# Verification of Safety Property of Line Tracer Program using Timed Automaton Model

Kozo Okano[†], Toshifusa Sekizawa[‡], Hiroaki Shimba[†], Hideki Kawai[‡], Kentaro Hanada[†], Yukihiro Sasaki[†], and Shinji Kusumoto[†]

[†]Graduate School of Information Science and Techlology, Osaka University, Japan
[‡]Faculty of Informatics, Osaka Gakuin University, Japan
{okano, h-shimba, k-hanada,y-sasaki, kusumoto}@ist.osaka-u.ac.jp
{sekizawa,09s0100}@ogu.ac.jp

***Abstract*** - Recently reliability of embedded systems has become very important. Such reliability can be ensured by formal verification techniques including model checking. We study such verification technique through a real example of embedded systems, a line tracer. This paper mainly describes how to model the behavior of the line tracer in a network of timed automata, as well as experimental results of verification. Using the model, several safety properties are successfully verified with a model checker, UPPAAL. The line tracer is built with LEGO Mindstorms kit. This paper also describes the implementation using LeJOS, a Java development environment for LEGO Mindstorms kit.

***Keywords***: Embedded System, Real-time System, Formal Verification, Timed Automaton

## 1 INTRODUCTION

Recently, embedded systems have become important in our society. Embedded systems exist everywhere in our daily life. Therefore, to ensure safety properties of the embedded systems becomes much important. In order to ensure such property, model checking techniques are often used. Most of model checking techniques are based on finite state machine model. The behavior of the target systems is modeled in a tuple of automata. In usual, program variables, such as integer variables are translated into finite state variables. For example, a general 32-bit integer variable might be translated into an 8-bit integer as long as the target program does not use constants with large values greater than 127. Even such modeling can detect important faults in design phases. On the other hand, some of embedded systems require time properties as their specification. In order to model such systems (real-time systems), several models have been proposed. Timed automaton is proposed by Alur and Dill[1]. The most interesting point of timed automaton is that it uses clock variables where the range of a clock variable is real numbers. Locations and transitions of timed automaton have constraints on clocks in limited syntax forms. Timed automaton, therefore, can represent naturally behavior of real-time systems. The famous verifier for timed automaton is UPPAAL[2], which is developed by Wang-Yi's research group. The timed automaton used in UPPAAL is a strong extension of the original timed automaton. It can deal with bounded integer variables and guard expressions on its transitions can express constraints on such variables. Several success applications of verification have reported, including verification of audio-video protocols[3], a gear controller[4], timeliness properties of multimedia systems [5], and so on.

Embedded systems sometimes control continuous systems. For examples, water level controller observes level of water in a certain water tank and controls incoming and outgoing valve flow associated with the tank. Note that the water level and the valve flow are usually continuous values. In order to deal with such a system consisting of discrete sub-systems and continuous sub-systems, hybrid automaton has been also proposed. Several studies have proposed simulators for hybrid systems.

Our research question is how to verify formally behavior of such hybrid systems[6]. Our first step of the research is to find what properties can be verified using conventional verifiers such as UPPAAL and so on, through a real application. We use a line tracer as a real application for the following reasons.

- It contains time properties as design specification;
- We can implement a real system with reasonable costs using LEGO Mindstorms kit [7]; and
- We can freely describe the control program in Java using LeJOS[8], which is free software for LEGO Mindstorms.

Through experiments, we have succeeded in verification of safety properties of a line tracer, using timed automaton model and UPPAAL.

The rest of the paper is organized as follows. Sec. 2 provides preliminaries. Sec. 3 and 4 will describe the model and implementation of our line tracer. Sec.5 and 6 show some preliminary but promise experiments and discussion. Finally, Sec. 7 concludes the paper. The final section also provides future plan of our work.

## 2 PRELIMINARIES

Here, we will provide several definitions and notions used in this paper.

### 2.1 Timed Automaton

A timed automaton is an extension of the conventional automaton with clock variables and constraints for expressing real-time dynamics. They 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 inequality of the following 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.*

The above time constraints are used to mark edges and nodes of the timed automata 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 a guard, a reset and a location invariant, we introduce some notations with regard to clock valuation. 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 shall intorcuce 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 $v$.

Dynamics of a timed automaton can be expressed via a set of states and their evaluations. Changes of one state to a new state can be as a result of firing of an action or elapse of 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 a set of whole states of $\mathscr{A}$, where $\mathbb{R}^C_{\geq 0}$ is a whole set of clock evaluation 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.*

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

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

**Definition 4 (semantic 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 $\sigma$ is finite or infinite sequence of transitions of $\mathscr{T}(\mathscr{A})$.*

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

## 2.2 UPPAAL

UPPAAL[2] is a famous model checker for extended timed automata by Wang-Yi et al. It also supports model checking for the conventional timed automata. UPPAAL allows verification of expressions described in an extended version of CTL. In addition, it 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. The model of the system can be created from multiple timed automata which are synchronized together via CCS-like synchronization mechanisms.

The important point is that even the extended timed automaton used in UPPAAL cannot deal with real variables except clocks. We, therefore, have to round real values to integer values when we model the target systems.

# 3 MODEL

The term "line trace" means that a vehicle traces a course starting from a certain point. The point might be on the course or not. The course is assumed to be painted in black color on white background with the same width. For example, an oval course (the same as the track used in an athletic field) is used.

A model for a line tracer consists of the following three models:

- Controller Behavior,
- State Transition of Environment, and
- Disturbance.

Controller behavior can be modeled using a state machine. Usually, controller program changes values of some of state variables based on values of some state variables.

For example, the state variables of a line tracer will be the location of the tracer, the locations of the right and left sensors, the output values of the right and left sensors, direction of the line tracer, the rotation speed of left and right wheels, and so on.

The output values of the right and left sensors are used as inputs of the controller. The rotation speed of left and right wheels are used as outputs of the controller

Table 1: State Variables of a Line Tracer

| variable | description |
|---|---|
| $x$: | x-coordinate of the center of a line tracer |
| $y$: | y-coordinate of the center of a line tracer |
| $\theta$: | direction of a line tracer |
| $slx$: | x-coordinate of the left sensor of a line tracer |
| $sly$: | y-coordinate of the left sensor of a line tracer |
| $srx$: | x-coordinate of the right sensor of a line tracer |
| $sry$: | y-coordinate of the right sensor of a line tracer |
| $wl$: | revolution speed of the left wheel of a line tracer |
| $wr$: | revolution speed of the right wheel of a line tracer |
| $sl$: | the sensed value of left sensor |
| $sr$: | the sensed value of right sensor |

Table 2: Constants

| constant | description |
|---|---|
| $w$ : | width between left and right wheels of the line tracer |
| $los$: | offset to the left sensor from the vehicle center |
| $ros$: | offset to the right sensor from the vehicle center |

$los, ros$ are tupple of $(l, a)$, where $l$ and $a$ are the distacbe and angle by the center of the vehicle, respectively.

State transition of environment can be normally represented in differential equations on state variables. In a hybrid system, such equations are used, while in a finite sate model, differential-difference equations are used as approximation.

For a line tracer, the principle state variables are summarized in Table 1.

We need some other constants to model, especially constants on the size of the line tracer. Table 2 shows some of them. Figure 1 also illustrates the relations on the state variables and constants.

Let assume that a line tracer turns with the speed of left and right wheels at $h_s$ and $l_s$. Then equations of motion can be given as follows.

$$\frac{d\theta}{dt} = \frac{h_s - l_s}{w} \quad (1)$$

$$\frac{dx}{dt} = -r_c \cdot \sin\theta \cdot \frac{d\theta}{dt} \quad (2)$$

$$\frac{dy}{dt} = r_c \cdot \cos\theta \cdot \frac{d\theta}{dt} \quad (3)$$

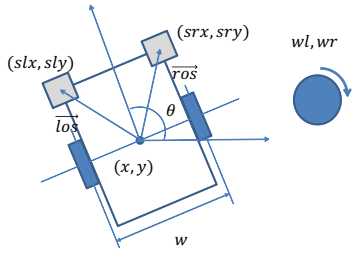


Figure 1: Constants and State Variables

Table 3: Conversion Table for Sine Function

| domain of $x$ (degree) | round of $100\times \sin(x)$ |
|---|---|
| $[0, 10)$ | 8 |
| $[10, 20)$ | 26 |
| $[20, 30)$ | 42 |
| $[30, 40)$ | 57 |
| $[40, 50)$ | 71 |
| $[50, 60)$ | 82 |
| $[60, 70)$ | 91 |
| $[70, 80)$ | 96 |
| ⋮ | ⋮ |
| $[350, 360)$ | -9 |

$$r_c = \frac{w}{2} \cdot \frac{h_s + l_s}{h_s - l_s} \quad (4)$$

Disturbance can be modeled as uncertain error for each of observation variables. For example, a line tracer has sensors. The value $s$, the output of the sensor may change with uncertain value as like the following equation:
$s_o = s_r + \varepsilon(s)$, where variables $s_o$, $s_r$, and $\varepsilon(s)$ represent the observed value, ideal value and error in observation, respectively.

### 3.1 Quantization

The timed automaton used in UPPAAL can model the controller behavior. It, however, uses integer variables only. As we know, most of state variables must have values in real. Therefore, we have to approximate such variables into integer variables.

Most of state variables use trigonometric functions (see equations (2) and (3)). Thus, we have to approximate the functions to round up into integers as long as we use finite models. Though, the values of trigonometric functions range in [-1,1], it is not a good idea that we use only three values -1, 0, and 1. Therefore, we assume that trigonometric functions range in [-100, 100]. Also we adopt degree as unit for angle. Table 3 shows an approximation conversion table for sine function.

### 3.2 Sampling

Yet another problem is that we cannot deal with functions on time. Usually state variables can be represented as function on time, however, even UPPAAL does not provide functions on time. Therefore, we have to regard state variables as discrete signals.

Sampling is a great tool to reduce a continuous signal into a discrete signal. For a discrete signal, we can model its change on time as an timed automaton with update functions.

Let's consider again state variables, $x$, $y$, $\theta$, $slx$, $sly$, $srx$, $sry$, $wl$, $wr$, $sl$, and $sr$. In usual, $slx$ and $sly$ are calculated using $x$ and $y$ with some parameters in Table 2. The values of $sl$ and $sr$ are also determined from the value of $x$, $y$, $los$, $ros$, and a course model, which consists of some parameters and the equations of the course. The values of $wl$ and $wr$ are determined by the controller.

Table 4: Logic for Color Sensors

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

Therefore, we need calculate the current value of $x$, $y$ and $\theta$ like as equations (1) $\sim$ (4).

Using sampling and update functins, we can model that the values of variables are updated every some fixed unit of time using small deltas. We will explain concrete update expressions in Sec. 5.

## 4 IMPLEMENTATION

LEGO Mindstorms NXT[9] is a kit for assembling robots with various actuators and sensors by LEGO®. Users can program its behavior. The actuators include stepping motors which users can accurately control rotation angles. The sensors include color sensors, touch sensors, sound sensors and so on. Various programming languages are provided for control of the NXT kit. The famous languages are NXC (Not eXactly C)[10] and LeJOS. LeJOS is a development environment for Java. NXC and LeJOS have classes for the above sensors and actuators.

This research uses LeJOS for developing the line tracer. Our line tracer has two color sensors locating left front and right front of the tracing car.

Table 4 shows the logic for the sensors. For example, if LeftSensor and RightSensor sense white and black colors, respectively, then the controller issues the turn right command to motors.

The output of sensors is a bounded integer value. If the value is greater than some threshold, then controller treats it as white. For the command, left and right wheel motors react independently. For example, "turn left" command makes left and right wheel motors speed up and dowm, respectively

Figure 2 shows the controller in LeJOS. Figure 3 shows the implemented line tracer.

## 5 EXPERIMENTS

Here, we deal with an ideal model. Therefore, we ignore disturbance. Figures 4 and 5 corresponds to controller behavior model and state transition of enviroment model. In this experiment, we use a simple controller program, where revolution speed of wheels has only two values, $h_s$ and $l_s$. Moreover we assume that sensors only tell white and black colors on the track. In other words, the values of $sl$ and $sr$ are determined by only the position of the line tracer. On the other hand, we model the delay of sensors and actuators. Concretely, we have parameters $d_s$, $d_a$, and $d_t$ for delay between the time when program senses color and the time when the sensors obtain the values of colors, delay between the time when program issues a command and the time when the motor reacts, and sleeping time for next sense-act loop, respectively. This modeling represents real behavior of a line tracer.

Figure 4 shows the control behavior model of the program.

```
import lejos.nxt.Button;
import lejos.nxt.ColorSensor;
import lejos.nxt.SensorPort;
import lejos.nxt.ColorSensor.Color;
import lejos.nxt.LCD;
import lejos.nxt.Motor;
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 (rid == BLACK)
       motor.B.setSpeed(LS);
     else        motor.B.setSpeed(HS);
     if (lid == BLACK)
       motor.C.setSpeed(LS);
     else
       motor.C.setSpeed(HS);
     if (Button.readButtons()
       == Button.ENTER.getId())
       break;
   }
  }
}
```

Figure 2: Controller in LeJOS

Figure 3: The Implemented Line Tracer

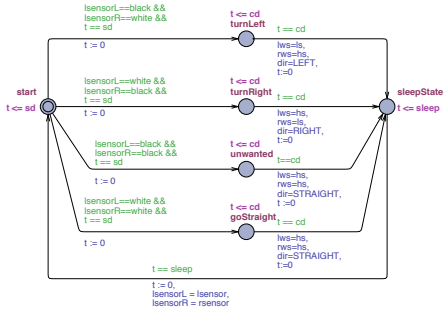


Figure 4: Timed Automaton Representing the Controller

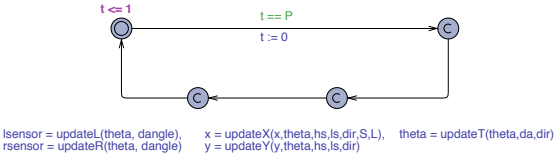


Figure 5: Timed Automaton Representing Update

Figure 5 shows the timed automaton which updates periodically state variables every unit of time. The automaton periodically calls functions updateX, updateY, updateT, updateL, and updateR which update state variables $x, y, \theta, sl$, and $sr$, respectively. The automaton first updates the value of $\theta$, and then values of $x$ and $y$. Finally it updates values of $sl$ and $sr$ based on the values of $x, y$, and $\theta$.

The following equations are equations for $\theta, x$, and $y$ used in update functions.

$$\theta' = \theta + \alpha \quad (5)$$

$$x' = x + \frac{wl + wr}{2} \cos\theta \quad (6)$$

$$y' = y + \frac{wl + wr}{2} \sin\theta \quad (7)$$

$$\alpha = 90 \cdot \frac{wr - wl}{w \cdot \pi} \quad (8)$$

If we assume that the unit of time is small then the moving distance of the vehicle can be approximated to $(h_s + l_s)/2$. The above equations uses this fact.

Please note that we actually use not sin but pseudo sin $/100$ defined in Table 3. Also we let the values of the parameter $p$ range in $[0, 360]$ by using an expression $(p + 360)\%360$.

We also assume that the course is a straight line along with $x$-axis.

We can verify the following queries:

1. $E\Diamond(900 < x)$.
2. $E\Diamond(\texttt{C.turnRight})$.
3. $E\Diamond(\texttt{C.turnLeft})$.
4. $E\Diamond(\texttt{C.unwanted})$.

Table 5: Parameters under Verification

| params | value | description |
|---|---|---|
| $wc$: | 100 | width of the track line |
| $w$: | 120 | width between left and right wheels of the line tracer |
| $los$: | $(180, 30°)$ | offset to the left sensor from the vehicle center |
| $ros$: | $(180, -30°)$ | offset to the right sensor from the vehicle center |
| $h_s$: | 12 | high speed |
| $l_s$: | 6 | low speed |
| $x_0$: | -200 | initial value of x-coordinate of the center of the vehicle |
| $y_0$: | 200 | initial value of y-coordinate of the center of the vehicle |
| $\theta_0$: | $340°$ | initial value of direction of the vehicle |
| $d_s$: | 1 | time delay of sensors |
| $d_a$: | 1 | time delay of actuators |
| $d_s$: | 2 | periodical sleeping time |

5. $A\Box\neg(\texttt{C.unwanted})$.
6. $E\Diamond(\texttt{C.goStraight})$.
7. $A\Box((x > 280) \Rightarrow (-100 < y < 100))$.
8. $A\Box((x > 280) \Rightarrow (\theta < 10 \vee 350 < \theta))$.
9. $E\Diamond((x > 280) \Rightarrow \texttt{C.turnRight})$.
10. $E\Diamond((x > 280) \Rightarrow \texttt{C.turnLeft})$.

The first query (1) means that the line tracer will reach the area $x > 900$. Queries (2) and (3) mean that the controller eventually reach state C.turnRight and C.turnLefit. Queries (4) and (5) mean that the controller eventually reach state C.unwanted and that the controller never reach state C.unwanted, respectively, where both of sensors detect black color. Query (6) means that the controller eventually reach state C.goStraight.

Queries (7), (8), (9) and (10) uses the assumption that a line tracer is in stable state. Please note that we think after the point $x = 280$, the tracer is in stable. We can observe it from several traces of simulation. The traces can be obtained from the UPPAAL using the simulation mode view.

Queries (7) and (8) mean that the line tracer roughly keeps the track and appropriate direction, respectively, in the stable state.

The last two queries mean that the line tracer eventually turns left or right even if the tracer is in stable state.

Every of the verifications (except the query (4)) has succeeded with the parameters in Table 5. Every verification is performed within one sec. using UPPAAL ver. 4.0.13 academic licence on Windows 7 64 bit OS, Intel Core i7 960 3.20GHz, with 12 GB memory.

Query (4) passed if we change the parameter $los$ and $ros$ as (170, 30°) and (170, -30°). In this time, of course, Query (5) changes to false.

We assume that if the value of $x$ becomes greater than 1000 then the value of $x$ is reset to 50. This device let a line tracer run infinitely in the finite state model.

Figure 6 shows verification process using UPPAAL.

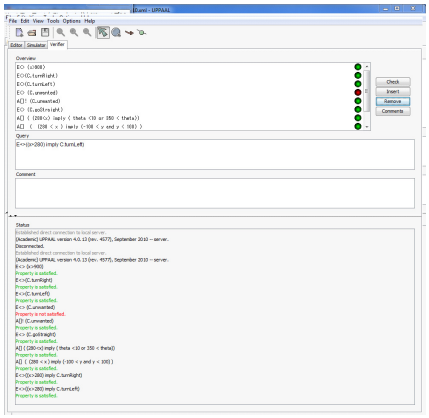

Figure 6: Verification using UPPAAL

## 6 DISCUSSIONS

Here, we will describe discussions on the experiments and hybrid systems.

### 6.1 Discussions on The Experiments

The results are not enough to convince us that the line tracer runs safely. The results, however, show that from the theoretical point of view, our approach using a verifier for timed automata, will work.

The parameters used in verification are not the same to the parameters used in the implementation. This might lessen the validity of the model. However, the proportional relations of the parameters are acceptable. For example, the width $ws$ between the left sensor and the right sensor is 190 and it is greater than 100, the width of the track.

The value $ws$ is greater than 120, the width of the line tracer. It is very different from the implementation in Figure 3. The parameters are, however, acceptable. Also the wheel speed 6 or 12 is acceptable with regard to the size of the line tracer.

The workload of modeling is in fact not less (it takes over 2 month-persons) due to our limit of know-how on modeling, especially how to deal with continuous model. Some of parameters in Table 5 are very sensitive, in other words, if these values are different by a little, the behavior of the whole system differs; consequently, verification will fail. For example, with the parameters in Table 5 if we change the value of $d_s$ as 4, then the verification fails.

You might think that slow wheel speed increases the possibility of success of verification. In other words, the slower a line tracer moves, the more successively it keeps the track. However, due to the quantization, a small wheel speed causes the delta values per unit of time to be 0 in our model. Therefore, we cannot set smaller value than 6 as the low speed of the wheel. Such a problem can be resolved by increasing the physical sizes in the model. However, such a revise, in turn, causes so called state explosion in which a model checker cannot response in a reasonable time or exhausts whole of the memory space.

Nevertheless such situations, it shows the importance of design analysis and verification in an early stage of development.

During the modeling, we think that there should be an automated generation tool which translates from an abstract parameter model to a concrete UPPAAL model, as well as a simple tool to analysis counter-examples and simulation results obtained from UPPAAL. They would be very useful to refine the model.

### 6.2 Hybrid System

In order to analyze the model more precisely, hybrid systems seem promise models.

Hybrid system[11] is a system in which continuous dynamics and discrete dynamics are mixed with time progress. Hybrid systems are important in many fields such as physics and control engineering. Several approaches are proposed to deal with hybrid systems. One of the approaches is hybrid automaton[12] which is a formal model for describing mixed discrete-continuous systems. Hybrid automaton consists of variables, control graph, continuous flow, discrete jump and events. A model checker for linear hybrid automata is HyTech[13]. Another approach is hybrid constraint languages such as Hybrid cc[14] and HydLa[15]. These languages are declarative and provide power to write programs with logical formulas. Execution environment of these languages are implemented, Hybrid cc interpreter and Hyrose, respectively.

A line tracer can be a hybrid system by describing its movement using differential equations and its control program in discrete time. However, there are difficulties if a line tracer is modeled accurately. For example, modeling with characteristics of motors and sensors, and disturbances are difficulties. Fehnker et al. presented a study of verification of behaviors of a line tracer[16]. In the paper, the authors presented verification of a safety property, a line tracer move along a straight line and never run off the line, by constructing a model using hybrid I/O automata and correctness proof. However, as the authors mentioned, some kinds of time is not considered such as time delay between two motors.

## 7 CONCLUSION

We have modeled a controller of a line tracer in timed automata. Also we have verified the model to ensure that the line tracer keeps the track, using a model checker, UPPAAL.

Future plans are summarized as follows. First, we want to model a PID controller (proportional-integral-derivative controller), which is a kind of feedback controls. PID control enables a line tracer to behave more smoothly. PID control, however, needs some historical data on the past values of state variables, and also requires complicate calculation, thus hybrid modeling becomes more suitable. We want to use hybrid model, as well as its verifiers and simulators to determine suitable parameters for PID control. It is said that to find suitable parameters for PID control for an instance of problems, is

a difficult problem for a long time. We think our approach might work well.

Another direction of our research is timing analysis of motor delay. From preliminary experiments, we have found that motor delay cannot be ignored for design of controller program if we want to obtain a high quality controller.

## ACKNOWLEDGEMENT

This research is partially supported by Grant-in-Aid for Scientific Research (C) (21500036).

## REFERENCES

[1] R. Alur and D. L. Dill: "A theory of timed automata," Journal of Theoretical Computer Science, 126(2), pp.183-235 (1994).

[2] J. Bengtsson and W. Yi: "Timed Automata: Semantics, Algorithms and Tools," In Lecture Note in Computer Science, vol.3098, pp.87-124 (2004).

[3] J. Bengtsson, W. O. D. Griffioen, K. J. Kristoffersen, K. G. Larsen, F. Larsson, P. Pettersson and W. Yi: "Verification of an Audio Protocol with bus collision using UPPAAL," In Lecture Note in Computer Science, vol.1102, pp.244-256 (1996).

[4] M. Lindahl, P. Pettersson, and W. Yi: "Formal Design and Analysis of a Gear Controller: An Industrial Case Study using UPPAAL," In Lecture Note in Computer Science, vol.1384, pp.289-297 (1998).

[5] B. Bordbar and K. Okano: "Verification of Timeliness QoS Properties in Multimedia Systems," In Lecture Note in Computer Science, vol.2885, pp.523-540 (2003).

[6] J. Fitzgerald, P.G. Larsen, K. Pierce, M. Verhoel, and S. Wolff: "Collaborative Modelling and Co-simulation in the Development of Dependable Embedded Systems," in Proceedings of IFM 2010, pp.12-26 (2010).

[7] B. Bagnall: "Intelligence Unleashed: Creating LEGO NXT Robots with Java," Variant Press (2011).

[8] LeJOS Java forLEGO Mindstorms: `http://lejos.sourceforge.net`

[9] LEGO Mindstorms NXT Official website: `http://www.legoeducation.jp/mindstorms/`

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

[11] J. Lunze and F. Lamnabhi-Lagarrigue: "Handbook of Hybrid Systems Control: Theory, Tools, Applications," Cambridge University Press (2009).

[12] T. A. Henzinger: "The theory of hybrid automata," In Proc. of Eleventh Annual IEEE Symposium on Logic in Computer Science, LICS '96, pp.278-292 (1996).

[13] T. A. Henzinger, Pei-Hsin Ho, and H. Wong-Toi: "HYTECH: A model checker for hybrid systems," In Lecture Notes in Computer Science, vol.1254, pp.460-463 (1997).

[14] V. Gupta, R. Jagadeesan, V. Saraswat, and D. G. Bobrow: "Programming in hybrid constraint languages," In Lecture Notes in Computer Science, vol.999, pp.226-251 (1995).

[15] K. Ueda, H. Hosobe, and D. Ishii: "Declarative semantics of the hybrid constraint language HydLa," Computer Software, JSSST, 28(1) pp.306-311 (2011). (in Japanese).

[16] A. Fehnker, F. W. Vaandrager, and M. Zhang: "Modeling and verifying a lego car using hybrid I/O automata," In Proc. of 3rd Int. Conf. on Quality Software, pp.280-289. IEEE Computer Society (2003).