Position Estimation for Control of a Partially-Observable Linear Actuator

Ethan Li
Department of Computer Science
Stanford University
ethanli@stanford.edu

Arjun Arora
Department of Electrical Engineering
Stanford University
aarora52@stanford.edu

Jay Whang (CS 238)
Department of Computer Science
Stanford University
jaywhang@stanford.edu

Abstract

In this work, we propose and compare position estimation approaches to enable control of a low cost linear actuator with partially observed states. Our goal is to use machine learning techniques to obviate the need for expensive hardware while still being able to achieve precise and accurate actuation to arbitrary target positions. We analyze the performance of regression methods for this task and evaluate their application to feedback control in comparison to controllers with ground truth position. We find that regression methods achieve the required position estimation accuracy, and a simple linear model integrated into feedback control performs well on our control task given hardware limitations. These results demonstrate that a machine learning approach can enable precise actuation with our lower-fidelity hardware.

1 Problem Statement

1.1 Background

Electromechanical linear actuators are used in a wide variety of applications to create electronically controlled linear motion. However, few (if any) provide a wide range of precise motion at commercial prices appropriate for low-cost devices. We are working with Prof. Ingmar Riedel-Kruse on development of a low-cost educational liquid-handling robotics kit which proposes a design for linear actuation using position sensing at a lower precision than is standard (Fig. 1).

High-precision position control of this actuator could enable use of our robotics kit for scientific protocols but at roughly 5% of the cost of the cheapest commercially-available liquid-handling robot [1], making benchtop lab automation widely accessible for classrooms and do-it-yourself biotechnology experimentation. However, the proposed actuator cannot be controlled using classical approaches due to the lack of a high-precision position feedback sensor. The control strategy must also cope with uncertainties arising from slop in the mechanical components, transient electrical responses in the motor, and variable timing delays from serial data transmission in the control loop.

Prior work proposing application of machine learning methods to state estimation for low-level control of physical systems with partially observable dynamics has been carried out on related systems. For example, [2] applied recurrent neural networks for state estimation in a partially-observable variant of the cart-pole system, with training and evaluation performed in a physics simulator, and system state completely hidden except pole angle, which is observed perfectly. Among evaluations of machine learning methods for real hardware and unreliable sensors, [3, 4, 5, 6, 7, 8, 9] report success with regression methods to compensate for environmental disturbances of force and displacement sensors,

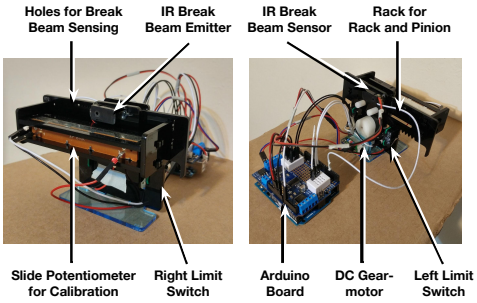

Figure 1: Prototype of the proposed linear actuator design. The proposed actuator consists of a DC motor which drives a rack-and-pinion mechanism. As the rack moves past an infrared break beam emitter-sensor pair, a pattern of holes cut into it alternately blocks and unblocks the beam of infrared light from the emitter, causing a binary output change from the sensor. A 6 cm slide potentiometer is additionally mounted to the rack to provide a ground truth position signal for controller evaluation.

and some of these papers evaluated the use of these methods for force control. Our work follows a similar approach to apply regression methods for state estimation during control of novel hardware from noisy and sparse sensor outputs to accomplish a different task.

1.2 Problem Formulation

For CS 229, we reduce the problem of actuator position control to that of actuator position estimation for a standard position-derivative (PD) controller with manually tuned control parameters, as shown in Fig. 2. Because PD control assumes enough observability about system state to compute derivates of feedback input at every time step, we require a system to estimate denoised feedback input at every time step from sensor data which are sparse in time, noisy, and only indirectly dependent on actuator position. We focus on stateless regression methods which take an input vector of features at a single timestep and output a position feedback to the controller.

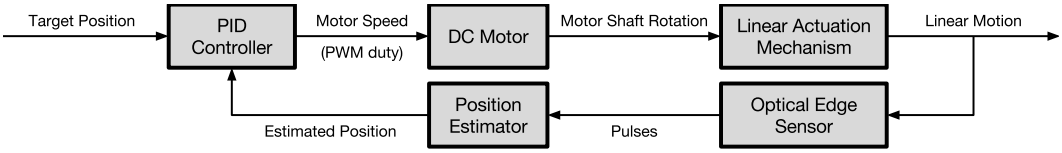

Figure 2: Block diagram of the proposed feedback control loop.

1.2.1 Evaluation

For the position estimation task, we evaluate regression models for position estimation by the root mean squared errors (RMSEs) of their estimates from ground truth positions for all readings across a set of episodes. Final evaluation of a position estimator is done by implementing it in a closed-loop controller of our hardware prototype as in Fig. 2. For the controller evaluation task, we score a controller during a control episode by the squared distances between the actuator's ground truth position and the target position, accumulated over every time step of the entire episode. We then compute the mean scores over a series of episodes. This score thus rewards rapid actuation towards the target and penalizes large positioning errors much more than small errors.

As benchmarks for our regression models, we also implemented and evaluated the following controllers:

- Naive Baseline estimator: we hand-crafted a position estimator function and implemented it on the microcontroller driving our linear actuator. This estimator accumulates edge counts as the motor moves. These edge counts are rescaled and used directly as the position estimate.
- Precise and Imprecise Gaussian Oracle estimators: our oracle position estimators are synthetically constructed as the ground truth positions with added Gaussian noise. Noise

parameters were chosen based on the desired accuracy of the position estimator, namely such that the estimated position is within 0.5 mm (for the precise oracle) and within 1 mm (for the imprecise oracle) 99.7% of the time.

2 Data

We collected data from the linear actuator to train our models and achieved a data collection rate of 100 readings per second, with each control episode lasting for approximately 1 second.

Our training data is stored in a CSV format with columns representing different features and rows representing a single reading. Readings from a single episode are stored in a consecutive block in the order they are generated. The features used for our experiments consist of the states of limit switches, the last limit switch pressed, accumulated edge counts from the optical edge sensor as the motor moves in each direction from the last limit switch pressed, the direction and power currently being delivered to the motor, and timing measurements of various events such as the time since the last edge was pressed. Ground truth positions are collected from a slide potentiometer as discrete positions from 0 to 1023, with 12 positions per millimeter.

We also constructed a modified dataset from the same readings but without time-dependent features, one from the same readings but with additional engineered features consisting of inverses of event timing measurements, and one from the same readings with engineered features and all two-feature polynomial interaction terms.

We ran the device for $N = 23060$ episodes, collecting total of 3.1 million readings, which were then split by a 80-20 train/dev split. We used our train set to train our position estimators. We had to exclude approximately 200K rows for episodes in which some edges were not counted by the optical sensor.

3 Methods

We evaluated four methods, all using implementations provided by the Scikit-learn library ([10]).

3.1 Linear Regression

This is the simplest and probably most familiar form of regression that we utilized. We simply use the closed form solution for Linear Regression given our feature Matrix X and training examples y_{train}. The loss function for this model is just the standard Mean Squared Error Loss

$$L = ||X\theta - y_{train}||^2$$

We used this linear regression model with our standard feature set; we did not need to rescale the features, as we found that rescaling to zero mean and unit variance had no effect on model performance.

3.2 Ridge Regression with Interaction Features

This model is the same as the above linear regression model except we add a regularization term to our loss function

$$L = ||X\theta - y_{train}||^2 + \alpha||\theta||^2$$

We used ridge regression rather than linear regression with the interaction features because we found that regularization was required to avoid overfitting on the interaction features due to linear dependence between various features. We found empirically that a regularization actor of $\alpha = 10$ was sufficient to prevent overfitting. As with linear regression, rescaling features had no effect on model performance.

3.3 Gradient Boosted Regression Trees

We used gradient boosting to fit a sequence of shallow decision trees as weak learners. At each stage of gradient boosting, a refined model for the next stage is built by adding another tree fit to the residual between ground truth and the prediction from the model at the current stage. We used gradient boosted regression trees with squared loss and engineered features; using 400 decision trees of depth 6 seemed to optimize accuracy while avoiding overfitting.

3.4 Support Vector Regression

Support Vector Regression is an extension of the use of Support Vector Machines used in class for classification, solving the objective

$$\min_{w,b,\zeta_i,\zeta_i^*} \frac{1}{2}||\theta||^2 + C\sum_{i=1}^{n}(\zeta_i + \zeta_i^*)$$

subject to:

$$y_i - \theta^T\phi(x_i) - b \leq \epsilon + \zeta_i$$
$$\theta^T\phi(x_i) + b - y_i \leq \epsilon + \zeta_i^*$$
$$\zeta_i, \zeta_i^* \geq 0, i = 1, ..., n$$

The decision function is :

$$\sum_{i=1}^{n}(\alpha_i - \alpha_i^*) * K(x_i, x) + \rho$$

Where K is the standard kernel of the SVR algorithm. The two kernels we used were the linear Kernel (which behaves similarly to linear regression) and the Radial Basis function (RBF) kernel.

We chose our cost penalty C to be 100 for the RBF Kernel and 1000 for the Linear Kernel as they seemed to produce the best output from a grid search of hyperparameters. All other hyperparameters are the standard values from the scikit-learn SVR API.

4 Results and Analysis

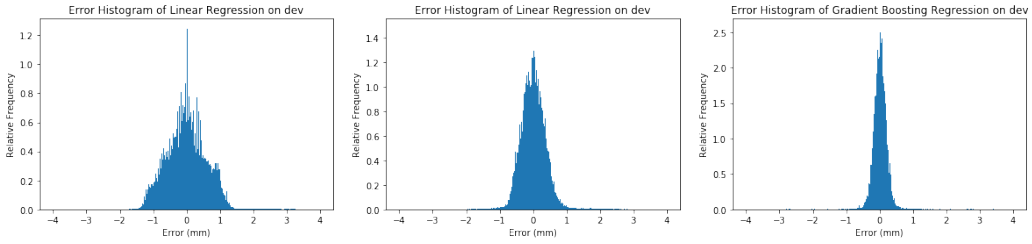

Figure 3: Histogram of dev errors from linear regression (left), ridge regression (middle), and gradient boosted regression (right).

4.1 Position Estimator Evaluation

Overall, all the position estimators we tested had lower RMSE than the baseline estimator except for the RBF Kernel SVR (Fig. 4). All models (except of course RBF Kernel SVR) had 95% of their predictions within 1.17mm of ground truth position (approximately the length between consecutive edges). However, only gradient boosted regression was able to achieve the same results as the imprecise Gaussian oracle (99.7% of data within 1 mm of ground truth). No estimator was able to achieve the results of the precise Gaussian oracle (99.7% of data within 0.5mm of ground truth). With our chosen hyperparameters, model overfitting (except for RBF Kernel SVR) was not an issue, as train and dev mean errors and RMSEs were close to each other.

From our results, we see that the standard linear models with linear features, namely linear regression and support vector regression with a linear kernel, can achieve reasonable results, even without applying feature engineering or boosting. However, to improve results past these standard techniques, we must be encode the non-linearities of the system into our feature set or use a non-linear model. This is reflected in the error histograms of linear regression, ridge regression with nonlinear features, and gradient boosted regression (Fig. 3): linear regression exhibits errors distributed in a non-Gaussian shape, while the other two models capture nonlinearities and exhibit errors distributed closer to a Gaussian distribution, suggesting that accounting for nonlinearities reduces model bias. The RBF Kernel SVR was unstable and difficult to scale to our training set size, producing results which were either inaccurate on train and dev or overfit to train, with a dev RMSE worse than our baseline.

Estimator	Train ME	Train RMSE	Dev ME	Dev RMSE
Precise Gaussian Oracle	N/A	N/A	0	0.167
Imprecise Gaussian Oracle	N/A	N/A	0	0.334
Naive Baseline	N/A	N/A	-0.216	0.751
Linear Regression	0	0.615	0.008	0.601
Ridge Regression	0	0.427	0.015	0.398
Support Vector Regression, Linear Kernel	-0.105	0.642	-0.068	0.603
Support Vector Regression, RBF Kernel	0.007	0.517	0.101	1.66
Gradient Boosting Regression	0	0.306	0.029	0.316

Figure 4: Performance of various position estimators. ME is an abbreviation for Mean Error. All errors are in units of millimeters.

4.2 Controller Evaluation

Controller	Mean Score (mm^2)	Final Position Error RMSE (mm)
Ground truth position, fast PD	-19571	1.85
Ground truth position, slow PD	-21882	1.17
Linearly regressed position, slow PD	-21951	1.45
Hysteresis (baseline)	-44974	10.9

Figure 5: Performance of various controllers, listed in descending order by score.

We then implemented our simple linear regression position estimator in a closed-loop controller (as in Fig. 2) with PD control and tuned control parameters to maximize score while maintaining a low edge-miscounting error rate on the optical sensor. We found that this controller scored the same as a controller with the same parameters but ground truth position instead of estimated positions and had only slightly worse final positioning accuracy. However, tuning a PD controller with ground truth positions without maintaining a low edge-miscounting error rate allowed significantly higher performance. These results together indicate that the limiting factor for position control in our current hardware prototype is from hardware design and not position estimation.

Finally, the unusually high final positioning errors with the PD controllers using ground truth positions indicate that controller parameters can be further tuned, particularly by adding an integral term for PID control.

4.3 Conclusion and Future Work

In this paper we explored the use of various regressors for estimating positions of a linear actuator with partially observable position. We found that modeling nonlinearities in the system improved estimator accuracy, and that estimators achieved acceptable position estimation accuracies. Models as simple as linear regression beat hand-crafted estimators by fitting on uncertainties in the system, and gradient boosted regression trees produced the best results by fitting on nonlinearities and interactions in sensor data. Further significant improvements in position estimator accuracy would likely require modeling of dynamic nonlinearities in the system, such as motor inductance; this would require stateful models, such as recurrent neural networks.

Results from our controller evaluation indicate that our next step is to improve hardware design for greater optical sensor edge counting robustness during high-speed actuation. This would contribute more to controller performance than improving position estimation with our current hardware prototype. However, the position estimation and control results reported here are sufficient to make our linear actuator usable in low-cost liquid-handling robotics. Thus, we have demonstrated the capability of regression methods to replace high-cost, high-fidelity hardware with low-cost, low-fidelity hardware.

5 Contributions

For this project, Ethan Li developed the linear actuator hardware and embedded control software and implemented the controller evaluation (http://github.com/ethanjli/linear-position-control). Ethan also investigated the linear regression, ridge regression, and gradient boosting methods.

Arjun Arora was responsible for using and debugging the Linear Kernel SVR and RBF Kernel SVR. He also provided Electrical Engineering guidance for PD controller implementation, tuning, and debugging.

All students (including Jay Whang (CS238)) ran and developed the Ipython notebook code and reported error metrics for their assigned position estimators (http://github.com/jaywhang/xtrm_learning). For CS 238, Jay and Ethan investigated neural network approaches to position estimation and partially observable control.

References

[1] Opentrons. `http://opentrons.com/`. Accessed: 2017-11-18.

[2] Siegmund Duell, Steffen Udluft, and Volkmar Sterzing. *Solving Partially Observable Reinforcement Learning Problems with Recurrent Neural Networks*, pages 709–733. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[3] Joshua Gafford, Finale Doshi-Velez, Robert Wood, and Conor Walsh. Machine learning approaches to environmental disturbance rejection in multi-axis optoelectronic force sensors. *Sensors and Actuators A: Physical*, 248:78–87, 2016.

[4] X. Wang. Non-linearity estimation and temperature compensation of capacitor pressure sensors using least square support vector regression. In *2008 IEEE International Symposium on Knowledge Acquisition and Modeling Workshop*, pages 1161–1164, Dec 2008.

[5] Jie-rong Guo, Yi-gang He, and Chang-qing Liu. Nonlinear correction of photoelectric displacement sensor based on least square support vector machine. *Journal of Central South University of Technology*, 18(5):1614, Oct 2011.

[6] Junqing Ma, Aiguo Song, and Jing Xiao. A robust static decoupling algorithm for 3-axis force sensors based on coupling error model and epsilon-svr. *Sensors*, 12(11):14537–14555, 2012.

[7] Sethu Vijayakumar and Stefan Schaal. Locally weighted projection regression: Incremental real time learning in high dimensional space. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 1079–1086, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[8] A. Chatterjee, S. Munshi, M. Dutta, and A. Rakshit. An artificial neural linearizer for capacitive humidity sensor. In *Proceedings of the 17th IEEE Instrumentation and Measurement Technology Conference [Cat. No. 00CH37066]*, volume 1, pages 313–317 vol.1, May 2000.

[9] H. Cao, Y. Yu, and Y. Ge. A research of multi-axis force sensor static decoupling method based on neural network. In *2009 IEEE International Conference on Automation and Logistics*, pages 875–879, Aug 2009.

[10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.