Verification of a Control Program for a Line Tracing Robot using UPPAAL Considering General Aspects

Toshifusa Sekizawa†, Kozo Okano‡, Ayako Ogawa*/**, and Shinji Kusumoto‡

†Department of Computer Science, College of Engineering, Nihon University, Japan
‡Graduate School of Information Science and Technology, Osaka University, Japan
* Faculty of Informatics, Osaka Gakuin University, Japan
** Sougou System Service Corporation
†sekizawa@cs.ce.nihon-u.ac.jp ‡{okano, kusumoto}@ist.osaka-u.ac.jp

Abstract - The demand for embedded systems have increased in our society. Ensuring the safety properties of these systems has also become important. Model checking is a technique to ensure such systems. Our target is formal verification of hybrid systems which contain both continuous and discrete behaviors. For the goal, we have studied properties of a line tracing robot built using LEGO Mindstorms with a control program written in LeJOS. We have already presented verification of safety properties of a control program for the application using model checker UPPAAL. In the previous study, we were in a preliminary stage and set limitations. In this presentation, we extend our previous study. In general, a real course can be expressed in combinations of straight and arc courses. First, we verify properties of the same control program for arc courses. Next, in case of the line tracer can not keep track, we analyze turning angle using counter examples. Above-mentioned two approaches are necessary from the standpoint of design phase.

Keywords: Embedded Systems, Formal Verification, Timed Automaton

1 INTRODUCTION

The demand for embedded systems have increased in our society. In these circumstances, it is important to ensure safety properties of embedded systems. Formal methods are mathematical based techniques for verification and development. Model checking is one of formal methods and is widely used in order to ensure properties. Model checking techniques take model and logical formula as their input. Given a model that represents a system under consideration, model checking automatically determines whether or not the model satisfies a given property by exhaustively searching for the state space of the model.

There are various kinds of model checking techniques. Most model checking techniques are based on the finite state machine. For example, a conventional model checking is based on Kripke structure and only deals with discrete variables. However, some embedded systems require time properties in their specification. Several models have been proposed to deal with such real-time systems. One of such approaches is the timed automaton [1]. Timed automaton uses clock variables which range over real numbers. Therefore, timed automaton model can naturally represent the behavior of real-time systems. One of major verifier for timed automaton is UPPAAL [2] in which extended timed automata is used to construct models. UPPAAL can deal with bounded integer variables and guard expressions on transitions which allow expressions of constraints on variables.

Embedded systems sometimes consist of continuous and discrete dynamics. Such systems are called hybrid systems [3]. We are motivated to verify the behavior of embedded control systems. Especially if these embedded systems are considered to be hybrid systems. For the goal, we have presented verification of safety property of a control program for a line tracing robot using model checker UPPAAL. In the previous study, we were in a preliminary stage and set limitations. For example, we only considered a straight line as a course.

In this study, we extend our previous study. First, we verify that the same control program can trace an arc course. This is because a real course can be expressed in combination of straight and arc courses. Therefore, verifying the tracer for arc courses should be important to show applicability of model checking. Next, if the line tracer is not able to run along an arc course, we analyze turning angle using counter examples. Above-mentioned approaches will be useful to check performance properties in the design phase.

The roadmap of this paper is as follows. Sec. 2 outlines the foundations of our work and briefly describe our previous study. Sec. 3 show specification of a line tracer, and its implementation is described in Sec 4. Then, formal models used in verification are described in Sec. 5. Verification results are presented in Sec. 6, and Sec. 7 offers some discussion of these results. Finally, Sec. 8 provides a concluding summary and outline our future work.

2 PRELIMINARIES

In this section, we outline the background to our work and briefly show our previous study.

2.1 Model Checking

Model checking [4] is an automatic formal verification technique. Given a model that represents a system under consideration, and a logical formula that represents a property to be verified, model checking automatically determines whether or not the model satisfies a given property by exhaustively searching for the state space of the model. There are various kinds of model checking depending on expressive power

of model and logical formula. In this study, we use timed automata to express models and Computation Tree Logic (CTL) for formulas. We use a model checker UPPAAL which takes timed automata as models and CTL formulas as property.

2.1.1 Timed Automata

A timed automaton is an extension of the conventional automaton with clock variables and constraints for expressing real-time dynamics. These are widely used in the modeling and analysis of real-time systems.

Definition 1 (constraints) *We use the following constraints on clocks.*

1. *C represents a finite set of clocks.*
2. *Constraints $c(C)$ over clocks C are expressed as inequalities in the following BNF (Bacchus Naur Form).*

$$E ::= x \sim a \mid x - y \sim b \mid E_1 \wedge E_2,$$

where $x, y \in C, \sim \in \{\leq, \geq, <, >, =\}$, and $a, b \in \mathbb{R}_{\geq 0}$, in which $\mathbb{R}_{\geq 0}$, is a set of all non-negative real numbers.

Time constraints are used to mark edges and nodes of the timed automata and for describing the guards and invariants.

Definition 2 (timed automaton) *A timed automaton $\mathscr{A}$ is a 6-tuple (A, L, l_0, C, T, I), where*

- *A: a finite set of actions;*
- *L: a finite set of locations;*
- *$l_0 \in L$: an initial location;*
- *C: a finite set of clocks;*
- *$T \subseteq L \times c(C) \times A \times 2^C \times L$ is a set of transitions. The second and fourth items are called a guard and clock resets, respectively; and*
- *$I : L \to c(C)$ is a mapping from location to clock constraints, called a location invariant.*

A transition $t = (l_1, g, a, r, l_2) \in T$ is denoted by $l_1 \overset{a,g,r}{\to} l_2$.

A map $v : C \to \mathbb{R}_{\geq 0}$, is called a clock assignment (or clock valuation). We define $(v + d)(x) = v(x) + d$ for $d \in \mathbb{R}_{\geq 0}$ and some $x \in C$.

For guards, resets and location invariants, we introduce some notation for clock valuations. For each guard $g \in c(C)$, a function $g(v)$ stands for the valuation of the guard expression g with the clock valuation v. For each reset r, where $r \in 2^C$, we introduce a function denoted by $r(v)$, and let $r(v) = v[x \mapsto 0], x \in r$. For each location invariant I, we shall introduce a function denoted by $I(l)(v)$, which stands for the valuation of the location invariant $I(l)$ of location l with the clock valuation v.

The dynamics of a timed automaton may be expressed via a set of states and their evaluations. Changes from one state to a new state may be as a result of either the firing of an action or an elapsed time.

Definition 3 (state of timed automaton) *For a given timed automaton $\mathscr{A} = (A, L, l_0, C, T, I)$, let $S = L \times \mathbb{R}^C_{\geq 0}$ be the complete set of states of $\mathscr{A}$, where $\mathbb{R}^C_{\geq 0}$ is a complete set of clock evaluations on C.*

The initial state of $\mathscr{A}$ can be given as $(l_0, 0^C) \in S$. For a transition $l_1 \overset{a,g,r}{\to} l_2$, the following two transitions are semantically defined. The first one is called an action transition, while the latter one is called a delay transition.

$$\frac{l_1 \overset{a,g,r}{\to} l_2, g(v), I(l_2)(r(v))}{(l_1, v) \overset{a}{\Rightarrow} (l_2, r(v))}, \qquad \frac{\forall d' \leq d \quad I(l_1)(v + d')}{(l_1, v) \overset{d}{\Rightarrow} (l_1, v + d)}$$

The semantics of a timed automaton can be interpreted as a labeled transition system.

Definition 4 (semantics of a timed automaton) *For a timed automaton $\mathscr{A} = (A, L, l_0, C, T, I)$, an infinite transition system is defined according to the semantics of $\mathscr{A}$, where the model begins with the initial state. By $\mathscr{T}(\mathscr{A}) = (S, s_0, \overset{\alpha}{\Rightarrow})$, the semantic model of $\mathscr{A}$ is denoted, where $\alpha \in A \cup \mathbb{R}_{\geq 0}$.*

Definition 5 (run of a timed automaton) *For a timed automaton $\mathscr{A}$, a run σ is finite or infinite sequence of transitions of $\mathscr{T}(\mathscr{A})$.*

$\sigma = (l_0, \nu_0) \overset{\alpha_1}{\Rightarrow} (l_1, \nu_1) \overset{\alpha_2}{\Rightarrow} (l_2, \nu_2) \overset{\alpha_3}{\Rightarrow} \cdots$

2.1.2 Computation Tree Logic

In model checking, properties are written as logical formulas. Computation Tree Logic (CTL) [5] is a temporal logic suited to dealing with such formulas. Using CTL we are able to describe properties relating to behaviors of a program for a line tracer robot.

Let AP be a set of atomic propositions. The syntax of CTL is defined as follows:

$$\begin{aligned}\varphi ::= \quad & \bot \mid \top \mid p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \varphi \to \varphi \\ & \mid \mathtt{AX}\varphi \mid \mathtt{EX}\varphi \mid \mathtt{A}\Diamond\varphi \mid \mathtt{E}\Diamond\varphi \mid \mathtt{A}\Box\varphi \mid \mathtt{E}\Box\varphi \\ & \mid \mathtt{A}[\varphi_1 \ \mathtt{U} \ \varphi_2] \mid \mathtt{E}[\varphi_1 \ \mathtt{U} \ \varphi_2],\end{aligned}$$

where p is an atomic proposition in AP. The symbols $\bot$, $\top$, $\neg$, $\vee$, $\wedge$ and $\to$ have their usual meanings. The symbols X ("next"), $\Diamond$ ("eventually"), $\Box$ ("globally"), and U ("until") are temporal operators. The symbols A ("always") and E ("exists") are path quantifiers. Intuitively, temporal operators represent statements of a path, and path quantifiers represent statements on one or more paths which are branching forwards from a state. In a CTL formula, temporal operators are preceded by a path quantifier. Due to space limitation, we omit semantics. Please refer to Emerson [5] for details of the semantics of CTL.

For example, a safety property that "variable x is less than 10 for all paths" is written as a CTL formula $\mathtt{A}\Box(x < 10)$.

2.1.3 UPPAAL

UPPAAL [2], is a popular model checker for extended timed automata. It supports model checking for both conventional and timed automata. UPPAAL allows verification of expressions described in an extended version of CTL. Note that, a property to be verified is called a query in the field of verification of timed automaton. Given a model and a query,

UPPAAL checks whether or not the model satisfies the query. If the query does not hold, UPPAAL returns a counter example. A counter example is a run of the model, and presents sequence of locations that query does not hold. In addition, UPPAAL supports local and global integers and primitive operations on integers, such as addition, subtract and multiplication with constants. Such expressions are also allowed on the guards of transitions. System models can be created from multiple timed automata which are synchronized via a CCS (Common-Channel Signaling)-like synchronization mechanisms. An important point is that, with the exception of clocks, the extended timed automaton used in UPPAAL cannot deal with real valued variables. We, therefore, have to round real values to integer values when we model the target systems.

2.2 Results from a Previous Study

In this subsection, we briefly mention about our previous study [6], [7]. The question at the core of our research is formal verification of embedded systems as hybrid systems. For that goal, our first step is verifying time-related properties of a real embedded application using UPPAAL. We set our application to a line tracing robot constructed by LEGO Mindstorms [8] with a control program written in Java base language LeJOS [9].

We presented verification of safety properties of the program for line tracing robot in terms of design verification. In the verification, we constructed two models expressed by timed automata, one for the control program and one for the motion control depending to the course. To construct these models, it is required that behaviors have to be modeled in discrete steps except for time clock. Sampling and quantization techniques are applied for the purpose. We showed experimental results of verification and presented model checking has power to check behaviors. We considered time-delay in the verification. However, the study was in preliminary stage because we set some limitations, such as no disturbance and handling only straight lines. Even limitations were set, we think that our previous study showed applicability of model checking for verifying real embedded systems.

3 SPECIFICATION

The whole system of line tracer consists of two parts; courses and a line tracer. We describe specifications in this section.

3.1 Course

For a line tracer robot, a *course* is a black line painted on white ground. Assume that course width w is constant. In general, a real course can be expressed in combination of straights and arcs. We verified that our control program can trace a straight course in our previous study. Therefore, we consider arc courses in this study. An arc course is expressed by radius r and central angle α.

3.2 Line Tracer

A *line tracer* is a vehicle which traces a course. In this study, we fix a line tracer that consists of a body, two motors, and two color sensors. Figure 1 illustrates the relationships between constants and state variables. Table 1 summarizes state variables associated with the line tracer. Additionally, $\overrightarrow{los}$ and $\overrightarrow{ros}$ are vectors from the gravity center (x, y) to left sensor and right sensor, respectively.

Table 1: State Variables of a Line Tracer

variable	description
(x, y)	coordinate of the center
θ	direction
$lsensor$	sensed value of the left sensor
$rsensor$	sensed value of the right sensor
(x_{sl}, y_{sl})	coordinate of the left sensor
(x_{sr}, y_{sl})	coordinate of the right sensor
v_l	revolution speed of the left wheel
v_r	revolution speed of the right wheel
lw	half width of the tracer

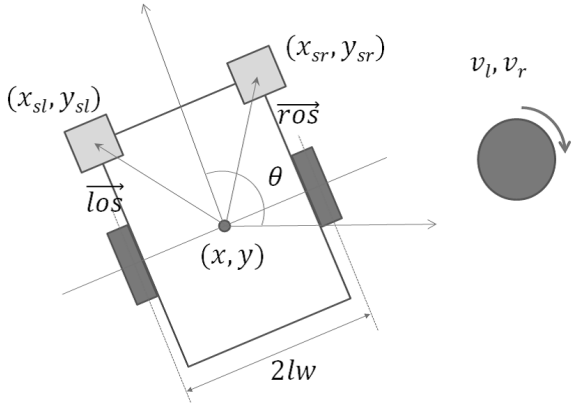


Figure 1: Constants and State Variables

A color sensor can discriminate colors. In this study, we assume that read value of the color sensor is two-valued, black and white, by setting threshold. Then the line tracer reads colors of the course using two color sensors, and determines its motion by changing left or right wheel speeds. Table 2 shows the controller logic associated with read values of two color sensors. If, for example, the left sensor and the right sensor sense white and black respectively, then the line tracer will "turn right". This is done by setting left wheel speed to high speed HS and right wheel speed to low speed LS. Note that there are delays in sensors and actuators, for example sleeping time before next sense-act loop and motor reaction.

Table 2: Logic for Color Sensors

		RightSensor	
		black	white
LeftSensor	black	go straight	turn left
	white	turn right	go straight

4 IMPLEMENTATION

LEGO Mindstorms NXT [8] is a kit for assembling robots and machines with various actuators and sensors. The default programming language for LEGO NXT is Mindstorms, but there are other languages such as NXC (Not eXactly C) [10] and LeJOS [9] which supply various classes for NXT sensors and actuators. We use LeJOS for making the control program of the tracer. This is because Mindstorms is GUI base language and does not suit for modeling. Instead, LeJOS is Java based language and is easier to construct models from a program.

Figure 2 shows our implemented controller program written in LeJOS. In this research, we use the same program used in our previous study mentioned in Sec. 2.2. Then, we try to verify that the program can trace arc courses.

5 MODEL

The line tracer system described in Sec. 3 is converted into two models; `Controller` model and `Motion` model. We introduce these models in this section,

Both `Controller` model and `Motion` model are expressed in timed automata. However, most of the state variables used in a line tracer have real values, and UPPAAL can only handle integer variables except for clock. Therefore, it is required to approximate state variables for discrete values to construct models in timed automata. We presented approximation of the state variables by applying sampling and quantization techniques in our previous study. In this study, we use the same models. Please refer papers [6], [7] for detail information of discretization techniques. Note that, we have modeled in the relative scale in this study. Therefore, units are not specified.

5.1 `Controller` Model

`Controller` model is a timed automaton which represents controller program for the line tracer. Fig. 3 shows `Controller` model which corresponds to the implementation in Sec. 4. Please refer to Table 1 which summarizes variables used in `Controller` model.

As described in Sec. 3, `Controller` decides motor speeds according to the four possible combinations of read values of the two color sensors. From the initial location represented as double circle, there are four transitions. Each of the transition corresponds to a pair of real value of sensors.

5.2 `Motion` Model

`Motion` model is a timed automaton which represents motions of the line tracer's coordinates of the gravity center and read values of color sensors. The line tracer keep on moving while the control program does not work because of delay or sleep time. Therefore, coordinates of center should be updated as independent of the `Controller` model to express behavior of the tracer. Fig. 4 shows the timed automaton which updates states variables at regular, discrete time intervals. The automaton of `Motion` model periodically calls functions updateX, updateY, updateTheta, up-

```
import lejos.nxt.*;
public class Controller {
  public static void main(String[] args)
        throws Exception {
    int rid,lid;
    final int HS = 420, LS = 120, BLACK = 7,
    MS = 360, HSEC = 500;
    Color colorR ,colorL;
    ColorSensor sensorR =
      new ColorSensor(SensorPort.S3);
      // 1(S3):right
    ColorSensor sensorL =
      new ColorSensor(SensorPort.S4);
      // 2(S4):left
    Motor motor = new Motor();
    motor.B.setSpeed(MS);
    motor.C.setSpeed(MS);
    Thread.sleep(HSEC);
      // wait for devices to be stable
    motor.B.forward();
    motor.C.forward();
    while(true) {
      rid = sensorR.getColorID();
      lid = sensorL.getColorID();
      if (lid == BLACK
         && rid != BLACK) {
        motor.C.setSpeed(LS);
        motor.B.setSpeed(HS);
      } else if (lid != BLACK
                && rid == BLACK) {
        motor.C.setSpeed(HS);
        motor.B.setSpeed(LS);
      } else if (lid == BLACK
                && rid == BLACK) {
        motor.C.setSpeed(HS);
        motor.B.setSpeed(HS);
      } else if (lid != BLACK
                && rid != BLACK) {
        motor.C.setSpeed(HS);
        motor.B.setSpeed(HS);
      }
      if (Button.readButtons()
          == Button.ENTER.getId())
        break;
    }
  }
}
```

Figure 2: Controller Program in LeJOS

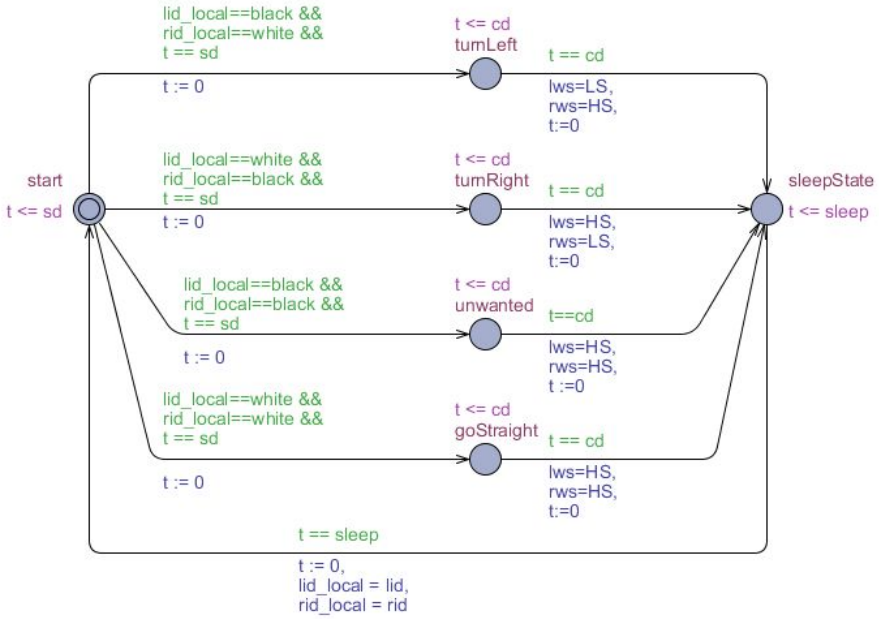


Figure 3: `Controller` Model

dateLSensor, and updateRSensor which update state variables x, y, θ, lid and rid, respectively. Note that, lid and rid are two-valued variables associated with read values of the sensors.

Read values of sensors depend on the coordinates of gravity center, the angle of the line tracer, and the course. Gravity center and angle are expressed by integer variables in this model. It is also required discretely handling of the course. There will be two methods for handling. First one is quantization, mapping the continuous course to discrete values. Second one is equation representation, the course is expressed in a formula. In this study, we adopt the second method. Let (s_x, s_y) be the coordinates of left or right sensor. Then, the read value of the sensor is decided to be black if (s_x, s_y) satisfies the following formula. Otherwise, the read value is decided to be white.

$$\left(r - \frac{w}{2}\right)^2 \leq s_x^2 + s_y^2 \wedge s_x^2 + s_y^2 \leq \left(r + \frac{w}{2}\right)^2$$

where r is radius of a circle course, and w is line width.

6 EXPERIMENTAL RESULTS

described in Sec. 5. In this section, we verify correctness of the control program. Verifications were performed using UPPAAL 4.0.13 running on Windows 7 (64 bit), Intel Core i5-2400, 3.10GHz, with 8GB memory.

6.1 Verification of Specification

A line tracer is expected to trace a course. First, we verify whether or not the controller program satisfies the property. Therefore, what we need to verify is, i) the tracer runs along the course within a certain range, and ii) the tracer keeps on taking its route, i.e., does not get stuck.

To verify these requirements, we need to fix some initial values. Let initial values be as follows.

- coordinates of gravity center $(x, y) = (r, 0)$
- direction of the tracer $\theta = 90°$
- width of the course $w = 100$

In addition, we set the following values associated with the tracer.

- half width of the tracer $lw = 60$
- distance between center and a sensor $ds = |\overrightarrow{los}| = |\overrightarrow{ros}| = 180$
  The angle between $\overrightarrow{los}$ and $\overrightarrow{ros}$ is 60 degrees.
- high / low wheel speeds $HS = 12, LS = 6$
- sensing interval is 1, and sensing delay is 1

Note that, sensing interval and sensing delay are modeled as an unit time of UPPAAL. It should be also noted that the parameters used in verification are not the same as those used in implementation.

We then check the correctness of the line tracer by verifying the following queries.

1. $\mathtt{A}\Box$ (*firstquadrant* $\rightarrow$ *inrange*)
   where *firstquadrant* is $x \geq 0 \wedge y \geq 0$, *inrange* is $\left(r - \frac{w}{2} - ds\right)^2 \leq x^2 + y^2 \wedge x^2 + y^2 \leq \left(r + \frac{w}{2} + ds\right)^2$, and ds is a distance between center and a sensor, *i.e.,* $ds = |\overrightarrow{ros}| = |\overrightarrow{los}|$.

2. $\mathtt{E}\Diamond\, (x < 0 \wedge y > 0)$

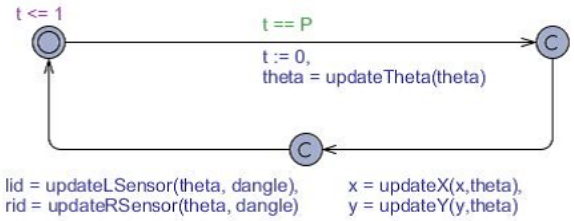


Figure 4: `Motion` Model

Query 1 represents that the gravity center of the tracer is always located within a certain range, $w/2 + ds$, from the line in the first quadrant. Note that we consider the gravity center (x, y) in this query, therefore ds is added to the allowable distance from the course. Here, target domain is limited to the first quadrant, because if the whole area is set to be a target, state explosion problem occurs. In addition, even if the target area is restricted, query 1 can not be verified because of the state explosion problem.

To solve these problems, we slightly modified `Motion` model. We added a new location named `STOP` to `Motion` model. If the gravity center goes outside the first quadrant, then transit to the location `STOP`. This modification works on verification of query 1. Instead, we also have to modify query 1 considering the new location `STOP`. New query $1'$ is as follows.

$$1'. \quad \mathtt{A}\Box\,(\textit{firstquadrant} \rightarrow \textit{inrange} \ \vee \mathtt{M.STOP})$$

where `M` is the variable name for `Motion` model in UPPAAL and `M.STOP` represents the location `STOP` in `Motion` model.

It is easily understand that the verification result for query $1'$ depends on the radius of the arc course. We verified query $1'$ by changing radius r. As a result, query $1'$ holds if $r \geq 277$ and does not hold if $r \leq 266$.

Query 2 is reachability checking that the line tracer eventually reaches to the second quadrant. This query is necessary to check behavior of the tracer, because query 1 only describe the distance from the course and does not describe movement. It makes no sense to check query 2 if query $1'$ does not hold. According to the above-mentioned results for query $1'$, we verified query 2 for $r \geq 277$. Then, we obtain verification results that query 2 holds for $r \geq 277$.

Ideally, conjunction of the two queries should be verified at once. Unfortunately, UPPAAL does not allow nesting of path quantifiers in a formula. Therefore we verified the queries one by one. However, when we consider both two queries together, it is possible to judge whether or not the tracer satisfies the specification. Note that, we verified dependency of radius by hand, but it is possible to be automated by generating UPPAAL model.

6.2 Analysis of Turning Angle

It is easily understand that verification results of query $1'$ depend on wheel speeds of the tracer. For example, if the tracer moves slowly, it will be able to keep on tracing longer. However, verification results of query $1'$ and query 2 do not describe distance from the initial position.

We calculate turning angle of the tracer by analyzing counter examples of query $1'$ for various wheel speeds. For that purpose, high wheel speed HS and low wheel speed LS are changed into $HS' = C_{ms}HS$ and $LS' = C_{ms}LS$ where C_{ms} is a coefficient. Then, we verify query $1'$ for some C_{ms}. When the query does not hold, we obtain a counter example which consists of a sequence of locations in evidence. UPPAAL has a function to generate the shortest trace as a counter example. By analyzing the counter example, it is possible to calculate the coordinate where the tracer turns off from the course. As an example, let radius r be fixed to 250. This is because that we know the tracer is not able to keeps on track in the first quadrant from the verification results in Sec. 6.1. Then, we think about intersection of the course and orbit of the tracer. Let the intersection be P, coordinates of before turning off be Q, and coordinates of after turning off be Q'. Then, P is an intersection of circle $x^2 + y^2 = (r \pm (w/2 + ds))^2$ and a line passing through Q and Q'.

Table 3 shows C_{ms}, Q, Q', P and α, where α (deg) is angle between x-axis and line passing through the origin and point P, obtained from the shortest counter examples. Note that there are no results for $C_{ms} = 1/2$ in Table 3, because query $1'$ holds. It is not surprisingly that verification results depend on wheel speeds. Query $1'$ holds for $C_{ms} = 1/2$ should be reasonable because this setting means slower move that arrows the tracer keeping on track. Fig. 5 shows a result of the orbit of the tracer obtained from the counter example, intersection P, and turning angle α for $C_{ms} = 2/3$ and $r = 250$. From the results except for $C_{ms} = 1/2$, central angle α is roughly constant. This result can be interpreted that angle α is the minimum turning radius for $r = 250$. This results seems natural, however, it indicates that model checking can be applied to analyze properties relating to turning angles.

7 DISCUSSION

In this section, we discuss our experiments and future work.

7.1 Discussion on the Experiments

We briefly return to our basic focus on our research question. We are motivated to know applicability of formal ver-

Table 3: Speed Dependency and Turning Angle ($r = 250$)

C_{ms}	Q	Q'	P	α (deg)
1/2	—	—	—	90 <
2/3	(214, 428)	(213, 435)	(213.7, 429.8)	63.6
1	(262, 394)	(262, 405)	(262.0, 402.2)	56.9
2	(310, 360)	(314, 384)	(310.9, 365.7)	68.5
3	(310, 336)	(213, 396)	(310.9, 365.7)	68.5
4	(306, 306)	(213, 384)	(310.9, 365.7)	68.5

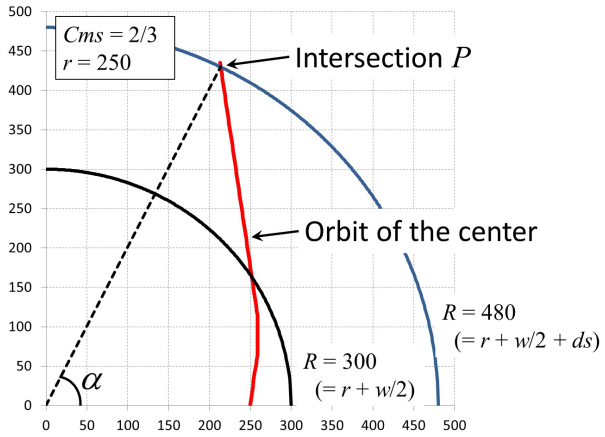


Figure 5: Orbit of the Tracer, Intersection, and Turning Angle

ification to real embedded systems, especially control continuous systems. Continuous systems are essentially hybrid systems, but we set our first target to verifying time-related properties. We also set another research question that we want to know applicability of verification techniques from the view point of design verification.

In this study, we divided the circle course into an arc course, the first quadrant, because of the state explosion problem. Here, we consider possibility of verification for tracing the entire route of the circle. To tackle this problem, straightforward modeling seems unpromising according to the verification results in Sec. 6. To reduce the size of state space, one possibility is applying abstraction techniques such as data mapping and predicate abstraction. Another possibility will be combination of theorem proving and model checking.

Experimental results combined with our previous study, behavior of a line tracer is verified based on specification and a control program. We think our verification results indicate usefulness of model checking. However, there are still problems remained to verify real embedded system. One problem is scalability. Through our studies, parameters used in verification are not the same as those used in implementation and differ from LEGO Mindstorms kit in size. However, we believe that our parameter settings are acceptable to show applicability of model checking. The reason why we adjust parameters is the state explosion problem. If we set parameters as the same as real used values, the size of state space becomes too large, and model checker cannot respond in a reasonable time or it exhausts its available memory. This problem is widely known in the field of model checking.

Another problem is that we are not yet consider effects of errors and distributions. When we think of real embedded system, behaviors of the systems are disordered by disturbances or errors. It is natural that disturbances and error probabilistically occur. However, timed automaton is not suited for probabilistic event. Here, we give a little more thought to the tracer constructed by LEGO Mindstorms as a real embedded system. It is reported in [11] that motor speed of LEGO Mindstorms kit is approximately proportional to the parameter, but has error. Through this study, we have tried to handle errors associated with wheel speeds. We assumed that wheel speed includes a certain amount of error. If such error exists, errors are cumulated and make an impact on the position of the tracer. We confirmed that such errors affect to the result of verification. Unfortunately, we have not yet obtained systematic results.

7.2 Related Work

In this section, we briefly describe related work on formal verifications associated with control engineering.

One of similar researches is verification of real-time control program using UPPAAL [12]. In this paper, the authors constructed a brick sorter system using LEGO RCX and wrote control programs in Not Quite C (NQC). The paper presents verification of safety and liveness properties by automatic translation from the control program into UPPAAL models. Through the research, abstraction and reduction techniques are applied to construct discrete models from continuous systems. This approach is similar to ours, however, the brick sorter system is essentially a discrete system even though it contains time dependencies.

As with many control systems, a line tracer can be considered as a hybrid system by describing their movements using differential equations and their control programs in discrete time. It is generally accepted that real embedded systems are too big to fully verify. Therefore, it is usual to focus on important behaviors. As an example of hybrid approaches, paper [13] described the verification of the behaviors of a line tracer by constructing a model using hybrid I/O automata and correctness proofs. In that paper, the authors presented verification of safety property, that is, a line tracer should move along a straight line and never run off. However, the authors noted that some time details, such as time delay between two motors, were not considered

In verification of robotics, a survey of model checking of the control system of NASA robotics systems is reported [14]. In this survey, the authors summarize various techniques for verification and show verification of a robot control system. Safety and liveness properties are verified, but these properties were not related to continuous dynamics. Even though the survey does not cover the handling of continuous dynamics, it is a good resource. As a similar area, the verification of a real vehicle is presented [15]. Even though our aim is the verification of continuous systems, our approach in reflects those above, *i.e.*, conversion to timed automata using quantization and sampling.

8 CONCLUSION

In our previous study, we have verified that a line tracer runs along a straight line. In this research, we used the same control program for the tracer and showed the same models can keep track on arc courses. These are verified using UPPAAL with timed automata and logical formulas. We also presented that if the tracer cannot run from the first quadrant to the second quadrant, it is possible to calculate turning angle by analyzing counter examples.

We hope to extend this study to the analysis of more real embedded systems including disturbances and errors. To that purpose, expressive power of timed automaton is not sufficient as described in Sec. 7. We plan to express models in probabilistic timed automata (PTAs). We also intend to use the latest version of probabilistic model checker PRISM [16] which supports PTAs.

Another direction of future work includes a PID controller (proportional-integral-derivative controller), which is a widely used feedback control system. We used simple specification to control the line tracer, but PID control is widely used in control systems and control engineering. When PID control is applied to a line tracer, it enables smooth motion. However, PID control is essentially hybrid system, which continuous and discrete dynamics are mixed with time progression. Several approaches have been proposed to handle hybrid systems. One of these approach is hybrid automata [17] which is a formal model for describing discrete-continuous systems.

REFERENCES

[1] R. Alur, and D.L. Dill, "A theory of timed automata," Theoretical Computer Science, Vol.126, No.2, pp.183–235 (1994).

[2] J. Bengtsson, and W. Yi, "Timed Automata: Semantics, Algorithms and Tools," in Lectures on Concurrency and Petri Nets, Advances in Petri Nets (4th ACPN'03), Vol.3098 of Lecture Notes in Computer Science (LNCS), pp.87–124 (2004).

[3] A. Schild, and J. Lunze, Control design by means of embedded maps, in Handbook of Hybrid Systems Control, eds. J. Lunze, and F. Lamnabhi-Lagarrigue, chapter 6.5, pp.231-247, Cambridge University Press (2009).

[4] E.M. Clarke, O. Grumberg, and D. Peled, Model Checking, MIT Press (1999).

[5] E.A. Emerson, "Temporal and Modal Logic", Handbook of Theoretical Computer Science, Vol.B, chapter 16, pp.995–1072, Elsevier (1990).

[6] K. Okano, T. Sekizawa, H. Shimba, H. Kawai, K. Hanada, Y. Sasaki, and S. Kusumoto, "Verification of Safety Property of Line Tracer Program using Timed Automaton Model," International Workshop on Informatics (IWIN2012), pp.136–142 (2012).

[7] K. Okano, T. Sekizawa, H. Shimba, H. Kawai, K. Hanada, Y. Sasaki, and S. Kusumoto, "Verification of Safety Properties of a Program for Line Tracing Robot using a Timed Automaton Model," Special Issue of the International Journal of Informatics Society (IJIS), Vol.5, No.3, pp.147–155 (2014).

[8] LEGO Mindstorms NXT official website, `http://www.legoeducation.jp/mindstorms/`.

[9] LeJOS Java for LEGO Mindstorms, `http://lejos.sourceforge.net`.

[10] NXC Tutorial, `http://bricxcc.sourceforge.net/nbc/nxcdoc/NXCtutorial.pdf`.

[11] K. Yamabe, "Measurement of Performance Characteristic of LEGO NXT using LeJOS," (2011), (undergraduate thesis in Osaka Gakuin University, written in Japanese).

[12] T.K. Iversen, K.J. Kristoffersen, K.G. Larsen, M. Laursen, R.G. Madsen, S.K. Mortensen, P. Pettersson, and C.B. Thomasen, "Model-Checking Real-Time Control Programs - Verifying LEGO MINDSTORMS Systems Using UPPAAL," In Proc. of 12th Euromicro Conference on Real-Time Systems, pp.147–155 (2000).

[13] A. Fehnker, F.W. Vaandrager, and M. Zhang, "Modeling and Verifying a Lego Car Using Hybrid I/O Automata," Models, Algebras, and Logic of Engineering Software, Vol.191, pp.385–402 (2003).

[14] N. Sharygina, J. Browne, F. Xie, and V. Levin, "Lessons learned from model checking a NASA robot controller," Formal Methods in Systems Design Journal, pp. 241–270 (2004).

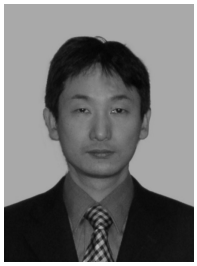
[15] M. Proetzsch, K. Berns, T. Schuele, and K. Schneider, "Formal Verification of Safety Behaviours of the Outdoor Robot RAVON," Fourth International Conference on Informatics in Control, Automation and Robotics (ICINCO), pp. 157–164 (2007).

[16] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of Probabilistic Real-time Systems," Proc. 23rd International Conference on Computer Aided Verification (CAV'11), eds. G. Gopalakrishnan, and S. Qadeer, Vol.6806, pp.585–591 (2011).

[17] T.A. Henzinger, "The Theory of Hybrid Automata," In Proceedings of the 11th Annual Symposium on Logic in Computer Science, pp.278–292 (1996).

(Received December 11, 2013)
(Revised May 7, 2014)

Toshifusa Sekizawa received his MSc degree in physics from Gakushuin University in 1998, and Ph.D. in information science and technology from Osaka University in 2009. He previously worked at Nihon Unisys Ltd., Japan Science and Technology Agency, National Institute of Advanced Industrial Science and Technology, and Osaka Gakuin University. He is currently working at College of Engineering, Nihon University. His research interests include model checking and its applications.



Kozo Okano received the BE, ME, and Ph.D degrees in Information and Computer Sciences from Osaka University, in 1990, 1992, and 1995, respectively. Since 2002, he has been an associate professor in the Graduate School of Information Science and Technology, Osaka University. In 2002, he was a visiting researcher of the Department of Computer Science, University of Kent at Canterbury. In 2003, he was a visiting lecturer at the School of Computer Science, University of Birmingham. His current research interests include formal methods for software and information system design. He is a member of IEEE_CS, IEICE of Japan and IPS of Japan.

Ayako Ogawa received the BSc in informatics from Osaka Gakuin University in 2014. She is currently working at Sougou System Service Corporation. Her research interests include system development methodology and verification of behaviors of vehicles using model checking.

Shinji Kusumoto received the BE, ME, and DE degrees in information and computer sciences from Osaka University in 1988, 1990, and 1993, respectively. He is currently a professor in the Graduate School of Information Science and Technology at Osaka University. His research interests include software metrics and software quality assurance technique. He is a member of the IEEE, the IEEE Computer Society, IPSJ, IEICE, and JFPUG.