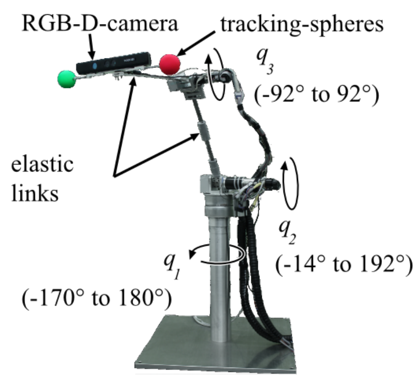
Multiple Linear Regression for Robot Calibration

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 (./glucose.ipynb)](./glucose.ipynb), 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 (http://www.rst.e-technik.tu-dortmund.de/cms/en/research/robotics/TUDOR_engl/index.html)](http://www.rst.e-technik.tu-dortmund.de/cms/en/research/robotics/TUDOR_engl/index.html). As part of the project, they have created an excellent public dataset: [MERIt (http://www.rst.e-technik.tu-dortmund.de/cms/en/research/robotics/TUDOR_engl/index.html#h3MERIt)](http://www.rst.e-technik.tu-dortmund.de/cms/en/research/robotics/TUDOR_engl/index.html#h3MERIt) -- 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.

Load and Visualize the Data

First, import the modules we will need.

In [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 (http://www.rst.e-technik.tu-dortmund.de/cms/en/research/robotics/TUDOR_engl/index.html#h3MERIt)](http://www.rst.e-technik.tu-dortmund.de/cms/en/research/robotics/TUDOR_engl/index.html#h3MERIt). But, this dataset is large. Included in this repository are two of the ten experiments. Each experiments corresonds to 80 seconds of recorded motion. We will use the following files:

- [exp1.csv (./exp1.csv)](./exp1.csv) for training

- exp2.csv (./exp2.csv) for test

Below, I have supplied the column headers in the `names` array. Use the `pd.read_csv` command to load the data. 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 (../simp_lin_reg/auto_mpg.ipynb) for examples of using the `pd.read_csv` command.

In [11]:
```
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 ($\mu$m /m
    'ddq1', 'ddq2', 'ddq3'                  # Joint accelerations (rad/sec^2)
]
# TODO
df = pd.read_csv('exp1.csv', names=names)
```

Print the first six lines of the pandas dataframe and manually check that they match the first rows of the csv file. The picture following is the first 6 rows read from .csv file.

1	0	-6.71E-06	2.4958	-1.1345	-7.88E-21	-4.9407e-3	3.91E-29	-0.08162	-0.40812	-0.30609	-269.25	-113.2	3.5918	1.5786	-9.90E-19	-6.2103e-3	4.92E-27
2	0.01	-6.71E-06	2.4958	-1.1345	-2.26E-21	-4.9407e-3	2.63E-31	-0.03741	-0.37241	-0.26698	-270.91	-116.05	1.4585	-1.7398	4.25E-19	-1.7669e-3	-1.38E-27
3	0.02	-6.71E-06	2.4958	-1.1345	-6.47E-22	-4.9407e-3	1.76E-33	-0.06632	-0.40302	-0.31459	-269.25	-112.97	3.5918	0.86753	3.23E-19	-4.9906e-3	-4.12E-28
4	0.03	-6.71E-06	2.4958	-1.1345	-1.85E-22	-4.9407e-3	1.18E-35	-0.06802	-0.43703	-0.28398	-269.97	-114.39	1.6956	-0.08059	1.50E-19	-1.3943e-3	-1.17E-28
5	0.04	-6.71E-06	2.4958	-1.1345	-5.31E-23	-4.9407e-3	-0.00527	-0.05272	-0.40472	-0.30779	-269.97	-114.15	3.1177	0.86753	5.93E-20	-3.582e-32	-0.37708
6	0.05	-6.71E-06	2.4958	-1.1345	-1.52E-23	-4.9407e-3	0.000325	-0.08843	-0.42342	-0.29589	-269.25	-114.15	2.4066	-0.08059	2.16E-20	-1.1413e-3	0.29303

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`.

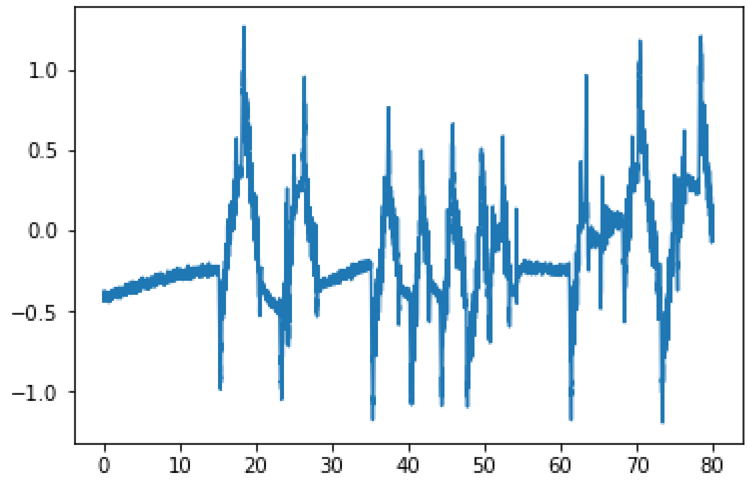
In [18]:
```
# TODO
df.head(6)
```

Out[18]:

	t	q1	q2	q3	dq1	dq2	dq3	I1	I2	
0	0.00	-0.000007	2.4958	-1.1345	-7.882100e-21	-4.940656e-321	3.913100e-29	-0.081623	-0.40812	-0.306
1	0.01	-0.000007	2.4958	-1.1345	-2.258200e-21	-4.940656e-321	2.626200e-31	-0.037411	-0.37241	-0.266
2	0.02	-0.000007	2.4958	-1.1345	-6.469800e-22	-4.940656e-321	1.762500e-33	-0.066319	-0.40302	-0.314
3	0.03	-0.000007	2.4958	-1.1345	-1.853600e-22	-4.940656e-321	1.182800e-35	-0.068020	-0.43703	-0.283
4	0.04	-0.000007	2.4958	-1.1345	-5.310600e-23	-4.940656e-321	-5.270900e-03	-0.052715	-0.40472	-0.307
5	0.05	-0.000007	2.4958	-1.1345	-1.521500e-23	-4.940656e-321	3.252600e-04	-0.088425	-0.42342	-0.295

In [16]:
```
# TODO
y = df['I2']
t = df['t']
plt.plot(t,y)
```

Out[16]: `[<matplotlib.lines.Line2D at 0x7fecd70e3d30>]`



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']`

In [21]:
```
# TODO
# ytrain = ...
ytrain = y
# Xtrain = ...
Xtrain = df[['q2','dq2','eps21', 'eps22', 'eps31', 'eps32','ddq2']]
```

Fit a Linear Model

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

In [22]:
```
from sklearn import linear_model

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

Train the model on the training data using the `regr.fit(...)` method.

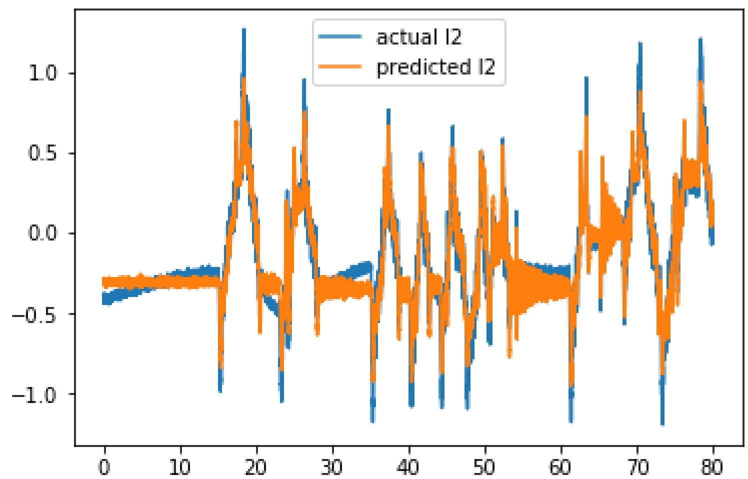
In [24]:
```
# TODO
regr.fit(Xtrain,ytrain)
```

Out[24]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

Plot the predicted and actual current `I2` over time on the same plot. Create a legend for the plot.

In [30]:
```
# TODO
y_tr_pred = regr.predict(Xtrain)
plt.plot(t,y)
plt.plot(t,y_tr_pred)
plt.legend(['actual I2', 'predicted I2'])
```

Out[30]:
```
<matplotlib.legend.Legend at 0x7fecc098dd30>
```



Measure the normalized RSS given by

$$\frac{RSS}{ns_y^2}.$$

In [33]:
```
# TODO
# RSS_train = ...
RSS_train = np.mean((y_tr_pred-ytrain)**2)/(np.std(ytrain)**2)
print("RSS per sample = {0:f}".format(RSS_train))
```

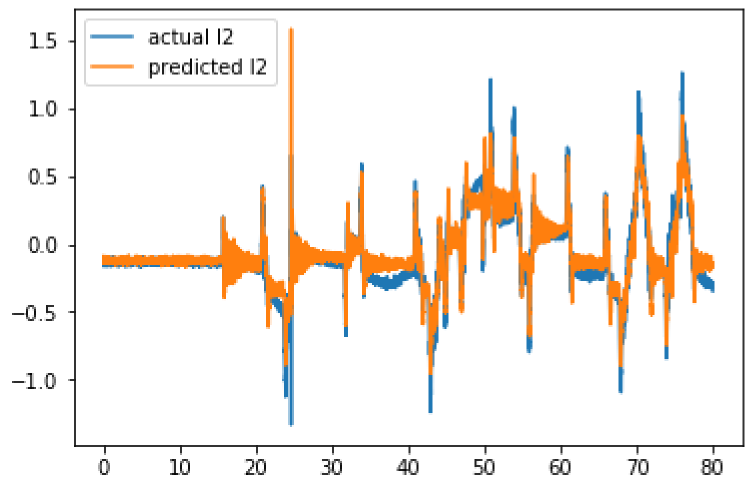
```
RSS per sample = 0.095833
```

Measure the Fit on an Indepdent Dataset

Load the data in `exp2.csv`. Compute the regression predicted values on this data and plot the predicted and actual values over time.

In [36]:
```
# TODO
df2 = pd.read_csv('exp2.csv', names=names)
t2 = df2['t']
ytrain2 =  df2['I2']
Xtrain2 = df2[['q2','dq2','eps21', 'eps22', 'eps31', 'eps32','ddq2']]
y_tr_pred2 = regr.predict(Xtrain2)
plt.plot(t2,ytrain2)
plt.plot(t2,y_tr_pred2)
plt.legend(['actual I2', 'predicted I2'])
```

Out[36]: `<matplotlib.legend.Legend at 0x7fecc081e908>`



Measure the normalized RSS on the test data. Is it substantially higher than the training data?

In [37]:
```
# TODO
RSS_train2 = np.mean((y_tr_pred2-ytrain2)**2)/(np.std(ytrain2)**2)
print("RSS per sample = {0:f}".format(RSS_train2))
```

```
RSS per sample = 0.126780
```

In [ ]:
```
Yes, it is substantially higher than the training data.
```