Controller Synthesis for ROS-based Multi-Robot Collaboration

Xudong Zhao[1], Rui Li[1], Wanwei Liu[1], Hao Shi[1], Shaoxian Shu[2], Wei Dong[1]
[1]*College of Computer Science, National University of Defense Technology, Changsha, China*
[2]*Hunan Institute of Traffic Engineering, Changsha, China*
{zhaoxudong13, lirui18, wwliu, shihao14}@nudt.edu.cn, shushaoxian@163.com, wdong@nudt.edu.cn

***Abstract*—Given a multi-robot system and the high-level tasks for the robots. How to ensure the correct behavior of robots to complete their tasks is critical. In this paper, we design a framework that can automatically generate correct-by-construction controllers for multi-robot system. In this framework, we propose a multi-robot specification based on the Temporal Logic Synthesis Format (TLSF) to guarantee the robots' behavior. And two execution algorithms were proposed to abstract the synthesized automata into high-level controllers. These controllers were integrated into Robot Operating System (ROS) to control actions and movements of robots in Gazebo, which is an open-source 3D robotics simulator. Based on this framework, we developed a toolkit, which allows users to achieve system-level controller synthesis and simulation capable to be used for research such as mission planning, motion planning, automatic obstacle avoidance and so on.**

***Index Terms*—multi-robot system, robot operating system, temporal logic synthesis format, controller synthesis**

I. INTRODUCTION

In the past few years, the study of autonomous robots has become increasingly appealing. Robots have been employed in many application domains. Especially in the fight against the novel coronavirus pneumonia that broke out in early 2020, robots are widely used in various applications include food delivery, medicine delivery, temperature measurement, disinfection, etc. Robots effectively replace humans for operation, reducing the possibility of cross-infection. People increasingly realize the importance and convenience of robots in human life.

Multi-robot systems (MRS) is defined as a group of robots coordinated to perform some complex tasks that cannot be completed by a single robot [1]. Therefore, in some particular scenarios, we usually need MRS to complete tasks. However, in practice, most application scenarios are uncertain and dynamic, robots may behave unexpectedly in these scenarios. How to ensure the correctness of robot behavior in these dynamic environments is critical.

Based on these considerations, the researchers used formal methods such as model-checking [2] and synthesis to ensure the correctness of the robot's behavior. One important approach is to use Linear Temporal Logic (LTL) [3] as the specification to generate controllers of robots [4] [5]. Extensive research has been carried out on how to translate the high-level specifications into robot controllers. Finucane et al. developed a toolkit called LTLMoP [6]. It can translate the user-written LTL specification into a robot controller, and the controller can be executed to simulate the behavior of the robot in a two-dimension area. Their work successfully bridges the gap between high-level specifications and low-level robot controllers.

However, this tool only provides controller synthesis and simulation for a single robot. To apply these studies in multi-robot scenarios, we extend their work to MRS. In this paper, we introduce a framework that automatically translates the multi-robot specification based on the Temporal Logic Synthesis Format (TLSF) [7] into high-level controllers for the multi-robot system. These controllers can be integrated into ROS [8], which is an opensource software architecture that contains a variety of libraries and packages suitable for robots. Then these controllers were executed by appropriate packages and tools based on ROS. To better demonstrate the experimental results, the behavior of robots is simulated in Gazebo [9].

In order to prove the practicality of this framework, we developed a toolkit for designing, executing, and simulating multi-robot controllers generated automatically from the multi-robot specification. Our contributions include the following three aspects:

- First, the framework we proposed provides a complete development process for multi-robot collaborative tasks, including automata synthesis, generation and execution of controllers.
- Second, we propose two algorithms that can more efficiently abstract the synthesized automata into high-level controllers.
- Third, we achieve system-level simulation in Gazebo with ROS, which makes it easier to experiment with physical robots.

The rest part of this paper is structured as follows: Section II summarizes the theoretical basis of this paper; In section III, we elaborate on the various components of this framework; Then we present an example of collaborative task in section IV, which briefly introduce how to use our toolkit; We conclude this paper in section V.

Corresponding author: Wei Dong. This work was supported by National Natural Science Foundation of China (No.61690203, No.61532007) and National Key Reseach and Development Program of China (No.2017YFB1001802).

DOI reference number: 10.18293/SEKE2020-082

II. PRELIMINARIES

A. Linear Temporal Logic

Linear Temporal Logic (LTL) is a modal temporal logic which has been widely used to model the change of a reactive system over time.

LTL Syntax. Let AP be a set of atomic propostions with temporal logic $\mathtt{X}$ (`next`) and $\mathtt{U}$ (`until`), where $p \in AP$ is a Boolean variable. LTL formulas are defined according to the follwing grammar:

$$\varphi \ ::= \ p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathtt{X}\varphi \mid \varphi \mathtt{U} \varphi$$

LTL Semantics. Semantics of an LTL formula φ are defined on an infinite sequences $\pi = \pi_1\pi_2\cdots$ of truth assignment to the atomic propostions $p \in AP$, where $\pi(i)$ denote the i-th element of π and $\pi(i) \in 2^{AP}$. The satisfaction relationship $\models$ between π, i and a LTL formula φ is defined as follows:

$$\begin{aligned}
&\pi, i \models p && iff \quad p \in \pi(i) \\
&\pi, i \models \neg\varphi && iff \quad \pi, i \not\models \varphi \\
&\pi, i \models \varphi_1 \vee \varphi_2 && iff \quad \pi, i \models \varphi_1 \ or \ \pi, i \models \varphi_2 \\
&\pi, i \models \mathtt{X}\varphi && iff \quad \pi, i+1 \models \varphi \\
&\pi, i \models \varphi_1 \mathtt{U} \varphi_2 && iff \quad \exists k \geqslant i \ with \ \pi, k \models \varphi_2 \ and \\
& && \quad \forall i \leqslant j < k \ with \ \pi, j \models \varphi_1)
\end{aligned}$$

The formula $\mathtt{X}\varphi$ means that φ is true in the next position of the sequence. $\varphi_1 \mathtt{U} \varphi_2$ indicates that φ_2 will be true somewhere in the future, and φ_1 must be maintained as true until φ_2 is true.

The sequence π satisfies formula φ if $\pi, 0 \models \varphi$. The temporal operators of LTL $\mathtt{G}$(`always`), $\mathtt{F}$(`eventually`):

- $\mathtt{F}\varphi \equiv \mathtt{true}\mathtt{U}\varphi$;
- $\mathtt{G}\varphi \equiv \neg\mathtt{F}\neg\varphi$;

Where $\mathtt{G}\varphi$ with *always* and $\mathtt{F}\varphi$ with *eventually* express the properties that φ will always hold true in every position of the sequence and φ will be true at some position of the sequence in the future respectively. Further, $\mathtt{GF}\varphi$ indicates that φ is `true` infinitely often.

B. General Reactivity(GR(1))

LTL formulas are particularly suited to model the evolution of a reactive system where atomic propositions can be divided into two parts: system input (environment) and output (system). However, the realizability of LTL is 2-EXPTIME-complete, which increases computational overhead.

To reduce the computational complexity into an acceptable range, we consider a special class of temporal logic formulas [10]. The GR(1) fragment of LTL specification consists of environment assumptions and system guarantees. A GR(1) synthesis problem is defined as a game between a system player and an environment player. We expect the system wins the game. In other words, we can always synthesize controllers that generate behavior strategies satisfying given specifications. A GR(1) game structure is organized as follows:

- $\mathcal{X}$ is the set of input variables controlled by environment, $\mathcal{X}'$ is the value of $\mathcal{X}$ in the next state;
- $\mathcal{Y}$ is the set of output variables controlled by system, $\mathcal{Y}'$ is the value of $\mathcal{Y}$ in the next state;
- θ^e is an assertion over $\mathcal{X}$ characterizing the initial states of the environment;
- θ^s characterizes an assertion over $\mathcal{X} \cup \mathcal{Y}$ characterizing the initial states of the system;
- ρ^e characterizes transition relation of the environment over $\mathcal{X} \cup \mathcal{Y} \cup \mathcal{X}'$;
- ρ^s characterizes transition relation of the system over $\mathcal{X} \cup \mathcal{Y} \cup \mathcal{X}' \cup \mathcal{Y}'$;
- $\mathcal{J}^e_{i \in 1..m}$ is a set of justice requirements of the environment;
- $\mathcal{J}^s_{j \in 1..n}$ is a set of justice requirements of the system;

The acceptance condition is finally defined as:

$$(\theta^e \wedge \mathtt{G}\rho^e \wedge \mathtt{GF}\mathcal{J}^e) \rightarrow (\theta^s \wedge \mathtt{G}\rho^s \wedge \mathtt{GF}\mathcal{J}^s)$$

where $\mathtt{G}\rho^e$ and $\mathtt{G}\rho^s$ are safety conditions over the environment and the system while $\mathtt{GF}\mathcal{J}^e$ and $\mathtt{GF}\mathcal{J}^s$ are liveness properties over the environment and the system.

C. Robot Operating System (ROS)

ROS [8] is a framework for robotics research and development. The core of ROS is communication mechanism. ROS is a peer-to-peer network of nodes that communicate with each other using custom ROS messages that are based on TCP/IP. Each node can be used to control the behavior of the robot, process the information obtained by the sensors, etc. In an MRS, we regard each robot as a node, and all nodes are connected to a ROS master. Through the master, each node can locate and communicate with other nodes by three different methods:

(1) Topic: For real-time and periodic messages, the topic is the best choice. The node that subscribes to messages from a topic is called the topic's subscriber, while the node that publishes messages to a topic is called the topic's publisher.
(2) Service: Service communication is two-way. It can send messages and return feedback. The service consists of two parts: the requester (Client) and the responder/service provider (Server). The client sends a request, waits for the server to process it and returns a reply. The entire service communication is completed through a "request-response" mechanism.
(3) Actionlib: Actionlib is used to execute a long-term communication process. The actionlib communication process can be viewed at any time, and the request can be terminated. Actionlib works in client-server mode and is a two-way communication mode.

The ROS ecosystem includes some tools to analyze and simulate robot behavior. Gazebo [9] is a three-dimensional physics simulation platform with a powerful physics engine, high-quality graphics rendering, convenient programming, and graphical interfaces. Gazebo can add the physical properties of the robot and the surrounding environment to the model, such as mass, coefficient of friction, coefficient of elasticity,

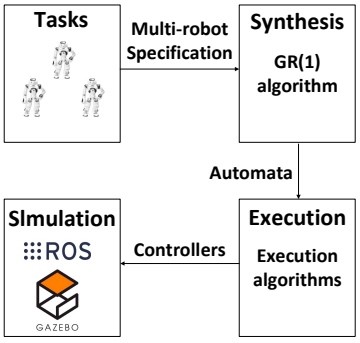


Fig. 1: Overview of the framework

etc. Therefore, we can simulate physical phenomena in the real world and show them as much as possible in this simulation environment.

III. CONTROLLER SYNTHESIS AND SIMULATION FOR MRS

Given a multi-robot system and tasks to be completed by the robots, the objective of our framework is to automatically generate controllers for each robot and simulate the behavior of robots in Gazebo. The framework consists of four parts, the relationship between them is illustrated in Figure 1 and the functions of them are introduced as follows.

A. Multi-robot specification

TLSF [7] is a high-level format for the specification of synthesis problems. Compared with LTL, it is more readable, therefore, users can easily write and read expressive specifications. Another advantage of TLSF is that it's easy to support by synthesis tools. After writing the specification, the $Synthesis\ Format\ Conversion\ Tool$ (SyFCo) can compile TLSF specifications into LTL specifications.

In a multi-robot system, a robot regards other robots as part of the environment. A robot's environment propositions can be sensed by its sensors and obtained by communicating with other robots. To express the robot's perception ability, execution ability and communication relationship with other robots, a tuple $\mathcal{R} = \langle S, A, C \rangle$ for each robot in MRS was defined:

- S is the environment variables gained by robot;
- A is the action variables performed by the robot's actuators;
- C is the communication node that are used to communicate with each other through network.

Based on the advantages of TLSF and the tuple we difined, we propose a multi-robot specification. Take two robots as an example, where:

$R_1 = \langle \{R2.act2\}, \{act1\}, node_1 \rangle$

$R_2 = \langle \{R1.act1\}, \{act2, act3\}, node_2 \rangle$

The specification for these two robots are shown in Figure 2. Keywords env (environment), sys (system) are set of input variables and output variables respectively and keywords asm (assumption), gar (guarantee) characterize initial conditions, transition relations and justice requirements for environment and system respectively.

```
main R1 {
  env { R2.act2;}
  sys { act1;}
  asm { GF(R2.act2);}
  gar { G(R2.act2->X(act1));}}

main R2 {
  env { R1.act1;}
  sys { act2,act3;}
  asm { GF(R1.act1);}
  gar { G(R1.act1->X(act3));}}
```

Fig. 2: An example of multi-robot specification

The users could write the corresponding specification for each robot in MRS according to the multi-robot specification format we proposed. Based on the capabilities of the robot and the collaborative task to be completed, the continuous behavior of the robot should be abstracted into a finite set of propositions by a discrete formalism (TLSF). These propositions consist of the sensor information the robot perceives, the actions to be performed. The specification also includes the topological information of the robot's task area. Same as a single robot, in an MRS, each robot has a corresponding specification and its automaton that synthesized from these specifications.

B. Specification to Automaton

Take two robots as an example, Figure 3 illustrates the process of synthesizing, as mentioned in section II, we use GR(1) as the synthesis algorithm and automatically synthesize automaton from the multi-robot specification by a tool called JTLV [11].

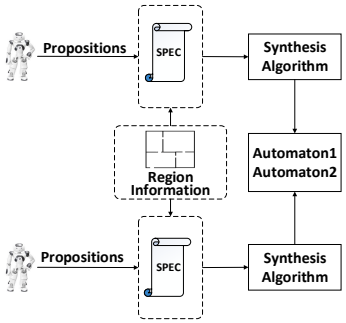


Fig. 3: The synthesis of automaton

To more intuitively explain the process of synthesizing specification into an automaton, the following briefly introduces the synthesis algorithm in [10].

As mentioned before, the synthesis algorithm was used to solve the game between the robot and the environment. Consider a game structure G: $\langle \mathcal{X}, \mathcal{Y}, \theta^e, \theta^s, \rho^e, \rho^s, \mathcal{J}^e, \mathcal{J}^s \rangle$, the initial state of robot and environment is $s_{\mathcal{X} \cup \mathcal{Y}}$ where $s_{\mathcal{X} \cup \mathcal{Y}} \models \theta_e \wedge \theta_s$. Then from the initial state, both the robot and the environment make decisions that determine their

next states, the environment choose an input $s_{\mathcal{X}'}$ such that $(s, s_{\mathcal{X}'}) \models \rho^e$ and the system choose an output $s_{\mathcal{Y}'}$ such that $(s, s_{\mathcal{X}'}, s_{\mathcal{Y}'}) \models \rho^s$. The winning condition for the game is given as a GR(1) formula $\phi = (\texttt{GF}\mathcal{J}^e \rightarrow \texttt{GF}\mathcal{J}^s)$, the implication between justice goals J^e of the environment and J^s of the robot. In other words, no matter what the environment does, the robot can always find a way to proceed and satisfy the GR(1) formula ϕ, we say that the robot is winning and an automaton can be synthesized from the specification. Otherwise, we say that the environment is winning and the specification is unrealizable. Once the task specification of the robot is realizable, the synthesis algorithm is to find a winning strategy that the robot should follow to complete the desired task.

The strategy synthesised by the algorithm can be viewd as an automaton $\mathcal{A} = (\mathcal{X}, \mathcal{Y}, \mathcal{Q}, Q_0, \gamma, \delta)$:

- $\mathcal{X}$ is the set of input (environment) propositions,
- $\mathcal{Y}$ is the set of output (robot) propositions,
- $\mathcal{Q}$ is the set of states,
- $\mathcal{Q}_0 \subset \mathcal{Q}$ is the set of initial states,
- $\gamma : \mathcal{Q} \rightarrow 2^{\mathcal{X} \cup \mathcal{Y}}$ is the state labeling function where $\gamma(q)$ is the set of robot propositions and input propositions that are true in state q, i.e., states hold the environment inputs.
- $\delta : \mathcal{Q} \rightarrow 2^{\mathcal{Q}}$ is the transition relation. If current state is q and at next point environment inputs is $s_{\mathcal{X}}$, then q' is the successor state of q if and only if $s_{\mathcal{X}} \models \gamma(q')$.

A run of a strategy is sequence $s = s^0_{\mathcal{X}}, s^0_{\mathcal{Y}}, s^1_{\mathcal{X}}, s^1_{\mathcal{Y}}, \cdots$, s.t. $\forall i : \big((q_i, s^i_{\mathcal{X}}, q_{i+1}) \models \delta\big) \wedge \big(s^i_{\mathcal{Y}} = \gamma(q_i)\big) \wedge \big(s^{i+1}_{\mathcal{Y}} = \gamma(q_{i+1})\big)$. Based on this sequence, the discrete path of the robot can be acquired which guides the robot to choose a region to go or activate/deactivate the different robot actions.

C. Automaton to Controller

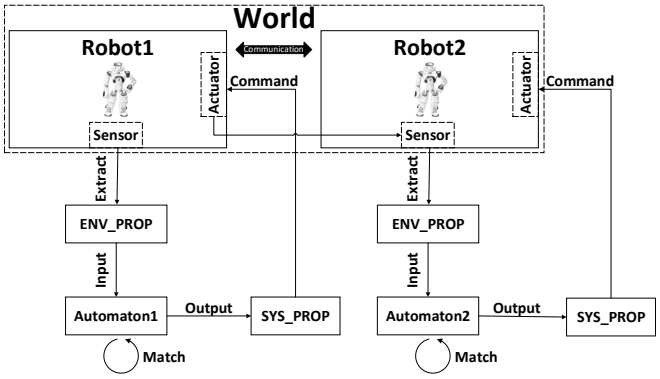


Fig. 4: Turn automatons into controllers

We introduce that the continuous behavior of the robot is abstracted into a discrete specification and then synthesized into an automaton. This part introduces how to turn these discrete automata into continuous controllers.

As shown in figure 4, an automaton is actually a finite state machine (FSM). Different states include the robot's perception of dynamic environments and the actions that the robot should perform. We propose an algorithm to deal with environment propositions, which are gained by the sensors of robots and the communication between robots. Furthermore, based on the Boolean value of these propositions, this algorithm can match the corresponding state in the automaton.

Algorithm 1: Discrete automaton to continuous controllers

```
Input: Automaton A
CurrState ← q_0 ∈ Q_0
SuccStateSet ← δ(q_0)
Actions ← {a_1, a_2, ···} ∈ γ(q_0)
Execute(Actions)
while True do
    InputVal ← SenseOrInformed();
    FoundState ← False;
    foreach q_i ∈ SuccStateSet do
        if InputVal ⊨ γ(q_i) then
            NextState ← q_i;
            Execution(CurrState, NextState);
            CurrState ← NextState;
            SuccStateSet ← δ(CurrState);
            FoundState ← True;
            BREAK;
        end
    end
    if Foundstate is False then
        ERROR('Invalid Input')
    end
end
```

As shown in Algorithm 1, in the beginning, the robot is in its initial state, which is defined as the current state. According to this state, the robot executes current actions and get the successor states set. In each step, the robot obtains its environmental information and determines input values through its sensors or by communicating with other robots. Based on these inputs and current state, we can get the $NextState$ in the successor states set. The robot then executes the actions and complete the state transition, the execution algorithm is illustrated in Algorithm 2. When the next state is not found in the set of successor states, which means that the environment violated its assumptions, the execution will report an error and stop running.

When it comes to the cross-regional problem, we propose some simple controllers. Generally, we define those control robot's actuators as execute controllers, which can complete execution in a short time, while those control robot's motion are navigation controllers. Compared with the former, navigation controllers take a longer time to complete execution. The process of executing these controllers is shown in Algorithm 2. Take a transition between states as an example, when the robot is in different regions between the current state and the next state, the navigation controller should be executed first, and then the execute controller. That is, the robot will only execute the action when it reaches the designated region.

Robot movement is achieved through the navigation package in ROS. When both states are in the same region, the action is executed directly.

Algorithm 2: Execution

Input: Current state q_i and Next State q_{i+1}
$CurrState \leftarrow q_i$
$CurrRegion \leftarrow r_i \in \gamma(q_i)$
$NextState \leftarrow q_{i+1}$
$NextRegion \leftarrow r_i \in \gamma(q_{i+1})$
$Actions \leftarrow \{a_1, a_2, \cdots\} \in \gamma(q_{i+1})$
if $NextRegion \neq CurrRegion$ **then**
  $Navigation\_Controller(NextRegion)$;
  $Execute\_Controller(Actions)$;
else
  $Execute\_Controller(Actions)$;
end

D. Simulation

To simulate multi-robot missions in Gazebo, the experimental environment must be configured in advance. The various components include the world model, map information, navigation packages, and communication method for configuration are elaborated as follows. To minimize the user's workload as much as possible, the corresponding navigation packages files and the program for communication between robots will be generated automatically according to the number of robots.

1) World: As shown in figure 4, the environment that the robot simulates in Gazebo is called world. As the name indicates, the world file is to simulate a real physical world. It can simulate physical parameters, such as gravity, friction coefficient, elastic coefficient, etc. For the tasks to be completed by the MRS, the user could build a corresponding world model using Gazebo's building editor, which allows the user to import a floor plan and add walls, stairs, and doors relatively easily.

2) Map: After establishing the world model, we control the robot to move in the world and use laser radar or depth camera to generate laser data and convert it into the map. This is the troublesome part of the simulation process. Then we manually extract the topological information of the generated map.

3) Navigation: Navigation and localization are important parts of the robot's tasks. The robot performs navigation and localization according to the established map. There are two packages in ROS that can be used directly.

- $move\_base$: It can plan the global path according to a given target position, and also plan a local path based on nearby environments to avoid obstacles.
- $amcl$: It is a probabilistic localization system for a robot moving in two-dimensional area. It implements the adaptive Monte Carlo localization approach [12], which can get the position of the robot by using a particle filter against a known map.

IV. IMPLEMENTATION AND EXAMPLE

The toolkit we developed is modular and includes three modules. For different collaborative tasks, users can configure the number and propositions of robots to write multi-robot specification. Then the realizable specification can be synthesized into automata, finally, turn these automata into high-level controllers and simulate in Gazebo. For the sake of clarity, we introduce a robot collaboration experiment based on the home service scenario. As shown in figure 5, we design a workspace for robots. Then according to this sketch, we create a world model in Gazebo.

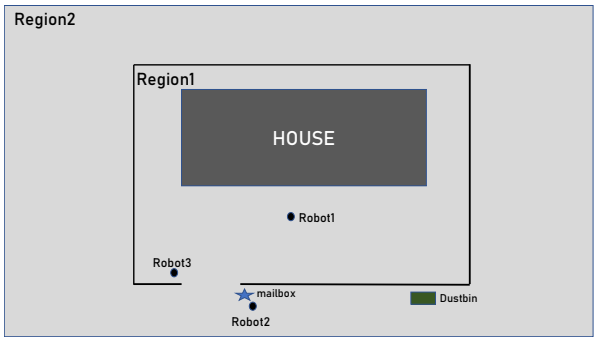


Fig. 5: Example diagram

In this example, $region_1$ is inside the yard while $region_2$ is the outside of the yard. Three home service robots work in these two areas, Among them, $Robot_1$ is mainly responsible for patrolling the $region_1$, and is also responsible for handling some chores in family life. For example, when there is mail in the mailbox, it will fetch it, and when waste is detected on the ground, it will pick it up and throw it into the dustbin in $region_2$. $Robot_2$ is responsible for delivering the mail. After delivering the mail, $Robot_1$ will be informed. $Robot_3$ is responsible for security tasks. When it detects that a thief is trying to break into the house, it will send a warning and inform $Robot_1$ to come and defend. At the same time, $robot_2$ is never allowed to enter $region_1$.

As defined in section III, we formalize this multi-robot system based on the capabilities of robots and the collaborative tasks, where:

$R_1 : \langle \{waste, R2.mail, R3.thief\}, \{pick, fetch, catch\}, node_1 \rangle$
$R_2 : \langle \{mailbox\}, \{deliver, informR1\_mail\}, node_2 \rangle$
$R_3 : \langle \{thief\}, \{catch, informR1\_thief\}, node_3 \rangle$

The multi-robot specification for robots is given in figure 6, where $R2.mail$ means R1 gets mail proposition by communicating with R2, so is $R3.thief$. After synthesizing these automata from the specification, we turn them into continuous control commands and simulate in Gazebo. There are six screenshots shown in figure 7.

V. CONCLUSION AND FUTURE WORK

In this paper, we propose a framework that can automatically generate correct-by-construction controllers for multi-robot system. To simulate the behavior of robots, these generated controllers were executed in Gazebo with ROS,

which provides state-of-the-art algorithms for several known problems in robotics. We develope a toolkit to implement this framework. Because ROS targets at real robots and not only 3D simulation, it is easier to embed these controllers for simulation into real robots. Compared to experiments using real robots, it is low-cost, convenient and extensible for research.

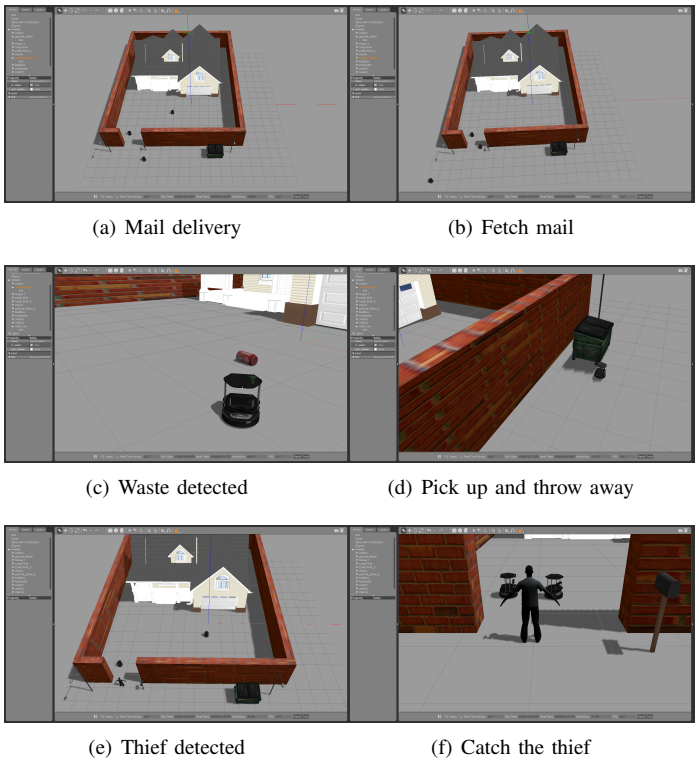
In future work, we plan to provide a user-friendly interface and experiment with real robots. Besides, we will combine methods in the field of robot control to extend the functionality of this framework as much as possible.

```
main R1{
  env {
    Waste;
    R2.Mail;
    R3.Thief;}
  sys {
    Pick;
    Catch;
    Patrol;
    Fetch;}
  asm {
    !Waste;
    !R2.Mail;
    !R3.Thief;}
  gar {
    !Pick;
    !CatchThief;
    !Patrol;
    !Fetch;
    GF Patrol;
    G(X Waste -> X Pick);
    G(X R2.Mail -> X Fetch);
    G(X R3.Thief -> X Catch);}}

main R2{
  env {
    Mailbox;}
  sys {
    Deliver;
    InformR1_mail;}
  asm {
    !Mailbox;}
  gar {
    !Deliver;
    !InformR1_mail;
    G(X Mailbox -> X Deliver);
    G(Deliver -> X InformR1_mail);
    G !Region1;}}

main R3{
  env {
    Thief;}
  sys {
    InformR1_Thief;
    CatchThief;}
  asm {
    !Thief;
    GF Thief;}
  gar {
    !InformR1_Thief;
    !CatchThief;
    G(X Thief -> X InformR1_Thief);
    G(X Thief -> X CatchThief);}}
```

Fig. 6: Specification for home service robots



(a) Mail delivery (b) Fetch mail

(c) Waste detected (d) Pick up and throw away

(e) Thief detected (f) Catch the thief

Fig. 7: Experimental run of the service robots scenario

REFERENCES

[1] A. Gautam and S. Mohan, "A review of research in multi-robot systems," in *2012 IEEE 7th International Conference on Industrial and Information Systems (ICIIS)*, Aug 2012, pp. 1–5.

[2] E. M. Clarke and B. Schlingloff, "Model checking," in *Handbook of Automated Reasoning (in 2 volumes)*, 2001, pp. 1635–1790.

[3] A. Pnueli, "The temporal logic of programs," in *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pp. 46–57.

[4] I. Filippidis, S. Dathathri, S. C. Livingston, N. Ozay, and R. M. Murray, "Control design for hybrid systems with tulip: The temporal logic planning toolbox," in *2016 IEEE Conference on Control Applications (CCA)*, Sep. 2016, pp. 1030–1041.

[5] H. Kress-Gazit, "Robot challenges: Toward development of verication and synthesis techniques [errata]," *IEEE Robot. Automat. Mag.*, vol. 18, no. 4, pp. 108–109, 2011.

[6] C. Finucane, G. Jing, and H. Kress-Gazit, "Ltlmop: Experimenting with language, temporal logic and robot control," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, October 18-22, 2010, Taipei, Taiwan*, pp. 1988–1993.

[7] S. Jacobs, F. Klein, and S. Schirmer, "A high-level LTL synthesis format: TLSF v1.1," in *Proceedings Fifth Workshop on Synthesis, SYNT@CAV 2016, Toronto, Canada, July 17-18, 2016*, pp. 112–132.

[8] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng, "Ros: an open-source robot operating system," vol. 3, 01 2009.

[9] N. P. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems, Sendai, Japan, September 28 - October 2, 2004*, pp. 2149–2154.

[10] N. Piterman, A. Pnueli, and Y. Sa'ar, "Synthesis of reactive(1) designs," in *Verification, Model Checking, and Abstract Interpretation, 7th International Conference, VMCAI 2006, Charleston, SC, USA, January 8-10, 2006, Proceedings*, pp. 364–380.

[11] A. Pnueli, Y. Sa'ar, and L. D. Zuck, "Jtlv: A framework for developing verification algorithms," in *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings*, pp. 171–174.

[12] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "Monte carlo localization for mobile robots," in *1999 IEEE International Conference on Robotics and Automation, Marriott Hotel, Renaissance Center, Detroit, Michigan, USA, May 10-15, 1999, Proceedings*, pp. 1322–1328.