Development of Decentralized Based Reactive Control Strategy for Intelligent Multi-Agent Mobile Robotics System

Mohd Ridzuan Ahmad, Shamsudin H.M. Amin, Rosbi Mamat

Center for Artificial Intelligence and Robotics (CAIRO),
Faculty of Electrical Engineering,
Universiti Teknologi Malaysia,
8130 Johor Bahru, Malaysia.
Emails: ridzuan@adi.fke.utm.my, shams@suria.fke.utm.my, rosbi@suria.fke.utm.my

Abstract

Multi-agent system is one of the most popular research interests in robotics nowadays. How would collections of simple robots aid us in human endeavours? Let us think about an ant colony. Ants have little capability, but when many of them are working together, they can do incredible tasks. By examining the major control architectures available in robotics area, decentralized-based reactive control architecture gives the tools as required. Priority-based scheme is used in arbitration level to select which behaviour to run. It does not need to gather all information from various sensors to plan its action like deliberative control which is also known as sensors fusion that clearly required a lot of computational time. Besides that, the mapping that the deliberative control architecture produces might not be valid at the time when changes occur between the time it gathers the information and the time it plans its actions. Instead, intelligence will arise when these simple agents work together to perform some complex task cooperatively. Our multi-agent system consists of three mobile robots that are required to transfer a large object through a restricted area. We are using a leader - follower strategy to accomplish this task. Intelligence in multi-agent system in our context is that any agents have a possibility to be a leader or a follower depending on current situation of the environment. This means that all agents are homogeneously equal in hardware and software. By having this capability, it will ensure the burden of the programming task will be decreased because there will be only one program needed for the whole system. In this paper, decentralized reactive control architecture is used to develop an intelligent multi-agent cooperative strategy to perform complex task like carrying load and navigation.

Key Words: Multi-agent system, reactive control architecture, control algorithm.

1 Introduction

Multi-agent system is a nature of life. In human domain, this statement is always true [1, 2, 3]. How about in the artificial intelligence domain? In previous years, most of the researchers concentrated on a single agent. But, slowly the researchers realized the advantages of using multi robots in the achievement of complex task where a single agent cannot do it alone [4, 5, 6, 7]. In dealing with multi-agent system, two aspects must be identified first: type of task and control architecture. The types of multi-agent's tasks have been explored in three major classifications, which are known as merely coexisting: loosely coupled and tightly coupled [8]. In merely coexisting: multiple robots coexist in a shared environment, but do not even recognized each other (merely as obstacles). While in loosely coupled; multiple robots shared an environment and sense each other and may interact, but do not depend on one another (members of the group can be removed without significant effect). In our work, we are concentrating on the tightly coupled task where multiple robots cooperate on a specific task, usually by using communication, turn taking, and other means of tight coordination.

Control architecture in robotics area can be divided into two parts: single robot system and multi-robot system. For a single robot system, there are Deliberative, Hybrid, Reactive and Behavior-based control architectures [8]. These four control architectures are grouped together into two control approaches under multi-robot system, which are Centralised (Deliberative and Hybrid) and Decentralised (Reactive and Behavior-based) [8]. By examining these control architectures and aiming of simplicity in the multi-agent system, reactive control architecture gives the tool for developing the control algorithm for our multi-agent system. The philosophy of simplicity in multi-agent system is not saying we should build robot in simple worlds and then gradually increase the complexity of the worlds. Rather by
building simple agent in the most complex world we can imagine and gradually increasing the complexity of the robot [9]. It is interesting to define intelligence in terms of group behavior instead of individual behavior in a multi-robot domain. This means that each agent does not have to be too complex in terms of hardware and software. Instead, intelligent group behaviors will emerge when these simple agents are working together. This group behavior is also known as emergent behavior and can be observed when interaction between several robots occurred, but are hidden inside a single agent. In many cases, reactive and behaviors-based systems are designed to take advantage of such interactions [4]. They are designed to include emergent behaviors. This paper is organized as follows: section 2 discusses the strategy to accomplish the main tasks. Section 3 discusses the development of reactive control algorithm for multi-agent system. Section 4 concludes this paper.

2 Strategy for Main Task Achievement

2.1 Main Tasks for Multi-Agent System

Our multi-agent system consists of three mobile robots that are capable of searching and passing through an unknown passage in an indoor environment while holding one large object on top of them, as shown in Figure 1. The main idea is to use as simple agent (in terms of hardware and software) as possible, and with the help of inter-agent communication, they can do the complex task in a style. The main task of these three mobile robots is to transfer one large object through an unknown passage. Leader-followers strategy has been used where one mobile robot will be a leader while the rest will be followers.

Figure 1: Multi-agent system

From here, the main task can be divided into four sub-tasks for task accomplishment as stated below:

- Holding and carrying load safely
- Navigating in a team
- Searching for the passage
- Passing through the passage (orientation) and task confirmation.

2.2 Holding and Carrying a Load

In the absence of a manipulator, a supporting base is used to holding the load and provides some sort of small movement from the load. The idea is to avoid static placement of the load on top of the robot, which will cause the system to be push-and-pull situation between agents. The supporting base is illustrated in Figure 2. In Figure 3, the supporting base is attached on top of the mobile robot.

Figure 2: Supporting base

Figure 3: Supporting Base on top of mobile robot

Here, agents must make sure that the load is on top of them and that load is stable while they are navigating. For the load detection, one push button is used and located at point a. While for the load stabilization, the system is released to move with small movement freedom in translation (x and y-axes) and rotation (z-axis). This will avoid a push-pull situation between agents as shown in the Figure 4.
2.3 Group Navigation

Since the task of the multi-agent is under tightly-coupled, this means that all agents are physically related by means of the load through the horizontal distributions, making the group navigation behavior difficult to achieve. There are three issues in group navigation: distance maintenance, direction maintenance and obstacle avoidance. Most of the researchers used explicit inter-agent communication to achieve these behaviors according to [10, 11, 12, 13] although there are also some researchers who used explicit one which only relied on the sensor readings as indicated by [14, 15]. In our case, we are using the previous one. Even though, this will increase the complexity of both hardware and software of the multi-agent system, we found that its use cannot be avoided. For distance maintenance, four limit switches are attached at the point b, c, d and e as shown in Figure 2. These points indicate maximum and minimum limit in y and x axes respectively. For direction maintenance, the RF inter-agent communication is used to inform the changing of the direction of the robots. There are four commands of directions as shown in Figure 5.

2.4 Passage Searching

In this behavior, there are two infrared transmitters act as beacons placed at the passage to tell its location. The positions of the beacons are shown in Figure 6. The function of beacon label IR1 is to tell the agents that they are near to the passage, while beacon label IR2 informs the agents that they are passing through the passage.

By using this strategy, there are two possibilities that might occur. First, the agents will find the wall first instead of the passage. In this case, the agents will execute the behavior of wall following until they find the signal from the beacon IR1. The other possibility is the agents will straight away find the passage. In this case, the agent will receive the signal from the IR2 to tell them that they are passing through the passage.

2.5 Task Completion

When the passage has been detected based on the signal received by beacon IR2, the agents will move forward. This action will cause the rear agent (follower) to detect the wall as an obstacle (consequence from obstacle avoidance). This will execute the obstacle avoidance behavior until all agents have successfully passed through the passage. This is illustrated in Figure 7.
3 Multi-Agent Reactive Control Algorithm

3.1 Behaviors as Finite State Machines (FSM)

In developing of reactive control algorithm, it is wise to start it with the finite state machine. A finite-state machine (FSM) is an abstract computational element which is composed of a collection of states. Given a particular input, a finite state machine may change to a different state or stay in the same state. The specification of an FSM includes rules that determine the relationship between inputs and state changes [16]. This is shown in Figures 8-15.

Figure 8: Load Detection Behavior

In pseudo-code structure:

Outputs: (motor_command), (buzzer_command), (RF_command)

State-1: If button_load_detect = All

Else If button_load_detect = Local_Load

Switch to State-2

Else If button_load_detect = RF_Load

Switch to State-2

State-2: If time_in_this_state > timeout-1

Switch to State-3

Else RF_command = No_Load_Detect

Release

State-3: If time_in_this_state > timeout-2

Switch to State-4

Else motor_command = Stop

Release

State-4: If time_in_this_state > timeout-3

Switch to State-1

Else buzzer_command = On

Release

Figure 9: Obstacle Avoidance Behavior

Figure 10: Distance Maintenance Behavior

Outputs: (motor_command), (RF_command)

State-1: If IR_obstacle_detect = Nil

Release

Else If IR_obstacle_detect = IR_obstacle

Switch to State-2

Else If IR_obstacle_detect = RF_obstacle

Switch to State-2

State-2: If time_in_this_state > timeout-1

Switch to State-3

Else RF_command = Obstacle_Detect

Release

State-3: If time_in_this_state > timeout-2

Switch to State-4

If time_in_this_state > timeout-3

Switch to State-5

Else motor_command = Stop

Release

State-4: If time_in_this_state > timeout-4

Switch to State-1

Else motor_command = Backward

Release

State-5: If time_in_this_state > timeout-5

Switch to State-1

Else motor_command = Turn_Right

Release

Authorized licensed use limited to: IEEE Xplore. Downloaded on January 12, 2009 at 19:10 from IEEE Xplore. Restrictions apply.
Else If \( \text{time_in_this_state} > \text{timeout} - 1 \)
Switch to State-8
Else
motor_command = Forward
Release
State-7: If
\( \text{IR_wall_detect} = \text{IR_top} \)
Switch to State-4
Else If
\( \text{IR_wall_detect} = \text{IR_bottom} \)
Switch to State-5
Else If
\( \text{IR_wall_detect} = \text{RF_wall} \)
Switch to State-2
Else
motor_command = Wall_Detect
Release
State-8: If
\( \text{time_in_this_state} > \text{timeout} - 1 \)
Switch to State-1
Else
motor_command = Turn_Left
Release

Figure 11: Wall Detection Behavior

Output: \((\text{motor_command}), (\text{RF_command})\)
State-1: If
\( \text{IR_wall_detect} = \text{Nil} \)
Release
Else If
\( \text{IR_wall_detect} = \text{Local}\_\text{IR}\_\text{wall} \)
Switch to State-2
Else If
\( \text{IR_wall_detect} = \text{RF_wall} \)
Switch to State-3
Else
motor_command = Wall_Detect
Release
State-2: If
\( \text{time_in_this_state} > \text{timeout} - 1 \)
Switch to State-3
Else
motor_command = Stop
Release
State-3: If
\( \text{time_in_this_state} > \text{timeout} - 2 \)
Switch to State-4
Else
motor_command = Stop
Release
State-4: If
\( \text{time_in_this_state} > \text{timeout} - 4 \)
Switch to State-1
Else
motor_command = Turn_Right
Release

Figure 12: Wall Following Behavior

Output: \((\text{motor_command}), (\text{RF_command})\)
State-1: If
\( \text{IR_middle} = \text{Nil} \)
Release
Else If
\( \text{IR_middle} = \text{IR_top} \)
Switch to State-2
Else If
\( \text{IR_middle} = \text{IR_bottom} \)
Switch to State-5
Else If
\( \text{IR_middle} = \text{RF_contact} \)
Switch to State-3
Else
motor_command = Stop
Release
State-2: If
\( \text{time_in_this_state} > \text{timeout} - 1 \)
Switch to State-3
Else
motor_command = Passage.detect
Release
State-3: If
\( \text{time_in_this_state} > \text{timeout} - 2 \)
Switch to State-5
Else
motor_command = Stop
Release
State-4: If
\( \text{time_in_this_state} > \text{timeout} - 3 \)
Switch to State-6
Else
motor_command = Turn_Left
Release
State-5: If
\( \text{time_in_this_state} > \text{timeout} - 4 \)
Switch to State-6
Else
motor_command = Passage.detect
Release
State-6: If
\( \text{time_in_this_state} > \text{timeout} - 5 \)
Switch to State-1
Else
motor_command = Forward
Release

Figure 13: Passage Detection
3.2 Behavior’s Stimulus Response Diagram

We are using subsumption-based control architecture to implement our reactive control algorithm. This means that in the arbitration level, priority-based

scheme is used to select which behavior to run, instead of using vote scheme. The role of leader and follower of each agent can be swapped depending on which agent first found the passage. This is illustrated in Figure 16. In the control architectures, there are two aspects need to be clarified; Response encoding method and Coordination method. These two aspects are closely related and will affect the control architecture that going to be implemented. Response encoding is the way of mapping a range of stimulus with its associated behaviors. The stimulus is a signal produced from several types of sensors.

This signal then invokes the behaviour that maps with it. There are three types of mapping; Null, Discrete and Continuous. Null mapping will provide no motor response when its stimulus occurs. Discrete mapping will provide a response from an enumerated set of prescribed choices. Continuous mapping will produce a motor response that is continuous over stimulus range. In our control architecture, discrete mapping is used to produce motor response based on a particular stimulus signal. Discrete mapping is moderate
between the other two in term of motor response it will produce. Null will never produce motor response; while continuous will produce continuous response, which will burden the processor tasks. Again, this discrete mapping can be further grouped into two; Situated-action and Rule-based. In a situated-action, stimulus consists of a finite set of (situation, response) pairs. Sensing provides the index for finding the appropriate situation. This method required the programmer to identify all required situations needed. This also means that, it limits the environment to consist of only selected situations. This is difficult to implement into a real world, which consists of numerous unexpected situations. In our control architecture, rule-based method is used. In this method, stimulus is represented as a collection of "IF antecedent THEN consequent". The antecedent consists of a list of preconditions while the consequent is a motor response. There are no predetermined situations needs to be recognized. This will ensure the availability of the system to be executed in the real world.

Coordination method is a method for constructing a system consisting of multiple behaviors. In our case based on Figure 16, there are eight behaviors. There are two types of coordination methods available; Competitive and Cooperative. Normally, although it is not the rule, the discrete encoding is used with a competitive coordination method and vice versa. In competitive method, each behavior is chosen based on whether prioritisation or action selection. This means that the output is one from the listing behaviors. This is different with cooperative method where it blends the outputs of multiple behaviors in some way consistent with the agent's overall goals then produce new output to represent the overall behavior. In our system, competitive method is used.

Figure 16 also shows four emergent behaviors. These emergent behaviors are load detection, team navigation, wall searching and passage passing. These behaviors are not actually program priorly but can be shown when this system running in the real world. This emergent behavior is consequences from the interaction of individual behaviors from each agent. The agent's individual behaviors are load detection (note that this is not the same with the load detection emergent behavior), distance maintenance, obstacle avoidance, task completion, passage detection, wall detection, wall following and cruise.

4 Conclusion

The control architecture constraints the way an autonomous robot senses, reasons and acts, thus affecting its task performance. For single agent system, there are four famous types of control architectures available, known as deliberative, hybrid, reactive and based-based. These control architectures are also valid in multi-agent system but lies between two control approaches. These control approaches are known as centralized and decentralised. Deliberative and hybrid fall under centralised control approach, while reactive and based-based in opposite domain. In our case, the decentralised approach is more efficient to be used since it reduces the complexity of the agent in area of software and hardware. It is interesting to realise that intelligence will arise when these simple agents interact with each other. Most researchers classified this intelligence as the emergent characteristic in the decentralised control approach. Our main philosophy in multi-agent research is stated in below statement:

"Complex, fast, and intelligent multi-agent system comes from an interaction of simple agent in both hardware and software domains"

Acknowledgements

The research reported here was done at the UTM Robotics Laboratory. Support for this research was provided by Malaysian Ministry of Science, Technology and the Environment under the Intensification of Research in Priority Areas (IRPA) project number: 09-02-06-0084, and also the National Science Fellowship (NSF) Award for the first author.

References


