Chapter 7
Localization for Mobile Networks

7.1 Overview

In previous chapters, we have introduced many localization algorithms for wireless ad hoc and sensor networks. All those algorithms have a common assumption: nodes reside at fixed locations after being deployed, and localization can be done once for all. Recently, mobile networks are emerging because of the following two major reasons:

– *Passive motion*. In some applications, wireless nodes are attached to animals or people moving throughout an environment, e.g., in the ZebraNet [89], sensor nodes are placed on a sampled set of zebras. In addition, in some applications, e. g., ecosystem monitoring, wireless networks are deployed in dynamic environments, where nodes are in passive motion with the surroundings. For example, sensor nodes deployed on a sea surface are facing the motion of flows and waves [7, 90].
– *Active motion*. With the development of robotic platforms, wireless nodes can control underlying inexpensive robots and move autonomously for wide-area surveillance and reconnaissance. Mobility of nodes can improve network performance in a wide range of aspects, including increasing capacity [91], enhancing security [92], and improving connectivity [93].

Node mobility gives rise to new challenges on localization. The most straightforward and essential one is that localization is no longer a one-time task but a continuous and repeated procedure. In general, three ways exist for mobile network localization:

– Equip mobile nodes with global positioning system (GPS) receivers. However, as suggested when we discuss static networks, this solution faces nonavailable GPS signal and high hardware and energy costs, which make it not suitable for large-scale, low-cost mobile wireless networks. Nevertheless, it is more reasonable when adopted in mobile networks than static networks.
– Re-execute localization algorithms for static networks periodically to compute the real-time locations of mobile nodes. One important drawback of this strategy is that, in a highly dynamic environment where nodes move fast, localization

Y. Liu and Z. Yang, *Location, Localization, and Localizability: Location-awareness Technology for Wireless Networks*, DOI 10.1007/978-1-4419-7371-9_7,

algorithms need to be triggered frequently to meet accuracy requirements. In this case, significant energy and communication cost are inevitable.
– Design new techniques to deal with mobility. Localization algorithms for mobile networks often use some a priori information of node mobility, e.g., maximum velocity, to decrease energy and communication costs and improve location accuracy.

Compared to the first two trivial solutions, the third one is more appealing to large-scale mobile networks and recently has attracted a lot of research efforts. Thus, we focus on this kind of solution in this chapter. Without special statement, when referring to localization algorithms for mobile networks, we mean the third kind of solutions listed above in the rest of this chapter.

The remainder of this chapter is organized as follows. First, we introduce the Monte Carlo localization (MCL) algorithm, which casts the mobile localization problem as a Markov process, and employs sequential Monte Carlo (SMC) methods to resolve it. Then, we present the convex approximation localization (CAL) algorithms. Departing from MCL, CAL maintains a convex polygon or circle to approximate the potential location of each node rather than location samples. Both MCL and CAL focus on using beacon nodes to localize their neighboring nodes and propagate global location information, and update new location based on sequential measurements. We also discuss the moving-baseline localization (MBL) algorithm, whose emphasis is to construct a globally consistent view of the network from the perspective of each individual node, i.e., distances and velocities of other nodes with respect to some node. Finally, some techniques for universal localization (locating both static and mobile nodes simultaneously) are depicted.

7.2 Monte Carlo Localization

7.2.1 Particle Filtering

In applications where the state of a system has to be estimated from some observations, the system can be formulated by using a Bayesian model in which the posterior distribution of the state of the system is only determined by the current observations and state of the system. In dynamic systems, observations arrive sequentially, and it is therefore required to update the posterior distribution of the system state upon arrival of new observations. Formally, the state of the system $\{x_t\}_{t = 0, 1, 2, \ldots}$ is modeled as a Markov process with initial distribution $p(x_0)$ and transition prior $p(x_t \mid x_{t\text{-}1})$. The observations $\{y_t\}_{t = 1, 2, \ldots}$ are assumed to be conditionally independent given $\{x_t\}$ and with marginal distribution $p(y_t \mid x_t)$.

If the initial state and the observations can be modeled by a linear Gaussian state-space model, it is possible to derive an exact analytical expression to compute the evolving sequence of posterior distributions. This is the well-known and widely

accepted Kalman filter. However, the states and observations of real systems are often very complex, typically involving elements of non-Gaussianity, high dimensionality, and nonlinearity, which usually preclude analytic solution. To address this challenge, particle filters, also known as sequential Monte Carlo (SMC) methods, are widely employed in practice, which are a set of simulation-based techniques providing a convenient approach to compute the posterior distributions in non-Gaussian environments.

The key idea of particle filtering is to represent the state distribution of a system by a set of N weighted samples:

$$p(x_t|y_{1,2,\ldots,t}) \approx \left\{x_t^i, w_t^i\right\}_{i=1,2,\ldots,N}, \tag{7.1}$$

where $p(x_t|y_{1,2,\ldots,t})$ is the posterior distribution of the system state at time t given observations $\{y_k\}_{k=1,2,\ldots,t}$, x_t^i is a sample of x_t, and w_t^i is the normalized importance weight associated with x_t^i. The number of samples maintained is an essential system parameter: a minimum of samples should be available so that the set of samples converges to the posterior distribution. A number of particle filters have been proposed in the literature. Here, we illustrate the major steps of particle filtering by introducing the most typical one, bootstrap filter. Actually, all SMC-based algorithms to be discussed in this chapter employ bootstrap filter since it is easy to implement and computationally efficient.

Algorithm 7.1 depicts the three steps of bootstrap filter: initialization, importance sampling, and selection. In the initialization step, N samples are randomly selected according to the initial distribution of the system, $p(x_0)$. In the second step, based on the transition prior $p(x_t \mid x_{t-1})$ and N samples x_{t-1}^i representing the previous

Algorithm 7.1 Bootstrap filter

1 Initialization step: $t = 0$
  for $i = 1$ to N **do**
    sample $x_0^i \sim p(x_0)$
  end for
  set $t = 1$
2 Importance sampling step:
  $W = 0$
  for $i = 1$ to N **do**
    sample $\tilde{x}_t^i \sim p(x_t|x_{t-1}^i)$ and set $\tilde{x}_{0:t}^i = (\tilde{x}_{0:t-1}^i, \tilde{x}_t^i)$
    calculate the importance weight $\tilde{w}_t^i = p(y_t|\tilde{x}_t^i)$
    $W = W + \tilde{w}_t^i$
  end for
  for $i = 1$ to N **do**
    $\tilde{w}_t^i = \tilde{w}_t^i / W$
  end for
3 Selection step:
  resample with replacement N particles $\left\{x_{0:t}^i\right\}_{t=1,2,\ldots N}$ from the set $\left\{\tilde{x}_{0:t}^i\right\}_{t=1,2,\ldots N}$ according to the importance weights
  set $t \leftarrow t + 1$ and go to Step 2

state $x_{t\text{-}1}$, N new samples x_t^i are chosen to represent the current state x_t. The importance weights w_t^i associated with x_t^i are computed based on the marginal distribution $p(y_t|x_t)$ and normalized. In the last step, after resampling according to the importance weights, the particles with small weights are eliminated while those with high weights are multiplied. The finalized N samples are with the same weight, i.e., $1/N$.

7.2.2 Sequential Monte Carlo Localization

From Algorithm 7.1, three distribution functions that are essential to bootstrap filter exist: the initial distribution $p(x_0)$, the transition prior $p(x_t|x_{t-1})$, and the marginal distribution $p(y_t|x_t)$. We discuss these three functions in the context of network localization. Although the Monte Carlo methods can apply to range-based localization for mobile networks [94], we focus on range-free localization here for the simplicity of discussion. The principles presented can be directly applied to the range-based localization.

Initial distribution. In general, nodes initially have no information about their locations except beacons. Therefore, the distribution $p(x_0)$ can be first modeled as a uniform distribution over the whole field of interest. One way to improve the performance of MCL, especially the convergence speed, is to use expensive localization algorithms for static networks, such as manual configuration, to provide relatively accurate $p(x_0)$.

Transition distribution. In the MCL framework, the motion of nodes is modeled as a Markov process: the location of a node at time t, x_t, is only determined by x_{t-1}, its location at time $t-1$. Generally, other than knowing its upper bound speed $v_{\max}$ (probably as a system configuration parameter), a node is unaware of its moving speed and direction. In other words, for each node, x_t must be contained in the circular region centered at x_{t-1} with radius $v_{\max}$. It is also assumed that node speed is uniformly distributed in the interval $[0, v_{\max}]$. Accordingly, the transition distribution $p(x_t|x_{t-1})$ is given by

$$p(x_t|x_{t-1}) = \begin{cases} \frac{1}{\pi v_{\max}^2}, & \text{if } d(x_t, x_{t-1}) \le v_{\max}, \\ 0, & \text{otherwise}, \end{cases} \tag{7.2}$$

where $d(x_t, x_{t-1})$ denotes the Euclidean distance between x_t and x_{t-1}. This distribution function reflects the fact that the unknown motion of a node increases the uncertainty of its location. The larger the value of $v_{\max}$ is, the more uncertainty is introduced in each step. If an accurate mobility model is available, such as an accurate moving velocity or orientation, the transition distribution can be adjusted accordingly to provide better predictions.

Marginal distribution. The marginal distribution $p(y_t|x_t)$ represents the relationship between the observation y_t and the system state x_t at time t. Since we focus on range-free localization here, the only information available to each node is the existence of its neighbors. The 1-hop neighbors of a node u are nodes that can communicate with u directly. The 2-hop neighbors are nodes that communicate with at least one of 1-hop neighbors of u directly but cannot communicate with u directly. We can extend these definitions to k-hop neighbors. In the unit disk graph (UDG) model with communication radius r, for a k-hop neighbor v of u, the following geometric constraints is satisfied:

$$\begin{aligned} 0 \leq |\pi(v) - \pi(u)| \leq r \ \text{ if } \ k = 1, \\ r < |\pi(v) - \pi(u)| \leq k*r \ \text{ if } \ k \geq 2, \end{aligned} \tag{7.3}$$

where π (u) and π (v) denote the physical locations of u and v, respectively. Apparently, the greater the value of k is, the more communication cost and larger localization latency would be introduced, and the geometric constraints become less useful. Therefore, in practice, only 1-hop and 2-hop neighbors are used for location estimation.

Different schemes on using 1- and 2-hop neighbors lead to different marginal distributions. In particular, communication and computation cost trades off with the localization accuracy. Here, we review two existing works on MCL. The work presented in [95] relies on only 1- and 2-hop beacons. On the one hand, it is efficient in terms of communication and computation; on the other hand, abundant beacons are required to provide accurate location results. In contrast, the MSL in [96] uses information from all 1- and 2-hop neighbors, i.e., including beacon nodes and ordinary nodes.

Let $\left\{x_t^i\right\}_{i=1,2,\ldots,N}$ be the sample set representing the location of a node u at time t obtained based on $p(x_t|x_{t-1})$. Let S and T denote the sets of beacons which are 1-hop neighbors and 2-hop neighbors of u, respectively. In [95], the marginal distribution of u is given by

$$p(y_t|x_t^i) = \begin{cases} \frac{1}{N}, & \text{if } \forall s \in S, 0 \leq |\pi(s) - x_t^i| \leq r, \text{and}, \forall s \in T, r < |\pi(s) - x_t^i| \leq 2r, \\ 0, & \text{otherwise}, \end{cases} \tag{7.4}$$

where N is the number of samples maintained by the system as discussed previously. Intuitively, not all samples in $\left\{x_t^i\right\}_{i=1,2,\ldots,N}$ would have nonzero weights. To address this issue, particle filtering adopted in [95] makes some revisions at the "importance sampling" step of Algorithm 7.1: instead of generating N samples, it repeats the sampling procedure until there are N samples with nonzero weights. Since all the N surviving samples are equally weighted, the selection step of Algorithm 7.1 is actually eliminated.

Since information from all 1- and 2-hop neighbors is used, it is very complicated to compute the marginal distribution [96]. Let ST denote all 1-hop and 2-hop

neighbors of node u. For each sample $x_t^i, i = 1, 2, \ldots, N$ chosen for u, $p(y_t|x_t^i)$ is computed by

$$p(y_t|x_t^i) = \prod_{s \in ST} w'_{x_t^i}(s), \tag{7.5}$$

where $w'_{x_t^i}(s)$ is the partial weight associated with s, one 1-hop or 2-hop neighbor of node u. If s is a beacon, $w'_{x_t^i}(s)$ is calculated using (7.4), except substituting 1 for $1/N$. Otherwise, let $\{s_i\}_{i=1,2,\ldots,N}$ denote the sample set for s, and $w'_{x_t^i}(s)$ is computed as follows: if s is a 1-hop neighbor, then $w'_{x_t^i}(s) = \sum_{s_j:|s_j - x_t^i| \le r + v_{\max}} p(y_t|s_j)$; if s is a 2-hop neighbor, then $w'_{x_t^i}(s) = \sum_{s_j: r - v_{\max} \le |s_j - x_t^i| \le 2r + v_{\max}} p(y_t|s_j)$. MSL in [96] employs a threshold to eliminate those samples with low importance weights. The threshold adopted in MSL is $\beta = (0.1)^k$, where k is the number of 1-hop and 2-hop neighbors of a node.

7.3 Convex Approximation Localization

Now, we discuss another kind of localization for mobile wireless networks: convex approximation localization (CAL). Departing from MCL, in which a set of discrete points is maintained to represent the location candidate region of each node, CAL employs a convex polygon or circle to approximate the location region that contains the physical location of a node. The approximation simplifies the locate region update procedure due to node motions and decreases network communication cost. This scheme, however, cannot make use of nonconvex constraints [97]. One typical example of nonconvex constraint is the negative information [98], e.g., a node does not hear from a beacon located at somewhere. In general, however, the negative information plays an important role in refining location estimations. In the rest of the section, we discuss two existing convex approximation localization algorithms: one is convex polygon based, and the other one is circle based.

In [97], each node maintains a convex polygon that represents its location candidate region. At any time, the polygon associate with a node is sufficiently large to contain the physical location of the node, which is the key invariant of this algorithm. The centroid of the polygon is used as the estimated location of the node, and the size of the polygon can serve as a measure of the estimated uncertainty. Since it is often assumed that the radio range of a node is a perfect circle, how to represent a circle is critical for polygon-based localization. Figure 7.1a illustrates an example of using regular polygon to approximate a perfect circle. The major benefit of using polygons is that the intersection of convex polygons can be efficiently computed, compared to circles and curves.

The CAL algorithm proposed in [97] consists of the following three steps:

1. *Initialization*. Beacons start with small regular polygons to approximate a circle centered at the location of the beacon and with radius ε, whose value is chosen to

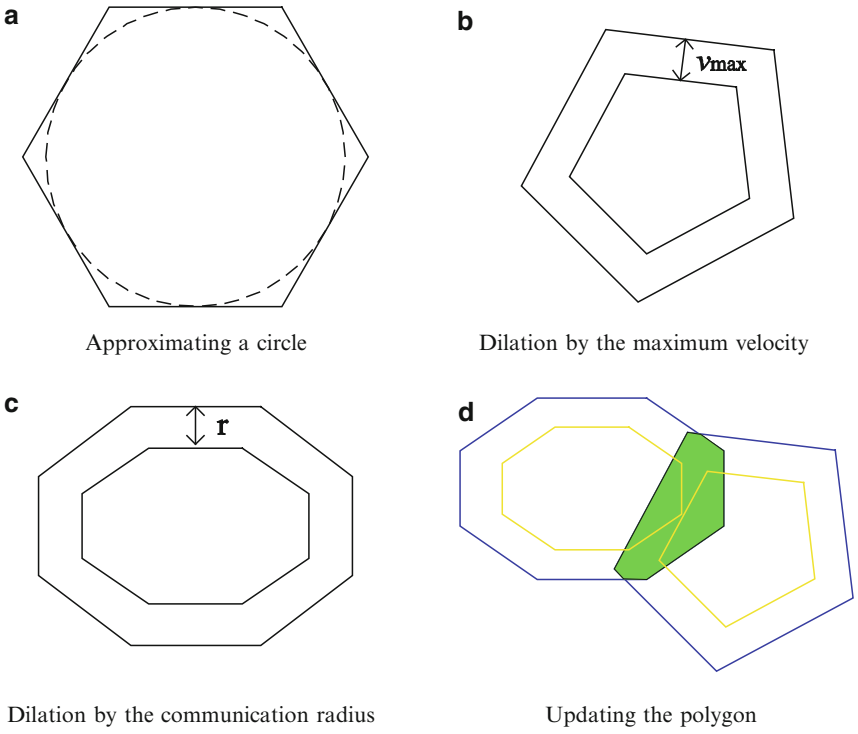

Fig. 7.1 Convex polygon approximation. (**a**) Approximating a circle, (**b**) dilation by the maximum velocity, (**c**) dilation by the communication radius, (**d**) updating the polygon

reflect the uncertainty associated with that estimated location. Since the to-be-localized nodes have no nontrivial a priori knowledge about their locations, they use the whole field of interest as their initial region.

2. *Dilation*. There are two kinds of dilation at each time step. The first one is due to the mobility of nodes. Each to-be-localized node dilates its polygon outward by $v_{\max}$ (illustrated in Fig. 7.1b), the maximum velocity. This dilation maintains the invariant that the true location of the node is in the polygon. The second one is to dilate the polygon by r (illustrated in Fig. 7.1c), the radio range. After two dilations, a node informs the final polygon (the candidate region) to its neighbors.
3. *Update*. Each node receives a set of polygons from its neighbors and then computes the intersection of those received polygons and its own polygon. The result is the new location polygon for that node. Figure 7.1d shows such a procedure. Intersecting a convex polygon with other convex polygons yields a smaller convex polygon with high probability.

CAL has the following salient features: free range, fully distributed, and no need for routing infrastructure, computationally efficient, etc.

Xi et al. [99] propose EUL, which utilizes the relationship between neighboring nodes to update their locations and then filter impossible positions that are out of neighbors' radio range. The algorithm mainly consists of two phases:

1. *Initial location estimation (ILE)*. Each node estimates its initial location by trilateration.
2. *Collaborative neighbor update (CNU)*. Each node updates location boundary according to previous location and neighbors' information.

ILE utilizes an improved DV-hop positioning algorithm to make all nodes know their initial locations, which is the foundation of CNU. In DV-hop, there always are sizable estimated errors in the last hop distances between nodes and seeds. ILE modifies these errors based on the number of second-to-last hop nodes such that the node with more second-to-last hop nodes has a shorter last hop distance. Formally, for an unknown node i, they substitute $h_i - 1 + 1/t_i$ for h_i, where h_i is the hop count from node i to an anchor and t_i is the number of second-to-last hop nodes.

CNU, as the core of EUL, achieves efficient localization with the help of mobility. In this phase, all nodes move freely, which causes a constant change of network connectivity.

Nodes move with variable speed and direction, both of which can be described as random variables. With regard to its current status, a node knows nothing except the maximum speed $v_{\max}$. For the ease of discussion, EUL assumes that nodes have the same radio range r. CNU aims to characterize the area that contains all possible locations of a node. There are two main stages in this phase: prediction and correction.

In the prediction stage, nodes estimate their current locations according to their previous locations and $v_{\max}$. At first, a node predicts possible location set S_1 in a circle whose center is its initial location estimated in ILE and radius is $v_{\max}$. In a subsequent time slot t, S_t is concentric with S_{t-1} are concentric circles, and radius of $v_{\max}$ greater than S_{t-1}. Figure 7.2a shows the relation between S_t and S_{t-1}.

In the correction stage, collaborating with neighbors, each node filters impossible locations. If a node has neighbors, it should be in the overlap of these neighbors' radio range including both new neighbors and constant neighbors. Thus the possible

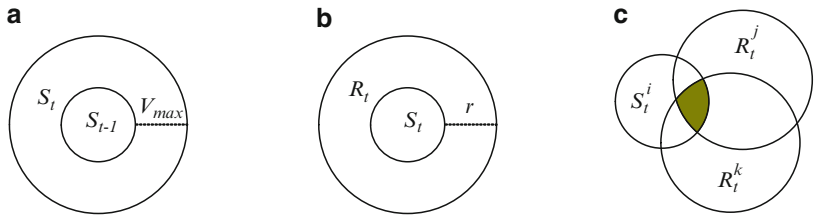

Fig. 7.2 Collaborative neighbors update. (**a**) The prediction of a node's location range; (**b**) the prediction of a node's radio range; (**c**) the correction of a node's location range

radio range should be estimated before filtering. Figure 7.2b shows the relationship between a node's possible radio range R_t and S_t. The node estimates R_t according to S_t and r similar to location range prediction process. Each node computes the overlap region between its possible location set and neighbors' possible radio range as its new location set.

There is a knotty problem that the boundary of a node's location set is irregular. It is hard to describe and compute when the number of neighbors is large. Approximation by maximum inscribed circle (MIC) is an effective method to regularize boundary. The center of MIC is the intersection point of circles, which have common centers with S_t or R_t. That is,

$$\begin{cases} (x_1 - a_1)^2 + (x_2 - b_1)^2 - (x_3 - c_1)^2 = 0 \\ (x_1 - a_2)^2 + (x_2 - b_2)^2 - (x_3 - c_2)^2 = 0 \\ \qquad\qquad \vdots \\ (x_1 - a_n)^2 + (x_2 - b_n)^2 - (x_3 - c_n)^2 = 0, \end{cases} \tag{7.6}$$

where (a, b) and c denote the center and radius of S_t or R_t, and (x_1, x_2) and x_3 are the center and radius of MIC, respectively. When n is greater than 3, the equation set is a nonlinear overdetermined set of equations, which requires nonlinear optimization methods to approximate. Nonlinear least-squares method is used to fit m observations with a model that is nonlinear in n unknown quantities ($m > n$). EUL [99] uses the Gauss–Newton method which is based on linear approximation of the objective function to approximate an optimum solution, where the Jacobian, J, is a function of constants:

$$J(x_k)^{\mathrm{T}} J(x_k) + \nabla f(x_k) = 0. \tag{7.7}$$

7.4 Moving-Baseline Localization

Moving-baseline localization (MBL) [100] deals with the absence of a fixed reference frame. The MBL problem arises when a group of nodes moves in an environment where no external coordinate reference is available. The goal of MBL is to enable each node to infer the spatial relationship and motion of all other nodes with respect to itself.

To model the pairwise interactions among nodes in a mobile network, a concept of dynamic network is introduced in [100], in which an edge between nodes i and j exists if and only if they can exchange information, i.e., it is assumed that when edge ij exists, a discrete sequence of range measurements $r_{ij}(t)$ is available at node i,

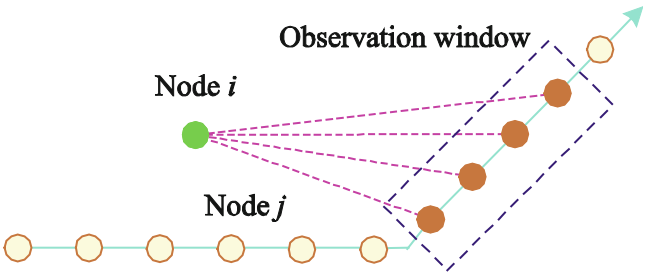

Fig. 7.3 Time-series range data $r_{ij}(t)$

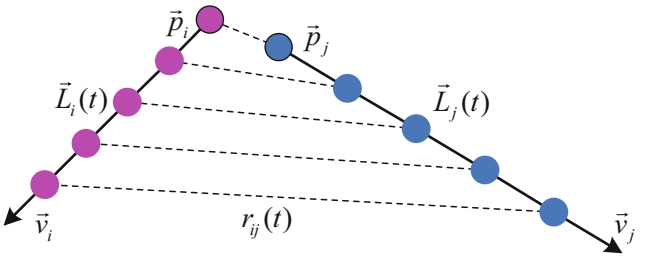

Fig. 7.4 MBL recovers four degrees of freedom per node

describing the measured range from node i to node j at time t observed by node i, as illustrated in Fig. 7.3.

Starting from the simplest case of MBL, where each node is moving along a straight-line path at a constant speed, recovering node trajectories can be cast as a low-dimensional optimization, as shown in Fig. 7.4. Formally, four degrees of freedom per node need to be recovered: the values of $\vec{p}_i$ and $\vec{v}_i$ in the following expression

$$\vec{L}_i(t) = \vec{p}_i + t \cdot \vec{v}_i, \tag{7.8}$$

where $\vec{p}_i$ and $\vec{v}_i$ represent the origin (corresponding to two degrees of freedom) and the velocity vector (corresponding to the other two degrees of freedom) of the motion of node i, and $\vec{L}_i(t)$ is the location of that node at time t. Accordingly, the distance ranges $r_{ij}(t)$ between nodes i and j lie on a hyperbola is defined by

$$r_{ji}^2(t) = m_{ji}^2 + (t - t_{ji}^c)^2 s_{ji}^2, \tag{7.9}$$

where t_{ji}^c denotes the time when nodes i and j are closest to each other, and m_{ji} represents the node separation distance at t_{ji}^c, and s_{ji} is the relative speed, i.e., $\left\|\vec{v}_j - \vec{v}_i\right\|$. These parameters are illustrated in Fig. 7.5. Clearly, as long as $H_{ji} = (s_{ji}, t_{ji}^c, m_{ji})$ is available, the distance r_{ji} between nodes i and j at any specific time can be computed by (7.9), as shown in Fig. 7.5. Because not any pair of nodes i

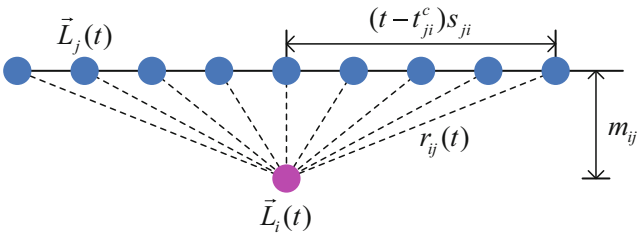

Fig. 7.5 Distance ranges lie on a hyperbola

and j is connected in the dynamic network, to achieve the goal of enabling each node to infer the spatial relationship and motion of all other nodes with respect to itself, an optimal global motion solution is needed to be reconstructed from all available H_{ji}, which involves the following steps.

Hyperbola estimation. The basic block of MBL algorithm is to estimate the motion hyperbola parameters $H_{ij} = (s_{ij}, t^c_{ij}, m_{ij})$ from time-stamped range measurements. Based on the motion model, i.e., (7.9), given a sequence of n discrete distance range measurements between two nodes, $(r_z, t_z), z = 1, 2, \ldots, n$, the following quadratic model is considered:

$$r_z^2 = \gamma t_z^2 + \beta t_z + \alpha + \varepsilon_z. \tag{7.10}$$

Instead of using parametric regression methods like the ordinary least-squares estimator, which are sensitive to data containing significant noise and outliers (modeled as ε_z), to estimate $\hat{\gamma}$, $\hat{\beta}$, and $\hat{\alpha}$, nonparametric robust quadratic fitting is adopted [100]. This method performs well even when the error distribution associated with the data is not normal. When $n \geq 3$, $\hat{\gamma}$, $\hat{\beta}$, and $\hat{\alpha}$ can be determined, and then the motion hyperbola parameters $H = (s, t^c, m)$ are calculated by

$$\hat{s} = \sqrt{\hat{\gamma}},\ \hat{t}^c = \hat{\beta} \Big/ (-2\hat{\gamma}) \text{ and } \hat{m} = \sqrt{\hat{\alpha} - \hat{\beta}^2 \Big/ (4\hat{\gamma})}. \tag{7.11}$$

Based on the recovered parameters $H = (s, t^c, m)$, the hyperbola illustrated in Fig. 7.5 can be reconstructed. In general, more samples can improve the accuracy of the estimation. After computing the motion hyperbola parameters, each node shares this information with its neighbors to estimate local clusters.

Path estimation geometry. The estimated motion hyperbola parameters only capture the relative position and motion of a pair of nodes. To build a local cluster, three relations among nodes i, j, and k are needed to infer the relative motion of the node triangle, just like using distance information of three nodes to build a local coordinate system. From the perspective of node i, this problem can be cast as fixing i at the origin and determining the motions of j and k in the frame of i. Analysis in [100] shows that three motion relations are enough to tackle this problem.

Algorithm 7.2 Local cluster localization

Input: Node i and its neighbors *Neighbors*
Output: *LocalCluster* represented by a set of (ID, position, velocity) tuples of *Neighbors*
$doneNodes = \phi$
Initialize *LocalCluster* as triangle (i, j_0, k_0) by randomly picking j_0, k_0 from *Neighbors*
Add j_0, k_0 to *doneNodes*
for node $j \in doneNodes$ **do**
  for node $k \in Neighbors - doneNodes$ **do**
    if H_{ji}, H_{jk} and H_{ik} are available **then**
      Construct triangle (i, j, k)
      Merge (i, j, k) into *LocalCluster*
      Add k to *doneNodes*
    end if
  end for
end for

Local Cluster Localization. After constructing each triangle in its own frame, each node calculates the motion of its neighbors following a procedure analogous to chained trilateration. The algorithm for local cluster localization is outlined in Algorithm 7.2.

Global View Construction. To get a global view of a dynamic network, we can repeatedly find the best alignment for each pair of local clusters, i.e., merging two local clusters consistently, that share three or more noncollinear nodes, each time adding a local cluster to the major component. This problem is formulated as the absolute orientation problem, which can be solved efficiently by the eigendecomposition. Consider the problem of aligning cluster 2 to cluster 1. Let S denote the common nodes between cluster 1 and cluster 2. For $i \in S$, we use $(P_i^{(1)}, V_i^{(1)})$ and $(P_i^{(2)}, V_i^{(2)})$ to represent the position and velocity of node i in cluster 1 and cluster 2, respectively. The Euclidean transformation from $P^{(2)}$ to $P^{(1)}$ is recovered by minimizing the sum of squared residuals:

$$(\bar{R}, \bar{T}) = \arg\min_{R,T} \sum_{i \in S} \left\| P_i^{(1)} - R(P_i^{(2)}) - T \right\|^2, \tag{7.12}$$

where R corresponds to a rotation, while T is a translation. After solving for $(\bar{R}, \bar{T})$ by the eigendecomposition method, we solve

$$\bar{V} = \arg\min_{V} \sum_{i \in S} \left\| V_i^{(1)} - \bar{R}(V_i^{(2)}) - V \right\|^2, \tag{7.13}$$

to get the velocity offset $\bar{V}$ that best shifts velocities in cluster 2 to align with those of cluster 1.

7.4.1 Techniques for Universal Localization

Universal localization refers to the localization algorithms that can be applied to both mobile and static wireless networks. For static networks or slightly mobile networks, i.e., the maximum velocity of nodes $v_{\max}$ is zero or close to zero, the geometry relationship among nodes changes slowly. In other words, in each step, little information from network measurements is useful for refining the localization results. In this case, the strategy adopted by [96, 99] is to substitute $v_{\max}$ with $v_{\max}+\alpha$. As discussed in [96], there is a trade-off with the value of α. On the one hand, the greater the value of α is, the more uncertainty is introduced, because we use a circle with radius $v_{\max}+\alpha$ instead of $v_{\max}$ to represent the candidate region of each node. On the other hand, if α is small, it cannot provide enough variability in network measurements when the network is static or slowly mobile. Results from [96, 99] demonstrate that setting $\alpha = 0.1r$ works well in practice, where r is the radio range of network nodes.