Estimating Object Inertia Properties for Trajectory Control of Manipulator

Aamir Rasheed

Introduction

Historically, robots have been used in controlled environments, where variables such as position and inertial properties of objects are known. As robots become more ubiquitous, techniques are required to help robots deal with increasing uncertainty. In this project, I will develop an approach for trajectory tracking of a manipulator that has successfully picked up an object of unknown inertial properties. My goal is to estimate the relevant inertial properties of an object so that the robot arm can continue following the desired trajectory.

Robotics Background

Trajectory Tracking with PID Control

To track a trajectory, we use a PID control law

$$\begin{bmatrix} F^* \\ M^* \end{bmatrix} = \ddot{x}_d - k_v(\dot{x} - \dot{x}_d) - k_p(x - x_d) \tag{1}$$

evaluated at each time step, where $\ddot{x}_d$, $\dot{x}_d$, and x_d are our desired acceleration, velocity, and position vectors with dimensions 6x1 (3 for linear motion and 3 for rotational motion) of the end effector of the robot. $\dot{x}$, and x are 6x1 vectors describing the end-effector's current velocity and position, k_v is the integral gain, k_p is the proportional gain, and the output, F^* and M^*, together form the 6 x 1 acceleration vector (3 dimensions of linear acceleration, 3 dimensions of rotational acceleration) we have computed to attempt to stay on the trajectory.

Generating Torques with the Dynamics Equation

To convert the desired acceleration vector into motion, we must compute the necessary torque commands for each joint's motor through the standard robot dynamics equation. We assume the robot is a multi-linked arm with n joints and $n + 1$ links, where the $n + 1$th link is the end-effector of the robot, i.e. the part of the robot following the trajectory. To convert the acceleration vector computed above to torque commands, we use the standard robotic dynamics model

$$\Gamma = AF^* + b + g \tag{2}$$

where A is the n x 6 kinetic energy matrix of the robot, F^* is the 6×1 acceleration vector from our PID control law, b is the $n \times 1$ vector of centrifugal and Coriolis forces, g is the $n \times 1$ joint gravity torques vector, and Γ is the $n \times 1$ vector containing the torque commands sent to each of the n motors associated with each joint. It's important to note that in practice, we ignore centrifugal and Coriolis forces due to estimation errors.

Problem Statement

For my project, I have chosen to tackle the problem of a manipulator following a trajectory while grasping an object of unknown intertial properties. To simplify the project, we assume that the object is already rigidly grasped with respect to the end-effector of the robot.
When there is an extra mass at the end effector, we add a term to our dynamics equation (2) to compensate for the extra forces, resulting in

$$\Gamma = AF^* + b + g + {}^0J_{com}^T \begin{bmatrix} F_{mass} \\ M_{mass} \end{bmatrix} \tag{3}$$

where ${}^0J_{com}$ is the $6 \times n$ Jacobian matrix of the center of mass (COM) of the object frame with respect to the base frame, and F_{mass} and M_{mass} are the 3×1 vectors representing, respectively, the 3-dimensional force and 3-dimensional moment needed at the end effector to compensate for the mass. To compute F_{mass} and M_{mass}, we model the mass at the end effector as a point mass with mass m at its COM, with the vector r representing the position of the center of mass with respect to the end effector frame. Using this, we can then model F_{mass} and M_{mass} as

$$\begin{bmatrix} F_{mass} \\ M_{mass} \end{bmatrix} = \begin{bmatrix} m\hat{g} \\ -\hat{g} \times mr \end{bmatrix} \tag{4}$$

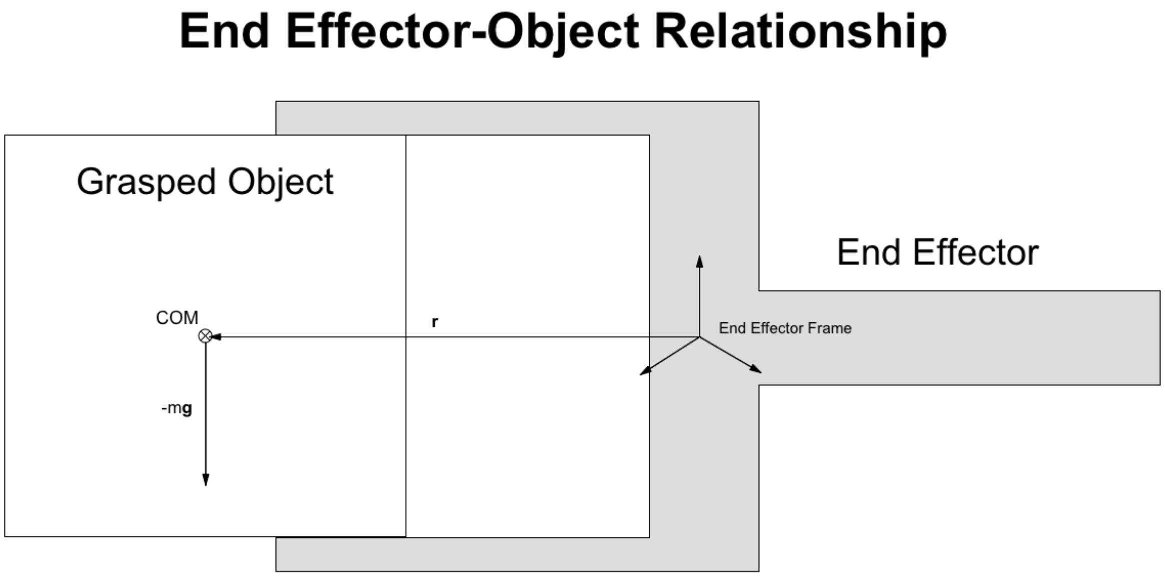

Figure 1

where $\hat{g}$ is the 3×1 world frame gravity vector (denoted as bolded g in Figure 1), m is a scalar for the mass at the COM, r is the vector from the end effector frame origin to the COM frame, and $\hat{g} \times mr$ is the cross product. We can decompose (4) to separate the knowns (gravity vector) from the unknowns (mass at COM and location of COM) to yield

$$\begin{bmatrix} F_{mass} \\ M_{mass} \end{bmatrix} = \begin{bmatrix} \hat{g} & 0 \\ 0 & -[\hat{g}]_x \end{bmatrix} \begin{bmatrix} m \\ mr \end{bmatrix} \tag{5}$$

where $-[\hat{g}]_x$ is a 3×3 cross product operator matrix, m is a 1×1 scalar, and mr is a 3×1 vector. So, in order to determine the force necessary to maintain the trajectory of the object (as well as compute the Jacobian at the com) we must find out what the mass and COM are with the use of the inbuilt force and moment sensor in the end effector of the robot. We can model the end effector mass and force as

$$\begin{aligned} F_s &= F_{mass} + F_{bias} \\ &= m\hat{g} + F_{bias} \\ &= \begin{bmatrix} \hat{g} & I \end{bmatrix} \begin{bmatrix} m \\ F_{bias} \end{bmatrix} \qquad (6) \\ M_s &= M_{mass} + F_{bias} \\ &= r \times m\hat{g} + F_{bias} \\ &= \begin{bmatrix} -[\hat{g}]_x & I \end{bmatrix} \begin{bmatrix} mr \\ M_{bias} \end{bmatrix} \qquad (7) \end{aligned}$$

where F_s and M_s are the sensor readings of force and moment sensed at the end effector and F_{bias} and M_{bias} are the errors in the sensor readings. So, to estimate F_{mass} and M_{mass} for the mass-compensated trajectory tracking, we have to estimate m, mr, F_{bias}, and M_{bias}.

Method

General Learning Approach

To compensate for the underlying unmodeled noise in the sensor readings, we can rewrite (6) and (7) as a general learning problem, resulting in

$$h(F_s) = \begin{bmatrix} m \\ F_{bias} \end{bmatrix} \qquad (8)$$

$$g(M_s) = \begin{bmatrix} mr \\ M_{bias} \end{bmatrix} \qquad (9)$$

where h and g are learned mappings from the sensor readings F_s and M_s to the unknown inertial properties vector m, and mr and sensor biases F_{bias} and M_{bias}.

To do this, we take samples of sensor reading data of a manipulator grasping objects of different masses and COMs. For each object, we record the mass and center of mass in a table

$$\begin{aligned} Y &= \{(m_1, mr_1), (m_2, mr_2), \ldots, (m_n, mr_n)\} \\ Y_0 &= \{m_1, m_2, \ldots, m_n\} \\ Y_1 &= \{mr_1, mr_2, \ldots, mr_n\} \end{aligned}$$

for n objects. For each object i, we will record sensor data of the robot

$$\begin{aligned} X_i &= \{(F_{s,1}, M_{s,1}), (F_{s,2}, M_{s,2}), \ldots (F_{s,d}, M_{s,d})\} \\ X_{i,0} &= \{F_{s,1}, F_{s,2}, \ldots F_{s,d}\} \\ X_{i,1} &= \{M_{s,1}, M_{s,2}, \ldots M_{s,d}\} \end{aligned}$$

as it moves around a trajectory t with d sensor readings, where each $F_{s,j}$ and $M_{s,j}$ is a 3×1 vector at sensor reading j. Then, we can reframe (8) and (9) as:

$$h(X_{i,0}) = Y[i]_0 \qquad (10)$$

$$g(X_{i,1}) = Y[i]_1 \qquad (11)$$

where $X_{i,0}$ refers to the sensor samples F_s for object i, $X_{i,1}$ refers to the sensor samples M_i for object i, $Y[i]_0$ refers to object i's mass m, and $Y[i]_1$ refers to object i's COM mr, and h and g are functions that map from the sensor readings to the object mass properties.

We generated trajectory data using SAI 2 (Stanford's Simulation and Active Interfaces robotics simulation platform) for 20 different cubes, with COMs varying from 1 to 2m from the grasping point, and masses ranging from 5-10kgs.

Linear Regression Approach

To solve this using linear regression, we can rewrite (6) and (7) as

$$W \begin{bmatrix} X_{i,0} \\ X_{i,1} \end{bmatrix} = \begin{bmatrix} m \\ F_{bias} \end{bmatrix} \tag{12}$$

$$\tag{13}$$

where W is a $4 \times (d \times 3)$ weight matrix representing the coefficients weighing each sensor reading to generate the output. We can fit this weight matrix using linear regression.

Neural Network Approach

The architecture consisted of an input layer of size $2 \times 3 \times d$, a number of hidden layers, and an output layer of size 4. Unfortunately, due to time constraints, not many architectures were tested, though we had the best result of the few we did test with 150 nodes each in the hidden layers.

Results, Discussion, and Future Work

Due to time constraints, not many models were tested. We tested the results on the generated trajectory data mentioned above. The linear regression model performed better than the neural network model. The linear regression model estimated the mass of the object incorrectly by about 3kgs on average, and the COM by about 0.3m on average, whereas the neural network estimated the mass of the object incorrectly by about 3.5kgs on average, and the COM was estimated incorrectly about about 0.5m on average.

Unfortunately, these results are not accurate enough to be legitimately useful for the problem this approach was attempting to tackle. My hypothesis is that this is due to the trajectory not having enough variation to generate varied enough data, so my future approach would include tuning the trajectory to get more varied training. I would also look further into the field of weight estimation from a control theory perspective more, to get more inspiration for potentially bringing something new to a learning perspective.

References

Python Packages Used

1. scikit-learn
2. keras
3. numpy

References

[1] Khatib, Oussama et al. "A Unified Approach for Motion and Force Control of Robot Manipulators: The Operational Space Formulation" IEEE Journal of Robotics and Automation. Feb. 1987. stanford.edu. `https://cs.stanford.edu/groups/manips/publications/pdfs/Khatib_1987_RA.pdf`.

Github Repo

`https://github.com/aamirrasheed/estimating_robot_params.git`