2014 IEEE International Conference on Robotics & Automation (ICRA)
Hong Kong Convention and Exhibition Center
May 31 - June 7, 2014. Hong Kong, China

Timed Automata Based Motion Planning for a Self-assembly Robot System*

Rui Wang[1], Ping Luo[1], Yong Guan[1], Hongxing Wei[2], Xiaojuan Li[1] Jie Zhang[3], Xiaoyu Song[4]

Abstract— **Sambot is a module robot system, with the advantages of self-assembly. A target robotic configuration can be organized by a group of Sambots. A novel motion planning method for Sambot configuration using model checking is presented in this paper. This hierarchical method contains two layers. The abstract logic layer is responsible for the discrete planning of Sambots configuration. The robot and the environment are all modeled as timed automata. System requirements are formalized as Computational Tree Logic (CTL) formulas. Model checking is applied on the system model. The verification result gives the optimal discrete plans for the configuration of Sambot. In physical layer, a sample-based planner generates the trajectory trace considering the dynamics of Sambot and the suggested high level plans. The experiment results illustrate the effectiveness of our approach.**

I. Introduction

Distributed multi-robot systems have become an attractive field of research in the robotics community. Sambot [1], [2] is a multi-robot system developed by the Beihang University of China. Sambot can organize a group of robot modules into a target robotic configuration without human intervention. Similar to the swarm behavior of social insects [3], self-assembly is the process by which a group of smart components autonomously organize into configurations or structures [4]. Sambots have the advantages of both swarm robots and modular robots.

The correctness of a system plays a critical role in high dependability. Formal methods are an important means for analyzing and ensuring the reliability of complex systems. Model checking has proved to be a powerful automatic verification technique [5]. It has been successfully applied to hardware design and the verification of communication protocols. In recent years, this technique has been used to verify and synthesize the problem of motion planning in the field of robotics.

Many researchers have turned their attention to self-assembly algorithms and simulations in multi-robot systems. Jones and Mataric proposed the concept of Intelligent Self-assembly, where each module has limited and local sensing capability and local rule-based control, which can be employed to assemble the desired target [6]. Kelly and Zhang developed a stochastic optimization algorithm to build a structure by self-assembly [7]. Grady et al. establish a distributed self-assembly control method for a given target configuration called SWARMORPH [8].

Formal methods were adopted in the creation of control for modular robotics [9]. Tasks were expressed in structured English and then translated to LTL formulas to synthesize the control plan, but real-time character cannot be modelled. A geometry-based multilayered approach was proposed to solve motion planning with temporal goals [10]. This method combined the sample based planning and LTL synthesis. Many researchers have adopted a formal method for multi-robot coordination. A plan-merging paradigm was used to guarantee the coherent behavior of robots in all situations [11]. Hybrid automata were applied to model and verify the coordinate control method [12]. Paper [13] presented Petri Net Plans (PNP) based on a distributed execution algorithm for high-level robot and multi-robot programming. [14] used timed automata to model and verify the motion planning problem. Robots and obstacles were modeled as a network of timed automata. For the motion planning in this paper, we do not know the exact allocated destination of every robot. We only know whole configuration. Our planning method can find the optimal traces and corresponding destination of each robot. The correctness of the results is guaranteed by model checking.

In this paper, we present a multi-layer method for modeling and verifying the motion planning problems of a Sambot system. In the abstract logic layer, the seed and Docking Sambots (DSA) are all modeled by timed automata. The components are synchronized by passing signals through channels. The system requirements are specified and formalized as CTL [15] properties. The shortest traces of configurable property indicate the planning traces for multi-robots. We demonstrate that the design model satisfies the required properties, by using a symbolic model checker *Uppaal* [16], [17]. The model used in *Uppaal* extends timed automata with bounded integer variables, data structure, and location urgency [18], [19]. In the physical layer, the trajectory trace that considers the dynamic model of Sambot and suggested high level plans is given by sample-based planning. This two layer method can get a approximate discrete plan rapidly and then generate a practical trace considering the dynamic equation. An example is given to illustrate the strategies that are employed. The experimental results demonstrate the

*This research was supported by the BJNSF (4122017),NSFC(61373034,61303014) ISTCP (2011DFG13000,2010DFB10930)

[1]Rui Wang, Ping Luo, Yong Guan and Xiaojuan Li come from Beijing Engineering Research Center of High Reliable Embedded System, Beijing Key Laboratory of Electronic System Reliability Technology, College of Information Engineering, Capital Normal University, Beijing, China, 100048. rwang04@163.com

[2]Hongxing Wei with the School of Mechanical Engineering and automation, Beihang University, Beijing, China.

[3]Jie Zhang with College of Information Science and Technology, Beijing University of Chemical Technology, Beijing, China

[4]Xiaoyu Song with the ECE Dept, Portland State University, Portland, USA

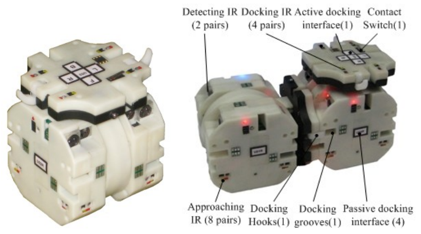

Fig. 1. Photographs of Sambot

effectiveness of our approach.

The paper is organized as follows. Section II contains a detailed description of the Sambot system and its system requirements. The theory on timed automata is given in Section III. In Section IV, the motion planning method and formal models of the Sambot system are presented. The validation result is shown in Section V. Section VI concludes the paper.

II. Sambots System

The Sambot is a self-assembly modular robot system with the advantages of self-reconfigurability and self-assembly. It has an overall size of $80mm \circ 80mm \circ 102mm$ and weighs $400\ g$. Every robot contains a power supply, micro-controller, actuators, sensors, and a communication system. Each Sambot has some infrared sensors that detect the target and guide the process of self-assembly. As shown in Fig.1 the upper side and the front and back sides of Sambot are equipped with infrared sensors. A hook-groove mechanism is used to realize the docking. Once preparation for docking is ready, two robots can connect together with the connector structure. There is a pair of grooves on the front, back, left, and right sides of Sambot. During docking, the hooks can be completely inserted into the grooves of another Sambot to finish the process of docking and locking. Each Sambot is capable of performing computations and decisions. They can communicate with each other and with the main computer to get the control input through a wireless network.

A. Self-assembly of Sambots

A Sambot is an autonomous mobile robot that is realized by wheels. A Sambot can form robotic organisms connected with one another to carry out some tasks. In our previous work [2], we explored the use of local communication and simple finite state machine (FSM) to produce self-assembly control algorithms.

On a Sambot platform, three types of Sambot coexist, including Docking Sambots (DSA), SEED, and Connected Sambots (CSA). First, SEED starts the self-assembly process. Then, the DSA achieves self-assembly with SEED under the command of the behavior-based controller that is independent of the target configuration. SEED and CSA execute a configuration comparison algorithm to control the configuration growth of the robotic organisms. The process of the experiment is shown in Fig.2. At first, the seed is in the middle of the workspace, four DSAs are in the corner.

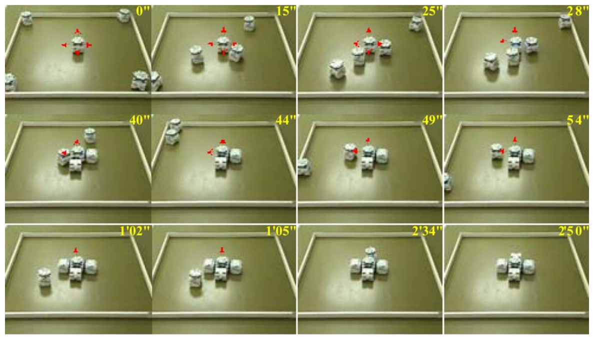

Fig. 2. The assembling process of Sambot

After the assembling process, a cross is obtained at last. The main disadvantage of previous method is that these DSAs search the SEED randomly and aimlessly. If a DSA goes to a opposite direction, it may cost a lot in terms of time.

In this paper, we first plan the shortest traces of all DSAs with the model checking method. Then, under the constraint of dynamic functions, we design the control input of DSAs so that the DSAs can move along the predesigned trajectory and lock to the SEED without collision. Before planning, we need to know the locations of the DSAs and the SEED.

B. System Requirements

In this system, safety is of the utmost importance. The DSAs move within a restricted space. It is necessary to ensure that these DSAs do not crash into each other. The DSAs can initially be located in any place. In the original version, the DSA searches randomly by itself to find possible ways of achieving aim configuration. However, this process takes a long time. In addition, DSAs might crash into each other. Our intention was therefore to develop a planning method to first get the traces rapidly, and then avoid collision. The system requirements are listed below.

1) *Collision avoidance*: DSAs should not crash into each other.
2) *Configurable*: all DSAs should reach the destination and complete the configuration.

III. The Theory of Timed Automata

Timed automata [20], [21] are finite automata with extension of time variables. Let V be a finite set of variables including clock variables C and data variables X, $V = C \cup X$ and $C \cap X = \emptyset$. We use $\psi(V)$ to denote invariant and guard formulas. We have $\psi ::= e \mid \psi \wedge \psi$, e takes the form like $c \sim n$ or $x \sim n$, $c \in C, x \in X$, $\sim \in \{\leq, \geq, =, <, >\}$ and $n \in \mathbb{N}$. The assignment operation is defined as $v := expresion$, $v \in V$. Let U denote all the assignment formulas.

Definition 1 Timed automaton is a tuple $\mathcal{A}(L, l_0, A, V, I, E)$

- L: a finite set of locations
- l_0: initial locations
- A: a finite set of actions

- V: a finite set of variables
- $I : L \to \psi(V)$ a location constraint function
- E: a finite set of edges

$E \subseteq L \circ \psi(V) \circ A \circ U \circ L$. Each edge has a source location l, a target location l'. When guard $g \in \psi$ is satisfied, the transition happens and a subset of variables in V are updated by formula $r \in U$. An edge $e\langle l, g, a, r, l'\rangle$ can be written as $l \xrightarrow{g,a,r} l'$.

The automaton starts at the initial state l_0 with all clocks initialized 0. With time passing by, the clock variables increase at the same rate satisfying the invariant constraints $I(l_0)$. The system can remain still in this location or transit to l_1 if the variables satisfy an edge enabling guard g. With the transition, action a is taken and variables are updated by formula r.

Definition 2 The semantics of a timed automaton $\mathcal{A}(L, l_0, A, V, I, E)$ is defined as a labeled transition system $\mathcal{S}(\mathcal{A}) = \langle S, s_0, \to\rangle$. $S \subseteq L \circ mathbbR$ is a set of states, s_0 is the initial state, $\to \subseteq S \circ (U \cup A) \circ S$ is the set of relations, divided into the following two cases.

- Elapses of time transitions: for $d \in \mathbb{R}^+$, $(l, u) \xrightarrow{d} (l, u + d)$, if for $\forall d' \leq d$, u and $u + d'$ satisfy $I(l)$, and
- Location switch transitions: $(l, u) \xrightarrow{a} (l', u')$, if $\exists e\ (l, a, g, r, l') \in E$, $u' = r(u)$, u satisfies guard g, $r \in U$ and u' satisfies $I(l')$.

A complex is composed of components. The controller communicates with the environment concurrently. We introduce timed automata networks $\overline{\mathcal{A}} = \mathcal{A}_1 \parallel ... \parallel \mathcal{A}_n$, and $\mathcal{A}_i = (L_i, l_i^0, A_i, V_i, I_i, E_i)$, n is the automata number in the network. These automata share a common set of action variables. Vector $\bar{l} = (l_1, ..., l_n)$ is the location vector of timed automata network $\overline{\mathcal{A}}$. The invariant function $I(\bar{l})$ is the conjunctions of the constraints of all $\mathcal{A}_i$, $I(\bar{l}) = \wedge_i I_i(l_i)$. $\bar{l}[l_i'/l_i]$ denotes that l_i of $\overline{L}$ is replaced by l_i'.

IV. Motion Planning Method

In this paper, we try to find a solution to the following problem:

Problem: Given a configuration task and a seed, find a global trajectory plan for all robots with the least amount of moving time, and at the same time, avoids collisions. The proposed hierarchical planning method is shown in Fig.3. The prior knowledge of this method is the location of seed and the location of the DSAs. The hierarchical method consists of two layers. The highly abstract logic layer constructs the timed automata representing the workspace through a partition of the environment and the discrete behavior of robots. Collision avoidance, time bound, and configurable properties are formulated as CTL formulas. The model checker tools can generate high-level motion commands of moving to an adjacent region based on the optimal time constraint. These discrete plans are sent to a low-level physical layer, which generates the trajectory based on the dynamic equations of Sambot. The control input drives the Sambots through cells and forms the target configuration. The hierarchical method reduces the scale of model checking to avoid state space explosion, and also considers the physical features of Sambots.

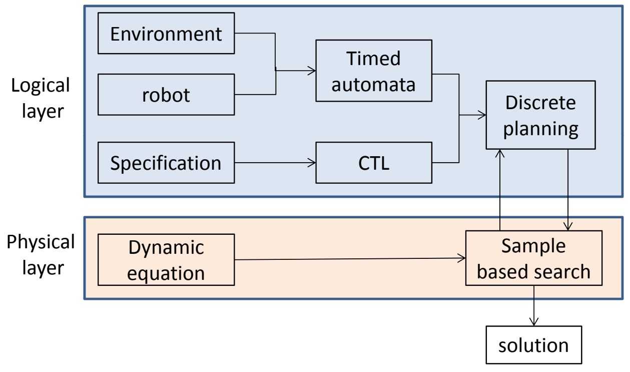

Fig. 3. Hierarchical planning method

A. Problem Formulation

1) Prior knowledge: We assume that the workspace is rectangular with the size of M. SEED and DSAs can be located at any place in the workspace. Given the target configuration $\mathcal{G} =< g_0, ..., g_N >$, where N is the number of DSAs, and the initial position of SEED and DSAs, our method can generate the optimal trajectories for all DSAs.

2) Partitioning the environment: The workspace is a continuous space. In order to use a logical method based on an atomic proposition, we have to divide the space into discrete areas. Given a robot workspace $mathbbRW \subset R^2$, the partition divides the workspace into adjacent and disjoint cells. $D = \bigcup_{i=1}^{Dn} D_i$ and $D_i \cap D_j = \Phi$, where $i, j \in [1, Dn]$ and $i \neq j$. Therefore the partition is a mapping $\eta :\ mathbbRW \to \mathbb{D}$. The cells can be convex polygons. In this paper, we adopt a square partition, because the Sambot, and the docking side are all square. The workspace is divided into $Dn = LX \circ HY$ cells. LX and HY are constants that denote the horizontal and vertical length of $\mathbb{D}$. The size of each square is fixed as $200mm * 200mm$, so as to make enough room for one Sambot.

$$\mathbb{D}\ltimes = \{(x, y) \in \mathbb{Z}^2 \mid x = [\frac{m}{200}], y = [\frac{n}{200}], (m, n) \in \mathbb{RW}\},$$

and [ ] denotes the Rounding Function. As a result, the continuous working space is mapped into a discrete two-dimensional space, with the size of $LX \circ HY$.

A group of robots are moving through the cells. The boolean proposition $0_1,\ 0_2, ..., 0_{Dn}$ is introduced to indicate whether or not the cell is occupied by a robot. One position may not be occupied by two robots at the same time.

$$0_i = \begin{cases} 1 & \text{if} \quad R \in D_i \\ 0 & \text{if} \quad R \notin D_i \end{cases}$$

3) Configuration: For each Sambot, five docking interfaces are available, including one active interface and four passive ones in its front, left, back, and right side. In order

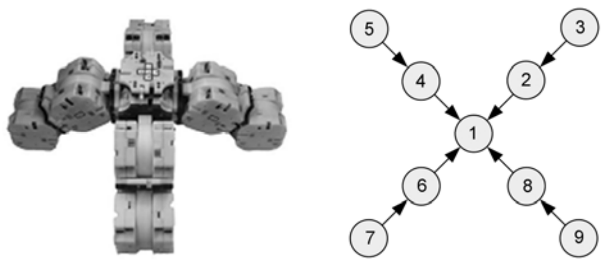

Fig. 4. The cross configuration

to successfully accomplish the process of self-assembly, it is necessary to have a proper method of describing the configuration. In [2], integers of -1, 1, 0 are adopted to represent the three kinds of connection states in which a docking interface may stay: active connection, passive connection, and no connection. The connection state of a Sambot is described by a quaternion $g = (a_1, a_2, a_3, a_4)$, where $a_i \in \{1, -1, 0\}$ and represent the connection state of its front, left, back and right interfaces. Thus, with the Seed as the first quaternion, all of the N DSAs of robots consist of a two-dimensional array named a Configuration Connection State Table, $\mathcal{G}[N+1][4]$, which describes the configuration of the Sambots. For example, the configuration of Sambot from number 1 to number 9 in fig.4 will be

$$\mathcal{G} = ((-1 - 1 - 1 - 1), (1\ 0 - 1\ 0), (1\ 0\ 0\ 0), (1\ 0 - 1\ 0), \\ (1\ 0\ 0\ 0), (1\ 0 - 1\ 0), (1\ 0\ 0\ 0), (1\ 0 - 1\ 0), (1\ 0\ 0\ 0))$$

With a target configuration and the position of the Seed, we can figure out the destinations of the DSAs.

B. Sambot Model

Given a team of robots R_i, $i = 0, 1, ..., N-1$, $N \in I^+$ is the number of robots. They are separated into two types: SEED and DSA. The SEED is responsible for starting the self-assembly process while the latter docks actively. We mark SEED as R_0, while the DSAs range from R_1 to R_{N-1}. Each robot has its own location, expressed as $(xx, yy) \in \mathbb{D}$.

In the following we establish the timed automata models for SEED and DSAs in *Uppaal*. In order to reduce the communication signal and avoid state space explosion, we compress the model of the robot and controller into one model. Therefore, the robot model contains the control command and strategy.

1) Seed model: The central Sambot, sponsors the self-assembly process. The timed automaton of the *SEED* is presented in Fig. 5. There are two states for the *SEED*: $Init$ state and $Wait$ state, where the *SEED* is waiting for self-assembly to be accomplished. In the $Init$ state, it starts the process via a broadcast signal $assemble!$, which will be responded to by all of the DSAs via $ans[j]?$. The integer ii is defined to count the number of responses. According to the expected configuration $\mathcal{G}$ and its own position, the goal locations will be calculated via the defined function $Assemble()$ and be distributed randomly over all of the DSAs, through the function $setGoal()$. Thus, the *SEED* enters the location $Wait$.

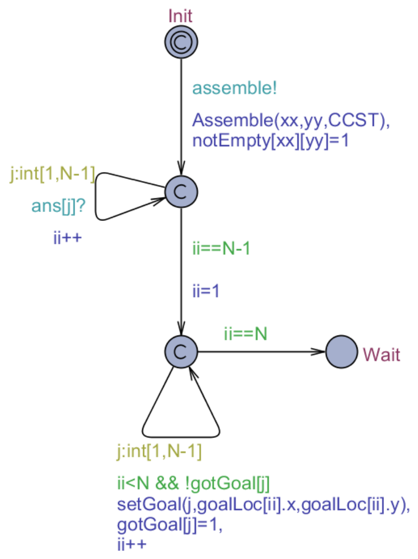

Fig. 5. The formal model of seed

2) DSA model: DSAs are restricted to moving horizontally and vertically in a Cartesian grid. In other words, a DSA can freely choose to move in the direction of east, west, south, and north. If the robot crosses one square, this is called one step. The time taken for that one step is denoted by a constant $t\_m = 1$.

Fig.6 shows the timed automaton of DSA. $clock\ \ t$ is defined in the local declaration of the DSA template. We also declare the process template with the parameters: $int[1, N]\ i, int[0, LX]\ xx, int[0, HY]\ yy$, where i denotes the ID number of the *DSA* and (xx, yy) are the coordinates of the robot in the workspace $\mathbb{RW}$. There are four locations in this model $L = \{Init,\ Ready,\ Moving,\ Done\}$, where $Init$ is the initial location.

Each *DSA* starts to move after receiving the signal $assemble!$ from the *SEED* model and gives a response along with its position uploaded. When *DSA* enters the location *Moving*, the *DSA* can move around freely and randomly as long as the cell is not occupied. Four self-loop transition edges in the location $Moving$ describe the four possible moving directions. Each edge is labeled with a common condition $(t >= t\_m, Loc[i]! = myGoalLoc[i])$, which indicates that *DSA* does not reach the destination. The distinct condition guarantees that the movement is legal. For example, the guard$(Loc[i].x < LX - 1, !notEmpty[Loc[i].x + 1][Loc[i].y])$ ensures the *DSA* can move east. $Loc[i]$, a defined structure that includes members of $int\ x, int\ y$, denotes the current position of the i^{th} *DSA*, while $myGoalLoc[i]$ is the target. When the transition takes place, the function of $moveEast()$ shall

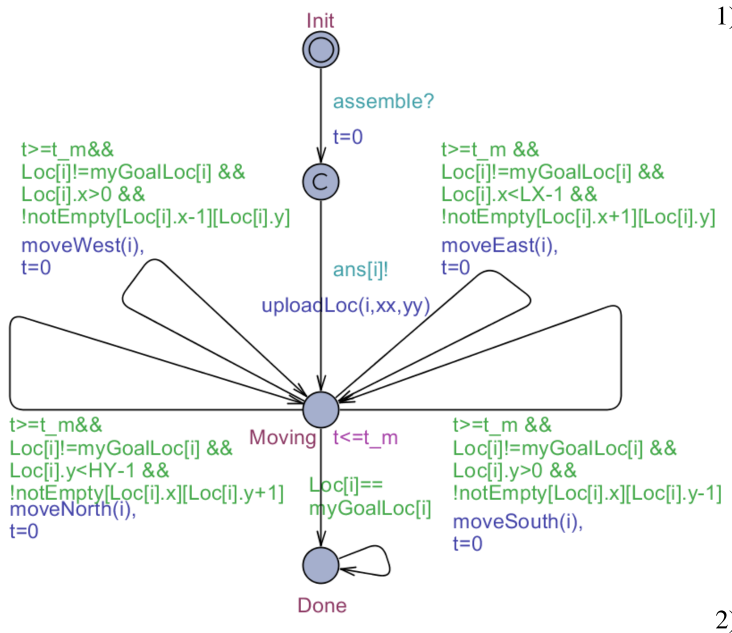

Fig. 6. The formal model of DSA

be invoked to update the current coordinates. As soon as the *DSA* approaches the goal position, the model enters the location *Done*, where the task of docking is accomplished.

V. Model Checking and Path Planning

The property specification used in $Uppaal$ is a simplified version of CTL. The path quantifiers are the same as those in CTL, but the temporal operators are just $F(<>)$ and $G([])$. The proposition logic has the operator $negation$ $(not,\ \neg)$, $conjuction$ $(and,\ \&)$, $disjunction$ $(or,\ ||)$, and $implication$ $(imply,\ \rightarrow)$. The reachability property, safety property, and liveness property can be checked.

Moreover, a property of the form $inf\ \{expression\} : list$ is added in the latest version of $Uppaal$, which returns the infima of the expression in the list, whereas $sup\ \{expression\} :\ list$ evaluates the suprema.

A. Verification of Properties

The following properties are specified based on the system requirements in section II. They are verified by $Uppaal$, and run on a computer with a Quad 2.66 GHz CPU with 2.0 GB of memory.

The experimental work-space is $2000mm \circ 2000mm$. The workspace is divided into $10 \circ\ 10$ grid, so, the grid size is $200mm * 200mm$. The whole system contains one seed and four DSAs. The initial position is shown in Fig.7. In this experiment the goal configuration is a cross. Therefore, the destinations of DSA are $(6,4)$, $(5,3)$, $(6,2)$, and $(7,3)$ by the translation algorithm.

1) *Configurable property*:All DSAs can arrive at the destination, which means completing the configuration with the shortest path.

$$E <>\ DSA1.Done\ and\ DSA2.Done\ and\ DSA3.Done\ and\ DSA4.Done$$

This property is satisfied. We do not assign the destination to DSAs. The best distribution plan will be found by completing the search algorithm for model checking. From the shortest trace provided by $Uppaal$, we can get the shortest path and their destination allocation for all DSAs. The trace of DSA4 is as follows:

$$(Init)\xrightarrow{assemble?}(Ready)\xrightarrow{ans[4]!}(Moving)\xrightarrow{\tau,mEast()}(Moving)\xrightarrow{\tau,mEast()}(Moving)\xrightarrow{\tau,mEast()}(Moving)\xrightarrow{\tau,moveSouth()}(Moving)\xrightarrow{\tau,moveSouth()}(Moving)\xrightarrow{\tau,moveSouth()}(Done)$$

2) *Collision avoidance property*: For the planning trajectories, DSAs will never crash into each other.

$$A[\ ]\ not((Loc[1] == Loc[2]\ or\ Loc[2] == Loc[3]\ or\ Loc[3] == Loc[4]\ or\ Loc[1] == Loc[3]\ or\ Loc[1] == Loc[4]\ or\ Loc[4] == Loc[2])\ and\ t > 0)$$

This property is satisfied because before the DSA moves, the DSA first asks if the approaching cell is occupied. Only if the cell is empty will the DSA move ahead.

3) *Execution timed property*:What is the moving time for finishing the following configuration?

$$inf\ \{\ DSA1.Done\ and\ DSA2.Done\ and\ DSA3.Done\ and\ DSA4.Done\ \} :\ t$$

The result is seven time units. This means that the target configuration will finish within six time units, which is the short possible time.

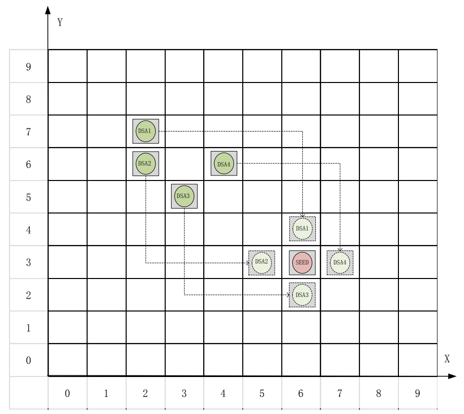

Fig. 7. Path given by Uppaal

B. Planning Trajectory Traces

As illustrated in Fig.7, a trajectory trace is generated by $Uppaal$, which satisfies both the above mentioned CTL formula of *Configurable property* and *Execution timed property*. The two properties demonstrate that all of the DSAs have finally arrived at the destination and that the shortest time has turned out to be seven time units, respectively. We can easily learn that one of the best assignment plans is the one in which DSA1 achieves the goal location (6, 4), DSA2 the location (5, 3), DSA3 the location (6 ,2), and DSA4 the location(7, 3). With this allocation, the overall time taken will be the least. This is just a high-level discrete planning.

Guided by high-level planning and considering the physical constraints of robots, the trajectory is obtained by a sampling-based algorithm. With the dynamics equation of robots, the Rapidly-Exploring Random Tree (RRT) algorithm [22] generates a tree of possible traces $T = (V, E)$. The initial position v_{init} is the root of the tree. After a time step Δt, the next vertex v_{new} is computed by the dynamics equation. Then, we see whether v_{new} is in the space of high level discrete traces. This step will be repeated until v_{new} is legal. In Fig.8 the random computed vertexes v_{new1}, and v_{new2} are outside the discrete plans generated in high level. So they are abandoned. Vertex v_{new3} is legal and selected as the next vertex. The computed trajectory follows the dynamics features of robots and also satisfies the verified properties.

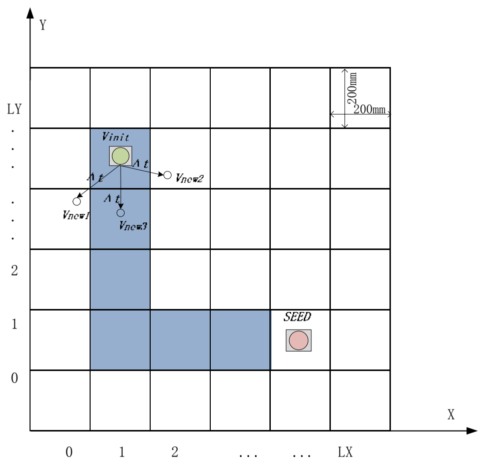

Fig. 8. The exploration step in sample-based algorithm

VI. Conclusion

In this paper, a multi-layer method was presented for the motion planning of a self-assembly module system. We discussed the creation of timed automata model from any possible configuration and get the optimal allocation by model checking. An example illustrated the effectiveness of our approach. This method can find the optional plan within short time, compared with the random and aimless wandering of DSAs in our previous work. We will establish the experimental platform and test our method in the future.

References

[1] H. X. Wei, and Y. Chen, and J.Tan and T. Wang. “Sambot: A self-assembly modular robot system”, *IEEE/ASME Transactions Mechatronics*, vol.14, no.4, pp745-757, 2011

[2] H. X. Wei,H. Li, J. Tan, and T. Wang. “Self-assembly control and experiments in swarm modular robots”, *Science China*, vol.55, no.4, pp1118-1131, 2012

[3] C. Anderson, G. Theraulaz, J. L. Deneubourg. “Self-assemblages in insect societies”. *Insectes Soc*, 2002, 49(2): 99C110

[4] R. Grob, M. Dorigo. “Self-Assembly at the macroscopic scale”. Proceedings of the IEEE, 2008, 96(9): 1490C1508

[5] E. M. Clarke, O. Grumberg, D Peled. “Model Checking”, The MIT Press, 1999

[6] C. V. Jones , M. J. Mataric. “From local to global behavior in intelligent self-assembly”. *In: Proceedings of IEEE International Conference on Robotics and Automation*, Taibei, 2003. 721C726

[7] J. Kelly, H.Zhang. “Combinatorial optimization of sensing for rule-based planar distributed assembly”. *In: Proceedings of IEEE/RSJ International Conference on Intelligence Robots and Systems*, Beijing, 2006. 3728C3734

[8] R. OGrady, A. L. Christensen, M. Dorigo. “SWARMORPH: multirobot morphogenesis using directional self-assembly”. *IEEE Trans Robot*, 2009, 25(3): 738C743

[9] S. Castro, S. Koehler, H. K. Gazit. “High-level control of modular robots”. *In proceeding of: 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems* , IROS 2011, San Francisco, CA, USA, September 25-30, 2011, 3120 - 3125

[10] A. Bhatia, L. E. Kavraki, M. Y. Vardi. “Sampling-based motion planning with temporal goals”. *IEEE International Conference on Robotics and Automation (ICRA)* , 3-7 May, 2010, 2689 - 2696

[11] R. Alami, S. Fleury, M. Herrb, F. Ingrand, F. Robert. “Multi-robot cooperation in the MARTHA project”, *IEEE Robotics & Automation Magazine*, vol.5, pp.36-47, 1998

[12] R. Alur, J. Esposito, M. Kim, V. Kumar, and I. Lee. “Formal modeling and analysis of hybrid systems: A case study in multi-robot coordination”, *Formal Methods*, pp212-323, 1999

[13] V. A. Ziparo, L. Iocchi, D. Nardi, P. F. Palamara, and H. Costelha. “Petri net plans: a formal model for representation and execution of multi-robot plans”, *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*Vol 1, pp79-86, 2008

[14] M. M. Quottrup and T. Bak, and R. I. Zamanabadi. “Multi-robot planning: A timed automata approach”, *IEEE International Conference on Robotics and Automation*, pp4417-4422, 2004.

[15] E. Emerson, ”temporal ans modal logic,” *in Handbook of Theoretical Computer Science*, J.van Leeuwen, Ed. MIT press, 1990, vol.B ch. 16, pp.995-1072

[16] G. Behrmann, A. David, and Kim G. Larsen. “A tutorial on Uppaal”. *In 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems*, 2004, LNCS 3185

[17] http://www.uppaal.com

[18] G. Behrmann, J. Bengtsson, A. David, K. G. Larsen, P. Pettersson, and W. Yi. “Uppaal Implementation Secrets”. *In Proceedings of the 7th International Symposium on Formal Techniques in Real-Time and Fault Tolerant Systems*, 2002

[19] J. Bengtsson and Y. Wang. “Timed Automata: Semantics, Algorithms and Tools”. *In Lecture Notes on Concurrency and Petri Nets*. LNCS 3098, Springer-Verlag, 2004

[20] R. Alur and D. L. Dill. “A theory of timed automata”. *In Theoret. Comput. Sci.*, Vol. 126, No.2, 1994, pp. 183-235

[21] R. Alur. “Timed automata”. *In Proceedings of Computer Aided Verification*, Trento, IT, LNCS 1633, Springer-Verlag, Jul. 1999, pp. 8-22

[22] S. M. LaValle. “Rapidly-exploring random trees: A new tool for path planning”. TR 98-11, Computer Science Dept., Iowa State University, 1998.