A decentralized localization scheme for swarm robotics based on coordinate geometry and distributed gradient descent

Vy-Long Dang[1], Binh-Son Le[1], Trong-Tu Bui[1], Huu-Thuan Huynh[1] and Cong-Kha Pham[2]

[1]*University of Science - Ho Chi Minh City, 227 Nguyen Van Cu, Dist. 5, Vietnam*
[2]*The University of Electro-Communications, 1-5-1 Chofugaoka, Chofu, Tokyo 182–8585, Japan*

Abstract. In this paper, a decentralized localization scheme using coordinate geometry and distributed gradient descent (DGD) algorithm is presented. Coordinate geometry is proposed to provide a rough estimation of robots' location instead of the traditional trigonometry approach, which suffers from flip and discontinuous flex ambiguity. Then, these estimations will be used as initial values for DGD algorithm to determine robots' real position. Evaluated results on real mobile robots show an average mean error of 2.56 cm, which is closed to the minimum achievable accuracy of the testing platform (2 cm). For a team of eight robots, the total average run time of the proposed scheme is 66.7 seconds. Finally, its application in swarm robotics is verified by experimenting with a self-assembly algorithm named DASH.

1 Introduction

The studies related to swarm robotics practically focus on how to coordinate large groups of relatively simple robots to carry out complicated tasks such as area exploration, map construction, or harvesting. To achieve highest performance, it is usually necessary for all robots to agree on a global coordinate system as well as their current position in this system.

Conventionally, the work in [1] has proposed a GPS free localization method for indoor applications by using trilateration and trigonometry. However, as pointed out by authors in [2], this method suffers flip and discontinuous flex ambiguity under noisy range measurements. Therefore, they proposed the use of robust quadrilaterals to avoid these errors by excluding nodes that do not meet the noise-threshold requirement. Another approach, which has become the major trend in wireless sensor networks (WSNs) in recent years, is to use global optimization techniques to find the most corrected position of sensor nodes [3, 4]. The limitation of this method is that it requires the use of anchor nodes which usually employ GPS to have prior knowledge of their actual position.

Another problem in determining a localization scheme for both WSNs and swarm robotics is that it has to be scalable, which means the algorithm can be applied in a system with hundreds or thousands of nodes. Therefore, centralized approaches, which use a center node to distribute network information to all nodes [5], is not favorable as they puts too much strain on the communication channel in large networks. Thus, decentralized approaches, where each node only needs to communicate with its neighbors, is usually applied [1–4]. These approaches are further divided into synchronous and asynchronous methods. Synchronous algorithms mean the state of all nodes in the system are synced. In other words, a node can only go to the next state when all nodes in the system have finished the current state. On the other hand, asynchronous algorithms mean each node can freely change its state without the need to wait for others, thus, it is faster but also more complicated than synchronous ones.

In this work, a synchronous decentralized localizations scheme for swarm robotic platforms is proposed. Instead of using trigonometry as in [1, 2], a coordinate geometry method was developed to avoid flip and discontinuous flex ambiguity. To further improve the accuracy of localization results, distributed gradient descent (DGD) algorithm [4] is also integrated into this scheme. In order to verify effects of real noises in range measurements to the proposed approach, it has been implemented on eight mobile robots. Similar to other swarm-robotic platforms, such as Kilobot [6] and E-puck [7], the designed robots can measure their distance to their neighbors, with a mean error of 2 cm, and communicate through wireless signals. Evaluated results show an average mean error of 2.56 cm, which is closed to the minimum achievable accuracy of the testing platform (2 cm). For a team of eight robots, the total average run time is 66.7 seconds. To verify that this localization scheme can be used to provide robots' position for more complicated swarm robotic applications, the self-assembly algorithm DASH [8] is also implemented and shows good results.

The rest of the paper is organized as follows. Section 2 presents the proposed localization scheme. Section 3 discusses the testing platforms. The evaluation results are shown in section 4. Finally, section 5 draws the conclusion.

2 The proposed localization scheme

2.1 State 1: Distance measurement

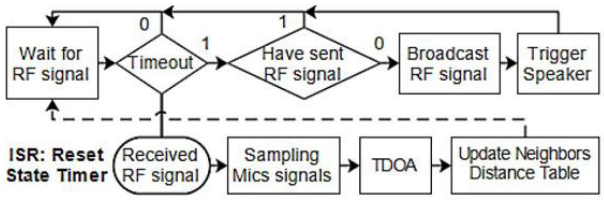

Figure 1. Distance measurement flow chart.

The flow chart of this state is shown in Fig. 1. At the start of the localization scheme, each robot will wait for a random period of time before broadcasting a message to its neighbours. The robots which receive this message will trigger a distance sensing algorithm. The measured results will be stored in a table and sent back to the broadcasting robot. After receiving all measured results, this robot will continue to listen for the same broadcasting message. If none is received until the timer stops, it will jump to the next state. At the end of this state, every robot will have one table storing its distance to its neighbours and their ID.

2.2 State 2: Exchange neighbours distance table

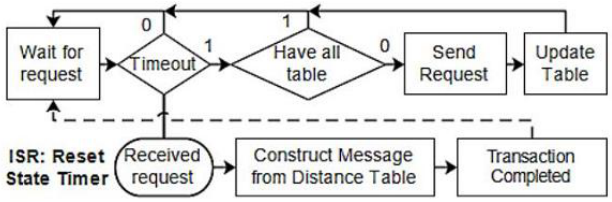

Figure 2. Exchanging measured results flow chart.

Using the ID from the table of state 1, each robot will request the table of its neighbours as shown in Fig. 2. Thus, every robot will now have a second table which stores the first table of its neighbours.

2.3 State 3: Trilateration

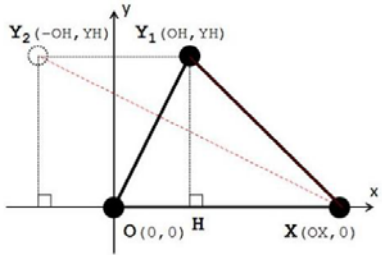

Figure 3. The coordinate of three robots in phrase 1.

Similar to [1, 2], this state has two phrases. Each robot will now establish its own local coordinate system based on the tables from state 1 and 2. It should be noted that no information is exchanged during this state.

2.3.1 Phase 1

At this state, every robot sees it as the origin, $robot_O$ or point O. Two neighbours which do not lie on the same line will be chosen. One is assumed to have the position $(d_{OX}, 0)$, $robot_X$ or point X, and the other is assumed to have a positive y coordinate, $robot_Y$ or point Y. The location of $robot_Y$ is calculated by using the triangle ΔOXY. The length of YH edge is calculated by Herons formula. The length of OH edge is determined by using Pythagoras theorem. Because $robot_Y$ has positive y coordinate, it only has two possible locations: Y_1 and Y_2. However, the correct position will have the minimum error between the distance vector and the measured distance as described in step 8 of Algorithm 1. Positions of these three robots will be stored in a third table.

Algorithm 1 Establish a local coordinate system

1: $\overrightarrow{O} \leftarrow (0,0)$
2: $\overrightarrow{X} \leftarrow (d_{OX}, 0)$
3: $s \leftarrow \frac{d_{OX}+d_{OY}+d_{XY}}{2}$
4: $d_{YH} \leftarrow \frac{2\sqrt{s(s-d_{OX})(s-d_{OY})(s-d_{XY})}}{d_{OX}}$
5: $d_{OH} \leftarrow \sqrt{d_{OY}^2 - d_{YH}^2}$
6: $\overrightarrow{Y_1} \leftarrow (d_{OH}, d_{YH})$
7: $\overrightarrow{Y_2} \leftarrow (-d_{OH}, d_{YH})$
8: **if** $|d_{XY} - \|\overrightarrow{XY_1}\|| < |d_{XY} - \|\overrightarrow{XY_2}\||$ **then**
9: $\quad \overrightarrow{Y} \leftarrow \overrightarrow{Y_1}$
10: **else**
11: $\quad \overrightarrow{Y} \leftarrow \overrightarrow{Y_2}$
12: **end if**

2.3.2 Phase 2

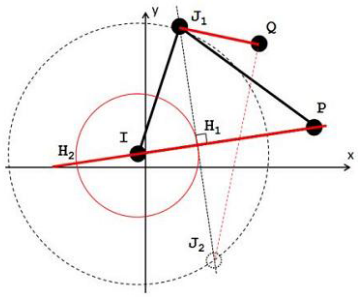

Figure 4. Finding the position of point J from three known points I, Q, and P.

Traditionally, by using trigonometry [1], we can locate the position of an unknown point J from three known points I, P, and Q, as long as they do not lie on the same line. But this approach is not accurate in noisy range measurements and can result in errors such as flip ambiguity or discontinuous flex ambiguity, which are well explained in [2]. To overcome this issue, we propose another approach as explained in Algorithm 2 and below:

• **Step 1 to 3**: As shown in Fig. 4, the linear equation $D_{(IP)}$ in general form can be constructed from any two already located robots.

• **Step 4 to 13**: Calculating the location of point *H*, with *JH* is the semiperimeter of the triangle Δ*IPJ*. From the circle equation, we find two possible locations of *H*, H_1 and H_2. The length of *IH* is obtained by applying Pythagoras theorem to the right triangle Δ*IHJ*. Similar to phrase 1, the correct location of point *H* has the minimum error between the distance and the measured distance.

• **Step 14 to 23**: Since *JH* and *IP* are two perpendicular lines, we can construct the linear equation in general form. Again, get two possible locations of $robot_J$. Finally, by comparing the error between distance vectors and the measured distances, we get the correct location.

Algorithm 2 Calculate position of a neighbor robot

$\overrightarrow{IP_x} \leftarrow P_x - I_x$
$\overrightarrow{IP_y} \leftarrow P_y - I_y$
Line equation $D_{(IP)} : (-IP_y)x + (IP_x)y + (IP_y \times I_x - IP_x \times I_y) = 0$
$s \leftarrow \frac{d_{IP}+d_{PJ}+d_{IJ}}{2}$
$d_{JH} \leftarrow \frac{2\sqrt{s(s-d_{IP})(s-d_{PJ})(s-d_{IJ})}}{d_{IP}}$
Circle equation $C_{(I,d_{IH})} : (x-I_x)^2+(y-I_y)^2 = d_{IH}^2$
$\overrightarrow{H_{1,2}} \leftarrow$ solve $D_{(IP)} = C_{(I,d_{IH})}$
$d_{PH} \leftarrow \sqrt{d_{PJ}^2 - d_{JH}^2}$
if $|d_{PH} - \|\overrightarrow{PH_1}\|| < |d_{PH} - \|\overrightarrow{PH_2}\||$ **then**
  $\overrightarrow{H} \leftarrow \overrightarrow{H_1}$
else
  $\overrightarrow{H} \leftarrow \overrightarrow{H_2}$
end if
$D_{(JH)} : (IP_x)x + (IP_y)y + (-IP_y \times H_y - IP_x \times H_x) = 0$
$C_{(I,d_{IJ})} : (x - I_x)^2 + (y - I_y)^2 = d_{IJ}^2$
$\overrightarrow{J_{1,2}} \leftarrow$ solve $D_{(JH)} = C_{(I,d_{IJ})}$
if $|d_{QJ} - \|\overrightarrow{QJ_1}\|| < |d_{QJ} - \|\overrightarrow{QJ_2}\||$ **then**
  $\overrightarrow{Robot_j} \leftarrow \overrightarrow{J_1}$
else
  $\overrightarrow{Robot_j} \leftarrow \overrightarrow{J_2}$
end if

As can be seen, the coordinate of other robots can now be calculated as long as they are the neighbor of at least three already located robots. In other words, their ID appears in the first table of $robot_O$, $robot_X$, and $robot_Y$. Algorithm 2 will go through all neighbors in the first table of $robot_O$ and try to find their position. If new robots are successfully localized, their position and ID will be stored in the third table. Obviously, they can now also be used to locate others. Therefore, whenever a new position is added, the algorithm will go through all remaining neighbors of $robot_O$ again.

To further reduce localization errors before going to the next state, each robot will use the gradient descent algorithm to optimize its calculated results as described in Algorithm 3, with $\nabla_i F(\overrightarrow{Robot_i})$ is obtained by (1), where N_i is the group of neighbors of $robot_i$ which are also belong to the local coordinate system of $robot_O$. At the end of this state, tables from state 1 and 2 are no longer needed and can be removed to free memory. The table obtained from this state is called the local coordinate table.

$$\nabla_i F(\overrightarrow{Robot_i}) = \sum_{j \in N_i} \frac{(\overrightarrow{Robot_{ij}})}{\|\overrightarrow{Robot_{ij}}\|} (\|\overrightarrow{Robot_{ij}}\| - d_{ij}) \quad (1)$$

Algorithm 3 Gradient Descent

while true **do**
  for $robot_i \in$ `thirdTable` **do**
    if $|\overrightarrow{Robot_i} - \overrightarrow{Robot_{iold}}| >$ `allowedError` **then**
      $\overrightarrow{Robot_{iold}} \leftarrow \overrightarrow{Robot_i}$
      $\overrightarrow{Robot_i} \leftarrow \overrightarrow{Robot_{iold}} -$ `stepSize`$\times \nabla_i F(\overrightarrow{Robot_{iold}})$
    end if
  end for
  if no new update in `thirdTable` **then**
    End algorithm
  end if
end while

There are two reasons why our proposed algorithm is more accurate than trigonometry approach. Firstly, in the conventional method, angles are used to determine the sign of *y* coordinate of a robot. As distance measurements are not accurate, this comparison has to tolerate a noise margin. If this margin is too small or too large, the calculated *y* coordinate is undefined and will cause a flip ambiguity. This error can happen even with robots having high *y* value. On the contrary, our method uses the error between calculated and measured distances. Therefore, a large error can only happen when $robot_Q$ lies near *x* axis. Obviously, this issue can be solved easily by chosen the robot farthest from *x* axis as robot *Q*. Secondly, the conventional method only uses a subset of distance measurements to localize a point while in our proposed scheme, all distance measurements will be used in the gradient descent algorithm to minimize calculated errors.

2.4 State 4: Create a unique global coordinate system

After their local coordinate is established, a robot will be chosen as the global origin based on two simple rules:

• The higher number of neighbours, the higher priority.

• If the number of neighbours is equal then the smaller ID, the higher priority.

Each robot will broadcast a vote message containing its ID and its number of located neighbours from state 3. When a robot receives the higher priority vote, it will replace its current vote and broadcast this vote to its neighbours. Otherwise, it will continue to broadcast its current vote. After the origin robot has been determined, its local coordinate system will be considered as the global coordinate and no further changes are necessary. Then, this robot will request its neighbour to rotate their coordinate table to the global one. After these neighbours have rotated, they will continue to request their neighbours to rotate and so on. In short, this process starts

at the origin robot and spreads out to the whole system. The rotation algorithm is the same as the one proposed in [1].

As a result of randomly choosing $robot_X$, $robot_Y$, and the global origin, the network direction will differ each time the localization scheme is run. If this is undesirable, these 3 robots can be predetermined in state 3 and the origin voting phrase can be skipped.

2.5 State 5: Errors correction & position updating

Because of the use of angle rotation in state 4, the localization results will have some inaccuracies such as reflection error [2]. For example, $robot_I$ and $robot_J$ successfully locate $robot_K$ position. If the coordinate table of these robots rotates a correct angle, then the position of $robot_K$ will be the same in both tables. However, if an error occurs, the position of $robot_K$ stored in $robot_I$ will differ with the one stored in $robot_J$. Furthermore, these errors will propagate and become larger at farther nodes. To correct this, Distributed Gradient Descent (DGD) algorithm, proposed in [4], is modified to apply to swarm robotics as described briefly in Algorithm 4. It should be noted that $\nabla_i F(\overrightarrow{Robot_i})$ is also calculated similarly as in (1).

Algorithm 4 Distributed Gradient Descent

1: **if** receive run DGD request **then**
2: **if** Do not have a global location b yet **then**
3: Gets its position from the coordinate table of its neighbors.
4: b is the average value of these positions
5: **end if**
6: $l \leftarrow 0$
7: $\underline{b}(l) \leftarrow b$
8: **while** $|\underline{b}(l) - \underline{b}(l-1)| >$ `allowedError` **do**
9: **for** $robot_i \in Neighbors$ **do**
10: Ask $\underline{b}(l)$ of $robot_i$
11: Update $robot_i$ position in this robot coordinate table with $\underline{b}(l)$
12: **end for**
13: $\underline{b}(l+1) \leftarrow \underline{b}(l) -$ `stepSize`$\times \bigtriangledown_i F(\underline{b}(l))$
14: $l \leftarrow l + 1$
15: **end while**
16: Global position: $b = \underline{b}(l)$
17: **end if**

Another important feature in robotic localization is that when a robot moves, it has to quickly update its new location. To do this, after new distance measurements are updated, Algorithm 4 is run again. However the position of neighbors (step 9 to 12) will only need to be asked one time. Obviously, it still has to have at least three already located robots as its neighbors. This updating method not only offers fast response speed but also high accuracy.

3. The designed testing platform.

To verify the proposed localization scheme, the following requirements are expected:

- Each robot can measure the distance from itself to its neighbours.
- Each robot can communicate with its neighbours using wireless signals.

3.1 The robotic platform

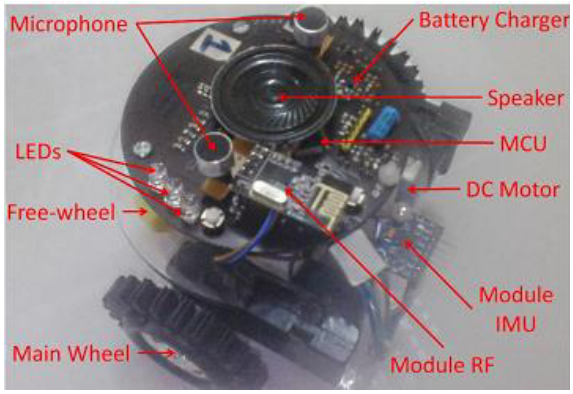

Figure 5. The designed robot.

The testing platform includes 8 mobile robots. Each has the same main board which consists of a micro-controller (ARM Cortex-M4F TM4C123GH6PM), an RF module (nRF24L01), two electret microphones with two two-stage amplifiers, a speaker, three status LEDs, a 6 degrees of freedom IMU, and a battery management circuit as shown in Fig. 5. The total part cost of each robot is $20.68.

3.2 The Control Board and Monitoring Software

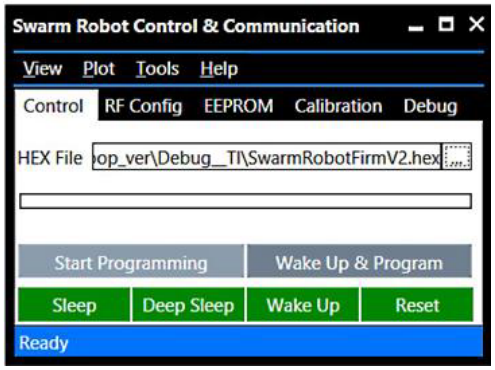

Figure 6. The monitor software.

Without a tool to command and communicate with robots, it is hard to develop and debug collective algorithms. To get over this situation, a wireless control board and a monitoring software (Fig. 6) using C# have been developed. The control board is connected to a PC using USB connection and will send user's commands to all or selective robots. In addition, a wireless bootloader protocol is also implemented to update all robots' firmware at the same time.

3.3 Distance Sensing

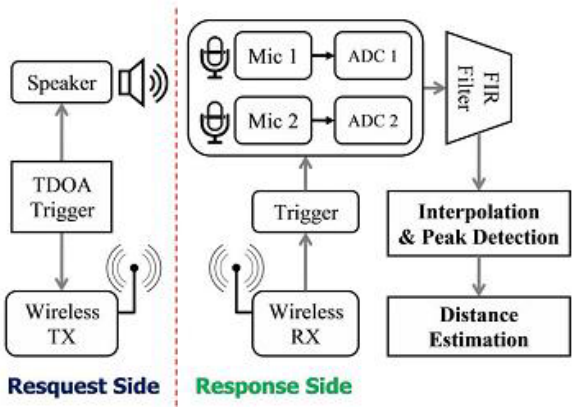

Figure 7. The distance measurement scheme proposed in [9].

For distance sensing purpose, the system and TDOA (time difference of arrival) algorithm proposed in [9] (Fig. 7) are implemented. First, the request side will broadcast an RF signal to other robots, then wait for a small amount of time before emitting an 8 KHz signal. Upon receiving the RF signal, the response side will turn on their microphones to sample the audible signal. The difference between the arrival time of the RF signal and the audible signal will be used to calculate the distance between the response and the request side. Similar to [9], each of our robot also stores two calibration parameters: S_0 (intercept) and M_s (slope), which are extracted using least square optimization. The maximum measurable distance of our system is 47.5 cm and the mean error is 2 cm.

4. Evaluation results

The proposed scheme is evaluated based on the error between measured results and known ground truth as expressed in (2), where, bx_i and by_i are the true position of $robot_i$, and x_i and y_i are its localization results.

$$\sigma_p = \sum_{N}^{i=1} \frac{(bx_i - x_i)^2 + (by_i - y_i^2)}{N} \qquad (2)$$

Table 1. Evaluation results from 10 test cases.

Case	Min	Average	Max
σ_p (cm)	2.21	2.56	3.52
Run Time (seconds)	54.5	66.7	96.7

Testing with 10 different cases (by varying the number of robots and their formation), as shown in Fig. 8, the observed average mean error is 2.56 cm. For an ideal localization algorithm, its σ_p cannot be lower than the mean error of distance measurements (2 cm in our system) [10]. Therefore, this result shows that the proposed localization method achieves high accuracy. Table I summaries the evaluated results and the processing time. Since there are several states require random wait time, the run time is affected heavily and fluctuates from 54.5 to 96.7 seconds, with an average value of 66.7 seconds. It should be noted that because our approach is a synchronous decentralized method, its scalability is better than centralized methods. However, because each robot has to wait for others to finish their state, an asynchronous decentralized approach may be employed in further researches to improve runtime.

Since localization results are the foundation for more complicated swarm-robotic applications, the proposed method is continued to be verified by combining it with the self-assembly algorithm DASH [8]. The localization scheme will provide robots' initial position as well as update their location after moving as described in the final state of section 2. The result is shown in Fig. 9, which successfully changes robots' formation from a star shape to a rectangular shape.

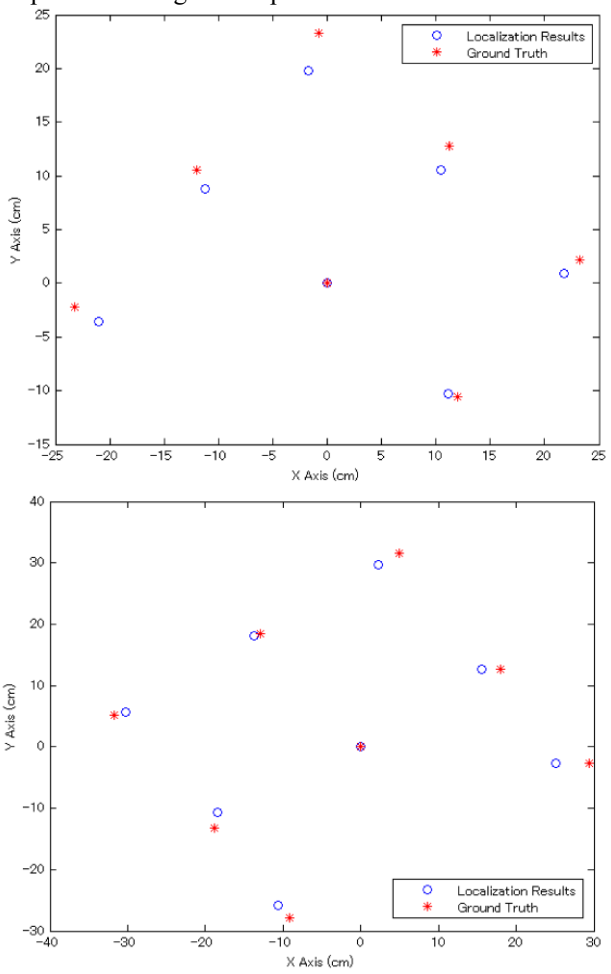

Figure 8. Localization results of (a) 7 (b) 8 robots.

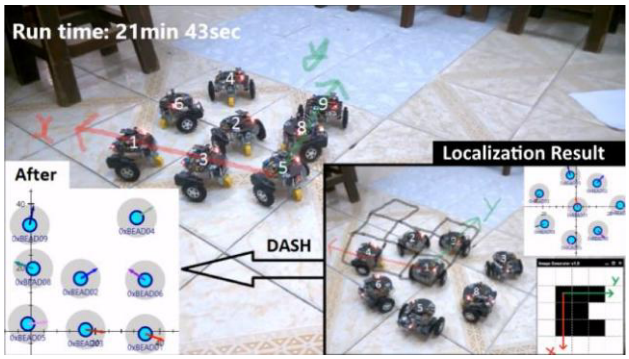

Figure 9. DASH algorithm testing result.

5. Conclusion

In this work, we have presented a new localization scheme using coordinate geometry to avoid flip and discontinuous flex ambiguity errors in the conventional method. After initial positions are produced, a global optimization using DGD is carried out to improve the accuracy of the whole system. The evaluation results on eight mobile robots, which have limited sensing capabilities, show high accuracy and good processing time. However, better run time can be achieved by employing an asynchronous decentralized approach. The proposed approach can also be applied in WSNs when GPS signals are not available. Finally, the proposed method is used to provide robots' position for a self-assembly algorithm named DASH and shows good results.

References

1. S. Capkun, M. Hamdi, J. P. Hubaux, *Hawaii Int. Conf. System Sciences*, 3 (2001)
2. D. Moore, J. Leonard, D. Rus, S. Teller, *Int. Conf. Embedded Networked Sensor Systems*, 50 (2004)
3. Q. Shi, C. He, H. Chen, L. Jiang, IEEE Trans. Signal Processing, **58**, 3328 (2010)
4. G.C. Calafiore, L. Carlone, W. Mingzhu, IEEE Trans. Systems, Man and Cybernetics, **42,** 1065 (2012)
5. T.G. Karimpanal, M. Chamambaz, W.Z. Li, T. Jeruzalski, A. Gupta, E. Wilhelm, *FinE-R*, 20 (2015)
6. M. Rubenstein, C. Ahler, R. Nagpal, *IEEE Conf. Robotics and Automation*, 3293 (2012)
7. A. Gutierrez, et al., *IEEE Conf. Robotics and Automation*, 3111 (2009)
8. M. Rubenstein and W. Shen, *IEEE Conf. Intelligent Robots and Systems*, 1484 (2009)
9. G. Sineriz, M. J. Kuhlman, P. A. Abshire, *IEEE Sym. Circuits and Systems*, 717 (2012).
10. A. Savvides, W. Garber, S. Adlakha, R. Moses, M. B. Strivastava, *ISPN*, 317 (2003)