

# lab\_robot\_calib\_partial

September 26, 2021

## 1 Multiple Linear Regression for Robot Calibration

In this lab, we will illustrate the use of multiple linear regression for calibrating robot control. In addition to reviewing the concepts in the [multiple linear regression demo](#), you will see how to use multiple linear regression for time series data – an important concept in dynamical systems such as robotics.

The robot data for the lab is taken generously from the TU Dortmund's [Multiple Link Robot Arms Project](#). As part of the project, they have created an excellent public dataset: [MERIt](#) – A Multi-Elastic-Link Robot Identification Dataset that can be used for understanding robot dynamics. The data is from a three link robot:

We will focus on predicting the current draw into one of the joints as a function of the robot motion. Such models are essential in predicting the overall robot power consumption. Several other models could also be used.

### 1.1 Load and Visualize the Data

First, import the modules we will need.

```
[1]: import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
```

The full MERIt dataset can be obtained from the [MERIt site](#). But, this dataset is large. Included in this repository are two of the ten experiments. Each experiment corresponds to 80 seconds of recorded motion. We will use the following files: \* [exp1.csv](#) for training \* [exp2.csv](#) for test

If you are running this notebook on Google colab, you will need to run the following commands to load the files onto your local machine. Otherwise, if you have clone the repository, the files should be in the directory as the notebook and you can skip this step.

```
[2]: import os
from six.moves import urllib

for fn_dst in ['exp1.csv', 'exp2.csv']:
    fn_src = 'https://raw.githubusercontent.com/sdrangan/introml/master/
    ↳unit03_mult_lin_reg/%s' % fn_dst
```

```

if os.path.isfile(fn_dst):
    print('File %s is already downloaded' % fn_dst)
else:
    print('Downloaded %s' % fn_dst)
    urllib.request.urlretrieve(fn_src, fn_dst)

```

File exp1.csv is already downloaded

File exp2.csv is already downloaded

Below, I have supplied the column headers in the `names` array. Use the `pd.read_csv` command to load the training data in `exp1.csv`. Use the `index_col` option to specify that column 0 (the one with time) is the *index* column. You can review [simple linear regression demo](#) for examples of using the `pd.read_csv` command.

```

[3]: names = [
    't',                                # Time (secs)
    'q1', 'q2', 'q3',                  # Joint angle (rads)
    'dq1', 'dq2', 'dq3',               # Joint velocity (rads/sec)
    'I1', 'I2', 'I3',                  # Motor current (A)
    'eps21', 'eps22', 'eps31', 'eps32', # Strain gauge measurements ( $\mu\text{m} / \text{m}$ )
    'ddq1', 'ddq2', 'ddq3'             # Joint accelerations (rad/sec2)
]
# TODO
df = pd.read_csv('exp1.csv', index_col = 0, names = names, na_values = '?')

```

$\mu\text{m} / \text{m}$

Print the first six lines of the pandas dataframe and manually check that they match the first rows of the csv file.

```

[4]: # TODO
df.head(6)

```

```

[4]:      q1      q2      q3      dq1      dq2      dq3  \
t
0.00 -0.000007  2.4958 -1.1345 -7.882100e-21 -4.940656e-321  3.913100e-29
0.01 -0.000007  2.4958 -1.1345 -2.258200e-21 -4.940656e-321  2.626200e-31
0.02 -0.000007  2.4958 -1.1345 -6.469800e-22 -4.940656e-321  1.762500e-33
0.03 -0.000007  2.4958 -1.1345 -1.853600e-22 -4.940656e-321  1.182800e-35
0.04 -0.000007  2.4958 -1.1345 -5.310600e-23 -4.940656e-321 -5.270900e-03
0.05 -0.000007  2.4958 -1.1345 -1.521500e-23 -4.940656e-321  3.252600e-04

      I1      I2      I3  eps21  eps22  eps31  eps32  \
t
0.00 -0.081623 -0.40812 -0.30609 -269.25 -113.20  3.5918  1.57860
0.01 -0.037411 -0.37241 -0.26698 -270.91 -116.05  1.4585 -1.73980
0.02 -0.066319 -0.40302 -0.31459 -269.25 -112.97  3.5918  0.86753

```

```

0.03 -0.068020 -0.43703 -0.28398 -269.97 -114.39 1.6956 -0.08059
0.04 -0.052715 -0.40472 -0.30779 -269.97 -114.15 3.1177 0.86753
0.05 -0.088425 -0.42342 -0.29589 -269.25 -114.15 2.4066 -0.08059

```

```

          ddq1          ddq2          ddq3
t
0.00 -9.904900e-19 -6.210306e-319  4.917400e-27
0.01  4.248100e-19 -1.766878e-319 -1.381100e-27
0.02  3.233800e-19 -4.990557e-320 -4.117300e-28
0.03  1.500500e-19 -1.394253e-320 -1.173100e-28
0.04  5.932400e-20 -3.581976e-321 -3.770800e-01
0.05  2.164600e-20 -1.141292e-321  2.930300e-01

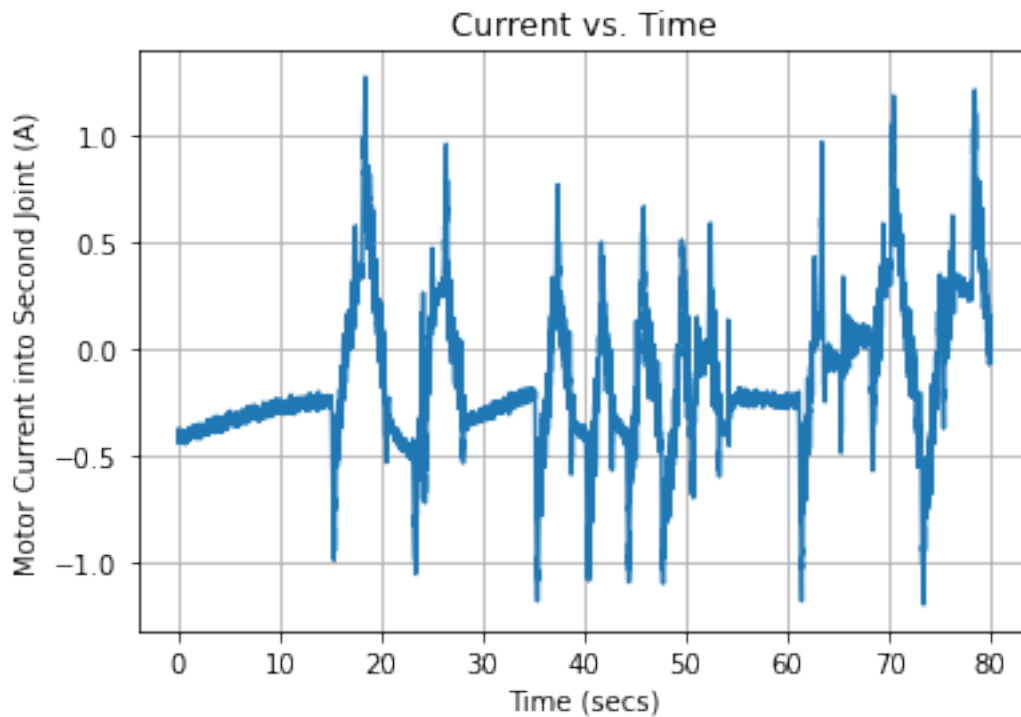
```

From the dataframe `df`, extract the time indices into a vector `t` and extract `I2`, the current into the second joint. Place the current in a vector `y` and plot `y` vs. `t`. Label the axes with the units.

```

[5]: # TODO
t = df.index
y = df['I2']
plt.plot(t,y)
plt.xlabel('Time (secs)')
plt.ylabel('Motor Current into Second Joint (A)')
plt.title('Current vs. Time')
plt.grid()

```



Use all the samples from the experiment 1 dataset to create the training data: \* `ytrain`: A vector of all the samples from the `I2` column \* `Xtrain`: A matrix of the data with the columns: `['q2','dq2','eps21','eps22','eps31','eps32','ddq2']`

```
[6]: # TODO
ytrain = df['I2']
Xtrain = df[['q2','dq2','eps21','eps22','eps31','eps32','ddq2']]
```

## 1.2 Fit a Linear Model

Use the `sklearn.linear_model` module to create a `LinearRegression` class `regr`.

```
[7]: from sklearn import linear_model

# Create linear regression object
# TODO
regr = linear_model.LinearRegression()
```

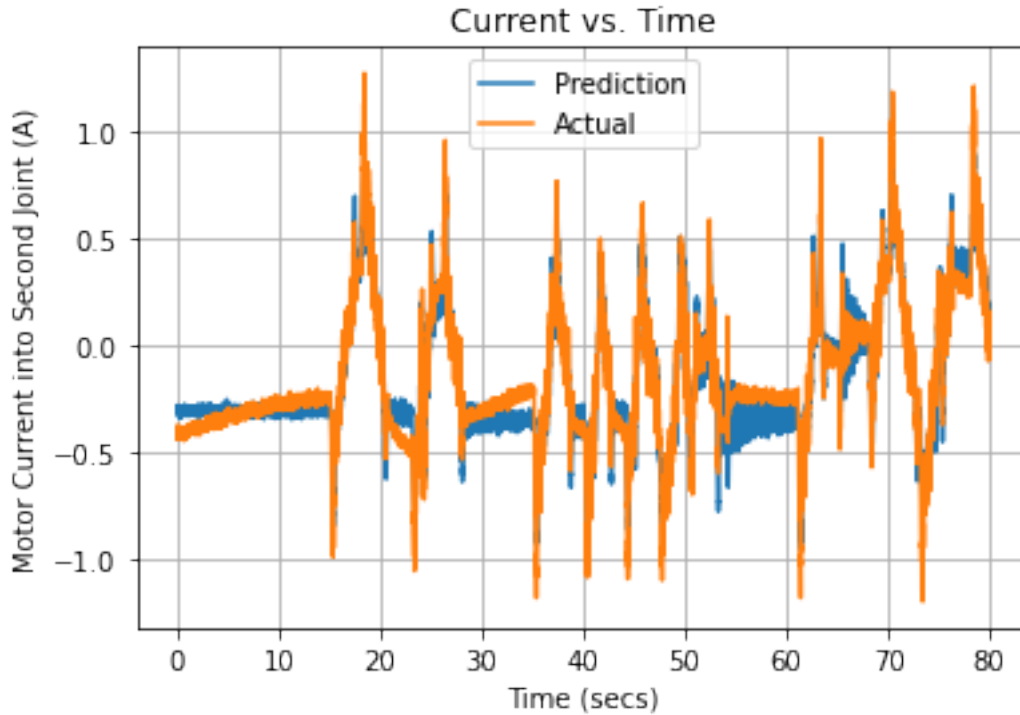
Train the model on the training data.

```
[8]: # TODO
regr.fit(Xtrain,ytrain)
```

```
[8]: LinearRegression()
```

Using the trained model, compute, `ytrain_pred`, the predicted current. Plot `ytrain_pred` vs. time `t`. On the same plot, plot the actual current `ytrain` vs. time `t`. Create a legend for the plot.

```
[9]: # TODO
ytrain_pred = regr.predict(Xtrain)
plt.plot(t,ytrain_pred,label = 'Prediction')
plt.plot(t,ytrain,label = 'Actual')
plt.legend()
plt.xlabel('Time (secs)')
plt.ylabel('Motor Current into Second Joint (A)')
plt.title('Current vs. Time')
plt.grid()
```



Measure the normalized RSS given by  $\text{RSS} / (n \hat{s}_y^2)$ .

```
[10]: # TODO
RSS_train = np.mean((ytrain_pred-ytrain)**2)/(np.std(ytrain)**2)
RSS_train
```

[10]: 0.09583263861233143

### 1.3 Measure the Fit on an Independent Dataset

Up to now, we have only tested the model on the same data on which it was trained. In general, we need to test model on independent data not used in the training. For this purpose, load the data in `exp2.csv`. Compute the regression predicted values on this data and plot the predicted and actual values over time.

```
[11]: # TODO
names = [
    't',                                # Time (secs)
    'q1', 'q2', 'q3',                  # Joint angle (rads)
    'dq1', 'dq2', 'dq3',               # Joint velocity (rads/sec)
    'I1', 'I2', 'I3',                  # Motor current (A)
    'eps21', 'eps22', 'eps31', 'eps32', # Strain gauge measurements ( $\mu\text{m} / \text{m}$ )
    'ddq1', 'ddq2', 'ddq3'             # Joint accelerations ( $\text{rad}/\text{sec}^2$ )
]
```

```
]
# TODO
df2 = pd.read_csv('exp2.csv', index_col = 0, names = names, na_values = '?')
df2.head(6)
```

```
[11]:
```

	q1	q2	q3	dq1	dq2	dq3	I1 \
t							
0.00	-0.000007	1.9024	0.26063	-0.000364	4.940656e-321	0.012596	-0.096928
0.01	0.000013	1.9024	0.26073	0.000739	4.940656e-321	0.012095	-0.028908
0.02	-0.000007	1.9024	0.26086	-0.000580	4.940656e-321	0.011596	-0.059517
0.03	0.000013	1.9024	0.26099	0.001409	4.940656e-321	0.013933	-0.079923
0.04	-0.000007	1.9024	0.26110	-0.001273	4.940656e-321	0.010793	-0.025507
0.05	-0.000007	1.9024	0.26124	0.001928	4.940656e-321	0.011915	-0.083324

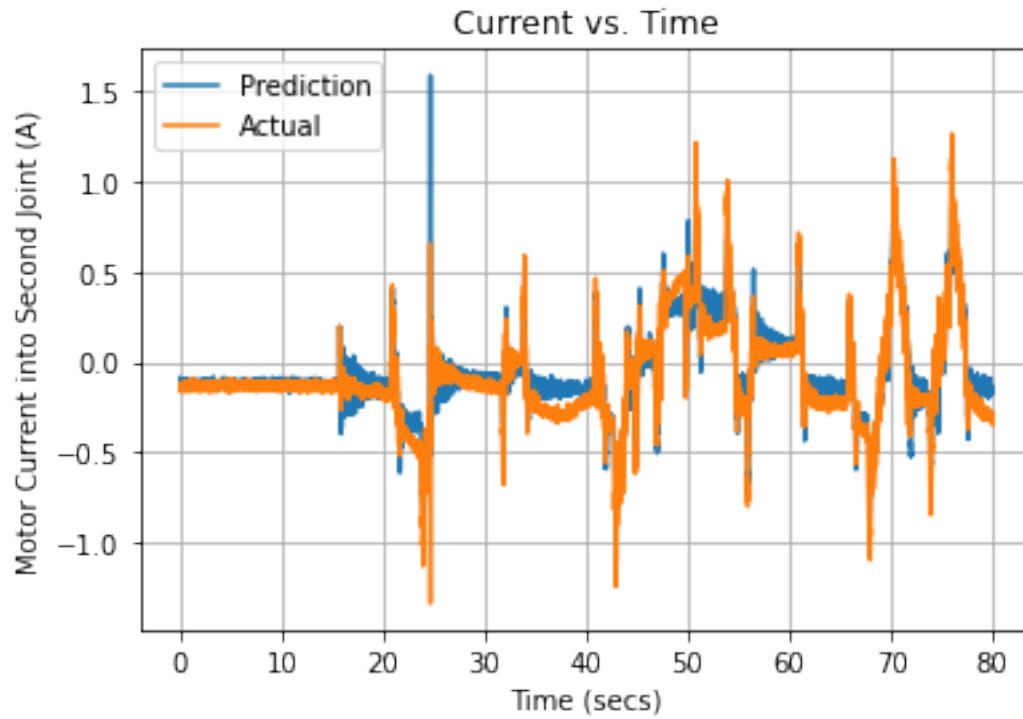
	I2	I3	eps21	eps22	eps31	eps32	ddq1 \
t							
0.00	-0.15134	-0.017005	-130.83	-41.856	-6.3635	5.13410	-0.045712
0.01	-0.11903	-0.020406	-138.18	-51.100	-14.6590	-5.05820	0.125580
0.02	-0.13944	-0.047614	-139.36	-51.812	-14.6590	-5.29520	-0.130080
0.03	-0.15304	-0.023807	-135.57	-48.019	-11.3410	-0.79168	0.213010
0.04	-0.12924	-0.006802	-135.81	-49.204	-12.0520	-2.21390	-0.276490
0.05	-0.14964	-0.034010	-139.60	-53.471	-16.0820	-6.95450	0.323560

	ddq2	ddq3
t		
0.00	6.210306e-319	1.582900
0.01	1.766878e-319	0.414660
0.02	4.990557e-320	0.082286
0.03	1.394253e-320	0.190650
0.04	3.581976e-321	-0.170400
0.05	1.141292e-321	0.031745

```
[12]: ytest = df2['I2']
Xtest = df2[['q2','dq2','eps21', 'eps22', 'eps31', 'eps32','ddq2']]
```

```
[13]: ytest_pred = regr.predict(Xtest)
plt.plot(t,ytest_pred,label = 'Prediction')
plt.plot(t,ytest,label = 'Actual')
plt.legend()
plt.xlabel('Time (secs)')
plt.ylabel('Motor Current into Second Joint (A)')
plt.title('Current vs. Time')
plt.grid()
```



Measure the normalized RSS on the test data.

```
[14]: # TODO
      RSS_test = np.mean((ytest_pred-ytest)**2)/(np.std(ytest)**2)
      RSS_test
```

```
[14]: 0.12678048804762676
```

```
[ ]:
```