Recall: Programming Assignment 3 (PA2A)

- Write a program that will iteratively prediction and correction based on the location measurements and inferred motions shown below.

```
def predict(mean1, var1, mean2, var2):
    new_mean = mean1 + mean2
    new_var = var1 + var2
    return [new_mean, new_var]

def correct(mean1, var1, mean2, var2):
    new_mean = (var2 * mean1 + var1 * mean2) / (var1 + var2)
    new_var = 1/(1/var1 + 1/var2)
    return [new_mean, new_var]

measurements = [5., 6., 7., 9., 10.]
motion = [0., 1., 1., 2., 1.]
measurement_sig = 4.
motion_sig = 2.
mu = 4
sig = 10000

#Please print out ONLY the final values of the mean
#and the variance in a list [mu, sig].

# Insert code here
```

Kalman Filter Prediction Update in 2D

$$X_t = (x_t, \dot{x}_t)^T$$

$$x_t = x_{t-1} + \dot{x}_{t-1} \times \Delta t + \frac{1}{2}\ddot{x}_{t-1} \times \Delta t^2$$

$$\dot{x}_t = \dot{x}_{t-1} + \ddot{x}_{t-1} \times \Delta t$$

$$\ddot{x}_t = \alpha$$

$$X_t = A_t X_{t-1} + B_t u_t + \varepsilon_t$$

$$z_t = C_t X_t + \delta_t$$

$$A_t = \begin{pmatrix} 1 & \Delta t \\ 0 & 1 \end{pmatrix}$$

$$B_t = \begin{pmatrix} \Delta t^2/2 \\ \Delta t \end{pmatrix}$$

$$u_t = (\alpha)$$

$$C_t = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$\overline{bel}(X_t) = \begin{cases} \bar{\mu}_t = A_t \mu_{t-1} + B_t u_t \\ \bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t \end{cases}$$

Kalman Filter Correction Update in 2D

$$X_t = (x_t, \dot{x}_t)^T$$

$$x_t = x_{t-1} + \dot{x}_{t-1} \times \Delta t + \frac{1}{2}\ddot{x}_{t-1} \times \Delta t^2$$

$$\dot{x}_t = \dot{x}_{t-1} + \ddot{x}_{t-1} \times \Delta t$$

$$\ddot{x}_t = \alpha$$

$$X_t = A_t X_{t-1} + B_t u_t + \varepsilon_t$$

$$z_t = C_t X_t + \delta_t$$

$$A_t = \begin{pmatrix} 1 & \Delta t \\ 0 & 1 \end{pmatrix}$$

$$B_t = \begin{pmatrix} \Delta t^2/2 \\ \Delta t \end{pmatrix}$$

$$u_t = (\alpha)$$

$$C_t = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$bel(x_t) = \begin{cases} \mu_t = \bar{\mu}_t + K_t(z_t - C_t \bar{\mu}_t) \\ \Sigma_t = (I - K_t C_t)\bar{\Sigma}_t \end{cases} \quad \text{with} \quad K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$$

Kalman Filter Updates in 2D

$$X_t = A_t X_{t-1} + B_t u_t + \varepsilon_t$$

$$z_t = C_t X_t + \delta_t$$

$$A_t = \begin{pmatrix} 1 & \Delta t \\ 0 & 1 \end{pmatrix} \quad B_t = \begin{pmatrix} \Delta t^2/2 \\ \Delta t \end{pmatrix} \quad u_t = (\alpha)$$

$$C_t = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Prediction

$$\overline{bel}(X_t) = \begin{cases} \bar{\mu}_t = A_t \mu_{t-1} + B_t u_t \\ \overline{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t \end{cases}$$

Correction

$$bel(x_t) = \begin{cases} \mu_t = \bar{\mu}_t + K_t(z_t - C_t \bar{\mu}_t) \\ \Sigma_t = (I - K_t C_t)\overline{\Sigma}_t \end{cases} \quad \text{with} \quad K_t = \overline{\Sigma}_t C_t^T (C_t \overline{\Sigma}_t C_t^T + Q_t)^{-1}$$

Prediction Update

```python
import numpy as np

# Mean and standard devition of prior (or intial) state
x_mu = 4000       # distance
v_mu = 280        # speed
x_sig = 10.0      # uncetainty in distance
v_sig = 10.0      # uncertainty in speed

# state vector and covariance matrix
X = np.array([[x_mu], [v_mu]])
COV = np.array([[x_sig**2, 0], [0, v_sig**2]])

dt = 1.0          # time interval to update
ac = 2.0          # Acceleration

# Matrixes A, B, u_t
A = np.array([[1.0, dt], [0, 1.0]])
B = np.array([[0.5*dt**2], [dt]])
u_t = np.array([[ac]])

p_x_sig = 20.0  # uncetainly in prediting distance
p_v_sig = 5     # uncertainty in predicting speed

# Covariance matrix in predicting
R_t = np.array([[p_x_sig**2, 0], [0, p_v_sig**2]])

def predict2D(X, COV):
    X_bar = A.dot(X) + B.dot(u_t)
    COV_bar = A.dot(COV).dot(A.T) + R_t
    return X_bar, COV_bar

X_bar, COV_bar = predict2D(X, COV)
```

$$\begin{pmatrix} * & 0 \\ 0 & * \end{pmatrix}$$

$$A_t = \begin{pmatrix} 1 & \Delta t \\ 0 & 1 \end{pmatrix}$$

$$B_t = \begin{pmatrix} \Delta t^2/2 \\ \Delta t \end{pmatrix}$$

$$u_t = (\alpha)$$

$$\overline{bel}(X_t) = \begin{cases} \bar{\mu}_t = A_t \mu_{t-1} + B_t u_t \\ \bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t \end{cases}$$

Correction Update

```python
import numpy as np
from numpy.linalg import inv

# the predicted state and coveraince matrix
X_bar = np.array([[4281.0], [282.0]])
COV_bar = np.array([[600.0, 100.0], [100.0, 125.0]])

# measurements and Uncertainties
x_obs = 4260
v_obs= 282
obs_x_sig = 25
obs_v_sig = 6

# measurement mean vector and covarance matrix
z_t = np.array([[4260.0], [282.0]])
Q_t = np.array([[obs_x_sig**2, 0], [0, obs_v_sig**2]])

# Matrix C
C = np.identity(2)

def correct2D(X_bar, COV_bar, z_t):
    S = C.dot(COV_bar).dot(C.T)+Q_t
    K_t = COV_bar.dot(C.T).dot(inv(S))
    X = X_bar + K_t.dot((z_t - C.dot(X_bar)))
    COV = (np.identity(2) - K_t.dot(C)).dot(COV_bar)
    return X, COV

X, COV = correct2D(X_bar, COV_bar, z_t)
```

$$z_t = C_t X_t + \delta_t$$

$$C_t = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

X

```
[[4271.28655361]
 [ 281.59620777]]
```

COV

```
[[289.09066631   12.01762585]
 [ 12.01762585   27.5203632 ]]
```

$$bel(x_t) = \begin{cases} \mu_t = \bar{\mu}_t + K_t(z_t - C_t\bar{\mu}_t) \\ \Sigma_t = (I - K_t C_t)\bar{\Sigma}_t \end{cases} \quad \text{with} \quad K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$$

Programming Assignment 4 (PA2B)

- Complete the following program that will iteratively predict and correct based on the distance and speed measurements and a given initial state and acceleration by inserting your code in the specified spaces below.

```
def predict2D(X, COV):
    X_bar = A.dot(X) + B.dot(u_t)
    COV_bar = A.dot(COV).dot(A.T) + R_t
    return X_bar, COV_bar

def correct2D(X_bar, COV_bar, z_t):
    S = C.dot(COV_bar).dot(C.T)+Q_t
    K_t = COV_bar.dot(C.T).dot(inv(S))
    X = X_bar + K_t.dot((z_t - C.dot(X_bar)))
    COV = (np.identity(2) - K_t.dot(C)).dot(COV_bar)
    return X, COV

for i in range(len(x_obses)):
    X_bar, COV_bar = predict2D(X, COV)
    z_t = np.array([[x_obses[i]], [v_obses[i]]])
    X, COV = correct2D(X_bar, COV_bar, z_t)

print (X)
print (COV)
```

```
import numpy as np
from numpy.linalg import inv

# Mean and standard devition of intial) state
x_mu = 4000        # distance
v_mu = 280         # speed
x_sig = 10.0       # uncetainty in distance
v_sig = 10.0       # uncertainty in speed

# measurements and Uncertainties in measuring
x_obses = np.array([4260, 4550, 4860, 5110])  # distances
v_obses = np.array([282, 285, 286, 290])      # speeds

obs_x_sig = 25   # uncetainty in distance measuring
obs_v_sig = 6    # uncertainty in speed measuring

dt = 1.0         # time interval to update
ac = 2.0         # Acceleration

p_x_sig = 20.0   # uncetainty in prediting distance
p_v_sig = 5      # uncertainty in predicting speed

# initial state and covariance matrix
# insert your code here


# Matrixes A, B, C, u_t
# insert your code here


# Covariance matrix R_t in predicting
# Insert your code here


# Covariance matrix Q_t in measuring
# Insert your code here
```

Kalman Filter **Prediction** Update in 4D

$$X_t = (x_t, y_t, \dot{x}_t, \dot{y}_t)^T$$

$$x_t = x_{t-1} + \dot{x}_{t-1} \times \Delta t + \frac{1}{2}\ddot{x}_{t-1} \times \Delta t^2$$

$$y_t = y_{t-1} + \dot{y}_{t-1} \times \Delta t + \frac{1}{2}\ddot{y}_{t-1} \times \Delta t^2$$

$$\dot{x}_t = \dot{x}_{t-1} + \ddot{x}_{t-1} \times \Delta t \qquad \ddot{x}_t = \alpha$$

$$\dot{y}_t = \dot{y}_{t-1} + \ddot{y}_{t-1} \times \Delta t \qquad \ddot{y}_t = \beta$$

$$X_t = A_t X_{t-1} + B_t u_t + \varepsilon_t \qquad z_t = C_t X_t + \delta_t \qquad u_t = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$$

$$A_t = \begin{pmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad B_t = \begin{pmatrix} \Delta t^2/2 & 0 \\ 0 & \Delta t^2/2 \\ \Delta t & 0 \\ 0 & \Delta t \end{pmatrix}, \qquad C_t = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\overline{bel}(x_t) = \begin{cases} \bar{\mu}_t = A_t \mu_{t-1} + B_t u_t \\ \bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t \end{cases}$$

Kalman Filter Correction Update in 4D

$$X_t = (x_t, y_t, \dot{x}_t, \dot{y}_t)^T$$

$$x_t = x_{t-1} + \dot{x}_{t-1} \times \Delta t + \frac{1}{2}\ddot{x}_{t-1} \times \Delta t^2$$

$$y_t = y_{t-1} + \dot{y}_{t-1} \times \Delta t + \frac{1}{2}\ddot{y}_{t-1} \times \Delta t^2$$

$$\dot{x}_t = \dot{x}_{t-1} + \ddot{x}_{t-1} \times \Delta t$$

$$\ddot{x}_t = \alpha$$

$$\dot{y}_t = \dot{y}_{t-1} + \ddot{y}_{t-1} \times \Delta t$$

$$\ddot{y}_t = \beta$$

$$X_t = A_t X_{t-1} + B_t u_t + \varepsilon_t$$

$$z_t = C_t X_t + \delta_t$$

$$u_t = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$$

$$A_t = \begin{pmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$B_t = \begin{pmatrix} \Delta t^2/2 & 0 \\ 0 & \Delta t^2/2 \\ \Delta t & 0 \\ 0 & \Delta t \end{pmatrix}$$

$$C_t = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$bel(x_t) = \begin{cases} \mu_t = \bar{\mu}_t + K_t(z_t - C_t\bar{\mu}_t) \\ \Sigma_t = (I - K_t C_t)\bar{\Sigma}_t \end{cases} \quad \text{with} \quad K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$$

Programming Project #2: Kalman Localization in 4D

PP2-KalmanFilter4D.py

- Write a program that will iteratively predict and correct based on the distance and speed measurements and a given initial state and acceleration in 4D.

```
# Mean and standard deviation of initial state in x-direction
x_mu = 4000       # distance
xv_mu = 280       # speed
x_sig = 10.0      # uncetainty in distance
xv_sig = 10.0     # uncertainty in speed

# Mean and standard deviation of initial state in y-direction
y_mu = 3000       # distance
yv_mu = 180       # speed
y_sig = 10.0      # uncetainty in distance
yv_sig = 10.0     # uncertainty in speed

# measurements and uncertainties in measuring in x-direction
x_obses = np.array([4260, 4550, 4860, 5110])  # distances
xv_obses = np.array([282, 285, 286, 290])     # speeds

# measurements and uncertainties in measuring in y-direction
y_obses = np.array([3181, 3366, 3552, 3742])  # distances
yv_obses = np.array([184, 186, 190, 194])     # speeds

obs_x_sig = 25   # uncetainty in distance measuring in x-direction
obs_xv_sig = 6   # uncertainty in speed measuring in x-direction
obs_y_sig = 25   # uncetainty in distance measuring in y-direction
obs_yv_sig = 6   # uncertainty in speed measuring in y-direction

dt = 1.0         # time interval to update
acx = 2.0        # Acceleration in x-direction
acy = 3.0        # Acceleration in y-direction
```

```python
p_x_sig = 20.0  # uncetainty in prediting distance in x-direction
p_xv_sig = 5    # uncertainty in predicting speed in x-direction
p_y_sig = 20.0  # uncetainty in prediting distance in y-direction
p_yv_sig = 5    # uncertainty in predicting speed in y-direction
```

```python
# Enter your code below to provide
# (1) initial state: X and covariance matrix: COV
# (2) Matrixes: A, B, C, u_t
# (3) Covariance matrix in predicting: R_t
# (4) Covariance matrix in measuring: Q_t



# Do not add or change any code below
def predict4D(X, COV):
    X_bar = A.dot(X) + B.dot(u_t)
    COV_bar = A.dot(COV).dot(A.T) + R_t
    return X_bar, COV_bar

def correct4D(X_bar, COV_bar, z_t):
    S = C.dot(COV_bar).dot(C.T)+Q_t
    K_t = COV_bar.dot(C.T).dot(inv(S))
    X = X_bar + K_t.dot((z_t - C.dot(X_bar)))
    COV = (np.identity(4) - K_t.dot(C)).dot(COV_bar)
    return X, COV

for i in range(len(x_obses)):
    X_bar, COV_bar = predict4D(X, COV)
    z_t = np.array([[x_obses[i]], [y_obses[i]], [xv_obses[i]], [yv_obses[i]]])
    X, COV = correct4D(X_bar, COV_bar, z_t)

print (X)
print (COV)
```

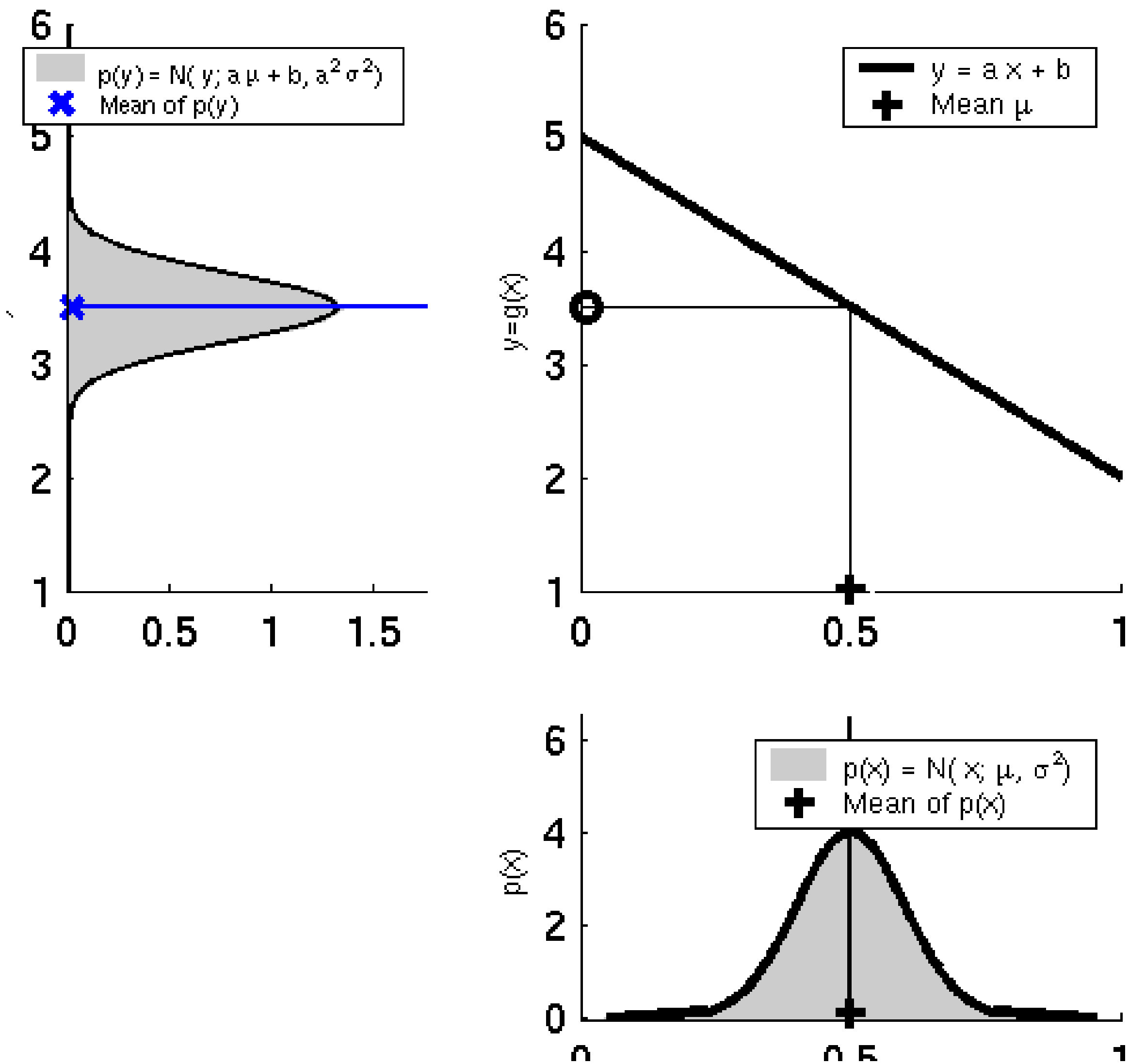
Nonlinear Dynamic Systems

- Most realistic robotic problems involve nonlinear functions

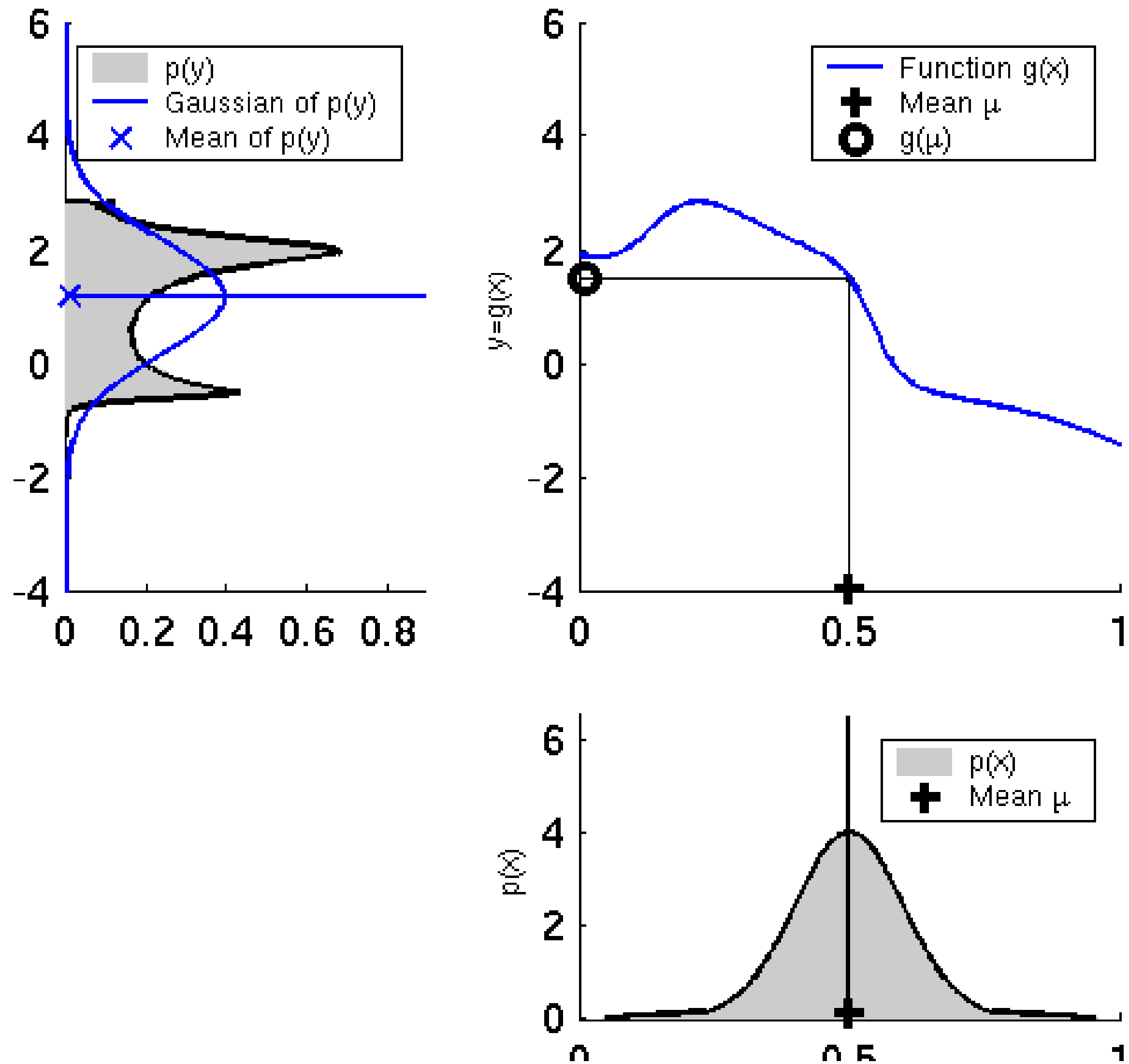
$$x_t = g(u_t, x_{t-1})$$

$$z_t = h(x_t)$$

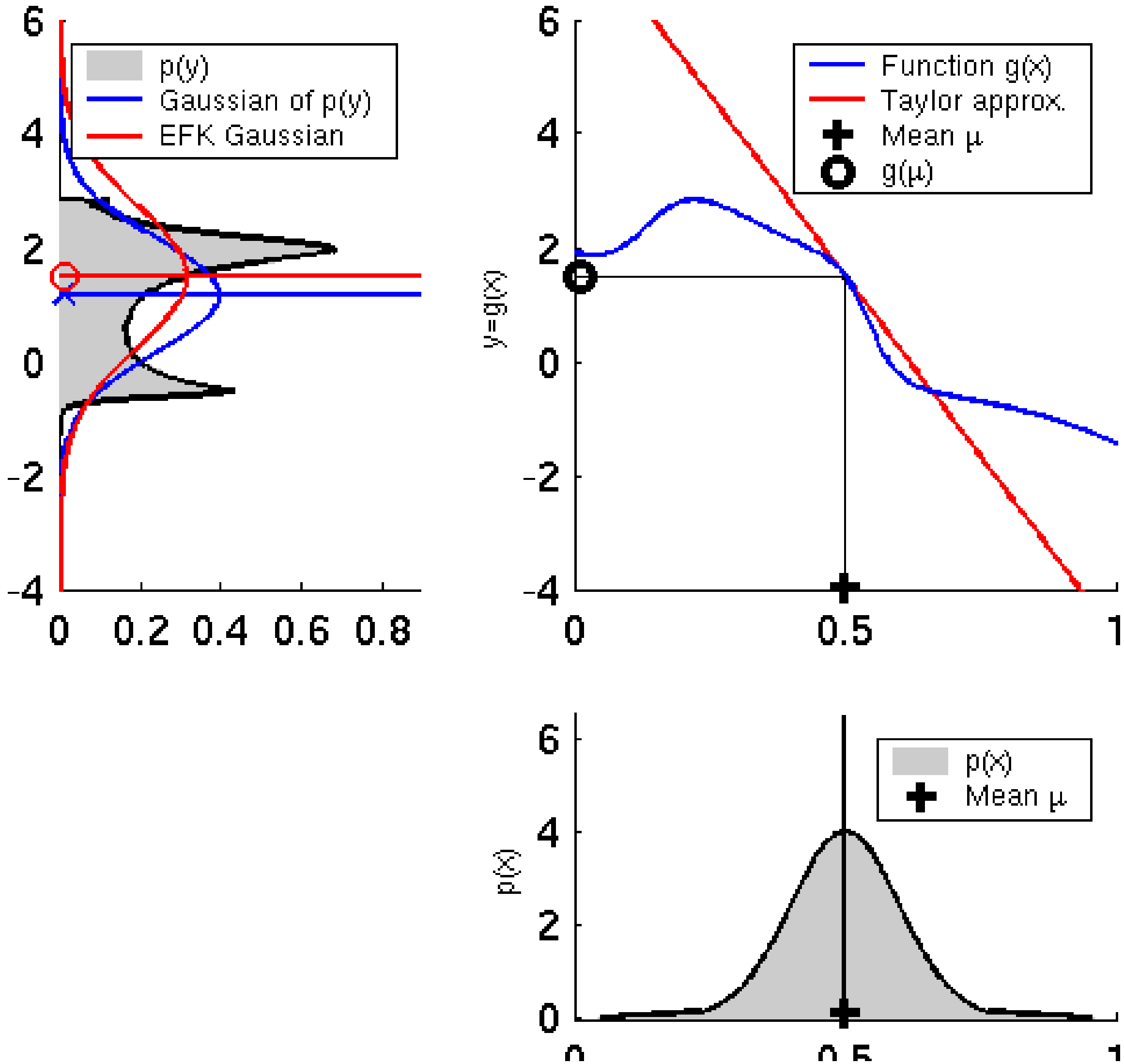
Linearity Assumption Revisited



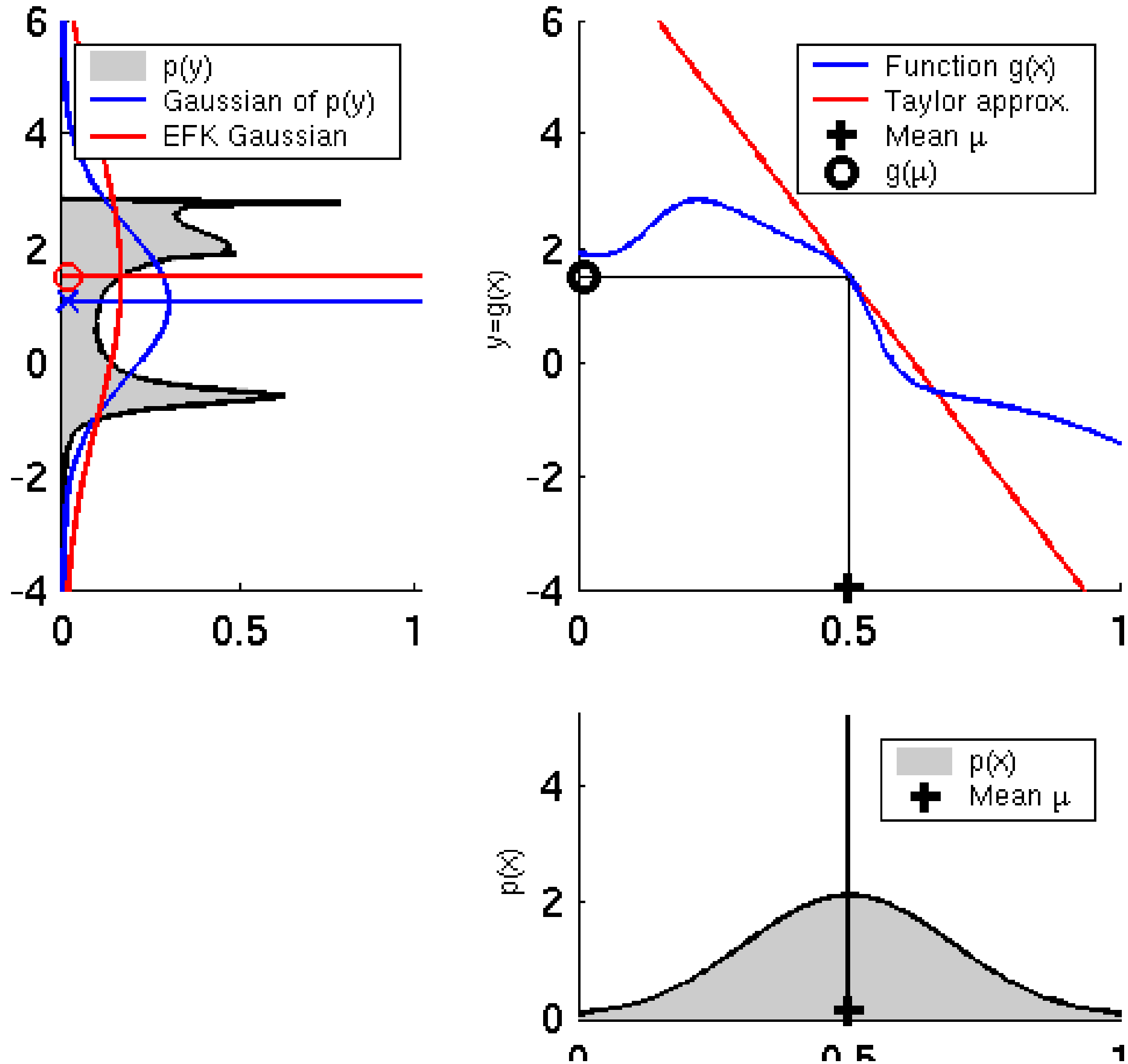
Non-linear Function



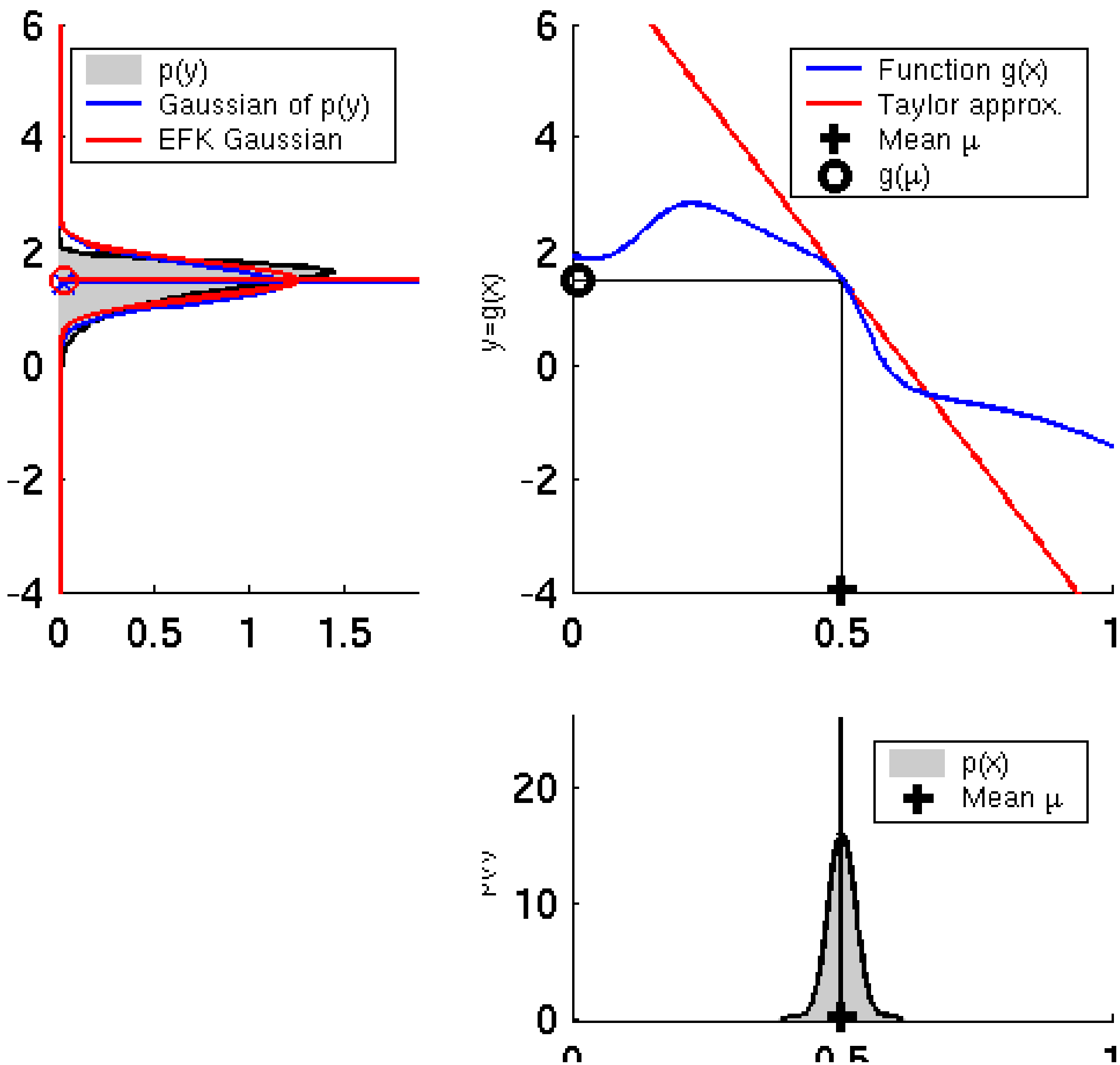
EKF Linearization (1)



EKF Linearization (2)



EKF Linearization (3)



EKF Linearization: First Order Taylor Series Expansion

- Prediction:

$$g(u_t, x_{t-1}) \approx g(u_t, \mu_{t-1}) + \frac{\partial g(u_t, \mu_{t-1})}{\partial x_{t-1}} \, (x_{t-1} - \mu_{t-1})$$

$$g(u_t, x_{t-1}) \approx g(u_t, \mu_{t-1}) + G_t \, (x_{t-1} - \mu_{t-1})$$

- Correction:

$$h(x_t) \approx h(\bar{\mu}_t) + \frac{\partial h(\bar{\mu}_t)}{\partial x_t} \, (x_t - \bar{\mu}_t)$$

$$h(x_t) \approx h(\bar{\mu}_t) + H_t \, (x_t - \bar{\mu}_t)$$

$x_t = g(u_t, x_{t-1})$

$z_t = h(x_t)$

EKF Algorithm

$x_t = A_t x_{t-1} + B_t u_t + \varepsilon_t$

$z_t = C_t x_t + \delta_t$

1. **Extended_Kalman_filter**($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$):
2. Prediction:
3. $\bar{\mu}_t = g(u_t, \mu_{t-1}) \longleftarrow \bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$
4. $\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t \longleftarrow \bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$
5. Correction:
6. $K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1} \longleftarrow K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$
7. $\mu_t = \bar{\mu}_t + K_t (z_t - h(\bar{\mu}_t)) \longleftarrow \mu_t = \bar{\mu}_t + K_t (z_t - C_t \bar{\mu}_t)$
8. $\Sigma_t = (I - K_t H_t) \bar{\Sigma}_t \longleftarrow \Sigma_t = (I - K_t C_t) \bar{\Sigma}_t$
9. Return μ_t, Σ_t

$$\boxed{H_t = \frac{\partial h(\bar{\mu}_t)}{\partial x_t}} \qquad \boxed{G_t = \frac{\partial g(u_t, \mu_{t-1})}{\partial x_{t-1}}}$$

Localization

> "Using sensory information to locate the robot in its environment is the most fundamental problem to providing a mobile robot with autonomous capabilities." [Cox '91]

- **Given**
  - Map of the environment.
  - Sequence of sensor measurements.
- **Wanted**
  - Estimate of the robot's position.
- **Problem classes**
  - Position tracking
  - Global localization
  - Kidnapped robot problem (recovery)