
# ML HW4.B

ML | spring 1401 | Dr.ABDI

---

Student name : **Amin Fathi**

Student id : **400722102**

در تعریف توابع اولیه از لینک شماره ۱ استفاده شده است که آن هم بر اساس ویکیپدیا است و در کل با اسلاید های ۵۲ هم تطبیق دارد .

$$\mathbf{x}_k = \mathbf{F}_k\mathbf{x}_{k-1} + \mathbf{B}_k\mathbf{u}_k + \mathbf{w}_k$$

where

- $\mathbf{F}_k$ is the state transition model which is applied to the previous state $\mathbf{x}_{k-1}$;
- $\mathbf{B}_k$ is the control-input model which is applied to the control vector $\mathbf{u}_k$;
- $\mathbf{w}_k$ is the process noise, which is assumed to be drawn from a zero mean multivariate normal distribution, $\mathcal{N}$, with covariance, $\mathbf{Q}_k$: $\mathbf{w}_k \sim \mathcal{N}(0, \mathbf{Q}_k)$.

At time *k* an observation (or measurement) $\mathbf{z}_k$ of the true state $\mathbf{x}_k$ is made according to

$$\mathbf{z}_k = \mathbf{H}_k\mathbf{x}_k + \mathbf{v}_k$$

where

- $\mathbf{H}_k$ is the observation model, which maps the true state space into the observed space and
- $\mathbf{v}_k$ is the observation noise, which is assumed to be zero mean Gaussian white noise with covariance $\mathbf{R}_k$: $\mathbf{v}_k \sim \mathcal{N}(0, \mathbf{R}_k)$.

در جزوه به صورت زیر آمده است .

$$P(\mathbf{x}_{t+1} \mid \mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t+1}; \mathbf{F}\mathbf{x}_t, \Sigma_x)$$
$$P(\mathbf{z}_t \mid \mathbf{x}_t) = \mathcal{N}(\mathbf{z}_t; \mathbf{H}\mathbf{x}_t, \Sigma_z),$$

و فرمول های زیر در ویکی پدیا :

## Predict [ edit ]

| | |
|---|---|
| Predicted (*a priori*) state estimate | $\hat{\mathbf{x}}_{k\mid k-1} = \mathbf{F}_k\hat{\mathbf{x}}_{k-1\mid k-1} + \mathbf{B}_k\mathbf{u}_k$ |
| Predicted (*a priori*) estimate covariance | $\mathbf{P}_{k\mid k-1} = \mathbf{F}_k\mathbf{P}_{k-1\mid k-1}\mathbf{F}_k^\mathsf{T} + \mathbf{Q}_k$ |

## Update [ edit ]

| | |
|---|---|
| Innovation or measurement pre-fit residual | $\tilde{\mathbf{y}}_k = \mathbf{z}_k - \mathbf{H}_k\hat{\mathbf{x}}_{k\mid k-1}$ |
| Innovation (or pre-fit residual) covariance | $\mathbf{S}_k = \mathbf{H}_k\mathbf{P}_{k\mid k-1}\mathbf{H}_k^\mathsf{T} + \mathbf{R}_k$ |
| *Optimal* Kalman gain | $\mathbf{K}_k = \mathbf{P}_{k\mid k-1}\mathbf{H}_k^\mathsf{T}\mathbf{S}_k^{-1}$ |
| Updated (*a posteriori*) state estimate | $\hat{\mathbf{x}}_{k\mid k} = \hat{\mathbf{x}}_{k\mid k-1} + \mathbf{K}_k\tilde{\mathbf{y}}_k$ |
| Updated (*a posteriori*) estimate covariance | $\mathbf{P}_{k\mid k} = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\,\mathbf{P}_{k\mid k-1}$ |
| Measurement post-fit residual | $\tilde{\mathbf{y}}_{k\mid k} = \mathbf{z}_k - \mathbf{H}_k\hat{\mathbf{x}}_{k\mid k}$ |

که به صورت زیر در پی دی اف درسی آمده است :

$$\begin{aligned}\mu_{t+1} &= \mathbf{F}\mu_t + \mathbf{K}_{t+1}(\mathbf{z}_{t+1} - \mathbf{H}\mathbf{F}\mu_t)\\ \Sigma_{t+1} &= (\mathbf{I} - \mathbf{K}_{t+1}\mathbf{H})(\mathbf{F}\Sigma_t\mathbf{F}^\top + \Sigma_x)\,,\end{aligned}$$

❌ که در آن، ماتریس k (Kalman Gain) تعریف می‌شود:

$$\mathbf{K}_{t+1} = (\mathbf{F}\Sigma_t\mathbf{F}^\top + \Sigma_x)\mathbf{H}^\top(\mathbf{H}(\mathbf{F}\Sigma_t\mathbf{F}^\top + \Sigma_x)\mathbf{H}^\top + \Sigma_z)^{-1}$$

بنابراین قسمت اولیه کد که در گیت هاب موجود بود کاملا درست بود و فقط کافی است ماتریس ها را به درستی انتخاب کنیم .

ماتریس ها را هم در ادامه از همان لینک ویکی پدیا در می آوریم .

با این فرض که ما مکان و سرعت محرک را هر dt ثانیه می سنجیم ( روی سوال ۰.۱ ذکر شده است )

$$\mathbf{x}_k = \begin{bmatrix} x \\ \dot{x} \end{bmatrix}$$

$$\mathbf{x}_k = \mathbf{F}\mathbf{x}_{k-1} + \mathbf{G}a_k$$

$$\mathbf{F} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}$$

$$\mathbf{G} = \begin{bmatrix} \frac{1}{2}\Delta t^2 \\ \Delta t \end{bmatrix}$$

so that

$$\mathbf{x}_k = \mathbf{F}\mathbf{x}_{k-1} + \mathbf{w}_k$$

where

$$\mathbf{w}_k \sim N(0, \mathbf{Q})$$

$$\mathbf{Q} = \mathbf{G}\mathbf{G}^{\mathsf{T}}\sigma_a^2 = \begin{bmatrix} \frac{1}{4}\Delta t^4 & \frac{1}{2}\Delta t^3 \\ \frac{1}{2}\Delta t^3 & \Delta t^2 \end{bmatrix}\sigma_a^2.$$

$$\mathbf{z}_k = \mathbf{H}\mathbf{x}_k + \mathbf{v}_k$$

where

$$\mathbf{H} = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

and

$$\mathbf{R} = \mathrm{E}\left[\mathbf{v}_k\mathbf{v}_k^{\mathsf{T}}\right] = \begin{bmatrix} \sigma_z^2 \end{bmatrix}$$

که همگی را به صورت زیر در کد تعریف کرده ایم و لازم به ذکر است مکان اولیه را ۰ و سرعت اولیه را ۱۰ و هر رفتار مدل را هر ۰.۱ بار و برای ۱۰۰ گام بررسی کردیم :

```
dt = 0.1
F = np.array([[1, dt], [0, 1]])
H = np.array([1, 0]).reshape(1, 2)
Q = np.array([[0.25 * (dt **4) * 0.5, 0.5 * (dt**3) * 0.5], [ 0.5 * (dt**3) * 0.5 , 0.5*(dt**2)]])
R = np.array([0.5]).reshape(1, 1)
```

دو نویز متفاوت برای مشاهدات در نظر گرفتیم ، یکی برای مکان و یکی برای سرعت :

```
noise1 =np.random.normal(0, 0.5, 100)
print(noise1)
noise2 = np.random.normal(0, 1, 100)
```

اولین مشاهده را هم به صورت زیر تعریف کردیم :

```
newmesuer = [ 0 , 10]
```

و هر بار مشاهدات مکان و زمان را ( کلا ۱۰۰ بار تعریف کردیم ) ثبت می کنیم و در دو لیست جدا دخیره می کنیم.

```
newmesuer = np.dot(F, newmesuer) + [noise1[k] , noise2[k]]
print(newmesuer)
measurementsofplace.append(newmesuer[0])
measurementsofspeed.append(newmesuer[1])
```

سرعت پیش بینی مان هم که ثابت است و برابر ۱۰ :

```
speed = [10] * 100
```

مکان پیش بینی و سرعت پیشبینی با فیلتر کالمن را هم به روش زیر به دست می آوریم:

```
for z in measurementsofplace:
  a = kf.predict()
  place = a[0][0]
  speed = a[1]
  predictions_place.append(np.dot(H, a)[0])
  predictions_speed.append(speed)
  kf.update(z)
```

اختلاف بین سرعت پیش بینی شده و به دست آمده توسط سنسور ها و همچنین مکان پیشبینی شده و به دست آمده توسط سنسور ها را در این دو لیست ذخیره میکنیم.

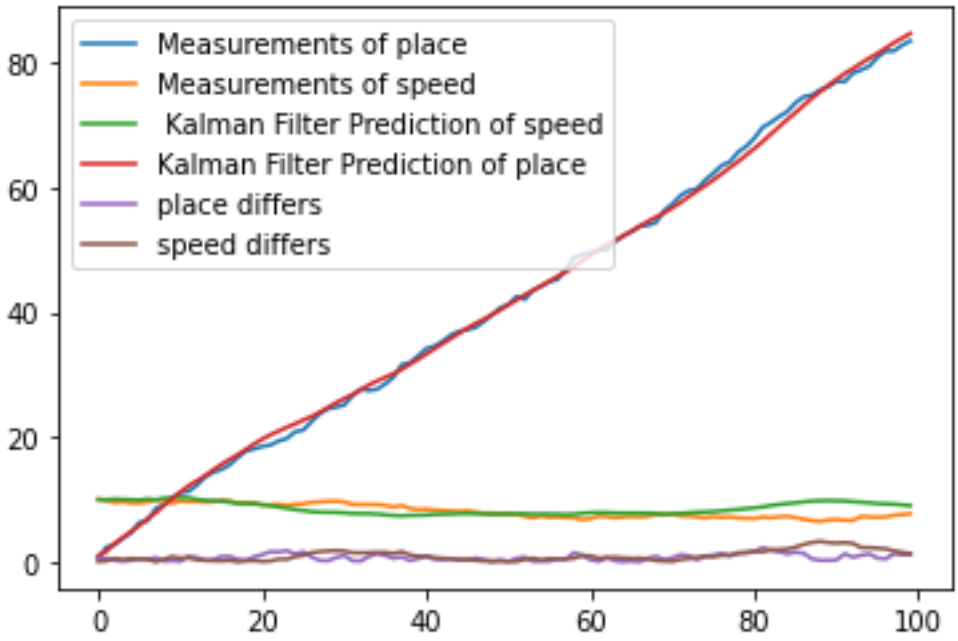
```
difference  = np.abs( measurementsofplace - np.ravel(predictions_place))
difference2 = np.abs( measurementsofspeed - np.ravel(predictions_speed))
```

حال همه لیست ها را نشان می دهیم :

```
plt.plot(range(len(measurementsofplace)), measurementsofplace, label = 'Measurements of place')
plt.plot(range(len(measurementsofspeed)), measurementsofspeed, label = 'Measurements of speed')
plt.plot(range(len(predictions_speed)), predictions_speed, label = ' Kalman Filter Prediction of speed')
plt.plot(range(len(predictions_place)), np.ravel(predictions_place), label = 'Kalman Filter Prediction of place')
plt.plot(range(len(difference)), difference, label = 'place differs')
plt.plot(range(len(difference2)), difference2, label = 'speed differs')
```

و همچنین اختلاف ها را هم پرینت می کنیم

```
print( 'PLACE diifer is :' , difference)
print('speed differ is : ' ,difference2 )
```

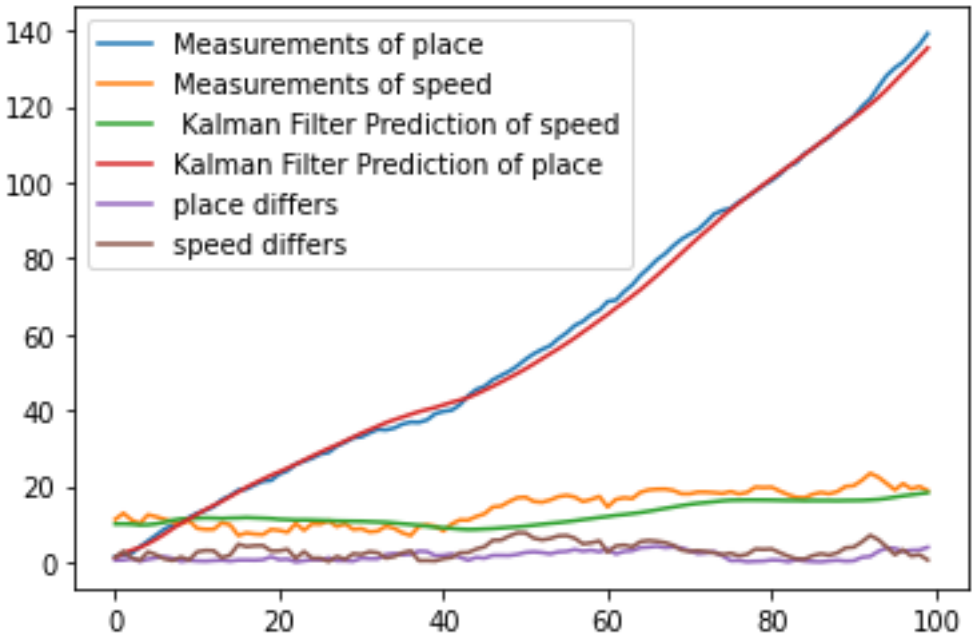



نمودار برای مکان اولیه ۰ و سرعت ثابت ۱۰ ، با واریانس نویز مکان = ۰.۵ و سرعت = ۰.۲

مشاهده میشود که تخمین فیلتر کالمن برای سرعت ثابت برابر با ۱۰ ، خط سبز رنگ است و سرعت به دست آمده از سنسور هم همان حوالی است و نارنجی رنگ است و کمی نویز دارد ، مکان هم به دست آمده توسط سنسور خط آبی است که تقریب فیلتر کالمن خط قرمز است.

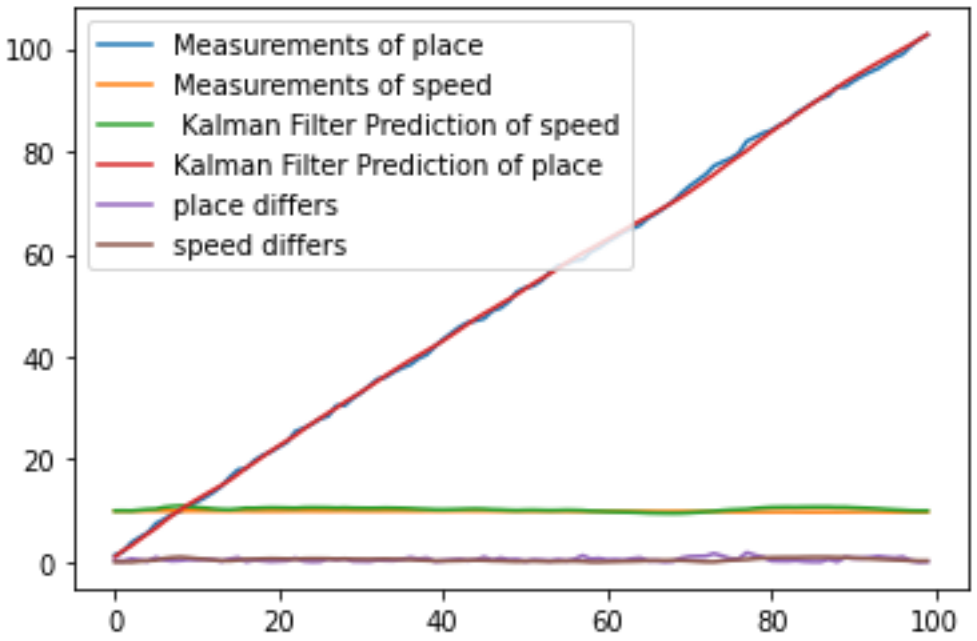
اختلاف ها بین فیلتر کالمن و سنسور  هم دو خط بنفش رنگ در پایین جدول هستند.

مشخصا با افزایش واریانس نویز حس گر سرعت ، سرعت نویزی ثبت خواهد شد و تاثیر خود را بر مکان نیز خواهد گذاشت چرا که مکان حاصل جمع مکان فعلی + سرعت * dt است. مانند شکل زیر



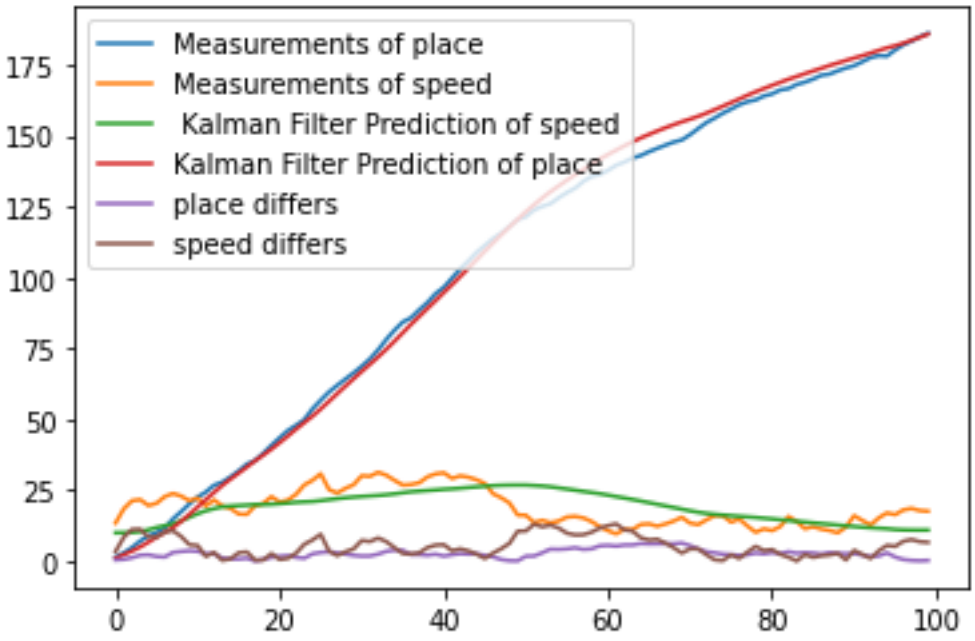

نمودار برای مکان اولیه ۰ و سرعت ثابت ۱۰ ، با واریانس نویز مکان = ۰.۵ و سرعت = ۱

با کاهش نویز هم مشخصا همه چیز خطی تر می شود :

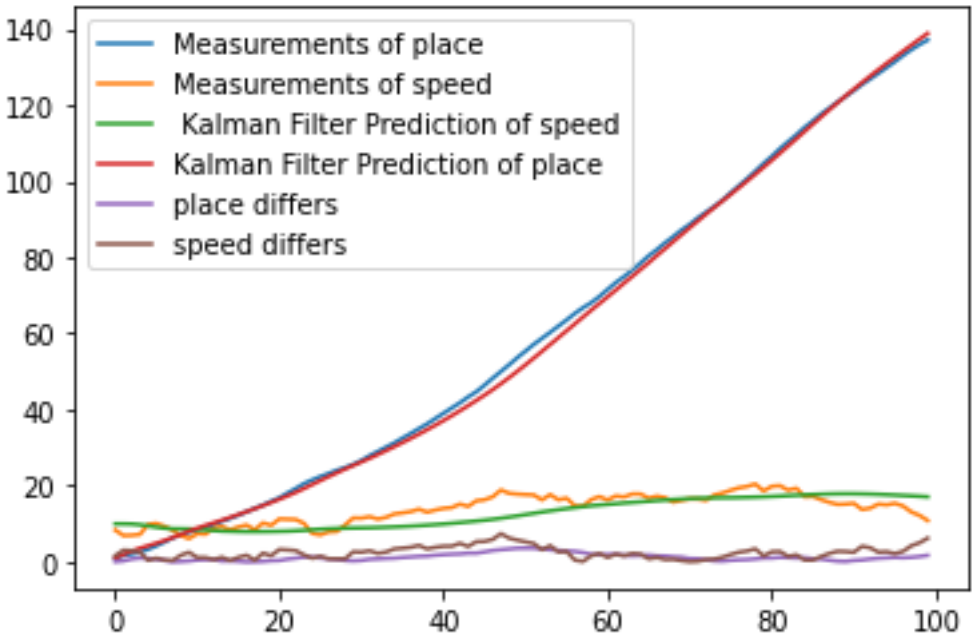



نمودار برای مکان اولیه ۰ و سرعت ثابت ۱۰ ، با واریانس نویز مکان = ۰.۵ و سرعت = ۰.۰۱

نمونه ای دیگر از اثرات افزایش واریانس نویز سرعت



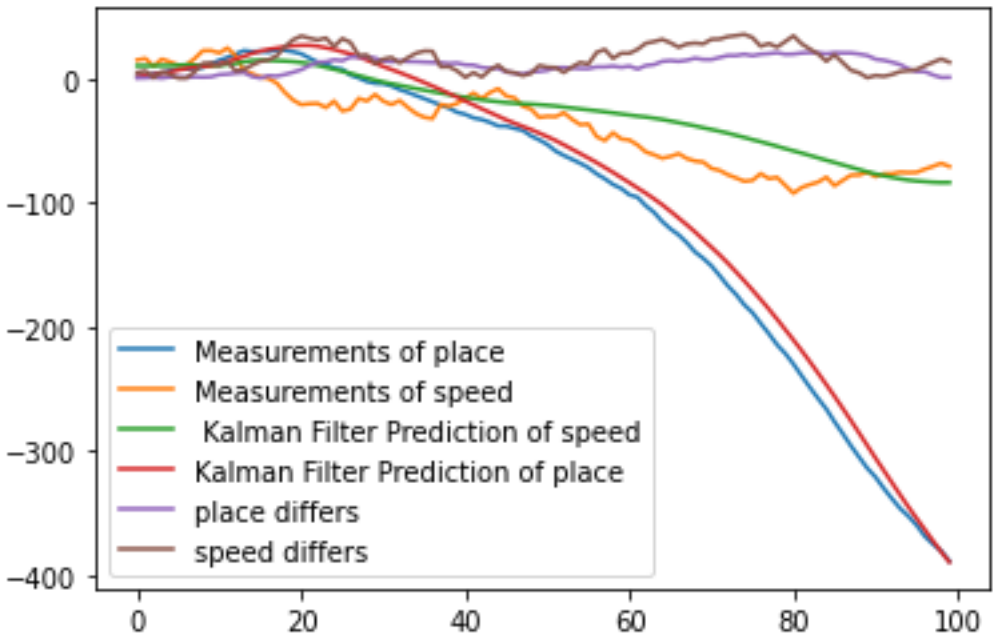

نمودار برای مکان اولیه ۰ و سرعت ثابت ۱۰ ، با واریانس نویز مکان = ۰.۲ و سرعت = ۲




نمودار برای مکان اولیه ۰ و سرعت ثابت ۱۰ ، با واریانس نویز مکان = ۰.۱ و سرعت =۱

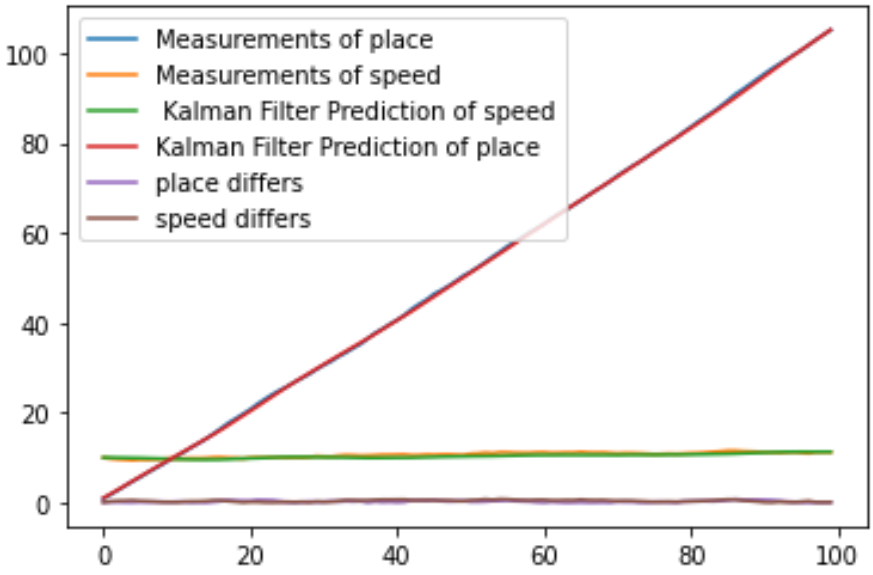
در کل هر چه نویز سرعت بالاتر است و پرشدت تر ، مکان هم از حالت خطی خارج می شود و کمی خمیده میشود، با سرعت منفی هم مکان معکوس میگردد و محرک به عقب بر میگردد

بالارفتن بیش از حد واریانس و در واقع شدت نویز حسگر مکان و سرعت باعث منفی شدن سرعت و بازگشت محرک هم میتواند بشود مثلا در شکل زیر نویز به قدری است که سرعت منفی شده ( در جهت عکس)



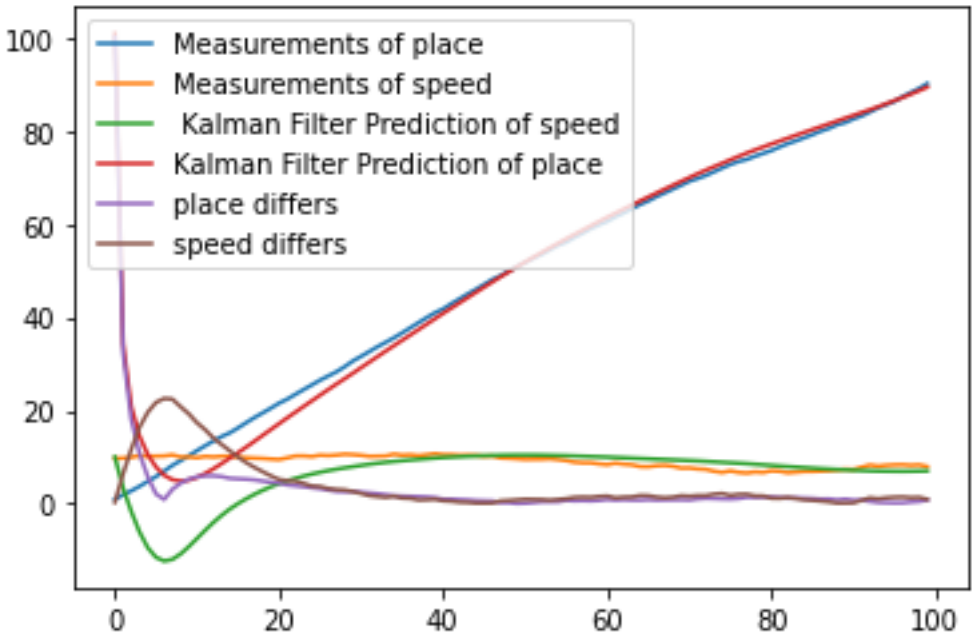

نمودار برای مکان اولیه ۰ و سرعت ثابت ۱۰ ، با واریانس نویز مکان = ۱ و سرعت =۴

و پایین بودن نویز تقریب درست تری از آنچه واقعا در حال رخ دادن است می دهد و اختلافات مشاهده و پیش بینی هم زیاد نیست .



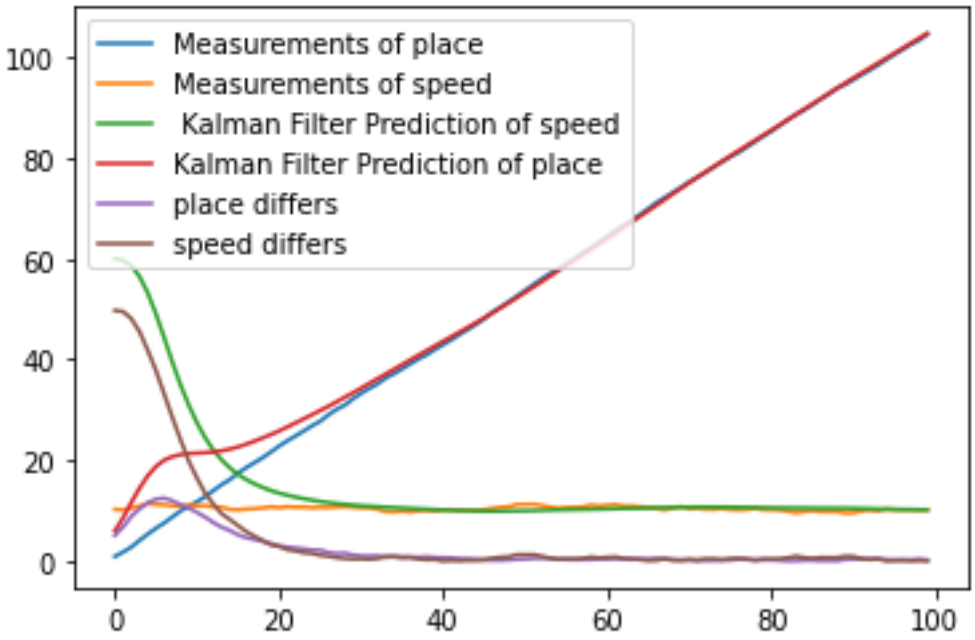

نمودار برای مکان اولیه ۰ و سرعت ثابت ۱۰ ، با واریانس نویز مکان = ۰.۱ و سرعت =۰.۱

در صورتی هم که مکان اولیه با واقعی متفاوت باشد و اختلاف زیاد باشد هم :




مشاهده می شود بعد از مدتی فیلتر کالمن به درستی تخمین میزند

این امر برای سرعت هم صدق می کند :




ولی خب در ابتدای امر اختلاف زیاد است و رفته رفته کاهش پیدا می کند .

لینک ها :

[GitHub - zziz/kalman-filter: Kalman Filter implementation in Python using Numpy only in 30 lines.]

[Kalman filter - Wikipedia]

در مورد فنر هم از قوانین زیر که در فایل پی دی اف پیوست امده است استفاده شد

$$F(t) - c\dot{x}(t) - kx(t) = m\ddot{x}(t)$$

$$m\ddot{x} = F - c\dot{x} - kx$$

$$\ddot{x} = \frac{1}{m}(F - c\dot{x} - kx)$$

Higher order differential equations can typically be reformulated into a system of first order differential equations

$x_1$= Position
$x_2$= Velocity/Speed

We set
$x = x_1$
$\dot{x} = x_2$

This gives:
$\dot{x}_1 = x_2$
$\dot{x}_2 = \ddot{x} = \frac{1}{m}(F - c\dot{x} - kx) = \frac{1}{m}(F - cx_2 - kx_1)$

Finally:

$$\ddot{x} = \frac{1}{m}(F - c\dot{x} - kx) \Rightarrow \boxed{\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= \frac{1}{m}(F - cx_2 - kx_1) \end{aligned}}$$

Given:

$$\dot{x}_1 = x_2$$
$$\dot{x}_2 = \frac{1}{m}(F - cx_2 - kx_1)$$

Using Euler:

$$\dot{x} \approx \frac{x(k+1) - x(k)}{T_s}$$

Then we get:

$$\frac{x_1(k+1) - x_1(k)}{T_s} = x_2(k)$$
$$\frac{x_2(k+1) - x_2(k)}{T_s} = \frac{1}{m}[F(k) - cx_2(k) - kx_1(k)]$$

This gives:

$$x_1(k+1) = x_1(k) + T_s x_2(k)$$
$$x_2(k+1) = x_2(k) + T_s \frac{1}{m}[F(k) - cx_2(k) - kx_1(k)]$$

Then we get:

$$x_1(k+1) = x_1(k) + T_s x_2(k)$$
$$x_2(k+1) = -T_s \frac{k}{m} x_1(k) + x_2(k) - T_s \frac{c}{m} x_2(k) + T_s \frac{1}{m} F(k)$$

Finally:

$$x_1(k+1) = x_1(k) + T_s x_2(k)$$
$$x_2(k+1) = -T_s \frac{k}{m} x_1(k) + (1 - T_s \frac{c}{m}) x_2(k) + T_s \frac{1}{m} F(k)$$

Discrete System:

$$x_1(k+1) = x_1(k) + T_s x_2(k)$$
$$x_2(k+1) = -T_s \frac{k}{m} x_1(k) + (1 - T_s \frac{c}{m}) x_2(k) + T_s \frac{1}{m} F(k)$$

$$A = \begin{bmatrix} 1 & T_s \\ -T_s \frac{k}{m} & 1 - T_s \frac{c}{m} \end{bmatrix}$$

We can set it on Discrete state space form:

$$x(k+1) = A_d x(k) + B_d u(k)$$

$$B = \begin{bmatrix} 0 \\ T_s \frac{1}{m} \end{bmatrix}$$

This gives:

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \end{bmatrix} = \begin{bmatrix} 1 & T_s \\ -T_s \frac{k}{m} & 1 - T_s \frac{c}{m} \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} + \begin{bmatrix} 0 \\ T_s \frac{1}{m} \end{bmatrix} F(k)$$

$$x(k) = \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix}$$

و مقادیر را به صورت زیر ست میکنیم :

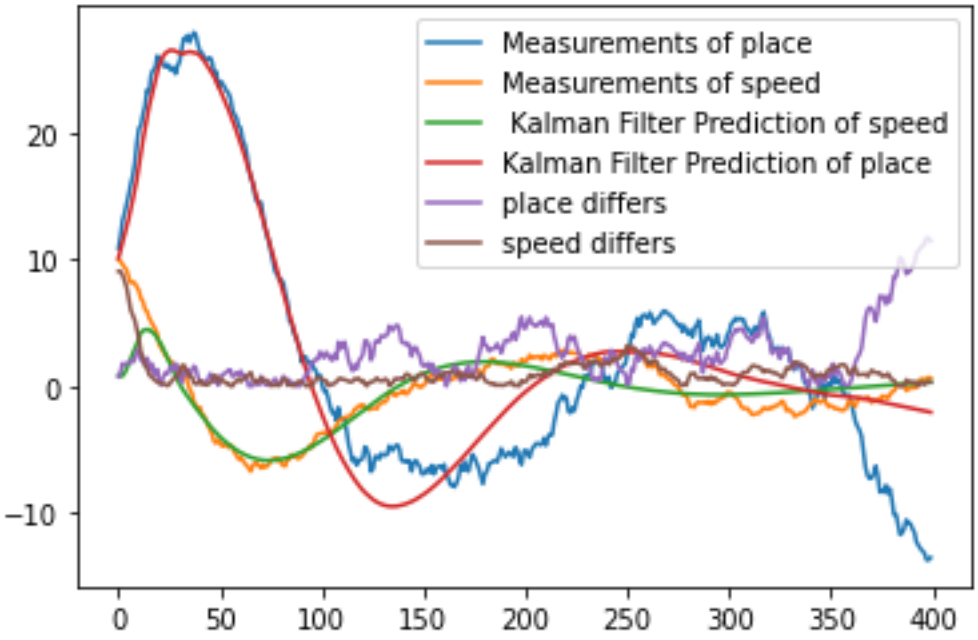
نیروی F برابر ۵ نیوتون ، جرم برابر ۲۰ کیلوگرم ، ضریب سختی فنر برابر ۲ ، ثابت دامپر = ۲ و تغییرات را در هر ۰.۱ ثانیه می سنجیم ، و جمعه ۴۰۰ گام خواهیم رفت ، کل داستان همانند قست قبل است فقط این بار نیروی F را در این قسمت از ورودی می دهیم و در متغیر U :

```
def predict(self, u = 5 ):
    self.x = np.dot(self.F, self.x) + np.dot(self.B, u)
    self.P = np.dot(np.dot(self.F, self.P), self.F.T) + self.Q
    return self.x
```

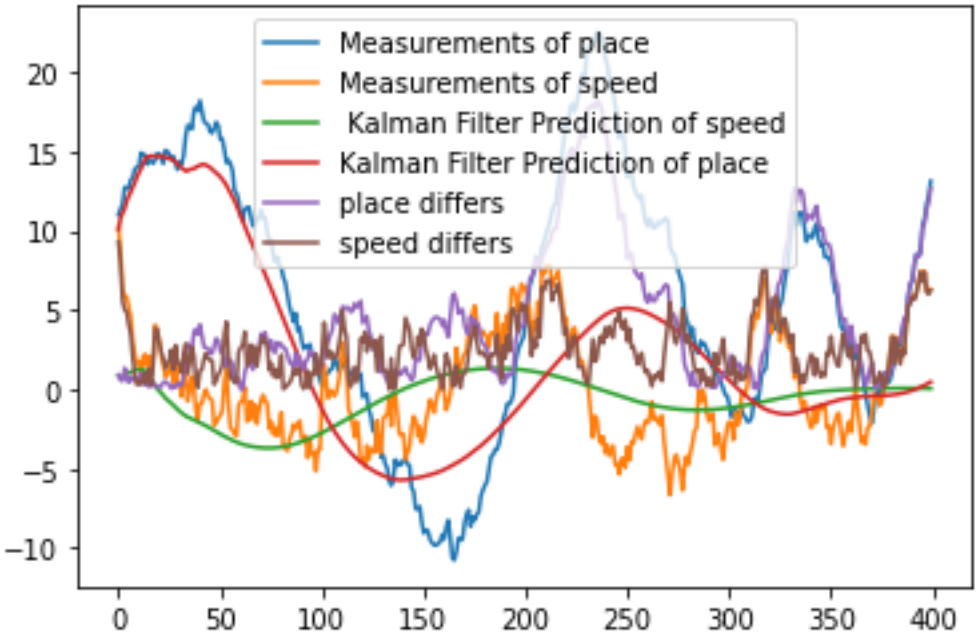
بقیه قسمت ها دقیقا همانند قبل است و توضیح نمیدهیم وهمان کارهای قبل را انجام میدهیم تا فیلتر کالمن را به دست اوریم :

```
dt = 0.1
k =2
m = 20
F = 5
c = 4
F = np.array([[1, dt], [(dt * (-k) )/m, 1 - ((dt *c) / m)]])
B= np.array([0 , dt / m])
H = np.array([1, 0]).reshape(1, 2)
Q = np.array([[0,0], [0,  0.5 * (dt**2)/(m ** 2)]])
R = np.array([0.5]).reshape(1, 1)
newmesuer = [ 0 , 10.]
kf = KalmanFilter(F = F, H = H, Q = Q, R = R , B=B)
predictions_place = []
predictions_speed = []
measurementsofplace = []
measurementsofspeed = []
noise1 =np.random.normal(0, 0.1, 400)
print(noise1)
noise2 = np.random.normal(0, 0.2, 400)
for k in range(400):
  newmesuer = np.dot(F, newmesuer) + [noise1[k] , noise2[k]]
  print(newmesuer)
  measurementsofplace.append(newmesuer[0])
  measurementsofspeed.append(newmesuer[1])
```

مشاهده می شود که به مرور زمان فنر به نقطه آزاد می رسد و از نوسان می افتد ( طول این نقطه را صفر در نظر گرفتیم و منحنی قرمز بیانگر آن است ) ، سرعت هم میراست و پس از مدتی صفر می شود ( خط سبز) ، سایر نمودار ها هم ورودی همراه با نویز (آبی پررنگ برای مکان و نارنجی برای سرعت ) و تفاوت ها در رنگ ها قهوه ای و بنفش است ، لازم به ذکر است مکان منفی به معنای جمع شدن فنر است و سرعت فنر هم اگر منفی باشد یعنی فنر دارم جمع می شود .

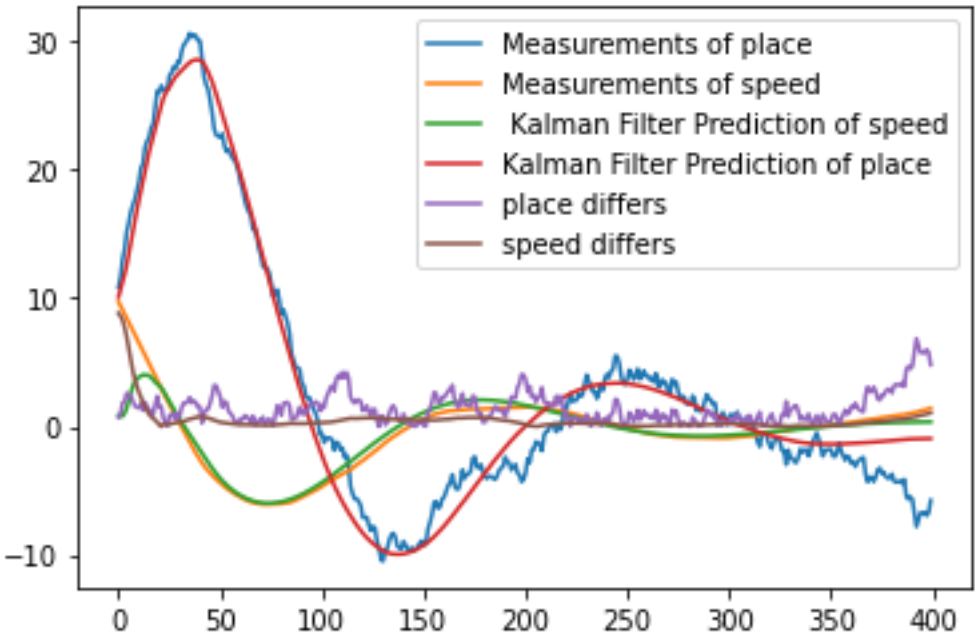



مکان اولیه = ۱۰ ، سرعت = ۱۰ ، واریانس نویز مکان = ۰.۵ ، واریانس نویز سرعت = ۰.۲



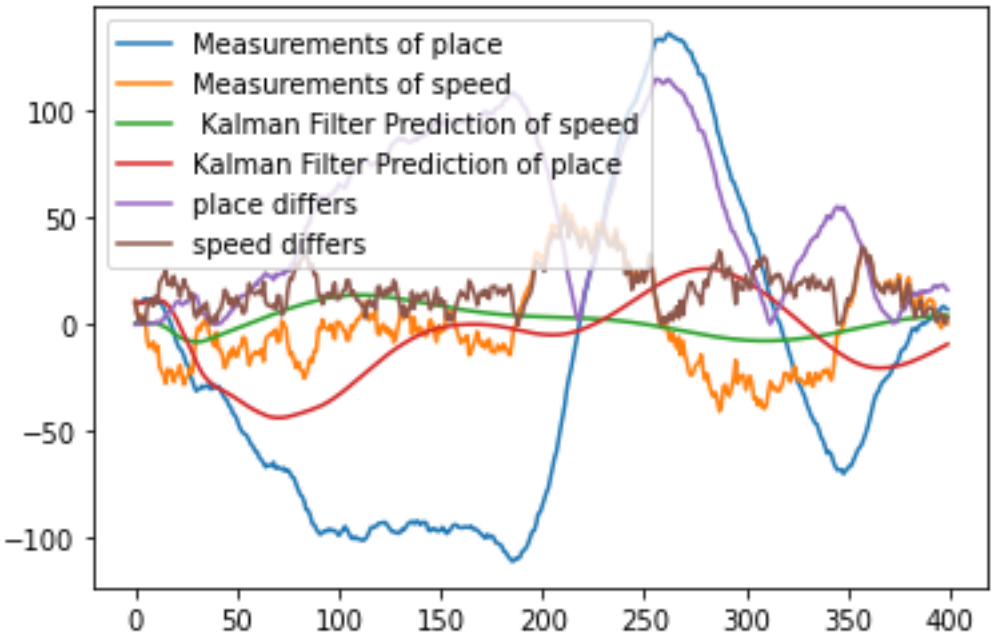

مکان اولیه = ۱۰ ، سرعت = ۱۰ ، واریانس نویز مکان = ۰.۵ ، واریانس نویز سرعت = ۱

نویز بالا باعث افزایش خطا می شود ولی در نهایت کالمن به ۰ همگرا میشود در یک فرایند نویزی .



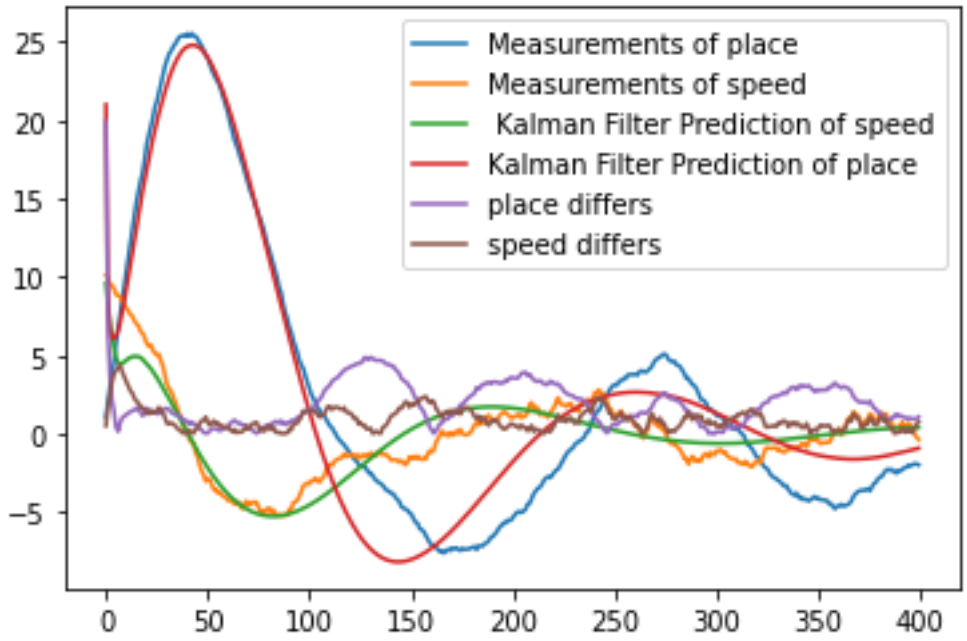

مکان اولیه = ۱۰ ، سرعت = ۱۰ ، واریانس نویز مکان = ۰.۵ ، واریانس نویز سرعت = ۰.۰۱

همانطور که میبینید در شکل بالا نویز سرعت بسیار کم است و دو منحنی نارنجی و سبز تقریبا با هم مشابه اند .




مکان اولیه = ۱۰ ، سرعت = ۱۰ ، واریانس نویز مکان = ۱ ، واریانس نویز سرعت = ۴

حتی در شکل بالا مشاهده می شود که نویز بالا باعث شده منحنی سینوسی بودن در بیایید و داخل قوس قوسی پیدا شود .




شروع از نکته دیگر و اختلاف بین واقعیت  و پیش بینی هم اگر چه در ابتدا خطای زیادی به همراه دارد اما همانند تصویر بالا و پایین مشخص است که به مرور زمان کالمن به درستی تخمین میزند، تصویر بالا برای خطا در مکان است و تصویر پایین برای خطا در سرعت :