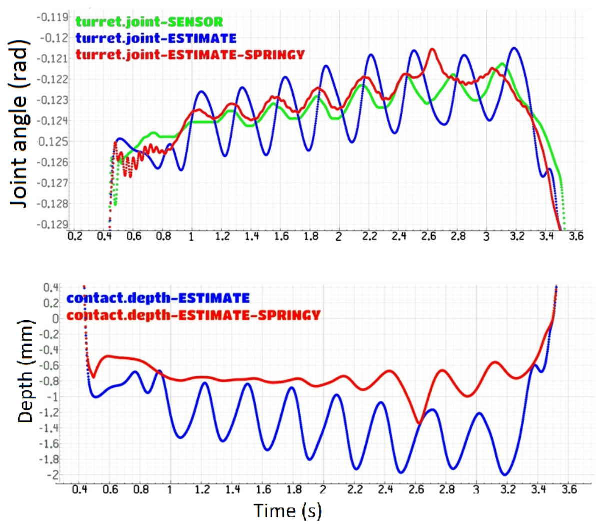

Fig. 3: Example of the effect of modelling the end effector as attached with a spring to the rest of the robot. In this experiment the end effector applies force on the object with a circular motion, causing lateral deformations of its silicon cover. **Top:** an estimate of a joint angle. Two things are to be noted here: 1) The joint estimate when not using a springy end effector (blue graph) has much higher amplitude compared to the sensory reading (green graph). This is result of the compliant robot links. 2) It also is phase-shifted, as a result of the silicon cover. The estimate with springy end effector (red graph) tracks the sensory readings much better. **Bottom:** Estimated contact penetration between the finger and the object. The estimate with the springy end effector (red graph) is much flatter than the one with rigid end effector (blue graph). As the finger was pressed against the object throughout the experiment the contact depth should not exhibit $1mm$ fluctuations.

spring-damper coefficients are another set of parameters we optimize over. Another benefit is that these springs can also explain bending and deformations of the kinematic structure. Figure 3 shows how the springy end-effector helps explain the sensory data better.

VI. Experimental Platform

A. Hardware Overview

We explore these algorithms with our Phantom Manipulation Platform. It consists of several Phantom Haptic Devices. We use each of them as a robotic finger. Each haptic device is a 3-DOF cable driven system shown in figure 4. Joints are equipped with optical encoders with resolution of about 5K steps per radian. The actuation is done by Maxon motors (Maxon RE 25 #118743) that despite the low reduction ratio are able to achieve 8.5N instantaneous force and 0.6N continuous force at nominal position. Even though they were designed as haptic devices the API allows for direct torque control which is what allows us to use them as general manipulators. The control loop is running at 2KHz.

For testing and developing our algorithms we use a single robot that performs various motions against an object. The robot was equipped with a silicon covered fingertip to enable friction and reliable grasping of objects. The softness of the rubber was one of the challenges that motivated us to

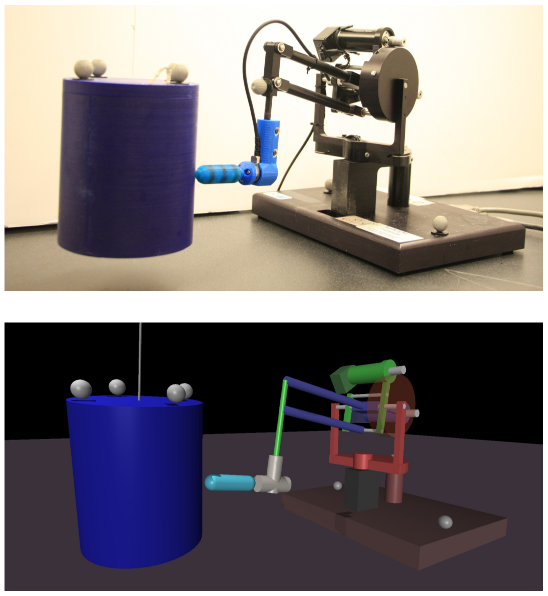

Fig. 4: **Top:** A single phantom robot manipulating a 3D-printed cylinder. **Bottom:** Model of the scene in MuJoCo. The white spheres are Vicon markers used for motion tracking.

carefully model contacts. There is a 6D force/torque sensor (ATI Nano 17) attached between the end effector and the robot. For our manipulation object, we used a 3D-printed cylinder with known sizes.

We also rely on Vicon motion capture system, which gives us position data at 240Hz. While being quite precise ($0.1mm$ error), the overall accuracy is significantly worse ($< 1mm$) due, in part, to imperfect object and manipulator models. The manipulated object is also equipped with an IMU (gyroscope and accelerometer) producing inertial data at 1.8KHz.

B. Data collection

When collecting data for system identification we want to have no unmodelled external perturbances. With our Phantom manipulation platform this is easy to achieve as we have several identical robots and teleoperation is a viable option since they are back-drivable. The experimental data that involves interaction with the manipulated object was collected via teleoperation with force feedback. For identifying the intrinsic robot model, we used a PID controller following a predefined trajectory.

Each sensor had different update rates therefore input data is collected at different frequencies. It poses a great software engineering challenge to handle non-homogeneous input frequencies. Therefore we resample all inputs at varying frequency. We assign finer resolution to interesting parts of the trajectory (contacts between robot and object) and lower resolution when the robot is just moving in the air. The frequency thus varied between 200Hz and 1KHz. Inaccuracies produced by this operation are easily tolerated since our state variables are optimization variables as well and any problems introduced by this operation will be cleaned up in the optimization phase.

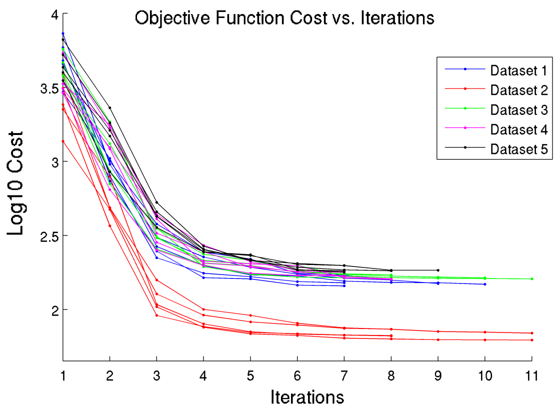

Fig. 5: Shown are the optimization traces for 5 datasets, each dataset colored differently. Here we run only the estimation part of our framework and for each dataset we perform 5 runs each using different model parameters found as described in section VII-B.

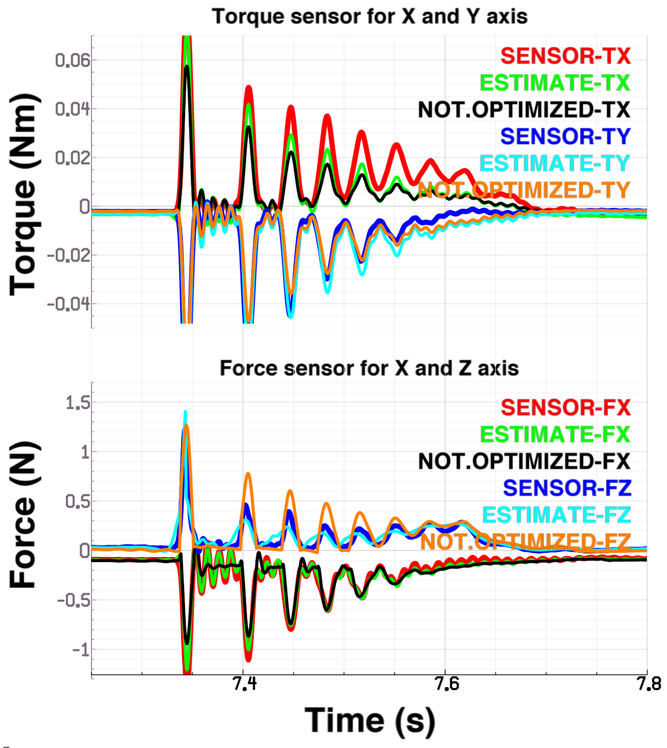

Fig. 6: ATI Nano17 sensor readings during typical interaction between the end effector and a hanging object. Shown is the difference between the sensory readings (red and blue graphs), the estimates predicted by our physics simulator at the end of a full trajectory estimation, considering both sensors in the cost function (green and cyan graphs) and the estimates generated when not penalizing deviations of the corresponding sensor channel (black and orange graphs). While considering all sensor inputs in the optimization gives us more accurate estimates, omitting some sensors does not result in overfitting and our framework correctly predicts the missing sensor values. Only 4 out of the 6 sensor channels are shown since the other two closely match the shown channels.

VII. Results

Figure 6 is an excerpt of our results showing the loadcell readings during a typical tap of the end effector on the hanging object. We note the oscillatory nature of the contact, as well the intermittent period within the contact duration where the force readings are flat. What happens is that the finger bounces off the object and contact reoccurs several times until a firm contact is established. This is due to the compliant structure of the robot as well as to the weight difference between the robot links and the object which is about 3 times heavier.

Figure 5 shows the objective function value as a function of optimization iteration. The overall shape is consistent across different datasets. The number of iterations required to reach close to the minimum is about 6.

A. *Leave-one-sensor-out prediction*

Here we show that our framework is robust to missing sensory readings. We run the estimation two times: 1) not penalizing for deviations from the force output of the ATI Nano17 and 2) not penalizing for deviations from the torque output. This simulates missing sensor readings. Figure 6 shows that our predictions of the missing sensor readings match the sensor output well.

TABLE II: Model parameters values from different datasets

	Parameter			
Dataset #	*Coefficient of friction*	*Contact stiffness* $\frac{N}{m}$	*Spring stiffness* $\frac{Nm}{rad}$	*Spring damping* $\frac{mNm}{\frac{rad}{s}}$
Nominal	1.000	100	50.0	10.0
$ID\#1$	0.247	221	48.4	19.9
$ID\#2$	0.288	94.8	38.5	3.2
$ID\#3$	0.268	196	41.9	16.6
$ID\#4$	0.213	292	46.9	12.4
$ID\#5$	0.250	338	41.4	20.2

B. *Cross-validation between datasets*

Here we show that our framework is robust to overfitting. We perform joint system identification (sysID) and estimation on each of 5 datasets independently. The model parameters found in each of the 5 runs are shown in table II. Then we perform only the estimation part of the algorithm on all 5 datasets using the model parameters from: 1) hand tuned model parameters and 2) the five sets of parameters found found previously when running sysID (labeled $ID\#1 \ldots ID\#5$) on each of the datasets. The minimized objective function values are summarized in table III. Within each dataset the cost when using the parameters estimated from the same dataset does not differ significantly from the cost when using parameters identified using the other datasets. In all cases, the resulting cost is significantly lower than when using the hand-tuned model parameters. The model parameters found using different datasets are fairly close with few exceptions. The contact stiffness and the damping of the end-effector spring parameters vary between the different datasets. This is discussed in the next section.

TABLE III: Crossvalidation between different datasets

	Model parameters from					
Dataset #	*Manual*	$ID\#1$	$ID\#2$	$ID\#3$	$ID\#4$	$ID\#5$
1	246.7	**112.8**	122.8	103.2	114.4	106.3
2	124.6	60.96	**50.69**	58.45	56.10	56.97
3	296.8	126.0	154.8	**133.4**	126.7	151.9
4	282.7	143.2	144.4	126.9	**128.2**	137.3
5	342.8	118.5	157.9	111.0	171.0	**147.1**

C. Robustness to parameter initialization

We run our algorithm on the same dataset, starting with 33 different initial parameter values. Table IV shows the resulting distributions of final cost and the estimated parameters. Again we note that the contact stiffness and the damping of the end-effector spring parameters have bigger variance. This indicates that the datasets do not contain enough variability of dynamical motion which will be addressed in future work.

TABLE IV: Final values distribution

Parameter	Units	Mean	Variance
Final Cost		126.7	18.0
Friction coefficient		0.2770	0.0079
Contact softeness	$\frac{N}{m}$	272.3	102.9
Spring stiffness	$\frac{Nm}{rad}$	41.3	0.93
Spring damping	$\frac{mNm}{\frac{rad}{s}}$	15.86	1.51

VIII. Conclusion and Future Work

We present a framework for system identification and dynamically consistent state estimation. We show results for basic robot-object interactions. By enabling consistent state estimation we hope to improve model based controllers in the real world. We show that using a hand-tuned model parameters produces worse estimates than when using system identification together with the estimation.

There are two venues for future work. First, we will improve the optimization tools by enabling constrained optimization in the parameter space. A problem we encountered is the optimizer asking our physics simulator to reason about non-physical values such as negative mass or coefficient of friction. Another improvement would be to optimize over different behaviors jointly as well as to increase the complexity of the scene by adding more robotic fingers.

While we performed various validations and show that our optimization is robust to change in input data, it remains to prove its effectiveness to control applications. The most important venue for future work is to implement realtime version of the algorithm and apply the realtime estimation in a model based controller. Closing the loop with a controller would be the ultimate validation tool as we can quickly determine what is the quality of the results needed to achieve certain tasks. Another advantage is that we will be able to check which modelling choices are helpful in estimation and not a hindrance in control. In general, a more complicated model would give us better explanatory power in the estimation phase but would make controller or planner's work harder. Keeping a sane middle level is very important and closing the loop is what would allow us to do so. We see our framework as a tool to discover new ways to model robots and their interaction with the environment in applications of state estimation and control.

References

[1] 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.

[2] V. Kumar, Y. Tassa, T. Erez, and E. Todorov, "Real-time behaviour synthesis for dynamic hand-manipulation," in *Proceedings of the International Conference on Robotics and Automation (ICRA 2014).*

[3] E. Todorov, "Convex and analytically-invertible dynamics with contacts and constraints: Theory and implementation in mujoco," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on.* IEEE, 2014, pp. 6054–6061.

[4] K. Lowrey, S. Kolev, Y. Tassa, T. Erez, and E. Todorov, "Physically-consistent sensor fusion in contact-rich behaviors," in *IEEE/RSJ International Conference on Intelligent Robots and Systems, (IROS)*, 2014.

[5] L. E. Zhang and J. C. Trinkle, "The application of particle filtering to grasping acquisition with visual occlusion and tactile sensing," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on.* IEEE, 2012, pp. 3805–3812.

[6] E. Wan, R. Van Der Merwe, *et al.*, "The unscented kalman filter for nonlinear estimation," in *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000.* IEEE, 2000, pp. 153–158.

[7] M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit, *et al.*, "Fastslam: A factored solution to the simultaneous localization and mapping problem," in *AAAI/IAAI*, 2002, pp. 593–598.

[8] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking," in *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on.* IEEE, 2011, pp. 127–136.

[9] T. Schmidt, R. Newcombe, and D. Fox, "Dart: Dense articulated real-time tracking," *Proceedings of Robotics: Science and Systems, Berkeley, USA*, vol. 2, 2014.

[10] T. Schmidt, K. Hertkorn, R. Newcombe, Z. Marton, M. Suppa, and D. Fox, "Depth-based tracking with physical constraints for robot manipulation," in *IEEE International Conference on Robotics and Automation*.

[11] M. C. Koval, M. R. Dogar, N. S. Pollard, and S. S. Srinivasa, "Pose estimation for contact manipulation with manifold particle filters," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on.* IEEE, 2013, pp. 4541–4548.

[12] M. Chalon, J. Reinecke, and M. Pfanne, "Online in-hand object localization," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on.* IEEE, 2013, pp. 2977–2984.

[13] G. Gilardi and I. Sharf, "Literature survey of contact dynamics modelling," *Mechanism and machine theory*, vol. 37, no. 10, pp. 1213–1239, 2002.

[14] E. Drumwright, D. Shell, *et al.*, "An evaluation of methods for modeling contact in multibody simulation," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on.* IEEE, 2011, pp. 1695–1701.

[15] D. Verscheure, I. Sharf, H. Bruyninckx, J. Swevers, and J. De Schutter, "Identification of contact dynamics parameters for stiff robotic payloads," *Robotics, IEEE Transactions on*, vol. 25, no. 2, pp. 240–252, 2009.

[16] T. Wu, Y. Tassa, V. Kumar, J. Movellan, and E. Todorov, "Stac: simultaneous tracking and calibration," in *IEEE/RAS International Conference on Humanoid Robots (HUMANOIDS)*, 2013.

[17] M. Schmidt, "Minfunc," 2005.

[18] I. Mordatch, Z. Popović, and E. Todorov, "Contact-invariant optimization for hand manipulation," in *Proceedings of the ACM SIGGRAPH/Eurographics symposium on computer animation.* Eurographics Association, 2012, pp. 137–144.