Characterizing Glovebox Automation Tasks using Partially Observable Markov Decision Processes

Adam Allevato and Mitch Pryor

Nuclear and Applied Robotics Group, The University of Texas at Austin, 10100 Burnet Rd, Austin, Texas 78758, USA
allevato@utexas.edu, mpryor@utexas.edu

INTRODUCTION

As robots become more prevalent in everyday life, companies are implementing new hardware and software systems that provide improved human-robot interaction, but at the expense of reduced reliability and performance. These service robots tolerate of some level of uncertainty, as long as they can complete their tasks satisfactorily. But in the industrial-nuclear complex, such uncertainty can literally be fatal. Customers such as the Department of Energy desire 100% accuracy and repeatability of our nuclear robotic systems to ensure safe task execution.

Before deploying a robot into a nuclear environment, the robot should be validated and verified to ensure that it will perform according to its requirements with as high a probability as possible. To quantify the performance of a robotic system (which is inherently probabilistic and uncertain because of sensor inaccuracies), we look to the field of cyberphysical system verification (hereafter referred to as system verification). By using techniques from pure mathematics and formal logic, system verification can evaluate a system, and, under certain assumptions, guarantee the correct execution of a system according to some specification.

In this work we wish to prove the execution correctness of an unspecified radiochemistry manipulation task that will take place in a glovebox. By analyzing a purposely generalized task, we can show that system verification is applicable across a wide domain, then use the technique to verify solutions to specific problems. Our system setup includes an environment that has an inherent state, sensors which produce an observation from the environment, and a robotic system that can take specific actions, which may lead to a variety of different environment states (due to inherent uncertainty). Together, these three concepts (state, sensors, and actions) are well represented by a Partially Observable Markov Decision Process (POMDP), especially for scientific applications, including autonomous robots in hazardous environments [1]. We use POMDP system verification tools to generate a provably correct strategy for our robotic system, allowing it to select the correct series of actions (based only on sensor inputs and internal memory) to reach the goal state, completing the task.

RELATED WORK

The idea of remotely handling radioactive material is not new. In the 1940s, locations such as Idaho National Lab developed cable-driven remote manipulators, allowing a manipulator in a hot cell to "puppet" an operator using a complex mechanical system [2]. In 1984, cable-driven master-slave configurations were still the current state of the art, being deployed at Oak Ridge National Laboratory (ORNL) [3].

In more recent years, the Advanced Recovery and Integration Extraction System (ARIES) system has been deployed at Los Alamos National Laboratory (LANL) to automate the process of dismantling nuclear weapons [4]. ARIES employs several automation tools and robots, including two Fanuc LR Mate 100i manipulators.

The ARIES system has been successful in carrying out extremely well-defined tasks and procedures, and its creators even write "robots will soon be appearing in a glovebox near you" [5], but open-ended or one-off tasks require more versatile systems that are still robust. Labs such as LANL have been slow to adopt the idea of robotics in gloveboxes, partially because of the lack of guarantees on task execution.

UT Austin's Nuclear and Applied Robotics Group (NRG) is helping expand glovebox automation capabilities by showing proof-of-concept of various tasks inside gloveboxes. The lab has demonstrated force control [6], safety architectures [7], and vision-based detection and classification [8], all in support of robotic glovebox procedures. This research looks at the higher-level task planning required for such procedures.

Kaelbling et al. [9] point out that POMDPs are only well-suited to problems with finite state spaces, making them a poor tool for visual recognition and pose detection. However, assuming that 6DOF pose detection and grasping can be completed via traditional means, POMDPs can produce high-level plans for task completion in a safe, effective manner, even in the presence of uncertainty.

APPROACH

We first provide an overview of POMDPs and system strategies. Then, we describe the general radiochemistry task and details of the system, actions, uncertainties, and environment states. Finally, we present the software-based approach used to characterize the system and generate a strategy.

Partially Observable Markov Decision Processes

A POMDP is defined formally by a tuple of six values, $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{O}, O)$ [10].

- $\mathcal{S}$ is the finite set of states the environment may be in, and represents the actual state, not the value estimated by the system.
- $\mathcal{A}$ is the finite set of actions that the system may take.
- $\mathcal{T}$ is the state transition model, which characterizes how the system moves from state to state as various actions are taken by the system, encoding a probability distribution over the set $\mathcal{S} \times \mathcal{A}$. For each action $a \in \mathcal{A}$, the probability of transitioning from state s to s' is encoded as $\mathcal{T}(s, a, s')$.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{R}$ is the reward function, which provides real-valued rewards to the system for taking actions while

the environment is in a specific state. A successful execution strategy will seek to maximize the system's reward, either in the short-term or in the long-term.

- O is the finite set of observations. The system has a way of sensing its environment, and O enumerates the possible sensor responses. The number of observations is usually less than the number of environmental states, which means that multiple different states may produce the same observations. Since the system must rely only on its sensors' observations, and the observations can be ambiguous, the system has incomplete knowledge of the environment state during execution. This limited state awareness is what produces the "partial observability" of a POMDP.

- O is the observation function, which provides a unique probability distribution over O for each value of $\mathcal{S} \times \mathcal{A}$. In other words, O represents the multiple different observations that may be observed for each state/action pair.

Some authors define a POMDP using an additional reward function, R, and seek to develop systems that will maximize the total reward over time [11], [12]. This definition is not useful for our discussion, because it implies that the system can succeed to varying degrees. We consider our system to have only two possible outcomes: success and failure, and so we do not use the reward function and restrict ourselves to a standard POMDP definition.

System Modeling

The system to be modeled is that of a robotic manipulator inside an enclosed environment that includes a stationary visual sensor. The manipulator can pick up objects from an entrance point and place them at various locations inside the environment, but other than this, the environment remains stationary. The visual sensor is located in a position where it can view the entrance point to detect and classify new environment objects, but cannot detect objects after the manipulator has moved them. By keeping the system purposely high-level, the analysis procedure can apply to specific systems easily. As an example, this representation can model an industrial manipulator deployed in a glovebox transfer port. NRG has developed a system of this type (Fig. 1), consisting of a Yaskawa Motoman SIA5 7-DOF serial manipulator, Robotiq 3-finger gripper, and Asus Xtion Pro RGB/depth camera, all mounted to a lead-lined glovebox with 4 square meters of work surface.

The robot (with attached gripper) is deployed through a transfer port using NRG's custom mounting stand ([13]) for convenience, but it could also be deployed through one of the smaller glove ports. The Xtion Pro camera mounts to the outside of the glovebox behind a layer of leaded glass, providing a view of approximately 60% [1] of the glovebox work surface (including the second transfer port) while also protecting the image sensor. As mentioned above, NRG has

[1]Adding more cameras to the system could provide a full view of the glovebox workspace, but this was unnecessary for our task.

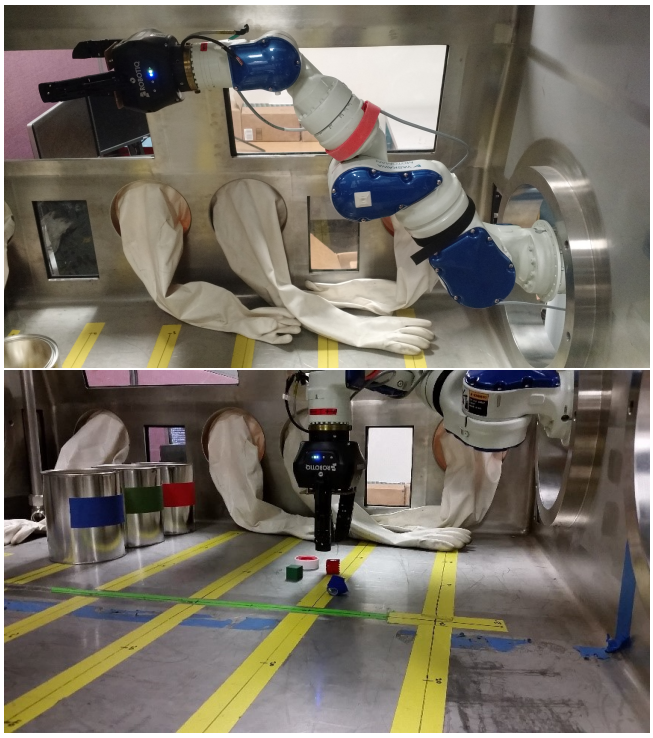

Fig. 1. Above: The transfer port-deployed SIA5 at NRG. This picture was taken from the approximate position of the depth camera mounted on the glovebox. Below: the robot performs a pick and place task.

already performed a number of vision-based manipulation tasks using this and similar systems (see Figure 1), so this already validated system is an ideal platform for the concepts in this paper.

Once the system has been modeled, the next step is to describe the task. The glovebox work area begins empty, and items will enter the glovebox through the transfer port. The operator will add items to the glovebox one at a time, from a set of three objects: Container Type A, Container Type B, and Sample. The vision system will automatically detect and classify the new object, then take the appropriate action, with the end goal being to place a sample, which is assumed to be radioactive, inside a container without ever placing the sample directly on the work surface. The vision system is unable to discern between Container Type A and Container Type B, and the sample can be placed into either type of container. After each object is added to the glovebox, the system can a) request a different object be put in the transfer port, b) pick/place a container from the transfer port, or c) pick/place a sample from the transfer port. Attempting to pick the wrong type of object from the transfer port will result in a system error.

This task is purposely general to prove the concept of system verification for many procedures, but the task includes important parts of a POMDP, as will be seen.

After the task has been defined, the system, environment, and actions map to a POMDP as follows. The set of states is the combination of all possible glovebox states (empty, Container A in place, Container B in place, Con-

tainer A & B in place, sample placed in container) with all possible entry states (Container A in transfer port, Container B in transfer port, sample in transfer port). An additional state, `err`, represents an unrecoverable system fault, or failure. The system's actions are a direct map to the three actions listed above: `request_new_item`, `move_container`, or `move_sample`. The observations are `container_in_transfer`, `sample_in_transfer`, and error (we assume that the system is able to detect when it reaches an error state). The observation mapping function assumes a perfect sensor, with the exception of the inability to discern between different types of container. All states that have a container in the transfer port produce the `container_in_transfer` observation, and all states that have a sample in the transfer port produce `sample_in_transfer`.

The set of goal states for the environment is the set of all states where a sample has been moved into the glovebox after a container has already been moved into the glovebox. The system halts execution after this state is reached because the task is complete.

The "solution" to a POMDP is a strategy $S : \mathcal{M}, \mathcal{S} \rightarrow \mathcal{M}, \mathcal{A}$, which works similarly to a finite state machine. The strategy will select an action $a \in \mathcal{A}$ based on the current state $s \in \mathcal{S}$ and some memory value $m \in \mathcal{M}$. The memory value can be thought of as an internal state associated with the agent implementing the strategy. Using m, a strategy can make informed decisions about what action to take based on its past experience. Take the example of a robot in a maze. After choosing a path from an intersection, the robot updates its internal memory m to "remember" that it has taken a path. The next time the robot encounters the same intersection, it can refer its memory to avoid revisiting the same path. In our implementation, m is updated after every action, so the system is able to keep track of its path through the state space in search of the goal state. The memory value m need not be an explicit listing of previous actions, it can simply be an integer value that the strategy understands, and it can be defined differently for different strategies. Because of this, $\mathcal{M}$ is only useful in the context of a specific strategy, and is not part of the POMDP definition.

Performing system verification on our POMDP will find a *winning strategy*, if one exists. A winning strategy will almost surely satisfy the requirements. "Almost surely" refers to the formal probabilistic concept of probability 1, or that there are a finite number of cases where the strategy will fail to reach the goal. However, since this finite number of failure cases are part of an infinite solution space, and all the other (infinite) solutions result in success, the strategy will always succeed when implemented in a real-world system.

POMDPs with Parity Objectives Solver (PPS) [14] was used to generate a winning strategy from our POMDP. PPS is a Java program that reads POMDPs using a special script syntax. The program takes as inputs states, observations, and a transition model which have been explicitly defined in a file, then finds a finite-memory winning strategy (or informs the user that such a strategy does not exist).

To use PPS, the strategy's goal must be defined in terms of a parity automaton, which is a state transition system where each state is assigned a nonnegative number. PPS's winning strategy ensures that the lowest-numbered state that repeats infinitely often has an even number. This parity objective is easier to solve than an arbitrary formal logic formula or other ways of expressing a goal. To convert our goal states into a parity objective, we take the manual approach as described in [10]: we simply assign the set of goal states to have parity 0 (even), and all other states to have parity 1 (odd). The winning strategy will revisit even-numbered (i.e. goal) states infinitely many times, ensuring that the system completes the manipulation task as desired. Once the system reaches an error state, the only transitions that can occur (no matter what actions are taken) lead to the same error state, so a path that visits an error state will stay there forever, and will never reach a goal state. The winning strategy must revisit an even-numbered (goal) state infinitely many times, so it will be designed to never get "stuck" in an error state.

TABLE I. The winning strategy for the glovebox manipulation task. Based on the current memory state (**m src**) and observation (**obs**) in each step, the system will choose a line, take the corresponding **action**, and change the internal memory state to the associated **m dest**.

line	m src	obs	action	m dest
1	1	`o_a`	`request_item_change`	2
2	1	`o_b`	`request_item_change`	3
3	1	`o_s`	`request_item_change`	4
...				
55	12	`o_a`	`request_item_change`	13
56	12	`o_f`	`place_sample`	14
57	13	`o_b`	`request_item_change`	11
58	13	`o_s`	`request_item_change`	12
59	13	`o_a`	`request_item_change`	13
...				
63	14	`o_f`	`place_sample`	14

RESULTS

PPS generated the results quickly for this 17-state, 51-transition system, producing a winning strategy in approximately 0.1 seconds [2]. The software returns a strategy in the form of a mapping between memory states, observations, and actions. Each entry in the strategy tells the system which action to take for each possible combination of system memory and observation. Because the various memory values define different internal states, the system can take different actions based on previous parts of the execution. A truncated version of the strategy is shown below in Table I.

The system begins with internal memory $\mathcal{M} = 1$. In this state, the system can observe one of three actions: `o_a`, `o_b`, or `o_s`. These three possibilities map to lines 1, 2, and 3 of Table I. Based on the observation from the sensors, the system will take an action–in this case, it will always per-

[2]The machine used to run PPS was a Windows 10 machine with a 4th-generation Intel i7 and 16GB of RAM

form `request_item_change`– and then update the internal memory state to 2, 3, or 4.

Solution Technique

On line 59, the system will repeatedly ask the operator for a different item (action `request_item_change` until something besides a Container Type A is observed (`o_a`). Presumably, this represents a memory state where accepting a Type A container would cause a system fault. The memory-observation-action mapping is powerful enough to encode the logic required, including loops, to complete the task effectively without ever reaching an error state.

In line 56 of the strategy, the system observes that it has completed its task (the observation is `o_f`, so it goes to state 14 (line 63), which loops infinitely. The run is complete. In this strategy file, every run will eventually lead to `o_f`, and will never get stuck in a fault state.

In general, the winning strategy is intuitive. It continues to ask the operator for a new item until it detects a container, then moves the container into the glovebox. If containers continue to be inserted, it will handle them appropriately (rejecting or moving) until a sample is inserted, at which time it will put the sample into one of the previously-loaded containers.

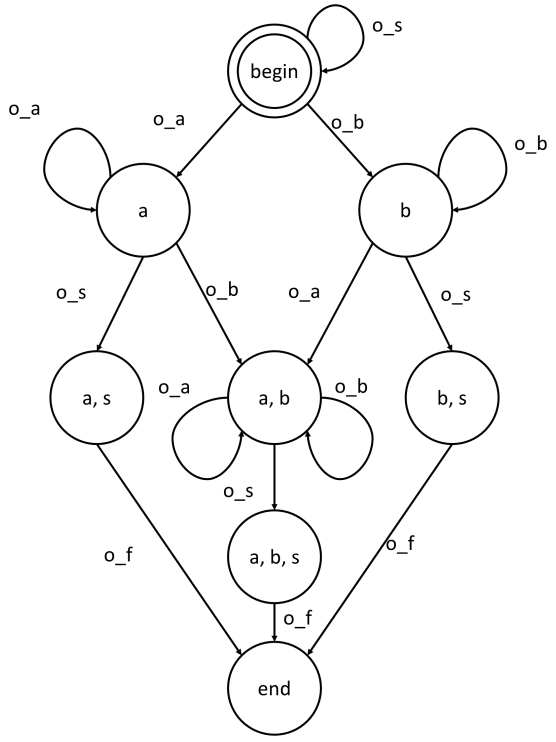

Fig. 2. A simplified representation of the winning strategy. Actions and memory states are not shown in this graph. Each node represents an environment state; for example, "a" means that a Type A Container has been placed into the environment. Arrows represent transitions caused by each object that could be placed in the transfer port. The goal is to place the sample ("s") after placing one or both containers, which results in an observation of `o_f`, or "task finished."

This straightforward strategy becomes quite complex using the memory-observation-action syntax, as shown by the amount of reasoning required when reading the strategy line by line. PPS generates a graph of states, observations and actions, but this graph has 29 internal memory states, and so is too complex to be helpful. A simplified version is shown in Figure 2.

As complex as this strategy becomes using the memory-action form, it could most likely have been coded by hand using basic loops and logic, because of the simplicity of the problem. However, even for this simple system, a hard-coded solution would have no guarantee of proper execution, and it could get "stuck" or exhibit unexpected behavior. More complex tasks will only exacerbate the problem, emphasizing the need for automatic strategy generation.

It is important to note that the POMDP-generated model provides a viable alternative to either finite state machines or goal-oriented action programming (GOAP) [15], both of which have been used by NRG in the past. The method in this paper has the advantage of being machine-generated and provably correct, so it does not have any unexpected states that could trip up the system because of inadequate planning. However, this blessing is also a curse, in that it is very difficult for a human to reason out the system's behavior by looking at the strategy. In addition, system verification is only valid if the original assumptions on system behavior hold. If, for example, an action's possible outcomes change because of unexpected changes to the environment, the system may not perform perfectly. Although the system and environment may arrive at the goal state, there are no guarantees.

CONCLUSIONS AND FUTURE WORK

Applying cyberphysical system verification to a robot manipulation problem presents unique challenges, but also unique results. POMDP solvers allow the development of a provably-correct almost-sure winning strategy to satisfy a parity objective. By converting a definition of a basic glovebox manipulation task to a POMDP/parity objective pair, software can provide strong guarantees of the completion of the task. The strategy is robust to incomplete sensor information and partially unknown state transition models, providing the correct system action to take for each possible sensor observation. By generating strategies automatically for manipulation tasks, we achieve greater robustness, safety, and confidence in our manipulation system's performance.

The next step for this research is to implement the strategy on a physical hardware system, such as the port-deployed manipulator at NRG.

Simple automated tools would speed up development of future POMDPs and associated strategies. As mentioned, the syntax used to define POMDPs requires explicit definition of all states, observations, and transition probabilities. For large POMDPs, the definition file grows exponentially in size. A script to automatically generate definition files for large processes would greatly ease the development of new strategies. Once a script has been developed, additional proof of concept for POMDP-based verification would include generating strategies for a number of common glovebox tasks, such as sampling, counting, sorting, or packing. Finally, the generated

strategies could be integrated into hardware platforms at NRG or elsewhere.

ACKNOWLEDGMENTS

The authors would like to thank Krishnendu Chatterjee, Martin Chmelik, Raghav Gupta, and Ayush Kanodia for the development of the PPS tool, and Ufuk Topcu for his guidance regarding cyberphysical system verification.

This material is based upon work supported by a Department of Energy Nuclear Energy University Programs Graduate Fellowship.

REFERENCES

1. A. R. CASSANDRA, "A survey of POMDP applications," in "Working Notes of AAAI 1998 Fall Symposium on Planning with Partially Observable Markov Decision Processes," (1998), vol. 1724.
2. "CRL Central Research Laboratories | DESTA-CO," `http://web.archive.org/web/20151208150328/http://www.destaco.com/crl-products-equipment.html`, accessed: 2016-01-05.
3. J. N. HERNDON ET AL., "The state-of-the-art model M-2 maintenance system," in "presented at the American Nuclear Society Topical Meeting on Robotics and Remote Handling in Hostile Environments," (April 1984).
4. C. TURNER and J. LLOYD, "Automating ARIES," *LANL Actinide Research Quarterly*, pp. 32–35 (April 2008).
5. P. C. PITTMAN ET AL., "Automation of the LANL ARIES lathe glovebox," in "The American Nuclear Society Ninth Topical Meeting on Robotics and Remote Systems," (2001).
6. B. O'NEIL ET AL., "Hazardous workspace modeling for manipulators using spatial hazard functions," in "Safety, Security, and Rescue Robotics (SSRR)," (November 2012).
7. A. ALLEVATO ET AL., "Using a depth camera for object classification in nuclear gloveboxes," in "American Nuclear Society Student Conference," (April 2015).
8. B. O'NEIL, *Object Recognition and Pose Estimation for Manipulation in Nuclear Materials Handling Applications*, Ph.D. thesis, The University of Texas at Austin (2013).
9. L. KAELBLING ET AL., "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, pp. 99–134 (1998).
10. K. CHATTERJEE ET AL., *PPS: User Manual* (2015), accessed 2016-01-06.
11. D. BRAZIUNAS and C. BOUTILIER, *Stochastic local search for POMDP controllers*, National Library of Canada= Bibliothèque nationale du Canada (2004).
12. A. Y. NG and M. JORDAN, "PEGASUS: A Policy Search Method for Large MDPs and POMDPs," in "Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence," Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2000), UAI'00, pp. 406–415.
13. J. A. HASHEM, "Integrating fixed and flexible solutions for glovebox automation," Tech. Rep. LA-UR-11-04457 (Jul 2011).
14. K. CHATTERJEE ET AL., "What is Decidable about Partially Observable Markov Decision Processes with omega-Regular Objectives," in S. R. D. ROCCA, editor, "Computer Science Logic 2013 (CSL 2013)," Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2013), *Leibniz International Proceedings in Informatics (LIPIcs)*, vol. 23, pp. 165–180.
15. J. ORKIN, "Symbolic representation of game world state: Toward real-time planning in games," in "Proceedings of the AAAI Workshop on Challenges in Game Artificial Intelligence," (2004), vol. 5.