Dokumentasjon for Oblig 1: Kalman Filter

Valg av filter:

Til denne oppgaven kunne man få et brukbart resultat både ved bruk av alpha-beta, alpha-beta-gamma eller et Kalman filter. Til denne oppgaven valgte jeg å benytte et alpha-beta-gamma filter siden det basert på min vurdering ga det beste resultatet med tanke på prosent treff og tidsbruk for å implementere et Kalman filter. I tillegg var det ikke strengt talt forventet at man skulle bruke et Kalman filter. Alpha-beta-gamma filter ga et bra resultat som det vil bli skrevet mere om senere i rapporten.

Likningene som ble brukt i filteret:

Siden det ble brukt et alpha-beta-gamma filter så ble to sett med likninger tatt i bruk, som begge inneholder utregninger for både posisjon, fart og akselerasjon. Disse to er State Update Equation og State Extrapolation Equation.

State Update Equation:

Likning for å estimere nåværende posisjon:

$$\hat{x}_{n,n} = \hat{x}_{n,n-1} + \alpha * (z_n - \hat{x}_{n,n-1})$$

Her bruker man forrige prediksjon + alpha verdi ganget med målingen man har fått minus forrige prediksjon.

Likning for å estimere nåværende fart:

$$\hat{\dot{x}}_{n,n} = \hat{\dot{x}}_{n,n-1} + \beta * \left(\frac{z_n - \hat{x}_{n,n-1}}{\Delta t}\right)$$

Nåværende estimat for fart kalkuleres ved bruk av forrige prediksjon av farten + beta ganget med posisjonsmålingen – forrige estimat av posisjon delt på tidsintervallet

Likning for å estimere nåværende akselerasjon:

$$\hat{\ddot{x}}_{n,n} = \hat{\ddot{x}}_{n,n-1} + \gamma * \left(\frac{z_n - \hat{x}_{n,n-1}}{0.5 * \Delta t^2}\right)$$

Nåværende estimat av akselerasjons kalkuleres ved å bruke forrige prediksjon av akselerasjonen + en gamma verdi ganget med det samme som i forrige ligning bortsett fra at det blir delt på 0,5 ganget med tidsintervallet som er opphøyd i andre.

State Extrapolation Equation:

Likning for å predikere neste posisjon:

$$\hat{\mathrm{x}}_{\mathrm{n+1,n}} = \hat{x}_{n,n} + \hat{\dot{x}}_{n,n} * \Delta t + \ddot{x}_{n,n} * \frac{\Delta t^2}{2}$$

Likningen predikerer neste posisjonen ved å bruke nåværende estimat av posisjon + nåværende estimatet av farten ganget med tidsintervallet pluss nåværende estimat av akselerasjonen ganget med tidsintervallet kvadrert delt på to.

Likning for å predikere neste fart:

$$\hat{\dot{\mathrm{x}}}_{\mathrm{n+1,n}} = \hat{\dot{x}}_{n,n} + \ddot{x}_{n,n} * \Delta t$$

Likingen predikerer den neste farten ved å bruke nåværende estimat av fart pluss nåværende estimat av akselerasjon ganget med tidsintervallet.

Likning for å predikere neste akselerasjon:

$$\hat{\ddot{\mathrm{x}}}_{\mathrm{n+1,n}} = \hat{\ddot{x}}_{n,n}$$

For å predikere neste akselerasjon bruker vi bare det nåværende estimatet av akselerasjonen.

Valg av initialverdier og alpha beta gamma verdier:

Valget av disse verdiene har vært en kombinasjon av å velge tall som er direkte oppgitt i oppgavebeskrivelsen/koden, eller basert på hvor støyfull målingene vi får sendt fra pygame er.

De første verdiene som er satt som x_current, v_current og a_current er direkte basert på hvordan pygame sitt koordinatsystem fungerer og med tanke på de verdiene som er gitt i koden. Disse verdiene er som følgende:

```
self.x_current = 0
self.v_current = 0.3
self.dt = 1
self.a_current = 0.003
self.alpha = 0.5
self.beta = 0.0001
self.gamma = 0.00001

self.x_pred = self.x_current + self.v_current*self.dt + self.a_current*((self.dt**2)*0.5)
self.v_pred = self.v_current + self.a_current*self.dt
self.a_pred = self.a_current
```

x_current er satt til 0 som initialverdi med tanke på at klossen som skal treffes starter sin vei i x koordinat lik 0. v_current satt til 0.3 og a_current satt til 0.003. Disse verdiene er satt på bakgrunn av farts og akselerasjonsverdiene vi får oppgitt i HitTheTarget.py

alpha, beta og gamma verdiene er satt i henhold til støyen i målingen og det som har vist seg til å gi et godt resultat som generelt sett ligger mellom 88% og 91%.

Som initialverdier til neste prediksjon av posisjon, fart og akselerasjon blir State Extrapolation Equation kjørt i init metoden når Kalman klassen blir kalt. Dette blir gjort i henhold til beskrivelsen av alpha beta gamma filter på siden KalmanFilter.net. Ved å kjøre denne prediksjonen får vi en god del mere prosent treffrate på filteret vårt siden vi har en litt bedre prediksjon på neste måling enn å bare for eksempel sette disse til initialverdiene x_current, v_current og a_current som det ble forsøkt å gjøre tidligere.

Beskrivelse av funksjonen estimate_current_position_and_velocity:

State Update Equation:

```
 # State update equation
self.x_current = self.x_pred + self.alpha*(zi - self.x_pred)
self.v_current = self.v_pred + self.beta*((zi-self.x_pred)/self.dt)
self.a_current = self.a_pred + self.gamma*((zi-self.x_pred)/(0.5*(self.dt**2)))
```

Koden for state Update equation blir i funksjonen kjørt før state extrapolation equation i henhold til beskrivelsen av et a-b-y filter. Her brukes de prediksjonsverdiene som blir regnet ut i init metoden til klassen.

State Extrapolation Equation

```
 # State extrapolation equation
self.x_pred=self.x_current +self.v_current*self.dt+self.a_current*((self.dt**2)*0.5)
self.v_pred = self.v_current + self.a_current*self.dt
self.a_pred = self.a_current
```

Ligningene er det samme som i init metoden bortsett fra at nå regner man med de verdiene man har regnet ut ovenfor. Disse verdiene vil så bli brukt i neste iterasjon av State Update Equation

Resultater

Resultatene jeg har fått fra min implementasjon har vært nokså bra og ligger generelt sett som sagt rundt 88 til 92%. Under har jeg en utregning av det gjennomsnittlige resultatet av filteret basert på 3 forskjellige game loops hvor alle har kjørt ca 500 iterasjoner.

Første game loop resultat:

```
Hit rate after 500 iterations:
Without filter: 6.2 %
With filter:    92.2 %
(IntroAI) PS C:\Users\isakh\Desktop\Intro AI og ML>
```

Andre game loop resultat:

```
Hit rate after 500 iterations:
Without filter: 7.6 %
With filter:    88.4 %
(IntroAI) PS C:\Users\isakh\Desktop\Intro AI og ML>
```

Tredje game loops resultat:

```
Hit rate after 500 iterations:
Without filter: 5.0 %
With filter:    92.0 %
(IntroAI) PS C:\Users\isakh\Desktop\Intro AI og ML>
```

Dette kan muligens være litt for få og faktisk få en nøyaktig gjennomsnittlig verdi, men det viser fortsatt til en viss grad hvordan resultatet ligger mellom 88 og 92%.

Gjennomsnittet: (92,2 + 88,4 + 92,0) / 3 = 90,8666667%