Paper Submission

12th ICCRTS
"Adapting C2 to the 21st Century"

Paper title: Evolving Control Logic Through Modeling and Simulation

Topic : Modeling and Simulation, C2 Technologies and System

Name of Authors: How Khee Yin, Victor Tay, DSTA, Singapore
Yeo Ye Chuan, Sui Qing, Cheng Chee Kong, DSO, Singapore

Point of Contact: Victor Tay

Organisation: Defence Science Technology Agency (DSTA)

Complete Address: Directorate of Research and Development
71 Science Park Drive, #02-05
Singapore 118523

Phone: 065-68795215

Email: tsuhan@dsta.gov.sg

## *Abstract*

One of the C2 paradigm shifts in the Information Age is Power to the Edge[1]. Unmanned vehicle operations are increasingly based on this new paradigm, i.e. from tele-operation to full autonomy, such that there is shared awareness, effective collaboration and actions synchronization.

Traditionally, the control logic for unmanned vehicle operations is handcrafted using Modelling and Simulation (M&S) tools. However, as the nature of operations becomes more complex and diversified this manual approach of logic programming becomes extremely difficult and tedious. To attain full autonomy, a more efficient method to programming the control logic is required.

In this paper, we describe how M&S can be leveraged to provide a learning environment to evolve control logic for an unmanned vehicle team. Our approach coupled a rule learning system based on Genetic Algorithm and Reinforcement Learning to an M&S system. We call this Simulation-based Rules Mining (SRM). Our idea is that control rules can be learned from an M&S environment (i.e. simulated robots and simulated environment), and then transferred onto real robots in a real environment. We successfully applied SRM to evolve the control logic for a robotic team to carry out search and destroy mission in an urban room environment.

[1]Alberts and Hayes, 2003, CCRP Publications

# 1. Introduction

As we move into the 21st century, the role that unmanned vehicles play in civilian and military operations is becoming more prominent. For example, in urban operations, unmanned vehicles can augment the sensing capability and firepower of the human soldier. At the same time, the C2 paradigm for unmanned vehicles is also shifting from tele-operation to full autonomy. To achieve full autonomy, we need to tackle the problem of programming the logic to control and coordinate the unmanned vehicle team to achieve the given task. Traditionally, the logic is hand crafted with the aid of M&S tools (e.g. see [1]). However, as the nature of the operations undertaken by unmanned vehicles becomes more complex and diverse, this approach of control logic programming becomes extremely difficult and tedious. Thus more efficient methods need to be found. In this paper, we describe how we can use M&S to provide a learning environment to evolve the control logic for the unmanned vehicle team. Our approach coupled a rule learning system based on Genetic Algorithm (GA) and Reinforcement Learning (RL) to a M&S system. We call this approach Simulation Based Rule Mining (SRM) (see Fig. 1). Our idea is that control rules learned from the M&S environment (simulated robots and simulated environment) can then be transferred onto real robots in real environment. Compared to learning straight on the physical robots, learning on a simulator has the advantages of ease of repeating trials and mitigates the risk of damage to hardware.

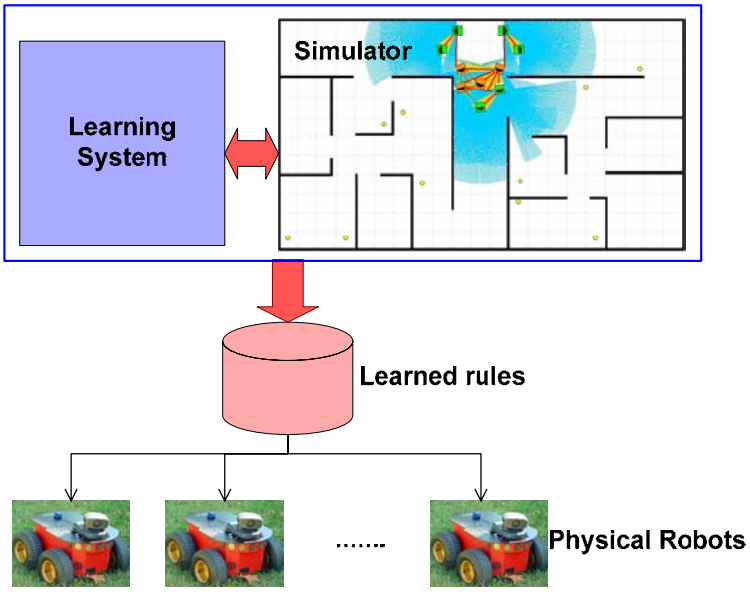


Fig. 1 Outline of SRM process.

In the SRM approach, the control logic for a robotic platform is encoded in the form of IF-THEN rules. Such rules are easy for humans to understand. In addition, using rules as the knowledge representation makes it easy for us to incorporate a priori knowledge into the system to bootstrap the learning process. A combined learning approach of RL and GA is proposed as the individual algorithms complement each other by tackling different aspects of the learning problem. GA can perform a distributed search through the rule space for sets of 'good' rules. However, the rulebases generated by GA usually contain non-mutually exclusive rules. In this situation, the rule priority is one metric to differentiate the 'good' rules from the 'bad' ones. RL can incrementally tune the rule priority of individual rules according to their

performance so that better-than-average rules have higher priorities than others in the long run.

We have tested the SRM approach to evolve the logic for controlling a robotic team to carry out search and destroy mission in an urban room environment. In this test, each robot needs only be equipped with sensors for navigation and target detection. There is no need for the robot to know its own position or to have the means to communicate with other robots. The learned control rules were shown to generalize well across similar room layouts and demonstrate similar performance in simulation and in the real environment.

# 2. Types of Multi-robot Systems

There are many ways to classify multi-robot systems. Three of the more commonly used criteria are

- Composition: Whether the individuals in the team are homogenous or heterogeneous in terms of roles, capabilities etc.
- Communication: Whether communication is enabled between robots in the team.
- Control logic design: Deliberative vs reactive

In this project, we will be designing the logic for a reactive homogenous team with no communication between individuals.

# 3. Task Description

The task is for the robot team to find and clear targets in an unknown urban environment. This is a generic task, encompassing practical applications like bomb disposal, waste clearance and search and rescue. To reduce the computation load on each robot, the robots do not have localization capabilities, nor do they communicate with each other explicitly. The operational requirement specifies that at least 70% of the targets must be found within the limited time given. This requirement balances the completeness of the solution with the time constraint. To design the control logic by hand for such a task will be very time consuming. Thus this problem is well-suited for testing our proposed approach of multi-robot auto-programming.

# 4. Evolving Control Logic for a Robot Team

This section describes the SRM system that we use to automatically generate the control logic for a homogenous robot team to cooperatively accomplish the given task.

## *4.1 System Architecture*

The system architecture is illustrated in Fig. 2. There are 3 major components in the system:

- The rule engine on each robot that makes decision on what action to take given the current local perception of the environment
- The Evaluator that monitors the performance of the robot team towards achieving the given task and provides feedback in the form of a reward signal.
- The learning system, based on the hybrid approach of Genetic Algorithm (GA) [2, 3] and Reinforcement Learning (RL) [4, 5], that incrementally improves the

performance the of robot team using the reward signal and the pooled experience of the robots.

Note that in the simulation, each robot is driven by its own rule engine. This means that the control of the robot team is distributed. To create a homogenous robot team, a common rulebase is used to drive all the robots.

The components are described in greater detail in the sections that follow.

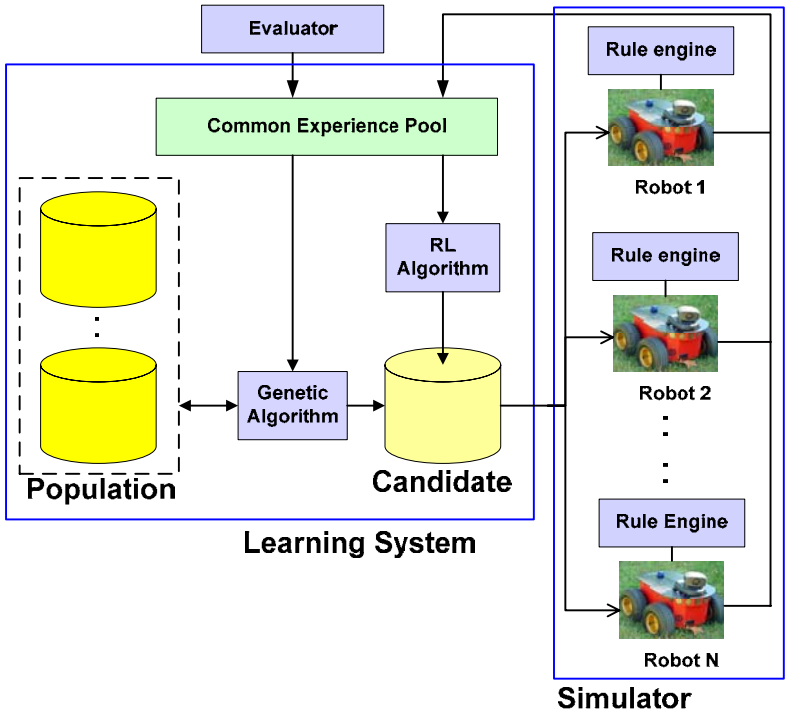


Fig. 2 The SRM system architecture.

## *4.2 Control Logic Representation and Inference*

In the SRM system, the control logic is represented as a set of IF-THEN rules. The advantage of using such a representation is that knowledge of the human expert can be easily incorporated into the learning process as such knowledge is naturally expressed as IF-THEN rules. Example of a rule in our system is given below.

```
RULE AvdFrontRobot  (strength = 0.85)
IF       ( Stall = false              AND
           No_of_tracks > 0)          AND
EXIST ( Range < 2.5                   AND
          Bearing = [-15, 15]         AND
          Identity = Robot)
THEN   Action = TurnLeft (Speed = 0, Turn_rate = 30)
```

This rule is interpreted as: If (a) I'm not stalled and (b) I sensed an object and (c) this object is a robot in my front sector very near to me, then my reaction will be to turn left on the spot and at the maximum turn rate to avoid the other robot.

The strength of a rule, a value between 0 and 1, is related to its priority. Its role will become apparent when we explain how the rule engine works below.

Given a rulebase and the current perceived state of the environment, the rule engine decides what action should be taken. The inference process of the rule engine can be summarised as follows:

1) Given the current state s, create the active set A(s), which is the set of rules with conditions matching the current state
2) If |A(s)| > 1 (because the rules are not mutually exclusive), select the rule with the highest strength to fire. If there is more than 1 rule with the highest strength, then randomly pick one among them.
3) Execute the action of the selected rule and transit to the new state. Go to step 1).

## *4.3 Learning Algorithm*

Our learning approach combines GA and RL. The reason behind adopting a combined approach is there are actually two aspects of the solution to learn, and the individual approaches tackle different aspects. The first aspect is to learn the set of rules that make up the solution. This will involve searching through the rule space for good rules. GA is suitable for tackling this aspect. However, during the search process, GA does not guarantee that the rules in the solution are mutually exclusive. In this situation, the strength of the rule can be used to distinguish the 'good' rules from the 'bad' ones. The strength of a newly created rule may be set to a default value, but this should be modified according to its performance, which in turn is measured by the reward from the Evaluator. RL is suitable for strength learning as it can incrementally tune the strength of the rule in relation to the series of rewards it received. A rule that receives a reward higher than its current strength will have its strength increased and vice versa. It is now apparent that the individual approaches that in our proposed algorithm complement each other to form an elegant overall solution. The skeleton of the algorithm is given below

***Algorithm* LearnRules**

```
1 Generation t := 0, Experience pool e:=NULL, done:=false
2 Initialize population p(t)
3 WHILE (!done)                    DO
4       FOR each rulebase r in population p(t)
5               e := Evaluate(r)
6               r := CreateAndDeleteRules(r, e)
7               FOR i: 1 to MAX_EPISODES
8                       e := Evaluate(r)                    } RL loop
9                       r := DistributeReward(r, e)
10              END FOR
11              Evaluate(r)                                 } GA loop
12      END FOR
13      done := CheckStoppingCondition(p(t))
14      IF (!done)
15              p(t+1) := Select(p(t))
16              p(t+1) := Crossover(p(t+1))
17              t := t + 1
18      END IF
19 END WHILE
```

*Algorithm* **Experience pool e := Evaluate(Rulebase r)**
20 e :=NULL
21 Duplicate r across all simulated robots
22 FOR t= 1 to MAX_STEPS
23 Make Decision for all simulated robots
24 Record experience for each robot i as $<s, A>_{i,t}$ (s is current state, A is the set of active rules)
25 END FOR
26 e := $\sum_{i,t}<s, A>_{i,t}$ (Pool experience from all robots across all time steps)
27 e := e + Get team reward from Evaluator

In the algorithm there is an outer GA loop and an inner RL loop. We shall first describe the GA loop. The GA maintains a population of rulebases. Each rulebase is a potential solution to the problem and has a fitness value associated with it. The fitness value measures the performance of the rulebase at solving the problem. For every iteration of the GA loop (also called a generation), the population undergoes 3 GA operations:

1) Each rulebase in the population is modified by the rule creation and delete operations (line 6). A new rule is derived from the parent rule by modifying the condition and/or action part of the parent, and the process is driven by the pool of experience collected from a previous evaluation episode (line 5). The delete operation prevents the rulebase from getting too large by periodically removing low strength rules or subsumed rules with similar strengths from the rulebase.
2) The fitness value of each rulebase is computed after all modifications to the rulebase are completed (line 11). A new population is then created from the old population (line 15) by selecting rulebases from the old population according to their fitness values. The higher the fitness value, the higher the probability of a rulebase being selected for the new population.
3) Every pair of rulebases in the new population exchange unique high strength rules through crossover (line 16).

Next we describe the RL loop. First, the rulebase undergoes an evaluation episode (line 8). RL is then used to distribute the reward received at the end of the episode among the rules that were active, thereby updating the strength of these rules incrementally (line 9). The update rule is

$$S_i(c,a) \leftarrow S_i(c,a) + \eta[r - S_i(c,a)] \qquad (1)$$

Here $\eta$ is the learning rate, $r$ is the payoff, $S_i$ is the strength of an active rule $i$, $c$ is the rule condition and $a$ is the rule action. The list of active rules is extracted from the common experience pool.

Lastly, we shall briefly describe the Evaluate() function which is used in both the GA and RL loops. It takes in a candidate rulebase as input, simulates the robot team executing the task (lines 22 to 25), gathers the individual robot's experiences into the common pool (line 26) and requests the reward from the Evaluator module (line 27). Recall from earlier that the common experience pool is used in the GA rule creation/delete functions and the RL strength update function. The use of the common experience pool can be viewed as allowing some form of experience sharing among

the robots. This can help to accelerate the learning process as each robot now has a richer set of experiences (its own and others') to learn from.

### *4.4 Evaluator*

This module computes a team reward $r$ after each evaluation. Apparently, the reward is computed differently for different tasks. For the target search and clearance task, we design the reward function to be

$$r = [\frac{n_c + 0.5 * n_d}{n_{total}}] / [\frac{\max(0.5 * t_{\max}, t)}{0.5 * t_{\max}}] \qquad (2)$$

where $n_c$ is the number of targets cleared, $n_d$ is the number of targets detected (but not cleared), $t$ is the actual time taken, $n_{total}$ is the total number of targets, $t_{max}$ is the maximum time allowed. This formulation considers both the completeness (in terms of proportion of targets found) and efficiency (in terms of time taken) of the solution.

## 5. Experiment Setup

Experiments are conducted both in simulation and in hardware. We simulated the Pioneer 3AT equipped with a laser range finder (maximum range 4 m, 180° field-of-view) and a fiducial sensor (maximum range 2 m). The processed sensor information available to the robot are: 1) Whether a target is in its gripper, 2) Whether it has stalled, 3) The number of tracks detected, 4) For each track, the range, bearing, orientation and identity. The list of robot actions are: 1) Consume target, 2) Avoid obstacle, 3) Approach target, 4) Approach opening, 5) Turn and 6) Move straight.

The actual Pioneer 3AT robots that we used are equipped with SICK200 LMS lasers. To enable the robots to recognise each other and the targets, reflective strips are attached to both sides of the robots and the targets. Each robot is controlled by a laptop running the rule engine (Fig. 3). This means that the control of the physical robot team, similar to that in the simulation, is distributed.

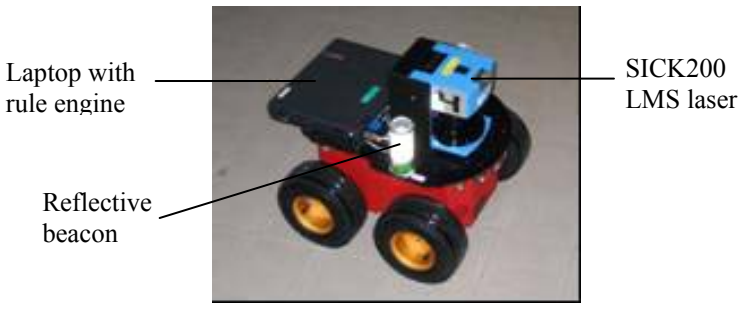


Fig. 3 The Pioneer 3AT robot used for hardware testing.

Table 1 summarizes the various setups that are used for the experiments. A scenario is defined by a fixed layout, number of robots and targets, and their positions. We generate multiple scenarios by varying the target positions for each setup. For each setup (except 1), several trials were conducted using different random seeds. The average team payoff over all trials is used as the performance metric.

Table 1: The different setups used in the experiment.

| Setup | Purpose | Layout (Area) | No. of robots | No. of targets | Max time steps | No. of scenarios |
|---|---|---|---|---|---|---|
| 1 | Learning | 8 room (252 $m^2$) | 10 | 10 | 1000 | 6 |
| 2 | Generalization test (different target positions) | 8 room (252 $m^2$) | 10 | 10 | 1000 | 6 |
| 3 | Generalization test (different no. of robots only) | 8 room (252 $m^2$) | 15- 35 | 10 | 1000 | 6 each |
| 4a | Generalization test (different layout only) | 6 room (209 $m^2$) | 10 | 10 | 1000 | 6 |
| 4b | | 8 room with narrower doors (252 $m^2$) | 10 | 10 | 1000 | 6 |
| 4c | | 10 room (382 $m^2$) | 10 | 12 | 1000 | 6 |
| 5a | Generalization test (different no. of robots and layout) | 12 room (408 $m^2$) | 20 | 15 | 1000 | 6 |
| 5b | | 16 room (441 $m^2$) | 20 | 15 | 1000 | 6 |
| 5c | | 30 room (910 $m^2$) | 35 | 30 | 1000 | 2 |
| 6 | Hardware transfer test | 5 room (106 $m^2$) | 5 | 4 | ~1000 | 5 |

Learning is conducted using setup 1 (see Fig. 4a), using a mix of random and *a priori* rules as the starting point. Each learning experiment ran for 15 generations. The best performing learned rulebase in these 15 generations is selected as the final rulebase. The generalization and transfer properties of the final rulebase are then investigated using setups 2 to 6.

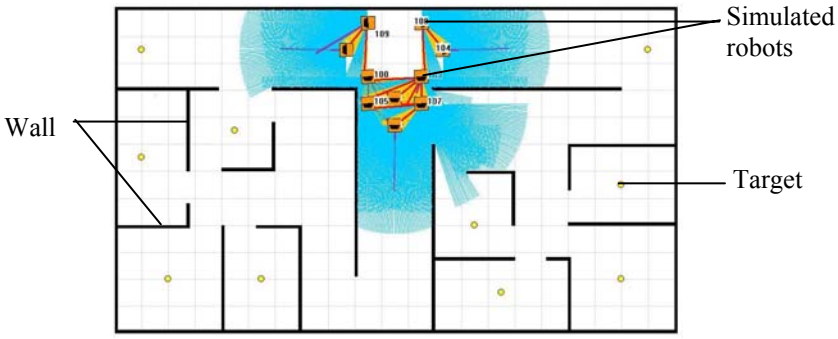


Fig. 4a Plan view of the 8-room learning layout.

Generalization refers to the performance change when the final rulebase is tested in a different setup. In practice, learning is usually conducted using a small number of robots (e.g. 10) or a subset of the full range of environment layouts as it quickly becomes computationally intractable otherwise. Thus the generalization value will indicate how well the learned rules can adapt to different operating conditions. For this test, we varied the target positions (setup 2), the number of robots for a fixed layout (setup 3), the layout for a fixed number of robots (setups 4 e.g. see Fig. 4b) and both the robot number and layout (setups 5 e.g. see Figs. 4c and 4d).

Transfer refers to the performance change when the final rulebase is tested in simulation and in hardware using the same setup. This aspect is tested using setup 6

(see Fig. 5). Because we are using a low fidelity noiseless simulator, the transfer value will be a good gauge of the sensitivity of the learned rules to noise in the actual operating environment.

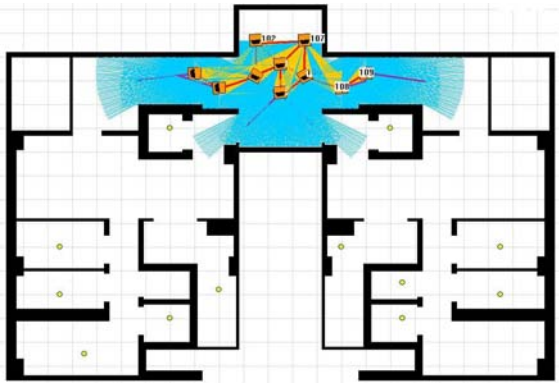

Fig. 4b Plan view of the 10-room testing layout.

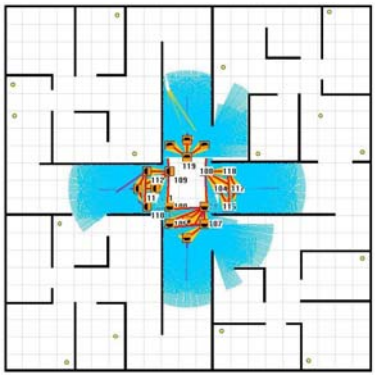

Fig. 4c Plan view of the 16-room testing layout.

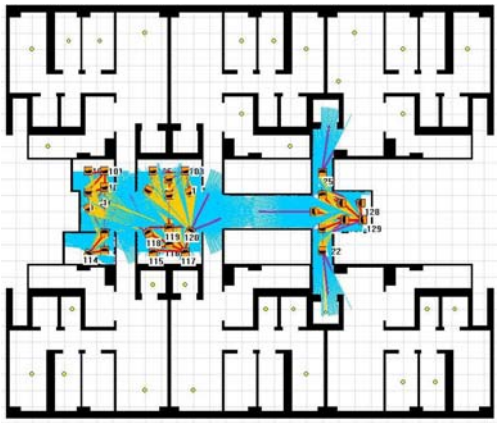

Fig. 4d Plan view of the 30-room testing layout.

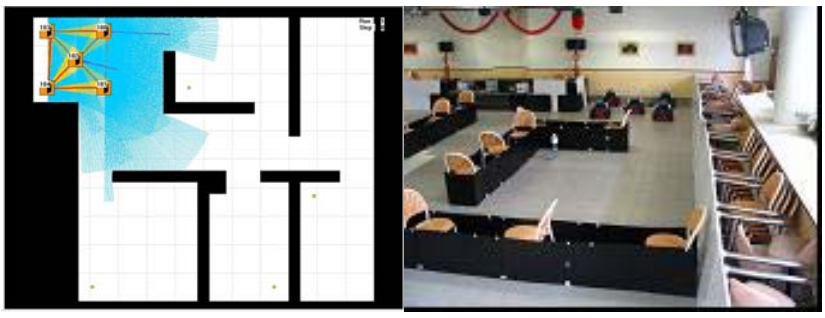

Fig. 5 Plan view of the hardware transfer testing layout.

The baseline is set based on the operational requirement. Using equation (2), it translates to a metric value of 0.35. The final rulebase should perform better than 0.35 on the learning set. Also, it is considered to have good generalization or transfer if its metric value is better than 0.35 on the test sets.

# 6. Results and Discussion

## *6.1 Generalization across Different Target Positions*

Based on Fig. 6, the learned rules performed better than the baseline on both the learning set and test set. This means that the learned rules meet the operational requirement and are robust to changes in target positions. This robustness suggests that varying target positions during learning helps to improve generalization in this case.


Fig. 6 Results with different target positions (Vertical lines represent standard deviation. The baseline is indicated by the red line.

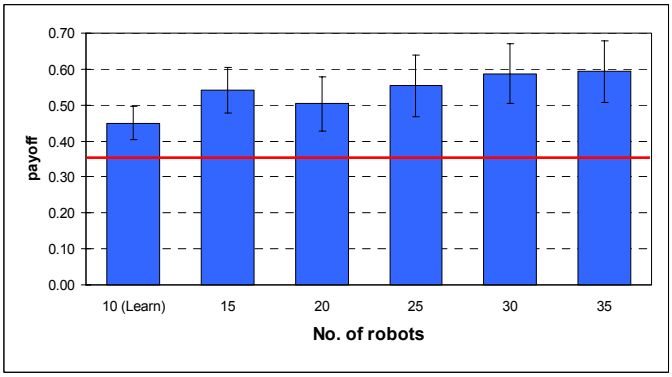


Fig. 7 Results with increasing robot numbers. (Vertical lines represent standard deviation). The baseline is indicated by the red line.

## *6.2 Scalability across Different Robot Numbers*

The trend obtained (Fig. 7) is typical of general scalability experiments: more significant increase in performance with initial increase in robot numbers (15, 25) followed by saturation for larger number of robots (30 and above). Given a fixed environment, apart from the possibility that increasing robot number may increase inter-robot interferences [1], another reason for this trend could be due to the varying contribution of the control logic and the robot numbers towards improving the

performance of the multi-robot system. For small number of robots, the control logic plays a major role in determining the overall performance. But as the number of robots increases, the contribution from the control logic is subsumed by that from the robot numbers. Following this trend for more robots, it is possible that there exist a critical number of robots beyond which the performance is largely determined by the number of robots. It will not be affected much by improving the control logic.

## 6.3 Generalization across Different Layouts

The complexity of a layout is determined by its structure e.g. the number of rooms and the average accessibility of the rooms. From Fig. 8, it is observed that generalization decreases with increasing layout complexity. Such a trend is expected, as generalization is dependent on the similarity of the test layout relative to the learning layout. That is why good generalization is obtained for the 6-room and 8-room (with narrower openings) but not the 10-room. In fact, we have identified the main bottleneck in the 10-room case to be the single doorway leading to the rooms (see Fig. 5b), which is not present in the other 2 cases. Theoretically, it is possible to improve generalization by using a learning set that covers a wider range of non-conflicting layouts. By non-conflicting, we mean that the complexity of the layouts should not be drastically different (e.g. a 8-room versus a 20-room) so that the assumption of the existence of a common solution to all the layouts in the learning set is still valid. For example, in our case it is possible improve generalization by adding layouts having similar prominent features as the 10-room case into our learning set.

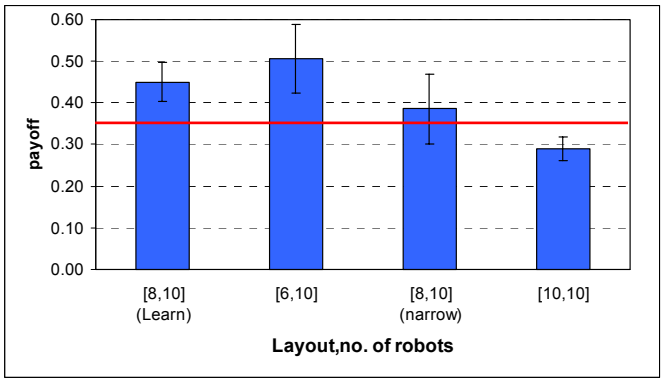


Fig. 8 Results with different room layouts. The x-axis labels are in the form [a, b], where a is the number of rooms, b is the number of robots. (Vertical lines represent standard deviation). The baseline is indicated by the red line.

## 6.4 Generalization across Different Layouts and Robot Numbers

The trend observed (Fig. 9) suggests that given sufficient number of robots, say more than the number of rooms, the expected performance on a complex problem is correlated to a scaled down version of the problem. In other words, the performance of the scaled down problem is a reasonably good prediction of the performance on a more complex problem. This result is encouraging as it suggests that it is not necessary to tackle a rather complex problem head on. Instead the better alternative is to formulate a smaller scale problem by identifying the salient features in the complex problem, and then learning on this simplified problem, which is faster and easier. The learned rules can then be applied on the original complex problem, given sufficient

number of robots. In our case the identification process is straightforward, since the layouts are simply aggregations of basic units (12 room≈2×(6 room), 16 room≈2×(8 room) (see Fig. 4c), 30 room≈3×(10 room) (see Fig. 4d)). However, for an arbitrary layout, it will not be so simple. To develop a more generic identification approach will be an area of further research for us.

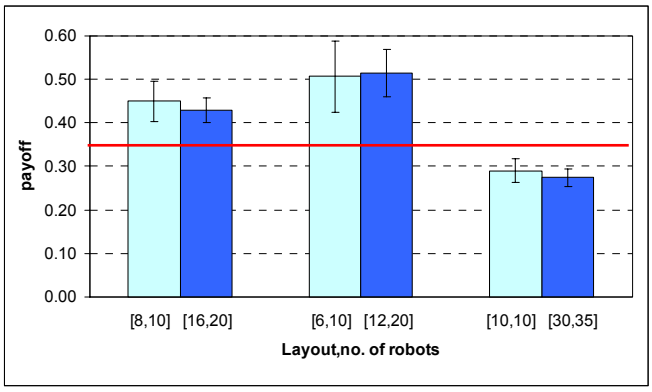


Fig. 9 Results with different room layouts and robot number. The x-axis labels are in the form [a, b], where a is the number of rooms, b is the number of robots. (Vertical lines represent standard deviation). The baseline is indicated by the red line.

### *6.5 Hardware transfer testing*

From Fig.10, it is observed that the performance of the learned rules on physical robots is comparable with that in simulation. This implies good transfer i.e. it is quite robust to noise in the actual environment. Although the result is based on limited number of trials, it demonstrates the potential of our proposed approach of auto-programming the robot team using simulation, as the learned rules worked well on the physical robots.

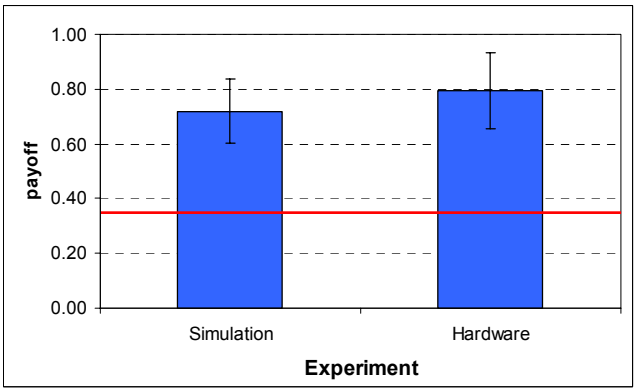


Fig 10 Performance of best learned rules in simulation and hardware. The baseline is indicated by the red line.

## 7 Conclusion and Future Work

In this paper, we present a learning approach combining GA and RL, which exploits the complementary strengths of both approaches, to learn the control logic for individuals in a multi-robot system. We successfully applied this approach to learn the control rules for individual robots in a team tasked to locate targets in a realistic urban

environment. The results showed that the learned rules generalize well to different target positions and to different layouts of similar or lower complexity to the learning layout. As the layout become more complex and different, generalization naturally decreases. But this effect may be reduced by using a learning set with more layouts. The performance of the learned rules improved with increasing robot numbers, but with diminishing returns as inter-robot interference unavoidably sets in with larger number of robots. Furthermore, the results also demonstrated that it is possible to overcome the difficulty of learning straight on a complex problem by formulating a simpler problem based on the salient features of the more complex problem, and learning on the simpler problem. Lastly, the performance on physical robots is shown to be similar to that in simulation, illustrating the potential of our approach of auto-programming the robot team using simulation.

Our future work will mainly focus on improving the learning performance of the system, particularly on the design of distributed individual payoff schemes for multi-robot systems. We will also examine more systematic approaches to "scale down" a complex problem so that learning becomes more tractable.

## *References*


[1] C. K. Cheng, G. Leng, "Cooperative search algorithm for distributed autonomous robots", in Proceedings of *IEEE International Conference on Intelligent Robotics and Systems,* 2004.

[2] D. E. Goldberg, *Genetic Algorithm in Search, Optimization and Machine Learning*, ed. 1989, Addison-Wesley.

[3] K. A. De Jong, "Using Genetic Algorithms for Concept Learning", *Machine Learning*, vol 13, pp 161-188, 1993.

[4] L. P. Kaelbling and M. L. Littman, "Reinforcement Learning: A survey", *Journal of Artificial Intelligence Research*, vol 4, pp 237-285, 1996.

[5] C. J. C. H. Watkins, "Learning from delayed rewards", Ph.D. diss., King's College, Cambridge, 1989.