2D Geometric Modeling and Verification of Line Tracing Robot Using UPPAAL Model Checker

Yuta NAKATANI
Department of Computer Science
Tokyo Institute of Technology
Tokyo, Japan
e-mail: nakatani@lambda.cs.titech.ac.jp

Shin-ya NISHIZAKI
Department of Computer Science
Tokyo Institute of Technology
Tokyo, Japan
e-mail: nisizaki@cs.titech.ac.jp

***Abstract*— Model checking is one of the formal methods for verification of hardware and software systems. A model checker verifies queries described in temporal logic formulas about a model defined as a state transition diagram. Since the model checkers make an exhaustive search of the state space, the key point is how to reduce the state space in order to avoid an explosion of the number of states. The UPPAAL model checker is a model checker based on Timed CTL which is suitable for handling real-time systems. A line tracing robot, a typical example of real-time embedded systems, is a small electric car with motors and photo-sensors that follows a line on the ground. In this paper, we study the methodology of modeling and verification of the line tracing robot. In particular, we focus on two-dimensional geometric modeling which is acceptable for model checking without leading to state explosion.**

Keywords-component; verification; model checking; line tracing robot; 2-dimensional geometric modeling

I. INTRODUCTION

Model checking is a verification method of software and hardware systems: if you give the specifications of the target system and a query to be confirmed, then the model checker examines all the possible states exhaustively and returns its validity. Various kinds of model checkers have been developed, for example, the SPIN Model Checker [1] and state-transition diagrams and queries logical formulas in temporal logic. In the SPIN model checker, models are written in the model description language Promela and queries in Linear Temporal Logic. The Promela codes are translated into a state transition diagram represented as C programming language codes. On the other hand, in UPPAAL, models are depicted as state transition diagrams using a graphical user interface and queries are described in Timed Computational Tree Logic. The state transition diagrams are extended by adding real-time constraints, which correspond to timed automata [3]. UPPAAL is utilized in order to verify real-time systems [4,5,6]. We have studied formalization of the practical system using model checkers [7,8].

In this paper, we expand the application of model checking to practical domains. We focus on control of a line tracing robot as such a practical application. In comparison with the other cases of model checking, there is a difficulty in how to model two-dimensional space where the robot is driven. 2D geometric modeling has been studied from various viewpoints. Spatial logic [9] is a typical example of a formal approach to 2D geometric modeling. We study 2D geometric modeling for model checking, in which we control number of states and state explosion in model checking.

II. MODELING OF LINE TRACING ROBOT

In this section, we introduce a line tracing robot and its controlling algorithm. We then show a 2D geometric model which is formalized discretely for model checking.

A. *Line Tracing Robot*

We consider the *Pololu 3pi Robot*[10] (Fig. 1) for research on model checking and 2D geometric modeling. It has two micro gearmotors, five photo sensors, a character LCD, three user push-buttons, and an ATmega328 microcontroller.

Figure 1. Pololu 3pi Robot

For simplicity's sake, we assume that the line tracing robot has two independent wheels with micro gear motors, three photo sensors, and one push-button.(Fig. 2)

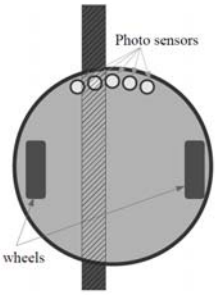


Figure 2. Simplified 3pi

If the left photo sensor is active and the right inactive, the left gear motor spins and the robot turns right (Fig. 3). If the right photo sensor is active and the left inactive, the right

gear motor spins and the robot turns left (Fig. 5). If both photo sensors are either active or inactive, both the gear motors spin and the robot goes straight ahead (Fig. 4).

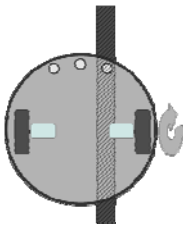


Figure 3. Right Turn

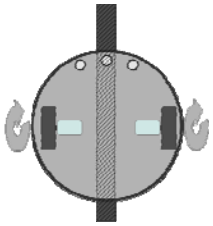


Figure 4. Going Straight Ahead

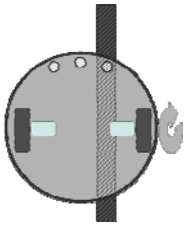


Figure 5. Left Turn

B. *Discrete 2D Modeling of Playground*

A playground of the line tracing robot is traditionally formalized as a 2-dimensional real coordinate space, $\mathbb{R}2$. In our study, we apply the model of the robot and the playground to the UPPAAL model checker. In order to reduce the size of the search space in model checking, we have to discretize it appropriately. Actually, we use a 10×10-grid as a model of the playground (Fig. 6).

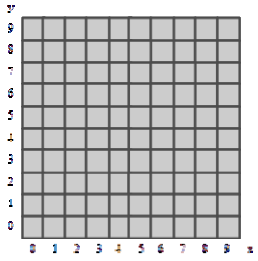


Figure 6. Grid Model of Playground

We consider a circle-shaped track (Fig. 7) as an example. We represent a playground as an array of length 10× 10 and a track as values on the array. The example of Fig. 7 is described as the following code for the model checker (Fig. 8). The array track in this figure stores a track line that the robot should trace. A cell on the track line in the grid is represented as a component of the array whose value is 1. The other cells correspond to the components of value 0.

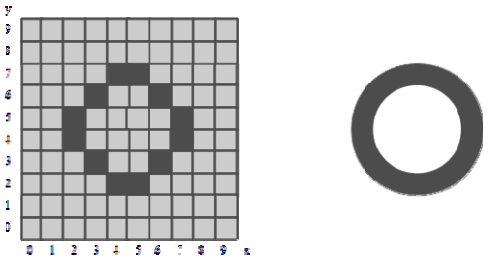


Figure 7. Grid Model of Circle-shaped Track

```
const int track[10*10] = {
      0,0,0,0,0,0,0,0,0,0,
      0,0,0,0,0,0,0,0,0,0,
      0,0,0,0,1,1,0,0,0,0,
      0,0,0,1,0,0,1,0,0,0,
      0,0,1,0,0,0,0,1,0,0,
      0,0,1,0,0,0,0,1,0,0,
      0,0,0,1,0,0,1,0,0,0,
      0,0,0,0,1,1,0,0,0,0,
      0,0,0,0,0,0,0,0,0,0,
      0,0,0,0,0,0,0,0,0,0
};
```

Figure 8. Representation of Circle-shaped Track

C. C. *Control of the Discrete 2D Model*

The real line tracing robot, 3pi, can move in various directions, turning rotation of each tire. Also, in order to reduce the number of states of our model, we simplify the movements of the line tracing robot: we suppose that the robot can move in only three directions: going straight ahead, right-turning, and left-turning (Fig. 10).

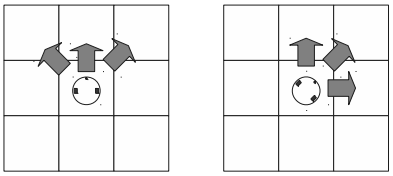


Figure 9. Movement of Line Tracing Robot

Moreover, we suppose that the robot faces only in eight directions: north, northeast, east, southeast, south, southwest, west, and northwest. These directions are represented as non-negative integers 0,1,2,...,7, respectively (Fig. 10).

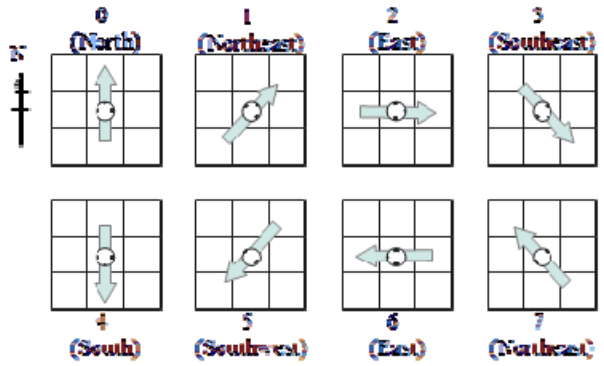


Figure 10. Eight Directions of Line Tracing Robot

If we let the current position be (x,y), then the next position is defined as Table 1. For each current direction, the next direction is defined as Table 1.

TABLE 1. NEXT POSITION FOR (X, Y)

	North	East	South	West
Straight-ahead	(x, y+1)	(x+1, y)	(x, y-1)	(x-1, y)
Right-turning	(x+1, y+1)	(x+1, y-1)	(x-1, y-1)	(x-1, y+1)
Left-turning	(x-1, y+1)	(x+1, y+1)	(x+1, y-1)	(x-1, y-1)

	Northeast	Southeast	Southwest	Northwest
Straight-ahead	(x+1, y+1)	(x+1, y-1)	(x-1, y-1)	(x-1, y+1)
Right-turning	(x+1, y)	(x, y-1)	(x-1, y)	(x, y+1)
Left-turning	(x, y+1)	(x+1, y)	(x, y-1)	(x-1, y)

For example, let the current direction be north and the current position (x, y). According to the definition in Table 1, there are three possibilities (x, y+1), (x+1, y+1), and (x-1, y+1) of the next position, and north, northeast, and northwest are the possibilities for the next direction, with three options of movement: going straight ahead, right-turning, and left-turning (See Fig. 11).

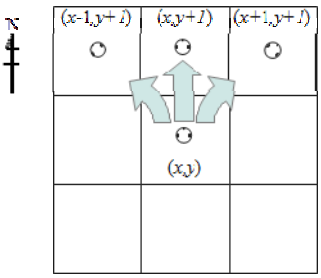


Figure 11. Next Position and Direction (North-facing Case)

III. VERIFICATION OF MOTION OF LINE TRACING ROBOT

There are various properties of the line tracing robot to be verified. In this paper, we focus on the property of its correct motion. The purpose of the line tracing robots is to keep moving along a given line. The action ``moving along a given line" has an ambiguity and we have to describe it areas a formal specification rigidly. Specification of the correctness consists of two notions, an *area restriction* and a *direction constraint.*

A. *Area Restriction*

First, we consider formalization of the area restriction. We regard the following two properties as equivalent:

- a robot traces a given line;
- the robot does not enter a restricted area defined as the outer side of the line.

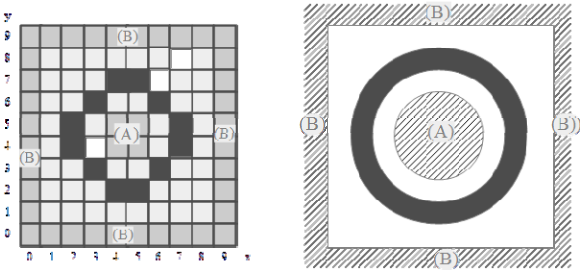


Figure 12. Restricted Area

We consider formalization of the restricted area in the example of Fig. 7. We show the restricted area as the hatched area in the right figure in Fig. 12. It is represented as a discrete model in the left figure. The sub-area (A) is expressed as the following inequalities:

$4 \le x \le 5$ and $4 \le y \le 5$.

Accordingly, the condition on the restricted area is formulated as an LTL formula:

```
A [ ] not
     (4 <= pos_x && pos_x <= 5 &&
      4 <= post_y && pos_y <= 5)
```

We formulate the subarea (B) as disjunction of the four inequalities:

$x \le 0$, $9 \le x$, $y \le 0$, $9 \le y$,

and therefore, we formalize it as an LTL formula

```
A [ ] not
     (pos_x <= 0 && 9 <= pos_x &&
      post_y <= 0 && 9 <= pos_y)
```

B. *Direction Constraint*

We next consider formalization of the direction constraint. We suppose a direction for each given track. For example, we suppose the direction shown in Fig. 13 for the circle-shaped track.

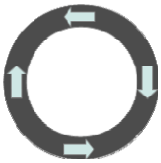


Figure 13. Supposed Direction for Circle Shaped Track

The actual direction of the robot is not fixed: in order to follow the track line the robot takes various directions which are not fixed uniquely. Therefore, we define the direction constraint as a condition that the robot does not go against the given direction.

For example, suppose that the robot should move southward. The robot takes various directions except northward, in order to follow the track line (Fig. 14).

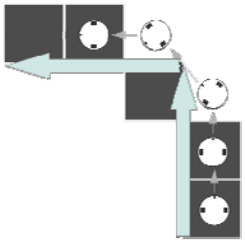


Figure 14. Possible Direction to Follow Track Line

Next, we consider part of the eight-figure track which are supposed for the robot to go northward (Fig. 15).

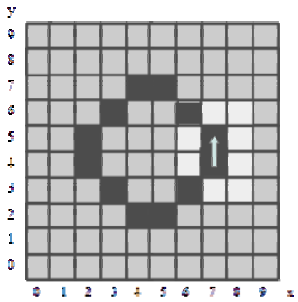


Figure 15. Track of Northward Direction

The area where the robot is supposed to go southward is formulated as follows.

```
A[]((6<=pos_x && pos_x<=8 &&
     3<=pos_y && pos_y<=6)
     imply (angle != 4))
```

The formula (angle != 4) means that the robot does not go southward.

Similarly, the direction constraints for the areas where the robot is supposed to go eastward, northward, and westward, respectively are formulated as follows.

```
A[]((1<=pos_x && pos_x<=3 &&
     3<=pos_y && pos_y<=6)
     imply (angle != 0))

A[]((3<=pos_x && pos_x<=6 &&
     1<=pos_y && pos_y<=3)
     imply (angle != 6))

A[]((3<=pos_x && pos_x<=6 &&
     6<=pos_y && pos_y<=8)
     imply (angle != 2))
```

The formulas (angle!=0), (angle!=6), and (angle!=2) mean that the robot does not go northward, westward, or eastward, respectively.

C. *Result of Model Checking*

Both the two LTL formulas of the area restriction for the circle shaped track

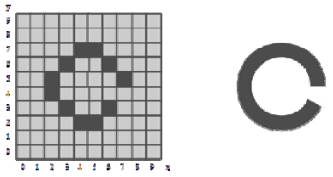
```
A[] not (pos_x<=0||9<=pos_x || pos_y<=0 || 9<=pos_y)     (1)

A[] not (4<=pos_x&&pos_x<=5&&4<=pos_y&& pos_y<=5)    (2)
```

are verified by UPPAAL model checker and the direction constraints

```
A[]((6<=pos_x && pos_x<=8 && 3<=pos_y && pos_y<=6)
                                    imply (angle !=4)    (3)

A[]((1<=pos_x && pos_x<=3 && 3<=pos_y && pos_y<=6)
                                    imply (angle !=0)    (4)

A[]((3<=pos_x && pos_x<=6 && 1<=pos_y && pos_y<=3)
                                    imply (angle !=6)    (5)

A[]((3<=pos_x && pos_x<=6 && 6<=pos_y && pos_y<=8)
                                    imply (angle !=2)    (6)
```

are all verified similarly by UPPAAL, too.

Secondly, we give an unexpected track to the model checker with the controller model and the queries which were used in the previous case. The track that we check next is a circle-shaped track with chipping, such as that shown in Fig. 16.

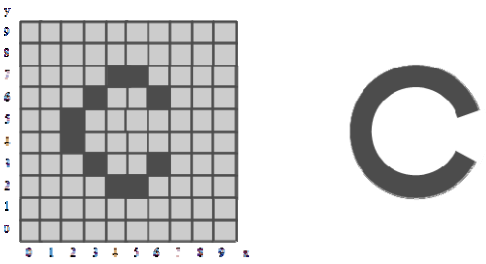


```
const int track[10*10] =
{
    0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,1,1,0,0,0,0,
    0,0,0,1,0,0,1,0,0,0,
    0,0,1,0,0,0,0,1,0,0,
    0,0,1,0,0,0,0,0,0,0,
    0,0,0,1,0,0,1,0,0,0,
    0,0,0,0,1,1,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0
};
```

Figure 16. C-shaped Track

Actually, in this case, the model checker verifies that the conditions of the area restriction (1) and (2), and those of the direction constraints (3),…,(6) are all satisfied.

Thirdly, we widen the gap in the track as in Fig. 17.



```
const int track[10*10] =
{
    0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,1,1,0,0,0,0,
    0,0,0,1,0,0,1,0,0,0,
    0,0,1,0,0,0,0,0,0,0,
    0,0,1,0,0,0,0,0,0,0,
    0,0,0,1,0,0,1,0,0,0,
    0,0,0,0,1,1,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0
};
```

Figure 17. C-shaped Track with Wider Gap

In this case, the direction constraints (3),…,(6) and the area restriction (2) are all satisfied. However, the area restriction (1) is *denied* by model checking. Simulation execution of a counterexample to the above is shown in Fig. 18.

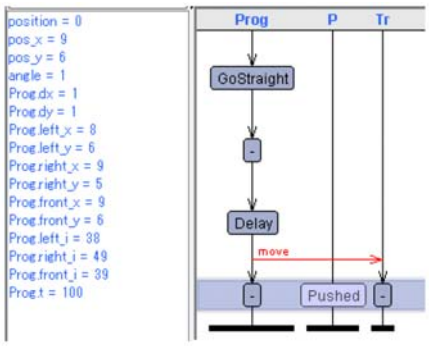


Figure 18. Simulation Execution of Counterexample

Figure 18 shows that the model checker arrives at the status that pos_x=9, pos_y=6 and angle=1, which means that the line tracing robot reaches the location (9,6) and then turns east, that is, the robot runs off the track.

The model checking shows the following properties.

- If the circle-shaped track has a narrow gap (i.e. a one-block gap), then the line tracing robot can follow the track without being affected by the gap.
- If the circle-shaped track has a wider gap (i.e. 2 blocks gap), then the line tracing robot runs off the track.

We know that UPPAAL model checker handles the model formalized here and can show positive and negative cases.

IV. Conclusion and Future Works

In this paper, we describe a model of the controller and tracks of a line tracing robot in order to verify it using the UPPAAL model checker. Though it is often difficult to process models using a model checker in a practical time, we showed that the model checker could verify our model successfully. We apply model checking to both positive and negative examples.

In this paper, we formalize the tracks as a 2-dimensional discrete plane, specifically, an array of length 10× 10. By simplifying the plane, we successfully carried out verification with the UPPAAL model checker. Although the simplified model is intuitively correct, it is not justified mathematically. Moreover, it is unknown whether our modeling method is applicable to other complicated cases. We should conduct further study of 2D modeling and its simplification for model checking.

We have not compared simplified models of a 2D plain with the usual continuous ones. A promising research direction is the cooperating method of simulation and model checking [11].

Acknowledgment

One of the authors, Yuta Nakatani, studied this research at Department of Computer Science. Now he belongs to Department of Communications and Computer Engineering. This work was supported by Grants-in-Aid for Scientific Research (C) (24500009).

References

[1] Gerard Holzmann. The SPIN Model Checker: Primer and Reference Manual. Addison-Wesley Professional, 1997.

[2] Johan Bengtsson, Kim Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. UPPAAL -- a tool suite for automatic verification of real-time systems. In Proceedings of the 4th DIMACS Workshop on Verification and Control of Hybrid Systems, volume 1066 of Lecture Notes in Computer Science, pages 232—243.Springer Berlin Heidelberg, 1996.

[3] Rajeev Alur and David L. Dill. A theory of timed automata. Theoretical Computer Science, 126(2):183—235, 1994.

[4] Klaus Havelund, Arne Skow, Kim~Guldstrand Lasen, and Kristian Lund. Formal Modeling and Analysis of an Audio/Video Protocol: An Industrial Case Study Using UPPAAL.In Proceedings of the 18th IEEE Real-Time Systems Symposium, pages 2—13. IEEE Computer Society Press, 1997.

[5] Klaus Havelund, Kim Guldstrand Larsen, and Arne Skou. Formal Verification of a Power Controller Using the Real-Time Model Checker UPPAAL. In Formal Methods for Real-Time and Probabilistic Systems, volume 1601 of Lecture Notes in Computer Science, pages 277—298. Springer Berlin Heidelberg, 1999.

[6] Marius Mikucionis, Kim Guldstrand Lasen, Jacob Illum Rasmussen, Brian Nielsen, Arne Skou, Steen Ulrik Palm, Jan Storbank Pedersen, and Poul Hougaard. Schedulability analysis using Uppaal: Herschel-Planck case study. In Proceedings of the 4th International Conference on Leveraging Applications, ISoLA 2010, volume 6416 of Lecture Notes in Computer Science, pages 175—190. Springer Berlin Heidelberg, 2010.

[7] Hiroki Kumamoto, Takahisa Mizuno, Kensuke Narita, and Shin-ya Nishizaki. Applying model checking to destructive testing and analysis of software system. Journal of Software, 8(5):1254—1261, 2013.

[8] Shin-ya Nishizaki and Takuya Ohata. Real-time model checking for regulatory compliance. In Proceedings of Second International Conference AIM/CCPE 2012, volume 296 of Communications in Computer and Information Science, pages 70—77. Springer-Verlag Berlin Heidelberg, 2013.

[9] Marco Aiello, Ian Pratt-Hartmann, and Johan van Benthem, editors. Handbook of Spatial Logics. Springer, 2007.

[10] Pololu 3pi robot user's guide.

[11] Ritsuya Ikeda, Kensuke Narita, and Shin ya Nishizaki. Cooperation of model checking and network simulation for cost analysis of distributed systems. international Journal of Computers and Applications, 33(4):323--329, 2011.