Robot Task Error Recovery Using Petri Nets Learned from Demonstration

Guoting (Jane) Chang and Dana Kulić

Abstract— **The ability to recover from errors is necessary for robots to cope with unexpected situations in a dynamic environment. Efficient error recovery should allow the robot to utilise existing knowledge of the task and learn new error recovery strategies from observation. This paper proposes an automatic error recovery procedure that allows the robot to handle both known and unknown error states using a Petri net representation of the task. For known error states, the robot can directly adjust the sequencing of actions using the Petri net representation to complete the task, while for unknown error states, the robot can learn how to perform error recovery from a human demonstrator by extending the existing Petri net. The proposed method is verified on a real robot performing a block stacking task.**

I. INTRODUCTION

In order for robots to function in a human environment and robustly perform different services, they need the ability to cope with unexpected situations by detecting and recovering from errors while performing tasks. It can be difficult to predict every possible error during the design stage and computationally costly to check for every pre-programmed error while performing a task [1]. Therefore, it would be preferable for the system to be able to use its existing knowledge of the task to recover from the error when possible, and learn new error correction strategies by extending the knowledge representation of the task when direct recovery is not possible.

This paper presents a method for using a learned Petri net representation [2], [3] of a task to achieve the task goal whenever an unexpected, yet known situation is encountered and to learn new error correction strategies from a human demonstrator when an unexpected and unknown situation is encountered. The new error correction strategy is learned by automatically extending the Petri net from the observed human demonstration. One of the main advantages of this approach is that the robot can learn to correct the most pertinent errors to the task, some of which may only become clear when the robot is actually performing the task. Given the robot's ability to dynamically adjust to unexpected situations, this approach can also be used for collaborative task execution.

II. RELATED WORK

The process of error recovery can be divided into error detection, error diagnosis and error correction [4]. Error detection and correction have received significant attention in the automation literature, particularly for automated manufacturing systems (AMS) [5], [4]. Common approaches include graphical models such as Petri nets [1], [6] and task planning and knowledge base methods [7], [8].

Error correction using Petri nets usually consists of adding an error recovery subnet to the normal Petri net. Examples include Petri nets extended with fuzzy logic [9] or timing [6] to perform error recovery for manufacturing processes via an error recovery subnet whenever a difference between the expected state and actual state is detected. However, in these works, both the nominal and error correction subnets are specified a priori and no learning occurs. The approach in [1] is similar to ours as it considers both cases where unexpected, but known states (no error or previously seen errors) and unknown states are encountered. However, while this work proves that adding recovery subnets preserves the overall properties of the Petri net, it does not specify exactly how the new error recovery subnets would be constructed and added to the Petri net.

The work of Gini and Gini [7] is an example of using a knowledge base for error recovery. This approach is similar to ours in that it considers the preconditions and postconditions of each action for detecting errors. However, their approach requires a human expert to encode the error recovery rules into the knowledge base and no learning occurs from the errors encountered.

There is also related work addressing error recovery in robot task performance [10], [11]. For example, [10] uses Petri nets to model action plans for soccer playing robots that include the possibility of failed actions. Their approach of structuring the Petri net is different from ours as both states and actions are encoded as part of the place component of the Petri net. Furthermore, there is no learning or dynamic extension of the Petri net. Yamazaki et al. [11] on the other hand use manipulation behaviours consisting of behaviour units for an assistive cleaning robot. Error recovery capabilities are designed into the behaviour units, but no learning occurs.

Error recovery has also been addressed through human-robot interaction and guided learning. For example, Mericli et al. [12] propose an approach for corrective human feedback during robot action selection in a humanoid soccer scenario. However, in their approach, only action selection is learned by the robot, while the overall task structure is hard coded. Lockerd and Breazeal [13] develop an approach for robot task learning through interactive demonstrations. The robot can query the human demonstrator for additional

This work was supported by the Ontario Graduate Scholarship Program and the Natural Science and Engineering Research Council of Canada.

G. J. Chang and D. Kulić are with the Department of Electrical and Computer Engineering, University of Waterloo, Ontario, Canada {g5chang, dana.kulic}@uwaterloo.ca

guidance when it cannot autonomously complete the task. Their approach differs from ours as their task is modeled symbolically via simple words representing the task goals. Similarly, Nikolescu and Mataric [14], [15] develop a sequential task learning framework using directed acyclic graphs where the robot can learn from observation and query the human demonstrator to correct the task model if errors are encountered. Their approach also relies on preconditions and postconditions of actions, but it differs from ours as their approach requires the error correction portion to be interleaved with the task execution in order for it to be incorporated into the correct location in the sequence. In our approach, even though our examples show the error correction during task execution, the error correction portion can also be shown as a separate demonstration and still be correctly added to the Petri net due to the additional encoding of object states in the Petri net.

III. PETRI NET OVERVIEW

Petri nets [16] are a process modeling methodology that can model sequential, parallel, synchronous, asynchronous, deterministic and/or stochastic processes [2], [3]. A Petri net consists of two main components: places (P) and transitions (T). Places represent states or conditions of the system being modeled, while transitions represent actions or events in the system. Petri nets can be represented either algebraically or as a bipartite graph with circles for places and rectangles for transitions. The arcs connecting places and transitions are indicated by the input places (I) and output places (O) for each transition, effectively defining the states or preconditions necessary for an action or event to take place and the states or postconditions resulting from a particular action or event.

To describe the dynamics of a Petri net, places in the Petri net are marked with $tokens$ to indicate the current state(s) or condition(s) of the system. The tokens indicate which transitions are $enabled$, i.e. which action(s) or event(s), if any, may take place. A transition is considered enabled if all of its input places have at least one token in them. The $firing$ of a transition transfers tokens from the input places of the transition to the output places.

The algebraic notation of a marked Petri net consists of a 5-tuple [2], [3]:

$$C = \{P, T, I, O, \mu_0\} \tag{1}$$

where

P is the finite set of all places p_i
T is the finite set of all transitions t_i where $P \cap T = \emptyset$
$I(t_i)$ is the set of all input places of transition t_i
$O(t_i)$ is the set of all output places of transition t_i
μ_0 is the initial marking (initial location of tokens)

A marking μ consists of a vector with an element for each place p_i to indicate the number of tokens in that place.

The reachability problem in Petri nets pertains to whether a particular marking of the Petri net can be attained given an initial marking [2], [3]. Reachability in Petri nets can be determined via the construction of a reachability tree [2], where the root node of the reachability tree is the initial marking. Each branch of the reachability tree is labeled by a corresponding, enabled transition which will lead to another node representing the resulting marking following the firing of the transition. The leaf nodes of the tree consist of the markings after which there are no more enabled transitions (terminal nodes) and markings which are the same as the marking of a previous node in the tree (repeated node).

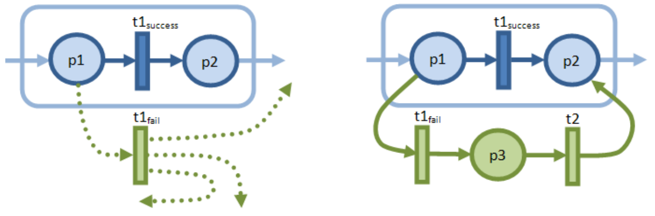
IV. LEARNING PETRI NETS

In our previous work [17], we developed a learning system that automatically creates Petri nets from observations of human demonstration of tasks. The learned Petri nets can then be used both to recognize known object states and motions when they are repeated by the human, and to generate sequences of motions that generalize the task and allow the robot to imitate it.

To model a particular task, the places in the Petri net are used to represent states of the objects related to the task, while the transitions in the Petri net are used to represent motions, i.e. actions taken by the human demonstrator. Automatic object and object state recognition is performed using self-organizing maps (SOMs) [18], while automatic motion recognition and imitation is achieved using dynamic motion primitive (DMP) motion models [19] and affinity propagation clustering [20]. Each place in the Petri net describes the relevant object states in terms of their absolute (referenced to the robot) and relative positions (referenced to other objects). The sequence of object states and motions observed in the task dictates the sequence of places and transitions of the Petri net, i.e. places representing object states observed before and after a particular motion will be added as input and output places (respectively) to the transition representing that motion in the Petri net.

This Petri net creation method can summarize multiple demonstrations of the same task into the same Petri net, because the algorithm can recognize when observed object states and motions correspond to existing places and transitions in the Petri net and do not need to be added. In the case when there are variations in how the same task is performed during the demonstration, the created Petri net will either already reflect the variation or additional places and transitions are added to include new ways of performing the task [17]. When there are several ways of performing a task, this will be automatically encoded in the Petri net with parallel branches, in this case the robot can chose one of the alternatives for performing the task, this would be modeled by one of the branch transitions firing.

Using the created Petri net, reachability trees can be constructed for a given initial marking and given a desired final marking, a path between these two markings can be found through the tree. This path will indicate the sequence of transitions, and thus, the actions required to imitate the task. Using the reachability tree, it is also possible to find paths for performing any sub-component of the task even though only a demonstration of the entire task was observed for creating the Petri net [17].



(a) Implicit transition fail (b) Learned error correction

Fig. 1. Petri net transition failing example

V. PROPOSED APPROACH

In this paper we extend the capabilities of the learned Petri nets to allow the robot to automatically detect and recover from error conditions. Errors in execution can occur due to sensing or actuation uncertainty, or due to unanticipated changes in the environment, e.g., due to human intervention. Error conditions are detected during task imitation by checking the actual state of the objects after performing each action corresponding to a transition in the Petri net and also before the very first action is taken (to ensure the initial state is correct). If the states of the objects in the environment differ from the expected states based on the output place(s) reached by the transitions in the path or the place(s) in the initial marking, an error condition is detected.

In order for the Petri net to handle error conditions, all transitions in the Petri net are implicitly assumed to be able to fail during task imitation. The success or failure of a transition in the Petri net is solely determined by the resulting output place of the transition as shown in Figure 1a:

1) A *successful* transition is one through which the desired output place is reached.
2) A *failed* transition is one through which any place other than the desired output place is reached. Two types of output places can be reached via a failed transition:
    a) A *known* place. In this case, it is possible to directly recover from the error by regenerating the reachability tree using the current marking for the Petri net and finding the paths (if any) that will lead to the final desired state.
    b) An *unknown* place. In this case, it is not possible to directly recover from the error. Instead, the robot should try to learn from a human demonstrator how to recover from the error encountered, so that it can handle the error in the future.

In order to learn how to recover from errors when the robot reaches an unknown place after a failed transition, the robot prompts the human demonstrator asking if it can learn to correct the error. The human demonstrator will make the decision of whether the robot can learn to correct the error or if the robot should ignore the error and continue with the task. For this work, errors that the robot can correct must fulfill the following conditions:

1) The object states must still be visible to the robot, i.e. the robot can still sense all the objects.

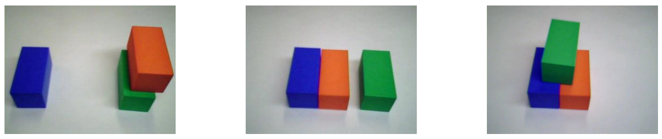
Algorithm 1 Task Execution Including Error Recovery

```
1:  generate reachability tree
2:  find path to final desired state
3:  *error detection:* detect states
4:  **if** environment states are as expected **then**
5:      **if** reached final desired state **then** exit
6:      **else**
7:          take next action in path and return to line 3
8:      **end if**
9:  **else** *error correction:*
10:     **if** environment states are known **then**
11:         regenerate reachability tree using Petri net
12:         find path to final desired state, return to line 3
13:     **else** environment states are unknown
14:         prompt user for help
15:         **if** can learn to correct error **then**
16:             capture video of human demonstration
17:             extract object, object states and motions
18:             extend Petri net, return to line 3
19:         **end if**
20:     **end if**
21: **end if**
```

2) The motion needed to correct the error must be doable by the robot, i.e. it must not involve impossible configurations or unreachable positions for the robot.

If the robot can correct the error, a video of the human demonstrator performing the motion for error correction is recorded. The demonstrated error correction should lead from an unknown object state to a known object state, i.e. existing place in the Petri net. From the recorded video, the robot obtains the unexpected and previously unknown error states, the motion performed by the human demonstrator and the resulting, known states. Using this information and knowing which transition led to the error states, the Petri net is extended by adding an arc from the known place(s) that precede the failed transition to the failed transition and from the failed transition to the new place(s) representing the error state. The new state place(s) then have an arc leading to a new transition representing the error correction motion which then connects back to a known place in the Petri net. An example of this is shown in Figure 1b, where p_3 is the new state place and t_2 is the new transition learned. This new sequence of states can represent an error correction strategy, or an alternative way for the robot to perform the task. Algorithm 1 outlines the main steps of task execution including the error recovery process.

Once the error recovery procedure has been incorporated into the Petri net, the robot can directly recover from the error the next time that it is encountered, as the error states are now part of the known states. Thus, if the error states are unexpectedly encountered again, the reachability tree will be regenerated to find a path starting from the place representing that error state and leading to the final desired state of the task. This reachability tree regeneration and path finding can



(a) Start (b) Intermediate (c) Finish

Fig. 2. 3 block stacking

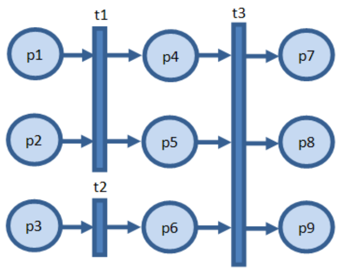


Fig. 3. Petri net

TABLE I

PLACE MEANINGS (L = LEFT, R = RIGHT, C = CENTER)

P	**Object Location**					**Other Object Above or Below**		
	L	**R**	**CL**	**CR**	**C**	**No**	**Above**	**Below**
p1		✓						✓
p2		✓					✓	
p3	✓					✓		
p4				✓		✓		
p5		✓				✓		
p6			✓			✓		
p7				✓			✓	
p8					✓			✓
p9			✓				✓	

TABLE II

TRANSITION MEANINGS (L = LEFT, R = RIGHT, C = CENTER)

T	**Block**	**Motion**
t_1	Red	From R towards CR
t_2	Blue	From L towards CL to the left of the red block
t_3	Green	From R towards C on top of the blue and red blocks

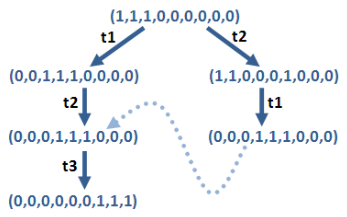


Fig. 4. Petri net reachability tree (dotted line connects repeated nodes)

be repeated multiple times during task imitation, if multiple errors are encountered.

The ability of the robot to adjust for both known and unknown output places allows for a collaborative approach between human and robot to complete the task. For example, if the human helps to move objects towards their final state, the robot will be able to adjust its next action based on the resulting states in the environment. The proposed approach also allows the robot to learn multiple ways of performing the task, as these are encoded in the alternative paths explored through error correction.

VI. EXPERIMENTS

To demonstrate the proposed approach, the NAO humanoid robot was used to perform a 3 block stacking task for which it has previously learned a Petri net representation based on human demonstration [17]. The human demonstration was recorded using NAO's camera. Color-based features were used to extract the block locations in every frame to detect the object states and object motion trajectories. In this task, a blue, red and green block are stacked as shown in Figure 2. The blue block starts on the left side (L) of the workspace as seen by the robot, while the red block starts on top of the green block on the right side (R). The red block is moved first towards the center right (CR) of the workspace, then the blue block is moved towards the center left (CL), i.e. next to the red block. Finally, the green block is stacked at the center (C) on top of the red and blue blocks. This task contains both a parallel (concurrent) and a sequential component, i.e. the order of actions for placing the first two blocks can be interchanged, but the action of placing the third block must come last. The resulting learned Petri net for this task is shown in Figure 3.

The meaning of each place in the Petri net, i.e. the object states that they represent, is summarized in Table I and the meaning of each transition is summarized in Table II. The learned Petri net correctly reflects that the moving of the red and blue blocks are parallel actions and their order of occurrence can be interchanged without affecting the end result.

Given initial and final markings $\mu_{initial} = (1,1,1,0,0,0,0,0,0)$ and $\mu_{final} = (0,0,0,0,0,0,1,1,1)$, the reachability tree constructed from the Petri net is shown in Figure 4 and the two paths found through the reachability tree are $t_1t_2t_3$ and $t_2t_1t_3$ which match the two alternative sequences of performing the task. The robot selects the first shortest path found for execution, i.e. $t_1t_2t_3$, but if additional information about the desirability of one sequence versus another were available, it would be possible to take the additional information into consideration.

The two types of error states handled by the proposed error recovery algorithm are *known error states* and *unknown error states*. The experimental results are obtained by letting the NAO robot imitate the task and adapt its behaviour according to the states reached so that appropriate error recovery is performed.

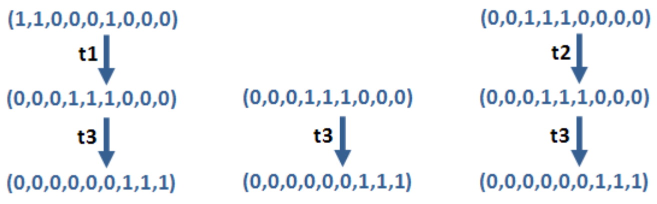
A. Known Error States

Given the Petri net in Figure 3, five possible valid markings of the Petri net exist as shown in Table III. Valid markings involve all three objects in physically possible states for which the algorithm can find a path to the final desired marking.

Thus, given any valid initial marking, there will be one correct output marking and four unexpected markings including the initial marking itself. These four unexpected markings

TABLE III
VALID MARKINGS (L = LEFT, R = RIGHT, C = CENTER)

μ	**Value**	**Block Positions**
μ_1	$(1,1,1,0,0,0,0,0,0)$	Initial state of stacking task
μ_2	$(0,0,1,1,1,0,0,0,0)$	Red at CR, green at R, blue at L
μ_3	$(1,1,0,0,0,1,0,0,0)$	Red at R, green at R, blue at CL
μ_4	$(0,0,0,1,1,1,0,0,0)$	Red at CR, green at R, blue at CL
μ_5	$(0,0,0,0,0,0,1,1,1)$	Final state of stacking task



(a) μ_3 case (b) μ_4 case (c) μ_2 case

Fig. 5. New reachability trees

correspond to the possible, known error states.

The algorithm should be able to recover from any of the known error states by finding a path through the reachability tree from the marking representing the error state to the final desired marking. The reachability tree is regenerated each time, as it is not assumed that the previously generated reachability tree contains all the valid markings already.

The output marking of t_1 (moving the red block) is used in this experiment to demonstrate recovery from known error states. The initial marking is $\mu_1 = (1,1,1,0,0,0,0,0,0)$. The expected, correct output marking following the execution of t_1 should be $\mu_2 = (0,0,1,1,1,0,0,0,0)$. However, a human demonstrator will change the outcome to the known error marking μ_1, μ_3, μ_4 or μ_5, i.e. the human demonstrator will move the blocks while the robot is performing the motion for t_1 so that the resulting state is unexpected but known. For each of these unexpected, known error states, the robot correctly determines that it has to use the reachability tree to find a new path leading to the desired final goal:

1) μ_1: For this unexpected output marking, the reachability tree remains the tree shown in Figure 4. The path that the algorithm finds is $t_1t_2t_3$, which is the correct sequence of actions to reach the final goal. This result represents the error of the red block not having moved even though the robot tried to move it via t_1. It shows that the robot is able to recognize that the action t_1 needs to be repeated and recover from the error.

2) μ_3: For this unexpected output marking, a new reachability tree as shown in Figure 5a is constructed from the Petri net and the path found is t_1t_3. This situation illustrates the case where the blue block is unexpectedly moved to the center left, while the red block is not moved even though it was expected to move to the center right. The robot switches to following the alternate sequence where t_3 follows t_1 to successfully complete the task.

3) μ_4: For this unexpected output marking, a new reachability tree as shown in Figure 5b is constructed from the Petri net and the path found is t_3. This case illustrates the situation where the blue block is unexpectedly moved towards the center when the robot is moving the red block, the robot recognizes this and chooses to only perform the last action t_3 to successfully finish stacking all three blocks.

4) μ_5: For this unexpected output marking, the task stacking is completed and the robot realizes that no reachability tree needs to be constructed and no more actions are necessary.

The cases of μ_4 and μ_5 as unexpected output markings simulate cases where a human decides to help the robot progress in the task and the robot correctly recognizes that some actions don't need to be performed anymore.

In order to verify that the algorithm is able to recover if μ_2 is the unexpected output marking and also to demonstrate that the algorithm is able to handle not only unexpected output markings, but also unexpected initial markings, the robot is presented with the red block already moved at the beginning of the task resulting in the initial marking of μ_2, even though the robot is expecting μ_1. In this case, the robot also successfully recognizes the error and regenerates the new reachability tree as shown in Figure 5c from the Petri net and finds the resulting path t_2t_3. Thus, μ_2 is also one of the recoverable known error states.

Finally, even though all the above results were given for t_1, the same recoverability capabilities apply to the output markings of t_2 and t_3. Furthermore, the robot can handle multiple errors during task imitation as the proposed algorithm simply checks whether the states are as expected after each transition (and also for the initial set of states) and regenerates the reachability tree and path as needed. This has been verified experimentally by changing the object states to a known, but unexpected state after t_2 and again after t_3. The robot is able to correctly finish the task each time.

B. Unknown Error States

For the 3 block stacking task, when placing the green block on top of the red and blue blocks, an “unknown” error state may be encountered. This is due to the joint limitations of the NAO robot which result in the green block being placed on the red and blue blocks at an angle. This misalignment of the green block causes occlusion of the blue block which results in different states being detected by the algorithm for the green and blue blocks.

This situation is an example of an unknown error state as the robot does not know by itself how to correct the misalignment of the green block. When the green block is improperly aligned, the robot prompts for help, asking if this is an error that it can learn to correct. The human demonstrator lets the robot know that it can correct this problem and the robot records a video of the human demonstrator showing how to tap the green block into alignment. Once the green block has been aligned properly, the robot is able to recognize the states and knows that the task has been completed.

Object state and motion information is then extracted from the recorded video and added to the Petri net resulting in the modified Petri net shown in Figure 6. The failed transition

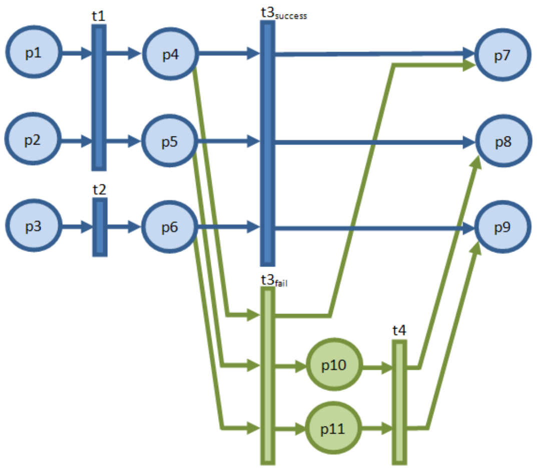


Fig. 6. Petri net with learned error correction

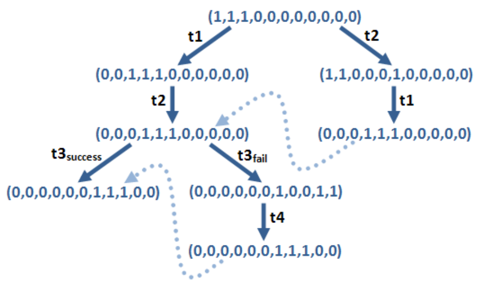


Fig. 7. Petri net reachability tree with learned error correction

along with the resulting new place representing the error state and new transition representing the error correction motion leading back to a known state have been included in the Petri net. The resulting Petri net reachability tree is shown in Figure 7. The branch containing the error correction is longer than the direct path to the final state, thus the robot will not pick that path from the paths found unless the error is encountered.

If the robot encounters the error state of the misaligned green block again during task imitation, the error state will be a known error state, which will be resolved by regenerating the reachability tree and searching for a path starting from the marking representing the error state.

Even though the Petri nets in this work are simple and most Petri net analysis approaches make use of the reachability tree, it has been shown that the reachability problem can potentially be computationally expensive [2], [3], [21]. This did not pose a problem for the Petri nets in this work, but for more complex tasks hierarchical Petri nets can be used, such that each Petri net being solved will still remain simple despite the complexity of the overall system.

VII. CONCLUSIONS AND FUTURE WORK

This paper proposes an automatic error recovery extension to an existing robot task learning system using Petri nets. The error recovery extension allows for direct error recovery from unexpected yet known states via regeneration of the Petri net reachability tree and finding a new path. Further, new error recovery strategies can be learned via observation of human demonstration for unexpected and unknown states. This error recovery approach results in minimal effort for the human demonstrator to teach the robot and only pertinent error recovery strategies will be learned by the robot. The proposed approach also holds potential for collaborative human robot task completion as the robot dynamically ascertains the states of the environment after each action.

In future work, hierarchical Petri nets and a continuous, probabilistic description of state may be considered for more complex tasks.

REFERENCES

[1] M.-C. Zhou and F. DiCesare, “Adaptive design of petri net controllers for error recovery in automated manufacturing systems,” *TSMC*, vol. 19, no. 5, pp. 963–973, 1989.
[2] T. Murata, “Petri nets: Properties, analysis and applications,” *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541 – 580, April 1989.
[3] J. L. Peterson, “Petri nets,” *ACM Computing Surveys*, vol. 9, no. 3, pp. 223–252, September 1977.
[4] P. Loborg, “Error recovery in automation - an overview,” 1994.
[5] M. d. Jeng, “Petri nets for modeling automated manufacturing systems with error recovery,” *Robotics and Automation*, vol. 13, no. 5, pp. 752–760, October 1997.
[6] N. G. Odrey and G. Mejia, “An augmented petri net approach for error recovery in manufacturing systems control,” *Robotics and Computer-Integrated Manufacturing*, vol. 21, no. 4-5, pp. 346–354, 2005.
[7] M. Gini and G. Gini, “Towards automatic error recovery in robot programs,” in *IJCAI*, 1983, pp. 821–823.
[8] T. Matsuoka, T. Hasegawa, and K. Honda, “A dexterous manipulation system with error detection and recovery by a multi-fingered robotic hand,” in *IROS*, 1999, pp. 418–423.
[9] T. Cao and A. C. Sanderson, “Modeling of sensor-based robotic task plans using fuzzy petri nets,” in *Computer Integrated Manufacturing and Automation Technology, 1994*, October 1994, pp. 73–80.
[10] H. Costelha and P. Lima, “Modelling, analysis and execution of robotic tasks using petri nets,” in *IROS*, 2007, pp. 1449–1454.
[11] K. Yamazaki, R. Ueda, S. Nozawa, Y. Mori, T. Maki, N. Hatao, K. Okada, and M. Inaba, “System integration of a daily assistive robot and its application to tidying and cleaning rooms,” in *IROS*, 2010, pp. 1365–1371.
[12] C. Mericli, M. Veloso, and H. L. Akin, “Task refinement for autonomous robots using complementary corrective human feedback,” *Int J Advanced Robotic Systems*, vol. 8, no. 2, pp. 68–79, 2011.
[13] A. Lockerd and C. Breazeal, “Tutelage and socially guided robot learning,” in *IROS*, 2004.
[14] M. N. Nicolescu and M. J. Matarić, “Learning and interacting in human-robot domains,” *TSMCA*, vol. 31, no. 5, pp. 419–430, 2001.
[15] ——, “Natural methods for robot task learning: Instructive demonstrations, generalization and practice,” in *AAMAS*, 2003, pp. 241–248.
[16] C. A. Petri, “Kommunikation mit automaten,” Ph.D. dissertation, Darmstadt University of Technology, Germany, 1962.
[17] G. J. Chang and D. Kulić, “Robot task learning from demonstration using petri nets,” in *RO-MAN*, 2013, pp. 31 – 36.
[18] T. Kohonen, “The self-organizing map,” *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464–1480, September 1990.
[19] S. Schaal, “Dynamic movement primitives a framework for motor control in humans and humanoid robotics,” *Adaptive Motion of Animals and Machines*, p. 261280, 2006.
[20] G. J. Chang and D. Kulić, “Motion learning from observation using affinity propagation clustering,” in *RO-MAN*, 2013, pp. 662 – 667.
[21] J. L. Peterson, *Petri Net Theory and The Modeling of Systems*, ser. Foundations of Philosophy Series. Prentice-Hall, 1981.