Estimation of Inertial Parameters in Simulation

Keegan Go Kenji Hata

***Abstract*—In this paper, we develop and test two methods for estimating the inertial parameters of a robot in simulation. This problem is difficult due to the high dimensional space, and the nonlinearity of the problem. Since our goal is to develop methods that could be applied to a real robot, we cannot examine the parameters directly and instead use an error that reflects how well a set of parameters can predict its future position and velocity. We find that minimizing this prediction error does not improve our estimation of the inertial parameters. However, these estimated parameters do significantly increase the accuracy of predicting future positions and velocities.**

I. Introduction

In robotics, it is essential for the model of the robot to be both accurate in order for the calculations necessary for control and operation to be precise for fine tasks. Previously, estimations of these parameters were done by disassembling a robot and meticulously computing the inertia parameters of each link. [1] However, disassembling a robot is a time-consuming process, especially for robots with complex link-joint structure and high degree of freedom (dof). Additionally, this approach does not scale well if we want to dynamically choose new payloads for the end effector.

Subsequent work has been done on creating methods for both offline and online estimation of parameters through examination of data about a robot's movements over time. Khosla and Kanade lay out one framework which takes the Newton-Euler formulation of the dynamics of a robot and turns the system into a linear system which can be solved. [2] This initial approach found that some parameters could not be identified due to the assumptions made, and so additional knowledge about the system is required in order to determine the parameters. Khosla continued this work, applying the method to the CMU DD Arm II. [3]

Since then, new methods for modeling robot dynamic parameters have been developed. Olsen and Petersen use a statistical approach and maximizing likelihood to generate an estimate. [4] Bompos used this method and applied it to the Mitsubishi PA-10 arm, but focused mostly on obtaining parameters for this device rather than developing a generalized approach. [5]

Following up on this work, we will develop two general methods to estimate the inertial parameters of any robot. Both of these methods represent a data centric approach that relies on collecting a large number of samples about the robots movements and using this data to attempt to determine the inertial parameters. Rather than use a sophisticated formulation of the robot, these methods use an error-metric to guide a search over the parameter space.

II. Setup and Data Model

A. Experimental Setup

Our testing framework is built on the Standard Control Library (SCL), which provides a complete robotic simulation and includes convenient interfaces for control and a physics engine to run the simulation. [6] We used SCL to perform both data collection as well as to test our estimated parameters.

B. Problem Specification

Our model for a robot uses 7 inertial parameters for each actuated link of the robot: mass, center of mass (x, y, z), and rotational inertia (xx, yy, zz). A configuration consists of a complete choice of these parameters for all links in a robot.

In our simulation, the configuration used to initialize the robot was considered the true configuration. This set of inertial parameters would be the unknown values that we try to estimate if our tests were run on a real unknown system. To accurately simulate this scenario, we never access this configuration directly, and can only learn about it by examining how the state of the robot changes over time when exposed to different forces. We also assume that we are given a configuration that represents an initial guess of true inertial parameters of the robot. While initially this was nothing more than a seeding point, we later chose to use the initial guess to impose some reasonable bounds on the problem (see Constraints section).

During testing, we decided to run our parameter estimation on the Puma[1], whose full configurations were provided by the Stanford Robotics Lab. On each robot, we ran a GC sinusoidal controller with different frequencies on each joint to ensure we tried a wide range of motions.

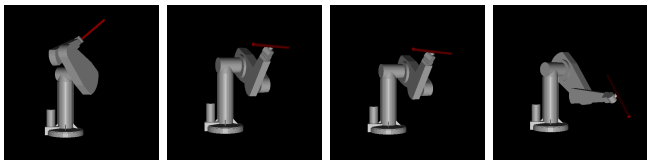

Fig. 1: Puma in SCL simulation

C. Parameter Error

Given a configuration with parameters $\{p'_1, ..., p'_n\}$, we can compute two measures of error. The first is called parameter error, and is

$$\sum_i \left(\frac{p_i - p'_i}{p_i} \right)^2 \tag{1}$$

where p_i is the value of the actual parameter. The division by p_i is needed because our inertial parameters inherently differ by up to 3 orders of magnitude. This division ensures that each parameter is fairly represented in the error.

D. Generalized Coordinate Error (GC Error)

However, since computing parameter error requires knowledge of the true parameters, we cannot use parameter error outside of an analysis of our methods. Therefore, we have

[1]The Puma is a 6-dof industrial robot consisting of all revolute joints.

developed a second formulation of error which we call generalized coordinate error or GC error. In generalized coordinates, each link's state can be represented by two scalars: one for position and one for velocity. A vector q is defined that contains all the position scalars for all links, and dq is similarly defined as containing all the velocities values for all links. q and dq together make up the complete state s of the robot at a given time. GC error is then computed as

$$\sum_i (s_i - s_p)^2 \tag{2}$$

where s_p is the predicted state and s_i is the actual state for some initial condition i.

In practice, we collect samples of initial states (position and velocity) as well as the forces that are being applied on our simulated robot. For each such condition we record the resultant state of the robot after a fixed time interval. This pair of initial state and final state make up one data point in our set. Given another configuration, we can then simulate running the robot from the same initial conditions with the same forces applied, and compare the final state to the recorded state obtained from simulating the robot with the true configuration.

E. Relationship between Parameter Error and GC Error

It is difficult to show any clear relationship between these two measure of error. However, they clearly must have a common global minimum because, if we chose our testing configuration to be the true parameters, we find both a zero parameter error and a zero GC error. Intuitively, our ability to predict our position-velocity state should get better as our configuration approaches the true values. Part of the objective here is to understand whether there exists a discernible relationship between these two quantities that we can use to estimate the parameters from only the GC error.

III. Algorithms

A. Nonlinear Optimization

Since the integration equations with respect to the parameters are nonlinear, we used a nonlinear optimization (NLO) algorithm to find a configuration of parameters which minimizes GC error. Our NLO starts with an initial estimate configuration and creates two mutations from this initial configuration, resulting in three seed configurations. We then use a nonlinear optimizer[2] on each of these seeds, and choose the configuration with the minimal GC error among the optimized configurations. This configuration becomes the new initial estimate and the process is repeated until the GC error becomes zero.

B. Genetic Algorithm

We have also implemented a genetic algorithm (GA) as another method to search for inertial parameters. In contrast to the NLO, the GA runs much more quickly because each iteration of the algorithm needs to simulate the robot once for each configuration in the population, whereas the NLO needs to simulate many times per configuration.[3] As a result, the GA can maintain a much larger population of configurations and run many more iterations in the same amount of time as NLO.

In our GA, we choose individuals - configurations - using roulette wheel selection where the fitness of the individual is measured as the inverse of the GC error. Mutations were performed at the level of single parameters, where a new parameter is selected from a normal distribution centered on the current value and with a standard deviation proportional to the current value. Crossovers were performed by selecting two individuals of the population and generating a new configuration whose parameters were randomly chosen from one of the two individuals.

In addition to the basic algorithm, we also implemented what we refer to as the "migration" step. After generating the new population, we find the centroid of all individuals, and then move all individuals away from this center point. This helps ensure that the population stays reasonably dispersed, and allows us to sample a greater portion of the configuration space.

C. Constraints

Since it is impossible for inertias to have negative values, we initially placed a non-negative constraint on inertias in our NLO and GA.

However, the lack of other constraints allowed the NLO and GA to output configurations that had some parameters with nearly-impossible values. Assuming that one takes a reasonable initial estimate of the robot's configuration, we further constrained all parameters to be within $\pm 50\%$ of our original estimation to prevent our search from drifting. In the case of robots such as the Puma, this assumption should hold. However, when parameter values are small, we acknowledge that this assumption becomes too strong, as the range of the constraint is consequently small; a different constraint may be needed.

Finally, recognizing that the entire robot's mass is constant and can be easily measured, we placed a constraint on the sum of the masses to be constant. This final constraint was placed only on the NLO due to the NLO's tendency to drift, whereas the GA's masses were stable.

IV. Results

Both of the methods described above yield configurations that do significantly better than the original initialized parameters. Below we show the graphs of the two errors for each method. Note that the GC error is significantly reduced on both robots in both methods. (Fig. 2)

These results also hold up for different patterns of motion other than the one data was collected on. (Fig. 3)

To understand how these error compare against those from other parameters, we generated a histogram of errors for configurations that are near the true configuration. Randomly generated a number of configurations where each parameter of

[2]We used NLopt, an open-source library created by MIT. The specific optimizing algorithm we chose was Constraint Optimization by Linear Approximation (COBYLA), which supports the constraints we use to narrow the search.

[3]NLO makes approximately ten thousand of these simulation calls per configuration optimization.

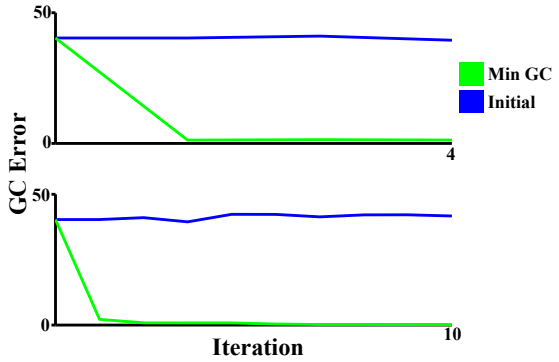

Fig. 2: GC errors of NLO (top) and GA (bottom) over the training iterations

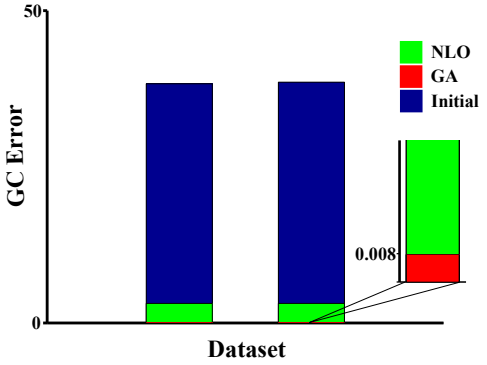

Fig. 3: GC error of two different control paths

the configuration was chose to be within a 2% range of the true value for that parameters. The green bar highlights the bucket into which our average error on the testing data falls into. (Fig. 4)

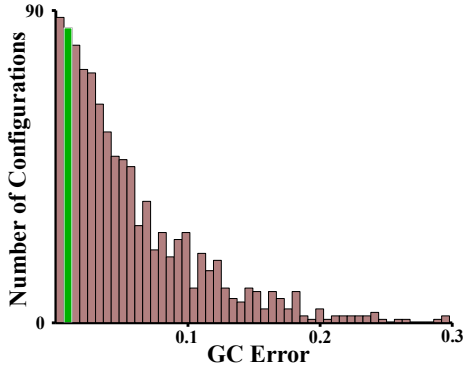

Fig. 4: GC errors of randomly generated configurations where parameters are within 2% of true values

A. Relation between Parameter Error and GC Error

We find that the correlation between GC error and parameter error was not as strong as we hoped. While they both have the same global optimum, we never found a case where the true configuration was generated. At each iteration of both NLO and GA, we recorded two configurations: the one that minimizes parameter error and the one that minimizes GC error. For each of these configurations we then compute the parameter error as a metric of whether our populations of configurations trend toward the true configuration. However, our results showed that finding low GC errors does not correlate well with finding low parameter errors. (Fig. 5)

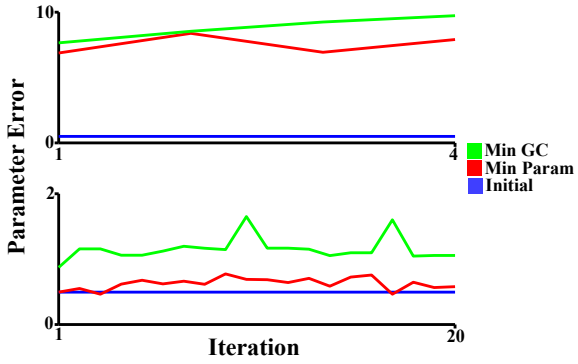

Fig. 5: Parameter errors of NLO (top) and GA (bottom) over the training iterations

One possibility is that our model of an inertial configuration is over-parameterized. Clearly there may be some parameters that can never be estimated due to the particular combination of links. In this two dimensional case, the product of the mass by the offset of the center of mass determines the rotational inertia. Since we can only apply torques to this revolute joint, we will never be able to tell the difference between any pair of mass and center of mass whose product is the correct rotational inertia.

V. Future Work

One possible change to make would be to sample over different motions and control patterns in hopes that GC error becomes a better approximator for parameter error. Another would be to combine the two algorithms to see whether the GA would help the NLO cover more of the parameter space.

Acknowledgment

We thank Samir Menon for providing us with SCL and advising us. We also thank Ellen Klingbeil for her helpful suggestions.

References

[1] Brian Armstrong, Oussama Khatib, and Joel Burdick. "The explicit dynamic model and inertial parameters of the PUMA 560 arm." IEEE International Conference on Robotics and Automation. Vol. 3. IEEE, 1986.

[2] Pradeep K. Khosla and Takeo Kanade. "Parameter identification of robot dynamics." IEEE Conference on Decision and Control. Vol. 24. IEEE, 1985.

[3] Pradeep K. Khosla. "Estimation of robot dynamics parameters: Theory and application." 1987.

[4] Martin M. Olsen and Henrik Gordon Petersen. "A new method for estimating parameters of a dynamic robot model." IEEE Transactions on Robotics and Automation. IEEE, 2001.

[5] Nikolaos A. Bompos, et al. "Modeling, full identification and control of the mitsubishi pa-10 robot arm." IEEE/ASME international conference on Advanced intelligent mechatronics. IEEE, 2007.

[6] Samir Menon. Standard Control Library. http://www.stanford.edu/~smenon/projects.html

[7] Steven G. Johnson. The NLopt nonlinear-optimization package. http://ab-initio.mit.edu/nlopt