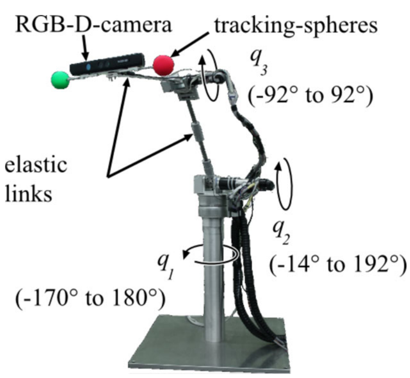
Multiple Linear Regression for Robot Calibration

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 TUDOR project, TU Dortmund Omni-Elastic Robot. 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.

Load and Visualize the Data

First, import the modules we will need.

In [3]:
```
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
```

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 experiments corresonds 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.

In [4]:
```
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

    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.

In [5]:
```
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(...)
df = pd.read_csv('exp1.csv', header=None,sep=',',names=names, index_col=0)
```

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

In [3]:
```
# TODO
df.head(6)
```

Out[3]:

	q1	q2	q3	dq1	dq2	dq3	I1	I2	I3	ep
t										
0.00	-0.000007	2.4958	-1.1345	-7.882100e-21	-4.940656e-321	3.913100e-29	-0.081623	-0.40812	-0.30609	-26
0.01	-0.000007	2.4958	-1.1345	-2.258200e-21	-4.940656e-321	2.626200e-31	-0.037411	-0.37241	-0.26698	-27
0.02	-0.000007	2.4958	-1.1345	-6.469800e-22	-4.940656e-321	1.762500e-33	-0.066319	-0.40302	-0.31459	-26
0.03	-0.000007	2.4958	-1.1345	-1.853600e-22	-4.940656e-321	1.182800e-35	-0.068020	-0.43703	-0.28398	-26
0.04	-0.000007	2.4958	-1.1345	-5.310600e-23	-4.940656e-321	-5.270900e-03	-0.052715	-0.40472	-0.30779	-26

	q1	q2	q3	dq1	dq2	dq3	I1	I2	I3	ep
t										
0.05	-0.000007	2.4958	-1.1345	-1.521500e-23	-4.940656e-321	3.252600e-04	-0.088425	-0.42342	-0.29589	-26

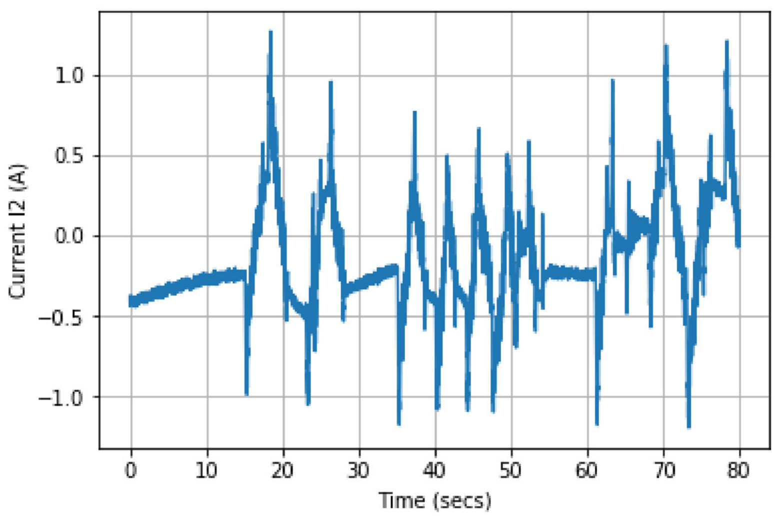
From the dataframe `df`, extract the time indices into a vector `t` and extract `I2`, the current into the second joint. Place the current in a vector `y` and plot `y` vs. `t`. Label the axes with the units.

In [4]:

```
# TODO
# y = ...
# t = ...
# plt.plot(...)
y = np.array(df['I2'])
t = np.array(df.index)
plt.plot(t,y)
plt.grid()
plt.xlabel('Time (secs)')
plt.ylabel('Current I2 (A)')
```

Out[4]:

```
Text(0,0.5,'Current I2 (A)')
```



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 [5]:

```
# TODO
# ytrain = ...
# Xtrain = ...
ytrain = np.array(df['I2'])
Xtrain = np.array(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 [6]:
```
from sklearn import linear_model

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

Train the model on the training data.

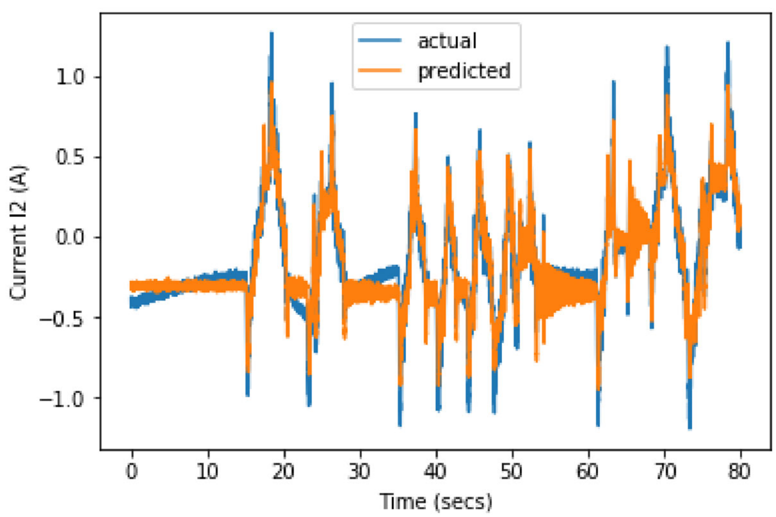
In [7]:
```
# TODO
regr.fit(Xtrain, ytrain)
```

Out[7]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

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.

In [8]:
```
# TODO
ytrain_pred = regr.predict(Xtrain)
plt.plot(t,ytrain)
plt.plot(t,ytrain_pred)
plt.legend(['actual', 'predicted'])
plt.xlabel('Time (secs)')
plt.ylabel('Current I2 (A)')
```

Out[8]:
```
Text(0,0.5,'Current I2 (A)')
```



Measure the normalized RSS given by

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

In [9]:
```
# TODO
RSS_train = np.mean((ytrain-ytrain_pred)**2) / np.mean((ytrain-np.mean(ytrain))**2)
```

```
RSS_train
```

Out[9]: 0.09583263861233197

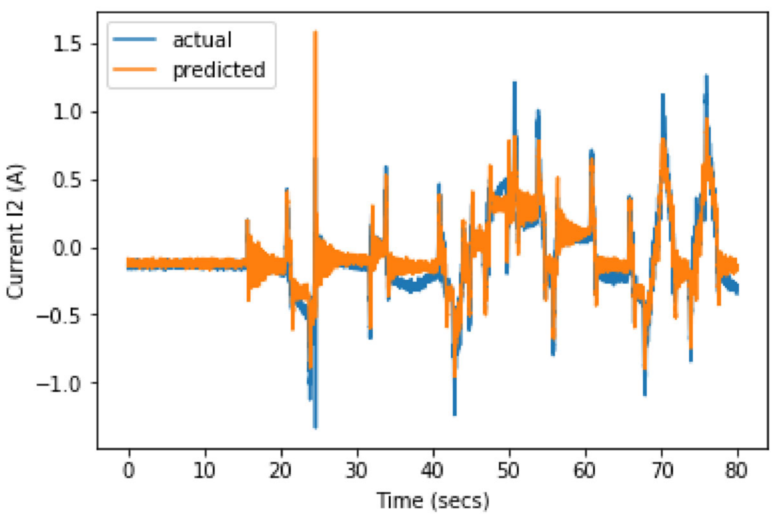
Measure the Fit on an Indepdent Dataset

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.

In [10]:

```
# TODO
df = pd.read_csv('exp2.csv', header=None,sep=',',names=names, index_col=0)
ytest = np.array(df['I2'])
Xtest = np.array(df[['q2','dq2','eps21', 'eps22', 'eps31', 'eps32','ddq2']])
ttest = np.array(df.index)
ytest_pred = regr.predict(Xtest)
plt.plot(t,ytest)
plt.plot(t,ytest_pred)
plt.legend(['actual', 'predicted'])
plt.xlabel('Time (secs)')
plt.ylabel('Current I2 (A)')
```

Out[10]: Text(0,0.5,'Current I2 (A)')



Measure the normalized RSS on the test data.

In [11]:

```
# TODO
RSS_test = np.mean((ytest-ytest_pred)**2) / np.mean((ytest-np.mean(ytest))**2)
RSS_test
```

Out[11]: 0.1267804880476244

In [ ]: