Security Controller Synthesis for ROS-based Robot

Xudong Zhao[1], Shaoxian Shu[2], Yutian Lan[1], Haining Feng[1], Wei Dong[1]
[1]*College of Computer Science, National University of Defense Technology, Changsha, China*
[2]*Hunan Institute of Traffic Engineering, Changsha, China*
zhaoxudong13@nudt.edu.cn, shushaoxian@163.com, {lanyutian18, fenghaining18, wdong}@nudt.edu.cn

***Abstract*—In recent years, robots have been widely used in people's daily lives. While bringing convenience to us, there are also potential threats. To ensure the correctness of the robot's behavior, scientists have conducted extensive research on generating security controllers for robots. In this paper, we propose a method that divides the robot controllers into task controller and security controller, this method greatly reduces the time required for controller generation. We use the Robot Operating System (ROS) as our experimental platform. Due to the poor security of ROS, robots are vulnerable to cyber-attacks. Therefore, we summarized several types of cyber-attacks against ROS and generate security controllers for robots. Then we simulate the generated controllers in Gazebo. Experimental results show that our method can reduce the impact of cyber attacks on robots and ensure the successful completion of the task.**

***Index Terms*—robot operating system, controller synthesis, cyber attack, linear temporal logic**

I. INTRODUCTION

Since the advent of the first robot in 1958, the research of robot technology has developed rapidly. Robots are widely used in human life, bringing a lot of convenience to people. The robots are employed in industrial areas such as warehousing, logistics, and manufacturing, and they also employed in non-industrial areas such as hospitals, farms, and home services. At the same time, the widespread use of robots has brought corresponding risks, especially in some critical application areas. In May 2015, due to malware on the robotic platform, the assembly robot behaved abnormally and killed a worker in a car factory. U.S. military base in April 2016, a robot mortar gun killed nine soldiers during military training because the malware in the system caused the robot to malfunction [1].

Therefore, in the development of robotic technology, how to protect robots from cyber attacks has become a key field of research. Scientists have conducted extensive research on the cybersecurity of robots. Martin et al. [2] presented a survey of attack prediction methods used in cybersecurity. They summarized some cases of prediction in cybersecurity and categorized these methods by theoretical background. In addition to predicting cyber attacks, scientists also studied the automatic generation of security controllers for robots. Hadas et al. [3] use Linear Temporal Logic (LTL) [4] as the specification of robots to ensure their behavior.

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).

In our previous work, a framework was proposed to automatically generate correct-by-construction controllers for multi-robot systems (MRS) [5]. And then these controllers were integrated into ROS and control the robots to perform certain tasks in Gazebo. This work makes it easier for users to experiment with real robots. There are still many deficiencies in this work: 1) For some complex task scenarios, there are many states of the robot, so the synthesis of the controller takes a long time and sometimes it even cannot be synthesized. 2) ROS has no built-in security, therefore it's an easy target for cyber attackers.

In this paper, we introduced a method that can greatly reduce the time required for controller synthesis. Then we summarized the security problems in ROS, discussed two types of security problems, and proposed corresponding protective measures for them respectively. In order to prove the effectiveness of the proposed methods, we conducted two experiments and simulated one of them in Gazebo.

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 framework of controller synthesis for the robot; Then we present two examples in section IV, which briefly introduce the method we proposed can effectively protect the robot; We conclude this paper in section V.

II. PRELIMINARIES

A. Linear Temporal Logic

Linear Temporal Logic (LTL) [4] 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 sequence $\pi = \pi_1\pi_2\cdots$ of truth assignment to the atomic propostions $p \in AP$, where $\pi(i)$ denotes 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:

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

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 property 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 [6]. 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 [7] 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.

(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 [8] 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, 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 SEPARATION

Temporal Logic Synthesis Format (TLSF) [9] 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. Therefore we use TLSF as the specification of robots. Then these specifications were synthesized into controllers by a GR(1) game-based reactive synthesis algorithm in [6]. The process of synthesis is shown in figure 1.

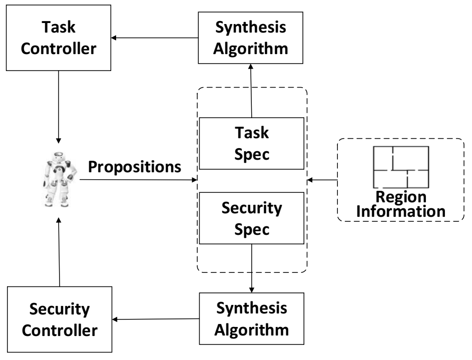


Fig. 1: Controller generation

A. Controller Synthesis

To more intuitively explain the process of synthesizing specification into controller, the following briefly introduces the synthesis algorithm in [6]. 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 $\phi = (\mathtt{GF}\mathcal{J}^e \rightarrow \mathtt{GF}\mathcal{J}^s)$, 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 the controller 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 $\mathcal{A} = (\mathcal{X}, \mathcal{Y}, \mathcal{Q}, \mathcal{Q}_0, \gamma, \delta)$:

- $\mathcal{X}$ 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.

B. Controller Separation

When the robot performs some complex tasks, it needs to do many actions in different areas, so it takes a long time to synthesize the corresponding controllers. In extreme cases, the controller cannot be synthesized. To solve this problem, we divide the robot controller into two parts, namely the task controller and the security controller, and these two types of controllers are loosely coupled. The task controller is used to control the robot to complete the specified task, and the security controller is used to ensure the safety of the robot's behavior.

The form of the controller we synthesized is actually a finite state machine (FSM), each state of the FSM includes input and output. For the task controller, the input is the environment changes perceived by the robot and the output is the action that should be performed by the robot. For the security controller, the input is the properties of the robot, such as the current position coordinates, battery power, memory, and other critical information that we care about, and the output is what the robot should do in response to different cyber attacks. Once the system detects that a certain safety property of the robot is violated, the security controller will immediately start and control the robot to perform the correct behavior.

IV. ROS-BASED SECURITY CONTROLLER

Although ROS is the most popular robot system in recent decades, its application is limited to research laboratories and has not been widely used in the industry. The reason is that there are many security issues in ROS. In this section, we introduced two kinds of safety issues and generated corresponding security controllers for them.

A. False Information

As mentioned in section II, robots can communicate with each other through master in ROS. However, a node can publish data for arbitrary topics without pre-authentication. This means that when the attacker and the robot are connected to the same network, the attacker can exploit this vulnerability and send wrong instructions to the robot, causing the robot to perform unexpected actions or move to dangerous areas. To prevent these accidents, we use TLSF as the specification of robots and synthesize controllers to control the robot's behavior. An example is illustrated here:

Taking the delivery robot in the hospital as an example, this robot works in a workspace as shown in figure 2. It has two tasks, one is to deliver food to each ward during mealtime, and the other is to deliver medicine to these four wards according to different needs. When the robot performs its task, the attacker can send false messages to the robot, such as controlling the speed of the robot or moving it to a dangerous place. Based on security considerations, we abstract the security behavior of the robot into specifications and automatically synthesize the controller. In addition to writing security specifications for the robot, the task specifications should be written according to the tasks performed by the robot. The task specification of the robot is shown in figure 3,

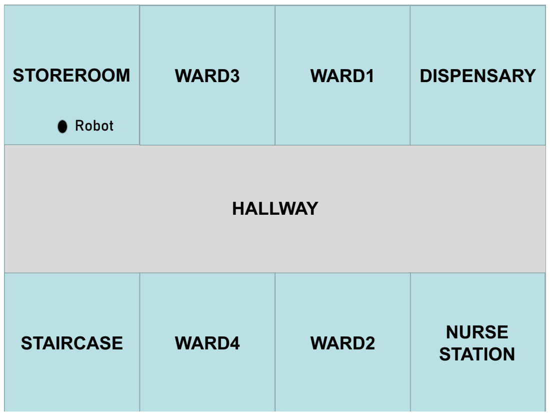


Fig. 2: Workspace of delivery robot

and the security specification of the robot is shown in figure 4.

The security specification means that when the robot is empty, it can move quickly ($0.8m/s - 1.5m/s$), but when the robot is delivering food or medicine, its maximum speed should not exceed $0.5m/s$. In any case, it must stop when it senses someone within one meter. Attackers may send false coordinates information to the robot and move the robot to a dangerous area. Therefore in this example, if it is detected that the robot enters the staircase, the security controller should immediately control the robot to leave the staircase and return to the storeroom. Because the robot is in danger of falling down the stairs in the staircase.

```
main R1{
  env {
    notification;
    medicine_1;
    medicine_2;
    medicine_3;
    medicine_4;}
  sys {
    stop;
    low_speed;
    fast_speed;
    go_to_nurses_station;
    carry_item;
    receive_notification;}
  asm {
    ! notification;
    ! medicine_1;
    ! medicine_2;
    ! medicine_3;
    ! medicine_4;}
  gar {
    G (notification -> receive_notification);
    GF (carry_item ->s.r1);
    GF (carry_item ->s.r2);
    GF (carry_item ->s.r3);
    GF (carry_item ->s.r4);
    GF (medicine_1 -> s.r1);
    GF (medicine_2 -> s.r2);
    GF (medicine_3 -> s.r3);
    GF (medicine_4 -> s.r4);
    G (notification -> next(go_to_nurses_station));
    G ((s.nurses_station & receive_notification) ->
         next(carry_item );}}
```

Fig. 3: Task specification of delivery robot

```
main R1{
  env {
    person;
    low_battery;}
  sys {
    stop;
    low_speed;
    fast_spee;
    carry_item;
    go_to_nurses_station;}
  asm {
    !person;
    !low_battery;
    GF !person;}
  gar {
    !carry_item;&
    fast_speed;
    !low_speed;
    !stop;
    !s.staircase;
    G (s.staircase -> next(s.storeroom));
    G (low_battery -> next(s.storeroom));
    G (person -> stop);
    G (carry_item -> low_speed);
    G (! carry_item -> fast_speed;}}
```

Fig. 4: Security specification of delivery robot

In our previous work, we generally wrote the robot's security specification and task specification together and automatically synthesized the controller. Therefore, it takes a long time to synthesize a controller with many propositions. As we introduced in section III, we divide the specifications of the robot into two parts and synthesize the corresponding controller respectively. TABLE I is a comparison of the synthesis time of different controllers. It can be seen that the second method has higher efficiency.

TABLE I: Time of controller synthsis

Controllers	**Time**
Global controller	50.594 seconds
Task controller	2.844 seconds
Security controller	0.033 seconds

B. Denial of Service Attack

Denial of service (DOS) [10] attack can trigger crash in a computer or network by flooding them with traffic, this attack is to use other hijacked computers on the network to launch an intensive "denial of service" attack on a specific target computer. This will consume the network and system resources of the target computer, making it impossible to provide services for normal users. All nodes in the network can publish data to topics subscribed by a certain target node in ROS, and these nodes can publish arbitrary content, so attackers can conduct DOS attack against this target node. Because there are many types of DOS attacks, to determine whether the robot is under DOS attack, we monitor the following indicators.

- CPU usage: The CPU usage of the robot.
- CPU temperature: The CPU temperature of the robot.
- RAM: Memory usage of the robot.

- Number of messages: Number of messages received by each node per period.
- Battery consumption: Power consumption per period.

We normally record data for these indicators and set thresholds for each indicator. When the robot performs a task, if it detects an anomaly in the above indicator, it can be determined that the robot is under DOS attack. Then we generate security controllers for robots based on the specific tasks they perform.

Most researchers focus on how to protect robots from cyber attacks. But in the worst case, when the robot is attacked and loses function, to complete the task first, there should be other robots as standby robots. Therefore, we introduce a task-first method, which is suitable for robots that perform critical tasks. When the robot performs a task, it records its state changes and sends the log file to the control center in real-time. When other robots or control centers find that a robot is attacked and loses its function, for example, it does not reach the designated location or make appropriate actions as planned. Due to the high priority of the task, the standby robot should be dispatched immediately to reach the designated state. Then the standby robot copies the controller strategy of the attacked robot and cooperates with other robots to complete the task.

For further explanation, we introduced an example of an MRS that performs security tasks at a private airport. As shown in figure 5, There are four robots performing tasks here. Robot 1 is mainly responsible for the patrol of the area. Robot 2 is the maintenance robot for the airport runway, robot 3 is the fire extinguishing robot, and robot 4 is the guard robot in general, but when an emergency occurs, It can be used as a standby robot to succeed other robots to complete tasks.

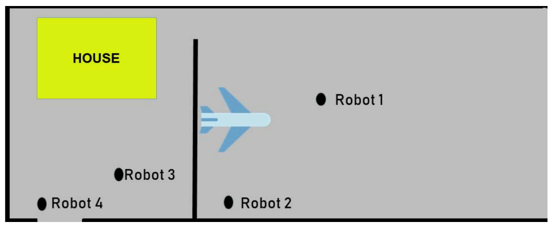


Fig. 5: Multi-robot workspace

During the patrol process of robot 1, when it detects abnormalities in the airport runway, such as cracks or holes on the ground, which may endanger the take-off and landing of the aircraft, it will set up danger signs on the spot and send the corresponding coordinates to robot 2. When the robot 2 receives the message, it will proceed to maintenance. When the robot 1 detects a fire, it will immediately notify the robot 3, which will open the fire hydrant and start extinguishing the fire. When it is detected that robot 1 is under DOS attacks, to prevent it from making dangerous actions, it should immediately turn off the power and stop at the same place for checking by the maintenance personnel. At the same time, to ensure the completion of the task, the standby robot 4 should be enabled to complete the task instead. The task specification and security specification of this MRS are written in figure 6.

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.

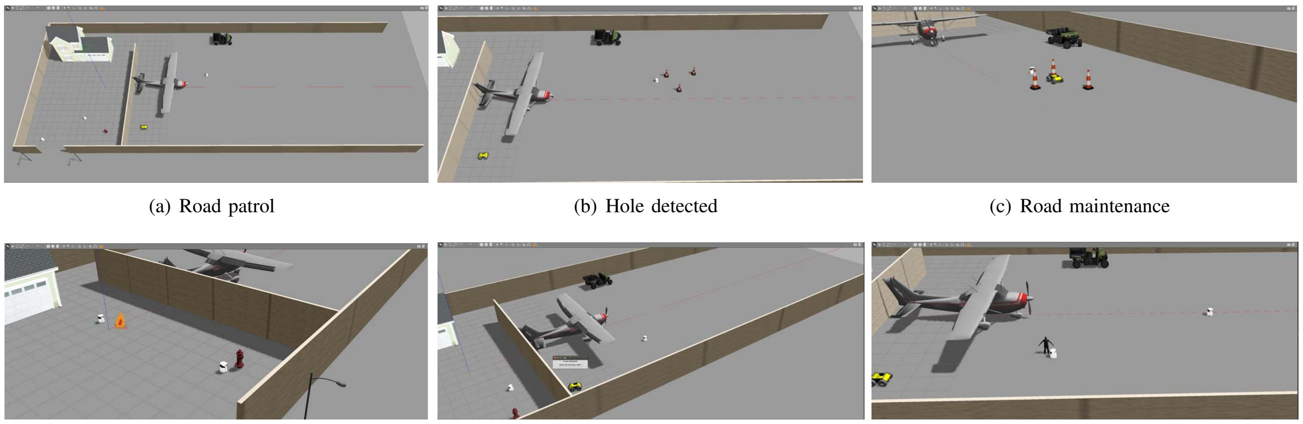
```
main R1{
  env {
    hole;
    fire;
    msg_number;
    RAM;
    CPU_temperature;
    battery_consumption;
    CPU_usage;}
  sys {
    inform_r2;
    inform_r3;
    inform_r4;
    DOS_attack;
    shut_down;}
  asm {
    !hole;
    !fire;
    !msg_number;
    !CPU_usage;
    !CPU_temperature;
    !RAM;
    !battery_consumption;}
  gar {
    ! inform_r2;
    ! inform_r3;
    ! inform_r4;
    ! DOS_attack;
    ! shut_down;
    G(hole -> inform_r2);
    G(fire -> inform_r3);
    G((msg_number & CPU_usage & CPU_temperature & RAM
    & battery_consumption) -> DOS_attack);
    G(DOS_attack -> inform_r4);
    G(inform_r4 -> X (shut_down);}}

main R2{
  env {
    receive_r1;}
  sys {
    maintenance;}
  asm {
    !receive_r1;}
  gar {
    !maintenance;
    GF(receive_r1 -> maintenance;}}

main R3{
  env {
    receive_r1;}
  sys {
    extinguishment;}
  asm {
    !receive_r1;}
  gar {
    !extinguishment;
    GF(receive_r1 -> extinguishment;}}

main R4{
  env {
    receive_r1;}
  sys {
    replace_r1;}
  asm {
    !receive_r1;}
  gar {
    !replace_r1 &
    GF(receive_r1 -> replace_r1;}}
```

Fig. 6: Specification for airport patrol robots



(a) Road patrol (b) Hole detected (c) Road maintenance

(d) Fire detected (e) DOS attack detected (f) Check and replace

Fig. 7: Experimental run of the airport patrol robots scenario

In this example, we use ROS as the system to control the movement of the robot and simulate the entire task process in gazebo. The simulation environment should be configured first, according to the design diagram in Figure 5, we build the world model in gazebo. As the name indicates, the world model is to simulate a real physical world. For the tasks to be completed by the robots, the user could build a corresponding world model using Gazebo's building editor. 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. Then the robots could perform navigation and localization according to the known map. The simulation process is shown in figure 7.

V. CONCLUSION AND FUTURE WORK

In this paper, to ensure the safety of the robot when performing tasks, we automatically synthesize a controller for the robot to constrain the robot's behavior. A highly efficient method is proposed to reduce the controller synthesis time. Also, we summarized some of the security problems in ROS and put forward protective measures against false information induction and DOS attacks. In particular, we used Gazebo as a simulation environment to conduct experiments to protect robots against DOS attacks.

In future work, we plan to pay more attention to improve the security of ROS, classify the possible attacks on ROS, and propose corresponding protective measures. At the same time, we will use real robots to conduct experiments and test the practicability of the method we presented.

REFERENCES

[1] A. Bhardwaj, V. Avasthi, and S. Goundar, "Cyber security attacks on robotic platforms," *Netw. Secur.*, vol. 2019, no. 10, pp. 13–19, 2019. [Online]. Available: https://doi.org/10.1016/S1353-4858(19)30122-9

[2] M. Husák, J. Komárková, E. Bou-Harb, and P. Celeda, "Survey of attack projection, prediction, and forecasting in cyber security," *IEEE Commun. Surv. Tutorials*, vol. 21, no. 1, pp. 640–660, 2019. [Online]. Available: https://doi.org/10.1109/COMST.2018.2871866

[3] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Temporal-logic-based reactive mission and motion planning," *IEEE Trans. Robotics*, vol. 25, no. 6, pp. 1370–1381, 2009. [Online]. Available: https://doi.org/10.1109/TRO.2009.2030225

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] E. Basan, M. Medvedev, and S. Teterevyatnikov, "Analysis of the impact of denial of service attacks on the group of robots," in *International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, CyberC 2018, Zhengzhou, China, October 18-20, 2018*, 2018.