# An Improved Hidden Markov Model for Indoor Positioning

Xingyu Ren[1], Di He[1(✉)], Xuyu Gao[1], Zhicheng Zhou[2], and Chih-Chun Ho[3]

[1] Shanghai Key Laboratory of Navigation and Location-Based Services, Shanghai Jiao Tong University, Shanghai, China
{dihe,gao_xuyu}@sjtu.edu.cn
[2] Shenzhen Dashi Intelligent Co., Ltd., Shenzhen, China
zhouzhicheng@chn-das.com
[3] Beijing Jizhi Digital Techonology Co., Ltd., Beijing Longfor Blue Engine Industrial Park, Building 6, No.8 Beiyuan Street, Chaoyang District, Beijing, China

**Abstract.** This paper proposes an indoor positioning method combines machine learning, IMU (Inertial Measurement Unit) and an improved HMM (Hidden Markov Model). HMM is the base framework, the latent states correspond to the location grid, which uses two methods to divide the area into grids with different granularity. The fine-grained grids are used to compute transition probability, and the coarse-grained ones for emission probability. IMU data can help to adjust transition probability between fine-grained grids, and some machine learning methods to estimate the emission probability in each coarse-grained grid. And for the particularity of the model, there's an improved Viterbi algorithm proposed to calculate the most likely path, which is a more robust version compared with the original one.

**Keywords:** Indoor Positioning · Hidden Markov Model · IMU · Machine Learning

## 1 Introduction

The main contribution of this paper is the proposed improved HMM and the modeling method for indoor positioning that combines classical fingerprinting method and inertial navigation method.

### 1.1 Indoor Positioning

Indoor positioning is a hot topic recently, and there are many methods to achieve it. Not like the outdoor positioning, the precise indoor positioning is more difficult to be realized due to the complex environmental problems.

In outdoor environment, one can use GPS and the geometric relationship to locate the position. But this method is not suitable for indoor environment.


F. Gao et al. (Eds.): ChinaCom 2022, LNICST 500, pp. 403–420, 2023.
https://doi.org/10.1007/978-3-031-34790-0_31

Usually, GPS signal is not available in indoor environment, or the signal is weak and not accurate because of fading and multipath. So, the indoor positioning methods are mainly based on the signal features (like strength) of the wireless signals, such as WiFi, Bluetooth, Zigbee, etc. Nowadays, 5G technology is applying and deploying steps by steps. Because of the properties of 5G, there're many indoor stations, which can also provide many signal features.

The most popular method is fingerprinting. It's like machine learning method, requires a pre-collected database of signal features on each pre-demarcated location. And use some model to train the data, this is called offline stage. When the user is in the indoor environment, the signal features can be collected and input to the trained model to get the prediction. This is called online stage. This method bypass the geometric method.

Another main method is use Inertial Navigation System (INS) to locate a path of positions. This method use the IMU to measure the acceleration and angular velocity of the user. And then use calculus to calculate the speed and position [2]. The shortage of INS is drift error will accumulate over time span, so usually people only use INS as assistance for other system. But in a short period of time it is reliable.

In this paper, those methods are combined with Hidden Markov Model (HMM) to achieve more accurate positioning. This method can utilize the history of user's movement to predict the current position. The IMU data can help to adjust the transition probability between grids, and original fingerprinting method or machine learning method can provide emission probability.

### 1.2 Hidden Markov Model

HMM is a statistical Markov model in which the problem being modeled is assumed to be a Markov process with unobservable ("hidden") states. As part of the definition, HMM requires that there is observable process whose outcomes are "influenced" by the outcomes of hidden states in a known way. Since hidden states cannot be observed directly, the goal is to learn about hidden state sequence by observing the observable sequence. HMM has an additional requirement that the outcome of observable process at time $t = t_0$ must be "influenced" exclusively by the outcome of the corresponding hidden state, and that the outcomes of previous hidden states (and the observations) at $t < t_0$ must not affect the outcome of $t_0$'s state and observation [12,18].

HMM are known for their applications to statistical mechanics, physics, chemistry, economics, finance, signal processing, information theory, pattern recognition – such as speech, handwriting, gesture recognition, part-of-speech tagging, musical score following, partial discharges and bio-informatics [5,10,11,13,15].

HMM have five main factors:

- State space: The latent states
- Observation space: The observable outcomes

- Transition probability between states: 2-D matrix-like table $A[i][j]$ is the probability of transitioning from state $i$ to state $j$
- Emission probability of each state: 2-D matrix-like table $B[o][j]$ is the probability of observing $o$ in state $i$
- Initial probability of each state

Viterbi algorithm [16] is used to solve the HMM's latent state sequence from observation sequence. This article will propose a new HMM that can utilize the IMU data to dynamically update the transition probability, in order to do that, the gridding method should be refined, whose "side effect" is making the position estimation more accurate in some way. The corresponding Viterbi algorithm is also proposed to improve the robustness of the algorithm.

## 2 Mathematical Modeling

As the description in Sect. 1.2, the property of HMM is perfectly suited for indoor positioning. However, it's necessary to model the indoor environment to fit those five properties in the HMM.

Let's make the HMM-based indoor positioning idea clear first. In the indoor positioning problem, the state of HMM is the location, the observation is the signal features, the transition probability is the probability of moving from one location to another, the emission probability is the probability of observing the signal features in each location.

Some symbols are defined as follows:

- State space which is location (grid) set: $\mathcal{L}$
- Observation space which is signal features set: $\mathcal{O}$
- Transition Probability is still using the original notation: $A[i][j]$, where $i$ and $j$ are the locations
- Emission probability is still using the original notation: $B[i][o]$, where $i$ is the location, $o$ is the signal features
- Initial probability is noted as $\Pi$, where $\Pi[i]$ is the initial probability of in location $i$

And the input observation sequence to compute the best latent state sequence is noted as $Y$, where $Y_i$ is the observation of the time $i$.

A location here is an abstract concept. In general, a location is a grid. So it relates to a coordinate in the real world.

Five main factors in HMM will be detailed in the following sections respectively.

### 2.1 Observation and Signal Features

The observation is one of the main concepts of HMM. As the name suggests, the observation is some observable that can be influenced by the hidden states. Here just use the signal features to describe the observation. In fingerprinting method,

the popular signal feature is the strength of the signal, or RSS (Received Signal Strength). RSS data collection is carried out at each grid in pre-demarcated areas.

There should be several base stations or APs in the indoor environment, so that it's possible to synthesize RSS information from multiple APs. If there's only one or two APs, the same RSS values may appear in many different places, resulting in ambiguity for computation of emission probability.

The RSS data of those concerned APs should be arranged carefully. There's a simple method to organize the RSS information. Suppose there're a group of APs and each of them has an unique ID. At first, determine a specific order of those APs (using the ID), and then when collecting the RSS data in each grid, always use the same order to organize the RSS data. For example, suppose there're 4 APs, whose ID are listed as $122, 124, 133, 135$. For one collection of a single acquisition, there are usually 4 RSS values, like $-101$ for AP 122, $-112$ for AP 124, $-88$ for AP 133, and $-94$ for AP 135. Then the representation of the RSS data is $[-101, -112, -88, -94]$. It's important to sort the RSS data of the corresponding AP in the same and determined order as the APs.

But there're some corner cases. First is that the RSS data of some APs may be missing. In the example above, if the No.124 AP is missing, fill it with a reasonable default value. In general it should use the minimum RSS value. For 5G signal collected by Android phone, the minimum RSS value that phone can collect is $-140$ [8], so the missing AP's data should use $-140$ or value that is less than $-140$. Second situation is there're more APs than needed. In this case, just simply ignore the data from extra APs.

In conclusion, the observation is a vector of RSS values that sorted in a pre-defined AP's order.

### 2.2 Emission Probability

The emission probability is the probability of observing the signal features in each grid. Formally, use $B$ to represent it, which is a function that takes the location and signal features as input and returns the probability of observing the signal features in that location.

In the original HMM, the signal features (Observation space) is generally a finite discrete set. But as it shows in Sect. 2.1, the observation is a vector of RSS values that, in most cases, belongs to an infinite set. So the emission probability needs to be calculated in other way.

Like traditional fingerprinting method, classification methods like KNN (K-Nearest Neighbors) [1] or DNN (Deep Neural Networks) [14] are usually used to classify the RSS vector into different classes, which here is the location grids. In the final step of classification, most of machine learning methods can produce a probability vector that represents the probability of each class/location. Declare the location (noted as $l$)'s probability given the signal features (noted as $o$) as $\mathrm{P}(L = l|O = o)$, where $L$ is a random variable of the grid and $O$ is the observation.

So $\mathrm{P}(O = o|L = l)$ can be deduced by $\mathrm{P}(L = l|O = o)$:

$$\begin{aligned} \mathrm{P}(O = o|L = l) &= \frac{\mathrm{P}(L = l|O = o)\,\mathrm{P}(O = o)}{\mathrm{P}(L = l)} \\ &\sim \mathrm{P}(L = l|O = o) \end{aligned} \tag{1}$$

Considering required value for Viterbi algorithm is the maximum/minimum, not the accurate value, so as long as all the emission probability are proportional to the true value at the same scale, $\mathrm{P}(O = o|L = l)$ can be replaced by $\mathrm{P}(L = l|O = o)$ directly.

More formally,

$$B[l][o] = \mathrm{P}(L = l|O = o) \tag{2}$$

In practice, to get the emission probability, use the similar approach as fingerprinting method. First collect the RSS data. For each grid, measure RSS many times to build the fingerprinting database. For each measurement, it is an observation as described in Sect. 2.1. This means for each grid, for each measurement, there's an RSS vector as an observation. And then those data will be used to train the KNN or some other machine learning models. Finally use the model to provide the emission probability.

In HMM, the input is an observation sequence, so each observation in that sequence will be passed to a machine learning method (like KNN) to compute $\mathrm{P}(L = l|O = o)$, then there is a $B$ for each $Y_i$. $Y_i$ is input to a machine learning model and get a probability distribution of each grid $l$. That's the emission probability at time $i$. So there will be a sequence of emission probability, which has the same length as $Y$. Formally, the notation (and the definition) of emission probability becomes: $B$ from now on is a sequence of emission probability, where $B_i$ is the emission probability related to $i$-th observation $Y_i$.

### 2.3 State of HMM

To determine the state space, the model uses grid map to partition the indoor environment. In Fig. 1, the indoor area is divided into the meshed grids. In original HMM, a grid is associated with a state, and the transition probability between two grids is related to the distance between those two grids. The observation of each grid is some signal feature (e.g. the strength of the signal or some aggregated information, will discuss later) there. The emission probability can be estimated by some machine learning method, which basically is a function that maps the signal feature to the probability that the signal feature occurs at each location, e.g. SVM, KNN, DNN, etc. These methods are called "fingerprinting" methods, the fingerprinting means the signal features. As we know, those machine learning methods require pre-collected data to train the model. Basically, the state space of HMM is the grid of the indoor positioning area. But the original grids with coarse granularity is not usable for HMM, especially for the transition probability part.

But this leads to a problem: the grid can't be too small, otherwise there's too much work to collect each grid's signal feature. Meanwhile the grid can't be

too large, otherwise the transition from one grid to its neighbor is kind of impossible, let alone the transition from one grid to further grids. So the transition probability doesn't make sense any more.

For example, consider the grid size (the length of side of each grid) $d$ is as large as about 5 meters, while a man's walking speed is about 1 m/s. And the measurement is at a frequency of about once per second. In this scenario, it's kind of impossible to transition from one grid to it's adjacent grid. So the transition probability between two grids is zero. Even we can set the transition probability from one grid to its neighbor grid to be some appropriate value, like 0.15 for four adjacent grid and 0.4 for current grid. But this would lead to another problem: when using Viterbi algorithm to solve HMM, every step will position at a grid's center point, and then the next step is still current grid if there's no large change of observation, or the next step will suddenly "jump" to the neighbor girds. And this is based on the assumption that the emission probability is accurate. This will reduce the HMM and Viterbi algorithm to original emission probability calculation, which is same as using the machine learning method directly: It can not utilize the historical information to predict the current location any more.

But the grid size can not be too small to just fit the measuring frequency. It is also difficult to measure and collect the RSS data and create fingerprints for each small grid. And further more, as described in the Sect. 2.1, for normal RSS measurement device, the resolution is not very high, so if the grid size is too small, there will be no difference of RSS between two adjacent grids. In the test environment later, we use an Android phone to measure the RSS. Because the accuracy of the phone is not very high, the size of the grid cannot be too small. Generally in the real indoor environment, when the size of the grid is less than 3m, the RSS of the adjacent grid is difficult to distinguish.

In order to utilize HMM, it's necessary to find a way to balance the two problems described above, or even solve both. That is to collect useful and distinguishable data easily and model the transition probability accurately.

In this study, we propose a solution to model the locations. Use two grid map with different granularity: a coarse grid map (which the grid size is large) for measuring RSS (observation) and computing emission probability, and a dense grid map (which the grid size is small) for transition probability computing. And then the states in HMM is now the small grids.

So now the location/state set $\mathcal{L}$ contains the fine-grained small grids. And $l$ will represent the small grid in general context.

But there's another problem: how to determine the small grid's emission probability? To combine the information of both maps together, we set the emission probability of each small grid is equal to the big one which contains the small one. More formally, assume the big grid is $l'$ and the small grid it contains is $l$. According to Eq. (2), the emission probability of the small grid is updated as follows:

$$B_i[l] = B[l][Y_i] = \mathrm{P}(L = l' | O = Y_i), \forall l \text{ belongs to } l' \tag{3}$$

Further more, this model will benefit the transition probability computation considering IMU data. This will be discussed in the Sect. 2.4.

This model has many advantages. It not only fixes the two problems about transition and observation, but also provides a simple way to describe the obstacles in the indoor environment. If the resolution of the small grid map is high enough, an obstacle, like a wall or a desk, can be modeled by simply removing a clique of some small grids.

### 2.4 Transition Probability and IMU

Another important part of HMM is the transition probability $A$, where $A[p][l]$ is the probability of transition from previous state (location/small grid) $p$ to current state $l$.

The plain idea to describe transition probability is that the closer the distance between two location, the greater the probability of moving to each other. So firstly it is suggested to compute the distance between two small grids.

Consider the grid map as an undirected (a two-way directed) graph. Each small grid is a vertex in the graph. Each small grid is only connected to its four directly adjacent grid (north, south, east, west), or maybe counts the four diagonal grid in and there will be 8 adjacent grid. The edge weight is simply the distance between those directly or diagonally adjacent grids. For example, in the squared grid map, the distance between two directly adjacent grids is the grid length, and between two diagonally adjacent grids is $\sqrt{2}$ times the grid length.

In the undirected graph, there're several all-pair shortest path algorithms. And there exists two famous ones: Floyd-Warshall algorithm [6] and Johnson's algorithm [9]. Suppose there are $V$ grids (vertices). Floyd's approach has the complexity of $O(V^3)$, and Johnson's approach usually has the complexity of $O(V^2 \log V)$.

But for simplicity, assume every grid is only connected to its four/eight neighbor grids. It's accurate enough for basic scenarios. Given this assumption, it's better to use BFS (Breadth-First Search) to compute the shortest path for each vertex. And the complexity is $O(V^2)$ [4].

It is worth noting that it's unnecessary to compute all vertices for each vertex, since normally when computing the transition probability, what matters is the grids around current one. But for simplicity and clear statements, we compute all vertices here, and will discuss this in Sect. 4.1 later.

Another thing worth mentioning is the obstacles in the environment will be modeled conveniently. As described in last section, the obstacles in the fine-grained grid map will simply be removed. So the distance between two grids that are across the obstacles will be large, resulting the transition probability is small, which makes sense.

For convenience, the distance map is noted as $D$, where $D[p][l]$ is the shortest path's length from $p$ to $l$.

Another problem for transition probability is how to utilize the IMU data, because IMU is generally available in most of mobile devices. It's proper to use a vector $\mathbf{v}$ to model the IMU data. Although the IMU's output is acceleration, but one can use integral to get the speed, and this is reliable in a short period of time. The $|\mathbf{v}|$ is speed, the $\angle\mathbf{v}$ is the direction. Assuming IMU can provide data

at given sampling rate $r$, the speed and direction, formally represented by vector $\mathbf{v}$, can be calculated. Base on the speed vector and the sample rate, HMM can predicate the possible movement between two sampling time. Assume current location of current grid $p$ is $\mathbf{x}$, the speed is $\mathbf{v}$ and sampling period is $\Delta t = 1/r$. For convenience, let $\boldsymbol{\delta} = \mathbf{v}\Delta t$. To compute the next position $\mathbf{y}$:

$$\mathbf{y} = \mathbf{x} + \boldsymbol{\delta} \tag{4}$$

Then we can determine which grid position $\mathbf{y}$ belongs to, and suppose that grid is $p'$.

One more thing worth consideration is the connectivity of those two grids. Since if there're some obstacles between $p$ and $p'$, the transition is impossible. There's a simple way to determine the connectivity of two grids, as the shortest distance between two grids is already computed and the obstacle information is stored in the distance map. If the distance $D[p][p']$ is much larger than the supposed distance, then there must be some obstacles.

Consider a probability distribution with the density function $\Phi$, this function should satisfy some conditions. The longer the distance between two grids, the smaller the probability will be. In general, normal distribution will satisfy. And then the transition probability between previous grid $p$ and current grid $l$, considering the speed $\mathbf{v}$, can be computed as follows:

1. Use $\mathbf{v}$ and Eq. (4) to get the real grid for previous grid, and denote it as $p'$.
2. If $p$ and $p'$ is not connected, the transition probability from $p$ to each other grid is 0. Because it is impossible to move from $p$ to $p'$, leaving alone the probability moving from $p'$ to $p$.
3. If not, the current transition probability $A'[p][l]$ is $\Phi(D[p'][l])$. (Attention: here it is $D[p'][l]$ not $D[p][l]$, cause the IMU data is taken into account).

The pseudo-code is shown in Algorithm 1: $A'$ is the transition function (which is a 2-D matrix) for a certain time.

For each observation in $Y$, the IMU data is usually different, so the transition probability for each observation is also different. So we use the original symbol $A$ as a sequence of matrices, where $A_i[p][l]$ is the transition probability from previous state $p$ to current state $l$, corresponding to the observation $Y_i$. And $A_i$ is $A'$ in Algorithm 1.

It is worth noting that the length of sequence $A$ is equal to the length of $Y$ minus 1. Because the IMU data and transition probability describes the movement from $Y_{i-1}$ to $Y_i$. For example, if there're 3 observations in $Y$, then the length of sequence $A$ is 2, where $A_2[i][j]$ uses the speed $\mathbf{v}$ at moment 1 to predict the movement from time 1 to 2, and $A_3$ use the speed at moment 2 to predict the movement from time 2 to 3 (note that there is no $A_1$).

As the algorithm delivered above, for some $p$ and $l$, the value $A_i[p][l]$ is missing, resulting the sum of those probability is less than 1. But same as in Sect. 2.2, there's no need to compute the exact value for Viterbi algorithm, reasonable values are enough.

**Algorithm 1:** Build Transition Probability at time $t$

**Data:** $\mathcal{L}, \boldsymbol{\delta}, D, \Phi$
**Result:** $A'$ as the transition probability function
**function** $(\mathcal{L}, \boldsymbol{\delta}, D, \Phi)$
**begin**
  **for** $p \in \mathcal{L}$ **do**
    let $\mathbf{x}$ be the coordination of $p$
    let $p'$ be the new location that point $\mathbf{y}$ belongs to, such that $\mathbf{y} \leftarrow \mathbf{x} + \boldsymbol{\delta}$
    **if** $p' \in \mathcal{L}$ and **not** $(D[p][p'] \gg |\boldsymbol{\delta}|)$ **then**
      **for** $l \in \mathcal{L}$ **do**
        $A'[p][l] \leftarrow \Phi(D[p'][l])$

### 2.5 Initial Probability

Initial probability is the probability of user at grid $p$ at time 1. Empirically the initial probability is set to be $1/|\mathcal{L}|$. Another option is to set the probability in proportion to the amount of data in each grid. Formally, the initial probability $\Pi$ is defined as: $\Pi[l] = 1$ or $\Pi[l] =$ the number of the fingerprints in $l$ ($l$ is the small fine-grained grid here, but as revealed in Eq. (3) in Sect. 2.3, here it's fine using the information of the coarse grid that contains current small grid) in the database. And here it's unnecessary to use exact value (which is the proportion of the number in each coarse grids), the reason is same as in Sect. 2.2.

## 3 Improved Viterbi Algorithm

Viterbi algorithm is a dynamic programming algorithm that compute the latent state sequence that is most likely to produce the observation sequence. In order to introduce the improved version of this algorithm, we first investigate the basic idea of Viterbi algorithm [16].

Viterbi Algorithm is a dynamic programming algorithm that an compute the latent state sequence from the observation sequence. Dynamic programming is a method to solve optimization problems, such as finding the minimum/maximum value. Those problems can be divided into sub-problems, and solutions of sub-problems can be reused to solve the original problem. It's a kind of induction in mathematics. For HMM, the problem is to find the most likely (maximum probability) latent sequence given the observation sequence. The sub-problem can be defined as: find the probability of most likely latent for the first $i$ moments (each observation is related to a moment), given the first $i$ observations of $Y$, the last latent state ($i$-th state) is $l$. Suppose the solution of this sub-problem is $V[i][l]$, then the relation of bigger sub-problem and smaller sub-problem is:

$$V(i,l) = \max_{p}\{V[i-1][p] \times A_i[p][l] : p \in \mathcal{L}\} \times B_i[l] \tag{5}$$

And the base case is: at moment 1, for each $l$,

$$V[1][l] = \Pi[l] \times B_1[l] \tag{6}$$

While computing the solution of each sub-problem, it's necessary to record the previous state that leads to the maximum probability of current state. Formally, $psi$ is a table, where

$$psi[i][l] = \arg\max_p \{V[i-1][p] \times A_i[p][l] : p \in \mathcal{L}\} \tag{7}$$

For the original problem, the maximum probability of the latent state sequence (the path) is:

$$\max_l V[|Y|][l] \tag{8}$$

To recover the total path, we start from the last state, which is

$$\hat{l}_{|Y|} = \arg\max_l V[|Y|][l] \tag{9}$$

Then the previous state can be calculated by looking up the table $psi$:

$$\hat{l}_{t-1} = psi[t][\hat{l}_t] \ , t = |Y|, |Y|-1, |Y|-2, \cdots, 2 \tag{10}$$

For the convenience, Viterbi Algorithm can be divided into the following 3 parts:

1. Initialization: use the Eq. (6) to initialize $V[1][l]$ for each $l$.
2. Forward Computation: use the Eq. (5) to compute $V[i][l]$ for each $i$ and $l$, from 2 to $|Y|$. At same time record $psi$.
3. Backward Recovering: find the last state using Eq. (9), then use $psi$ to recover the whole path (Eq. (10)).

The original Viterbi Algorithm has a problem. In the original model, both the transition probability $A$ and the emission probability $B$ contain a lot of zeros. In $A$, usually only the nearby grids have positive number. When computing $B$, depending on the machine learning method selected, usually it has $\mathrm{P}(L = l|O = o) = 0$ for many $l$. Sometimes there's only one $l$ that has non-zero probability (which is 1).

Combining the above two facts, and considering the Eq. (5), it's clear that as computing $V[i][*]$ from $t = 1$ to $|Y|$, there're more 0s in $V[i][*]$. Normally it it not a problem, since $V[i][l] = 0$ means that it's impossible to land on grid $l$ at time $i$. And in fact, that's a good thing because it's more accurate. But sometimes in the real application it appears that

$$V[i][l] = 0 \ , \forall l \in \mathcal{L}$$

And then it can't use Eq. (5) to compute the next state. This will lead the algorithm to fail.

Although this situation is rare, it will still reduce the robustness of the proposed algorithm. So we propose an improved version of Viterbi Algorithm. The basic idea is re-initialize the algorithm on failure. There are some improvements in the corresponding 3 parts of the algorithm.

### 3.1 Initialization

For the initialization subroutine, enable it to re-initialize for each moment $t$. Then Eq. (6) becomes:

$$V[t][l] = \Pi[l] \times B_t[l] \tag{11}$$

The pseudo code for this simple subroutine is described in Algorithm 2. The input is the state space $\mathcal{L}$, the initial probability $\Pi$, the emission probability $B$ (a sequence), and time $t$. The table $V$ is going to be initialized, it's a kind of output, but here the reference of $V$ is passed as a parameter so that it can be modified.

**Algorithm 2:** Initialize Subroutine

```
Data: L, V, Π, B, and t
function init(L, V, Π, B, t)
begin
    for l ∈ L do
        V[t][l] = Π[l] × B_t[l]
    if V[t][l] = 0, ∀l ∈ L then
        error
```

### 3.2 Backward Recovery

For the backward recovering subroutine, enable it to recover the path for any range $[s, t]$. Further more, it's possible that at time $t$, there's no valid previous state to reach current state. This will be explained in the next section. So the basic routine is still find the maximum state for $t$, such as Eq. (9), here it becomes:

$$\hat{l}_t = \arg\max_l V[t][l] \tag{12}$$

And then use $psi$ to compute previous state from $t-1$ to $s$, like Eq. (10). But if there's no valid previous state ($psi[i][seq]$ is null), then use (12) to calculate maximum state for $i$. It's worth mentioning that if the previous states are null for all $i$, then this will reduce to single point positioning (like using machine learning method to predict the location directly), which makes sense.

The pseudo code is described in Algorithm 3. The input is the state space $\mathcal{L}$, the subproblem's table $V$, recovering table $psi$, and the range $[s, t]$ for which the recovering subroutine runs. The output is the path $seq$ for that range, but here it's passed as a reference. $seq$ is a sequence of length $|Y|$ but not equal to $t - s + 1$, and the subroutine will fill the recovered path in $seq$ from $s$ to $t$. This will be more clear in the next Sect. 3.3.

**Algorithm 3:** Recover Subroutine

**Data:** $\mathcal{L}, V, psi, seq$, and $[s, t]$
**function** recover $(\mathcal{L}, V, psi, [s, t])$
**begin**
  $seq[t] \leftarrow \arg\max_l \{V[t][l] : l \in \mathcal{L}\}$
  **for** $i \leftarrow t-1$ **downto** $s$ **do**
    $seq[i] \leftarrow psi[i][seq[i+1]]$
    **if** $seq[i]$ is null **then**
      $seq[i] \leftarrow \arg\max_l \{V[i][l] : l \in \mathcal{L}\}$

### 3.3 Forward Computation

The forward computation is the main part of Viterbi Algorithm. Same as the original Viterbi algorithm, first initialize the basic case at time 1, by calling Algorithm 2. And then start the forward computation from $t = 2$ to $|Y|$.

As mentioned before, the original Viterbi Algorithm will fail when $V[i][l] = 0$ for all $l$. So when it happens, just reset the algorithm.

There's a variable $s$ to record the starting time of the current computation procedure. Initially, $s = 1$. While computing from $i = 2$, or if at some time $i$, $V[i][l] = 0$ for all $l$, then start the reset procedure: First, recover the path $seq$ from last start point $s$ to current break point $i - 1$, by calling Algorithm 3. After this, the path in range $[s, i-1]$ is recovered. Then set $s = i$, to use for next recovering subroutine. Then time $i$ becomes the initial time and re-initialize $V[i][*]$ by calling Algorithm 2.

The pseudo code is described in Algorithm 4. The input is the state space $\mathcal{L}$, the initial probability $\Pi$, the transition probability (sequence with length of $|Y|-1$) $A$, the emission probability (sequence with length of $|Y|$) $B$. The output is the best path $seq$ for the observation sequence $Y$.

Note that we use $|B|$ as the length of observation sequence $Y$ because $B$ is the sequence derived from $Y$ (as described in Sect. 2.2), and they have the same length.

At line 11, it's possible that $V[i-1][p] \times A_i[p][l] = 0, \forall p \in \mathcal{L}$, then $psi[t-1][l]$ is null. That's why we need to consider this situation in Algorithm 3.

Actually, some small operations can also be realized through the proposed approach. For example, a threshold can be defined to indicate whether there is a failure in the process: if there are more than 95% of $V[i][*]$ is 0, it fails. Or if $\max_l V[i][l] < 0.001$, it fails. And then call the re-initialization procedure.

There's a degeneration case: if $V[i][*] \equiv 0 \forall i$, the reset procedure will be called each time. This will lead to a situation that the positioning algorithm is just using the initial probability $\Pi$ and the emission probability of time $i$: $B_i$. As mentioned in Sect. 2.5, if $\Pi[l] \equiv 1$, the degeneration situation is just using the machine learning method for single prediction. If $\Pi[l] =$ the number of the

**Algorithm 4:** Improved Viterbi Algorithm

**Data:** $\mathcal{L}, A, B, \Pi$
**Result:** $seq$: best path
**function** viterbi($\mathcal{L}, A, B, \Pi$)
**begin**
  $s \leftarrow 1$
  allocate $V$ as an array of size $|B|$
  allocate $psi$ as an array of size $|B| - 1$
  allocate $seq$ as an array of size $|B|$
  init($\mathcal{L}, V, \Pi, B, s$)
  **for** $i \leftarrow 2$ **to** $|B|$ **do**
    **for** $l \in \mathcal{L}$ **do**
      $V[i][l] \leftarrow \max_p\{V[i-1][p] \times A_i[p][l] : p \in \mathcal{L}\} \times B_i[l]$
      $psi[i-1][l] \leftarrow \arg\max_p\{V[i-1][p] \times A_i[p][l] : p \in \mathcal{L}\}$
    **if** $V[i][l] = 0, \forall l \in \mathcal{L}$ **then**
      recover($\mathcal{L}, V, psi, seq, [s, i-1]$)
      $s \leftarrow i$
      init($\mathcal{L}, V, \Pi, B, s$)
  recover($\mathcal{L}, V, psi, seq, [s, |B|]$)
  **return** $seq$

fingerprints in $l$ in the pre-collected database, this will be a kind of naive Bayes method, or maximum a posteriori method [7].

### 3.4 Re-clarification of the Symbols

Since there are many changes of the symbols in HMM through the narration, those symbols are listed again:

- emission probability sequence $B$. $|B| = |Y|$. $B_t$ is the emission probability at time $t$, which compute from the observation at time $t$.
- markov transition probability sequence $A$. $|A| = |Y| - 1$. $A_t$ is the transition probability from $t-1$ to $t$.
- state space is the small grid set $\mathcal{L}$ and the observation space is basically an infinite set.
- The large (coarse) grid is used to collect the fingerprints and the symbol is NOT $\mathcal{L}$.

## 4 Implementation Notes

### 4.1 Transition Probability

In Sect. 2.4, the transition probability is based on the IMU data (as speed). So for each moment, IMU data should be taken into consideration for the transition probability. This is a time consuming process.

Here we can define a threshold of distance $d_t$. If the distance between previous state (grid) is less than $d_t$, compute the transition probability. Otherwise, the transition is impossible. This will reduce the scale of computation. When compute $V$ in Viterbi Algorithm, it can just ignore the impossible transition.

Another improvement is using cache. First, partition the IMU data (speed) into discrete values. It can partition on both direction and magnitude of speed. Then for each discrete value, compute the transition probability and store them to cache. When the new IMU data is updated, find which discrete value it belongs to, and then get the transition probability from cache.

Another thing is locality [3]. Define the transition probability as $A[l][p]$ where $p$ is still previous state and $l$ is current state. This will accelerate the computation of finding maximum previous state part in Viterbi Algorithm.

### 4.2 Probability Representation

Generally probability is a floating number, if we use the floating number directly, as we computing the chain of probability in Viterbi Algorithm, it will cause numerical error (called "underflow") [17].

It's better to use logarithm probability. And the change the multiplying operation to summarization operation. And this will avoid the numerical error.

## 5 Real Application

The real application ground is shown in Fig. 1. The size of the real application ground area is $20\,\text{m} \times 32\,\text{m}$. The size of coarse grid is $4\,\text{m} \times 4\,\text{m}$. This is used for fingerprints collection. The size of fine grid is $0.5\,\text{m} \times 0.5\,\text{m}$ (not shown in Fig. 1). Because human step size is usually larger than 0.5m, so the transition between nearby grids makes sense. There are some places containing obstacles, which are marked as red X-shape. For those obstacles that are not large enough to fill the whole grid, it is counted as the whole grid.

First, we collect the RSS fingerprints for each coarse grid in the ground and train the KNN model.

Second, which is the real positioning process, we walk along the pre-defined test route (blue arrows in Fig. 1), and collect the RSS data, the direction and speed information. Meanwhile, we also record the real position for each collection as truth value.

Finally, we use the collected RSS data, the direction and speed information to predict the position using KNN model and HMM model. There're some preprocessing steps for the collected data. For the RSS data, we remove the outliers and normalize the data. For the direction and speed information, because the real output fluctuates a lot, we use the moving average method to smooth the data. And we round the direction to four directions: north, south, east and west.

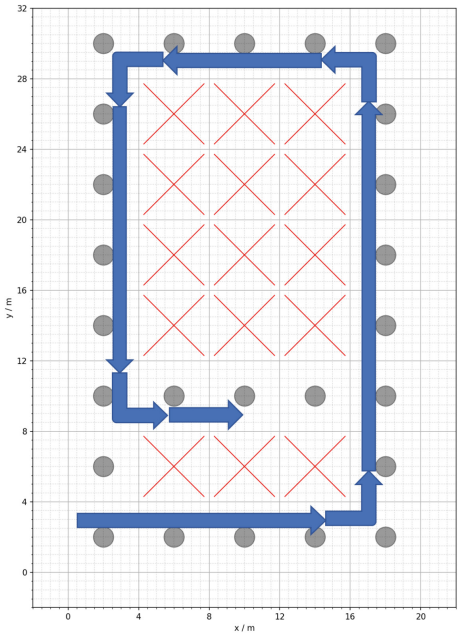


**Fig. 1.** Test Ground

Since we need to calculate the emission probability from each observation using KNN, in the Table 1, the results by using KNN method are also given. The first column is the No. of observation sequence, and there're 93 observations in $Y$. The second and third column is the error distance, which is, for each observation, the distance between truth value and prediction of KNN and HMM. Due to the limitation of page size, the table only shows some representative results and the full picture of 93 observations is shown in Fig. 2.

From Table 1 and Fig. 2, it can be found that the effect of HMM method is better than that of KNN. And HMM is smoother, there're no large jumps. Especially, at time 48, the error of KNN reaches 28 m. It is found in the test log that the real position is (13 m, 29 m), but KNN predicts it's at (11 m, 1 m). The reason is KNN's prediction depends on single observation, while HMM can utilize the observation sequence (and speed information) to predict the position.

The RMSE (Root Mean Square Error) of KNN is 3.83532 m, and the RMSE of HMM is 1.04212 m.

**Table 1.** Test Results

| No. of Obs | Error Distance | |
|---|---|---|
| | *KNN* | *HMM* |
| 1 | 2 | 0 |
| 2 | 0 | 1.58114 |
| 3 | 2 | 0 |
| ⋮ | ⋮ | ⋮ |
| 16 | 2 | 2.54951 |
| 17 | 5.65685 | 2.54951 |
| 18 | 6 | 3.53553 |
| 19 | 4 | 3 |
| ⋮ | ⋮ | ⋮ |
| 46 | 4.4724 | 1.58114 |
| 46 | 4.4724 | 1.58114 |
| 47 | 2 | 0 |
| 48 | 28.0713 | 0 |
| ⋮ | ⋮ | ⋮ |
| RMSE | 3.83532 | 1.04212 |

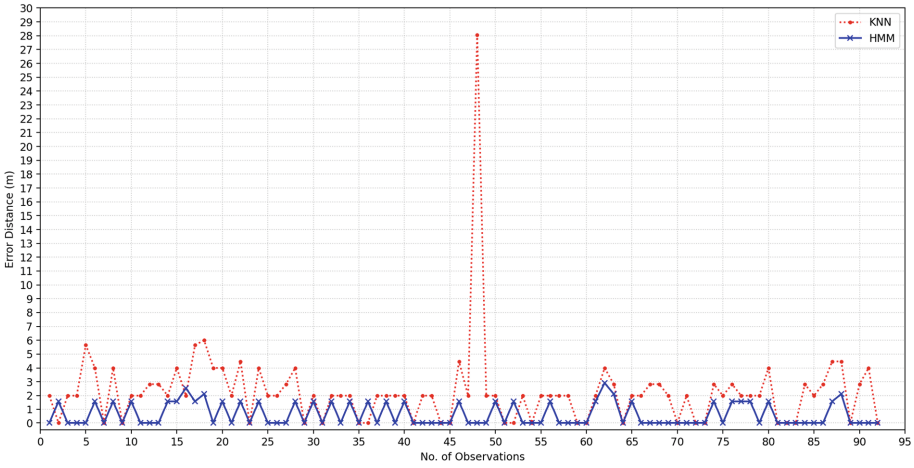


**Fig. 2.** Error Distance of KNN and HMM

## 6 Conclusion

The improved HMM is based on the emission probability sequence $B$ and the markov transition probability sequence $A$. The emission probability is computed from the observation using machine learning method like KNN. This is based on fingerprinting method, requires onerous labor to collect fingerprints for each grid, and then build the database. The markov transition probability is computed from the IMU data (speed) and the distance between each location, so it contains the movement information of the user at that moment. The data collecting part requires the grid not too small, or it's too difficult to collect data and the data between small grids are not distinguishable because of the low resolution of normal sensors. But to compute transition probability, the grids can not be too large, or it will be impossible to move from one grid to another in one sampling period. To solve the contradiction of emission probability modeling and transition probability modeling, two kinds of girdding methods are used. The small one is used to calculate the transition probability and the large one is used to calculate the emission probability. And the small grids can also be used to model the obstacles in the environment.

After modeling, in real practice, sometimes the original Viterbi Algorithm can not work well due to the condition when all $V[i][l]$ become 0. So an improved algorithm which adds reset mechanism is proposed. Each time when the algorithm fails, it will reset the variables and output the best path of the last correct range. It will increase the robustness of the algorithm.

Finally, the real application results show that the proposed method is more accurate, smoother, and there're no large jumps between adjacent predications.

**Acknowledgment.** This research work is supported by the National Natural Science Foundation of China under Grant No. 61971278 and 62231010, the Longfor Group and Shanghai Jiao Tong University Joint Research Project under Grant No. XM22018 "Customer Flow Positioning and Analysis in Commercial Scenarios", and the Shenzhen Science and Technology Innovation Commission Undertaking Major National Science and Technology Project under Grant No. CJGJZD20210408092601004.

## References

1. Altman, N.S.: An introduction to kernel and nearest-neighbor nonparametric regression. Am. Stat. **46**(3), 175–185 (1992)
2. Bejuri, W.M.Y.W., Mohamad, M.M., Omar, H., Omar, F.S., Limin, N.A.: Robust special strategies re sampling for mobile inertial navigation systems (2019)
3. Bryant, R., O'Hallaron, D.R.: Locality, pp. 604–608. Pearson (2016)
4. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. MIT press, Cambridge (2009)
5. Ernst, J., Kellis, M.: ChromHMM: automating chromatin-state discovery and characterization. Nature Methods **9**, 215–6 (2012)
6. Floyd, R.W.: Algorithm 97: shortest path. Commun. ACM **5**(6), 345 (1962)
7. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016). http://www.deeplearningbook.org

8. Google Inc.: Cellsignalstrengthnr.getcsirsrp (2022). https://developer.android.com/reference/kotlin/android/telephony/CellSignalStrengthNr?hl=en#getCsiRsrp(). Online version of the Android API documentation
9. Johnson, D.B.: Efficient algorithms for shortest paths in sparse networks. J. ACM **24**(1), 1–13 (1977)
10. Li, N., Stephens, M.: Modeling linkage disequilibrium and identifying recombination hotspots using single-nucleotide polymorphism data. Genetics **165**, 2213–33 (2004)
11. Pardo, B., Birmingham, W.: Modeling form for on-line following of musical performances, vol. 2, pp. 1018–1023 (2005)
12. Ross, S.M.: 4 - Markov chains. In: Ross, S.M. (ed.) Introduction to Probability Models, 12th edn., pp. 193–291. Academic Press (2019)
13. Satish, L., Gururaj, B.I.: Use of hidden Markov models for partial discharge pattern classification. IEEE Trans. Electr. Insul. **28**(2), 172–182 (1993)
14. Schmidhuber, J.: Deep learning in neural networks: an overview. Neural Netw. **61**, 85–117 (2015)
15. Starner, T., Pentland, A.: Real-time American sign language recognition from video using hidden Markov models. In: Proceedings of International Symposium on Computer Vision - ISCV, pp. 265–270 (1995)
16. Viterbi, A.: Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. IEEE Trans. Inf. Theory **13**(2), 260–269 (1967)
17. Wikipedia contributors. Arithmetic underflow – Wikipedia, the free encyclopedia (2021). https://en.wikipedia.org/w/index.php?title=Arithmetic_underflow&oldid=1043452199. Accessed 7 Aug 2022
18. Wikipedia contributors. Hidden Markov model—Wikipedia, the free encyclopedia (2022). Accessed 28 Sept 2022