# Design of kalman filter

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt
```

In [4]:

```
# data

x0 = 4000 # m
v0 = 280  # m/s

#obserations(values from sensor)
x0_obs = 4000 # m
v0_obs = 280  # m/s
x1_obs = 4260 # m
v1_obs = 282  # m/s
x2_obs = 4550 # m
v2_obs = 285  # m/s
x3_obs = 4860 # m
v3_obs = 286  # m/s
x4_obs = 5110 # m
v4_obs = 290  # m/s

#initial conditions
ax = 2    # m/sec^2
v0 = 280  # m/s
del_t = 1 # sce
del_x = 25 # m

#process error in process(Covariance Matrix)
del_p_x = 20 # m
del_p_vx = 5 # m/sec

#obseravtion error
del_x = 25 # m
del_vx = 6 # m/sec

#noise and distrubance
wk = 0
Qk = 0
zk = 0

# Data For graph
predicted_value_position = [x0]
predicted_value_velocity = [v0]

measurement_value_position = [x0_obs,x1_obs,x2_obs,x3_obs,x4_obs]
measurement_value_velocity = [v0_obs,v1_obs,v2_obs,v3_obs,v4_obs]

kalman_value_position = [x0]
kalman_value_velocity = [v0]
```

In [5]:

```
x_k1 = np.array([[x0],[v0]])

# The initial process covariance Matrix
# p_k = [(del_p_x)^2  del_p_x*del_p_vx; del_p_x*del_p_vx  (del_p_vx)^2]
p_k = np.array([[(del_p_x  * del_p_x ), (del_p_x  * del_p_vx)],[(del_p_x  * del_p_vx), (del_p_vx *
del_p_vx)]])
p_k[0,1] = 0
p_k[1,0] = 0


for t in range(1,5):

# The predicted state
```

```python
# x_kp = A*x_k1 + B*u_k + wk      (u_k = ax)
    x_kp = np.dot((np.array([[1,del_t],[0,1]])), x_k1) + (np.array([[(del_t * del_t)/2],[del_t]]) *
ax) + wk

    predicted_value_position.append(x_kp[0])
    predicted_value_velocity.append(x_kp[1])

# The predicted process covariance Matrix
# p_kp = A*p_k*A' + Qk
    A = np.array([[1,del_t],[0,1]])
    p_kp = np.dot(A,(np.dot(p_k,A.transpose()))) + Qk
    p_kp[0,1] = 0
    p_kp[1,0] = 0

# Calcuate the kalman gain
# k = (p_kp*H')/(H*p_kp*H' + R)
    H = np.array([[1,0],[0,1]])
    R = np.array([[25*25,0],[0,6*6]])
    k = np.divide((np.dot(p_kp,H.transpose())),(np.dot(H,np.dot(p_kp,H.transpose())) + R))
    k[0,1] = 0
    k[1,0] = 0

# The new observation
# yk = C*y_lm + zk
    c = np.array([[1,0],[0,1]])
    if t == 1:
        y_lm = np.array([[x1_obs],[v1_obs]])
    elif t == 2:
        y_lm = np.array([[x2_obs],[v2_obs]])
    elif t == 3:
        y_lm = np.array([[x3_obs],[v3_obs]])
    elif t == 4:
        y_lm = np.array([[x4_obs],[v4_obs]])
    yk = np.dot(c, y_lm) + zk

# Calculate the current state
# xk = x_kp + k(yk - H*x_kp)
    xk = x_kp + np.dot(k, (yk - np.dot(H,x_kp)))
    x_k1 = xk   # xk will add with the first stage (1) the predicted state

    kalman_value_position.append(xk[0])
    kalman_value_velocity.append(xk[1])

# Update the process covariance matrix
# pk = (I - k*H)p_kp
    I = np.array([[1,0],[0,1]])
    pk = np.dot((I - np.dot(k, H)), p_kp)
    p_k = pk
```

```
/srv/conda/envs/notebook/lib/python3.6/site-packages/ipykernel_launcher.py:30: RuntimeWarning:
invalid value encountered in true_divide
```

In [6]:

```python
# position
print(predicted_value_position)

print(measurement_value_position)

print(kalman_value_position)
```

```
[4000, array([4281.]), array([4555.5]), array([4839.14124475]), array([5131.38286854])]
[4000, 4260, 4550, 4860, 5110]
[4000, array([4272.5]), array([4553.85054707]), array([4844.15764332]), array([5127.05898493])]
```

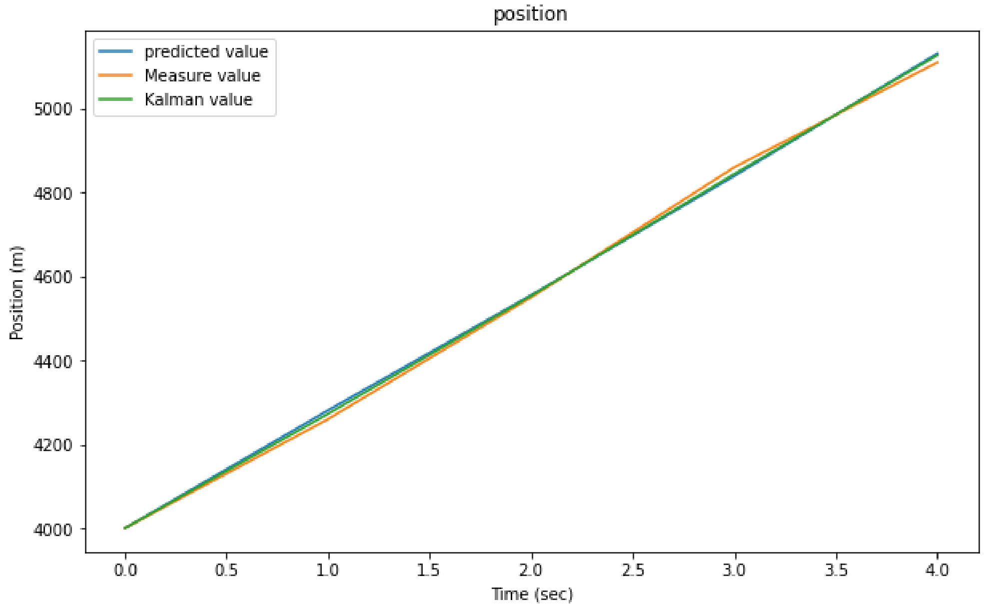
In [7]:

```python
# velocity
print(predicted_value_velocity)

print(measurement_value_velocity)

print(kalman_value_velocity)
```

```
[280, array([282.]), array([284.]), array([286.29069767]), array([288.22522523])]
[280, 282, 285, 286, 290]
[280, array([282.]), array([284.29069767]), array([286.22522523]), array([288.55147059])]
```

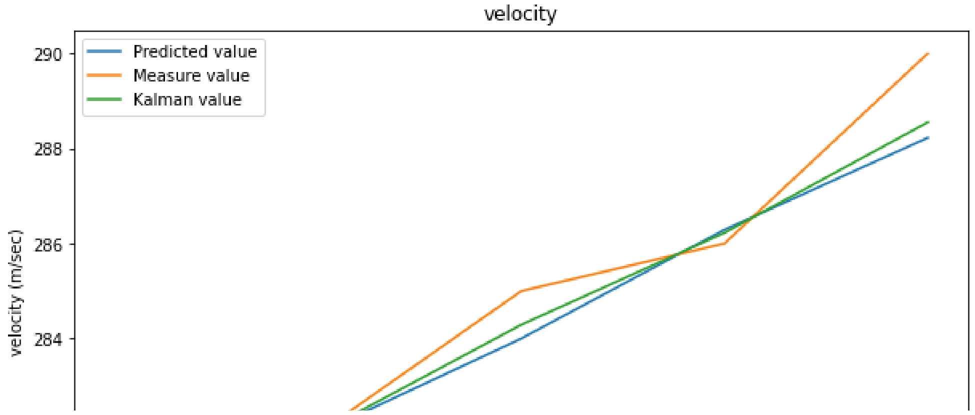
In [10]:

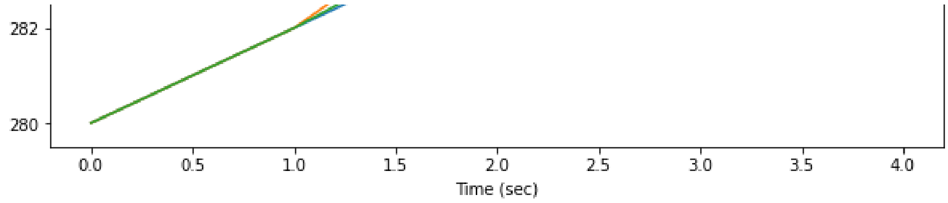
```python
t = range(0,5)
# position

plt.plot(t, predicted_value_position, label = 'predicted value')
plt.plot(t, measurement_value_position, label = 'Measure value')
plt.plot(t, kalman_value_position, label = 'Kalman value')
plt.xlabel("Time (sec)")
plt.ylabel("Position (m)")
plt.title("position")
plt.legend()
plt.rcParams['figure.figsize'] = (10,6)
plt.show()
```




In [9]:

```python
# velocity

plt.plot(t,predicted_value_velocity, label = 'Predicted value')
plt.plot(t,measurement_value_velocity, label = 'Measure value')
plt.plot(t,kalman_value_velocity, label = 'Kalman value')
plt.xlabel("Time (sec)")
plt.ylabel("velocity (m/sec)")
plt.title("velocity")
plt.legend()
plt.rcParams['figure.figsize'] = (10,6)
plt.show()
```






In [ ]: