First I load in the data from a run.

```
In [1]: import bicycledataprocessor as bdp

        VIRTUAL_ENV -> /home/moorepants/bicycle-env/lib/python2.7/site-packages
```

```
In [2]: dataset = bdp.DataSet()
```

```
In [3]: trial = bdp.Run(311, dataset, forceRecalc=True)

        Initializing the run object.
        Loading metadata from the database.
        Loading the raw signals from the database.
        Loading the bicycle and rider data for Charlie on Rigid Rider
        We have foundeth a directory named: /media/Data/Documents/School/UC Davis/Bicycle
        Mechanics/BicycleParameters/data/bicycles/Rigidcl.
        Found the RawData directory: /media/Data/Documents/School/UC Davis/Bicycle
        Mechanics/BicycleParameters/data/bicycles/Rigidcl/RawData
        Looks like you've already got some parameters for Rigidcl, use forceRawCalc to recalculate.
        There is no rider on the bicycle, now adding Charlie.
        Loaded the precalculated parameters from /media/Data/Documents/School/UC Davis/Bicycle
        Mechanics/BicycleParameters/data/riders/Charlie/Parameters/CharlieRigidclBenchmark.txt
        Computing signals from raw data.
        The minimun of the error landscape is 0.312625 and the provided guess is 0.310000
        Optimization terminated successfully.
                 Current function value: 2.702983
                 Iterations: 11
                 Function evaluations: 22
        Optimization terminated successfully.
                 Current function value: 2.703001
                 Iterations: 11
                 Function evaluations: 22
        This is what came out of the minimization: [ 0.31220996]
        Extracting the task portion from the data.
        ===============================================================================
        Run # 311
        Environment: Horse Treadmill
        Rider: Charlie
        Bicycle: Rigid Rider
        Speed:7.0
        Maneuver: Balance With Disturbance
        Notes:
        ===============================================================================
```

Now I will pick off the roll angle measurement, ϕ, and the roll rate measurement, ω. The task signals store them as radians and radians per second, respectively. Notice that there may be a slight bias to the roll angle (0.6 deg).

```
In [4]: phi = trial.taskSignals['RollAngle'].convert_units('degree')
```

```
In [5]: phi.mean()

Out[5]: 0.58953246049759434
```

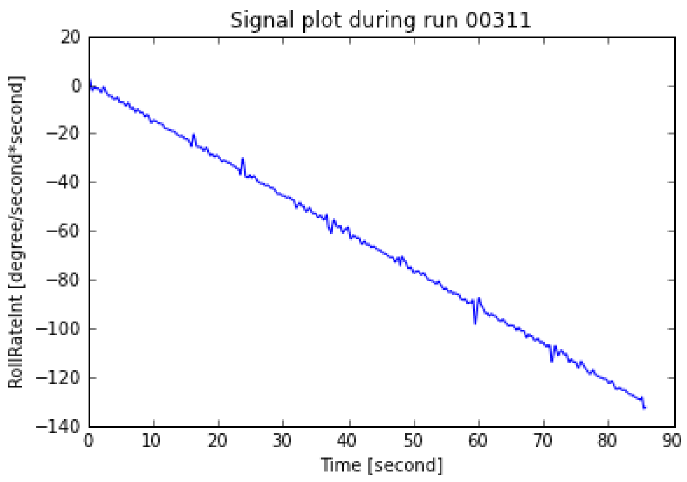
```
In [6]: omega = trial.taskSignals['RollRate'].convert_units('degree/second')
```

```
In [7]: time = phi.time()
```

Numerically integrate the rate gyro signal using the initial condition from our roll angle measurement. Notice the very linear drift from the rate gyro measurements. The linear trend can be subtracted to try and compensate for the drift.

```
In [8]: omegaInt = omega.integrate(initialCondition=phi[0])
```

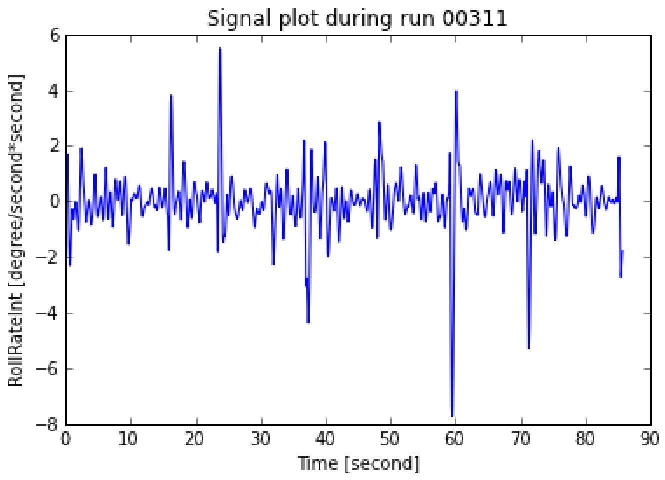
```
In [9]: omegaInt.plot()
In [9]: omegaInt.plot()
```



```
Out[9]: [<matplotlib.lines.Line2D at 0x47b8990>]
```

```
In [10]: omegaInt = omega.integrate(initialCondition=phi[0], detrend=True)
```

```
In [11]: omegaInt.plot()
```



```
Out[11]: [<matplotlib.lines.Line2D at 0x7f4b144f5dd0>]
```

Now, lets build a simple complementary filter to get an estimate of the roll angle, $\hat{\phi}$. Notice that I do not detrend the integral of roll rate for the filter.

```
In [12]: from scipy.signal import bilinear, lfilter
```

```
In [13]: def complementary(phi, omega, br):
             """Returns and estimate of the angle.

             Parameters
             ----------
             phi : array_like, shape(n,)
                 An angle measurement.
             omega : array_like, shape(n,)
                 An rate measurement of phi.
             br : float
                 The filter break frequency.

             Returns
             -------
```

```
    phiHat : ndarray, shape(n,)
        The estimate of the angle.
    """
    low = bilinear([br], [1, br], 200)
    high = bilinear([1, 0], [1, br], 200)
    phiHat = lfilter(low[0], low[1], phi) + lfilter(high[0], high[1], omega.integrate())
    return phiHat
```

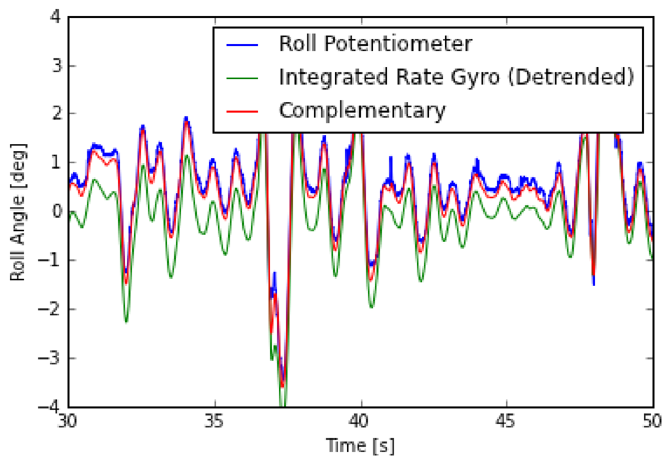
In [14]:
```
br = 10.
```

In [15]:
```
phiHat = complementary(phi, omega, br)
```

The plot shows the measured roll angle from the trailer potentiometer, the integrated and detrended rate gyro signal, and the result of the complementary filter.

In [16]:
```
plot(time, phi, time, omegaInt, time, phiHat)
legend(('Roll Potentiometer', 'Integrated Rate Gyro (Detrended)', 'Complementary'))
xlim((30, 50))
ylim((-4, 4))
xlabel('Time [s]')
ylabel('Roll Angle [deg]')
```

Out[16]:
```
<matplotlib.text.Text at 0x7f4b14509710>
```



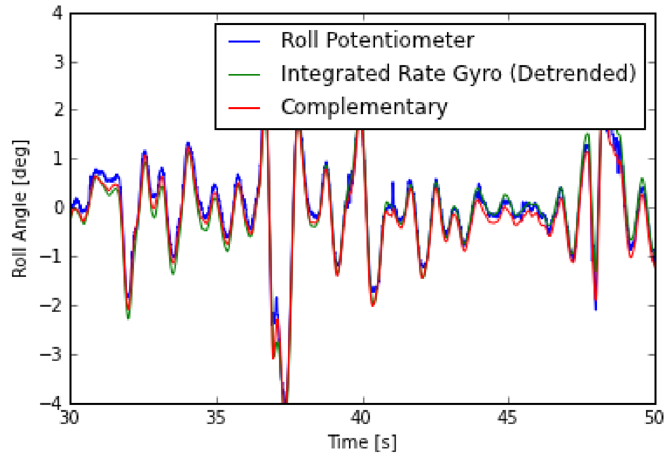
The roll angle measurement typically has a small bias due to my less than accurate calibration techniques. Notice what happens if I subtract the mean from the roll angle signal. Now all three match up. So the previous complementary filter the roll angle bias plays a role.

In [17]:
```
phi = trial.taskSignals['RollAngle'].convert_units('degree').subtract_mean()
```

In [18]:
```
phiHat = complementary(phi, omega, br)
```

In [19]:
```
plot(time, phi, time, omegaInt, time, phiHat)
legend(('Roll Potentiometer', 'Integrated Rate Gyro (Detrended)', 'Complementary'))
xlim((30, 50))
ylim((-4, 4))
xlabel('Time [s]')
ylabel('Roll Angle [deg]')
```

Out[19]:
```
<matplotlib.text.Text at 0x5286790>
```

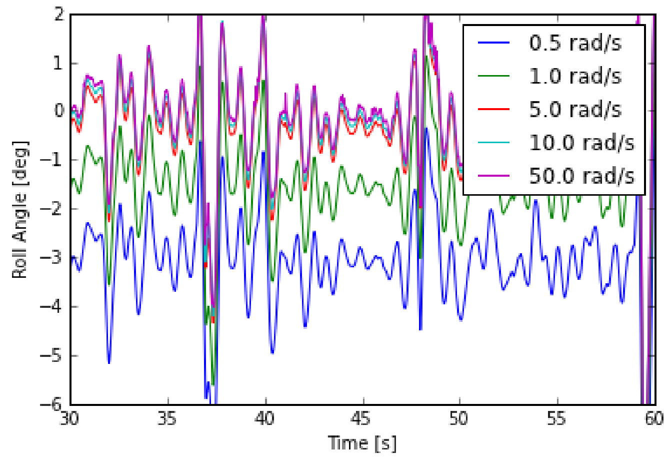


I'm curious how the break frequency affects the estimate. So this plot gives the results of several break frequencies.

```
In [20]: frequencies = [0.5, 1.0, 5.0, 10.0, 50.0]
```

```
In [21]: for f in frequencies:
             phiHat = complementary(phi, omega, f)
             plot(time, phiHat, label=str(f) + ' rad/s')
         legend()
         xlim((30, 60))
         ylim((-6, 2))
         xlabel('Time [s]')
         ylabel('Roll Angle [deg]')
```

```
Out[21]: <matplotlib.text.Text at 0x7f4b1494c950>
```



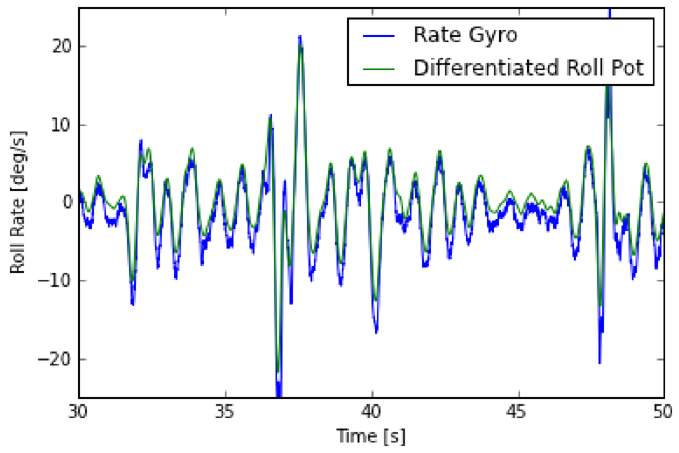
Numerically differentiate, ϕ, to get $\dot{\phi}$. This increases the noise to low pass filter it with a 2nd order Butterworth filter at 10 hz.

```
In [22]: phiDot = phi.time_derivative().filter(10.)
```

Differentiating the roll angle measurement seems to give similar result as the rate gyro measurement.

```
In [23]: plot(time, omega, time, phiDot)
         legend(('Rate Gyro', 'Differentiated Roll Pot'))
         ylim((-25, 25))
         xlim((30, 50))
         xlabel('Time [s]')
         ylabel('Roll Rate [deg/s]')
```

```
Out[23]: <matplotlib.text.Text at 0x7f4b14c498d0>
```



Now let's see how the angular acceleration estimates look from the two measured signals.

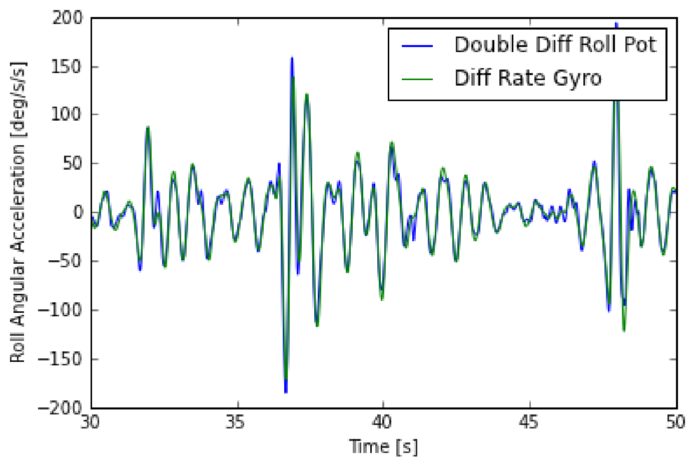
```
In [24]: phiDDot = phi.filter(10.).time_derivative().time_derivative()
```

```
In [25]: omegaDot = omega.time_derivative().filter(10.)
```

Each signal also gives similar results.

```
In [26]: plot(time, phiDDot, time, omegaDot)
         ylim((-200, 200))
         xlim((30, 50))
         legend(('Double Diff Roll Pot', 'Diff Rate Gyro'))
         xlabel('Time [s]')
         ylabel('Roll Angular Acceleration [deg/s/s]')
```

```
Out[26]: <matplotlib.text.Text at 0x7f4b151d7c10>
```

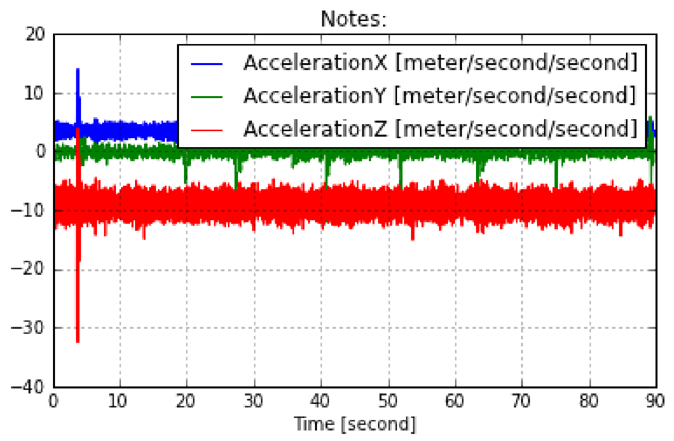


Roll Estimate From Accelerometer During A Run

I'm curious what kind of estimation the accelerometers give for the roll angle. Let's first check the acceleration readings from the VectorNav.

```
In [27]: fig = trial.plot('AccelerationX', 'AccelerationY', 'AccelerationZ', signalType='truncated')
```

Run: 00311, Rider: Charlie, Speed: 7.0m/s
Maneuver: Balance With Disturbance, Environment: Horse Treadmill



This plot seems reasonable. The gravity vector is primarily in the xz-plane. The bump is clearly seen at the beginning and the lateral perturbations are seen in the lateral acceleration signal.

Now let's store the low pass filtered acceleration components and find the magnitude of the gravity vector throughout the run.

```
In [28]: phi = trial.truncatedSignals['RollAngle'].subtract_mean() # in degrees
```

```
In [29]: time = phi.time()
```

```
In [30]: ax = trial.truncatedSignals['AccelerationX'].filter(10.)
```

```
In [31]: ay = trial.truncatedSignals['AccelerationY'].filter(10.)
```

```
In [32]: az = trial.truncatedSignals['AccelerationZ'].filter(10.)
```

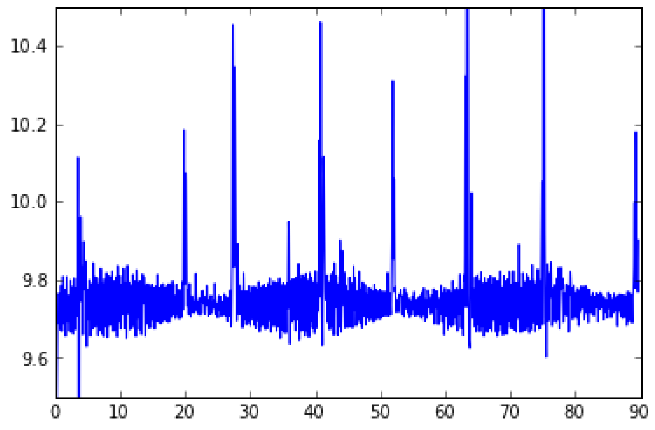
```
In [33]: import numpy as np
```

```
In [34]: g = np.array([ax, ay, az])
```

```
In [35]: magG = np.zeros(len(ay))
         for i, vec in enumerate(g.T):
             magG[i] = np.sqrt(vec[0]**2 + vec[1]**2 + vec[2]**2)
```

```
In [36]: plot(time, magG)
         ylim((9.5, 10.5))
```

```
Out[36]: (9.5, 10.5)
```



```
In [37]: magG.mean()

Out[37]: 9.7583543245740998

In [38]: magG.std()

Out[38]: 0.14386654741813623
```

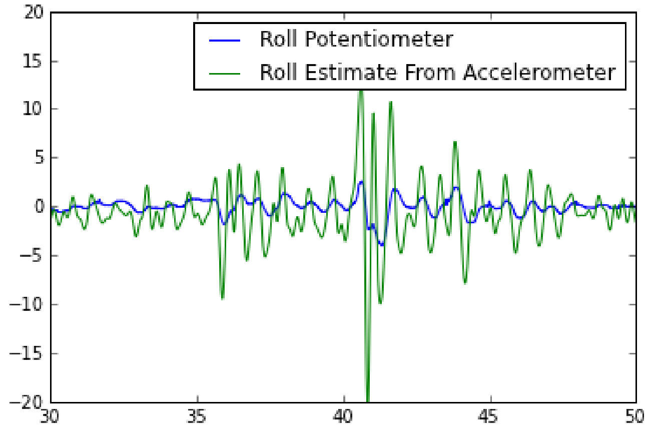
Notice that the mean magnitude of gravity is about what we expect given the standard deviation.

Next we can calculate estimates for the roll and pitch angles.

```
In [39]: phiFromAccel = np.arcsin(ay / magG) # in radians

In [40]: plot(time, phi, time, np.rad2deg(phiFromAccel))
         legend(('Roll Potentiometer', 'Roll Estimate From Accelerometer'))
         xlim((30, 50))
         ylim((-20, 20))

Out[40]: (-20, 20)
```



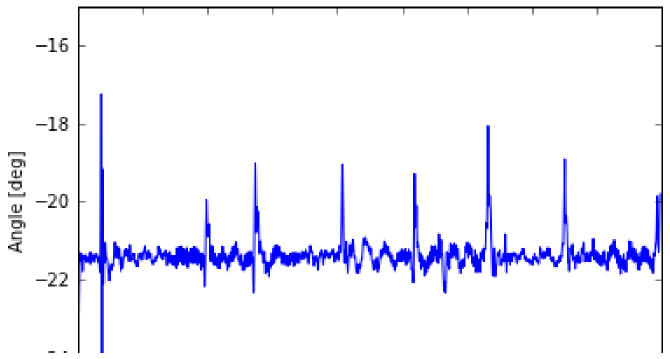
The estimate from the accelerometer predicts much larger magnitudes.

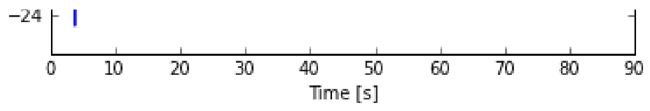
Now we can check out the pitch estimation.

```
In [41]: pitchFromAccel = -np.arcsin(ax / (magG * np.cos(phiFromAccel))) # in radians

In [42]: plot(time, np.rad2deg(pitchFromAccel))
         xlabel('Time [s]')
         ylabel('Angle [deg]')
         ylim((-25, -15))

Out[42]: (-25, -15)
```





In [43]:
```
from scipy.stats import nanmean, nanstd
```

In [44]:
```
np.rad2deg(nanmean(pitchFromAccel))
```

Out[44]:
```
-21.368793393878192
```

In [45]:
```
np.rad2deg(-trial.bicycleRiderParameters['lam'])
```

Out[45]:
```
-18.190784239490387
```

The mean pitch angle during the run is about 3.2 degrees different than the pitch angle we predicted from geometrical measurements of the bicycle.

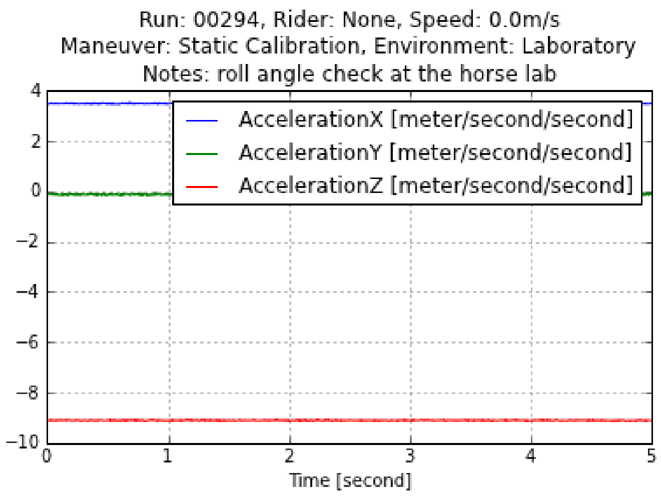
Static Calibration

Let's look at the static roll calibration which was done just prior to run 311.

In [46]:
```
rollCalib = bdp.Run(294, dataset)
```

```
Initializing the run object.
Loading metadata from the database.
Loading the raw signals from the database.
Computing signals from raw data.
===============================================================================
Run # 294
Environment: Laboratory
Rider: None
Bicycle: Rigid Rider
Speed:0.0
Maneuver: Static Calibration
Notes: roll angle check at the horse lab
===============================================================================
```

In [47]:
```
fig = rollCalib.plot('AccelerationX', 'AccelerationY', 'AccelerationZ', signalType='calibrated'
```



In [48]:
```
ax = rollCalib.calibratedSignals['AccelerationX']
```

In [49]:
```
ay = rollCalib.calibratedSignals['AccelerationY']
```

```
In [49]: ay = rollCalib.calibratedSignals['AccelerationY']
```

```
In [50]: az = rollCalib.calibratedSignals['AccelerationZ']
```

```
In [51]: g = np.array([ax, ay, az])
```

```
In [52]: magG = np.zeros(len(ay))
         for i, vec in enumerate(g.T):
             magG[i] = np.sqrt(vec[0]**2 + vec[1]**2 + vec[2]**2)
```

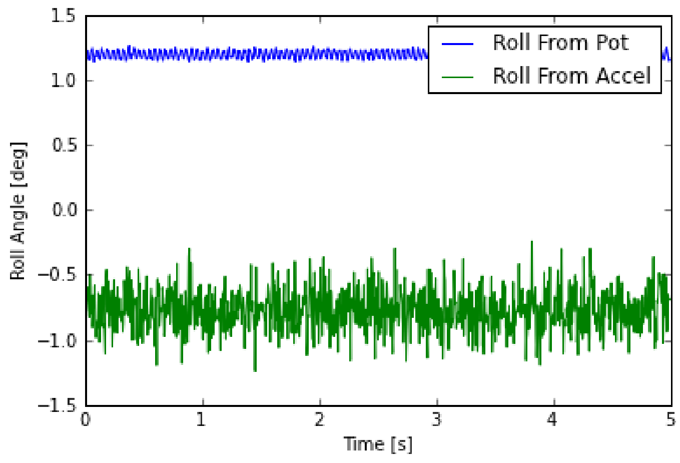
```
In [53]: phiFromAccel = np.arcsin(ay / magG)
```

```
In [54]: pitchFromAccel = -np.arcsin(ax / (magG * np.cos(phiFromAccel)))
```

```
In [55]: phi = rollCalib.calibratedSignals['RollAngle']
```

```
In [56]: plot(phi.time(), phi, ax.time(), np.rad2deg(phiFromAccel))
         legend(('Roll From Pot', 'Roll From Accel'))
         xlabel('Time [s]')
         ylabel('Roll Angle [deg]')

Out[56]: <matplotlib.text.Text at 0x5ff3290>
```



```
In [57]: print 'Roll From Pot', phi.mean(), '+/-', phi.std()

         Roll From Pot 1.19430277299 +/- 0.0275969450842
```

```
In [58]: print 'Roll From Accel:', nanmean(np.rad2deg(phiFromAccel)), '+/-', nanstd(np.rad2deg(phiFromAc

         Roll From Accel: -0.771317093286 +/- 0.159179863716
```

For the static calibration the roll potentiometer and the roll accelerometer give show almost a 2 degree difference in angle.

The pitch angle can also be checked. But we should keep in mind that the steer could be turned, but shouldn't effect the steer angle much.

```
In [59]: print 'Pitch Angle:', nanmean(np.rad2deg(pitchFromAccel)), '+/-', nanstd(np.rad2deg(pitchFromAc

         Pitch Angle: -20.9577081865 +/- 0.0955898437984
```

```
In [60]: steerAngle = rollCalib.calibratedSignals['RollAngle']
```

```
In [61]: print 'Steer Angle:', steerAngle.mean(), '+/-', steerAngle.std()

         Steer Angle: 1.19430277299 +/- 0.0275969450842
```

```
[illegible]
```

The pitch angle from geometry was found to be about 3 degrees different than the one estimated from the accelerometer.

```
In [62]: np.rad2deg(-trial.bicycleRiderParameters['lam'])
```

```
Out[62]: -18.190784239490387
```

If the roll angle from the accelerometer is taken as "true" then the roll angle from the trailer potentiometer has a bias of:

```
In [63]: nanmean(np.rad2deg(phiFromAccel)) - phi.mean()
```

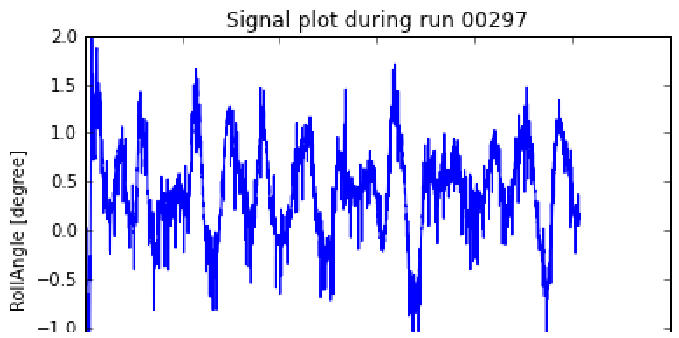
```
Out[63]: -1.9656198662709223
```

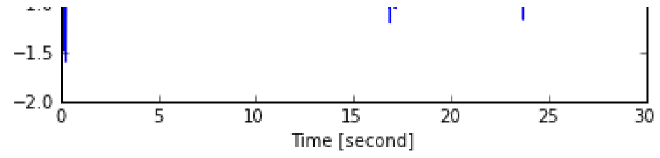
Let's see what the average roll angle for a run without disturbance would be.

```
In [64]: noDist = bdp.Run(297, dataset)
```

```
Initializing the run object.
Loading metadata from the database.
Loading the raw signals from the database.
Loading the bicycle and rider data for Charlie on Rigid Rider
We have foundeth a directory named: /media/Data/Documents/School/UC Davis/Bicycle
Mechanics/BicycleParameters/data/bicycles/Rigidcl.
Found the RawData directory: /media/Data/Documents/School/UC Davis/Bicycle
Mechanics/BicycleParameters/data/bicycles/Rigidcl/RawData
Looks like you've already got some parameters for Rigidcl, use forceRawCalc to recalculate.
There is no rider on the bicycle, now adding Charlie.
Loaded the precalculated parameters from /media/Data/Documents/School/UC Davis/Bicycle
Mechanics/BicycleParameters/data/riders/Charlie/Parameters/CharlieRigidclBenchmark.txt
Computing signals from raw data.
The minimun of the error landscape is 0.420842 and the provided guess is 0.415000
Optimization terminated successfully.
         Current function value: 1.808193
         Iterations: 12
         Function evaluations: 24
Optimization terminated successfully.
         Current function value: 1.808207
         Iterations: 11
         Function evaluations: 22
This is what came out of the minimization: [ 0.41868799]
Extracting the task portion from the data.
===============================================================================
Run # 297
Environment: Horse Treadmill
Rider: Charlie
Bicycle: Rigid Rider
Speed:4.0
Maneuver: Balance
Notes:
===============================================================================
```

```
In [65]: noDist.taskSignals['RollAngle'].convert_units('degree').plot()
```





```
Out[65]: [<matplotlib.lines.Line2D at 0x572e590>]
```

```
In [66]: noDist.taskSignals['RollAngle'][200:].convert_units('degree').mean()
```

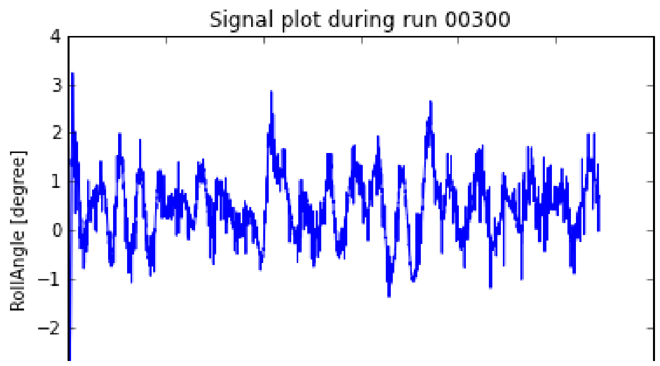
```
Out[66]: 0.40649502252537423
```

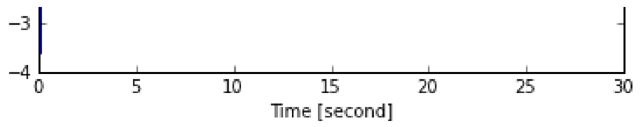
The bias found in the static calibration is about 3 times the mean of this run. I'm not sure how confident I am in the accelerometer's mounting relative to the bicycle frame. But there is also no way to know if the rider actually keeps the bicycle exactly veritcal on average during a run.

```
In [67]: noDist = bdp.Run(300, dataset)
```

```
Initializing the run object.
Loading metadata from the database.
Loading the raw signals from the database.
Loading the bicycle and rider data for Charlie on Rigid Rider
We have foundeth a directory named: /media/Data/Documents/School/UC Davis/Bicycle
Mechanics/BicycleParameters/data/bicycles/Rigidcl.
Found the RawData directory: /media/Data/Documents/School/UC Davis/Bicycle
Mechanics/BicycleParameters/data/bicycles/Rigidcl/RawData
Looks like you've already got some parameters for Rigidcl, use forceRawCalc to recalculate.
There is no rider on the bicycle, now adding Charlie.
Loaded the precalculated parameters from /media/Data/Documents/School/UC Davis/Bicycle
Mechanics/BicycleParameters/data/riders/Charlie/Parameters/CharlieRigidclBenchmark.txt
Computing signals from raw data.
The minimun of the error landscape is 0.308617 and the provided guess is 0.310000
Optimization terminated successfully.
         Current function value: 1.090243
         Iterations: 11
         Function evaluations: 22
Optimization terminated successfully.
         Current function value: 1.090252
         Iterations: 11
         Function evaluations: 22
This is what came out of the minimization: [ 0.30939453]
Extracting the task portion from the data.
===============================================================================
Run # 300
Environment: Horse Treadmill
Rider: Charlie
Bicycle: Rigid Rider
Speed:7.0
Maneuver: Balance
Notes:
===============================================================================
```

```
In [68]: noDist.taskSignals['RollAngle'].convert_units('degree').plot()
```





```
Out[68]: [<matplotlib.lines.Line2D at 0x5f82750>]
```

```
In [69]: noDist.taskSignals['RollAngle'].convert_units('degree')[200:].mean()
```

```
Out[69]: 0.49226255900387894
```

I'm not really sure what to trust as the absolute zero for the roll angle.

```
In [69]:
```