ISBN 978-981-11-3671-9
Proceedings of 2017 the 7th International Workshop on Computer Science and Engineering (WCSE 2017)
Beijing, *25-27 June*, *2017*, *pp. 1197-1207*
doi: 10.18178/wcse.2017.06.209

Towards a Robust Software Architecture for Autonomous Robot Software

Shuo Yang[+], Xinjun Mao, Sen Yang, Zhe Liu, Guochun Chen, Shuo Wang, Jiangtao Xue, Zixi Xu

College of Computer,
National University of Defense Technology

Abstract. The decision and execution of autonomous robot behaviors highly depend on constant perceiving of the situated environment, in order to respond to various changes. Existing control architectures for robot software generally follow a sequential sense-model-plan-act (SMPA) architecture, in which the sensing activity only interacts with the modelling activity that triggers the planning and acting activity. However, in dynamic world, the constant changes of environment not only influence robot's behavior decisions but also affects their plan execution. The robot behaviors to fulfill tasks are often required to be accompanied with a series of sensing activities. To deal with this issue, this paper presents a robust software architecture for autonomous robot software, in which sensing activity interacts with the modeling, planning and acting activities. Such approach enables autonomous robot software to constantly sense environment and obtain expected percepts in different phase of control loop. Therefor it enriches autonomous robot capability to take behaviors based on the sensing and improves the robustness of autonomous robot software. This paper details the model of architecture, designs behaviors schedule algorithm and presents a mutual data store mechanism to support the interactions between the activities in architecture. We also conduct a comparative experiment on humanoid robot NAO, and the result validates the enhanced robustness of our proposed architecture over traditional structure.

Keywords: autonomous robot software, robot software architecture, robustness

1. Introduction

Autonomous robots are increasingly applied in nonindustrial domains (e.g. hospital, family, military, space exploration), and are expected to gain success in several emerging application domains which are characterized by open environment, task-oriented adaptive behavior, and human-robot interaction [1]. In essence, they are actually a kind of complex cyber-physical and social ecosystem that performs both computational and physical behaviors in an autonomous way, according to the assigned tasks by interacting with open and dynamic environment.

To achieve full capabilities of autonomous robot, the software plays a critical role in aspects of managing physical devices, processing sensory information from sensors, scheduling robot's tasks and planning robot's behaviors. Moreover, the bi-directional interactions between the robot and situated environments pose new challenges for autonomous robot software. Generally, the situations where autonomous robots operate are highly dynamic and full of uncertainties. For example, the tour guide robot usually serves in large department stores where the movement of guests are actually unexpected and changing dynamically. On the one hand, the robot may cause state changes in situated environment by effects of its physical operations. On the other hand, the evolvement of environment has great impacts on the actual effects of robot plan executions. For example, the robot may collide with a chair, which suddenly changes its location, while executing the predefined navigation plan. Considering the dynamics and uncertainties of real-world environments, the robust plan execution requires that an autonomous robot must be able to continuously

[+] Corresponding author. Tel.: +8618674823466.
E-mail address: 1024809808@qq.com.

sense the environment and adapt its behaviors while executing plans, in response to unpredictable changes and exceptional situations that may suddenly emerge [2]. Therefore, to improve robustness of robot plan execution, it becomes increasingly significant for robot control software to keep continuous sensing along with the process of plan execution for run-time contingencies awareness [3].

However, the mostly commonly applied paradigm of sense-model-plan-act (SMPA) generally decomposes the robot software into a series of functional activities that carry out sequentially [4]–[6], which was first used in Shakey [7]. The main architectural characteristics of this paradigm are that sensing flowed into a world model, which was then used by the planner, and that plan was executed without directly using the sensors that created the model. However, this model proves inadequate in strictly real-time applications especially when robot operating in dynamic and unpredictable environments, as established world model and plans which are based on precedent sensed information, are usually out-of-date and become inapplicable when the robot begins to execute the plans. This is mostly caused by the amount of time and memory required to maintain and develop plans, that the state of the environment changes before plans can be carried out [7].

Essentially, for autonomous robots that operate in open and dynamic environments, it is urgently significant for the capability of continuously aware of run-time contingencies and unpredictable state changes. As a result, this paper proposes a robust software architecture that can effectively support parallel operations and continuous interactions between sensing and acting activities, instead of merely interacting between modelling activity in the SMPA architecture. Such approach can enrich the robustness of autonomous robot in terms of response to rapid environment changes in acting activity. Moreover, this paper implements the software architecture on the existing agent platform BDI4Jade [10], and presents key algorithms to support the parallel operation of sensing and acting activities.

The rest of paper is organized as follows: Section 2 discusses several related works about robot software architectures and their weakness for developing autonomous robot software. Section 3 describes a typical motivating scenario behind our work and discusses some requirements on robot software architecture. Section 4 presents an overview of the proposed architecture and its formal model. Section 5 presents the key algorithms and mechanisms behind the architecture implementation. In section 6, we present a comparative experiment on a real robot to validate the effectiveness and performance of the proposed architecture. Section 7 concludes this paper and discusses some directions for future research.

2. Related Works

In the field of robotics, “software architectures” are methodologies that supply structure and impose constraints on the way that robots are controlled [11]. Generally, several trends in the development of robot software architectures can be observed: reactive control, deliberative control, hybrid control, and behavior-based control [6], [13], [14].

At the early stage of robotic applications, reactive systems emerge as effective architectures for simple robot tasks, which are best characterized by a direct connection between sensors and actuators. The control is not mediated by a model but rather occurs as a low level pairing between stimulus and response [15]. Circuit architecture [16] groups reactive behaviors and logical formalisms into arbitrated collections. Action-Selection [9] uses activation levels as a dynamic mechanism of cooperative behavior selection. Individual behaviors are grouped as competence modules that respond when predefined conditions are detected.

As robot autonomy plus environmental diversity entail the need for deliberation [17], the research efforts focused on coherent integration of sensing, deliberating and acting in a dynamic world, in an attempt to develop completely rational mobile robots. [18] proposes a general purpose AI-based control architecture that deliberates about future states, plans for actions, and executes generated activities while monitoring plans for anomalous conditions.

To adapt quickly to rapid changes and deliberate over complex tasks, hybrid architectures arises as a combination of both previous paradigms, by blending deliberative planning and reactive mechanisms in order to overcome their disadvantages [6]. In [19], Stoytchev presents a hybrid robot architecture that combines three components: deliberative planning, reactive control and motivational drives. The reference

model architecture proposed by [20] is one of the most prevalent representations of this paradigm [21]. The architecture proposed in [19] combines three components: deliberative planning, reactive control and motivational drives. AuRA [22] is first to integrate hierarchical and reactive planning mechanisms.

The behavior-based architectures take the first steps in the model definition, with the representative work of R. Brooks on formalization of behavior [23]. Such architectures emphasis on a tight coupling between sensing and action, and decomposition into behavioral units [15], [22]. [24], [25] proposes a behavior-based SSVEP hierarchical architecture, consisting of an SSVEP model layer, a behavior mapping layer, and a humanoid controller layer. A bio-inspired, incremental, behavior-based architecture is proposed by Fabian [8], which is composed of two layers by using instinctive or deliberative reactions. [26] implements a control system on the basis of active object computing model which consists of event driven paradigm, event driven architecture, and UML state chart formulation.

As pointed in [5], [6], SMPA paradigm (**Fig.1**) is a nutshell in any control system discussed above. The extent to which these phases are instantiated in particular systems varies greatly. Specifically, the amount of deliberation (modelling and planning) can be strongly emphasized or totally neglected.

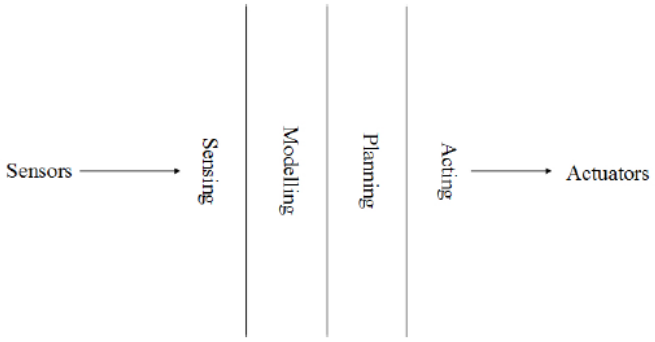


Fig. 1: The S-SMPA paradigm in robot software architectures.

3. A Motivating Scenario

Let us consider a motivating scenario of autonomous robots serving in the home. The domestic environment can be rather dynamic and open, as some moveable objects like chairs can be frequently moved from one position to another by human activities, or human walking around randomly inside the house, both of which may pose threats the navigating process of robot.

Fig.2 describes a typical scenario in domestic service. When the robot is assigned a task-goal "Fetch Tea", its on-board planner is called to generate an ordered set of task plans (such as "GetIn Kitchen" plan) that commit to achieve the goal, based on precedent world models. Each of plans consists of a finite ordered sequence of actuator-based actions which directly control the robot physical effectors to accomplish specific tasks. The acting process of executing an actuator-based action may meet with various run-time contingencies in situated environments. Take "go to door" action for example, while the robot is walking towards the door that connects with kitchen, moveable entities such as furniture or humans may suddenly appear in the way of robot, which may cause the robot fail to reach destination as a result of out-of-date beliefs about external world. Therefore, sensing activities of sensor-based actions are urgently required to keep constantly perceiving environmental changes for robust plan execution. The above mentioned features of a motivating scenario are thus specified as:

- *Situated environment is highly dynamic*: in open and dynamic environment, state changes occur unexpectedly and unanticipated. There are possible run-time contingencies such as moveable entities may suddenly appears in the way of robot, which may potentially cause an unexpected failure to robot plan execution.
- *World models need constantly updating*: in real world, when robot carry out the predefined behavior decisions that based on precedent world models, current world conditions may be inapplicable for plan execution, due to rapidly changing environment. Therefore, world models in robot need to be updated continuously.
- *Acting activities closely depend on sensing activities*: While robot is acting to accomplish a task, relevant sensing activities are supposed to keep continuously monitoring environmental changes and communicating

with acting activities. For example, sensors such as sonar and bumper sensors are required to run in parallel with process of moving ahead, so that the robot is constantly aware of external state changes.

—

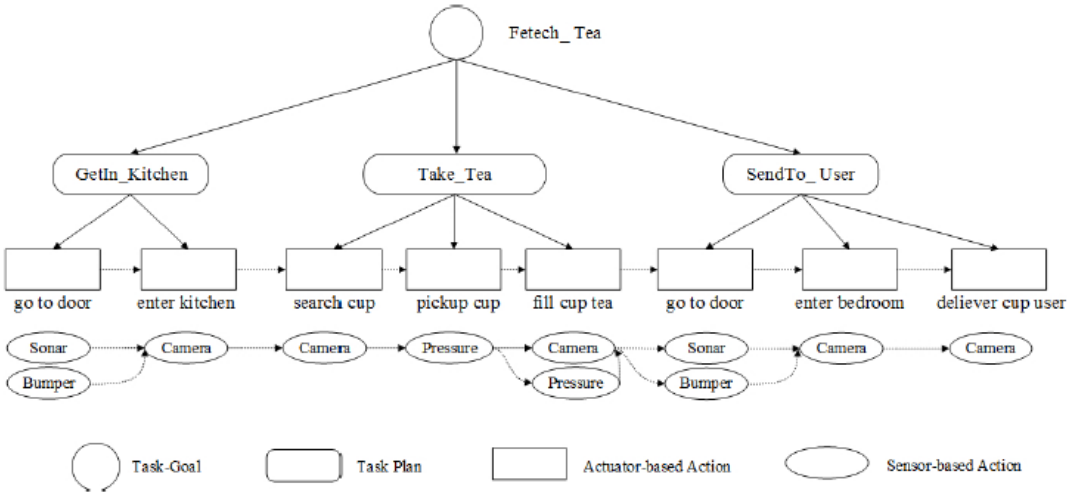


Fig. 2: A motivating scenario in robot domestic service.

4. The Robot Software Architecture

4.1. Overview

The structure of our proposed architecture is illustrated in Fig.3, which is based on two hierarchically layered controllers: deliberative controller and sensory controller. The deliberative controller represents the task-oriented deliberation process, which tightly integrates the activities of modelling, planning and acting. The sensory controller, on the other hand, represents a parallel process of sensing activities. Features of sensing activity are specified as follows:

- *Multi-source*: sensing activities are accomplished by multiple heterogeneous sensors that sense different aspects of world states, such as vision, audio and pressure.
- *Multi-mode*: distinct sensors typically run in multi-mode due to its inherent physical property. For instance, to detect an obstacle, sonar runs cyclically for continuous distance value input, while a camera takes a one-shot image.
- *Least commitment principle: for sake of energy conservation, the execution of a task-oriented plan usually requires limited aspects of sensory information that satisfy a least commitment principle. For example, in a navigation activity, only camera, sonar and bumper are required to work for obstacle detection, while other sensors such as microphone are unnecessary.*

The basis in simultaneous communications among the parallel layers lies in the mechanism of mutual data store, which sets a mutual memory space for both sensing and acting activities to manage and access run-time information.

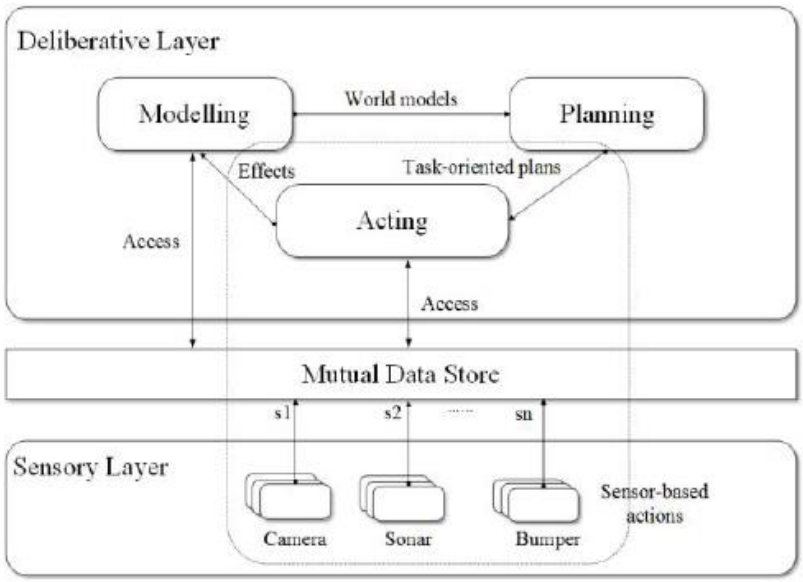


Fig. 3: The structure of proposed architecture

4.2. Formal Model

In this section, we establish a formal model for the architecture to well define the architecture components and describe relationships among them. The model of parallel control architecture is a tuple $< S, M, P, A, D >$:

- A (finite) set of sensor-based actions $S = \{s_0, ..., s_n\}, n \in N$; a primitive sensor-based action $s_i \in S$ can be defined as $s_i =< pre(s_i), per(s_i) >$ where $pre(s_i)$ and $per(s_i)$ represent precondition and perception obtained from the situated environment, respectively.
- A (finite) set of actuator-based actions $\{a_0, ..., a_m\}, m \in N$; a primitive actuator-based action $a_i \in A$ can be formalized as a triple $a_i =< pre(a_i), add(a_i), del(a_i) >$ where $pre(a_i)$, $add(a_i)$ and $del(a_i)$ represent the preconditions, addition effects and deletion effects on world state as a result of executing this action, respectively.
- A (finite) set of ordered set of task-oriented plans $P = \{p_0, ..., p_l\}, l \in N$; an ordered sequence of actuator-based actions $< a_0, ..., a_n >$ is said to be a plan p. A task-oriented plan $p_i \in P$ is a tuple $p_i =< t(p_i), c(p_i), bd(p_i) >$ where $t(p_i)$ is the invocation, indicating the event that causes the plan to be considered for adoption; $c(p_i)$, the context condition, encoding the situations under which this plan is applicable; and $bd(p_i)$ is the plan body consisting of some ordered sequence of steps $< a_0, ..., a_n >$, where each a_i is a primitive actuator-based action.
- The world model M is a tuple $< \omega, \varphi, \Pi >$ where ω represents the sensory input about situated environments, φ comprises knowledge after analysis and Π represents a task for an autonomous mobile robot. A task is represented by a triple $\Pi =< \Delta, \tau_O, \tau_G >$, where Δ is a planning domain, τ_O is the initial state and τ_G is the goal state. A task-oriented plan p_i can be selected by the planner at a state τ_O only if $c(p_i) \subseteq \tau_i$. While a sensor-based action s_j is instantiated whenever a running action $a_k \in P$ requires $per(s_j)$ for deliberation, namely $per(s_j) \subseteq pre(a_k)$. By applying p_i at state τ_i, a new state $\tau_{i+1} = add(a_n) \cup (\tau_i \setminus del(a_n))$ is observed.
- A library of data store $D = \{d_0, ..., d_n\}$ where each $d_i(k, v)$ contains a set of keys and corresponding values.

The control loop within our proposed architecture is illustrated in Algorithm 1. The first step is for establishing world models in robot beliefs at initial stage, consisting of gathering essential sensory information from sensing activities and data stores. In the loop cycle, task-oriented plans are generated by embedded planners based on precedent world models. Then the robot begins to execute plans in its acting activities together with accompanying sensor-based actions, via mutual data store for simultaneous communications between parallel processes. Meanwhile, the world model is constantly updated by up-to-date sensory input and actual effects of robot physical actions.

Algorithm 1 Control Loop of P-SMPA Architecture
1. **Procedure** ControlLoop(S, M, P, A, D)
2. establishInitialModel: $S \xrightarrow{D} M$
3. loop
4. generatePlans: $M \rightarrow P$
5. executePlans: $S \times P \xrightarrow{D} A$
6. updateModel: $A \times S \rightarrow M$
7. end loop
8. end Procedure

5. Implementation

As this paper mainly focuses on a parallel process of both acting and sensing activities, run-time supports for parallel execution are the key to implementation of our architecture. To implement the architecture, the control issue of parallel execution and interaction issue are to be solved.

- *Control issue*: In our architecture, a separation of concern has been achieved with task-oriented plans and sensor-based actions, which represent robot acting and sensing behaviors respectively. However, the control issue of coordinating execution orders for both parallel activities of sensing and acting needs to be addressed, to enable flexible transitions and cooperation between parallel processes.
- *Interaction issue*: As a robot is allowed to execute both acting and sensing behaviors in parallel, simultaneous communications between parallel behaviors are required to support run-time interactions. The issue of how to fast and efficiently exchange essential information remains to be handled.

5.1. Parallel Scheduling Algorithm

In our implementation, an actuator-based action a or sensor-base action s is regarded as a primitive behavior of a robot, while a task-oriented plan p resembles a FSM-based (Finite State Machine) composite behavior that consists of multiple states (sub-behaviors) and rules for state transitions. This algorithm extends the FSM-based agent behavior scheduling strategy [27] by incorporating additional parallel sensor behaviors and modifying behavior transition rules.

Algorithm 2 describes the scheduling algorithm for parallel execution of task-oriented plans and sensor-based actions. Firstly, a task-oriented plan p is transformed into a finite state machine $f(\Lambda,T)$ that consists of an ordered set of primitive behavior states Λ, and a set of transition rules T that defines the execution order of behavior states. Secondly, for a sensor-based action s_k that is required to run in parallel with an actuator-based action $a_i(per(s_k) \subseteq pre(a_i))$, f generates a parallel behavior state $\delta_i^{'}$ that add both states δ_i and s_k as sub-behaviors, and set condition e_δ for parallel behavior termination. The termination condition indicates that as sensor behaviors keep along the lifetime of corresponding actuator-based action, this parallel behavior $\delta_i^{'}$ will terminate as soon as δ_i stops running. In the end, the newly generated parallel behavior state $\delta_i^{'}$ will replace original state δ_i.

Algorithm 2 The parallel behavior scheduling algorithm **Input**: a set of task-oriented plans P and sensor-based actions S. **Output**: an ordered set of finite state machines F.
1. **for** a task-oriented plan $p \in P$ where $(bd(p) =< a_0,...,a_n >)$ **do**
2. $f(\Lambda,T) \leftarrow fsmGenerator(p)$
3. **for** each $a_i \in bd(p)$ **do**
4. $\delta_i \leftarrow a_i$
5. $\Lambda \leftarrow addBehaviorState(\Lambda,\delta_i)$
6. **end for**
7. **for** any $a_i \prec a_j$ or $a_i \succ a_j$ that $i, j \in [0,...,n]$ **do**
8. $r_{i,j} \leftarrow setTransitionRule(\delta_i,\delta_j,\prec or \succ)$
9. $T \leftarrow addTransitionRule(T,r_{i,j})$
10. **end for**
11. **for** a sensor-based action $s_k \in S$ that $per(s_k) \subseteq pre(a_i)$ **do**
12. $e_\delta \leftarrow setStopCondition(s_k,\delta_i)$
13. $\delta_i^{'} \leftarrow addParallelBehavior(s_k,\delta_i,e_\delta)$
14. $r_{i,j} \leftarrow setTransitionRule(\delta_i^{'},\delta_j,\prec or \succ)$
15. **end for**
16. end for

5.2. Mutual Data Store Mechanism

In the run-time of robot acting to achieve tasks, simultaneous communications between sensing and acting behaviors are essential for robot to be continuously aware of actual acting effects and

environmental state. To meet with strict time requirements on simultaneous communications, this paper proposes a mutual data store mechanism (see in Algorithm 3) for flexible and efficient communications among parallel behaviors. In our architecture, each of robot behaviors maintains an individual data store, which is an extension of a typical hash map structure. The individual data store helps a behavior to store and manage run-time data flexibly and efficiently, as the time costs for I/O operations are reduced largely. For a parallel pair of sensor-based action s_j and actuator-based action a_i, to enable a_i simultaneously communicate with s_j for in-time sensory information, a mutual data store helps establish a quick connection between two behaviors, thus an actuator-based action can flexibly access required sensory information from its parallel sensing activity.

Algorithm 3 Mutual Data Store Mechanism	
1.	**for** each $p(bd(p) =< a_0,...,a_n >) \in P$ **do**
2.	$d_{a_i}(k,v) \leftarrow a_i \bullet getDataStore()$
3.	**end for**
4.	**if** sensor-based action $s_j \in S$ that $per(s_j) \subseteq pre(a_i)$ **then**
5.	$d_{s_j}(k,v) \leftarrow s_j \bullet getDataStore()$
6.	$d_{s_j}(k+k',v+v') \leftarrow s_j \bullet put(k',v')$
7.	$d_{mutual(s_j,a_i)} \leftarrow setDataStore(d_{s_j}, d_{a_i})$
8.	$v' \leftarrow a_i \bullet get(d_{mutual(s_j,a_i)}, k')$
9.	**end if**

6. Experiments and Analysis

To validate effectiveness of the proposed architecture and evaluate plan execution robustness, this paper has conducted a comparative experiment with the original paradigm of SMPA architecture. In the experiment, the "GetIn Kitchen" plan in the motivating scenario was picked up for testing and evaluating the robustness of execution results. The experiment was tested on a humanoid mobile robot NAO in a lab environment. In the testing scenario (**Fig.4**), the robustness of plan execution could be measured at the sensitivity to unexpected external changes. In that case, we intentionally changed the world conditions for several times to interfere the normal process, and recorded how many times the robot had detected the changes. In our experiment, when "go to door" action executed, a robot might encounter unexpected obstacles in the way, which would possibly make the plan fail to reach the goal. To perform this action in our proposed architecture, sensors such as sonar and foot bumper kept running along with the moving process, while in the SMPA architecture, the sensing and moving processes were carried out alternately. To manually perform interferences, we picked up a tall box (0.8m) and a small ball (0.1m) to act as obstacles, which required various sensors to detect them, such as ultrasonic sonar and foot bumper. The obstacles were repeatedly placed in front of the robot for several times, and the recorded times of successfully avoiding obstacles were taken as a major measurement of plan execution robustness.

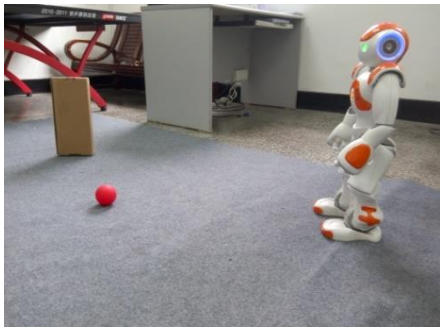


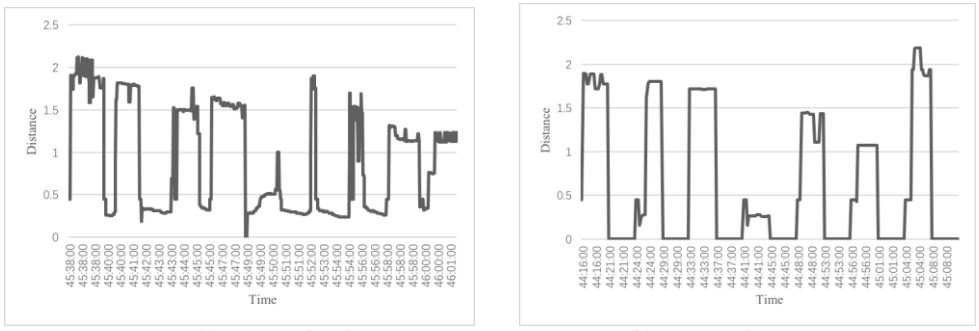
Fig. 4: A testing scenario in lab environment.

6.1. Experiment Result

For ultrasonic sonar, the detection range is 0.25~2.55m and the obstacle detection threshold is 0.5m. Detected obstacles are seen only if the distance is less than this threshold. When the sonar closes, it can detect nothing and the distance value remains 0.0m. As shown in Fig.5, we have interfered for 8 times during the moving process by repeatedly put a tall box in front of the robot. In Fig.5(a), there are 8 times when distance value decreases below the detection threshold, indicating the successful obstacle detection. However, as Fig.5(b) shows, there are 5 times when obstacles are detected in the working periods of sonar sensors.

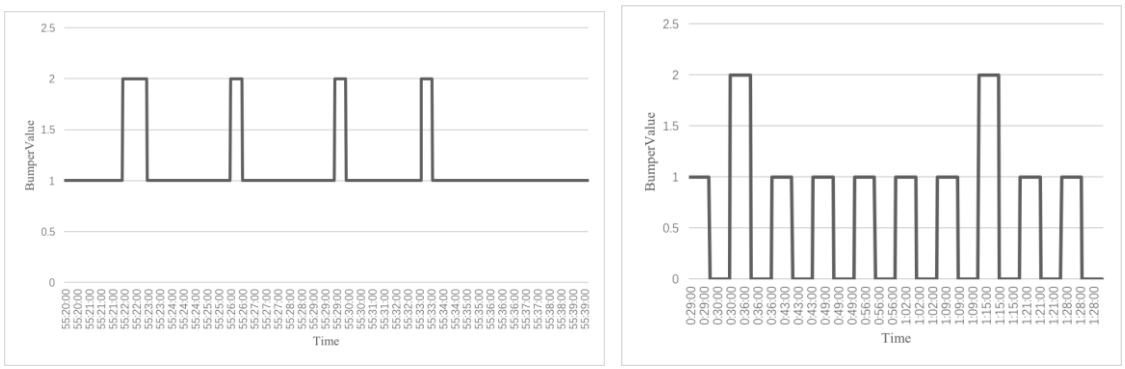
For the foot bumpers, there are two values "1" and "2" representing whether the bumper detects an obstacle or not. However, we denote the value of a closed bumper sensor as "0". In the whole process of going to the door, the small ball appeared 4 times in the way at certain time points. In Fig.6(a), there are 4 times when bumper value reaches "2", while in Fig.6(b), it is recorded 2 times when the value turns to "2" in the working periods.

In the experiment, we have repeatedly tested the robustness performances of both parallel and sequential approaches by adjusting the times of human interference. A detailed record of all the tests is summarized in Table I.



(a) Proposed architecture (b) SMPA architecture

Fig. 5: Sonar detection result.



(a) Proposed architecture (b) SMPA architecture

Fig. 6: Bumper detection result.

6.2. Summary and Analysis

As can be seen from Table.I, under the same world conditions and interferences, the proposed approach shows much better robustness performance, and usually spends less time than the original approach. In terms of the proposed architecture, both sonar and bumper have detected nearly all the obstacles in all the tests, reaching a much higher performance 66.7% ~ 100% compared with that of SMPA paradigm (33.3% ~ 66.7%). Moreover, to accomplish the task under similar conditions, the time cost for proposed architecture is generally much less than that of SMPA approach. For instance, when conducted 6 times of interferences for bumper test, the parallel approach spent 58 seconds while the sequential approach spent nearly more than twice time, reaching 117 seconds.

TABLE 1: PERFORMANCE EVALUATION OF THE TWO ARCHITECTURES

Architectures	Our proposed architecture	Original SMPA architecture

Interference Conditions		Obstacle(Tall Box) \ Obstacle(Small Ball)					
Sensor Types	Interference Times	Detected Times	Performance (%)	Time Cost(s)	Detected Times	Performance (%)	Time Cost(s)
Sonar	6	6	100	23	2	33.3	53
	7	7	100	42	4	57.1	87
	8	7	87.5	63	4	50	101
Bumper	4	4	100	20	2	50	59
	5	5	100	35	3	60	98
	6	4	66.7	58	2	66.7	117

7. Conclusion and Future Works

As traditional software architecture proves inadequate for supporting continuously sensing activities while robot acting to achieve tasks, a challenge that is faced by autonomous robot is how to improve robot sensitivity towards situated environment while executing plans, in order to enhance robustness of robot plan execution. Thereby, this paper presents robust software architecture and algorithms that support the parallel operation of sensing, modeling, planning and acting, enabling continuous sensing while executing plans. In particular, the proposed architecture proposes a new way of organizing the components of typical autonomous robot software into parallel structure, instead of traditional SMPA control model. Moreover, this paper proposes a series of algorithms for parallel behavior execution, and mutual data store mechanism for support of simultaneous communications between parallel processes of sensing and acting. A comparative experiment is conducted on a humanoid robot NAO in the motivating scenario. The results demonstrate improved robustness of plan execution via simultaneous execution of sensors and actuators. Due to strong sensitivity towards external changes, the proposed architecture is able to fast detect and react to run-time contingencies. Besides, our practical scenario shows that the average time cost for a plan in our architecture is much less than that of the tested traditional architecture. In future work, we will try to improve flexibility and autonomy in parallel operations of sensing and acting via distributed multi-agent methodology. First, we will further view an autonomous robot as a distributed multi-agent system, instead of a single rational agent. As stated in [28], since control of robots is by itself a large software project, the distribution of tasks to autonomous entities is a fruitful application of related techniques for coordination, communication and cooperation of agents. In the later work, a rational agent will play the role of deliberating to achieve specific tasks, including modeling, planning and acting, while a group of reactive agents will control the sensor devices to gather run-time information. In addition, the flexible coordination mechanism in multi-agent systems will be investigated to better support for the parallel processes of sensing and acting.

8. Acknowledgement

This work was supported in part by National Nature and Science Foundation of China under Granted Nos. 61379051, Program for New Century Excellent Talents in University under Granted No. NCET-10-0898.

9. References

[1] E. Karpas, S. J. Levine, P. Yu, B. C. Williams. Robust execution of plans for human-robot teams. 2015.

[2] A. Bouguerra, L. Karlsson, A. Saffiotti, Semantic knowledge based execution monitoring for mobile robots. IEEE International Conference on Robotics & Automation, 3693–3698. 2007.

[3] E. Shpieva, and I. Awaad. Integrating task planning, execution and monitoring for a domestic service robot. it – Information Technology 57(2), 1611–2776, 2015.

[4] N. C. Oza, A Survey of Robot Architectures. 1999.

[5] E. Spaho, K. Matsuo, L. Barolli, and F. Xhafa, Robot Control Architectures: A Survey. Lecture Notes in Electrical Engineering 253, 863–871, 2013.

[6] A. B. Ronov, and A. A. Migdisov, A survey of control architectures for autonomous mobile robots. *Journal of the Brazilian Computer Society* 4(3), 137–185, 1998.

[7] Nils J. Nilsson, Shakey the Robot. Sri International Menlo Park Ca 172(1991), 2, 1984.

[8] F. Rubilar, and Escobar, M. J. and Arredondo, T.: Bio-inspired architecture for a reactive-deliberative robot controller. International Joint Conference on Neural Networks, 2027–2035, 2014.

[9] Woolley, Brian G. and Peterson, Gilbert L.: Unified Behavior Framework for Reactive Robot Control. Journal of Intelligent & Robotic Systems 55(2-3), 155–176, 2009.

[10] Nunes, Ingrid and Lucena, Carlos J P De and Luck, Michael: BDI4JADE: a BDI layer on top of JADE. In: 9th International Workshop on Programming Multi-Agent Systems (ProMAS), 88–103 (2011)

[11] Szabolcs Michael de Gyurky and Mark A. Tarbell: The Architecture of the Autonomous System. John Wiley & Sons, Inc. (ProMAS), 1–21 (2013)

[12] Muoz, Manuel and Munera, Eduardo and Blanes, J. Francisco and Sim Jose E.: A Hierarchical Hybrid Architecture for Mission-Oriented Robot Control. Springer International Publishing, 363–380 (2014)

[13] Xu, H. and Brussel, H. Van: A behaviour-based architecture with attention control. Journal of Intelligent Manufacturing 9(2), 97–106 (1998)

[14] Das, Barnali and Ray, Dip Narayan and Majumder, Somajyoti: Experiments with a Behavior-Based Robot. Springer Berlin Heidelberg, 133–141 (2011)

[15] Tahboub, Karim A.: A Semi-Autonomous Reactive Control Architecture. Journal of Intelligent & Robotic Systems 32(32), 445–459 (2001)

[16] Kaelbling, Leslie Pack: An architecture for intelligent reactive systems. Reasoning About Actions & Plans, 395–410 (1987)

[17] Ingrand, Flix and Ghallab, Malik: Deliberation for autonomous robots: A survey. Artificial Intelligence, 1–40 (2014)

[18] Rajan, Kanna and Py, Frdric and Barreiro, Javier: Towards Deliberative Control in Marine Robotics. Springer New York, 91–175 (2013)

[19] A. Stoytchev and R. C. Arkin: Combining deliberation, reactivity, and motivation in the context of a behavior-based robot architecture. Proceedings IEEE International Symposium on Computational Intelligence in Robotics & Automation, 290–295 (2001)

[20] Albus, James S.: 4D/RCS: a reference model architecture for intelligent unmanned ground vehicles. Proceedings of SPIE - The International Society for Optical Engineering 4, 1–5 (2010)

[21] Yan, Yan and Tang, Zhenmin: Control Architecture for Autonomous Multi-Robot System: Survey and Analysis. Intelligent Computation Technology and Automation, 2009. ICICTA ’09. Second International Conference on, 376–379 (2009)

[22] Arkin, Ronald C. and Mackenzie, Douglas Christopher: Planning to Behave: A Hybrid Deliberative/Reactive Robot Control Architecture for Mobile Manipulation. International Symposium on Robotics & Manufacturing Maui Hi, 5–12 (1994)

[23] Brooks, R. A.: A Robust Layered Control System for a Mobile Robot. IEEE Journal on Robotics & Automation 2(1), 14–23 (1986)

[24] J. Zhao and W. Li and X. Mao and H. Hu and L. Niu and G. Chen: Behavior-based SSVEP Hierarchical Architecture for Telepresence Control of Humanoid Robot to Achieve Full Body Movement. IEEE Transactions on Cognitive and Developmental Systems PP(99), 1–1 (2016)

[25] Zhao, Jing and Meng, Qinghao and Li, Wei and Li, Mengfan: SSVEPbased hierarchical architecture for control of a humanoid robot with mind. Intelligent Control and Automation, 2401–2406 (2014)

[26] Multazam, A. W. and Mutijarsa, K. and Adiprawita, W.: Implementation of behavior-based control architecture robot for obstacle avoidance using active object computing model. IEEE International Conference on System Engineering and Technology 4, 1–5 (2014)

[27] Bellifemine, Fabio Luigi and Caire, Giovanni and Greenwood, Dominic: Developing Multi-Agent Systems with JADE. John Wiley & Sons, 317– 370 (2007)

[28] Burkhard, Hans Dieter: Agent Oriented Techniques for Programming Autonomous Robots. Fundamenta Informaticae 102(1), 49–62 (2010)