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:	Khee-Yin HOW, Victor TAY, DSTA, Singapore Ye-Chuan YEO, Qing SUI, DSO NL, Singapore
Point of Contact:	Victor TAY victor_tay@dsta.gov.sg
Organisation:	Defence Science & Technology Agency (DSTA) DSO National Laboratories (DSO NL)
Complete Address:	Defence Science & Technology Agency (DSTA) Directorate of Research and Development 71 Science Park Drive, #02-05 Singapore 118523 DSO National Laboratories (DSO NL) Cooperative Systems and Machine Intelligence Lab (CML) 20 Science Park Drive Singapore 118230
Phone:	65-68795077
Email:	hkheeyin@dsta.gov.sg, victor_tay@dsta.gov.sg yyechuan@dso.org.sg, sqing@dso.org.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 manual and centralized tele-operation to autonomous and de-centralized operation, such that there is shared awareness, effective collaboration and action synchronization amongst the unmanned vehicles.

Traditionally, the control logic for unmanned vehicle operations is manually handcrafted. However, as the nature of operations becomes more complex and diversified, this manual approach of logic programming becomes extremely difficult. Therefore, a more efficient method to program the control logic is required.

In this paper, we describe how Modelling and Simulation (M&S) can be leveraged to provide a learning environment to evolve control logic for an unmanned vehicle team to enable autonomous operation. 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). The 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. In this paper, we describe our work where we successfully applied SRM to evolve the control logic for a robotic team to carry out search and locate mission in an urban 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 manual and centralized tele-operation to autonomous and de-centralized operation. 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 manually handcrafted with the aid of M&S tools [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. Hence, more efficient methods are required.

In this paper, we describe how M&S can be used to provide a learning environment to evolve the control logic for the unmanned vehicle team. The approach we used coupled a rule learning system based on Genetic Algorithm (GA) and Reinforcement Learning (RL) to an M&S system. We call this hybrid approach Simulation-based Rule Mining (SRM) (Fig. 1).

The 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 direct learning on the physical robots, SRM has the advantage of ease in repeating trials, while mitigating risks of damage to the hardware.

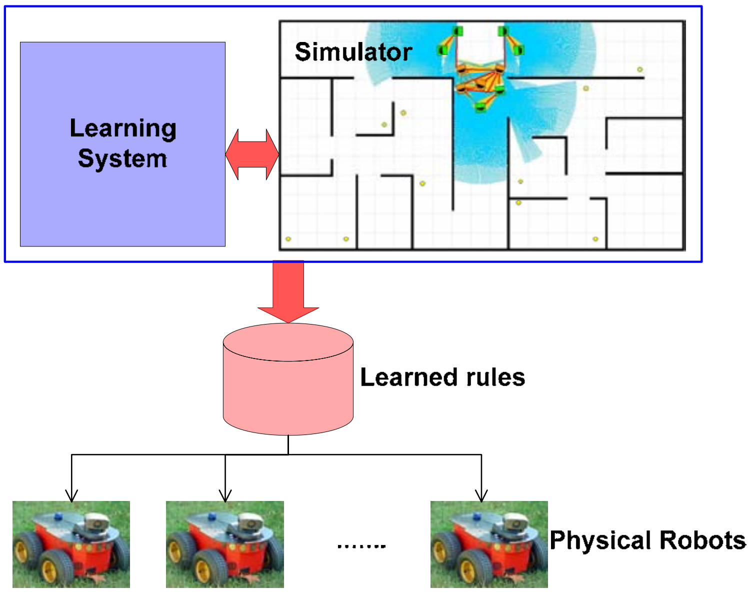

Fig. 1 – Outline of SRM process

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 and this makes it easy for a priori knowledge to be incorporated into the system to bootstrap the learning process. A hybrid 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 rule bases generated by GA usually contain non-mutually exclusive rules. RL complements by differentiating the ‘good’ rules from the ‘bad’. RL can incrementally tune the priority of individual rules according to their performance so that better-than-average rules have higher priorities than others in the long run.

We tested the SRM approach to evolve the logic for controlling a robotic team to carry out search and locate mission in an urban 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, with the robots demonstrating similar performance in simulation and in the real environment.

2. Task Description

Multi-robot systems are commonly classified as follows:

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

In this experiment, the aim was to design the logic for a reactive, homogenous robotic team with no communication between individuals. The task for the robot team was to find and clear targets in an unknown urban environment. This generic task can be translated into practical applications such as bomb disposal, waste clearance and search and rescue.

To reduce the computation load on each robot, the robots did not have localization capabilities, nor did they communicate with each other explicitly. Considering that the trade-off of using such ‘simple’ robots is that a complete solution cannot be guaranteed given finite time, we established an experimentation baseline of 70% of the targets found within the limited time given as basis for comparing our results. This baseline balanced the completeness of the solution with the time constraint. To design the control logic by hand for such a task would be very time consuming. Hence, this problem was well suited for testing our proposed approach of multi-robot auto-programming.

3. Evolving Control Logic for a Robot Team

This section describes the SRM system that was used to automatically generate the control logic for a homogenous robot team to cooperatively accomplish the task described above.

3.1 System Architecture

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

- The rule engine of each robot that decides what actions 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 GA [2, 3] and RL [4, 5], that incrementally improves the performance of the robot team using the reward signal and the pooled experience of the robots.

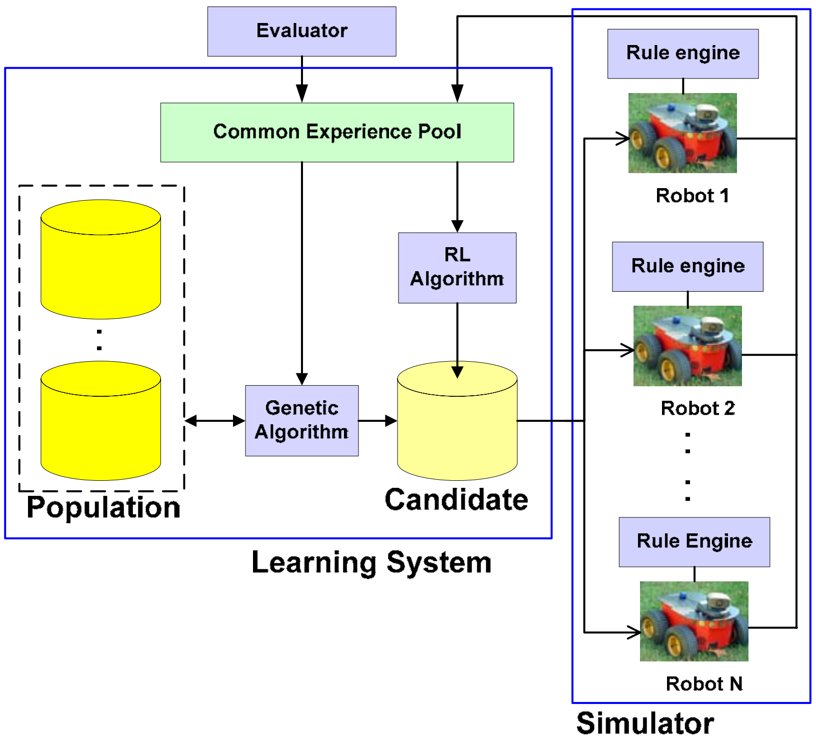

Fig. 2 – The SRM system architecture

Note that in the simulation, each robot was driven by its own rule engine. This means that the control of the robot team was distributed. To create a homogenous robot team, a common rule base was used to drive all the robots. The components are described in greater detail in the sections that follow.

3.2 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. An 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 rule base 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. 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).

3.3 Learning Algorithm

Our learning approach combines GA [2, 3] and RL [4, 5], with each technique tackling two different aspects of the solution to be learned.

The first aspect is to learn the set of rules that makes up the solution. This will involve searching through the rule space for good rules and GA, a stochastic search procedure based on the principles of natural evolution, 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 with respect 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. The individual approaches 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. The GA maintains a population of rule bases. Each rule base is a potential solution to the problem and has a fitness value associated with it. The fitness value measures the performance of the rule base 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 rule base is computed after all modifications to the rule base are completed (line 11). A new population is then created from the old population (line 15) by selecting rule bases from the old population according to their fitness values. The higher the fitness value, the higher the probability of a rule base being selected for the new population.

3) Every pair of rule bases in the new population exchange unique high strength rules through crossover (line 16).

Within the RL loop, 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 η 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.

The Evaluate() function, used in both the GA and RL loops, takes in a candidate rule base as input, simulates the robot team executing the task (lines 22 to 25), gathers the experiences of the individual robots 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.

3.4 Evaluator

This module computes a team reward r after each evaluation. 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.

4. Experiment Setup

4.1 Objectives

The aim of the experiments is twofold:

(a) To evaluate the effectiveness of the SRM approach in developing a solution to our problem in a synthetic environment.

(b) To investigate the generalization and transfer properties of the learned rules.

Generalization refers to the performance change when the learned rules are tested in an environment different from the one during learning. For our problem, learning is usually conducted using a small number of robots (e.g. 10) or a subset of the full range of environment layouts; otherwise it quickly becomes computationally intractable. The generalization value will indicate how well the learned rules can adapt to different operating conditions such as an increase in robot number or change in environment layout.

Transfer refers to the difference in performance when the learned rules are tested in simulation and in hardware using the same environment setup. 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.

4.2 Experiment Design

We designed multiple setups to meet the experimental objectives. A summary of the various setups that are used for the experiments is given in Table 1.

Learning is conducted using Setup 1 (see Fig. 3a for the layout), using a mix of random and *a priori* rules as the starting point. The target positions were varied during the learning process to simulate noise. Each learning experiment ran for 15 generations. The learned rule base with the highest average team reward over the set of target positions in these 15 generations is selected as the final rule base. The generalization and transfer properties of the final rule base are then investigated using Setups 2 to 6. Figures 3b and 3c illustrate the layout of a generalization and transfer test respectively. For each test setup, 5 to 10 trials were conducted using different random seeds and slightly varying target positions. The average team reward over all the trials is used as the test performance.

To determine if the SRM approach was effective, or how well the learned solution generalized and transferred to hardware, we compared the performance of the learned rules against our baseline of 70% of targets found in the limited time given (see section 2). Using equation (2), this translates to a metric value of 0.35.

Setup	Purpose	Layout (Area)	No. of robots	No. of targets	Environment
1	Learning	8 room (252 m^2)	10	10	
2	Generalization test (different target positions)	8 room (252 m^2)	10	10	
3	Generalization test (different no. of robots only)	8 room (252 m^2)	20 – 35	10	
4a		6 room (209 m^2)		10	
4b	Generalization test (different layouts only)	8 room with narrower doors (252 m^2)	10	10	Synthetic
4c		10 room (382 m^2)		12	
5a		12 room (408 m^2)	20	15	
5b	Generalization test (different no. of robots and layout)	16 room (441 m^2)	20	15	
5c		30 room (910 m^2)	35	30	
6	Hardware transfer test	5 room (106 m^2)	5	4	Synthetic and real

Table 1: The different setups used in the experiment. Note that for all the setups, t_{max}= 1000.

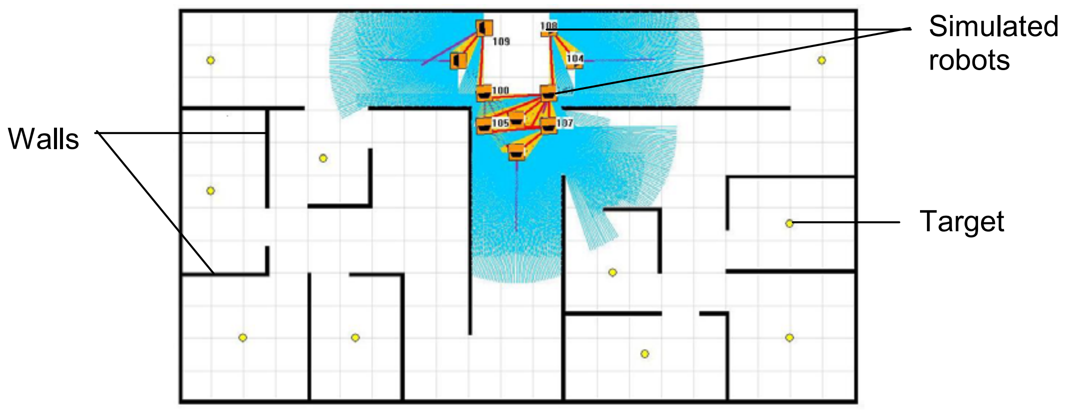

Fig. 3a – Plan view of the 8-room learning layout

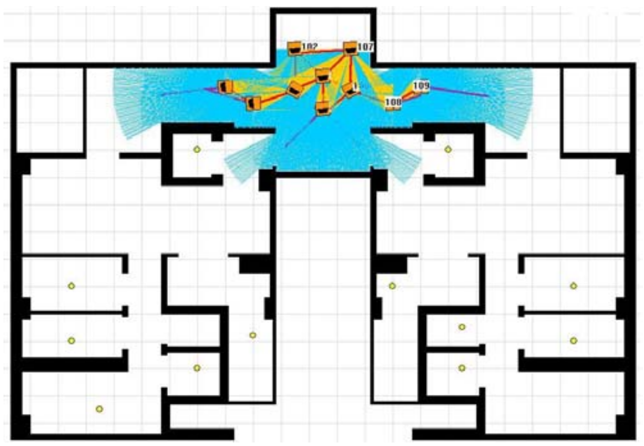

Fig. 3b – Plan view of the 10-room generalization test layout. Note the increase in complexity of this layout over the learning one.

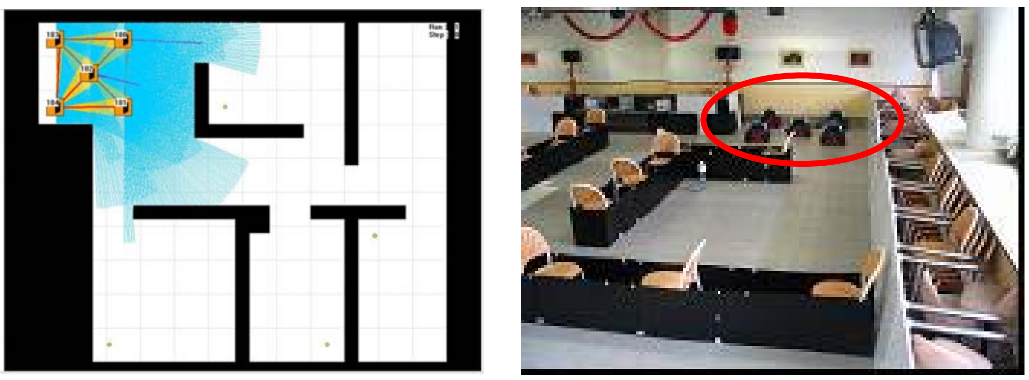

Fig. 3c – Plan view of the hardware transfer test layout. The physical robot team is highlighted in red.

4.3 Implementation Details

We simulated the Pioneer 3AT equipped with a laser range finder (effective range 4 m, 180° field-of-view) and a fiducial sensor (effective range 2m). The processed sensor information available to the robot comprised: 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 comprised: 1) Consume target, 2) Avoid obstacle, 3) Approach target, 4) Approach opening, 5) Turn and 6) Move straight.

The actual Pioneer 3AT robots used were equipped with commercial off-the-shelf laser range finders (SICK LMS200). To enable the robots to recognise each other and the targets, reflective strips were attached to both sides of the robots and the targets. Each robot was controlled by a laptop running the rule engine (Fig. 4). This meant that the control of the physical robot team, similar to that in the simulation, was distributed.

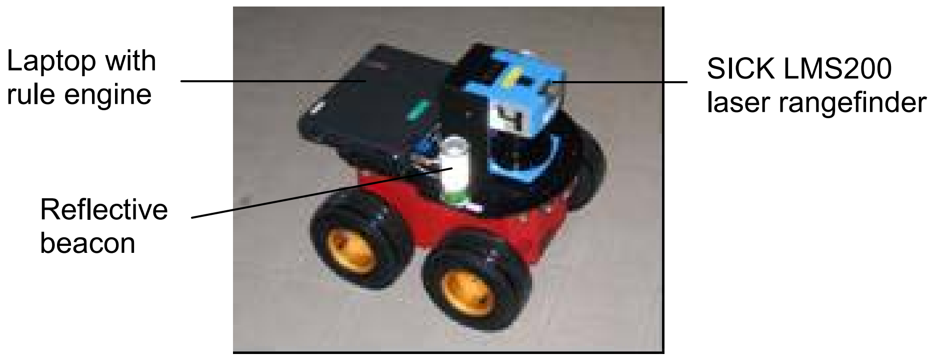

Fig. 4 – The Pioneer 3AT robot used for hardware testing

5. Results and Discussion

This section presents the results of the experiments. Refer to Table 1 in the previous section for more details of the experiment setup.

5.1 Generalization across Different Target Positions

Based on Fig. 5, the learned rules performed better than the baseline on both the learning setup and test setup. This meant that the learned rules a) constituted a viable solution to the multi-robot search and locate problem and b) were robust to changes in target positions (The t-test result suggests that the means are similar at the 95% confidence level). This robustness suggested that varying target positions during learning helps to improve generalization in this case.

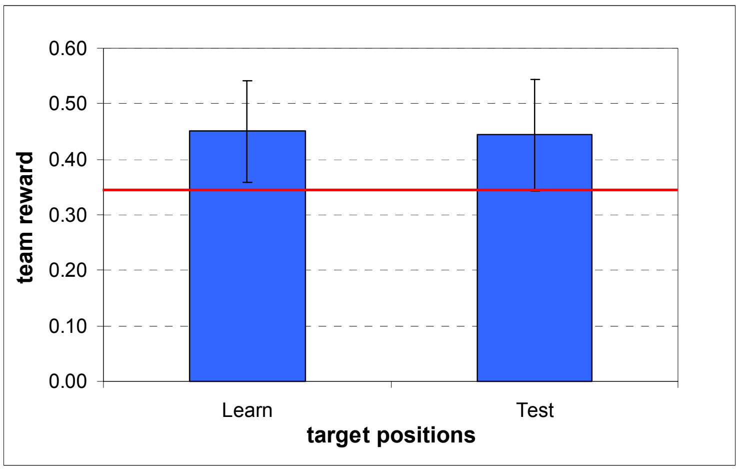

Fig. 5 – Results with different target positions (mean ± std dev).The baseline is indicated by the red line.

5.2 Scalability across Different Robot Numbers

The trend obtained (Fig. 6) was typical of general scalability experiments: more significant increase in performance with initial increase in robot numbers (20, 25) followed by saturation for larger number of robots (30 and above). Given a fixed layout, increasing robot number would initially increase the efficiency of the search process until a point where this improvement is negated by the increase in inter-robot interferences due to overcrowding.

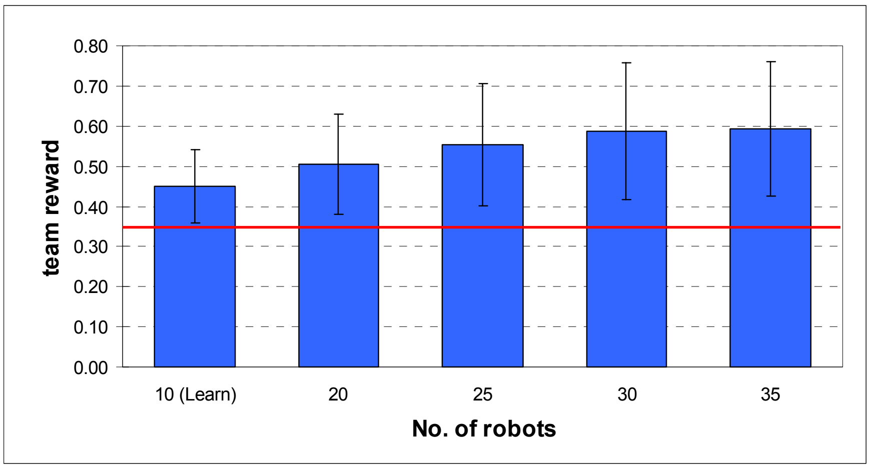

Fig. 6 – Results with increasing robot numbers (mean ± std dev). The baseline is indicated by the red line.

5.3 Generalization across Different Layouts

The complexity of a layout was determined by its structure, e.g. the number of rooms and the average accessibility of the rooms. From Fig. 7a, it was observed that generalization decreased, as the test layouts became more complex and different from the learning layout. This was to be expected, as generalization was dependent on the similarity of the test layout relative to the learning layout. In fact, we identified the main bottleneck in the 10-room case to be the single doorway leading to the rooms (see Fig. 7b). This feature, which greatly decreased the accessibility of the rooms, was not present in the other two test layouts or the learning layout. To improve generalization on the 10-room case, we could enrich the feature set in the current learning layout with prominent features present in the 10-room test layout. This would have the effect of reducing the difference between the learning and test layout. However, the trade-off of doing so was a possible increase in learning time, as the new learning layout was more complex.

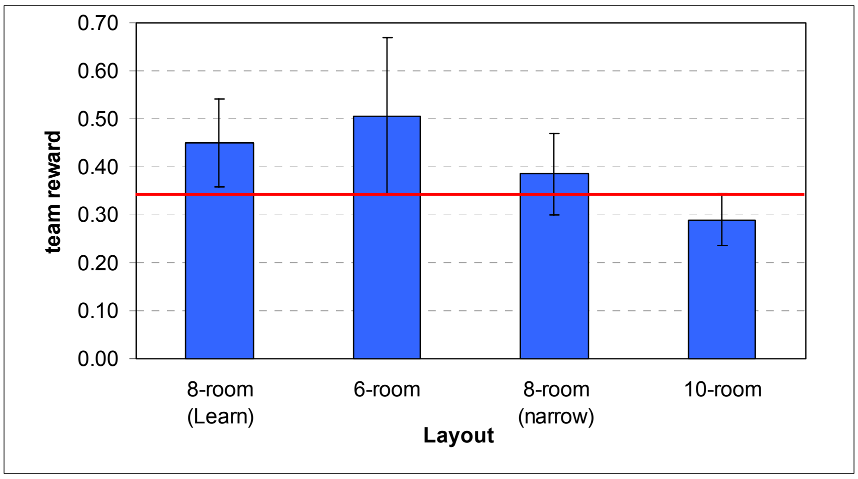

Fig. 7a – Results with different room layouts (mean ± std dev). The number of robots is fixed at 10 throughout. The baseline is indicated by the red line.

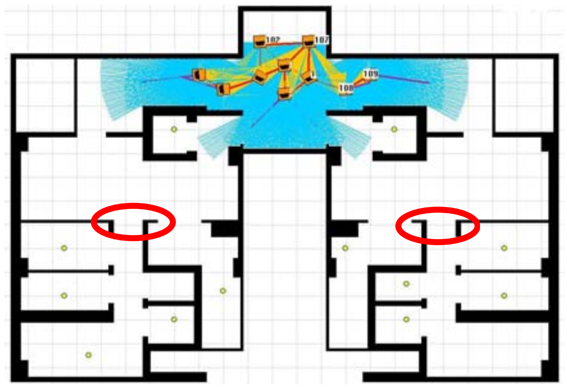

Fig. 7b – Illustration of bottleneck in the 10-room layout (in red). There are no alternative paths to the targets in the rooms other than through this doorway.

5.4 Generalization across Different Layouts and Robot Numbers

The trend observed (Fig. 8a) suggested that the expected performance on a complex problem was correlated to a scaled-down version of the problem. This result was encouraging as it suggested that it was not necessary to tackle a rather complex problem head on. Instead the better alternative might be to formulate a smaller scale problem by identifying the salient features in the complex problem, and then learning on this simplified problem, which would be faster and easier. The learned rules could then be applied on the original complex problem, given sufficient number of robots. In our case, the identification process was straightforward, since the layouts were simply aggregations of basic units (12-room ≈ 2 × (6-room), 16-room ≈ 2 × (8-room) (see Fig. 8b), 30-room ≈ 3 × (10- room)).

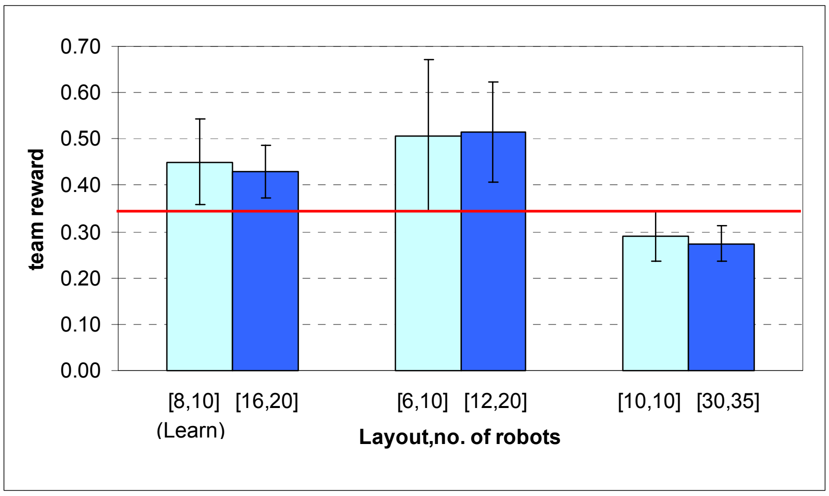

Fig. 8a – Results with different room layouts and robot number (mean ± std dev). The x-axis labels are in the form [a, b], where a is the number of rooms, b is the number of robots. The baseline is indicated by the red line.

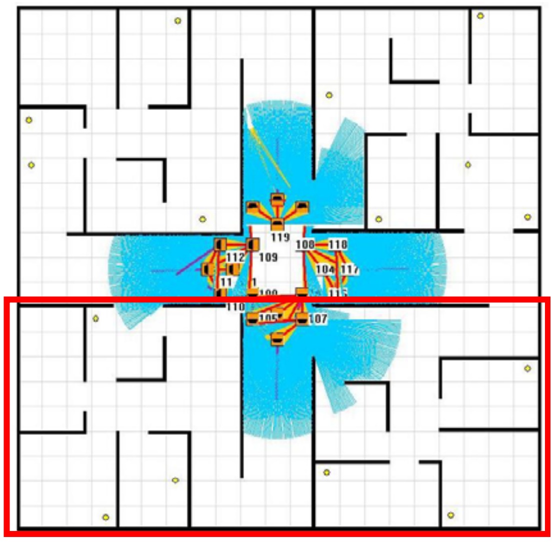

Fig. 8b – Plan view of the 16-room testing layout. The 8-room learning layout is highlighted in red.

5.5 Hardware transfer testing

From Fig. 9, it was observed that the performance of the learned rules on physical robots was comparable with that in simulation (the t-test result suggests that the means are similar at the 95% confidence level). This implied good transfer i.e. it was quite robust to noise in the actual environment. Although the result was based on limited number of trials, it demonstrated the potential of our proposed approach of auto-programming the robot team using simulation, as the learned rules worked well on the physical robots.

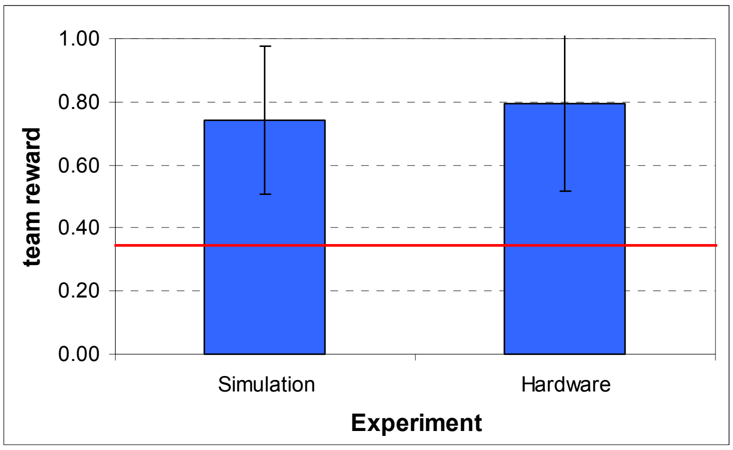

Fig 9 – Performance of best learned rules in simulation and hardware (mean ± std dev). The baseline is indicated by the red line.

6. Conclusion and Future Work

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

The results showed that:

- The learned rules generalized well to different target positions.
- Good generalization was observed for test layouts of similar or lower complexity to the learning layout. As the layout became more complex and different, generalization naturally decreased. However, this effect might be reduced by using a learning layout with richer set of features.
- The performance of the learned rules improved with increasing robot numbers, but with diminishing returns as inter-robot interference unavoidably set in with larger number of robots.
- It was 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.
- The performance on physical robots was shown to be similar to that in simulation, illustrating the potential of our approach of auto-programming the robot team using simulation.

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.

Acknowledgements

The authors would like to thank our colleagues Chee-Wee Wong, Liang-Ping Tan and Chee-Kong Cheng for insightful discussions and help in the implementation of the simulator and hardware.

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.