732A96: Advance Machine Learning

LAB 4: STATE SPACE MODELS

Arian Barakat, ariba405
Rebin Hosini, rebho150
Hao chi Kiang, haoki222
Joshua Hudson, joshu107
Carles Sans Fuentes, carsa564

Background:

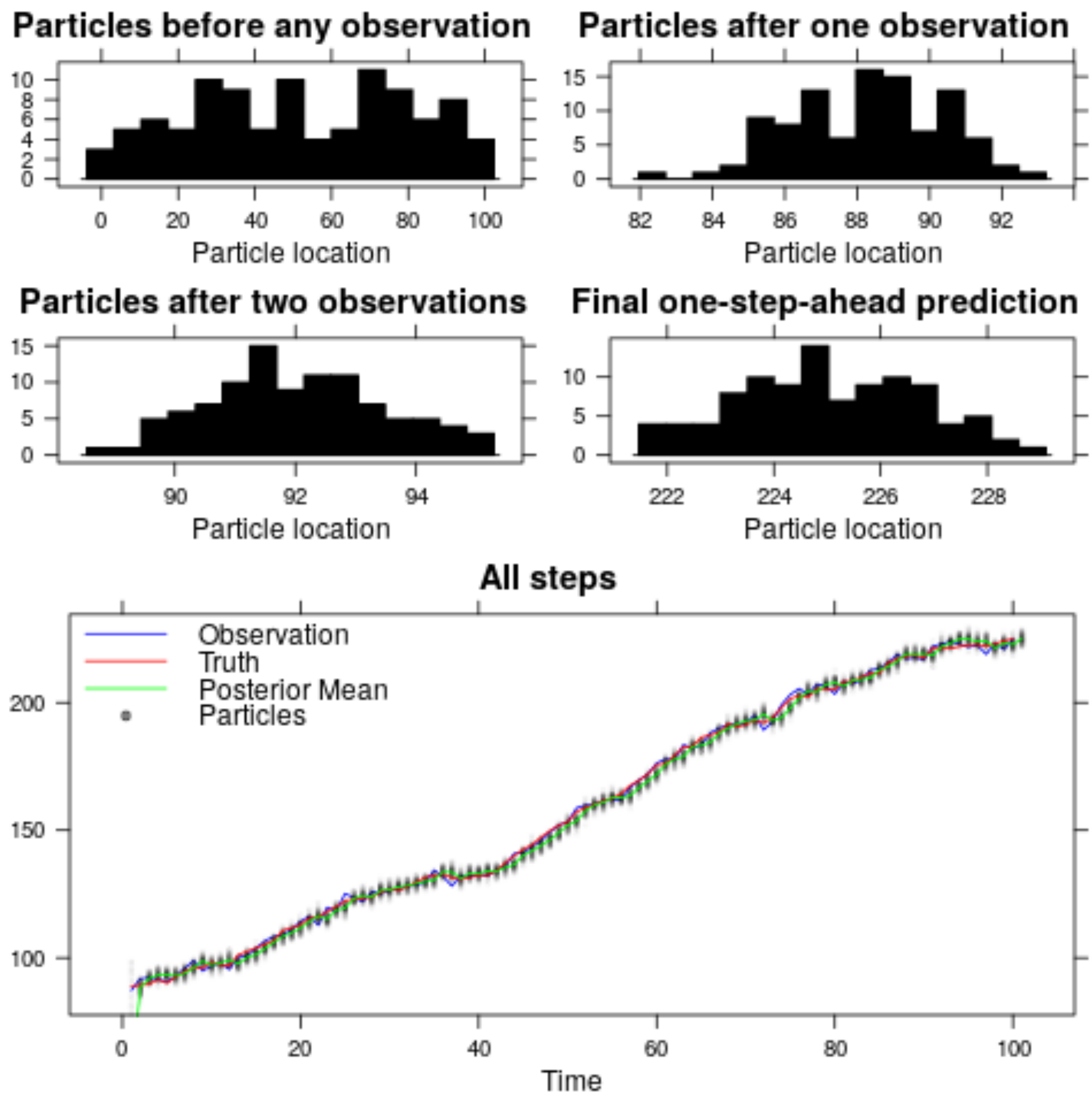
The purpose of the lab is to put in practice some of the concepts covered in the lectures. To do so, you are asked to implement the particle filter for robot localization. For the particle filter algorithm, please check Section 13.3.4 of Bishop's book and/or the slides for the last lecture on SSMs. The robot moves along the horizontal axis according to the following SSM:

See instruction sheet for model specification

Implement the SSM above. Run it for $T = 100$ time steps to obtain $z_{1:100}$ (i.e. states) and $x_{1:100}$ (i.e. observations). Use the observations (i.e. sensor readings) to identify the state (i.e. robot location) via particle filtering. Use 100 particles. For each time step, show the particles, the expected location and the true location. Repeat the exercise after replacing the standard deviation of the emission model with 5 and then with 50. Comment on how this affects the results. Finally, show and explain what happens when the weights in the particle filter are always equal to 1, i.e. there is no correction.

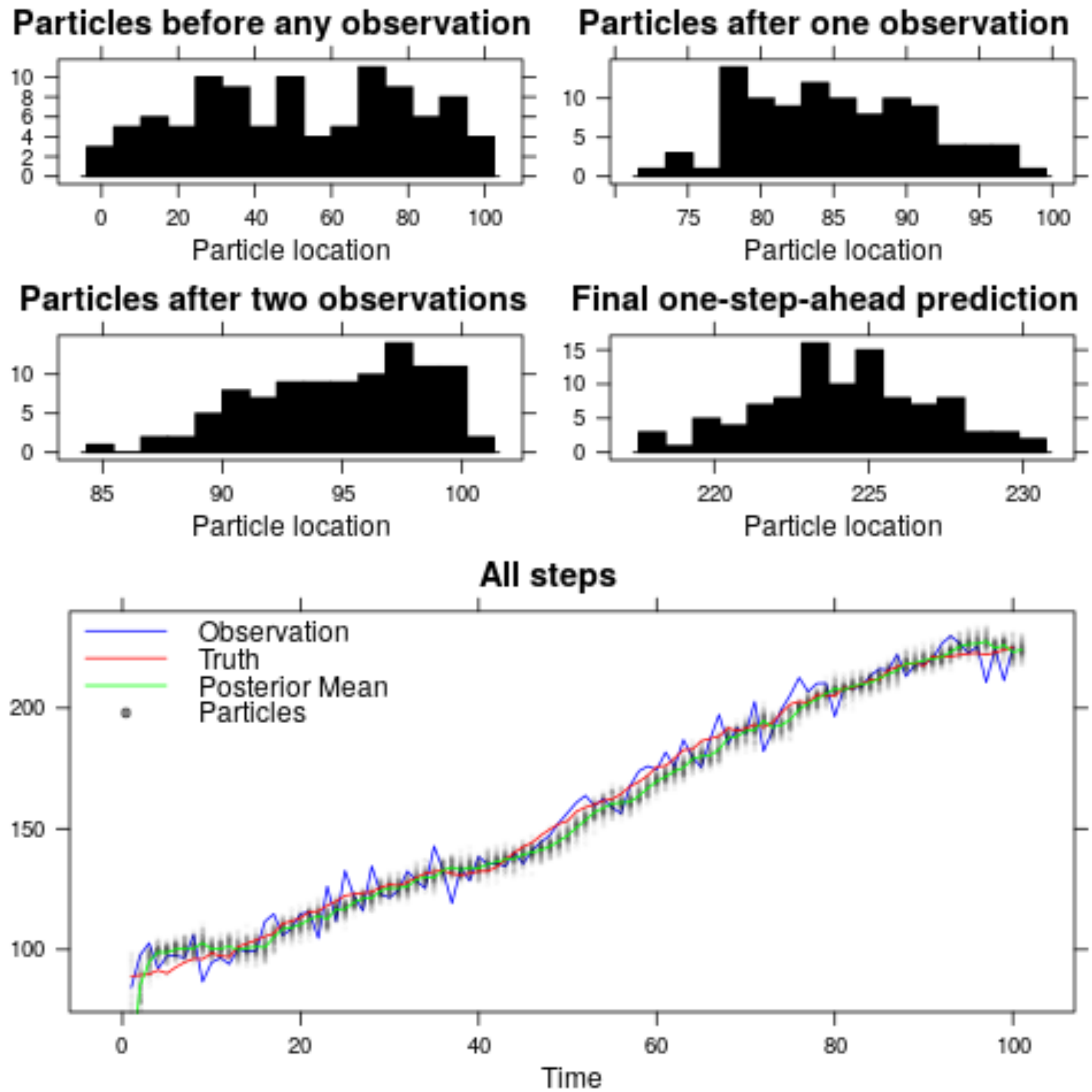
Standard Deviation: 1

Animation for Standard Deviation = 1



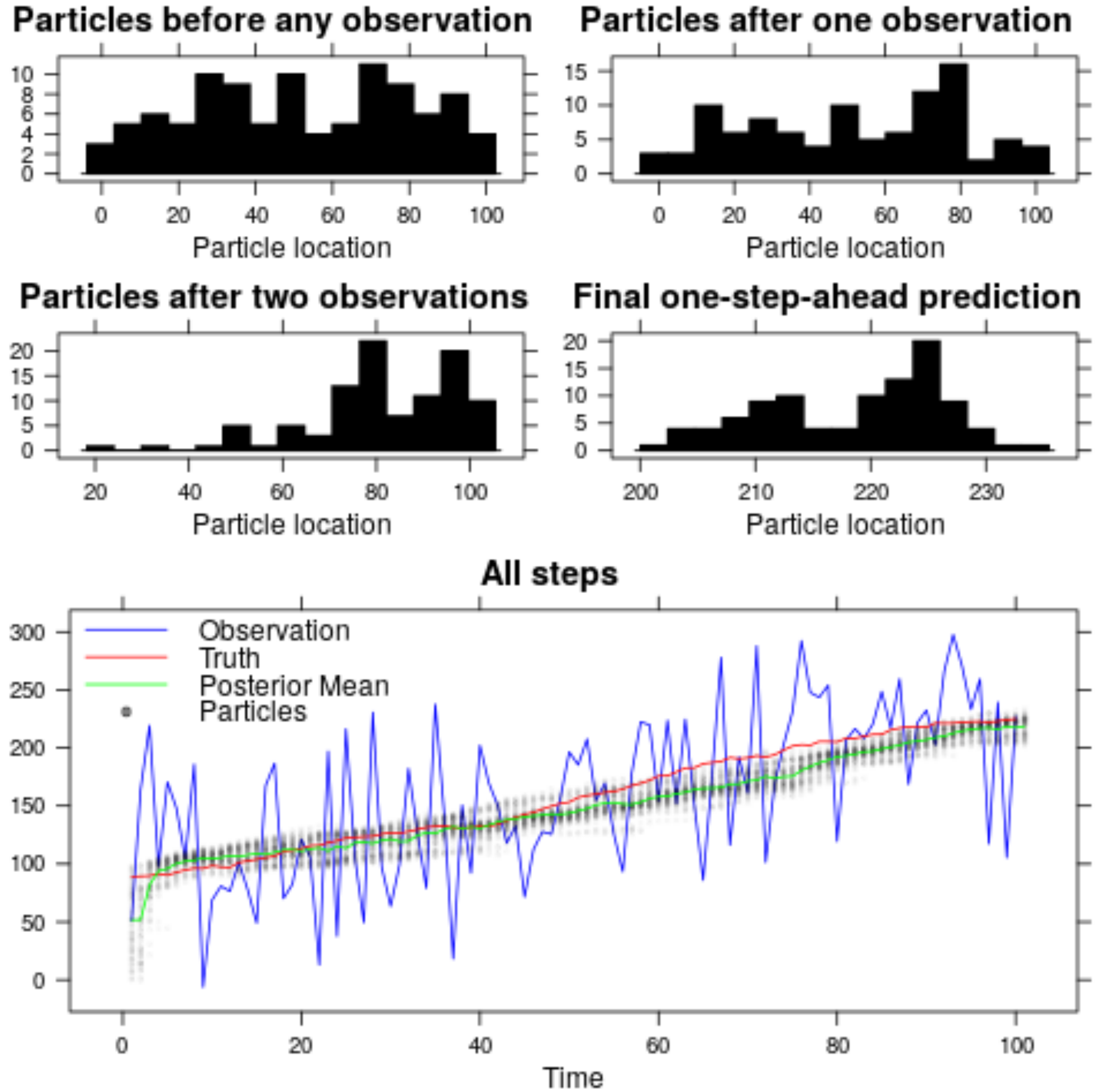
Standard Deviation: 5

Animation for Standard Deviation = 5



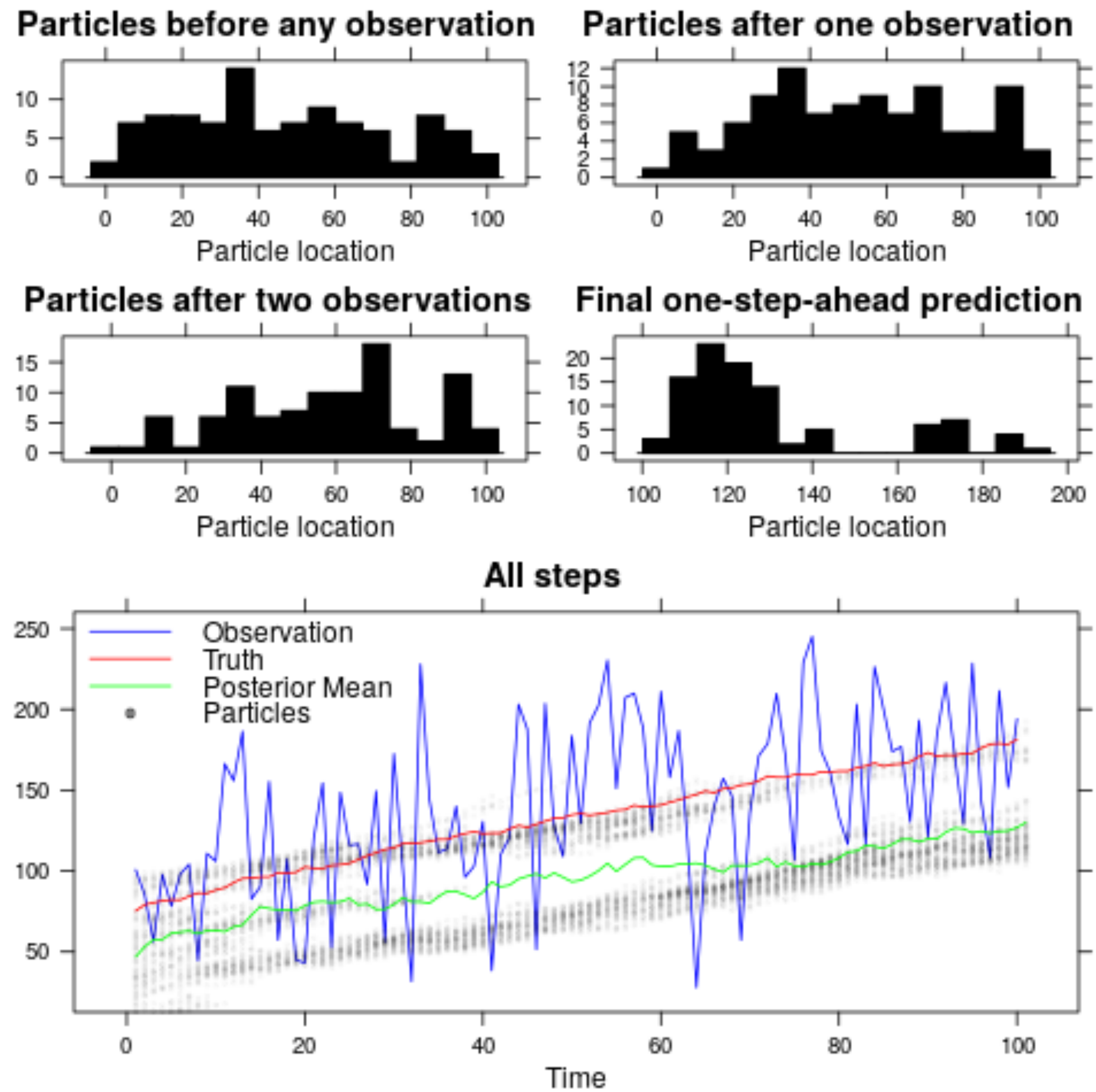
Standard Deviation: 50

Animation for Standard Deviation = 50

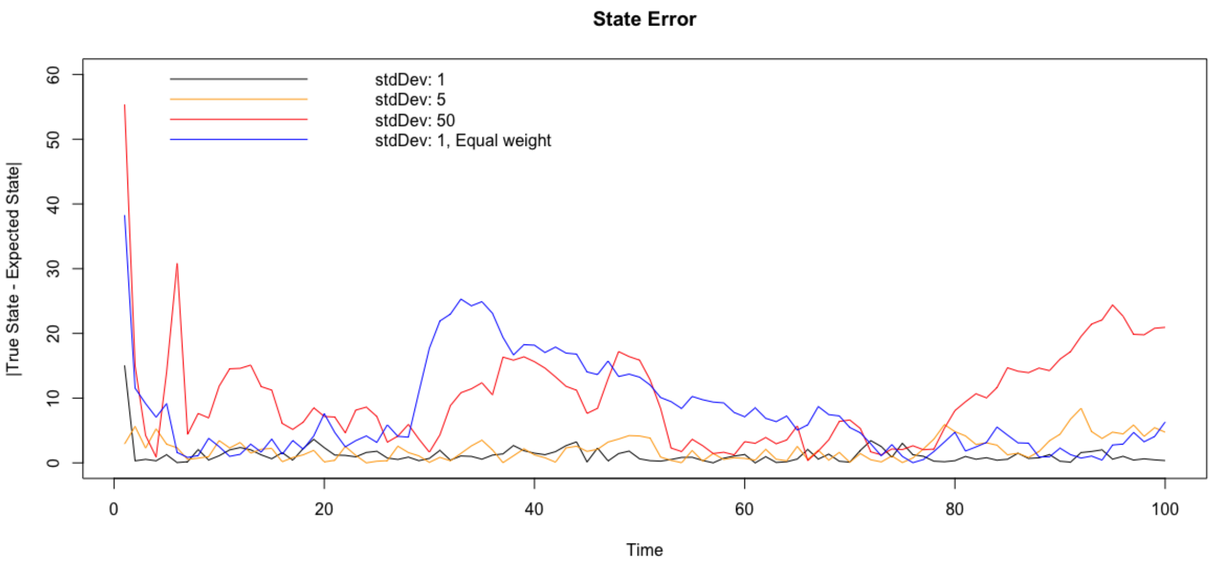


Equal Weight

Animation for Standard Deviation = 1, Weight Equal



State Error


State Error
stdDev: 1
stdDev: 5
stdDev: 50
stdDev: 1, Equal weight
|True State - Expected State|
0
10
20
30
40
50
60
0
20
40
60
80
100
Time

Discussion

By increasing the standard deviation for the emission model we're implying that we're less certain about the sensor measurements. This, in turn, will affect the weight distribution of our particles (distant particles from the true state will be given a higher weight compared a to a simulation with a lower standard deviation), which affect the position of the expected state. The consequence of this can be viewed in the *state error* figure, where we can observe that the absolute difference between the true and expected state increases as we increase the standard deviation.

In the case when we're giving all particles equal weight the problem of absolute difference is similarly ill-behaved as in the case when the standard deviation is set to 50. Once again, the behavior can be linked to the fact that distant particles (from the true state) have a relatively high weight.

Appendix

```r
knitr::opts_chunk$set(echo = FALSE,
                      warning = FALSE,
                      message = FALSE,
                      fig.height = 11,
                      fig.width = 9,
                      eval = FALSE)

library(ggplot2)
library(dplyr)
library(grid)
library(gridExtra)
library(knitr)
library(png)


img <- readPNG("../img/sd_1.png")
grid.raster(img)
img <- readPNG("../img/sd_5.png")
grid.raster(img)
img <- readPNG("../img/sd_50.png")
grid.raster(img)
img <- readPNG("../img/constant-weight.png")
grid.raster(img)
img <- readPNG("../img/errorPlot.png")
grid.raster(img)


sigma <- 1
sigmaE <- 1 #change sigma for each

sTransModel <- function(sd = 1, z_p){
  pos <- sample(1:3, size = 1, prob = c(1/3,1/3,1/3))
  dist <- list("d1" = rnorm(1, mean = z_p, sd = sd),
               "d2" = rnorm(1, mean = z_p+1, sd = sd),
               "d3" = rnorm(1, mean = z_p+2, sd = sd))
 return(dist[[pos]])
}

sEmissModel <- function(sd = sigmaE, z_t){
  pos <- sample(1:3, size = 1, prob = c(1/3,1/3,1/3))
  dist <- list("d1" = rnorm(1, mean = z_t, sd = sd),
               "d2" = rnorm(1, mean = z_t+1, sd = sd),
               "d3" = rnorm(1, mean = z_t-2, sd = sd))
 return(dist[[pos]])
}

emissModel <- function(sd = sigmaE, z_t, x_t){
    (dnorm(x_t, mean = z_t , sd = sd)+
     dnorm(x_t, mean = z_t + 1 , sd = sd)+
     dnorm(x_t, mean = z_t - 1, sd = sd))/3
}
```

```r
init <- function(n = 1, min = 1, max = 100){
  runif(n = n, min = min, max = max)
}

steps <- 100
z <- c()
x <- c()
z[1] <- init() # z_1
x[1] <- sEmissModel(z_t = z[1])

for (i in 2:steps){
  z[i] <- sTransModel(z_p = z[i-1], sd = sigma) # z_t
  x[i] <- sEmissModel(z_t = z[i], sd = sigmaE) # x_t

}

PF <- function(steps, sd = c(sigma,sigmaE)){
  weights <- matrix(0, ncol = steps, nrow = steps)
  Z <- matrix(ncol = steps, nrow = steps+1)
  Z[1,] <- init(n = steps, 0, 100)
  for (i in 1:steps){
    emission <- vapply(Z[i,, drop = TRUE],
                       FUN = function(zVal){emissModel(sd = sd[2], z_t = zVal, x_t = x[i])},
                       FUN.VALUE = numeric(1))
    weights[i,] <- emission/sum(emission)
    #Sample from current Z with probabilities weighted Z
    WZ <- sample(Z[i,], replace = TRUE, prob = weights[i,, drop = TRUE], size = steps)
    Z[i+1,] <- sapply(WZ, function(zprev) {sTransModel(sd = sd[1], z_p = zprev)})
  }
  return(list("Z" = Z, "W" = weights, "X" = x))
}

PFW <- function(steps, sd = c(sigma, sigmaE)){
  weights <- matrix(0.01, ncol = steps, nrow = steps)
  Z <- matrix(ncol = steps, nrow = steps+1)
  Z[1,] <- init(n = steps, 0, 100)
  for (i in 1:steps){
    emission <- vapply(Z[i,, drop = TRUE],
                       FUN = function(zVal){emissModel(sd = sd[2], z_t = zVal, x_t = x[i])},
                       FUN.VALUE = numeric(1))
    #Sample from current Z with probabilities weighted Z
    WZ <- sample(Z[i,], replace = TRUE, prob = weights[i,, drop = TRUE], size = steps)
    Z[i+1,] <- sapply(WZ, function(zprev) {sTransModel(sd = sd[1], z_p = zprev)})
  }
  return(list("Z" = Z, "W" = weights, "X" = x))
}

plotData <- function(outModel){
  E <- apply(outModel$W*outModel$Z[-nrow(outModel$Z),], MARGIN = 1, sum)
  xPos <- matrix(ncol = 2, nrow = 100)
  xPos[,1] <- outModel$X #Do not forget to change sigma outside of the PF fun
  xPos[,2] <- 0.2
  return(list("pos" = xPos, "E" = E))
```

```r
}

############################ Run robot

plotRobot <- function(outModel,steps = 100){
  o <- outModel
  X <- o$X
  Z <- o$Z
  for (i in 1:steps){
    plot(x = 1:250, y = rep(0,250), type = "l",
         col = "white",ylab = "y", xlab = "x")
    abline(h = 0, col = "black")
    #Expected
    points(x = plotData(o)$E[i], y = 0.8, type = "o")
    text(x = plotData(o)$E[i], y = 0.8, labels = "Expected", cex = 1)
    #True robot state
    abline(v = plotData(o)$pos[i], col = "orange")
    text(x = plotData(o)$pos[i], y = 0.11, labels = "Robot", cex = 1, col = "orange")
    #Particles
    points(x = Z[i,], y = rep(-0.1,length(Z[i,])), col = "red", pch = 16)
    Sys.sleep(0.3)
  }
}
#debugonce(PF)
set.seed(123456)
out <- PF(steps = 100)
outW <- PFW(steps = 100)


plotError <- function(object, add = FALSE, lineCol = NULL){
  steps <- length(object[["trueStates"]])
  if(!add){
    plot(x = 1:steps,
         y = abs(object$trueStates - object$exptectedState),
         type = "l",
         xlab = "Time",
         ylab = "|True State - Expected State|",
         main = "State Error",
         ylim = c(0,60))
  }

  if(add){
    if(is.null(lineCol)){
      stop("Provide line colour")
    }
    lines(x = 1:steps,
         y = abs(object$trueStates - object$exptectedState),
         col = lineCol)
  }

}
```

```r
# Error Plot

plotError(simSSMsd1)
plotError(simSSMsd5, add = TRUE, lineCol = "orange")
plotError(simSSMsd50, add = TRUE, lineCol = "red")
plotError(simSSMsd1_equalWeight, lineCol = "blue", add = TRUE)
legend("topright",
       legend = c("stdDev: 1",
                  "stdDev: 5",
                  "stdDev: 50",
                  "stdDev: 1, Equal weight"),
       lty = rep(1, times = 4),
       col = c("black", "orange", "red", "blue"),
       bty = "n")
```