

# Automated Planning of Motion Tasks for Multi-Robot Systems

Savvas G. Loizou Kostas J. Kyriakopoulos

Control Systems Laboratory, Mechanical Eng. Dept  
National Technical University of Athens, Greece  
{sloizou, kkyria}@central.ntua.gr

**Abstract**—In this paper we present an automated task planning methodology to automatically compose multi-robot motion tasks. A motion task is defined as an input-output module. Primitive motion tasks represented as modules (called primitive modules) are defined based on Linear Temporal Logic specifications. The task planning is then considered as a composition over compatible primitive modules in such a way that the inverted motion task module and the primitive modules form a closed chain. The problem is posed as an integer programming problem and is reduced to the combinatorial optimization problem of shortest dipath which can be solved in polynomial time. The basic safety and liveness specifications of the primitive task controllers are inherited by the resulting composed system and the requested task specifications are satisfied by construction, ensuring a correct design. The effectiveness of the proposed methodology is shown through computer simulation.

## I. INTRODUCTION

During the last decade an increasing shift of focus of the research community has taken place from single robot systems to multi-robot systems. Motion planning, and in extension the planning of motion tasks, is a key issue in the development of autonomous multi-robot systems capable of achieving complex multi-faceted tasks. Our main motivation comes from the field of micro-robotics, where a team of micro-robotic agents must cooperate to perform various tasks.

Most of the work in multi-robot literature treat the motion planning and task planning problems in isolation. In [1] the authors propose a framework for modeling multi-robot tasks using finite state automata to build a model based on entomological evidence that series of acts are regulated by local stimuli. In [2] the authors show that the multi-robot task allocation problem can be reduced to an instance of the optimal assignment problem. In [3] the authors incorporate spatiotemporal constraints in a nonlinear trajectory generator to produce a predesigned robot activity. The authors of [4] propose a software assisted task planning and a hierarchical execution scheme taking into account spatiotemporal constraints for a team of UAV's. In [5] an approach to task planning based on the probabilistic roadmap method is presented, taking into account geometrical constraints.

The authors want to acknowledge the contribution of the European Commission through contract IST-2001-33567-MICRON and IST-2001-32460-HYBRIDGE

Formally we consider a multi-robot motion task as a sequence of robot motions and operations, achieving a specified end product.

We tackle the automated multi-robot task planning problem adopting a component based approach in the sense that a task plan can be considered as a composition of existing components. Our approach tackles the problem in both the motion planning and the task planning levels. At the motion planning level, the system's evolution is dictated from Linear Temporal Logic specifications [6] applied on Multi-Robot Navigation Function (*MRNF*) [7], [8] based controllers. At the task planning level, the problem is posed as a module composition one [9] which is then reduced to the shortest dipath problem that is solvable in polynomial time.

The rest of the paper is organized as follows: In section II the system description and problem statement are presented. Section III discusses the motion controller construction from LTL specifications. Section IV formalizes the motion task as an input/output module and section V presents the automated task planning procedure. Section VI presents simulation results and the paper concludes with section VII

## II. SYSTEM DESCRIPTION & PROBLEM STATEMENT

Consider a multi-robot system, with  $m$  mobile robots, and their continuous workspace  $W \subset R^2$ . (The methodology can be applied as well to higher dimensional workspaces). Without loss of generality, we assume that each robot  $R_i$ ,  $i = 1 \dots m$  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 center of the disk and  $r_i$  is the radius of the robot. The configuration of each robot is represented by  $x_i$  and the configuration space  $C$  is spanned by  $X_C = [x_1^T \dots x_m^T]^T$ . The robot kinematics can be trivially described through the following first order kinematic model:

$$\dot{X}_C = u \quad (1)$$

where  $u$  is a control law. Moreover, let us consider a set of discrete states for the system and denote it with  $D$ . Those states may describe partitions of the workspace, an internal robot state, a tool state, etc. The multi-robot system can thus be described by the following hybrid system [10]:

$$\mathcal{R} = \{X, X_0, X_F, F, E, I, G, R\} \quad (2)$$

where:

- $X = D \times W$  is the state space

- $X_0 \subseteq X$  is the set of initial states
- $X_F \subseteq X$  is the set of final states
- $F : X \rightarrow TW$  assigns to each discrete state  $q \in D$  a vector field  $u = F(q, \cdot)$
- $E \subseteq D \times D$  is the set of discrete transitions
- $I : D \rightarrow 2^W$  assigns to each discrete state  $q \in D$  a set  $I(q) \subseteq W$  called the invariant
- $G : E \rightarrow D \times 2^W$  assigns to  $e = (q_1, q_2) \in E$  a guard of the form  $\{q_1\} \times U, U \subseteq I(q_1)$
- $R : E \rightarrow D \times 2^W$  assigns to  $e = (q_1, q_2) \in E$  a reset of the form  $\{q_2\} \times V, V \subseteq I(q_2)$

The trajectories of  $\mathcal{R}$  start in  $(q, x) \in X_0$  and consist of discrete transitions in  $D$  and the absolutely continuous evolutions in  $W$  are governed by (1). The vector field  $F(q_i, \cdot)$  is produced by an *MRNF* based controller (see section III) whose destination configuration is  $x_{C,g} \in I(q_i)$ .  $I(q)$  is typically the range of convergence of the controller  $u = F(q, \cdot)$ . The guard  $G$  specifies a subset of the state space where a certain transition is enabled. For each discrete location  $q$  there must be at least one guard for which  $x_{C,g} \in U$ . Since the trajectories of our system are absolutely continuous in  $W$ , the reset map  $R$  will be the identity map in  $W$ . Finally we assume that the hybrid system is non-blocking, i.e. from every state either a continuous evolution or a discrete transition is possible

The problem of automated motion task planning we are considering can thus be stated as follows:

*“Given the hybrid system (2), a task specification in terms of initial and final states of (2) and a set of vector fields, compose a controller achieving the given specification or report failure if such a controller does not exist.”*

### III. MOTION CONTROLLER SYNTHESIS BASED ON LTL SPECIFICATIONS

In composing the building blocks of a multi-robot motion task, we are going to use the automatic controller synthesis based on LTL specifications that was developed in our previous work (see [6] for the detailed constructive procedure). The resulting vector fields  $C_\varphi$  will be used at the discrete locations of  $\mathcal{R}$ .

Linear Temporal logic is an extension of propositional logic suitable for reasoning about infinite sequences of states. We use a fragment of LTL, equipped with the usual propositional connectives:  $\wedge$  : **and**,  $\vee$  : **or**,  $\neg$  : **not** and is extended with two temporal operators: “ $\bigcirc$ ” (next) and “ $\mathcal{U}$ ” (unless). The formulas of the logic are built from atomic propositions using propositional connectives and temporal operators. The sequences considered are isomorphic to natural numbers and each state is a propositional interpretation. Purely propositional formulas are interpreted in a single state and the temporal operators indicate in which state of a sequence their arguments must be evaluated. We denote by  $P$  the set of atomic propositions and recursively define the well formed formulas (wff) as follows:

- **true**, **false**,  $p$ ,  $\neg p$  are wff for all  $p \in P$ ;
- if  $\varphi_1$  and  $\varphi_2$  are wff, then  $\varphi_1 \wedge \varphi_2$  and  $\varphi_1 \vee \varphi_2$  are wff;

- if  $\varphi_1$  and  $\varphi_2$  are wff, then  $\bigcirc \varphi_1$ , and  $\varphi_1 \mathcal{U} \varphi_2$  are wff formulas;

Implication  $\varphi_1 \Rightarrow \varphi_2$  is defined as the abbreviation of  $\neg \varphi_1 \vee \varphi_2$ . From the unless operator, another commonly used operator can be defined:

$$\square \varphi = \varphi \mathcal{U} \text{false}$$

which is read “always” and requires that its arguments be true at all future points. Wff are interpreted over sequences of states  $\sigma : \mathbb{N} \rightarrow 2^P$ . For a sequence  $\sigma$ ,  $\sigma(i)$  denotes the  $i$ 'th state,  $\sigma[i]$  represents the prefix of  $\sigma$  obtained from the first  $i$  elements of  $\sigma$  and  $\sigma^i$  represents the suffix of  $\sigma$  obtained by removing the  $i$  first states, i.e.  $\sigma^i(j) = \sigma(i+j)$ . The truth value of a formula on a sequence  $\sigma$ , is taken to be the truth value obtained by starting the interpretation of the formula in the first state of the sequence, and is given by the following rules:

For any  $p \in P$ , wff formulas  $\varphi_1, \varphi_2$  and  $i \in \mathbb{N}$ :

- For all  $\sigma$ , we have  $\sigma \models \text{true}$  and  $\sigma \not\models \text{false}$
- $\sigma \models p$  iff  $p \in \sigma(0)$
- $\sigma \models \neg p$  iff  $p \notin \sigma(0)$
- $\sigma \models \varphi_1 \wedge \varphi_2$  iff  $\sigma \models \varphi_1$  and  $\sigma \models \varphi_2$
- $\sigma \models \varphi_1 \vee \varphi_2$  iff  $\sigma \models \varphi_1$  or  $\sigma \models \varphi_2$
- $\sigma \models \bigcirc \varphi_1$  iff  $\sigma^1 \models \varphi_1$
- $\sigma \models \varphi_1 \mathcal{U} \varphi_2$  iff either  $\sigma \models \varphi_1$  or  $\sigma \models \varphi_2$  or  $\exists i > 0$  such that  $\sigma[i] \models \varphi_1$  and  $\sigma^i \models \varphi_2$

Let us consider a simple example to illustrate the use of LTL. This example attempts to model a fragment of a possible behavior we expect from a micro-robotic multi-agent system during a biological cell transportation task. Assume robot 1 is carrying the cell and robot 2 is monitoring the procedure by tracking robot 1. If a fault occurs to robot 2 the system must slow down or stop until robot 3 replaces robot 2 and then continue execution of the task. Let  $T_{21}$  and  $T_{31}$  predicates be active when state feedback controllers for robot 2 and 3 respectively to track robot 1 are active and  $St_1$  predicate be active when a stalling controller for robot 1 is active. Moreover let predicate  $E_2$  be active when an error occurs to robot 2 and  $rT_{31}$  be active when robot 3 assumes it's tracking position. The specification can now be defined as:

$$\square G \wedge (T_{21} \mathcal{U} (E \wedge \bigcirc (St_1 \mathcal{U} (rT_{31} \wedge \bigcirc \square T_{31}))))$$

The predicate  $G$  is used to indicate the underlying *MRNF* controller establishing global convergence and collision avoidance for the system. Since safety and liveness are fundamental properties for our system, the global controller predicate must be active throughout the task and every formula  $\varphi$  is equivalent to  $\square G \wedge \varphi$ . The set of predicates  $P$  is the disjoint union:

$$P = \mathcal{O} \cup^* \mathcal{C} \cup^* \{G\}$$

of the set of observation predicates  $\mathcal{O}$ , controller predicates  $\mathcal{C}$  and the global controller predicate  $\{G\}$ . Observation predicates and the global controller predicate are uncontrollable events, but controller predicates are controllable events, in

the sense that we can turn on and off the corresponding controllers. This has to do with the fact that we consider a controller predicate to be active if and only if a control law associated with it is active, i.e. we consider that robot 1 being tracked by robot 2 is actually equivalent to controller  $T_{31}$  (which is a tracking controller) being active.

#### IV. MOTION TASK AS AN I/O MODULE

Consider a finite set of ports  $P = P_{in} \cup P_{out}$  with  $P_{in} \cap P_{out} = \emptyset$ , where  $P_{in}$  the set of input and  $P_{out}$  the set of output ports. We define a motion task as a module  $\mathcal{T}$  with a single input and a single output port:

$$\mathcal{T} = \{ in, out \} \quad (3)$$

where  $in \in P_{in}, out \in P_{out}$ . The motion task module will be used in specifying the required motion task.

We will need the following definition:

*Definition 1:* We define for the hybrid systems  $A, B$  the sub-automaton relation  $\sqsubseteq$  and we say that  $B \sqsubseteq A$  if the following conditions hold:

- $X_B \subseteq X_A$
- $D_{0_B} \subseteq \{q \in D_B \mid \exists q_i \notin D_B : (q_i, q) \in E_A\}$
- $D_{F_B} \subseteq \{q \in D_B \mid \exists q_i \notin D_B : (q, q_i) \in E_A\}$
- $I_B(q) \subseteq I_A(q) \setminus \{G(q, q_j) \mid q \in \{D_B \setminus D_{F_B}\}, q_j \notin D_B : (q, q_j) \in E_A\}$
- $F_B(q, I_B(q)) = F_B(q, I_B(q))$
- $E_B(q) = \{(q, q_j) \in E_A \mid q, q_j \in D_B\} \cup \{(q, q_j) \in E_A \mid q \in D_{F_B}, q_j \notin D_B\}$
- $G_B(q) = G_A(q), \forall q \in D_B$
- $R_B(E_B) = R_A(E_B)$

A motion task is considered to be composed by a set of composing blocks called the primitive motion task modules.

*Definition 2:* A primitive motion task module  $\mathcal{P}$  is the following tuple:

$$\mathcal{P} = \{in, out, \Delta\} \quad (4)$$

where

- $\Delta \sqsubseteq \mathcal{R}$  is a hybrid system, where
  - $F_\Delta = C_\varphi(x_g)$  with  $C_\varphi(x_g)$  the vector field resulted from the LTL formula  $\varphi$  with  $x_g$  the destination configuration of controller  $G$
  - $x_g \in G_\Delta(q_f), \forall q_f \in D_{F_\Delta}$
  - $I_\Delta \equiv \mathcal{R}(G)$ , where  $\mathcal{R}(G)$  denotes the range of convergence of the controller  $G$
- $in = \beta_{in}(X_{0_\Delta})$  where  $\beta_{in} : X \rightarrow P_{in}$
- $out = \beta_{out}(X_{F_\Delta})$  where  $\beta_{out} : X \rightarrow P_{out}$

*Definition 3:* Ports  $b \in P_{in}$  and  $d \in P_{out}$  are said to be compatible, if  $\forall \mathcal{P}_i, \mathcal{P}_o : b \in \mathcal{P}_i \wedge d \in \mathcal{P}_o$  the following properties hold:

- $\forall (q_f, q_j) \in E_{\Delta_o} : q_f \in D_{F_{\Delta_o}} \implies q_j \in D_{0_{\Delta_i}}$
- $G_{\Delta_o}(q_f, q_j) \subseteq \{q_f\} \times I_{\Delta_i}(q_j), \forall (q_f, q_j) \in E_{\Delta_o}, \forall q_f \in D_{F_{\Delta_o}}$

The set  $C \subset P_{in} \times P_{out}$  containing the compatible input-output port couples denotes the compatibility relation

Two modules can be connected as long as the output port of the one is compatible with the input port of the other.

## V. AUTOMATED PLANNING OF MOTION TASKS

### A. The Automated Module Composition Framework

The problem of automated planning of motion tasks we are considering, can be posed as a special case of the automated module composition problem [9], that is decidable and can be solved in polynomial time. The solution — if one exists — is non-unique in the general case and is optimal with respect to a given linear objective function representing a cost over the number of modules used. This can be used to express the fact that some modules might be more costly to implement than others.

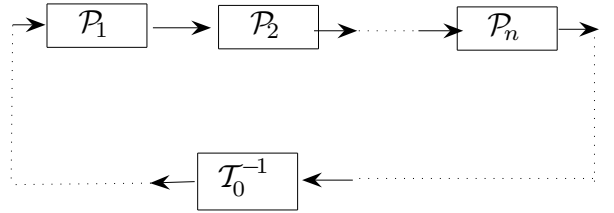


Fig. 1. Composition of  $n$  modules with the inverted motion task module to form a closed chain

### B. Formulation as an integer programming problem

A motion task is described by the initial and final states of the system<sup>1</sup>. Let  $X_0$  and  $X_f$  be the initial and final states of the system. We compose the motion task module as

$$\mathcal{T}_0 = \{\beta_{out}(X_0), \beta_{in}(X_f)\}$$

. Let  $\mathcal{M} = \{\mathcal{T}_0^{-1}, \mathcal{P}_1, \dots, \mathcal{P}_n\}$  be the set of the available modules, where  $\mathcal{T}_0^{-1}$  is the inverted motion task module where its input became its output and vice versa. Modules  $\mathcal{P}_i, i \in 1 \dots n$  are the available primitive motion task modules.  $\mathcal{M}_k$  denotes the  $k$ 'th element of  $\mathcal{M}$ . The module composition problem then is to create the smallest possible closed chain of connected modules, that contains the module  $\mathcal{T}_0^{-1}$  (see figure 1). Since the considered modules have only single input single output ports, no internal looping is possible, hence the eventual solution having the minimum possible number of modules, will contain maximum one instance of each module. Let  $x_j, j \in 0 \dots n$  denote the number of instances of module  $\mathcal{M}_j$ . The module composition problem can then be posed as the following integer programming problem:

$$\min \sum_{i=0}^n c_i x_i \quad (5)$$

$$x_0 = 1, x_j \in \{0, 1\}, j \in 1, \dots, n \quad (6a)$$

$$\sum_{In_i=p} x_i = \sum_{q \in P_{out} : \{p, q\} \in C} w_{p, q}, \forall p \in P_{in} \quad (6b)$$

$$\sum_{Out_j=q} x_j = \sum_{p \in P_{in} : \{p, q\} \in C} w_{p, q}, \forall q \in P_{out} \quad (6c)$$

<sup>1</sup>In practice only the final state is specified since the initial state is the current state that can be read out from the sensors

where  $c_i$  the cost of implementing module  $i$ , the sets  $P_{in}, P_{out}$  are the sets of input and, output ports resp., and  $w_{p,q}$  the number of connections between a copy of port  $p$  and a copy of port  $q$ . The integer programming problem (5) with the constraints (6) can be reduced to the well known combinatorial optimization problem of shortest dipath [11], in the following way: Let module

$$S = \{\text{in}(\mathcal{T}_0^{-1}), \emptyset\} \quad (7)$$

and module

$$F = \{\emptyset, \text{out}(\mathcal{T}_0^{-1})\} \quad (8)$$

where the operations  $\text{in}(\cdot)$  and  $\text{out}(\cdot)$  return the input and output port of module  $(\cdot)$ , and create the set of modules

$$\mathcal{M}' = \{S, F, \mathcal{P}_1, \dots, \mathcal{P}_n\} \quad (9)$$

. For each compatible pair  $c_k \in C, k = 1 \dots |C|$  create the set of module pairs whose I/O ports fulfil the compatibility relation  $c_k$ :

$$A_k = \{\{\mathcal{M}'_i, \mathcal{M}'_j\} : \text{out}(\mathcal{M}'_i) = \text{out}(c_k) \wedge \text{in}(\mathcal{M}'_j) = \text{in}(c_k), i \neq j\}$$

. Let

$$A = \bigcup_{k=1 \dots |C|} A_k \quad (10)$$

. Define the digraph

$$D = (\mathcal{M}', A) \quad (11)$$

and the length function

$$l(v_i v_j \in A) = c_i \in \mathbb{Q}_+$$

. Using  $S$  as the start node and  $F$  as the final node, find the shortest dipath  $S \rightarrow F$  in  $D$ . The shortest dipath problem can be solved in polynomial time by simple algorithms e.g. the one by Dijkstra [12], [13] which solves the problem in  $O(n^2)$ .

## VI. SIMULATIONS

To verify the feasibility of the proposed methodology we have set up a simulation of a micro-robotic platform with 5 micro-robots whose kinematics are described by (1) and two workspace features (see figure 2): a biological cell solution station denoted by  $Sol$  and a cell fixation station denoted by  $Fix$ . Those two features acted as obstacles to the robot team. A third feature of the workspace is the biological cell on which various experiments are carried out. In Table I we list the capabilities of each robot.

Robot \ Capability	Camera	$\mu$ -Syringe	Field Trap	SPM Tip
$R_a$	-	-	X	-
$R_b$	-	-	-	X
$R_c$	-	X	-	-
$R_d$	X	-	-	-
$R_e$	X	-	-	-

TABLE I  
ROBOT CAPABILITIES

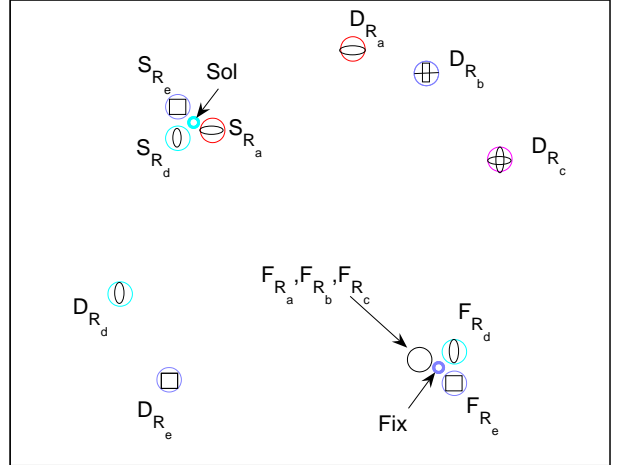


Fig. 2. Workspace features and Robot locations

In Table II listed are some of the properties of the primitive motion task modules. For all the primitive task modules the range of the  $G$  controller was the robot workspace, hence the compatibility conditions over the guard sets were satisfied. The robot workspace is shown in figure 2. The default location for robot  $R_x, x \in \{a, b, c, d, e\}$  is denoted by  $D_{R_x}$ . In general this is the place that a robot returns after finishing its job. The cell solution station positions for the electric field trap robot  $R_a$  as well as the camera robots  $R_d, R_e$  are denoted as  $S_{R_a}, S_{R_d}, S_{R_e}$  resp. The fixation station robot positions are denoted by  $F_{R_x}$ . All robot tools except the camera (which is always on) are assumed to have a  $\{0, 1\}$  state (on/off). The state of the workspace is denoted by the status of the biological cell  $\{Default, Fixed, Examined, Injected, Fixed + Examined, Fixed + Injected, Fixed + Examined + Injected\}$ . The discrete states of the hybrid system are thus composed from the predefined robot locations, the tool states and the workspace's features states.

In Table II, where locations are not mentioned, we assume default locations.  $x_i$  is the position of robot  $i \in \{a, b, c, d, e\}$ . For the purposes of this simulation we included a small database of 10 modules. However sufficiently large module databases can be handled efficiently under the proposed framework. Module  $\mathcal{P}_1$  moves the field trap robot and camera robots to their cell solution stations.  $\mathcal{P}_2$  activates the field trap to grasp a cell. Predicate  $C$  is the camera cell tracking controller. Module  $\mathcal{P}_3$  moves the field trap robot along with the camera robots in formation specified by the  $T_{d,c}^A$  controller predicate to the fixation station. Module  $\mathcal{P}_4$  fixes the cell at the fixation station.  $\mathcal{P}_5$  moves the SPM tip robot along with the camera robots to the fixation station.  $\mathcal{P}_6$  performs the cell examination.  $\mathcal{P}_7$  moves the micro-syringe robot along with the camera robots to the fixation station.  $\mathcal{P}_8$  performs the cell injection.  $\mathcal{P}_9$  moves the micro-syringe robot along with the camera robots to the fixation station if the cell has been previously examined.  $\mathcal{P}_{10}$  performs the cell injection if the cell has been previously examined.

The compatibility relation set  $C$  created by the given

	$\varphi$	$X_0^d$	$X_{ff}^d$
$\mathcal{P}_1$	$\square G$		$x_a \in S_{R_a},$ $x_d \in S_{R_d}$ $x_e \in S_{R_e}$
$\mathcal{P}_2$	$\square G \wedge C$	$x_a \in S_{R_a}$ $x_d \in S_{R_d}$ $x_e \in S_{R_e}$	$FT=1$ $x_a \in S_{R_a}$ $x_d \in S_{R_d}$ $x_e \in S_{R_e}$
$\mathcal{P}_3$	$\square G \wedge T_{d,e}^a \wedge C$	$FT=1$	$FT=1$ $x_a \in F_{R_a}$ $x_d \in F_{R_d}$ $x_e \in F_{R_e}$
$\mathcal{P}_4$	$\square G \wedge C$	$FT=1$ $x_a \in F_{R_a}$ $x_d \in F_{R_d}$ $x_e \in F_{R_e}$	$x_a \in F_{R_a}$ $x_d \in F_{R_d}$ $x_e \in F_{R_e}$ $CellFixed=1$
$\mathcal{P}_5$	$\square G \wedge C$	$CellFixed=1$	$x_b \in F_{R_b}$ $x_d \in F_{R_d}$ $x_e \in F_{R_e}$ $CellFixed=1$
$\mathcal{P}_6$	$\square G \wedge C$	$x_b \in F_{R_b}$ $x_d \in F_{R_d}$ $x_e \in F_{R_e}$ $CellFixed=1$	$x_b \in F_{R_b}$ $x_d \in F_{R_d}$ $x_e \in F_{R_e}$ $CellFixed=1$ $CellExamined=1$
$\mathcal{P}_7$	$\square G \wedge C$	$CellFixed=1$	$x_c \in F_{R_c}$ $x_d \in F_{R_d}$ $x_e \in F_{R_e}$ $CellFixed=1$
$\mathcal{P}_8$	$\square G \wedge C$	$x_c \in F_{R_c}$ $x_d \in F_{R_d}$ $x_e \in F_{R_e}$ $CellFixed=1$	$x_c \in F_{R_c}$ $x_d \in F_{R_d}$ $x_e \in F_{R_e}$ $CellFixed=1$ $CellInjected=1$
$\mathcal{P}_9$	$\square G \wedge C$	$CellFixed=1$ $CellExamined=1$	$x_c \in F_{R_c}$ $x_d \in F_{R_d}$ $x_e \in F_{R_e}$ $CellFixed=1$ $CellExamined=1$
$\mathcal{P}_{10}$	$\square G \wedge C$	$x_c \in F_{R_c}$ $x_d \in F_{R_d}$ $x_e \in F_{R_e}$ $CellFixed=1$ $CellExamined=1$	$x_c \in F_{R_c}$ $x_d \in F_{R_d}$ $x_e \in F_{R_e}$ $CellFixed=1$ $CellInjected=1$ $CellExamined=1$

TABLE II  
SEVERAL PROPERTIES OF PRIMITIVE MOTION TASK MODULES

primitive modules database is shown schematically in the digraph of figure 3 where a directed path from node  $j$  to node  $i$  exists iff  $(pin(\mathcal{P}_i), pout(\mathcal{P}_j)) \in C$ .

We set the task specification as follows:

- Initial Conditions: Default
- Final Conditions:
  - cell status: Fixed+Examined+Injected
  - tool status: Default
  - Robot locations: Anywhere

and create the motion task module  $\mathcal{T}_0$  with  $in(\mathcal{T}_0) = \beta(FinalConditions)$  and  $out(\mathcal{T}_0) = \beta(InitialConditions)$ . From  $\mathcal{T}_0$  we create the  $S$  and  $F$  modules (equations (7) and (8)) and then the  $\mathcal{M}'$  module from eq. (9). From eq. (10) we create the edge set and from eq. (11) we create the digraph  $D$ . The solution to the shortest dipath from  $S$  to  $F$  in  $D$  is then given by the

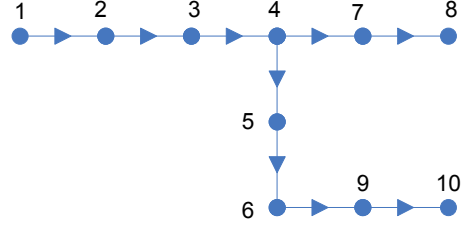


Fig. 3. Digraph representing the  $C$  set of the example

module sequence:

$$\{\mathcal{P}_1 \mathcal{P}_2 \mathcal{P}_3 \mathcal{P}_4 \mathcal{P}_5 \mathcal{P}_6 \mathcal{P}_9 \mathcal{P}_{10}\}$$

Figure 4 depicts the trajectories of the system. We can see that the primitive task module composition yields a system that satisfies the motion task specifications given in the abstract level of task planning, while the primitive task modules satisfy the LTL specifications at the level of motion planning. The system demonstrates safety and liveness properties by successfully avoiding collisions and deadlocks and converges to the specified sets.

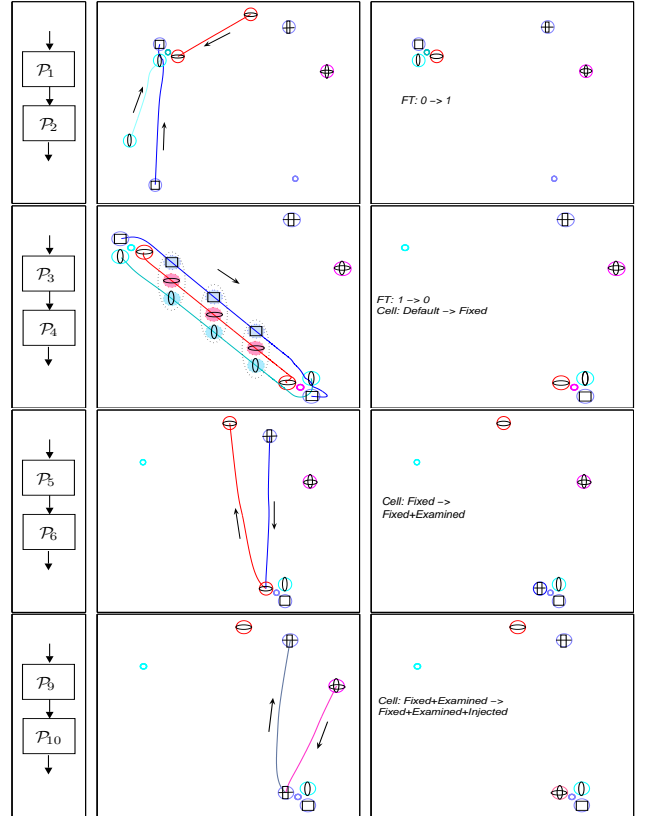


Fig. 4. Simulation

## VII. CONCLUSIONS

A methodology to automatically compose motion tasks, capturing both the task planning and the motion planning composition levels was presented. At the motion planning level, Multi-Robot Navigation Function controllers constructed from Linear Temporal Logic specifications dictate

the evolution of the system, while the task planning problem is posed as a composition of primitive task modules which is reduced to the shortest dipath problem. The multi-robot system is formally described under the hybrid systems framework capturing both the continuous and discrete aspects of motion tasks

Future work includes applying our methodology to automated fault handling, i.e. automatic reevaluation of a task plan in the case of a hardware failure in some robot, extending our work to decentralized setups and applying it to large scale robotic swarms.

#### REFERENCES

- [1] C. Kube and H. Zhang, "Task modelling in collective robotics," *Autonomous Robots*, vol. 4, pp. 53–72, 1997.
- [2] B. Gerkey and M. Mataric, "A framework for studying multi-robot task allocation," in *Multi-Robot Systems: From Swarms to Intelligent Automata*, A. S. et al, Ed. Kluwer Academic Publishers, 2003, vol. II, pp. 15–26.
- [3] F. Lian and R. Murray, "Cooperative task planning of multi-robot systems with temporal constraints," *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 2504–2509, 2003.
- [4] J. Sousa, T. Simsek, and P. Varaiya, "Task planning and execution for uav teams," *43rd IEEE CDC Conf. on Decision and Control*, pp. 3804–3810, 2004.
- [5] E. Guéré and R. Alami, "Let's reduce the gap between task planning and motion planning," *IEEE Int. Conf. on Robotics and Automation*, pp. 15–20, 2001.
- [6] S. Loizou and K. Kyriakopoulos, "Automatic synthesis of multi-agent motion tasks based on ltl specifications," *43rd IEEE Conference on Decision and Control*, pp. 153–158, 2004.
- [7] S. G. Loizou and K. J. Kyriakopoulos, "Closed loop navigation for multiple holonomic vehicles," *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 2861–2866, 2002.
- [8] S. Loizou and K. Kyriakopoulos, "Multi-robot navigation functions," *submitted*, 2004.
- [9] S. Tripakis, "Automated module composition," *Int. Conf. on Tools and Algor. for Constr. and Analysis of Systems*, 2003.
- [10] G. J. Pappas, "Hybrid systems: Computation and abstraction," Ph.D. dissertation, Univ.Carifornia at Berkeley, 1998.
- [11] M. Grötschel, L. Lovász, and A. Schrijver, *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag, 1988.
- [12] E. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik 1*, pp. 269–271, 1959.
- [13] R. Ahuja, T. Magnati, and J. Orlin, *Network Flows - Theory Algorithms and Applications*. Prentice-Hall, 1993.