Proceedings of the 2004 IEEE
International Conference on Robotics & Automation
New Orleans, LA • April 2004

Multi-Robot Planning : A Timed Automata Approach

Michael Melholt Quottrup
Aalborg University
Department of Control Engineering
Fredrik Bajers Vej 7C
DK-9220 Aalborg Øst, Denmark
Email: mmq@control.auc.dk

Thomas Bak
Danish Institute of Agricultural Sciences
Department of Agricultural Engineering
Schüttesvej 17
DK-8700 Horsens, Denmark
Email: thomas.bak@agrsci.dk

Roozbeh Izadi-Zamanabadi
Aalborg University
Department of Control Engineering
Fredrik Bajers Vej 7C
DK-9220 Aalborg Øst, Denmark
Email: riz@control.auc.dk

Abstract— **This paper describes how a network of interacting timed automata can be used to model, analyze, and verify motion planning problems in a scenario with multiple robotic vehicles. The method presupposes an infra-structure of robots with feedback controllers obeying simple restriction on a planar grid. The automata formalism merely presents a high-level model of environment, robots and control, but allows composition and formal symbolic reasoning about coordinated solutions. Composition is achieved through synchronization, and the verification software UPPAAL is used for a symbolic verification against specification requirements formulated in computational tree logic (CTL). In this way, all feasible trajectories that satisfy specifications and which moves the robots from a set of initial positions to a set of desired goal positions may be algorithmically analyzed. The trajectories can then subsequently be used as a high-level motion plan for the robots. This paper reports on the timed automata framework, results of two verification experiments, promise of the approach, and gives a perspective for future research.**

I. INTRODUCTION

The problem of controlling mobile multi-robot systems in a coordinated manner is becoming an important research issue. The vision being that multiple robotic vehicles may perform tasks faster and more efficient that a single robot. The application domains include container transshipment tasks in harbors, airports, and manufacturing areas, and formation-keeping and control in military applications [1], [2].

A number of different approaches have been taken in order to coordinate multi-robot system. A formalism for the composition of concurrent robot behaviors, using threaded petri nets, has been developed and used for the construction of simple automated factories, such as mobile robot bucket brigades [3]. In [4] multi-robot coordination is achieved by employing a plan-merging paradigm that guarantees coherent behavior of all the robots in all situations. A distributed negotiation mechanism for multi-robot coordination is considered in [5]. A hybrid control approach to action coordination and collision avoidance was taken in [6], [7]. A formal hybrid approach to the modeling and analysis of coordinated multi-robot systems was taken in [8].

In this work we provide a novel framework for the modeling of a mobile multi-robot system as a network of interacting timed automata [9] based on synchronization. The model is verified against specification requirements formulated in computational tree logic (CTL) using the verification tool UPPAAL [10]. For timed automata, problems such as reachability and CTL model checking are decidable [11], [12].

The methods developed here presupposes an infra-structure of robots with feedback controllers that constrain the robot to move in a planar grid, i.e. motion from one cell in the grid to a neighboring cell is constrained to move within the two cells. The automata formalism merely presents an abstraction (or high-level model) of this, but allows composition and formal symbolic reasoning about coordinated solutions. The system is described as interacting models of the possible behaviours of robots, control and environment, under velocity constraints set by the assumed hardware. Posing the problem in UPPAAL, enables us to algorithmically verify safety questions such as: will they collide?; and liveliness questions such as: are all robots able to reach their goal positions? We may also algorithmically investigate all feasible trajectories (or corresponding control actions) that satisfy specifications and which moves the robots from a set of initial positions to a set of goal positions. The trajectories are subsequently use as the high-level motion plan for the robots.

The paper is organized as follows: In Section II we introduce timed automata to model the mobile multi-robots. In Section III we describe the modeling of mobile multi-robots. In Section IV we verify (in UPPAAL) that the developed model satisfy the specification requirements. Finally, we discuss the developed approach.

II. TIMED AUTOMATA

A timed automaton is a finite-state automaton extended with a finite collection of real-valued clock variables [9]. The clocks are assumed to proceed at the same rate and measure the amount of time that have elapsed since they were reset. The clock values may be compared with natural numbers and reset to zero. Let $\mathbb{C}$ be a set of real-valued clock variables. Then $\mathcal{B}(\mathbb{C})$ is the set of formulas that are conjunctions of atomic constraints of the form $c \bowtie n$ and $c - d \bowtie m$ for all $c, d \in \mathbb{C}, \bowtie \in \{\leq, <, =, >, \geq\}$, $n \in \mathbb{N}$ and $m \in \mathbb{Z}$. Elements of $\mathcal{B}(\mathbb{C})$ are called guards over $\mathbb{C}$ and $2^{\mathbb{C}}$ denotes the power set of $\mathbb{C}$. We adopt the definition of a timed automaton from [13].

0-7803-8232-3/04/$17.00 ©2004 IEEE

Definition 1 (Timed Automaton): A timed automaton $\mathcal{A}$ over actions Act, atomic propositions $\mathcal{P}$, and clocks $\mathbb{C}$ is a tuple $\mathcal{A} = (L, l_0, E, I, V)$, where:

- L is a finite set of control locations;
- $l_0 \in L$ is the initial control location;
- $E \subseteq L \times \mathcal{B}(\mathbb{C}) \times Act \times 2^{\mathbb{C}} \times L$ is a finite set of edges, where an edge contains a source control location, a set of guards, a set of actions to be performed, a set of clocks to be reset, and a target control location;
- $I : L \rightarrow \mathcal{B}(\mathbb{C})$ is a function which for each control location l assigns a clock constraint, also called the invariant condition of the control location;
- $V : L \rightarrow 2^{\mathcal{P}}$ is a proposition assignment function which for each control location gives a set of atomic propositions true in that control location. $2^{\mathcal{P}}$ denotes the power set of $\mathcal{P}$.

A timed automaton can be viewed as a hybrid system in which there is one or more continuous variables $c_i, d_i \in \mathbb{C}$ that satisfies the differential equation $\dot{c}_i = 1$, $\dot{d}_i = 1$. A state of a timed automaton $\mathcal{A}$ is a pair (l, u) where l is a control location of $\mathcal{A}$ and u holds the current values for the clock variables. The initial state of $\mathcal{A}$ is (l_0, u_0), where u_0 assigns zero to all clocks in $\mathbb{C}$.

Definition 2 (Control location trajectory): A control location trajectory is a sequence of control locations $l_0\ l_1\ l_2\ \ldots$ such that $(l_i, l_{i+1}) \in E,\ \forall i \geq 0$.

A. Formal Analysis

Formal analysis is concerned with verifying whether a system satisfies a desired specification. Properties about the behavior of a system over time are naturally expressible in temporal logics, e.g. computation three logic (CTL) [14]. CTL is a temporal logic that contains existential quantifiers that range over trajectories and it allows to reason about how the states of the system evolve over time.

Formulas of temporal logic are thus used to formally specify desired properties of systems, such as reachability (whether a certain region of the state-space can be reached) and invariance properties. CTL formulas consists of atomic propositional logic formulas and temporal operators. The propositional logic formulas are expressions about the state of the system and temporal operators are expressions about trajectories into the future that the state of the system can follow. CTL formulas are interpreted over the tree of control location trajectories generated from a given state of the automaton.

In this approach we use CTL to specify and verify desired safety and liveness properties of a systems consisting of multiple mobile robots.

III. Modeling Multi-Robots

The following describes the modeling of multiple mobile robots as a network of timed automata that interact through synchronization channels. The system being modeled is divided into: environment (workspace and obstacles), robots, and control.

To allow formal analysis and verification of the system just described, we first partition a subset of the planar environment $\mathbb{R}^2$, into a Cartesian grid of disjoint cells. The robots will be restricted to move horizontal and vertical in the grid of disjoint cells. Cells in the grid may be occupied by static obstacles. The obtained partition of the environment enables the analysis of desired safety (e.g. obstacle avoidance) and liveness (e.g. reachability) properties.

The system is modeled and verified using UPPAAL [10]; an integrated tool for modeling, analysis, and verification (model checking) of real-time systems that can be modeled as a network of timed automata interacting through synchronization channels. UPPAAL allows the construction of process templates for the system being modeled. A set of process parameters may be declared for each process template. Process assignments are then used to declare instances of the process templates with specific process parameters. This allows the construction of a system that consists of a set of robots with different process parameters, a set of static obstacles, and a set of controls for the robots. Moreover, the movement and coordination of the mobile robots is handled by the verifier of UPPAAL.

A. Environment

In the following we consider a team of R_n robots, $n = 1, \ldots, N$, restricted to operate in a planar environment, $X \subset \mathbb{R}^2$. We denote by $x_i = \begin{bmatrix} x_{i1} & x_{i2} \end{bmatrix}^{\mathrm{T}}$, $x_i \in X$, a position in the planar environment. X is partitioned into a Cartesian grid of disjoint cells with resolution $\epsilon \in \mathbb{R}^+$. With a simple layout of X and a suitable ϵ, the result is a finite partition,

$$X = \bigcup_{U} C^{\epsilon}(z_i) \tag{1}$$

where $z_i = \begin{bmatrix} z_{i1} & z_{i2} \end{bmatrix}^{\mathrm{T}}$, $z_i \in \mathbb{Z}^2$, and U is the number of cells needed to cover X. Each cell in the partition $C^{\epsilon}(z_i) \subset \mathbb{R}^2$ is defined by,

$$C^{\epsilon}(z_i) = \{x_j \in \mathbb{R}^2 : z_{i1} - \frac{\epsilon}{2} < x_{j1} \leq z_{i1} + \frac{\epsilon}{2} \wedge z_{i2} - \frac{\epsilon}{2} < x_{j2} \leq z_{i2} + \frac{\epsilon}{2}\} \tag{2}$$

The partition defined by X divides the plane into a grid of $U = (S+1)(T+1)$ cells as illustrated in Fig. 1.

A group of static obstacles may be present in the grid, each modeled as occupying cells in the partition. The workspace of the robots $\mathcal{W}$, i.e. the obstacle free space in which the robots are restricted to move, is hence restricted to,

$$\mathcal{W} = X \setminus \bigcup_{M} O(z_i) \tag{3}$$

where M denote the number of static obstacles and $O(z_i)$ is an obstacle located at $\mathbb{Z}^2$. Thus, the defined workspace represents a shared space in which all the robots may move. With a proper selection of the grid size ϵ, collision may be avoided by not allowing the robots to occupy the same cell of the partition at any time instant.

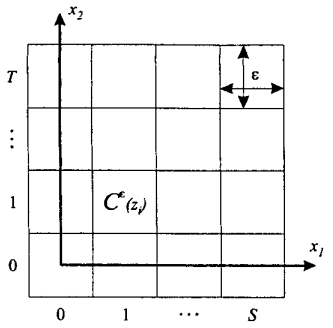

Fig. 1. Partition of X into a grid of cells $C^{\epsilon}(z_i)$.

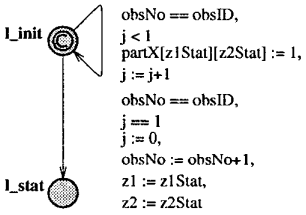

Fig. 2. Obstacle process template.

In the UPPAAL model the partition of X is represented as a two-dimensional integer array, `int[0,1] partX[hSize][vSize]` that is declared globally. `hSize` and `vSize` are constants that define the horizontal and vertical size of the array respectively. Thus, elements of the array represent cells in the partition of X. By default all cells of the array are initialized to zero, marking them as free cells. A particular cell of the array is marked as occupied by assigning it the value 1. For example, the assignment `partX[1][2]==1` marks the cell with midpoint $z_i = \begin{bmatrix}1 & 2\end{bmatrix}^{\mathrm{T}}$ as occupied.

The automaton $\mathcal{A}_O = (L, l_{init}, E)$, models a static obstacle as shown in Fig. 2. This automaton is used as a process template for declaring static obstacles. Static obstacles are declared with parameters: `const obsID, z1Stat, z2Stat`, where `obsID` is an unique identifier for the m-th obstacle, `(z1Stat,z2Stat)` is the static position of the obstacle. The local declarations of the obstacle process are: `int[0,hSize] z1; int[0,vSize] z2`, where `(z1,z2)` is the position of the static obstacle in the Cartesian grid.

$\mathcal{A}_O$ starts in the control location l_{init}, which is declared committed. By declaring this control location committed, an element in the array `partX` can be marked as occupied by an obstacle, without allowing any time delay in this control location. When the guard `(obsNo==obsId, j<1)` is enabled, the assignment `partX[z1Stat][z2Stat]:=1` is performed and the index integer `j` is incremented. The transition from l_{init} to l_{stat} will then become enabled since the guard `(obsNo==obsID, j==1)` is satisfied, resulting in a reset of `j`, an increment of `obsNo`, and the obstacle is given a static position by the assignments `z1:=z1Stat` and `z2:=z2Stat` in the Cartesian grid.

B. Robots

We restrict the robots to move horizontal and vertical in the Cartesian grid. The rationale behind this restriction is twofold. First, it reduces the state-space of the system and hence it reduces the complexity of model checking the system. Second, it reduces the number of cells that needs to be occupied when the robot moves.

Since clocks variables can only have non-negative values the position of the robots cannot directly be represented in a timed automaton using a clock. Instead, a clock $c \in \mathbb{C}$ is used to represent the time it takes the robot to move from one particular cell to a horizontal or vertical neighbour cell in the grid, i.e. $\frac{d}{dt}c = 1$.

Thus, the clock measures the amount of time spent in a particular control location, since it was reset. We restricts the time it takes the robot to move from one particular cell to a neighbour cell in the grid by imposing the constraint, $\texttt{cMin} < \texttt{c} < \texttt{cMax}$, which for $\texttt{cMin} \geq 1$ ensures that a movement of the robot takes minimum one time unit. The result is $\mathcal{A}_R = (L, l_{init}, E, I, V)$, modeling a robot,

- $L = \{l_{init}, l_{stop}, l_{mr}, l_{ml}, l_{mu}, l_{md}\}$ is the set of control locations;
- $l_{init} \in L$ is the initial control location;
- $E \subseteq L \times \mathcal{B}(\mathbb{C}) \times Act \times 2^{\mathbb{C}} \times L$ is the set of edges,

$$e_0 = (l_{init}, l_{init})$$
$$e_1 = (l_{init}, l_{stop})$$
$$e_2 = (l_{stop}, \texttt{moveRight?}, l_{mr})$$
$$e_3 = (l_{mr}, \texttt{cMin} < \texttt{c} < \texttt{cMax}, \texttt{c} = 0, l_{stop})$$
$$\text{etc.}$$

- $I : L \rightarrow \mathcal{B}(\mathbb{C})$ is a function which for each control location l assigns an invariant,

$$l_{mr}, l_{ml}, l_{mu}, l_{md} : \texttt{c} < \texttt{cMax}$$

The corresponding robot process template is shown in Fig. 3.

The local declarations of the robot process template are: `clock c; int[0,hSize] z1; int[0,vSize] z2; int[0,50] step`, `(z1,z2)` is the position of the robot in the Cartesian grid, and `step` is the number of robot movements. Process instances of the robot process template can be declared with the template parameters: `const robotID, z1Init, z2Init, cMin, cMax; chan moveRight, moveLeft, moveUp, moveDown`, where `robotID` is an unique identifier for the robot, `(z1Init,z2Init)` is the initial position of the robot, `(cMin,cMax)` is the clock constraint on the clock `c`,`moveRight, moveLeft, moveUp, moveDown` are the synchronization channels that allows control of the robot.

The timed automaton modeling the robot process template starts in the control location l_{init}. In this control location the robot is placed at its initial position in the grid, as specified by the parameters `z1Init` and `z2Init`. By declaring this control location committed the robot is placed at its initial position, without allowing any time delay in this control

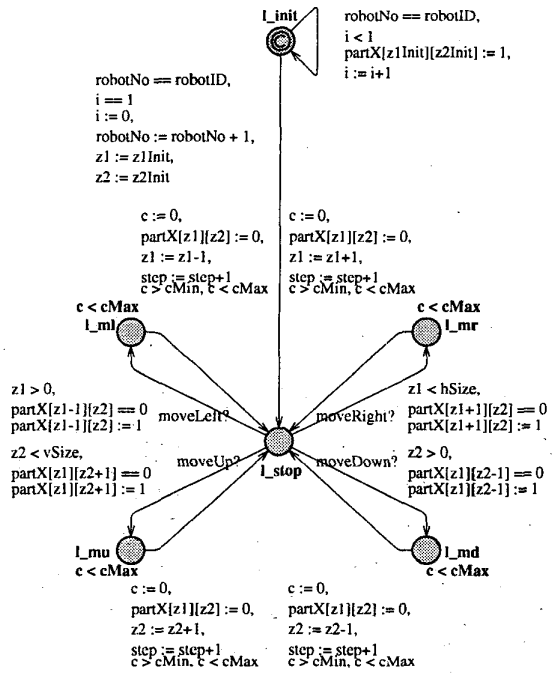

Fig. 3. Robot process template.

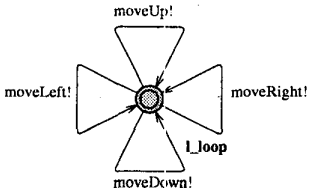

Fig. 4. Control process template.

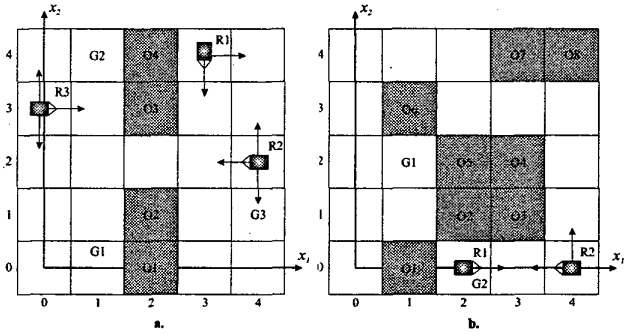

Fig. 5. Multi-robot systems. a. Three mobile robots that transverse a door, and b. two mobile robots moving in a maze.

location. Afterwards the control location l_{stop} is entered and the robot may receive control signals from the control process in order to move in one of the four possible directions. In this control location the synchronization channels are `moveRight`, `moveLeft`, `moveUp`, and `moveDown`. The synchronization labels are `moveRight?`, `moveLeft?`, `moveUp?`, `moveDown?`.

Synchronization channels are used to synchronize the control and robot processes as illustrated in Fig. 3 and Fig. 4, respectively. This is done by annotating edges in the models with synchronization labels evaluating to a synchronization channel. Complementary synchronization labels are of the form, e.g. `moveUp!` and `moveUp?`. The control and robot processes can synchronize on edges annotated with complementary synchronization labels if the guards of both edges are enabled. When the control and robot processes synchronize, both edges are fired at the same time and the current control location of both processes is changed.

C. Control

Robot control is achieved by the use of the synchronization labels, `moveRight!`, `moveLeft!`, `moveUp!`, `moveDown!`, representing the set of all possible control actions. This is encoded into a automaton, $\mathcal{A}_C = (L, l_{loop})$, as illustrated in Fig. 4.

This automaton is used as a process template for declaring control process instances. A control can be declared with the template parameters: `chan moveRight, moveLeft, moveUp, moveDown`. The control starts in the control location l_{loop}. In this control location the control process may send control signals, through the synchronization labels `moveRight!`, `moveLeft!`, `moveUp!`, `moveDown!`, in order to change the control location of the robot process with the complementary synchronization labels `moveRight?`, `moveLeft?`, `moveUp?`, `moveDown?`.

The update expression on the edge using the `moveRight!` is executed before the update expression on the edge with the `moveRight?` (see Fig. 3). This is interpreted as the control process sends control signals to the robot process in order to change its control location.

IV. Model Checking

A. Multi-robot System

In the following we consider two distinct concurrent mobile robot systems as illustrated in Fig. 5. The system in Fig. 5.a involves three mobile robots that have to transverse a door and the other system (see Fig. 5.b involves two robots that have to change positions in a maze. The system in Fig. 5.a consists of the following processes: Three robots (R1-R3), three controls (C1-C3), and four static obstacles (O1-O4).

The cells denoted G1, G2, G3 marks the goal positions of robots R1, R2, R3, respectively. The robots have to move from their initial to their goal positions while avoiding collision with each other and static obstacles.

The global declarations for the system in Fig. 5.a are,

```
clock time; const hSize 5; const vSize 3; const N 5;
const M 5; int[0,1] partX[hSize][vSize]; int[0,5] i
,j; int[1,N] robotNo; int[1,M] obsNo; chan mR1,mL1,
mU1, mD1,mR2,mL2,mU2,mD2,mR3,mL3,mU3,mD3;
```

The clock `time` is used to measure the global time for the system. The system is constructed in UPPAAL where process assignments are used to declare instances of the robot, control, and obstacle process templates, respectively,

```
R1:= Robot(1,3,4,1,4,mR1,mL1,mU1,mD1);
C1:= Control(mR1,mL1,mU1,mD1);
O1:= Obstacle(1,2,0);
```

For example, the instance `R1` of the robot process template is declared with the parameters: `robotID = 1, z1Init = 3, z2Init = 4, cMin = 1, cMax = 4; moveRight = mR1, moveLeft = mL1, moveUp = mU1, moveDown = mD1.` The instance `C1` of the control process template is declared with the parameters: `moveRight = mR1, moveLeft = mL1, moveUp = mU1, moveDown = mD1.` The instance `O1` of the obstacle process template is declared with the parameters: `obsID = 1, z1Stat = 2, z2Stat = 0.` The remaining robots, controls, and static obstacles of the first system are declared is a similar manner.

System properties to be specified are often expressed in terms of temporal logic formulas that describe the desired behavior of the system. The system properties are of the form: `process.controlLocation`. In the following we will check conjunctions of system properties, e.g. `R1.z1 and R1.z2`.

The following properties of the multi-robot system (see Fig. 5.a will be checked using the verifier in UPPAAL.

1) Safety Properties: **Prop. 1** (Collision Avoidance): For all control trajectories the robots never collide, after they start to move?

```
A[] not ((R1.z1==R2.z1) and (R1.z2==R2.z2) and
(R1.z1==R3.z1) and (R1.z2==R3.z2) and (R2.z1==R3.z1)
and (R2.z2==R3.z2) and (time >0))
```

Prop. 2 (Bounded Movement): For all control trajectories the robots never move outside workspace? (*only shown for R1*)

```
A[] R1.z1>=0 and R1.z1<=hSize and R1.z2>=0 and
R1.z2<=vSize
```

2) Liveness Properties: **Prop. 3** (Reachability 1): Does there exist a control trajectory where the robots eventually reach their goal positions?

```
E<> R1.z1==1 and R1.z2==0 and R2.z1==0 and R2.z2==2
and R3.z1==4 and R3.z2==1
```

Prop. 4 (Reachability with time requirement): Does there exist a control trajectory where the robots eventually reach their goal position within 10 tu?

```
E<> R1.z1==1 and R1.z2==0 and R2.z1==0 and R2.z2==2
and R3.z1==4 and R3.z2==1 and time<=10
```

Prop. 5 (Reachability with step requirement): Does there exist a control trajectory where the robots eventually reach their goal positions within 10 step movements?

```
E<> R1.z1==1 and R1.z2==0 and R2.z1==0 and R2.z2==2
and R3.z1==4 and R3.z2==1 and R1.step<=10 and
R2.step<=10 and R3.step <=10
```

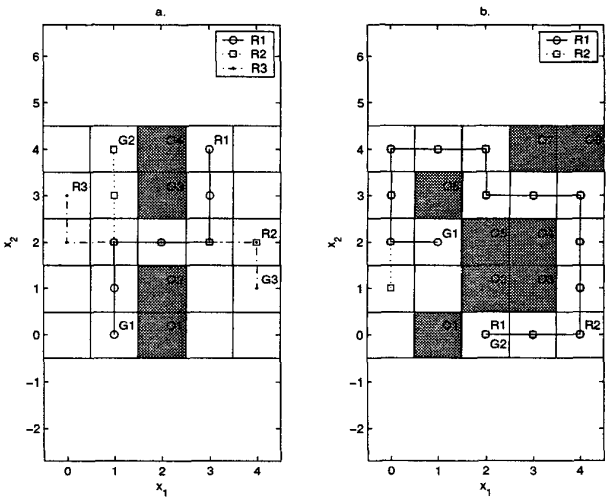

Fig. 6. Robot movements in the x_1x_2-plane. a. system with three robots, b. system with two robots.

V. RESULTS

In the following we provide the results from the verification of the properties of the multi-robot system as described in Section IV, Fig. 5.a. By construction the first safety property (Prop. 1) should be satisfied since the robots are able to occupy the cell they are moving to, enforced by the guards on the transitions to the move locations, e.g. `partX[z1+1][z2]==0`, see Fig. 3. This property was satisfied when using the verifier in UPPAAL. Moreover, the robots are able to move within the defined workspace, also enforced by the guards on the transitions to the move locations, e.g. `z1<hSize` (see Fig. 3) and hence Prop. 2 is satisfied. The robot movements in the x_1x_2-plane is shown in Fig. 6.a.

As shown the robots are able to reach their goal positions. Moreover, given the initial positions of the robots it was possible to verify that the goal positions were reachable within the specified time requirement (i.e. 10 tu). This is shown in Fig. 7.

Thus, Prop. 3 and Prop. 4 are satisfied. Finally, Prop. 5 is satisfied since both robots are able to reach their goal positions within 10 step movements.

Similar safety and liveness properties were verified for the multi-robot system involving only two robots as shown in Fig. 5.b. Given the initial positions of the robots and static positions of the obstacles, Prop. 1 to Prop. 4 were satisfied for this system. In Fig. 8 the robot movements in the x_1x_2-plane vs. time are shown.

The robot movements in Fig. 8 represents the fastest-time control trajectories of the robots. Moreover, the lower and upper bounds on the robot movements are both shown in Fig. 9. The upper bound represents the maximum time it may take a robot to reach its goal position.

Fig. 6.b and Fig. 8 shows that `R1` and `R2` can only reach their goal positions (the cells marked `G1` and `G2`, respectively)

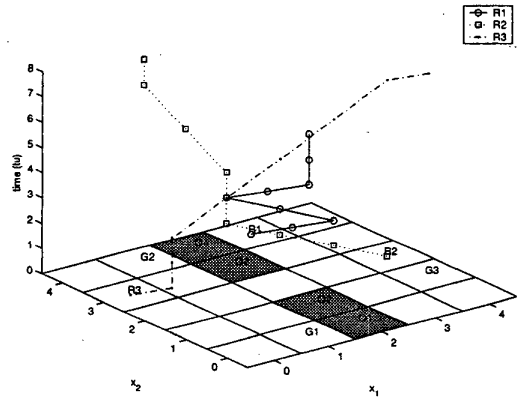

Fig. 7. Robot movements in the x_1x_2-plane vs. time.

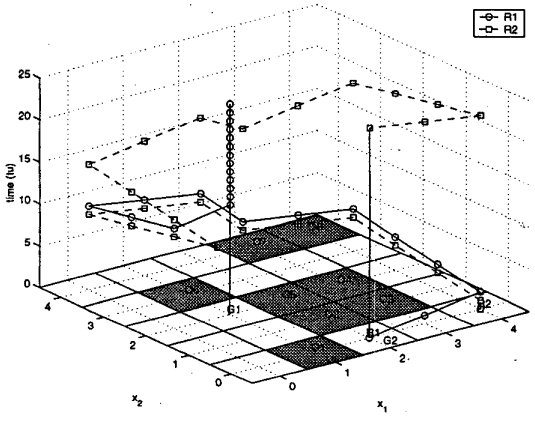

Fig. 8. Robot movements in the x_1x_2-plane vs. time.

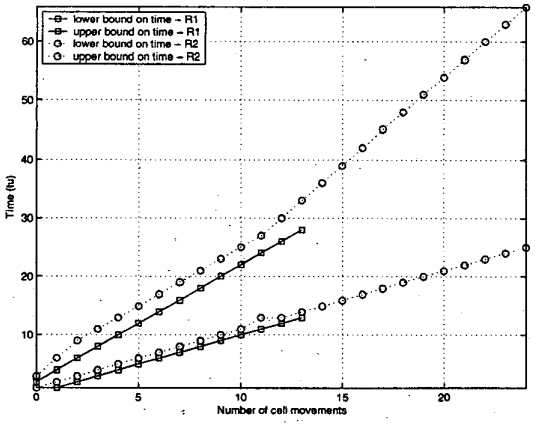

Fig. 9. Upper and lower bounds on time for robot movements.

if they change position.

VI. Conclusion

High level motion planning for multiple robots that perform various tasks in a common environment is regarded to be a major challenge and is subject to intense research. In this paper, we have used a timed-automata formalism to present robots' and controller behaviour in an environment that was restricted to a planar grid.

Subsequently, properties with direct impact on coordination solution, such as safety and liveness, where algorithmically verified and illustrated via two simple examples.

The salient feature in timed-automata formalism that is clocks enable us to refine the models and hence enhance our ability to address additional issues such as optimal solutions with respect to time or steps for a coordination problem involving different robots with different dynamic behaviours.

Understandably, model refinement implies exponential enhancement in the search space where the solution should be found. Finding locally optimal solutions in this respect would be a logical approach and is the subject of current research.

References

[1] T. Balch and R. Arkin, "Behavior-based formation control for multirobot teams," *IEEE Transactions on Robotics & Automation*, vol. 14, no. 6, pp. 926–939, Dec. 1998.

[2] R. Fierro, A. Das, V. Kumar, and J.P., "Hybrid control of formations of robots," in *Proc. IEEE International Conference on Robotics & Automation*, vol. 1, May 2001, pp. 157–162.

[3] E. Klavins and D. Koditschek, "A formalism for the composition of concurrent robot behaviors," in *Proc. IEEE International Conference on Robotics & Automation*, San Franscisco, CA, 2000, pp. 3395–3402.

[4] R. Alami, S. Fleury, M. Herrb, F. Ingrand, and F. Robert, "Multi-robot cooperation in the martha project," *IEEE Robotics & Automation Magazine*, vol. 5, pp. 36–47, Mar. 1998.

[5] B. Gerkey and M. Mataric, "Sold: Auction methods for multirobot coordination," *IEEE Trans. on Robotics & Automation*, vol. 18, no. 5, pp. 758–768, Oct. 2002.

[6] M. Egerstedt and X. Hu, "A hybrid control approach to action coordination for mobile robots," *Automatica*, vol. 38, no. 1, pp. 125–130, Jan. 2001.

[7] M. Egerstedt and C. Martin, "Conflict resolution for autonomous vehicles: A case study in hierarchical control design," *International Journal of Hybrid Systems*, vol. 2, no. 3, pp. 221–234, Sept. 2002.

[8] R. Alur, J. Esposito, M. Kim, V. Kumar, and I. Lee, "Formal modeling and analysis of hybrid systems: A case study in multi-robot coordination," in *LNCS 1708*, J. Wing, J. Woodcock, and J. Davies, Eds. Berlin Heidelberg: Springer-Verlag, 1999, vol. I, pp. 212–232.

[9] R. Alur and D. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, no. 2, pp. 183–235, 1994.

[10] K. Larsen, P. Pettersson, and W. Yi, "UPPAAL in a nutshell," *International Journal on Software Tools for Technology Transfer*, vol. 1-2, pp. 134–152, 1997.

[11] R. Alur, C. Courcoubetis, N. Halbwachs, T. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine, "The algorithmic analysis of hybrid systems," *Theoretical Computer Science*, pp. 3–34, 1995.

[12] R. Alur, T. Henzinger, G. Lafferriere, and G. Pappas, "Discrete abstractions of hybrid systems," in *Proc. of the IEEE*, vol. 88, no. 7, July 2000, pp. 971–984.

[13] K. Larsen, P. Pettersson, and W. Yi, "Model-checking for real-time systems," in *Proc. of the 10th International Conf. on Fundamentals of Computation Theory*, Dresden, Germany, Aug. 1995, pp. 62–88.

[14] E. Emerson, "Temporal and modal logic," in *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed. MIT Press, 1990, vol. B, ch. 16, pp. 995–1072.