JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 30, 637-652 (2014)

DOI:10.1688/JISE.2014.30.3.6

Multi-Step Learning to Search for Dynamic Environment Navigation*

CHUNG-CHE YU[1] AND CHIEH-CHIH WANG[1,2]
[1]*Graduate Institute of Networking and Multimedia*
[2]*Department of Computer Science and Information Engineering*
National Taiwan University
Taipei, 116 Taiwan
E-mail: fish60@robotics.csie.ntu.edu.tw; bobwang@ntu.edu.tw

While navigation could be done using existing rule-based approaches, it becomes more attractive to use learning from demonstration (LfD) approaches to ease the burden of tedious rule designing and parameter tuning procedures. In our previous work, navigation in simple dynamic environments is achieved using the Learning to Search (LEARCH) algorithm with a proper feature set and the proposed data set refinement procedure. In this paper, the multi-step learning approach with goal-related information is proposed to further capture the successive motion behavior of the user in complex environments. The behaviors of the demonstrator could be matched by the motion control module in which policies of the demonstrator are well captured.

Keywords: learning from demonstration, robot navigation, dynamic environments, learning to search, motion behavior learning

1. INTRODUCTION

The learning from demonstration (LfD) [1] techniques have been applied to develop control policies for mobile robots in recent years [2-6]. It would be natural and attractive to find a way to teach a robot to learn behaviors from demonstrations instead of complex rule designing, modeling, and parameter tuning procedures. It had been shown that the control policies in static environments could be learned using such LfD framework [6]. Thus, we further focus on teaching a robot to navigate in dynamic environments in this paper.

The issues of the "freezing robot problem" in robot navigation are discussed in detail [5]. Briefly speaking, a robot may freeze in one place or perform unnecessary maneuvers to avoid collisions. The goal of our work is to accomplish collision-and freezing-free navigation in dynamic environments. A model-free approach is used in which the motion control parameters only come from the implemented Learning to Search (LEARCH) [7] algorithm. The LEARCH algorithm had been shown that the robot locomotion and manipulation problems could be solved using the functional gradient methods [2]. In our previous work, the one-step greedy approach is further used to accomplish reactive motion control in a simple dynamic environment with the modification of a data refinement procedure [8]. The learning performance is enhanced due to this modification to well capture the behaviors of the demonstrator. The main assumption of the proposed ap-

Received February 28, 2013; accepted June 15, 2013.
Communicated by Hung-Yu Kao, Tzung-Pei Hong, Takahira Yamaguchi, Yau-Hwang Kuo, and Vincent Shin-Mu Tseng.
* This work was supported in part by National Science Council, National Taiwan University and Intel Corporation under Grants NSC 100-2221-E-002-238-MY2, NSC 102-2221-E-002-179, NSC 100-2911-I-002-001, and 101R7501.

proach is that the demonstrator would perform the same policy during the demonstration. The goal location is given from the other high level planning module and a valid control policy should exist for the demonstrator to navigate through the environment. The collected data could be used to extract both the perception and action feature vectors. Then the control policy for navigation could be learned using the proposed approach with these perception and action features.

To further analyze the performance of the proposed algorithm and step closer to the real-world setting, the experiments from a more complex simulation world are studied. It is discovered that the information of the goal is essential for the learner to learn the policy. Second, the learned control policy may not be stable during successive steps as the demonstrator because of the one-step greedy method. The multi-step learning algorithm with goal information is developed to solve these issues. It is assumed that the next few N steps worlds could be predicted based on the chosen actions of the controlled agent and other moving agents. The perception feature could then be extracted accordingly with the corresponding control feature.

The rest of this paper is organized as follows. Related works are briefly reviewed in section 2. The learning algorithm is presented in section 3. Our previous work will be briefly described and the multi-step learning algorithm with goal information will be covered. The experimental results in complex environments are presented in section 4. The conclusion and future work are listed in section 5.

2. RELATED WORKS

Robot navigation in dynamic environments is mostly accomplished using two modules: one is a planning system which generates a trajectory to reach the goal; the other is a local reactive motion control system which guides the robot to follow the planned trajectory and avoid any static and dynamic obstacle with imperfect environment models [9]. The failure cases of the planning module for robot navigation are discussed in detail and the rule-based nearness diagram (ND) navigation method [10, 11] is used as the reactive navigation module to avoid the dead states when the planning module fails. For the failure cases of the ND method, the robot stops and then turns around on itself to update the model of environments in all directions for finding any potentially navigable valley. The drawbacks of the ND method could be that it is hard to tune the parameters in crowded and highly dynamic environments. In addition, it could be challenging to generate human-like motion patterns based on the manmade rules.

Still based on a planning model, a crowd simulator is used to learn a path planner from example traces [4]. The planned path is generated based on the prior knowledge about the environment. Utilizing the information from local perception during the navigation, the values of environmental factors such as moving direction and velocity of the surrounding agents are continuously updated and the path to the goal location will also be refined if necessary. It is assumed that the estimation of the environmental features is updated at every time step and the re-planning is performed every few time steps. To plan such a path for the next few time steps, the prediction of the flow features from the moving crowds is needed. Thus, the Gaussian processes (GP) model [4] is used for the environmental feature estimation.

However, the GP model may not be feasible to solve the “freezing robot problem”.

The issues of the "freezing robot problem" are argued in [5]: due to the uncertainty explosion of moving agents, all forward paths may be decided as unsafe by a planner. Although the quality of the motion prediction could be improved, without noticing the fact of joint collision avoidance, the robot may still freeze in place or perform unnecessary maneuvers to avoid collisions. To solve the robot freezing problem, the researchers are inspired by the real world observation in which the phenomenon of joint collision avoidance could be observed in the real world for pedestrians to make sufficient spaces for navigation. Accordingly, the interaction Gaussian processes (IGP) model [5] is developed to estimate the crowd interaction from data by introducing the dependencies between Gaussian processes. The independent GP priors are coupled by multiplying in an interaction potential and the interaction could be modeled to present the occurrence of joint collision avoidance. The interaction potential contains a parameter to control the safety distance of the repulsion and another parameter for its strength. Even though some different settings and the resulting interaction potentials are shown in their work, it could be still challenging to tune the parameters. In the real world problems, this could be even worse as the interaction potential should not be the same based on each moving agent's own policy. Using one universal interaction potential to introduce the dependencies for the crowds could be improper whereas introducing parameters for each agent is even more impractical due to the high complexity.

Other than using the LfD frameworks, the overall collision probability can be used to rank candidate trajectories for accomplishing the navigation by considering the probability of colliding [12]. It is checked how serious the collision will be instead of checking if a collision occurs. In general, the aim of a robot is to reach its goal position while minimizing its collision probability. Although the navigation could be done in such settings, lots of parameters tuning and setting processes are still needed.

In our previous work, navigation in simple dynamic environments is achieved using the LEARCH algorithm with a proper feature set and the proposed data set refinement procedure [8]. In this paper, to further capture the successive motion behavior of the user in complex environments, the multi-step learning approach is proposed. The behaviors of the demonstrator could be matched by the reactive motion control module in which policies of the demonstrator are well captured.

3. THE LEARNING ALGORITHM

In this section, our previous one-step greedy approach with data set refinement procedure and learning results are briefly reviewed. Some implementation details may be omitted due to the page limit and can be found in our previous work [8]. Then the performance and issues in a more complex world are discussed. The multi-step learning approach with goal-related information is proposed to deal with these issues. The performance is significant enhanced in terms of training error and the testing behavior.

3.1 Implementation in a Simple Simulation World

The assumption of the original LEARCH algorithm is that the mapping from states to features is static and fully known a priori [3]. However, if the mapping from states to features is dynamic or not known a priori, the planned paths could not be generated dur-

ing the training process. Thus, the main difference between the original LEARCH algorithm and our previous works is that each example path is seemed as many example instances. Instead of planning a path, the one step greedy loss-augmented cost states are computed based on instances from the example paths [6, 8].

During the demonstration, the motion control data and the maps of the training environments are recorded to form the feature vectors for the training process. The full feature vector consists of the perception feature and the control feature. The recorded map with agents is used to extract local perception data from the user-controlled agent and a nine-dimensional feature vector is formed by the perception data. The nine entries consist of the left, ahead, and right information of the observed environment from the agent which are further composed of the Manhattan distance between the agent and the obstacle, the type of the obstacle, and the speed of the obstacle. –1 is used for static obstacles, 0 for clear spaces if there is no obstacle within the limit range, and 1 for moving agents. When there is no obstacle within the limit range, the distance is set to a maximum value and the speed is set to 0. Here, the maximum value is 10 because a grid world of size 3 by 8 only is used in our implementation. The left most and right most boundaries of the simulation world are considered as static obstacles whereas the top most and bottom most boundaries are considered as clean spaces. Then, the control feature is formed based on the control commands. The control feature is a four-dimensional binary feature to indicate the commands of *up*, *left*, *stop*, or *right*. Thus, if the user decides to *stop*, the control feature should be (–1, –1, 1, –1).

The procedure of the algorithm is presented in Algorithm 1 with step number $N = 1$ and the feature function mentioned above. The simple 0-1 loss function is used. In line 5, for each valid control command, the next one step feature is calculated. In line 6, using the current learned policy applied with the extracted features, the minimum cost one will then be selected to be the positive example for line 7. The corresponding user-provided demonstration data would be the negative example. The AdaBoost [13] algorithm with decision stump is then used to do the classification in line 10 to get the log-hypothesis h_t for the log-costmap updating procedure. The training process is terminated if the learned examples are close enough with the demonstration.

Algorithm 1 The revised LEARCH Algorithm

Require: training data, loss function, feature function, step number N

1. Initialize log-costmap to zero $S_0 = 0$
2. **for** training iteration $t_{train} = 1$ to $T - 1$ **do**
3. Initialize the data set to empty
4. **for** training data $i = 1$ to I with length larger than N **do**
5. Compute the loss-augmented costmap / states for N step(s)
6. Find the minimum cost loss-augmented path / state
7. Generate positive and negative examples
8. **end for**
9. Form a data set without ambiguity for some $t_{train} > 0$
10. Train a regressor or classifier on the generated data set to get log-hypothesis h_t
11. Update the log-costmap $S_{t+1} = S_t + \alpha_t h_t$ with step size α_t
12. **end for**
13. Return final costmap $\exp(S_t)$

It is observed that during the training, the training error of the classifier in line 10 may increase to indicate that many learned examples are the same as the demonstration. This could be shown in the blue curve in Fig. 1. The positive and negative examples could not be differentiated by the classifier because each pair of the corresponding positive and negative examples have the same feature value but with different labels. While those learned examples are consistent with the demonstration, there still may be some learned examples which are inconsistent with the demonstration. These ambiguous examples which would cause bad performance to the classifier should be eliminated while the others are still usable for the classifier. Thus, the line 9 is added to the algorithm compared with the original LEARCH algorithm.

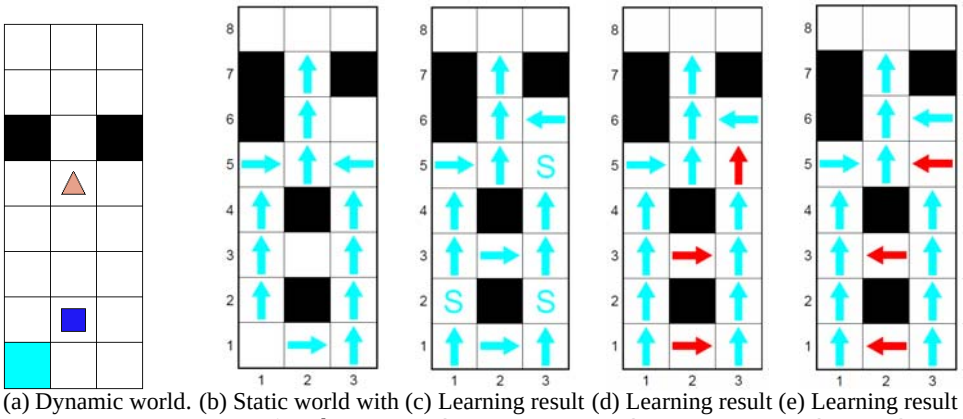
Fig. 1. The training error of the learned policy with loss and the classifier. As the training iteration increased, the error of the learned policy cost map exp (S_t) decreases while the error of the classifier in Algorithm 1 line 10 increases.

After some training iteration $t_{train} > 0$, if it is observed that the training error of the classifier exceeds some certain value like 0.4, the collected examples should be refined to get an example set without ambiguity. The learned policy could then be further improved in the following iterations.

Throughout this paper, there are three types of training error and will be described here for better understanding. The first type of training error comes from the classifier in the LEARCH algorithm and is called “training error of the classifier” as mentioned above. The second one is the training error with loss function in terms of control outputs compared with the demonstrations during the training process which is called “training error of the learned policy with loss”. The third one is the training error without loss function in terms of control outputs compared with the demonstrations as self-testing. For simplicity, it is called “training error” after section 3.4. The second and third one are calculated as: mismatched output instances/total demonstration instances. The mismatched error is only counted once for multiple output commands with the same cost.

3.2 Experimental Settings of the One-Step Greedy Approach

The simulation world is a grid world of 3 by 8. Example environments are listed in Figs. 2 (a) and (b). It is assumed that the approximate goal direction is always the front of the agent thus the control commands are only consisted of *up*, *left*, *stop*, and *right*. The moving speed of the controlled agent is limited to 1. A human subject is told to control the agent to navigate through the environment with the same control policy from any initial position to (2, 8). The navigable places in the environment may be occupied with other moving agents during the navigation. Those moving agents would move ahead at various speeds while avoiding the collision with another agent. After the data collection and training stages, different static and dynamic environments are used to test the learned policy.



(a) Dynamic world. (b) Static world with user's policy. (c) Learning result Settings 1. (d) Learning result Settings 2. (e) Learning result Settings 3.

Fig. 2. The snapshots of the simulation example world and learning results in the static environment using different settings. The white block indicates clean space. The black block means static obstacle. The cyan block indicates the controlled agent. The initial position of the user controlled agent may change in each training data. Other moving agents may have different colors while keeping the same color during the simulation for better visualization. The next-step motion of those agents will be displayed as triangle to indicate the moving *up* or rectangle when the agent decides to *stop* in the same position as in (a). The arrows, *S* symbols, and numbers are added in (b)-(e) for better understanding of the user's policy and the learning results with the corresponding position in the simulation world. The *S* symbol in (c) stands for *stop* command. Settings 1: perception feature with original LEARCH algorithm. Settings 2: perception feature and control feature with original LEARCH algorithm. Settings 3: perception feature and control feature with the modified LEARCH algorithm. The *stop* commands would cause the agent to get stuck in the static environment using perception feature only. The red (dark) arrows in (d) and (e) indicate the differences between the two settings.

3.3 Learning Results of the One-Step Greedy Approach

Three different settings are used to show their individual performances. In the first two settings, the original LEARCH algorithm is used but with different feature functions.

Only the perception feature is used in the first setting whereas the full feature vector which consists both of the perception and control features is used in second scenario. The third one is the proposed method. The modified LEARCH algorithm and full feature vector are used to increase the performance. The trained policies in the static environment are shown in Figs. 2 (c)-(e). It could be shown that the *stop* commands in Fig. 2 (c) would cause the agent to get stuck in the static environment using the perception feature only. On the other hand, in Figs. 2 (d) and (e), the agent could be led to navigate through the environment with any one of the two policies. Compare with the training policy from Fig. 2 (b), note that in (1, 1), (2, 3), and (3, 6), proper commands are learned based on the training examples. To further compare the performance between setting 2 and 3, we also notice that some commands differ from those in Fig. 2 (b). In Fig. 2 (d), the output command in (3, 5) differs from demonstration while in Fig. 2 (a), the output command in (2, 1) does not match the demonstration. However, with the proposed modification, the training error of the learned policy with loss is indeed reduced as shown in Fig. 3. Without the loss, the number of mismatch examples is 1 out of 142 using the proposed data refinement procedure whereas for setting 2, using the original LEARCH algorithm, the number is 10 out of 142.

Fig. 3. The training error of the learned policy with loss. The red curve is the result of original LEARCH algorithm while the black curve is the proposed method.

After the comparison in static environments is discussed, the results from dynamic environments are presented. In Fig. 4, the two moving agents in the mid line keep spaces for the particular agent to pass the gate way while moving ahead without collision. The particular cyan agent also recognizes the fact thus decides to utilize the free spaces and moves to the mid line to complete the navigation task. In Fig. 5, the agents in the mid line do not reserve free spaces for others to pass the gate way. In other words, the interaction potential is low for those agents. Unless using different parameters for the interaction potential, the interaction model could not be captured well. Whereas using the proposed model-free approach, the cyan agent could recognize the situation. The cyan agent decides to cut in the mid line and follows the agent ahead to complete the navigation. The intuition is that the interaction relations between the controlled agent and other moving agents are preserved from the collected data. The policy to interact with different

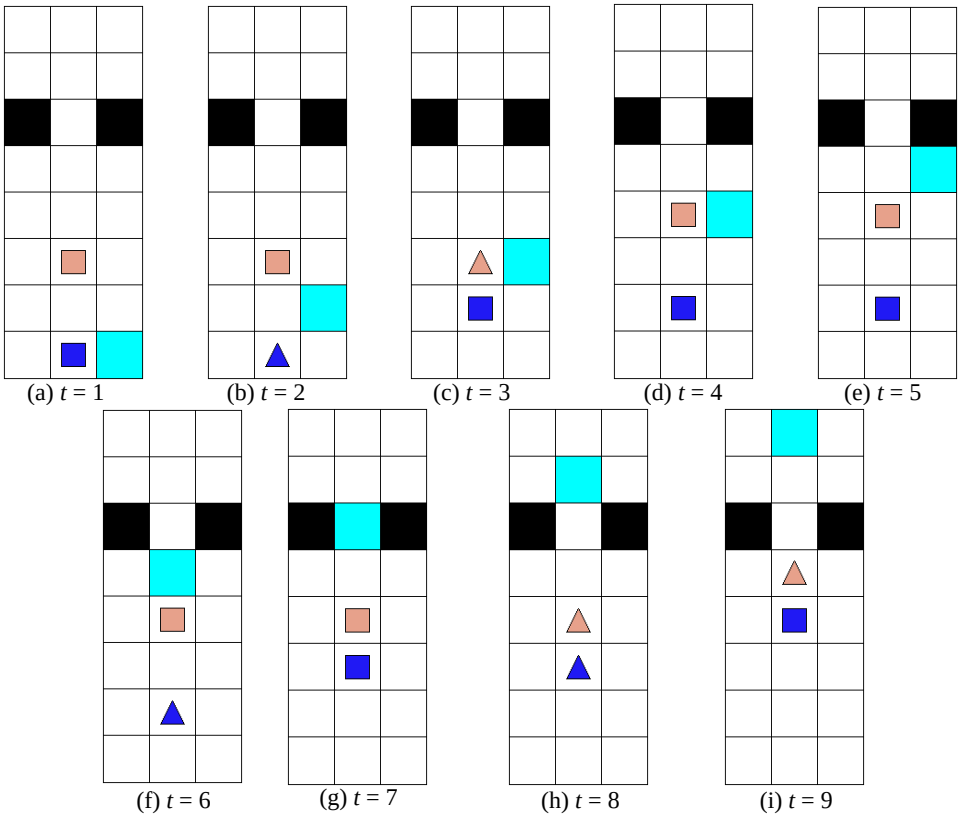


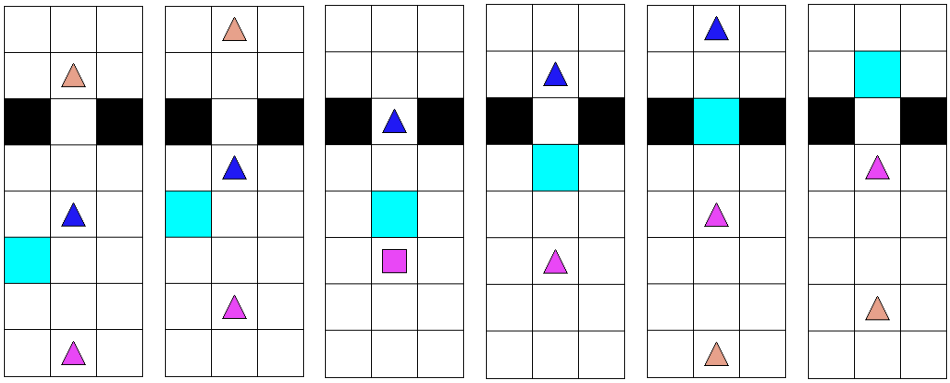
Fig. 4. One of the testing results. The behavior of joint collision avoidance is well captured. Utilizing the free spaces reserved by the two agents in the mid line, the cyan agent bypasses the two agents to navigate through the gate way.

moving agents is consistent and could be extracted by the learning algorithm to reproduce the same behavior.

Using the proposed approach, different control policies could be learned to personalize the control policy if needed. In the simulation results, another data set is collected to verify this statement in which a lining behavior is learned. The agent would be led to the mid line as soon as possible and follow the movers ahead to pass the gate way instead of trying to bypass the others as previous example shown in Fig. 5.

3.4 Multi-Step Approach

To further analyze the performance of the proposed algorithm, the experiments from a more complex simulation world are carefully studied. The simulation world is now an m by n map larger than the original 3 by 8 world. The user could only observe the surrounding 5 by 5 window in which the user controlled agent will always appear in the center of the window as in Fig. 6. All the moving agents have maximum moving speed 1 and could move four-neighbor direction or stay in the same position. The infor-



(a) $t = 1$ (b) $t = 2$ (c) $t = 3$ (d) $t = 4$ (e) $t = 5$ (f) $t = 6$

Fig. 5. Another testing result. When $t = 1$, the output command is *up*. This means the cyan agent tries to bypass other agents to navigate through the gate way. However, in this case, the agents in the mid line do not reserve free spaces for others to pass the gate way. The cyan agent recognizes the fact and decides to cut in the mid line at $t = 2$. After that, the cyan agent has to follow the agent ahead to complete the navigation.

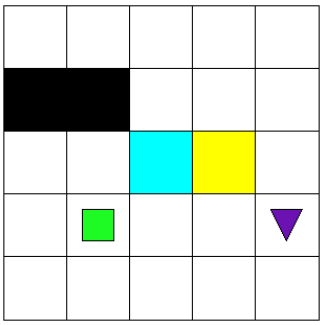


Fig. 6. The snapshot of the complex simulation window. The user controlled agent will always appear in the center of the window to simulate the robot-centered world. The new added light yellow block right next to the user controlled agent is the waypoint which may be placed outside the local window during the navigation.

mation of the next-step motion is displayed as triangle to indicate the moving direction or rectangle when the agent stays in the same position.

In order to evaluate the performance, the motion of the moving agent is pre-defined thus every control sequence could be reproduced for analyzing the results of different policies. During the experiments, it is soon discovered that the information of the goal would be essential for the learner to learn the policy instead of using perception feature and control feature only as in the simple simulation world. This could be shown in Fig. 7. Not only the perception feature only setting gets high error rate as expected, the training error is also high for using both the perception feature and control feature. It is found out that our previous argument: the information of goal / waypoint is not needed is only suitable for the simple environment and setting in section 3.2. While all the agents including the user controlled agent are heading up, the information of the goal would be similar between demonstrations and could be little or no help for training. However, the

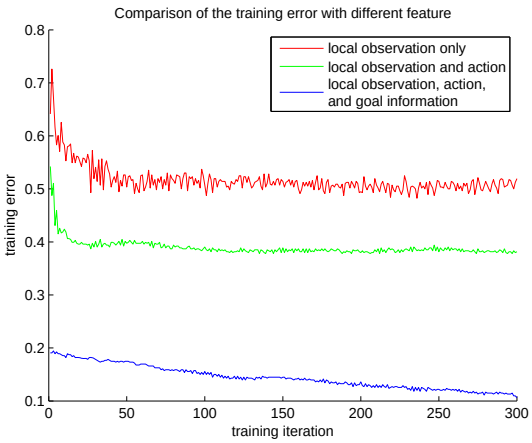


Fig. 7. Comparison of the training error with different feature functions. The highest red curve shows the training error using local perception feature only. The middle green curve is the result of using both perception feature and control feature. The lowest blue curve is the result of using additional goal-related information with perception feature and control feature.

user's control policy could be affected by the information of the goal / waypoint in the complex simulation world. It could be shown using Fig. 6 as an example. The control output should be *right* since the waypoint is on the right-hand side of the controlled agent. However, when the waypoint is on the left-hand side of the controlled agent, the *left* command would be appreciated as a proper control output. Without the goal-related information, the two cases would not be differentiated by the learner and the resulting output would always be the same no matter where the waypoint is.

Besides the goal-related information, it is also found that the learned control commands during successive steps may change frequently whereas the control commands are more stable in original demonstrations. For example, for the successive *up-up-right-right* commands, the learned control outputs for each individual example with step number $N = 1$ may be *up*, *right*, *right*, and *right*. However, when testing, the successive output commands may be executed as *up-right-up-right* because the second control output results in a world without user's demonstration. In this case, although the training error could be low, the behavior of successive steps may still not be similar with the demonstration. Thus, the multi-step learning algorithm with goal-related information is proposed to solve these issues and would be described below. Briefly speaking, the aim of the training process is to imitate the behavior of the demonstrator with multiple steps instead of the next greedy step.

3.5 Implementation in a Complex Simulation World

The whole procedure of the algorithm is presented in Algorithm 1 with step number N equal or larger than 1. In the implementation, the range of the step number N is from 1 to 5. The loss function is the normalized accumulated Manhattan distance for the N-step path compared with the demonstration which could be defined as

$$loss = \frac{\sum_{n=1}^{N} ManhattanDistance\left(Pos_n, DemoPos_n\right)}{2 * \sum_{n=1}^{N} n}. \quad (1)$$

The normalization term is calculated as the possible maximum accumulated Manhattan distance for the *N*-step path which could be pre-computed in advance for each step number. For better understanding, the testing result Fig. 8 is used as an example here. Assuming that the 3-step worlds are Figs. 8 (a)-(c) and the demonstration worlds are Figs. 8 (a), (d) and (e). The loss value would be (0 + 2 + 0) / (2 * (1 + 2 + 3)) = 1/6.

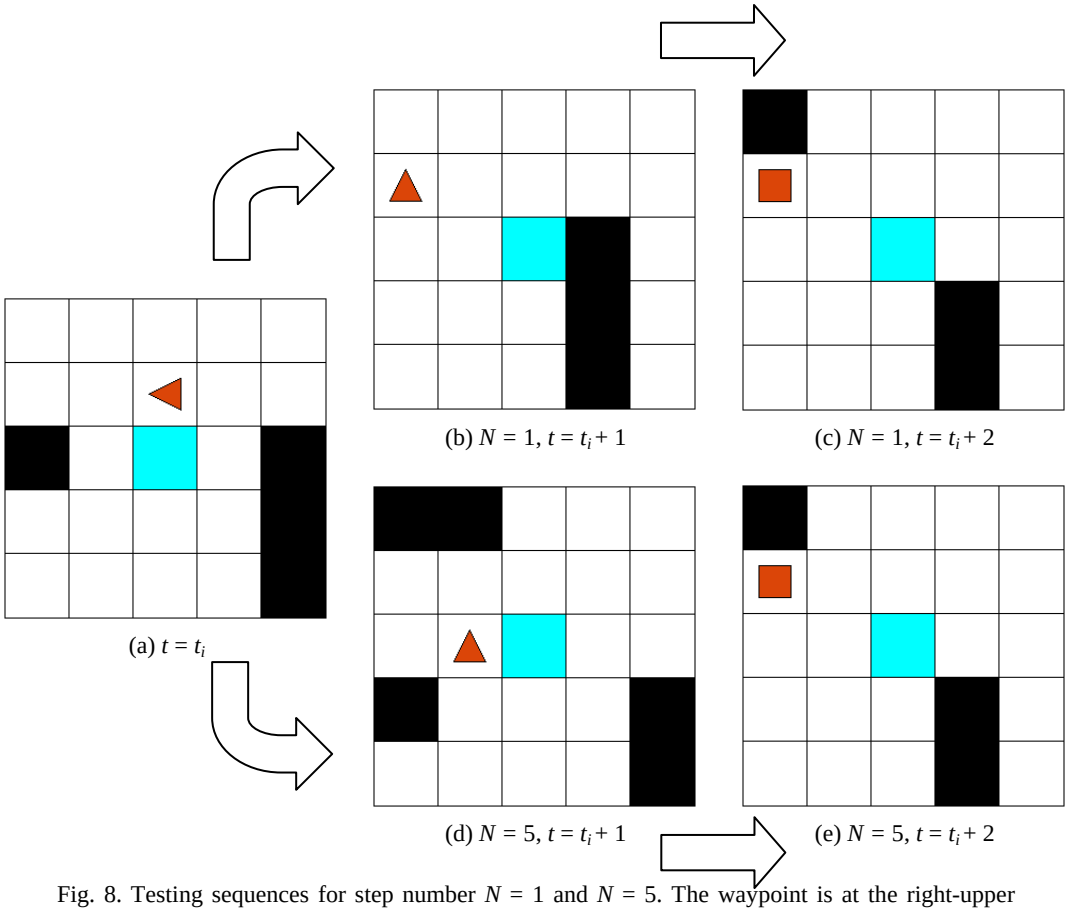


Fig. 8. Testing sequences for step number $N = 1$ and $N = 5$. The waypoint is at the right-upper place. When t at some $t_i - 1$, the output command is *up* and the resulting world is (a). When t_i, using the result of step number $N = 1$ would get the *right* command and the world becomes (b) with waypoint at the right-upper place. The next output command would be *up* and the final world is (c). However, based on the smooth motion style, the appropriate successive output command should be *up* when t_i and the resulting world would be (d) with waypoint at the right-hand side. The next output command would be *right* and the final world is (e). The agent changes moving direction twice using step number $N = 1$ but only changes once when using step number $N = 5$.

The feature function now mainly consists of local perception feature, control feature, and goal-related feature. For describing the 5 by 5 window, the type of the object and the moving direction of the object for each grid are used. The control feature is a five dimensional binary feature to indicate the commands of *up*, *left*, *stop*, *right*, or *down*. The goal-related feature consists of leaving/approaching the waypoint or not and if the agent reaches one waypoint.

The algorithm then starts with the initialization step in line 1. In line 5, for each valid control command, the feature vector of the next N steps is calculated. For each sequence of user-provided demonstrations with length L larger than N, there would be $L - N + 1$ negative examples for line 7. This could be done because the world of the next N steps is recorded during the demonstrations. Thus, the feature could then be extracted accordingly with the corresponding world. Then, for each other valid control sequence, it is assumed that the world of the next N steps could be predicted based on the chosen actions of the controlled agent and other moving agents. In the end, for each time step t, the next-N-step perception feature, control feature, and goal-related feature are computed. The feature vectors from $t + 1$ to $t + N$ are then concatenated to form a longer feature vector. Using the current learned policy applied with the extracted features, the minimum cost one will then be selected as the positive example for line 7. Still, the AdaBoost [13] algorithm with decision stump is used to do the classification and get the log-hypothesis h_t for the log-costmap updating procedure. The training process is terminated if the learned examples are close enough with the demonstration.

4. EXPERIMENTAL RESULTS

The experimental settings in complex dynamic environments and the learning results are presented in this section. The training errors with different step numbers are shown. The testing sequences are also presented to show the performance of the proposed multi-step framework.

4.1 Experimental Settings

There are waypoints in the simulation world and the human subject could be guided to perform the navigation in the dynamic environment. The whole map is larger than the local observation window and is shown in Fig. 9. These waypoints could be generated using any planning algorithm or pre-defined by hand in advance using the static map without any moving agent. During the demonstration, when the waypoint is not in the local observation window, the direction for approaching the next waypoint would be told to the user. Such information is similar with the goal-related feature mentioned in section 3.5. A human subject is asked to complete the navigation with the same policy and not rapidly change the moving direction as a controlling style. The data is then collected for the training process.

4.2 Learning Results and Discussion

The training error with different step number N is shown in Fig. 10. The error is counted when the predicted next one step is not the same as demonstration even for step

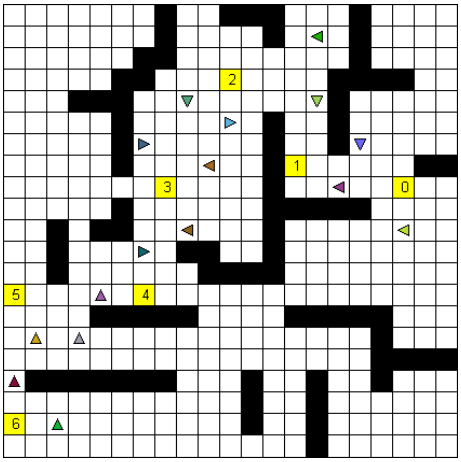


Fig. 9. The simulation world with waypoints and other agents. The yellow blocks with number are the waypoints. The other colored triangles are moving agents in this environment.

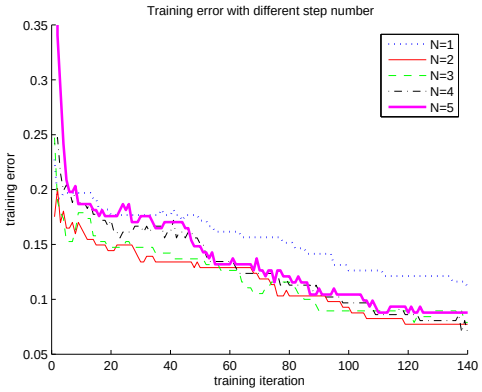


Fig. 10. The training error with different step number N. All the final training errors are similar while the training error with step number $N = 1$ is a little bit higher.

number N larger than 1. The reason is as follows. First, since it is only considered the control output of the next step when moving the agent instead of executing N steps at a time, it is suitable to compare the error in one-step version. Second, the training error would be large if the error is counted when the successive steps are not completely the same. It would be hard to judge if the large error for successive five-step setting is worse than small error for the one-step setting. In fact, the training error for successive five-step setting is about 0.4560 whereas the testing performance is better than using smaller step number.

For each testing scenario, either with different waypoints or moving agents, the training results with different step number are tested in order to compare the performance. It could be observed that the user's controlling style: keeping the smooth motion could be captured when using a larger step number N whereas the successive control behavior may not be well captured using step number $N = 1$. This could be shown in Fig. 8. The

smoother successive *up-up-right* commands are executed using step number $N = 5$ while the less smooth *up-right-up* commands are executed using step number $N = 1$.

Moreover, when using smaller step number, especially one-step only, the navigation may not be completed due to the local trap conditions. The controlled agent would perform *stop* command for other non-moving agents. However, there are still some limitations when using large step numbers. When encountering the trap conditions occurs in small step number, using large step number may result in directly hitting and then passing through other agents in order to complete the navigation. This could be shown in Fig. 11. The navigation could be completed while the safety may not be guaranteed. During the experiments, although the collision occurs, it happens less when using larger step number. Using smaller step number not only increases the chance of encountering local trap conditions but also the collision chance during the navigation. The main drawback of using larger step number should be the fact that it is more unrealistic and time-consuming to predict the future world precisely. The step number could only be chosen as a reasonable small number instead of a huge number in practical use.

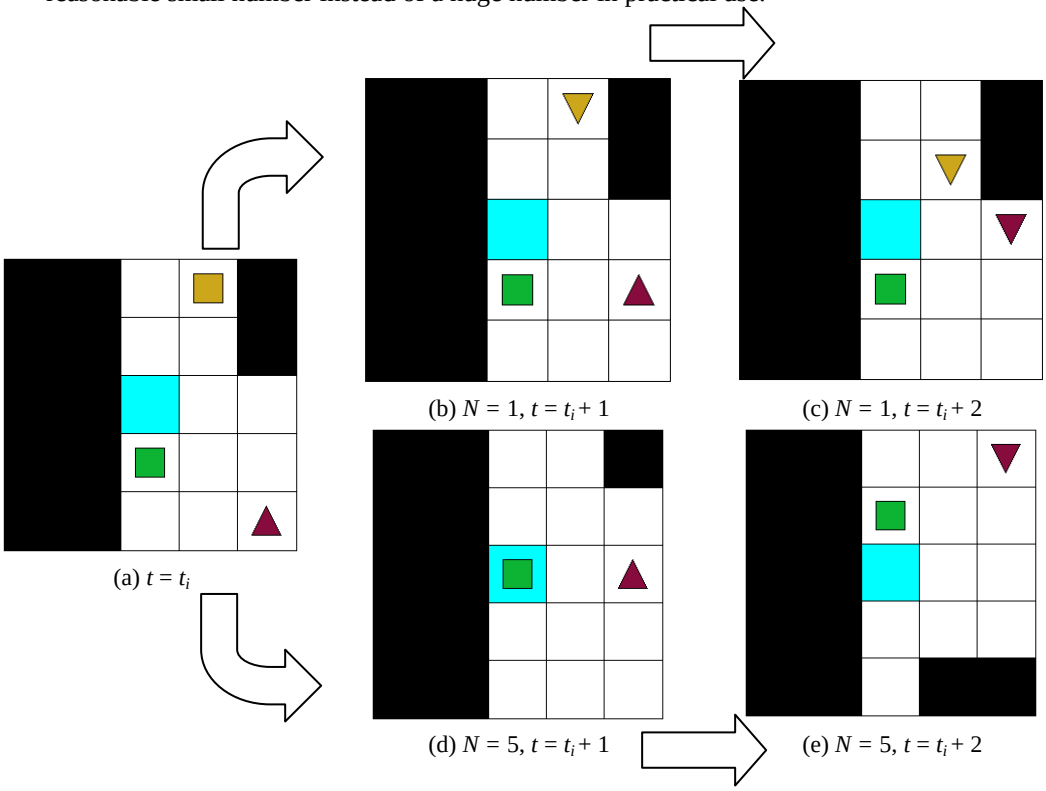


Fig. 11. Another testing sequences for step number $N = 1$ and $N = 5$. The waypoint is at the down direction. When t_i, using step number $N = 1$, the output command is *stop* and the resulting world is (b). The agent could not bypass the non-moving one and continue performs the *stop* again when $t_i + 1$ and results in the world (c). In the end, the agent gets stuck and could not complete the navigation. Using the result of step number $N = 5$ would get the *down* command and the world becomes (d). The controlled agent hits an agent however continuously navigates through the environment in the following steps (e).

From the experimental results, it could be seen that the performance would be better using a larger step number. The successive control behavior could be well captured than using a small step number. However, the step number could only be chosen as a small number instead of a huge one for getting reasonable prediction of the future world. In our implementation, the maximum number for N is 5 which is considered as a reasonable value compared with the size of the observation window and the maximum speed of the moving agents. Beside further improvement for collision-free navigation, how to determine the best step number N online instead of choosing a particular one could also be the future extensions of this work.

5. CONCLUSION AND FUTURE WORK

In this paper, the performance of our previous model-free learning approach with the data set refinement procedure is quickly reviewed and further analysis is studied in more complex simulation worlds. It is observed that the goal-related information is important for the learner and should be included as part of features. To learn the successive steps, the multi-step approach is developed and the behavior of the demonstrator is well captured by the learner to reproduce the similar control policy in the simulation results.

The collision event in the simulation should be avoided to further increase the performance. Besides using the rule-designed method to eliminate the output commands which result in collision when both training and testing, using the current LEARCH approach, it may be feasible to choose the minimum cost state which does not result in collision. However, it is not guaranteed that the second-best action could best explain the demonstration when the first-best action fails. Thus, it is our ongoing work to use the interactive learning mechanism to avoid the collision. The human subject could be involved in the learning process when needed and new demonstration could be provided for interactively increasing the learning performance. The interactive machine learning approach with human's demonstration would be a natural extension of LfD.

REFERENCES

1. B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and Autonomous Systems*, Vol. 57, 2009, pp. 469-483.
2. N. D. Ratliff, J. A. Bagnell, and S. S. Srinivasa, "Imitation learning for locomotion and manipulation," in *Proceedings of the 7th IEEE-RAS International Conference on Humanoid Robots*, 2007, pp. 392-397.
3. D. Silver, J. A. Bagnell, and A. Stentz, "Learning from demonstration for autonomous navigation in complex unstructured terrain," *The International Journal of Robotics Research*, Vol. 29, 2010, pp. 1565-1592.
4. P. Henry, C. Vollmer, B. Ferris, and D. Fox, "Learning to navigate through crowded environments," in *Proceedings of IEEE International Conference on Robotics and Automation*, 2010, pp. 981-986.
5. P. Trautman and A. Krause, "Unfreezing the robot: Navigation in dense, interacting crowds," in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010, pp. 797-803.

6. C.-C. Yu and C.-C. Wang, "Learning collision-free navigation from demonstration without global information," in *Proceedings of International Conference on Service and Interactive Robotics*, 2011, pp. 232-237.
7. N. D. Ratliff, D. Silver, and J. A. Bagnell, "Learning to search: Functional gradient techniques for imitation learning," *Autonomous Robots*, Vol. 27, 2009, pp. 25-53, 10.1007/s10514-009-9121-3.
8. C.-C. Yu and C.-C. Wang, "Collision- and freezing-free navigation in dynamic environments using learning to search," in *Proceedings of Conference on Technologies and Applications of Artificial Intelligence*, 2012, pp. 151-156.
9. J. Minguez and L. Montano, "Sensor-based robot motion generation in unknown, dynamic and troublesome scenarios," *Robotics and Autonomous Systems*, Vol. 52, 2005, pp. 290-311.
10. J. Minguez and L. Montano, "Nearness diagram navigation (nd): a new real time collision avoidance approach," in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2000, pp. 2094-2100.
11. J. Minguez and L. Montano, "Nearness diagram (nd) navigation: collision avoidance in troublesome scenarios," *IEEE Transactions on Robotics and Automation*, Vol. 20, 2004, pp. 45-59.
12. D. Althoff, J. Kuffner, D. Wollherr, and M. Buss, "Safety assessment of robot trajectories for navigation in uncertain and dynamic environments," *Autonomous Robots*, Vol. 32, 2012, pp. 285-302.
13. Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," in *Proceedings of the 2nd European Conference on Computational Learning Theory*, 1995, pp. 23-37.

Chung-Che Yu (余宗哲) received his B.S. degree in Computer Science and Information Engineering from National Taiwan University, Taipei, Taiwan, in 2007. He is currently a Ph.D. student at the Robot Perception and Learning Laboratory at National Taiwan University. His research interests include robot navigation, machine learning, and robot learning from demonstration.

Chieh-Chih (Bob) Wang (王傑智) earned his Ph.D. in Robotics from the School of Computer Science at Carnegie Mellon University in 2004. He received his B.S. and M.S. from National Taiwan University in 1994 and 1996, respectively. During his graduate study, he worked with the Bayesian vision group at the NASA Ames research center and at Z+F Inc. in Pittsburgh. From 2004 to 2005, he was an Australian Research Council (ARC) Research Fellow of the ARC Centre of Excellence for Autonomous Systems and the Australian Centre for Field Robotics at the University of Sydney. In 2005, he joined the Department of Computer Science and Information Engineering at National Taiwan University, where he is an Associate Professor and is pursuing his academic interests in robotics, machine perception and machine learning. Dr. Wang received the best conference paper award at the 2003 IEEE International Conference on Robotics and Automation (ICRA).