# lab2_exam

*Emil K Svensson*

*16 October 2017*

## Question 1

```
hidden_states <- paste("z",1:10,sep = "")
observed_states <- paste("x",1:10,sep = "")
start <- rep(0.1,10)

#Just to see the structure
#initHMM(states, observed_states)

# The transition probabilities, since we only can move
trans <- matrix(0,ncol = 10, nrow = 10)
diag(trans) <- 0.5
diag(trans[,-1]) <- 0.5
trans[10,1] <- 0.5
#trans[1,10] <- 0.5

# Making sure the probabilities sum to 1 in each row
#apply(trans, MARGIN = 1, FUN = sum)
#rep(1:5, each = 2)

# The Emission probabilities are our uncertaintiec in the position of the robot. Basicly the observatio

emission <-
  diag(1/5, 10)[, c(3:10, 1:2)] +
  diag(1/5, 10)[, c(2:10, 1)] +
  diag(1/5, 10) +
  diag(1/5, 10)[, c(10, 1:9)] +
  diag(1/5, 10)[, c(9:10, 1:8)]

colnames(emission) <- paste("x",1:10,sep = "")
rownames(emission) <- paste("z",1:10,sep = "")

rownames(trans) <- paste("z",1:10,sep = "")
colnames(trans) <- paste("z",1:10,sep = "")
```

```
# The different places the robot can be in
observed_states
```

```
##  [1] "x1"  "x2"  "x3"  "x4"  "x5"  "x6"  "x7"  "x8"  "x9"  "x10"
```

```
# The Hidden States
hidden_states
```

```
##  [1] "z1"  "z2"  "z3"  "z4"  "z5"  "z6"  "z7"  "z8"  "z9"  "z10"
```

```
# Where they start
start
```

```
##  [1] 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1
```

```
# Transition matrix is the probabilities the robot will move
trans
```

```
##      z1  z2  z3  z4  z5  z6  z7  z8  z9 z10
## z1  0.5 0.5 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
## z2  0.0 0.5 0.5 0.0 0.0 0.0 0.0 0.0 0.0 0.0
## z3  0.0 0.0 0.5 0.5 0.0 0.0 0.0 0.0 0.0 0.0
## z4  0.0 0.0 0.0 0.5 0.5 0.0 0.0 0.0 0.0 0.0
## z5  0.0 0.0 0.0 0.0 0.5 0.5 0.0 0.0 0.0 0.0
## z6  0.0 0.0 0.0 0.0 0.0 0.5 0.5 0.0 0.0 0.0
## z7  0.0 0.0 0.0 0.0 0.0 0.0 0.5 0.5 0.0 0.0
## z8  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.5 0.5 0.0
## z9  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.5 0.5
## z10 0.5 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.5
```

```
# Emission matrix states the uncertainties we have about the robots position
emission
```

```
##      x1  x2  x3  x4  x5  x6  x7  x8  x9 x10
## z1  0.2 0.2 0.2 0.0 0.0 0.0 0.0 0.0 0.2 0.2
## z2  0.2 0.2 0.2 0.2 0.0 0.0 0.0 0.0 0.0 0.2
## z3  0.2 0.2 0.2 0.2 0.2 0.0 0.0 0.0 0.0 0.0
## z4  0.0 0.2 0.2 0.2 0.2 0.2 0.0 0.0 0.0 0.0
## z5  0.0 0.0 0.2 0.2 0.2 0.2 0.2 0.0 0.0 0.0
## z6  0.0 0.0 0.0 0.2 0.2 0.2 0.2 0.2 0.0 0.0
## z7  0.0 0.0 0.0 0.0 0.2 0.2 0.2 0.2 0.2 0.0
## z8  0.0 0.0 0.0 0.0 0.0 0.2 0.2 0.2 0.2 0.2
## z9  0.2 0.0 0.0 0.0 0.0 0.0 0.2 0.2 0.2 0.2
## z10 0.2 0.2 0.0 0.0 0.0 0.0 0.0 0.2 0.2 0.2
```

With everything defined we can initialize the HMM-robot.

```
robot <- initHMM(hidden_states,observed_states,start,trans,emission)
```

## Question 2

The function below is built for answering Questions 2 to 7

```
mrRobot <- function(hmm,simobs = 100, what = "acc"){

  # Simulation the robot simobs times
  sim_rob <- simHMM(hmm,length = simobs)
```

```
  #extract the observations (x_t)
  observations <- sim_rob$observation

  #extract the hidden states z_t
  state_place <- sim_rob$states

### Filter
    # Forward function computes the filtering with log
    log_filter <- forward(hmm = hmm, observation = observations)

    # Remove the log-transformation
    filter <- exp(log_filter)

    # Normalizing
    norm_filter <-prop.table(filter, margin = 2)

    # Checking which probability in each column is the highest
    most_prob_filter <- apply(norm_filter, MARGIN = 2, FUN = which.max)

    # Accuracy for the filter
    accuracy_filter <-  sum(paste("z",most_prob_filter, sep = "")
                         == state_place) / length(state_place)



### Smoothed (in this package called the posterior)
    smoothed <- posterior(hmm=hmm,observations)

    # Normalizing
    norm_smoothed <-prop.table(smoothed, margin = 2)

    most_prob_smoothed <- apply(norm_smoothed, MARGIN = 2, FUN = which.max)

    # Accuracy for the smoothed
    accuracy_smoothed <-  sum(paste("z",most_prob_smoothed, sep = "")
                              ==  state_place) / length(state_place)

  #cat("The accuracy of the smoothed is",accuracy_smoothed)


### Most probable path (Viterbi)
    mpp <- viterbi(hmm,observations)

    accuracy_mpp <-  sum(mpp == state_place) / length(state_place)
    #cat("The accuracy of the Viterbi is",accuracy_mpp)




  #Just logical statements on what to return.
  if(what == "acc"){
  return( c(accuracy_filter = accuracy_filter,
            accuracy_smoothed = accuracy_smoothed,
```

```
            accuracy_mpp = accuracy_mpp))
  }

  if(what == "filter"){
    return(norm_filter)
  }

  if(what == "smooth"){
    return(norm_smoothed)
  }

  if(what == "mpp"){
    return(mpp)
  }

}
```

### Filter

```
mrRobot(hmm = robot,simobs = 100, what = "filter")
```

To much to print out, so run the code if you want to see the distribution

### Smoothed

```
mrRobot(hmm = robot,simobs = 100, what = "smooth")
```

To much to print out, so run the code if you want to see the distribution

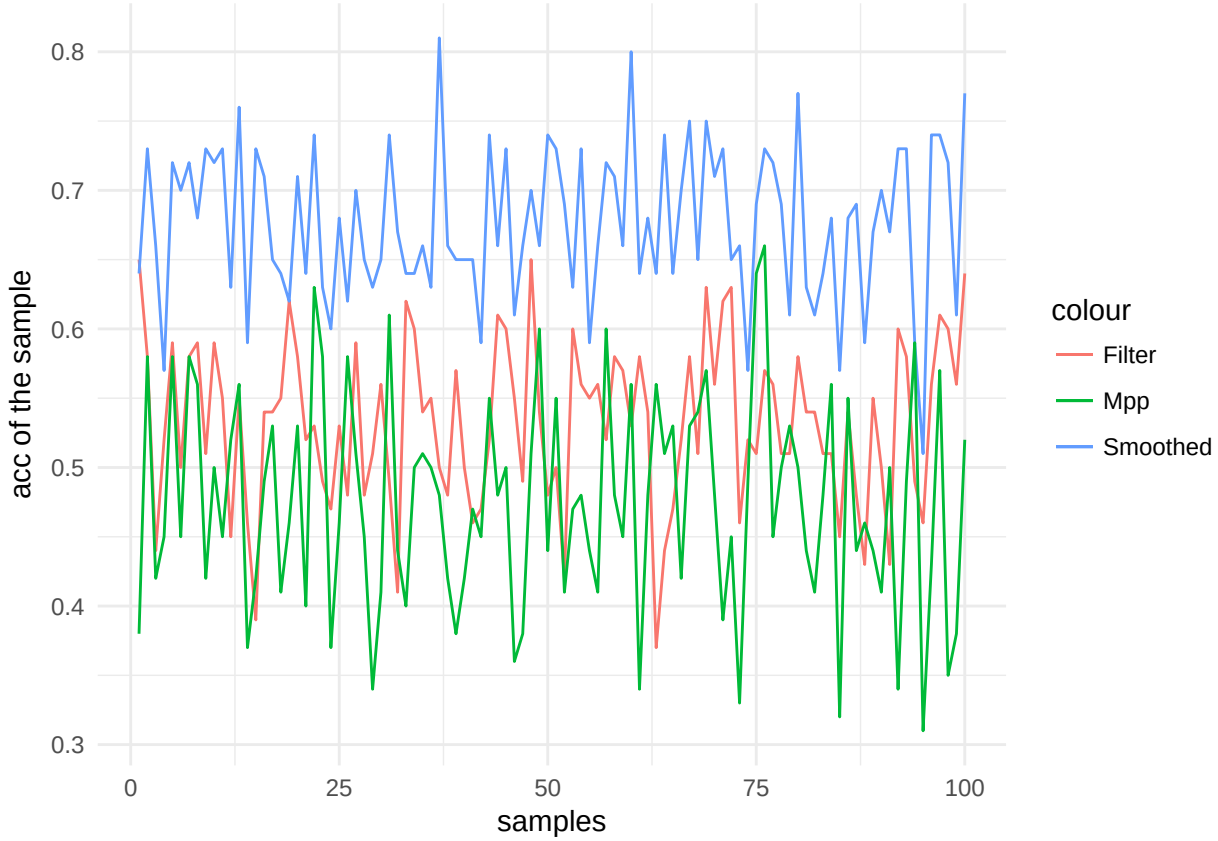
### Most probable path

```
mrRobot(hmm = robot,simobs = 100, what = "viterbi")
```

```
total_acc <- sapply(1:100,FUN = function(x){mrRobot(robot,100, what = "acc")} )

total_acc <- as.data.frame(t(total_acc))
total_acc$index <- 1:100


ggplot(data = total_acc) + geom_line(aes(x=index,y=accuracy_filter , col = "Filter")) +  geom_line(aes(
```




```
colMeans(total_acc[,-4])
```

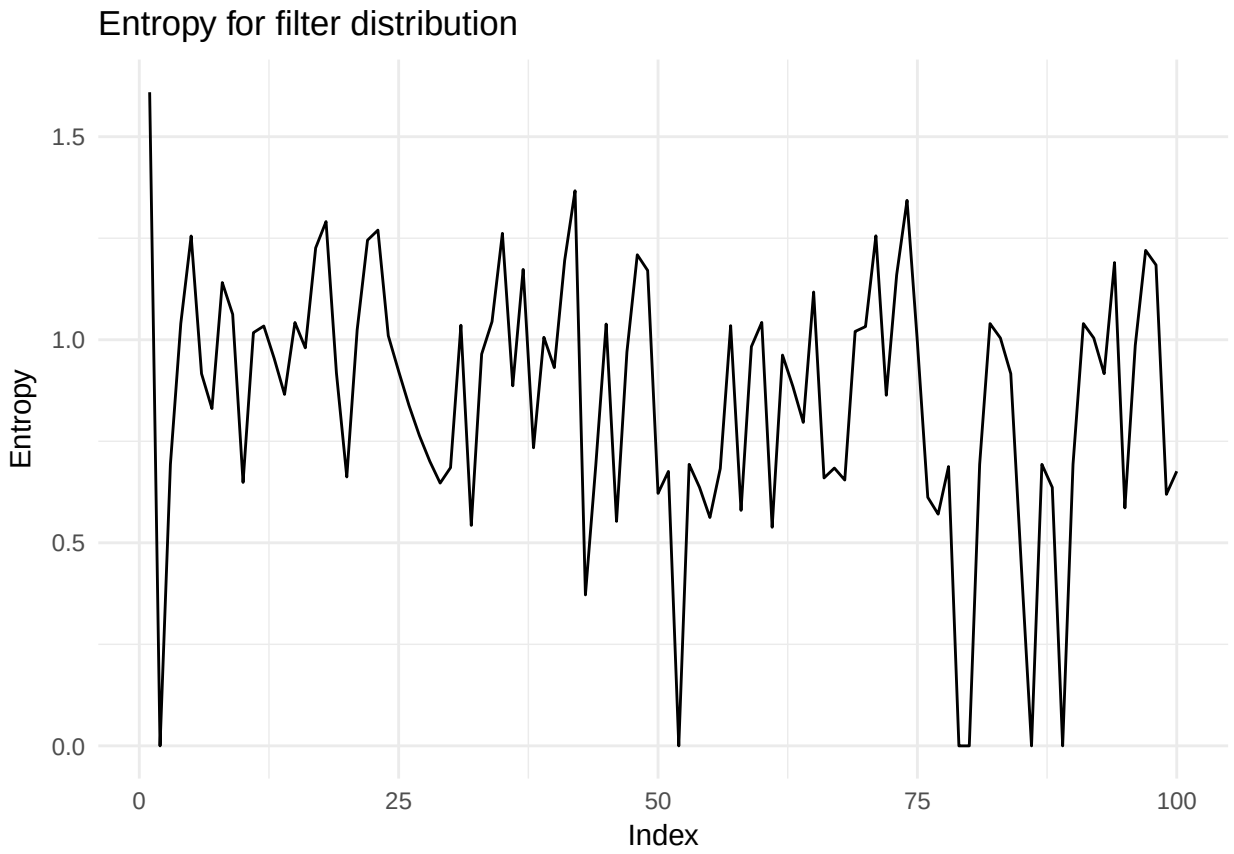
```
##   accuracy_filter accuracy_smoothed      accuracy_mpp
##            0.5321            0.6758            0.4753
```

## Question 6

```
# filter_data <- sapply(1,FUN = function(x){
#   mrRobot(robot,100, what = "filter")
#   })

filter_data <- mrRobot(robot,100, what = "filter")
filter_entropy <- data.frame(Index = 1:100 ,
                  Entropy =
                  apply(filter_data, MARGIN = 2, FUN =entropy.empirical))

ggplot(data = filter_entropy, aes(x = Index, y = Entropy)) + geom_line() + ggtitle("Entropy for filter
```




No, the entropy remains random even while increasing the number of observations added to the hmm. This is because it is markovian and only depends on the previous observation.

## Question 7

```
posterior <- filter_data[,100] # The last information of the robot aka the prior
transition <- robot$transProbs
transition %*% posterior
```

```
##
## from       [,1]
##   z1  0.0000000
##   z2  0.0000000
##   z3  0.2040816
##   z4  0.5000000
##   z5  0.2959184
##   z6  0.0000000
##   z7  0.0000000
##   z8  0.0000000
##   z9  0.0000000
##   z10 0.0000000
```

Since we have a markovian assumption that the only relevant state is the one we are in now and since all previous states feeds forward through z100 and provides it with information about previout observations and states we can just multiply z100 probabilities for each state with the transition probabilities to ge at prediction of how z101 and s101 will be like. In this case we were asked to get the hidden states probabilites

but if you want the observation you can just do a argmax over these probabilities to get were the next state should be.

## Forward Backward algorithm.

```
trans_probs <- diag(1/2, 10) +
  diag(1/2, 10)[, c(10, 1:9)]
emission_probs <-
  diag(1/5, 10)[, c(3:10, 1:2)] +
  diag(1/5, 10)[, c(2:10, 1)] +
  diag(1/5, 10) +
  diag(1/5, 10)[, c(10, 1:9)] +
  diag(1/5, 10)[, c(9:10, 1:8)]

emission_density <- function(x, z) {
  return(emission_probs[z, x])
}

transition_density <- function(z, previous_z) {
  return(trans_probs[previous_z, z])
}

# transition_density2 <- function(z,previous_z){
#   if( z == zt){
#
#     return(0.5)
#
#   } else if( (z + 1) == zt){
#     return(0.5)
#   } else return(0)
#
# }

get_alpha_scalar <- function(zt, xt, previous_alpha, previous_z) {
  # Args:
  #   zt Scalar, hidden state at which to compute alpha.
  #   xt Scalar, observed state.
  #   previous_alpha Vector, alpha for all z_{t-1}.
  #   previous_z     Vector, all z_{t-1}.

  summation_term <- 0
  for (i in 1:length(previous_z)) {
    summation_term <- summation_term +
      previous_alpha[i] * transition_density(zt, previous_z[i])
  }

  alpha <- emission_density(xt, zt) * sum(summation_term)
  return(alpha)
}

get_alpha <- function(Zt, xt, previous_alpha, previous_z) {
  # Args:
```

```
  #   Zt Vector, hidden states at which to compute alpha.
  #   xt Scalar, observed state.
  #   previous_alpha Vector, alpha for all z_{t-1}.
  #   previous_z     Vector, all z_{t-1}.

  alpha <- sapply(Zt, function(zt) {
    get_alpha_scalar(zt, xt, previous_alpha, previous_z)
  })

  return(alpha)
}

get_beta_scalar <- function(zt, next_x, next_beta, next_z) {
  # Args:
  #   zt        Scalar, hidden state at which to compute alpha.
  #   next_x    Scalar, observed next state.
  #   next_beta Vector, alpha for all z_{t+1}.
  #   next_z    Vector, all z_{t+1}.

  summation_term <- 0
  for (i in 1:length(next_z)) {
    summation_term <- summation_term +
      next_beta[i] * emission_density(next_x, next_z[i]) * transition_density(next_z[i], zt)
  }

  # P(z_(t+1) | z_t) =
  # 0.5 if z_t = z_(t+1)
  # 0.5 if z_t = z_t + 1
  # 0 otherwise


  return(summation_term)
}

get_beta <- function(Zt, next_x, next_beta, next_z) {
  # Args:
  #   Zt        Vector, hidden states at which to compute alpha.
  #   next_x    Scalar, observed next state.
  #   next_beta Vector, alpha for all z_{t+1}.
  #   next_z    Vector, all z_{t+1}.

  beta <- sapply(Zt, function(zt) {
    get_beta_scalar(zt, next_x, next_beta, next_z)
  })

  return(beta)
}

fb_algorithm <- function(
  observations,
  emission_density,
  transition_density,
  possible_states,
```

```
  initial_density) {

  t_total <- length(observations)
  cardinality <- length(possible_states)

  # Alpha
  alpha <- matrix(NA, ncol=cardinality, nrow=t_total)

  for (i in 1:cardinality) {
    alpha[1, i] <-
      emission_density(observations[1], possible_states[i]) * initial_density[i]
  }

  for (t in 2:t_total) {
    alpha[t, ] <- get_alpha(possible_states, observations[t], alpha[t - 1, ], possible_states)
  }


  # Beta
  beta <- matrix(NA, ncol=cardinality, nrow=t_total)

  beta[t_total, ] <- 1

  for (t in (t_total - 1):1) {
    beta[t, ] <- get_beta(possible_states, observations[t + 1], beta[t + 1, ], possible_states)
  }

  return(list(alpha = alpha, beta = beta))
}

filtering <- function(alpha) {
  alpha / rowSums(alpha)
}

smoothing <- function(alpha, beta) {
  alpha * beta / rowSums(alpha * beta)
}







robotHmm <- HMM::initHMM(
  States = 1:10,
  Symbols = 1:10,
  transProbs = trans_probs,
  emissionProbs = emission_probs
)

# Create a wrapper for simHMM to assign class to the output
simHMM <- function(hmm, length) {
```

```
  simulation <- HMM::simHMM(hmm, length)
  return(structure(simulation, class="HmmSimulation"))
}

# Simulate
nSim <- 100
robotSimultation <- simHMM(hmm=robotHmm, length=nSim)

#debugonce(fb_algorithm)

alphabeta <- fb_algorithm(observations = robotSimultation$observation,
                          emission_density = emission_density,
                          transition_density = transition_density,
                          possible_states = 1:10,
                          initial_density = rep(0.1, 10))

#filtering(alphabeta$alpha)
#smoothing(alphabeta$alpha, alphabeta$beta)


# plot(apply(filtering(alphabeta$alpha), 1, which.max), type = "l")
# plot(apply(smoothing(alphabeta$alpha, alphabeta$beta), 1, which.max), type = "l")
# lines(x = 1:100,robotSimultation$states, type = "l", col ="green")
#



# # # Test
# zt <- 5
# xt <- 6
# previous_alpha <- rep(0.1, 10)
# previous_z <- 1:10
# transition_density(zt, previous_z[5])
# get_alpha_scalar(zt, xt, previous_alpha, previous_z)


# # Test
# zt <- 1:10
# xt <- 6
# previous_alpha <- rep(0.1, 10)
# previous_z <- 1:10
# transition_density(zt, previous_z[5])
# get_alpha(zt, xt, previous_alpha, previous_z)
#


# # Test
# Zt <- 1:10
# next_x <- 6
# next_beta <- rep(0.1, 10)
# next_z <- 1:10
# transition_density(zt, previous_z[5])
# get_beta_scalar(5, next_x, next_beta, next_z)
```

```
# get_beta(zt, next_x, next_beta, next_z)
```

## Viterbi

```
# Define the transition, emission and initialization probabilities --------

emission_probs <- matrix(c(.2, .2, .2, 0, 0, 0, 0, 0, .2, .2,
                           .2, .2, .2, .2, 0, 0, 0, 0, 0, .2,
                           .2, .2, .2, .2, .2, 0, 0, 0, 0, 0,
                           0, .2, .2, .2, .2, .2, 0, 0, 0, 0,
                           0, 0, .2, .2, .2, .2, .2, 0, 0, 0,
                           0, 0, 0, .2, .2, .2, .2, .2, 0, 0,
                           0, 0, 0, 0, .2, .2, .2, .2, .2, 0,
                           0, 0, 0, 0, 0, .2, .2, .2, .2, .2,
                           .2, 0, 0, 0, 0, 0, .2, .2, .2, .2,
                           .2, .2, 0, 0, 0, 0, 0, .2, .2, .2), byrow=TRUE, nrow=10)

transition_probs <- matrix(c(.5, .5, 0, 0, 0, 0, 0, 0, 0, 0,
                             0, .5, .5, 0, 0, 0, 0, 0, 0, 0,
                             0, 0, .5, .5, 0, 0, 0, 0, 0, 0,
                             0, 0, 0, .5, .5, 0, 0, 0, 0, 0,
                             0, 0, 0, 0, .5, .5, 0, 0, 0, 0,
                             0, 0, 0, 0, 0, .5, .5, 0, 0, 0,
                             0, 0, 0, 0, 0, 0, .5, .5, 0, 0,
                             0, 0, 0, 0, 0, 0, 0, .5, .5, 0,
                             0, 0, 0, 0, 0, 0, 0, 0, .5, .5,
                             .5, 0, 0, 0, 0, 0, 0, 0, 0, .5), byrow=TRUE, nrow=10)

tProbDensity <- function(zt, zt_1) {
  return(transition_probs[zt_1, zt])
}

eProbDensity <- function(xt, zt) {
  return(emission_probs[zt, xt])
}

initProbDensity <- function(z0) {
  return(dunif(z0, min=1, max=10))
}


# Simulate data -----------------------------------------------------------

library(HMM)

robotHmm <- HMM::initHMM(
  States = 1:10,
  Symbols = 1:10,
  transProbs = transition_probs,
  emissionProbs = emission_probs
)
```

```r
simHMM <- function(hmm, length) {
  simulation <- HMM::simHMM(hmm, length)
  return(structure(simulation, class="HmmSimulation"))
}

nSim <- 100
robotSimultation <- simHMM(hmm=robotHmm, length=nSim)

X <- robotSimultation$observation
Z <- robotSimultation$states

# Implement Viterbi ------------------------------------------------------

possibleStates <- 1:10
get_omega <- function(Z, Omega, Z_next, x_next) {
  sapply(Z_next, function(z_next) {
    term1 <- log(eProbDensity(x_next, z_next))

    term2 <- sapply(Z, function(z) {
      log(tProbDensity(z_next, z))
    }) + Omega

    return(term1+ max(term2))
  })
}

get_phi <- function(Z, Z_next, Omega) {
  sapply(Z_next, function(z_next) {
    term <- sapply(Z, function(z) {
      log(tProbDensity(z_next, z))
    }) + Omega
    return(Z[which.max(term)])
  })
}

viterbi <- function(observations, possibleStates) {
  cardinality <- length(possibleStates)
  t_total <- length(observations)

  omega_0 <- vector("numeric", length = cardinality)
  for (i in 1:cardinality) {
    omega_0[i] <- log(initProbDensity(possibleStates[i])) +
      log(eProbDensity(observations[1], possibleStates[i]))
  }


  omega <- matrix(NA, nrow=t_total, ncol=cardinality)
  phi <- matrix(NA, nrow=t_total, ncol=cardinality)
  omega[1, ] <- omega_0

  for (i in 1:(t_total-1)) {
    omega[i+1, ] <- get_omega(possibleStates, omega[i, ], possibleStates, observations[i+1])
    phi[i+1, ] <- get_phi(possibleStates, possibleStates, omega[i, ])
```

```
  }

  mpp <- rep(NA, t_total)
  mpp[t_total] <- possibleStates[which.max(omega[t_total, ])]
  for (t in (t_total - 1):1) {
    mpp[t] <- phi[t + 1, possibleStates[mpp[t + 1]] == possibleStates]
  }

  return(list(path = mpp, omega = omega, phi = phi))

}

results <- viterbi(X, possibleStates)
results$path
```

```
##   [1]  8  9 10  1  1  1  1  1  2  2  2  2  2  3  4  5  6  6  6  6  7  8  9
##  [24] 10  1  1  1  1  1  2  3  4  5  6  7  8  9 10  1  1  1  1  1  2  3  4
##  [47]  5  5  5  5  5  5  5  5  5  6  7  7  8  8  8  8  9 10  1  1  1  2  2
##  [70]  3  3  3  3  4  5  5  5  5  6  6  6  6  6  6  7  7  7  7  7  8  9 10
##  [93]  1  1  1  1  2  3  3  3
```

```
results_HMM <- HMM::viterbi(robotHmm, X)

#cbind(results$path, results_HMM)
```