Defence Science Journal, Vol. 60, No. 1, January 2010, pp. 23-38

REVIEW PAPER

Temporal Logic Motion Planning

Motlatsi Seotsanyana

Council for Scientific and Industrial Research, Pretoria, South Africa.
E-mail: mseotsanyana@csir.co.za

ABSTRACT

In this paper, a critical review on temporal logic motion planning is presented. The review paper aims to address the following problems: (a) In a realistic situation, the motion planning problem is carried out in real-time, in a dynamic, uncertain and ever-changing environment, and (b) The accomplishment of high-level specification tasks which are more than just the traditional planning problem (i.e., start at initial state A and go to the goal state B) are considered. The use of theory of computation and formal methods, tools and techniques present a promising direction of research in solving motion planning problems that are influenced by high-level specification of complex tasks. The review, therefore, focuses only on those papers that use the aforementioned tools and techniques to solve a motion planning problem. A proposed robust platform that deals with the complexity of more expressive temporal logics is also presented.

Keywords: Motion planning, temporal logic, real-time, formal languages, automata, mobile agents

1. INTRODUCTION

The ever-increasing reliance of society on robotic applications has led to the need for highly reliable and safe robotic applications. There are a number of areas where these applications perform critical functions which include: transport-related applications such as intralogistics, automated parking garages, and autonomous vehicles; the mining-related applications such as automated mine vehicles, and mine sensing; the defence force-related applications such as autonomous vehicles; the hospital-related applications such as surgical procedures, etc. In these areas, failure of a robotic application may result in more than just a mere inconvenience, such as incorrect information by a robotic receptionist, loss of time, or may even worse cause catastrophic loss of human life in the case of mining automation, surgical procedures, etc. It is clear that the development of these applications requires a higher level of attention and it is also clear that the need for these applications will continue to grow. In all these areas, real-time motion planning is critical and the development of real-time motion planning algorithms requires a systematic approach, i.e., the use of accurate state estimators, theory of computation tools and formal methods.

Unfortunately, the precise state of a mobile vehicle is not always observable and to maintain an uncertain state estimate over the states of the vehicle, state estimators are used. State estimators refer to the techniques that are used to determine the values of the unknown quantities from one or more observations and these include: Kalman filter, extended Kalman filter, etc. These unknown quantities are due to the use of sensory data to obtain measurements of quantities of interest. Theory of computation, on the other hand, refers to the use of mathematical models to computationally deal with whether a problem is solvable, and if it is, how can it be handled efficiently? Formal methods refer to the use of mathematical techniques for the specification, development, and verification of software and hardware systems. The examples include: model checking and theorem proving.

Real-time motion planning of autonomous mobile vehicles is a complex and challenging problem and this is mainly due to inherently unreliable, continuous, and dynamic environments in which they operate. The ability of a motion planner to generate accurate and safe trajectories despite these conditions is fundamental to autonomous mobile vehicles to perform their tasks effectively and reliably. The promising approach to generate accurate and safe trajectories (control strategies) in motion planning involves the use of theory of computation and formal methods techniques and tools[1]. This paper therefore presents a review of motion planning papers that incorporate theory of computation tools such as formal languages, automata and formal methods techniques and tools such as model checking, temporal logic, etc., to develop motion planning frameworks and algorithms.

Belta[1], *et al.* refer to the aforementioned approach as symbolic motion planning. This is a top-down approach that allows an automatic generation of trajectories (control strategies) from high-level specifications expressed in regular languages, automata or temporal logics.

2. MOTIVATION

An autonomous mobile vehicle refers to a vehicle (or car) that drives entirely on its own without human driver and remote control. Various sensors are used to provide the vehicle with an internal world model, i.e., the representation of surrounding environment and positioning system of the

Received 22 September 2009

vehicle within the environment. The introduction of these vehicles provides a number of benefits to society: the reduction of car accidents, reduction (or eradication) of traffic congestions, and also have a direct impact on the areas such as mining, farming, health, constructions, etc.

The document prepared by Road Traffic Management Corporation[2] (RTMC) in South Africa reports the alarming road accidents despite the steady month-on-month decrease in the number of fatal crashes and fatalities since July 2006. From 1 April 2007 to March 2008, the total number of vehicles involved in fatal crashes was 15, 172 and the total cost of fatal crashes was approximately R 113.27 billion. The introduction of autonomous mobile vehicle has a potential to significantly reduce these road accidents and the corresponding costs incurred. In addition to safety, autonomous mobile vehicle can also bring a number of benefits to communities. That is, they can drive home human drivers (or passengers) who can't drive themselves, this includes: grand parents, school children, intoxicated drivers, etc.

Despite the fact that autonomous farming and construction vehicles have reached maturity, to some extend, autonomous cars on public roads are far from being commercialised because of the problems still facing this area. Public roads are inherently dynamic and uncertain, and accounting for moving obstacles where their intentions are not well-known in advance, is a challenging problem. However, the need for autonomous vehicle is considerable, especially in the mining industry where there are hazardous places for human beings.

The main challenge of autonomous mobile vehicles is to develop computationally efficient frameworks and algorithms that allow the interaction of these vehicles with ordinary people to accomplish their tasks. These tasks may be expressed in a human-like languages such as temporal logics to allow high-level specifications such as 'start at stop A, go to stop B and wait for 20 min, and then go to stop C, wait for 30 min before going to stop A again'. This approach will enable mobile robots to solve a large number of practical problems.

3. PROBLEM DEFINITION

In robotics, the current state of a robot includes both its physical environment and configuration, which are referred to as a state space. State space captures all possible situations that could arise—the state could, for example, represent the position and orientation of a robot, the locations of an obstacle, etc. In real-time motion planning, the current state is very important because it makes it possible to generate plans at real-time, and thus makes the task of designing, developing, and formally verifying the correctness of motion planning algorithms possible. Unfortunately, it is virtually impossible for a mobile robot to precisely know its state[3] and an appropriate approach—probably the only one—is to estimate its state from information provided by sensors, lasers, global positioning system (GPS), etc.

Information sourced from these devices poses a number of challenges for designing and developing safe and reliable real-time motion planning algorithms for autonomous robots. These challenges include: uncertainty due to the failure of a device, unreliability due to the noisy data, unsafe if these robots operate in hazardous environments such as mining industry, unavailability of data due to the loss of connection to the GPS, and so on.

Motion planning architectures mostly consist of three components: The world which represents robot's physical environment and configuration, the estimation techniques to filter sensory data, and motion planning. Motion planning is the focus of this paper and takes two inputs: the world model and formal specification, and produces one output (i.e., control strategies). The world model is the representation of ever-changing environment while formal specification refers to the formalised high-level specifications of static environment and tasks to be undertaken by the robot. Motion planning may consist of a number of levels (e.g., as presented by Belta[1], *et al.,* to eventually output control strategies to the robot.

Papers reviewed in this study are only those which solve some or all of the following problems:

- The motion planning problem is carried out in real-time, in a dynamic, uncertain and ever-changing environment.
- The accomplishments of high-level specification tasks, which are more than just the traditional planning problem (i.e., start at initial state A and go to the goal state B), are considered.

The high-level specification tasks such as 'start at stop A, go to stop B and wait for 20 min, and then go to stop C, wait for 30 min before going to stop A again' can be naturally translated into temporal formulas such as linear temporal logic (LTL), computational tree logic (CTL), or timed computation tree logic (TCTL). Hence, in section V, only motion planning techniques that include tools of theory of computation and formal methods are considered.

4. TEMPORAL LOGIC

The papers reviewed in Section V use temporal logics for specifying and designing motion planning algorithms. Therefore, an overview of temporal logics is presented. These logics have been used to precisely describe the properties of concurrent systems (such as safety and liveness properties) and were first introduced by Pnueli around 1977 for the specification and verification of computer systems. The mostly widely-used two types of temporal logics are LTL and CTL. More information on these logics is presented by Clarke[4], and Seotsanyana[5].

4.1 Linear Temporal Logic

The linear temporal logic (LTL) is a modal temporal logic with modalities referring to time. In LTL, formulas are encoded about the future of paths (or runs) such that a condition will eventually be true, that a condition will be true until another fact becomes true, etc.

4.1.1 Linear Temporal Logic Syntax

The syntax of LTL is defined in terms of atomic propositions,

logical connectives, and temporal operators. Atomic propositions are the most simple statements that can be made about the system in question and thus take the value true or false. Examples of atomic propositions are-the door is closed, x is less than 2, etc. Atomic propositions can be represented by alphabetic symbols such as *p* and *q*. The set of atomic propositions is referred to as AP. The boolean operators that are used in the syntax of LTL are $\vee, \wedge, \neg, \Rightarrow$ and $\Leftrightarrow$; in addition, there are several temporal operators, with the following meanings:

- $\Box$ denotes 'always',
- $\Diamond$ denotes 'eventually',
- *U* denotes 'strong until', and
- *X* denotes 'next'.

The structure of a formula of propositional LTL is given by the following grammar expressed in Backus-Naur form (BNF) notation:

$$\alpha ::= \mathrm{p} \mid \neg\alpha \mid \alpha \vee \beta \mid \mathrm{X}\alpha \mid \alpha \mathrm{U} \beta$$

The operators $\wedge, \Rightarrow, \Leftrightarrow$, *true*, false, $\Diamond$ and $\Box$ which are not mentioned in this syntax, can be thought of merely as abbreviations by using the following rules:

$$\alpha \wedge \beta \equiv \neg(\neg\alpha \vee \neg\beta) \quad \alpha \Rightarrow \beta \equiv \neg\alpha \vee \beta$$

$$\alpha \Leftrightarrow \beta \equiv (\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha) \quad true \equiv \neg\alpha \vee \alpha$$

$$false \equiv \neg true \quad \Diamond\alpha \equiv true\ U\ \alpha$$

$$\Box\alpha \equiv \neg\Diamond\neg\alpha$$

4.1.2 Linear Temporal Logic Semantics

The syntax defines how LTL formulas are constructed, but does not provide an interpretation of the formulas or operators. Formally, LTL formulas are interpreted in terms of a model defined as a triple *M=(S,R, Label)*,where

- *S* is a non-empty countable set of states,
- $R{:}S \rightarrow S$, is a function which assigns to each $s \in S$ a unique successor $R(s)$, and
- $Label{:}S \rightarrow 2^{AP}$, is a function which assigns to each state $s \in S$ the atomic propositions *Label(s)* that are valid in *s*.

The meaning of LTL formulas are defined in terms of a satisfaction relation, denoted by $\models$, between a model *M*, a states $s \in S$ and the formulas α and β. Therefore *M*, s $\models \alpha$ if only if α is valid in the state s of the model *M*. If it is understood from the context, *M* is dropped and the satisfaction relation is mathematically defined as follows:

$$\begin{array}{lll} s \models p & \text{iff} & p \in Label(s) \\ s \models \neg\alpha & \text{iff} & \neg(s \models \alpha) \\ s \models \alpha \vee \beta & \text{iff} & (s \models \alpha) \vee (s \models \beta) \\ s \models X\alpha & \text{iff} & R(s) \models \alpha \\ s \models \alpha U \beta & \text{iff} & (\exists j \geq 0 : R^j(s) \models \beta) \wedge \\ & & (\forall 0 \leq k < j : R^k(s) \models \alpha) \end{array}$$

Here, R^i is used to denote *i* applications of the function *R*. For example, $R^3(S)$ is the same as $R(R(R(s)))$ The formal interpretation of the other connectives, *true, false*, $\wedge, \Rightarrow$, $\Diamond$ and $\Box$ can be derived in a similar way from the definitions above.

4.2 Computation Tree Logic

The computation tree logic (CTL) is based on the concept that for each state there are many possible successors, unlike in LTL which is based on a model where each state *s* has only one successor *s'*. Because of this branching notion of time, CTL is classified as a branching temporal logic. The interpretation of CTL is therefore based on a tree rather than a sequence as in LTL.

4.2.1 Computation Tree Logic Syntax

The formulas of CTL consist of atomic propositions, standard boolean connectives of propositional logic, and temporal operators. Each temporal operator is composed of two parts, a path quantifier (universal $\forall$ or existential $\exists$) followed by a temporal modality ($\Diamond$ and $\Box$ *X*, *U*). Note that some authors use *G* and *F* for $\forall$ and $\exists$, respectively. The temporal modalities have the same meanings as in Section 4.1. The syntax is given by the BNF:

$$\alpha ::= p \mid \neg\alpha \mid \alpha \vee \beta \mid \alpha \wedge \beta \mid \exists X\alpha \mid \exists[\alpha U \beta] \mid \forall[\alpha U \beta]$$

4.2.2 Computation Tree Logic Semantics

CTL semantics slightly differs from that one of LTL, i.e., the notion of a sequence is replaced by a notion of a *tree*. The interpretation of CTL is defined by a satisfaction relation $\models$between a model *M*, one of its states *s* and some formula. Let $AP=\{p,q,r\}$ be a set of atomic propositions, $M=(S,R\ Label)$ be CTL-Model, $s \in S$, α and β be CTL-formulas. In order to define the satisfaction relation ($\models$), the following definitions are first given:

- A path is an infinite sequence of states $s_0, s_1, s_2, \ldots$ such that $(s_i, s_{i+1}) \in R$
- Let $\rho \in S^w$ denotes a path. For $i \geq 0, \rho[i]$ denotes the $(i+1)^{th}$ element of ρ, i.e., if $\rho = s_0, s_1, s_2, \ldots$ then $\rho[i] = s_i$
- $P_M(s) = \{\rho \in S^w \mid \rho[0] = s\}$ is a set of paths starting at *s*.

Just like in LTL if it is understood from the context, *M* can be dropped in the satisfaction relation $\models$ defined as follows:

$$\begin{array}{lll} s \models p & \text{iff} & p \in Label(s) \\ s \models \neg\alpha & \text{iff} & \neg(s \models \alpha) \\ s \models \alpha \vee \beta & \text{iff} & (s \models \alpha) \vee (s \models \beta) \\ s \models \exists X\alpha & \text{iff} & \exists \rho \in P(s) : \rho[1] \models \alpha \\ s \models \exists[\alpha U \beta] & \text{iff} & \exists \rho \in P(s) : \exists j \geq 0 : (\rho[j] \models \\ & & \beta \wedge \forall 0 \leq k < j : \rho[k] \models \alpha) \\ s \models \forall[\alpha U \beta] & \text{iff} & \forall \rho \in P(s) : \exists j \geq 0 : (\rho[j] \models \\ & & \beta \wedge \forall 0 \leq k < j : \rho[k] \models \alpha) \end{array}$$

4.3 Timed Computation Tree Logic

The temporal LTL and CTL, focus on the temporal order of events and do not explicitly state the actual time taken by these events. Time-critical robotic systems necessitate the consideration of quantitative time between the occurrence of events, i.e., the correctness of most robotic systems do not only depend on the functional requirements, but also on the time requirements. In this section, the syntax and semantics of timed computation tree logic (TCTL) is presented. But first, an overview of timed automata is given.

4.3.1 Timed Automata Syntax

Finite-state real-time systems are modelled with timed automata. A timed automaton is a standard finite-state automaton extended with a set of non-negative real-valued clock variables (or just clocks in short). Clocks are assumed to proceed at the same rate to measure the time elapsed since they were last reset. In order to formally define a timed automaton, clocks and clock constraints are first defined as follows:

- A clock is a variable ranging over $\mathbb{R}^+$ (where $\mathbb{R}^+$ represents non-negative real numbers)
- For set *C* of clocks with $x, y, z \in C$, a clock constraint α over C is defined by $\alpha ::= x \prec c \mid x - y \prec c \mid \neg\alpha \mid (\alpha \wedge \alpha)$, where and $\prec \in \{<, \leq\}$
- $\Psi(C)$ is the set of all possible clock constraints.

Clocks are defined to range over the non-negative real numbers, i.e., $x, y, z \in \mathbb{R}^+$. A state of a timed automaton consists of a location and values of clocks. Clock constraints are used to label the edges of a timed automaton and represent guards that are used to either enable or block transitions between locations. Clock constraints are also used to label locations and such constraints are then 'invariants' that limit the amount of time to be spent in a location. Formally, a timed automaton *A* over set of actions Σ, set of atomic propositions *AP* and set of clocks *C* is defined as a tuple *(L,l_0,l,Label)* where:

- L is a non-empty set of locations with the initial location $l_0 \in L$.
- $E \subseteq L \times \Psi(C) \times \Sigma \times 2^C \times L$ corresponds to a set of edges. $(l, g, a, r, l') \in E$ represents an edge from location l to location l' with clock constraint *g* (also known as enabling condition of the edge or guard) action *a* to be performed and the set of clocks r to be reset.
- $I : L \to \Psi(C)$ is a function which assigns a clock constraint (i.e., an invariant) for each location.
- $Label : L \to 2^{AP}$ is a function which assigns to each locations $l \in L$ set of atomic propositions that hold in the location.

4.3.2 Timed Automaton Semantics

The interpretation of a timed automaton is defined in terms of an infinite transition system and to formally define the semantics of the timed automaton, the 'clock assignment' function and state of a timed automaton are defined as follows:

- A clock valuation (clock assignment) *u* for the set of clocks *C* is a function $u : C \to \mathbb{R}^+$ assigning each clock $x \in C$ its value $u(x)$. Let the set of all clock valuations over *C* be denoted by *V(C)*. The clock evaluation has the following characteristics:
  - For $u \in V(C)$ and $d \in \mathbb{R}^+$, clock valuation $u + d$ over *C* means that all clocks are increased by *d*, that is $u(x) + d$ for all $x \in C$.
  - For $C' \subseteq C, u[C' \to 0]$ means that all the clocks in C' are assigned to zero, that is, all assigned and zero clocks in C' are reset, so that $u[C' \to 0](x) = 0$ for all $x \in C'$ and $u[C' \to 0](x) = u(x)$ for all $x \notin C'$. If C' is the singleton set $\{z\}$, just $u[z \to 0]$ shall be written.
  - For a given clock valuation $u \in V(C)$ and a clock constraint $\alpha \in \Psi(C)$, $\alpha(u)$ is a boolean value stating whether or not α is satisfied or not.
- A state is a pair (l, u) where l is a location of an automaton *A* and u is a clock valuation over *C*.

The operational semantics of a timed automaton *A=(L,E,I,Label)* over the clock set *C* is therefore defined by an infinite state transition system $M_A = (S, s_0, \to, Label)$ where:

- $S = L \times V(C)$ is the set of states,
- s_0 is the initial state of *A* (l_0, u_0),
- $\to$ is the transition relation with its members defined by the following two rules:
  - *action transition*: $(l,u) \xrightarrow{a} (l',u')$ if there is an edge $(l \xrightarrow{g,a,r} l')$ such that $g(u)$ holds and $u' = u[r \to 0]$, and $inv(u')$ holds for each $inv \in I(l')$;
  - *delay transition:* $(l,u) \xrightarrow{d} (l,u')$ if, for $d \in \mathbb{R}^+$, $u' = u + d$ and $inv(u + d')$ holds for all $d' \leq d$ and $inv \in I(l)$.
- $Label : S \to 2^{AP}$ is atomic proposition function extended from $Label : L \to 2^{AP}$ simply by *Label(l,u) = Label(l).*

4.3.3 Timed Computation Tree Logic Syntax

The syntax of TCTL is based on the syntax of CTL, extended with clock constraints. To clearly define the syntax, the following definitions are given:

- A *path* is an infinite sequence $s_0 a_0, s_1 a_1, \ldots$ states alternated by transition labels such that $s_i \xrightarrow{a_i} s_{i+1}$ for all $i \geq 0$, where a_i is either (g, a, r) or *d*.
- Let $\rho \in S^w$ denotes a path. For $i \geq 0, \rho[i]$ denotes the $(i+1)^{th}$ element of ρ (Section 4.2.2).
- $P_M(s) = \{\rho \in S^w \mid \rho[0] = s\}$ is a set of paths starting at *s* (Section 4.2.2).
- A *position* of a path is a pair *(i, d)* such that *d* equals 0 if $a_i = (g, a, r)$, and equal a_i otherwise.
- Let $Pos(\rho)$ be the set of positions in ρ. For convenience the state $(l_i, v_i + d)$ can also be written as $\rho(i, d)$.
- A total order of positions is defined by:
  - $(i,d) << (j,d')$ if and only if $(i < j) \vee (i = j \wedge d \leq d')$.

- Path ρ is called *time-divergent* if $\lim_{i\to\infty}\Delta(\rho,i)=\infty$, where $\Delta(\rho,i)$ denote the time elapsed from s_0 to s_i, i.e., $\Delta(\rho,0)=0$

$$\Delta(\rho,i)=\Delta(\rho,i)+\begin{cases}0 & if \quad a_i=(g,a,r)\\ a_i & if \quad a_i\in\Re\end{cases},$$

- Let $P_M^{\infty}(s)=\{\rho\in S^w \mid \rho[0]=s\}$ denote the set of time-divergent paths starting at *s*.

Let $p\in AP$ and *D* be a non-empty set of clocks that is disjoint from the clocks of *A* (i.e., *D* is the set of clocks of the TCTL-formulas and $C\cap D=\varnothing$), $z\in D$ and $\alpha\in\Psi(C\cap D)$. The TCTL-formulas are then defined by the following BNF:

$$\beta ::= p \mid \alpha \mid \neg\beta \mid \beta\vee\beta \mid z\ in\ \beta \mid \exists[\beta U\beta] \mid \forall[\beta U\beta]$$

A clock constraint α is defined over formula clocks and timed automaton clocks and thus allows comparison of both formula and timed automaton clocks. Clock *z* is known as a freeze identifier and bounds formula clocks in β. For instance, $\forall[\beta U_{\le 4}\phi]$ can be defined as $z\ in\ \forall[(\beta\wedge z\le 4)U\phi]$.

4.3.4 Timed Computation Tree Logic Semantics

For, $p\in AP$, $\alpha\in\Psi(C\cap D)$ is a clock constraint over $C\cup D$, model $M=(S,\to,L)$ is an infinite transition system, $s\in S$, $w\in V(D)$, and ψ,ϕ are TCTL-formulas. The satisfaction relation ⊨ is defined as follows:

$s,w\models p$	iff	$p\in L(s)$
$s,w\models\alpha$	iff	$v\cup w\models\alpha$
$s,w\models\neg\phi$	iff	$\neg(s,w\models\phi)$
$s,w\models\phi\vee\psi$	iff	$(s,w\models\phi)\vee(s,w\models\psi)$
$s,w\models z\ in\ \phi$	iff	$s,w[z\to 0]\models\phi$
$s,w\models\exists[\phi U\psi]$	iff	$\exists\rho\in P_M^{\infty}(s):\exists(i,d)\in Pos(\rho):(\rho(i,d),w+\Delta(\rho,i)\models\psi\wedge\forall(j,d')<<(i,d):\rho(j,d'),w+\Delta(\rho,j)\models\phi\vee\psi))$
$s,w\models\forall[\phi U\psi]$	iff	$\forall\rho\in P_M^{\infty}(s):\exists(i,d)\in Pos(\rho):(\rho(i,d),w+\Delta(\rho,i)\models\psi\wedge\forall(j,d')<<(i,d):\rho(j,d'),w+\Delta(\rho,j)\models\phi\vee\psi))$

5. CRITICAL REVIEW

The papers under review focus on the use of temporal logics in robot motion planning. Table 1 outlines their titles and corresponding acronyms, which are henceforth used to refer to them. In each paper, the explanation of the problem and the method used are presented. The corresponding results and critical analysis are discussed in Section 6.

Table 1. Reviewed papers

Acronym	Paper title
DEMTL	Discrete event models + temporal logic = Supervisory controller: automatic synthesis of locomotion controllers[6]
MRPTA	Multi-robot planning: A timed automata approach[7]
SPCRM	Symbolic planning and control of robot motion[1]
TLMPR	Temporal logic motion planning for mobile robot[8]
STLMP	Where's waldo? sensor-based temporal logic motion planning[9]
ASMMT	Automatic synthesis of multi-agent motion tasks based on LTL specification[10]

5.1 Discrete Event Modal + Temporal Logic

Antoniothi and mishra[6] develop a robust compiler program—similar to silicon compilers—to synthesize controller programs of different robotic applications and manufacturing tasks, based on discrete event systems (DES) theory[11], Petri Nets[12,13] and temporal logic[14]. The synthesiser (or compiler) takes two inputs: (1) a model of a robotic problem and (2) a set of high-level specifications expressed in temporal logic and outputs a synthesised controller that produces control commands.

The running example for testing and simulation of the synthesiser is the walking machine problem with four legs. The model of the machine is divided into two layers: discrete and continuous. The discrete layer refers to the scheme used to synchronise states of the legs and this is modelled with a finite state machine (FSM) (Fig. 1(b)). The six states shown in Fig. 1(b) correspond to different movements of a leg. Each leg has three links, as shown in Fig. 1(a).

The continuous layer is represented by different kinematic equations at each state of FSM. Each leg is represented by a FSM and a synchronised product FSM of the four leg FSMs has 1296 states and 5184 transitions. The transitions between these states are governed by the sensory data and only the position information is used for simulation purposes. The position p_r represents the rear position whereas p_f represents the front position of the legs. The system may allow the steps that are longer than the leg and this is restricted by a graph traversal which maintains the minimum and maximum of $\Delta(p_{feet})=\mid p_f-p_r\mid$. For example, the equation $\Delta(p_{feet})[Load_1,Driver_2]=\Delta(p_{feet})[\mathrm{Re\,cov}\,er_1,Driver_2]+\frac{1}{2}step$ assumes that the rear leg is moved a very small step and the equation

$$\Delta(p_{feet})[Load_1,Driver_2]=\Delta(p_{feet})[\mathrm{Re\,cov}\,er_1,Driver_2]+2\,step$$

assumes a full step distance. The desired constraint is that $\Delta(p_{feet})<l$ where *l* is derived from the mechanics of the Walking Machine.

Antoniothi and Mishra[6] followed a number of steps for the process of the controller synthesis: (a) The modification of the standard DES and the use of temporal logic for specification and verification properties; (b) A modified model checker is used to mark undesired states of the

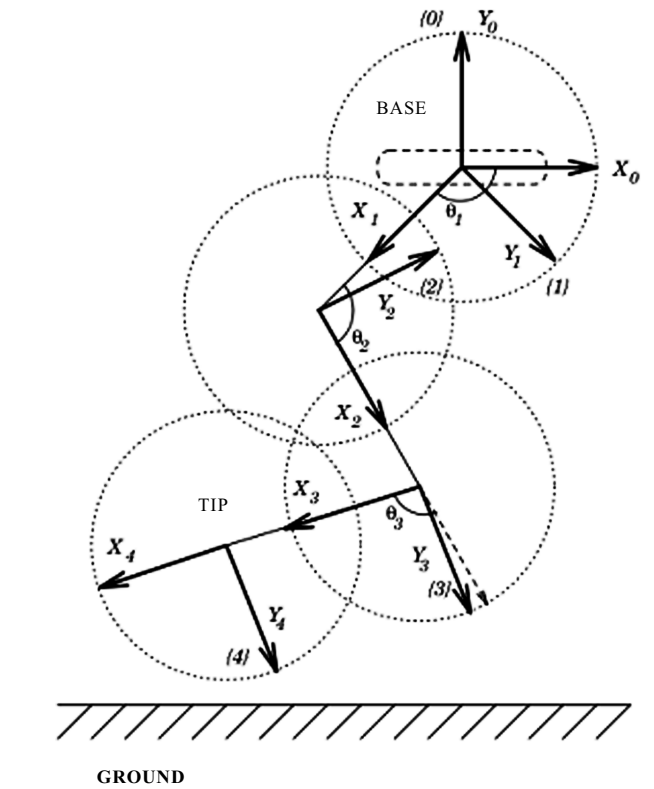


Figure 1. Leg assignment[6]: (a) three links of each leg and (b) the six states correspond to different movement of a leg).

machine. For example, the system should not be in a state where both legs of a train are both recovering, driving, or slipping (i.e., these states should be avoided). The CTL formula used to mark these states is:

$$AG(\neg state([Driver_1, Driver_2]) \wedge$$
$$\neg state([Driver_1, Driver_2]) \wedge$$
$$\neg state([Slipping_1, Slipping_2])),$$

and (c) the modified model checker is also used to verify the supervisory synthesiser to ensure that it maintains the required behaviour of the machine and this is the original intention of the authors, to verify the correctness of the synthesiser itself. More properties are also verified and the results are discussed in Section 6.

5.2 Multi-robot Planning: A Timed Automata Approach

Quothrup,[7] *et al.* discussed the use of theory of timed automata to model a motion planning problem in a multi-robot environment. Authors start by highlighting applications of multi-robot systems and various methods used to model communication among robots. These approaches include:

(a) threaded petri nets,
(b) use of a plan-merge paradigm,
(c) a distributed negotiation mechanism,
(d) hybrid control to action coordination and collision avoidance and
(e) formal hybrid to modelling and coordination.

The environment of a planar is assumed and robots can only move either horizontally or vertically. Properties of interest are expressed in CTL. The system is modelled with a matrix and three-template automata: the matrix models the workspace, an obstacle template models static obstacles on the planar grid, a robot template models a robot and a control template models concurrent communication between robots. The following five properties are verified on the system:

(1) collision avoidance,
(2) bounded movements,
(3) reachability
(4) reachability with time requirement, and
(5) reachability with step requirement.

To model multiple robots using timed automata, three models are developed: environment (i.e., workspace with obstacles), robots, and control.

Robots are restricted to operate in a planar environment $X \in \mathbb{R}^2$ where position, $x_i \in X$, in the planar is denoted by $x_i = [x_{i1}\ x_{i2}]$. The X is divided into partitions of disjoint cells with $\varepsilon \in \mathbb{R}^+$ (i.e., grid size) resolution. The following defines the planar environment in relation to its partitions:

$$X = \bigcup_U C^{\varepsilon}(z_i)$$

where, $z_i = [z_{i1}\ z_{i2}]$, $z_i \in \mathbb{Z}^2$ and U is the number of cells that cover the planar X. Then each cell in a partition is defined as follows:

$$C^{\varepsilon}(z_i) = \{x_j \in \mathbb{R}^2 : z_{i1} - \tfrac{\varepsilon}{2} < x_{j1} \le z_{i1} + \tfrac{\varepsilon}{2} \wedge$$
$$z_{i2} - \tfrac{\varepsilon}{2} < x_{j2} \le z_{i2} + \tfrac{\varepsilon}{2}\}.$$

Then X is divided into $U = (S+1)\ (T+1)$ number of cells, where $(S+1)$ is the number of columns and $(T+1)$is the number of rows. Static obstacles maybe present on the environment by occupying a cell on the grid. The workspace (i.e., a free space where the robot moves) is:

$$W = X \setminus \bigcup_M O(z_i)$$

where M denotes the number of static obstacles and $O(Z_i)$ is a location of an obstacle located at $\mathbb{Z}^2$. Therefore, the workspace W is shared among robots moving on the planar and with the proper size of ε the robots can be synchronised to avoid collision with each other. In the model checker Uppaal[15], the environment is defined as a

two dimensional array of integers: int[0,1] part$X$[$hSize$][$vSize$], where $hSize$ and $vSize$ are horizontal and vertical sizes, respectively. The occupied cells are assigned to zero (part$X$[$hSize$][$vSize$] = 0), while the free cells are assigned to one (part$X$[$hSize$][$vSize$]=1).

The robot is modelled by the following timed automaton $A_R = (L, l_{init}, E, I, V)$ where:

- $L = \{l_{init}, l_{stop}, l_{mr}, l_{ml}, l_{mu}, l_{md}\}$ is the set control locations,
- $l_{init} \in L$ is the initial location,
- $E \in L \times \mathrm{B}(C) \times Act \times 2^C \times L$ is the set of edges,
  $e_0 = (l_{init}, l_{init})$
  $e_1 = (l_{init}, l_{stop})$
  $e_2 = (l_{stop}, moveRight?, l_{init})$
  $e_3 = (l_{mr}, cMin < c < cMax, c = 0, l_{stop})$
  etc.
- $I : L \to \mathrm{B}(C)$ is the function that assigns invariant condition to locations.

The control process is modelled by the following automaton $A_C = (L, l_{loop})$. Where, L is the set of control locations, and l_{loop} is the only control location for the process.

There are two scenarios presented in this paper, the first one involves three robots that need to traverse through the door while avoiding other robots and the obstacles on the planar. The second situation involves two robots that have to change positions in a maze. The following properties are verified with Uppaal[15]:

a) *Safety Properties:*
   - For all control trajectories the robots never collide, after they start to move.
   - For all control trajectories the robots never move outside workspace.

b) *Liveness properties:*
   - Does there exist any control trajectory where the robots eventually reach their goal positions?
   - Does there exist a control trajectory where the robots eventually reach their goal positions?
   - Does there exist a control trajectory where the robots eventually reach their goal within 10 step movements?

The results of these properties are discussed in Section 6.

5.3 Symbolic Planning and Control of Robot Motion

Belta[1], *et al.* have stated the challenges of robot motion planning are stated. These challenges involve the development of computationally efficient algorithms which takes into account: (i) the constraints of the robot, and (ii) the complexity of environment while at the same time facilitating a detailed high-level specification of tasks. A normal basic motion planning problem is usually stated as a mobile robot moving from one initial location to a goal location while at the same time avoiding obstacles. The framework used to solve this problem is usually divided into three levels: The first level (called specification level) is about dividing the configuration state space into cells and these are represented by a graph. The second level (called execution level) finds the shortest path that avoids obstacles from the initial state to the goal state. And the third level (called implementation level) generates a reference trajectory and controllers are developed to follow the trajectory. In this paper, authors use theory of computation and formal methods tools to represent specification tasks, robot constraints and environment and they also coined the term symbolic to refer to the use of these tools. Symbolic motion planning can be easily incorporated into the three aforementioned levels of the framework. This paper outlines the challenges of incorporating the tools of theory of computation and formal methods in motion planning and Section 6 discusses the detailed analysis of these challenges.

5.4 Temporal Logic and Motion Planning for Mobile Robot

Fainkos,[8] *et al.* presented a novel approach for linking discrete AI planning with motion planning. The authors state that formal formulation of specification such as sequencing, etc., in motion planning provides new challenges such as introducing computationally efficient methods that deal with the complexity of these approaches. These properties can be expressed in temporal logics such as LTL and CTL. Authors differentiate their approach from previous related approaches that used model checking to generate discrete paths that satisfy temporal logic specifications. This previous research has resulted in the tools like: MBP[16], TLPLAN[17], and UMOP[18]. The aim of the paper is to generate continuous trajectories which satisfy temporal logic formulas. This is achieved through the following steps:

- the decomposition of workspace into cells[19,20],
- the use model checking NuSMV[21] to generate plans for discrete motion planning that satisfy LTL properties, and
- the generation of continuous trajectories that satisfy the specified LTL properties.

A robot was modelled that was operated in a polygonal environment P and the motion of the robot has been expressed as

$$\dot{x}(t) = u(t) \qquad x(t) \in P \subseteq \mathbb{R}^2 \qquad u(t) \in U \subseteq \mathbb{R}^2 \qquad (1)$$

where $x(t)$ is the position of the robot at time t and $u(t)$ is the control input. The objects of interest such as rooms, corridors, etc., are atomic propositions represented by a set $\Pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$ and the observation map, which is associated with Eqn. (1), is defined as

$$h_C : P \to \Pi \qquad (2)$$

and this is an observation map that takes continuous states of the robot and maps it to the set of propositions. The proposition is a convex set of the form:

$$P_i = \{x \in \mathbb{R}^2 | \bigwedge_{1 \le k \le m} a_k^T x + b_k \le 0, a_k \in \mathbb{R}^2, b_k \in \mathbb{R}\}$$

The relationship between the observation map and the set of atomic propositions (objects of interests) is defined as follows: if and only if x belongs to some related set

Given robot model (1), observation map (2), initial state $x(0) \in P$, and LTL formula φ, the problem was to construct control input $u(t)$ such that the resulting robot

trajectories satisfy the state $x(t)$. To solve this problem, let $x[t]$ defines the robot trajectories starting at state $x(t)$. The meaning of LTL-formula φ is defined in terms of a satisfaction relation, denoted by $\models_C$, over continuous robot trajectories $x[t]$. Then $x[t] \models_C \varphi$ states that the trajectory $x[t]$ starting at $x(t)$ satisfies the formula φ. Other LTL formulas can be constructed recursively.

The process of generating continuous robot trajectories that satisfy LTL formulas involves three steps:

- Discrete abstraction of robot motion,
- Temporal logic planning using model checking, and
- Continuous Implementation of discrete plan.

Step 1

The first step is to partition the workspace *P* into triangles and two reasons were related for their choice of the partitioning algorithm: First, there exist many efficient triangulation algorithms[22] and Second, the controller used in this paper is proved to be efficiently computable on triangles[20]. Therefore, the map $T : P \to Q$ sends states $x \in P$ to the finite set of triangles $Q = \{q_1, \ldots, q_n\}$. Given a partitioned workspace of *P*, the robot motions are defined by the following transition system

$$D = (Q, q_0, \to_D, h_D) \tag{3}$$

where, Q is a set states, $q_0 \in Q$ is a cell containing the initial state $x(0) \in P$, $\to_D \in Q \times Q$ is a transition relation, defined as $q_i \to_D q_j$ if and only if the triangles q_i and q_j are topologically adjacent to each other, and $h_D : Q \to \Pi$ is an observation map, where $h_D(q) = \pi$ if there is a state $x \in T^{-1}(q)$ such that $h_C(x) = \pi$. And $T^{-1}(q)$ contains all the states $x \in P$ labelled by q. The trajectory p of D is defined as a sequence $p[i] = p_i \to_D p_{i+1} \to_D p_{i+2} \ldots$, where $p_i = p(i) \in Q$.

Step 2

In step 2, the model checkers NuSMV[21] and SPIN[23] were used to generate trajectories $p[i]$ of the system D explained in the first step. However, model checking tools are not meant to generate trajectories (i.e., witnesses), but to verify the system and output yes if the property is satisfied or a counterexample if it is not satisfied. To generate trajectories that satisfy the formula φ, authors use counterexample algorithms in these tools. Since these counterexample algorithms produce a computation $p[i]$ that satisfies $\neg\varphi$ (i.e., $p[i] \models \neg\varphi$), so the original formula is negated and verified with one of the model checkers and the counterexample for $p[i] \models \neg(\neg\varphi)$ is generated. This trajectory satisfies $\varphi = \neg(\neg\varphi)$ and is used in the next step to guide the generation of a continuous trajectory.

Step 3

In the third step, to generate continuous trajectories that satisfy the formula φ, the following continuous transition system is defined

$$C = (P, x(0), \to_C, h_C) \tag{4}$$

where, P is a set of polygonals, $x(0) \in T^{-1}(T(x))$ is the initial state, $\to_C \subset P \times P$ is a transition relation, defined as $x \to_C x'$ between states in *P*, if and only if x and x' belong to adjacent triangles. $h_C : P \to \Pi$ is an observation map, where $h_C(x) = \pi$ maps continuous state to areas of interest (i.e., the set Π.)

For the system *C* to implement trajectories that are generated by any of the model checkers, the triangulation of *P* must satisfy bisimulation property[24]. That is, $T : P \to Q$ is called bisimulation if the following conditions hold for $\forall x, y \in P$

- If $T(x) = T(y)$ then $h_C(x) = h_C(y)$ (i.e., observation preserving)
- If $T(x) = T(y)$ then if $x \to_C x'$ then $y \to_C y'$ with $T(x') = T(y')$ (i.e., reachability preserving)

If bisimulation property is satisfied by the triangulation of the environment, the controllers can be designed that satisfy this property. There are a number of frameworks that can be used, including[20,25]. But the authors used the framework[20] and the reason for their choice is due to its computational properties in triangulated environments.

5.5 Sensor-based Temporal Logic Motion Planning

Kress-Gazit,[9] *et al.* have drawn a distinction between two approaches to motion planning: (a) bottom-up and (b) top-down. In bottom-up approach, the emphasis is put on generating control inputs to robot models that take a robot from one configuration space to another, while on the other hand top-down approach focus on finding discrete robot actions to achieve high-level complex tasks— including the interaction of robots in a multi-robot environment, sequencing of temporal actions, etc. High-level task planning and low-level motion planning were not possible until the advent of hybrid systems. Hybrid systems integrate discrete and continuous systems and this has made it possible to integrate high-level task planning and low-level robot motion planning. This new paradigm of hybrid systems has made it possible to have new approaches in robot motion, such as introduced by Fainkos[8,28], *et al.*, and Kloetzer and Belt[26] This paper build on their previous work[8,27], and in this paper their approach introduces two novelties in motion planning. First, the temporal logic used addresses sensor inputs directly and second, the use of fragment temporal logic called general reactivity (GR)[28], that is computationally polynomial. Its cheap complexity does not affect its expressiveness even though there are properties that cannot be addressed in this logic.The goal of this paper is to develop a framework that automatically and verifiably generate controllers that satisfy high-level specification tasks expressed in temporal logic and to achieve this, the following should be defined: the model of the robot, admissible environments, and the desired system specification.

- *Robot model*: Authors assume a robot is operating in a polygonal workspace *P* and the motion of the robot is expressed as

  $$\dot{p}(t) = u(t) \quad p(t) \in P \subseteq \mathbb{R}^2 \qquad u(t) \in U \subseteq \mathbb{R}^2 \tag{5}$$

  where, $p(t)$ is the position of the robot at time t and $u(t)$ is the control input. It blossomed that P is partitioned

into a number of cells $P_1, P_2, \ldots, P_n$ where $P = \bigcup_{i=1}^{n} P_i$ and $P_i \cap P_j = \emptyset$ if $i \neq j$. It is also assumed that the creates a set of propositions $\gamma = \{r_1, r_2, \ldots, r_n\}$. Therefore the proposition r_1 is true if and only if $p \in P_1$, while other propositions are false.

- *Admissible environments*: A robot interacts with its environment through sensors. It is assumed that sensors are binary and *m* sensor variables $\chi = \{x_1, x_2, \ldots, x_m\}$ are not modelled explicitly, instead high-level assumptions are made so that variables can model admissible environments. These admissible environments are modeled in LTL formulas of the form φ_e.
- *System specification*: The desired behaviour of the robot is expressed in LTL formula φ_s and the specifications which can be expressed in LTL include: coverage, sequencing, and avoidance properties.

Given robot model (5), initial state $t(0) \in P$, and some LTL formulae φ, the problem was to construct a control input $u(t)$ such that the resulting robot trajectories satisfy the state $p(t)$. To solve this problem, let $x[t]$ defines the robot trajectories starting at state $x(t)$. The meaning of LTL formula φ is defined in terms of a satisfaction relation, denoted by $\models_C$, over continuous robot trajectories $x[t]$. Then $x[t] \models_C \varphi$ states that the trajectory $x[t]$ starting at $x(t)$ satisfies the formula φ. Other LTL formulas can be constructed recursively.

Given an LTL formula, an automaton that generates an acceptable behaviour by the LTL formula was synthesised. Pneuli and Rosner[29] proved that the synthesis process is doubly exponential. However, the algorithm used in this paper is polynomial $O(n^3)$ time where *n* is the number of valuations for sensor and state variables[28]. The synthesis process is compared with a game played between the robot and the environment. First, the environment makes a transition according to its transition relation and then the robot does the same. If the robot can satisfy the LTL formula φ, no matter what the environment does, the robot is winning, otherwise the environment is winning and the desired behaviour cannot be achieved. Given the following winning condition (i.e., GR(1)) $\phi = \varphi_g^e \rightarrow \varphi_g^s$, the robot is winning if φ_g^s is true or φ_g^e is false. If the robot is winning an automaton that represents desired behaviour is synthesized and it is formally defined as tuple $A = (\chi, Y, Q, q_0, \delta, \gamma)$
where,

- χ is a set of input environment propositions,
- Y is a set of output system propositions,
- $Q \subset \mathbb{N}$ is a set of states,
- $q_0 \in Q$ is the initial state,
- $\delta : Q \times 2^{\chi} \rightarrow 2^{Q}$ is a transition relation, that is, $\delta(q, X) = Q' \subset Q$ where $q \in Q$ is a state and $X \subseteq \chi$ is the subset of sensor propositions that are true, and
- $\gamma : Q \rightarrow 2^{Y}$ is the set of state propositions that are true in state q.

The importance of this automaton is to generate a path that a robot can follow under admissible inputs. Given admissible input sequence $X_1, X_2, X_3, \ldots, X_j \in 2^{\chi}$, the automaton generates a run $\sigma = q_0, q_1, \ldots$. This run σ is interpreted as sequence $y_0 y_1, \ldots$ where $\gamma(q_i) = y_i$ is the label of the i^{th} state. The last step is to use the run σ to influence the continuous behaviour of the robot. There are a number of hybrid controllers presented by connel[19], *et al.,* and Belta Habelts[20] that can drive robots from region to region. In this paper, a controller that satisfies the so-called bisimulation property[24] was chosen and this controller is presented by Corner[19], *et al*.

5.6 Automatic Synthesis of Multi-agent Motion Tasks Based on LTL Specification

Loizou and kyriakopoulus[10] state that there is an increasing interest in the control theory to develop automated controllers that satisfy complex desired requirements. The key issue in this direction is the use of formal specification. Local controllers were synthesised based on specification represented in the form of graphs[30], while using LTL specifications[31]. Some studies use several motion description languages[32]. In this paper, LTL specification is used due to its capability to express properties quantitatively and its similarity to natural languages.

The following situation is assumed to test the methodology (1) there are *m* robots moving in the workspace $W \subset R^2$, (2) each robot $i = 1 \ldots m$ occupies a disk in the workspace $R_i = \{q \in R^2 : \| q - x_i \| \leq r_i\}$, where $x_i \in R^2$ is the centre of the disk and r_i is the radius. The configuration of each robot is presented by x_i and the configuration space by *C*. The kinematic model is

$$\dot{x} = u \tag{6}$$

and let $\phi(x, x_0, x_f)$ be a multi-robot navigation function, where robots do not overlap. Then the control law is

$$u = -\nabla \tag{7}$$

where, $\nabla \circ = [\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}, \ldots, \frac{\partial}{\partial x_n}] \circ$ and ϕ is a multi-robot navigation function that drives all robots from the feasible initial configuration x_0 to any feasible final state x_f.

Two levels of motion controllers are defined. The global convergent controller manages the primary motion task and set of other controllers that lie within its range of convergence. The focus is to synthesis Büchi automaton that realizes the behaviour of the requirements of a robotic system. The steps for achieving this objective are outlined as follows:

Step 1: Given an LTL formula ϕ, a Büchi automaton is constructed and accepts words that satisfy ϕ. After the construction, the largest non-blocking sub-automaton A_{ϕ}^{NB} is constructed from A_{ϕ}. If A_{ϕ}^{NB} is empty then, the LTL formula ϕ should be rewritten. The LTL formulas that are used to synthesis controllers are of the form $\square G \wedge \phi$, where $\square G$ refers to the global controller which must always be active.

Step 2: Using A_{ϕ}^{NB} as a model, a function $\Delta : S \times O \rightarrow \{0,1\}^{|C|}$ is created to generate observation and controller predicates.

Step 3: The last step is to define controllers that conforms to the following control law: $u = -k_1 \nabla \phi + \beta \cdot u_2^*$. The details of how to derive this equation is given Loizou and kyriakopoulus[10].

6. ANALYSIS

Antoniotti and Mishra[6] introduced some interesting research directions in motion planning. However, there are some research questions that are not yet answered and some issues that are not clearly explained. Some of these questions include:

- the failure to achieve the goal of the paper, that is, to develop a compiler similar to silicon compilers,
- the ambiguity of some of the CTL formulas. That is, there are some of the CTL specifications in which supervisor synthesis could fail, for example $AX(AG(p_1)) \vee AX(AG(p_2))$, where the labels $A(s_2) = p_1$ and $A(s_4) = p_2$, and the state s_2 and s_4 do not have out-going edges, but self-looping. The failure could be due to the maximal ambiguous controllable sub-language, and
- another open problem is to investigate the use of restricted CTL[33].
- In addition, there are no any real-time issues and obstacle avoidance algorithms of motion planning for the walking machine that are discussed.

Apart from unanswered research questions, there are some important issues that need some detailed explanation. This includes, the modified model checker and the input specification language for the model checker. The detailed discussion of the modified model checker is important as this might affect the complexity of model checking algorithms. Other problems are the description of the specification language for the state machines, and the interpretation of the results. These are not clearly explained in detail.

Antoniotti and Mishra[6] tested the usage of the system with a train of two legs: the rear and front legs. The state machine which represents the behaviour of a leg was outlined and unregulated verification of the property similar to the one explained in Section 5 (i.e., states which should be avoided) gives NIL result, which states that the property is not satisfied. In the case that there are some states that are undesirable, the model checker outputs those states which are removed from the desired behaviour. The following shows an example of the results:

```
CMUCL 7> (omega-op K legs uncontrollable-events)
  ;;Debugging deleted...
 >>OMEGA(0): removable states = ((D1 SL2) (SL1
  D2))
 ————————————
  ;;Debugging deleted...
 >>OMEGA(1): removable states = NIL
  #<Representation for the approximation to K>
 CMUCL 8>
```

Belta[1], *et al.* provide incomplete answers and also highlight problems and challenges to answer the following question: "can a computational framework allowing for specifying such a task in a high-level, human-like language, with automatic generation of provably correct robot control laws be developed?" These problems and challenges are centered around the concept of discretisation which can either be environment-driven or control-driven. In the latter, an environment is represented by linear temporal logic at implementation level. This representation poses a number open questions. It is not clear as to which is the best specification language to use, that is, whether to use LTL or CTL? This is a problem since there are high-level specification tasks which cannot be expressed in LTL or vice versa CTL. In addition, a too expressive temporal logic might affect the performance of the analysis. For dynamic mobile robots, it might not be possible to execute strings over partitioned regions as in environment-driven discretisation. The best approach is to do the discretisation at a controller level. The idea behind control-driven discretisation is to divide the system into subtasks e.g., sensing modality and the behaviour of each subtask make up words in motion description languages (MDLs)[34]. The approaches in control-driven discretisation include: control quanta, motion primitives and feedback encoding. Other low-complexity methods use experimental data to mimic, for example, the behaviour of a human operator. In multi-robot systems, the control strategies could be influenced by studying the behaviour of flocks of birds or school of fish which can lead to some predictable behaviour. Alternatively such communications or control strategies can be achieved through the use of embedded graph grammars[35].

However, some of the problems outlined in Section 3 are not addressed Belta[1], *et al.* Instead they provide more questions than answers! In the case of environment-driven discretisation, when methods such as model checking analysis and choice of specification languages are applied to a real-world problem, the following three questions need to be answered:

- Controllers guaranteeing robot transition from one region to another or making a region an invariant for a robot have not yet developed for robots with nonholonomic constraints,
- This approach should take into account constraints induced by digital controllers and sensors such as finite input and output spaces, and
- Given a team of locally interacting robots, and a high level specification over some environment, how can provably correct (local) control strategies be generated? What global (expressive) specifications can be efficiently distributed? How should local interactions (e.g., message passing versus synchronisation on common events) be modelled? In the case of control-driven discretisation, the following questions need to be answered:
  - (a) What is the best choice of motion primitives for achieving a given class of tasks?,
  - (b) Given an alphabet of motion primitives, what is the penalty associated with restricting the robots

trajectories to those obtained through combination of those primitives with respect to a larger set of primitives?, and

(c) Can this symbolic approach to motion planning be extended to multi-robot environment?

However, some of these questions could be answered through the proposed method presented in Section 7. The safety and liveness properties outlined in Section 5.2 are verified with Uppaal and all the properties are satisfied. The detailed results are depicted by Figures in Quottrup[7], *et al*. and these are not shown in this paper due to limited space. The same thing applies to work by Lozou and kyriakopoulos[10], that is, the synthesis is successfully done and the results are presented in the form of figures.

Quottrup[7], *et al.* use theory of timed automata to model a motion planning problem in a multi-robot environment. Despite the fact that the use of timed automata to model and coordinate communication in a multi-robot environment is a good idea, the authors of this paper ignored a number of real issues that need to be addressed in motion planning. These issues include: (a) the robot dynamics, and (b) the ever-changing environment. The use of a planar grid with static obstacles cannot be applicable to a real-world problem. The approach to motion planning in this paper is more about verification than motion planning issues such as motion algorithms, task planning, complexity and completeness, computational geometry, etc. The method proposed in Section 7 also uses theory of timed automata, but the dynamics of robots communicating in real-time and operating in an ever-changing environment are taken into account.

Fainekos[8], *et al.* generated continuous trajectories which satisfy temporal logic formulas. This is achieved through three steps:

- the decomposition of workspace into cells[19,20]
- the use model checking NuSMV[21] to generate plans for discrete motion planning that satisfy LTL properties, and
- the generation of continuous trajectories that satisfy the specified LTL properties.

This is a good approach that gives some hope that one day people will be able to interact with mobile robots safely. However, there are some concerns that need to be addressed to achieve this goal and this includes the use of model checking algorithms without any modification. This is because when model checking tools such as SPIN are used, these are likely to generate unnecessarily long paths (trajectories). For example, let's say Fig. 2 depicts an instance of a state space (of the environment) that is used to generate a path that satisfies some LTL property φ, (i.e., $p[i] \models \neg(\neg\varphi)$ as in Section 5). If model checkers, that employ a nested depth first search algorithm are used to compute strongly connected components (SCC) with an accepting cycle[23] (as shown in Fig. 2), the trajectory might be: $A \to B \to C \to D \to E$, where B represents some million of states from the initial state A. But, the shortest trajectory is: $A \to C \to D \to E$, from the initial state A. It is clear that the use of model checking algorithms without any modification might affect the performance of the synthesis process.

The problem formulation and method summarised in Section 5 facilitate the synthesis of the following properties[8]:

- The requirement is to visit rooms in no particular order, and it is formally defined as $(\Diamond r_1 \wedge \Diamond r_2 \wedge \Diamond r_3 \wedge \Diamond r_4 \wedge \Diamond r_5 \wedge \Diamond r_6)$. The generation of a trajectory by NuSMV is fast and the synthesis with MATLAB takes not more than 15s.
- The requirement is to visit room r_2 then room r_1 and then cover rooms r_3, r_4, r_5 while avoiding obstacles o_1, o_2 and o_3. The path is generated with SPIN. It is not mentioned how long each process takes, but the figure showing the environment and the path is shown.
- The requirement is to start in a white room and go to both black rooms. The environment for this requirement consists of 1156 rooms and its discrete abstraction consists of 9250 triangles. The path generation takes about 55s with NuSMV and the controller synthesis with MATLAB lasts for about 90s.

Kress-Gazit[9], *et al.* presented some promising results in using temporal logic in motion planning. Robots use sensors to gather information about their surroundings and GR(1) formulas are appropriate in representing this interaction. Since GR(1) is a class of LTL, if the same type of logic can be derived out CTL* a more robust and expressive logic can be found to synthesise many requirements in robot motion planning.

The method presented by Kress-Gazit[9], *et al.* is tested with two examples: (i) single robot-nursery scenario and (ii) multi robot–search and Rescue which are outlined as follows:

- The first example states that; "starting in region 1, keep checking whether a baby is crying in region 2 or 4. If you find a crying baby, go look for an adult in regions 6, 7 and 8. Keep looking until you find him. After finding the adult, go back to monitor the babies and so on.." It takes 2 seconds to synthesis an automaton that realizes the requirement and the automaton has 41 states.
- The second example states that; "In this search and rescue scenario, we employ two UAV's that continuously search regions 1, 2, 3, 7 and 8 for injured people. Once an injured person is found, a ground vehicle (ambulance)

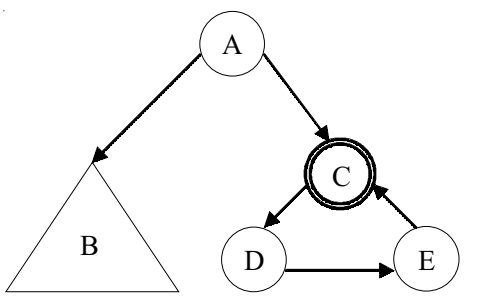


Figure 2. A graph representing a state space.

goes to the person's location and helps, the ground vehicle does not move. . ." In this case, it takes about 60 s to synthesis an automaton which consists of 282 states.

Loizou and Kyriakopoulos[10], also presented a good methodology in using temporal logic to synthesis motion planning controllers. The main concern was the refinement of the LTL formula if the largest non-blocking sub-automaton is empty. This concern is the same as the one that is highlighted by Fainekos[8], *et al.*, where model checking algorithms are used without modification to fit the problem at hand (i.e., temporal motion planning). The refinement of the LTL formula might increase the number of states for the automaton and thus affect the performance of the synthesis.

7. SUGGESTED APPROACH

In order for people to live and interact with robots, the goal is to make them able to perform complex tasks effectively and safely. This section presents a methodological approach that addresses most of the problems discussed in Section 6. Solutions to these problems are likely to bring the goal close to reality. A layered top-down framework is proposed and it consists of three main layers: (i) the formal specification layer, (ii) synchronisation layer, and (iii) planning and control layer. Figure 3 depicts an overview of the framework.

7.1 The Three Layers

- *First Layer:* The formal specification layer enables human beings to interact with a mobile robot using the human-like specification languages, referred to as formal specification languages. These specification languages are expressed in modal logics such as LTL [23], CTL[4], and TCTL[5,36]. One of the problems that need an attention at this layer is the expressiveness of a formal specification language. All the aforementioned temporal logics express different user requirements, i.e., there are requirements that are expressed in LTL, but cannot be expressed in CTL and vice versa. This is one of the problems mentioned by Belta[1], *et al.* The proposed approach, at this layer, is to study common user requirements for interacting with mobile vehicles and select a subset of temporal logic formulas from each temporal logic that addresses the requirements. This approach is likely to cover most, if not all, of the user needs to facilitate an effective and safe interaction of humans and mobile vehicles.
- *Second Layer:* The synchronisation of different temporal logic formulas into a synchronised product automaton. This automaton represents the discrete desired behaviour of the mobile robot due to its surroundings. The synchronizer outputs propositions of the current behaviour of the robot, the examples include: "obstacle = 20 (i.e., the vehicle is 20 m from an obstacle)", "TJunction=true, (i.e., the robot is at a T-junction)", "Stop A_time=15, (i.e., the robot stays at Stop A for 15 min)", etc. The true values of these propositions are synthesised from sensory data. This approach of outputting propositions instead of automaton given by is likely to improve the performance of a motion planner. The motion planner only processes a few propositions instead of the entire synthesised automaton. The problem of multi-robot communication, which is also mentioned by Belta[1], *et al.* can also be addressed at this layer, that is, the synchronizer can keep a synchronised behaviour of multiple mobile vehicles and outputs the desired propositions to different motion planners for each vehicle.
- *Third Layer:* Planning and control of a mobile vehicle. The layer involves two types of planning: the global path planning and local motion planning. In the latter, the well known algorithms such as A* can be used, while in the former some modified existing techniques such as rapidly-exploring random trees (RRT)[38] can be used to compute trajectories at real-time between path coordinates provided by global path planning. The modified algorithm will take as an input atomic propositions from the above layer (the synchroniser) and outputs trajectories (velocities, directions) based on the true values of the propositions. The control part of the layer can also use the well studied control techniques such as feedback controller policy[39].

7.2 Synthesis

One of the goals of the proposed approach in temporal motion planning in this paper is to make the synthesiser as expressive as possible. The development of a distributed synthesiser for the temporal logics LTL, CTL, CTL* and TCTL is likely to achieve this goal. The temporal logics LTL and CTL presented in Sections 4 are contained in the CTL* temporal logic[36]. Therefore, the syntax and semantics of the CTL* are assumed to be intuitively clear from that of the CTL as these are both interpreted on branching models. The following examples show some of properties which can be expressed in one temporal logic, but not the other:

a) $A[GF\, p \Rightarrow F\, q]$: This is an LTL formula for which an equivalent formulation of CTL does not exist. The formula states that if p holds infinitely often then q will eventually hold. This property is very important in a multi-robot environment, especially where robotic agents communicate over the network. That is, if one agent is infinitely often sending a message to another agent, the recipient agent will eventually receive it.

b) $AG\, EF\, p$: This is a CTL formula for which an equivalent formulation of LTL does not exist. This formula states that there is a path (trajectory) for which p holds irrespective of the current state. This is also an important property in robotics, where for instance, a robotic system can recover from an error state.

c) $EG\, p \Rightarrow AF_{<5}\, q$: This is a TCTL formula for which an equivalent formulation of both CTL and LTL does not exist. Both CTL and LTL deal with qualitative

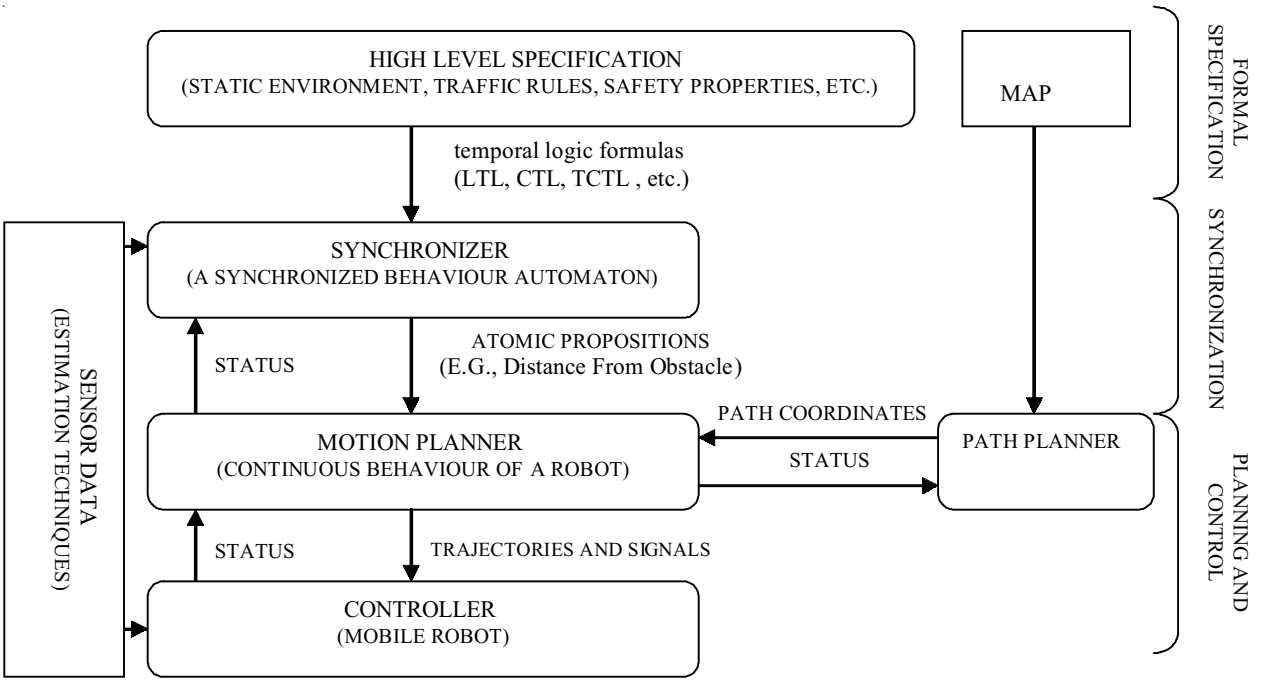


Figure 3. A proposed framework of a symbolic motion planning.

temporal properties and cannot express quantitative timing constraints. This is a punctuality requirement. For example, this may state that if a robotic agent has sent a message, it receives a reply within five units of time.

These are some of the properties to be synthesised from our proposed framework. However, Bair and Katoen[36], stated that the model checking of CTL* is hard. Model checking is an automatic verification technique for finite state concurrent systems and our approach is similar to developing model checking algorithms, though the output is not the same as in model checking. On the other hand, Fainekos[40], stated that the complexity of model checking CTL* is the same as model checking the LTL, the difference is that CTL* model checking requires some additional bookkeeping as it combines both CTL and LTL.

To synthesise an automaton A that satisfies a certain requirement φ during motion planning, the states of the automaton are computed on the fly due to the changing environment (i.e., the workspace). The set of these states is formally defined as $[\![\varphi]\!]_A = \{s \in S \mid (A,s) \wedge\!\%_s \varphi\}$ Where, $A = (S, S_0, R, AP, Label)$ is a Kripke structure. The computation of $[\![\varphi]\!]_A$ has its roots in a fixed point theory[36] and the following rules are applied to determine these states. Let φ and ψ be CTL* (or TCTL) formulas and p be atomic proposition (i.e, $p \in AP$), then:

$$
\begin{aligned}
[\![p]\!]_A &= \{s \in S \mid p \in Label(s)\} \\
[\![\neg\varphi]\!]_A &= S \setminus [\![\varphi]\!]_A \\
[\![\varphi \vee \psi]\!]_A &= [\![\varphi]\!]_A \cup [\![\psi]\!]_A \\
[\![\varphi \wedge \psi]\!]_A &= [\![\varphi]\!]_A \cap [\![\psi]\!]_A
\end{aligned}
$$

$$
\begin{aligned}
[\![EG\varphi]\!]_A &= [\![\varphi]\!]_A \cap \{s \in S \mid \exists s' \in R(s) \cap F(Z)\} \\
[\![AG\varphi]\!]_A &= [\![\varphi]\!]_A \cap \{s \in S \mid \forall s' \in R(s) \cap F(Z)\} \\
[\![EF\varphi]\!]_A &= [\![\varphi]\!]_A \cup \{s \in S \mid \exists s' \in R(s) \cap F(Z)\} \\
[\![AF\varphi]\!]_A &= [\![\varphi]\!]_A \cup \{s \in S \mid \forall s' \in R(s) \cap F(Z)\}
\end{aligned}
$$

Where, $R(s)$ is the successor state of $s \in S$ and Z is the powerset of S (i.e., 2^S) and $F(Z)$ is a recursive function that terminates when either a greatest fix-point or a least fix-point is reached. The computation of these sets of states requires a lot of computation and an intelligent autonomous system is proposed.

7.3 Intelligent Mobile Platform

Centralised systems have disadvantages that make these unsuitable for large-scale integration, including high reliance on centralised communication, high complexity, lack of scalability, and high cost of integration. The use of distributed intelligence system technologies avoids these weaknesses. Distributed intelligence systems are based on the use of cooperative agents, organised in software components that independently handle specialised tasks and cooperate to achieve system-level goals and achieve a high degree of flexibility. By distributing the temporal motion planning algorithms, it is possible to achieve greatly improved robustness, reliability, scalability, security, and safety in mobile robots. Key to achieving these benefits is the use of mobile agent system technologies that establish a peer-to-peer environment to enable coordination, collaboration, and cooperation within the network. The proposed framework shown in Fig. 3 is implemented on top of the mobile intelligent platform.

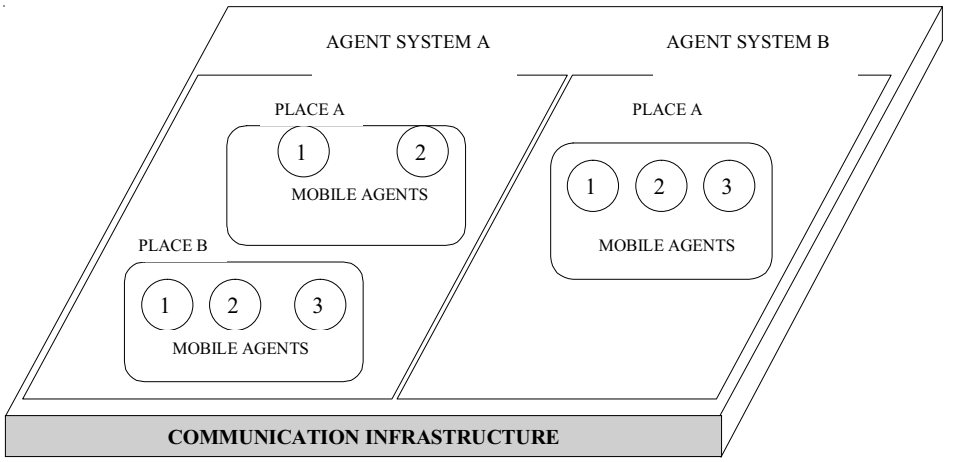


Figure 4. MAF intelligent platform[41].

Figure 4 depicts an overview of the platform. This platform consists of mainly three components: the mobile agents, the agent systems, and the places. Mobile agent is a computer program (or algorithm) that acts autonomously on behalf a person and has the ability to transport itself from one system in a network to another. It has its own thread of execution, so tasks can be performed on its own initiative[41]. It is proposed that the components shown in Fig. 3 be implemented as mobile agents to achieve the aforementioned advantages (i.e., scalability, efficiency, etc.).

An agent system is a platform that can create, interpret, execute, transfer, and terminate mobile agents. An agent system is also associated with a person that it acts on his or her behalf[41]. A place is a context within an agent system in which an agent can execute. The source and destination places can reside on the same agent system or on different agent systems that support the same agent profile[41].

8. CONCLUSIONS

Temporal logic real-time motion planning is a promising approach in designing and developing motion planning frameworks and algorithms for autonomous mobile vehicles. This paper reviewed six research papers that incorporate the use of theory of computation and formal methods tools and techniques. The aim of the first paper is to develop a compiler to synthesis controller programs for robotic applications and manufacturing tasks. The second paper shows how the theory of timed automata can be used to model, analyse, and verify a motion planning problem in a multi-robot environment. The third paper is a result of a workshop held in 2006 by the authors to address the issues, challenges, and problems of using theory of computation and formal methods, tools, and techniques in motion planning. They coined the term symbolic motion planning. The fourth paper uses existing model checking tools such as SPIN and NuSMV to generate a discrete path that satisfies a certain requirement. The path is then translated into continuous trajectory to drive the robot from some initial-state to the goal-state. The fifth and the sixth papers also use temporal logics to synthesise automata that realise some complex requirements in robot motion planning. The procedure is similar to that one of the fourth paper with some differences in the methodology and the temporal logic used. Some of the problems discussed in Section 5 are addressed by a proposed framework briefly presented in Section 7. The framework follows a top-down layered approach and consists of three layers: (i) the formal specification layer, (ii) the synchronisation layer, and (iii) the planning and control layer. The framework is also proposed to be implemented on top of an intelligent mobile platform. The reason for this choice is due to the advantages provided by distributed architectures such as robustness, reliability, scalability, security, and safety.

ACKNOWLEDGEMENT

The research conducted and reported here was funded by the Council for Scientific and Industrial Research (CSIR), South Africa, The authors would like to thank Professor Jitendra Raol for his advice and comments.

REFERENCES

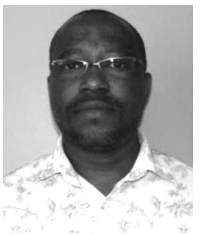
1. Belta, C.; Antonio; Egerstedt, M.; Frazzoli, E.; Klavins, E. & Pappas, G.J. Symbolic planning and control robot motion. *IEEE Robotics Autom. Mag.*, 2007, **14**(1), 61-70.
2. R.T.M. Corporation. Road Traffic Report. March 2008. http://www.arrivealive.co.za.
3. LaValle, S.M. Planning algorithms. University of Illinois, Cambridge University Press, 2006.
4. Clarke, E.M. & Emerson, E.A. Design and synthesis of synchronization skeletons using branching-time temporal logic. *In* Logic of Programs Workshop, Springer-Verlag, London, UK, 1982. pp. 52–71.
5. Seotsanyana, M. Formal specification, development, and verification of safety interlock systems: comparative case study, Ed. 1. VDM Verlag, September 2008.
6. Antoniotti, M. & Mishra, B. Discrete event models + temporal logic = supervisory controller: Automatic

synthesis of locomotion controllers. *In* IEEE International Conference on Robotics and Automation, 1995, pp. 1441-446.

7. Quottrup, M. M.; Bak, T. & Izadi-Zamanabadi, R. Multi-robot planning: a timed automata approach. *In* ICRA, 2004. pp. 4417-422.
8. Fainekos, G.E.; Kress-Gazit, H. & Pappas, G.J. Temporal logic motion planning for mobile robots. *In* Proceedings of the 2005 IEEE International Conference on Robotics and Automation, April 2005. pp. 2020-025.
9. Kress-Gazit, H.; Fainekos, G.E. & Pappas, G.J. Where's Waldo? Sensor-based temporal logic motion planning. *In* Proceedings of the IEEE Conference on Robotics and Automation, Vol. 2324, 2003. pp. 3546-551.
10. Loizou, S.G. & Kyriakopoulos, K.J. Automatic synthesis of multiagent motion tasks based on LTL specification. *In* 43rd IEEE Conference on Decision and Control, 14-17 February 2004.
11. Ramadge, P.J. & Wonham, W.M. The control of discrete event systems. *Proceedings of IEEE*, 1989, **77**(1), 81-98.
12. Holloway, L.E. & Krogh, B.H. Synthesis of feedback control logic for a controlled petri nets. *IEEE Trans. Autom. Contr.*, May 1990, **35**(5), 514-23.
13. McCarragher, B.J. & Asada, H. A discrete approach to the control of robotic assembly tasks. *In* IEEE International Conference on Robotics and Automation, 1993. pp. 331-36.
14. Clarke, E.M.; Emerson, E.A. & Sistla, A.P. Automatic verification of finite-state concurrent system using temporal logic specifications. *ACM Trans. Program. Langu. Sys.*, 1986, **8**(2), 244-63.
15. Larsen, K.G. Pettersson, P. & Yi, W. UPPAAL in a Nutshell. *Int. J. Software Tools Technol. Transfer*, Oct. 1997, **1**(1–2), 134-52.
16. Bertoli, P.; Cimatti, A.; Pistore, M.; Roveri, M. & Traverso, P. MBP: A model based planner. *In* Proceedings of the IJCAI'01 Workshop on Planning Under Uncertainty and Incomplete Information, Seattle, August 2001. www. citeseer.ist.psu.edu/bertoli01mbp.html
17. Bacchus, F. & Kabanza, F. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 2000, **116**(1-2), 123-91.
18. Jensen, R.M. & Veloso, M.M. OBDD-based universal planning for synchronized agents in non-deterministic domains. *J. Artifi. Intelli. Res.,* 2000, **13**, 13-189.
19. Conner, D.; Rizzi, A. & Choset, H. Composition of local potential functions for global robot control and navigation. *In* Proceedings of 2003 IEEE/RSJ International Conference on Intelligent Robots & Systems (IROS 2003). IEEE, 2007. pp. 3116–121.
20. Belta, C. & Habets, L. Constructing decidable hybrid systems with velocity bounds. *In* 43rd IEEE Conference on Decision and Control. Bahamas, 2004.
21. Cimatti, A.; Clarke, E.; Giunchiglia, F. & Roveri, M. NUSMV: A new symbolic model verifier. *In* Proceedings 11th Conference on Computer-Aided Verification (CAV'99), Series Lecture Notes in Computer Science, *edited by* N. Halbwachs & D. Peled, No. 1633. Springer, Trento, Italy, July 1999. pp. 495-99.
22. de Berg, M.; van Kreveld, M.; Overmars, M. & Schwarzkopf, O. Computational geometry: Algorithms and applications, Ed. 2. 2000.
23. Holzmann, G.J. The spin model checker: Primer and reference manual. Lucent Technilogies Inc., Bell Laboratories, Addison-Wesley, 2004.
24. Alur, R.; Henzinger, T.A.; Lafferriere, George, & Pappas, G.J. Discrete abstractions of hybrid systems. *In* Proceedings of the IEEE, 2000, pp. 971-84.
25. Habets, L. & van Schuppen, J. A control problem for affine dynamical systems on a full-dimensional polytope. *J. Artifi. Intelli. Res.,* **40**(1), 21-35, 2004.
26. Kloetzer, M. & Belta, C. A fully automated framework for control of linear systems from temporal logic specifications. *IEEE Trans. Autom. Contr.,* February 2008, **53**(1), 287-97.
27. Fainekos, G.E.; Kress-Gazit, H. & Pappas, G.J. Hybrid controllers for path planning: A temporal logic approach. *In* Proceedings of the 44th IEEE Conference on Decision and Control, December 2005. pp. 4885-890.
28. Piterman, N.; Pnueli, A. & Sa'ar, Y. Synthesis of reactive(1) designs. *Lecture Notes Comp. Sci.,* 2006, **3855**.
29. Pnueli, A. & Rosner, R. On the synthesis of a reactive module. *In* POPL '89: Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. ACM, New York, 1989. pp. 179-90.
30. Klavins, E. Automatic synthesis of controllers for assembly and formation forming. *In* Proceedings of the International Conference on Robotics and Automation, 2002.
31. Tabuada, T. & Pappas, G.J. Linear time logic control of linear systems. *Submitted, IEEE Transactions Autom. Contr.*, 2004.
32. Egerstedt, M. Motion description languages for multi-modal control in robotics. *In* Control Problems in Robotics, *edited by* A. Bicchi, H. Cristensen & D. Prattichizzo, 2002. pp. 75-90.
33. Mishra, B. & Clarke, E.M. Hierarchical verification of asynchronous circuits using temporal logic. *Theo. Compu. Sci.*, 1985, **38**, 269-91.
34. Egerstedt' M. & Brockett, R. Feedback can reduce the specification complexity of motor programs. *IEEE Trans. Autom. Contr.*, 2003, 48(2), 213-23.
35. Klavins, E.; Ghrist, R. & Lipsky, D. A grammatical approach to selforganizing robotic systems. *IEEE Trans. Autom. Control*, 2006, **51**(6), 949-62,
36. Baier, C. & Katoen, J.-P. Principles of model checking, Ed. 1. MIT Press, September 2008.
37. K.-G. H. & P. G.J., Automatically synthesizing a planning and control subsystem for the DARPA urban challenge. *In* IEEE International Conference on Automation Science and Engineering, Aug. 2008, Vol. 23, No. 26. pp. 766 -71.
38. Valle, L. Rapidly-exploring random trees: A new tool

for path planning. Iowa State University. Tech. Rep. 98-11, 1998.
39. Choset, H.; Lynch, K.M.; Hutchison, S.; Kantor, W.B.G.; Kavraki, L.E. & Thrun, S. Principle of robot motion: Theory and implementation. MIT Press, 2005.
40. Fainekos, G.E. An introduction to multi-valued model checking. Dept. of CIS, University of Pennsylvania, Tech. Rep. MS-CIS-05-16, September 2005.
41. O.M. Group. Mobile agent facility specification. January 2000, http://www.omg.com

Contributor



Mr Motlatsi Seotsanyana received his MSc (Computer Science) from the University of Stellenbosch, in 2007. He is currently a researcher at Mobile Intelligent Autonomous Systems, Modelling and Digital Science, Council for Scientific and Industrial Research, in South Africa. His current research interests include: formal methods, motion planning, software engineering, distributed systems, and database management systems.