POMDP-based Candy Server: Lessons Learned from a Seven Day Demo

Marcus Hoerger [*], **Joshua Song** [†], **Hanna Kurniawati** [‡], **and Alberto Elfes** [§]
[*†] School of Information Technology & Electrical Engineering, University of Queensland, Australia
[‡] Research School of Computer Science, Australian National University
[§] Robotics and Autonomous Systems Group, Data61, CSIRO
{m.hoerger, j.song}@uq.edu.au,
hanna.kurniawati@anu.edu.au, Alberto.Elfes@data61.csiro.au

Abstract

An autonomous robot must decide a good strategy to achieve its long term goal, despite various types of uncertainty. The Partially Observable Markov Decision Processes (POMDPs) is a principled framework to address such a decision making problem. Despite the computational intractability of solving POMDPs, the past decade has seen substantial advancement in POMDP solvers. This paper presents our experience in enabling on-line POMDP solving to become the sole motion planner for a robot manipulation demo at IEEE SIMPAR and ICRA 2018. The demo scenario is a candy-serving robot: A 6-DOFs robot arm must pick-up a cup placed on a table by a user, use the cup to scoop candies from a box, and put the cup of candies back on the table. The average perception error is $\sim$3cm ($\approx$ the radius of the cup), affecting the position of the cup and the surface level of the candies. This paper presents a strategy to alleviate the curse of history issue plaguing this scenario, the perception system and its integration with the planner, and lessons learned in enabling an on-line POMDP solver to become the sole motion planner of this entire task. The POMDP-based system were tested through a 7 days live demo at the two conferences. In this demo, 150 runs were attempted and 98% of them were successful. We also conducted further experiments to test the capability of our POMDP-based system when the environment is relatively cluttered by obstacles and when the user moves the cup while the robot tries to pick it up. In both cases, our POMDP-based system reaches a success rate of 90% and above.

Introduction

Decision making in the presence of uncertainty is a critical component in robot autonomy. The Partially Observable Markov Decision Processes (POMDPs) is a principled framework for solving such decision making problems. Although computing the exact POMDP solution is intractable(Papadimitriou and Tsitsiklis 1987) in general, the past decade has seen approximate POMDP solvers starting to become practical for various robotics problems.

As part of the Kinova MOVO Beta program at the IEEE SIMPAR and ICRA 2018, we apply a general on-line POMDP solver as the sole motion planner of a 6-DOFs manipulator performing a task that requires a relatively large number of steps to accomplish: The robot must move to pick-up a cup placed on a table, scoop candies using the cup, and put the cup back on the table (Figure 1(a) illustrates the task). In this scenario, the robot must trigger when scanning of the environment is required and cope with $\sim$3cm perception errors (due to minimal effort in camera calibration), user's behaviour of moving the cup when the robot is trying to pick it up, and changes to the surface of the candies after each scoop.

We divide the above task into four motion planning problems: Picking up the cup, Approaching the candy box, Scooping, and Placing the cup back on the table. Each problem was modeled as a POMDP and solved using an on-line solver. We found that with almost naive modelling, existing solvers are capable in handling the size of the state and observation spaces posed by each problem.

However, for the picking problem, a major difficulty comes from the curse of history. Due to the various "playful" user's behaviour, a planner needs to perform a large number of lookahead steps to compute a robust strategy. The problem is the size of possible future scenarios that a POMDP solver must take into account is exponential in the number of lookahead steps being considered. As a result, even recent sampling-based on-line solvers(Kurniawati and Yadav 2013),(Silver and Veness 2010),(Somani et al. 2013),(Sunberg and Kochenderfer 2018) have a time complexity that grows exponentially with the number of lookahead steps needed to generate a good solution. Methods have been proposed to alleviate the curse of history issue(He, Brunskill, and Roy 2010),(Kurniawati et al. 2011). However generally, on-line POMDP solvers perform well for lookahead step of up to 10–15 steps (within reasonable computational resources), which is often insufficient for the above problem.

Moreover, the curse of history is amplified by the large action space imposed by a 6-DOFs manipulator. Methods have been proposed to alleviate this issue(Seiler, Kurniawati, and Singh 2015),(Sunberg and Kochenderfer 2018). However, existing solvers can only perform well for problems with 3–4 continuous action spaces(Seiler, Kurniawati, and Singh 2015), while a method that can perform well for problems with 100,000 discrete actions was only recently proposed(Wang, Kurniawati, and Kroese 2018). To alleviate

[*]CSIRO and UQ International Scholarship
[†]Australian Postgraduate Award

this issue, we resort to use a small discrete number of fixed increments and decrements of the joint angles as the action space. This strategy helps to keep the size of the action space manageable, but at the cost of an increased planning horizon.

In this paper, we propose to alleviate the curse of history issue by using macro-actions, represented as finite state machines. We also present the POMDP models for each of the four problems mentioned above and how they are combined together, so as to allow a smooth transition between problems. Furthermore, we present the perception system and how it is integrated with the POMDP-based planner, and some implementation tips and tricks to ensure smooth operation of the POMDP planner on a physical 6-DOFs manipulator.

The above robotics scenario and system was demonstrated as part of a robotics demo that ran for a total of seven days. During this time, 150 runs were attempted, and our solution reached 98% success rate. After the demo, we slightly improved the system and conducted more systematic experiments, in particular in scenarios where a user displaces the cup that the robot tries to pick-up and in cluttered environments. Experiments in such scenarios indicate that our solution reaches no less than 90% success rate.

The System

The manipulator we use is a Kinova 6-DOFs Jaco arm with KG3 gripper, attached to a Kinova MOVO mobile manipulator. In this work, we use only one of the arms of the MOVO and viewed the MOVO as a static base for the Jaco arm. The sensor used is the one attached to the MOVO, i.e., Kinect v2 mounted on the MOVO's head and encoders at the joints of the Jaco arm. By default, MOVO comes with a non-calibrated Kinect v2 sensor. We perform internal calibration of the sensor, but not the external calibration. We found that the error due to no external calibration causes model and perception error that can easily be handled by our solver.

MOVO runs the Robot Operating System (ROS) and comes with two Next Unit of Computing (NUC) computers. Each NUC is an Intel Core i7-5557U@3.1GHz machine with 16 GB RAM. One of these computers is used for sensor processing, while the other is used to process control input and sensor scheduling. We use off-board computing for the planner. The off-board computing is an Intel Xeon E5-1620@3.6GHz with 16 GB RAM. This use of off-board computing is due to different ROS versions used by the solver and the interfaces to the robot.

Problem Scenario and Formulation

We consider a manipulation task in which the above Jaco arm interacts with its environment while being subject to significant uncertainties in motion, sensing and its understanding of the environment. Specifically, the manipulator must grasp and pick-up a cup (with known geometry) that is placed at an arbitrary position on a table in front of the robot, use the cup to scoop candies from a box of candies, and then place the cup containing candies back on the table. The location of the table, the location of the cup on the table, and the location of the box of candies are initially unknown. The average perception error in the object localization is 3cm. Furthermore, the cup location can change at run-time, e.g. a user might pick-up the cup and place it at a different location on the table while the robot is moving to pick it up, whereas the surface of the candies in the box changes after every scoop.

We divide the above task into 4 motion planning problems, i.e.:

Picking: Moving the manipulator from its initial configuration to pick-up a cup from a table, despite not knowing the exact initial position of the table and the cup, possible changes of the cup's position during run-time, and perception errors.

Pre-Scoop: Moving the cup to a pose above the candy box that is beneficial for scooping

Scooping: Scooping candies from a box using the cup, despite not knowing the exact surface of the candies and perception errors

Placement: Placing the cup back to the table, while ensuring that no candies fall out of the cup.

To enable robust operation despite the various types of uncertainties, we formulate each problem as a Partially Observable Markov Decision Process (POMDP). The different POMDP problems are designed to have overlapping state variables, so that the end of one problem automatically transition into the beginning of the next problem with ease.

POMDP Background

Formally, a POMDP is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{O}, T, Z, R, b_0, \gamma)$. The notations $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{O}$ are the state, action, and observation spaces of the robot. At each time-step, a POMDP agent is in a state $s \in \mathcal{S}$, executes an action $a \in \mathcal{A}$, perceives an observation $o \in \mathcal{O}$, and moves to the next state $s' \in \mathcal{S}$. Which next state it ends up depends on the transition function $T(s, a, s')$, which is a conditional probability function $P(s'|s, a)$ representing uncertainty in the effect of actions, while the observation o perceived depends on the observation function Z, which is a conditional probability function $P(o|s, a)$ representing uncertainty on sensing. The notation R is a state-action dependent reward function, from which the objective function is derived. The notations b_0 is the initial belief, while $\gamma \in (0, 1)$ is the discount factor to ensure that an infinite horizon POMDP problem remains a well-defined optimization problem. The solution to a POMDP problem is an optimal policy (denoted as π^*), which is a mapping $\pi^* : \mathcal{B} \mapsto \mathcal{A}$ from beliefs ($\mathcal{B}$ denotes the set of all beliefs, which is called the belief space) to actions that maximizes the expected total reward, i.e. $V^*(b) = \max_{a\in\mathcal{A}} \left[R(b, a) + \gamma \int_{o\in\mathcal{O}} Z(b, a, o)V^*(\tau(b, a, o))\mathrm{d}o\right]$, where $R(b, a) = \int_{s\in\mathcal{S}} R(s, a)b(s)\mathrm{d}s$ and $Z(b, a, o) = \int_{s'\in\mathcal{S}} Z(s', a, o) \int_{s\in\mathcal{S}} T(s, a, s')\mathrm{d}s\mathrm{d}s'$. The function τ computes the updated belief after the robot executes a from b and perceives o.

POMDP Formulation of the Picking Problem

In this problem, the state space is a joint product between the robot's and the cup configurations. Specif-

ically, the state space is defined as $\mathcal{S} = \Theta \times GripperStates \times GraspStates \times \Phi_{obj}$, where $\Theta = (-3.14rad, 3.14rad)^6$ are the joints angles of the arm, $GripperStates = \{gripperOpen, gripperClosed\}$ indicate whether the gripper is open or closed, and $GraspStates = \{grasp, noGrasp\}$ indicates whether the robot has successfully grasped the cup or not. A successful grasp means that the gripper grasps the cup from the side, so as to help accomplish the scooping problem. This condition can be checked easily because the geometry of the cup to be picked is known a priori. The last component, $\Phi_{obj} \subseteq \mathbb{R}^6$ is the set of all poses of the cup in the workspace of the robot. The orientation of the cup is represented as Euler angles, for compactness. Note that in this problem, we do not worry about gimbal lock issues because, the cup does not move until it is in contact with the robot and once in contact, rather than simulating the cup's motion, we derive the pose of the cup based on the robot's configuration when contact happens.

In this problem we do consider obstacles. However, to keep the size of the state space manageable, we do not include the obstacles as part of the POMDP state space. Uncertainties in the estimate of the poses and shapes of the obstacles are dealt with conservatively, by constructing bounding boxes around the obstacles, and enlarging them by the average localization error.

To keep the action space small, the set of primitive actions consists of the set of fixed angle increments/decrements for each joint (denoted as $\mathcal{A}_\theta \subseteq \mathbb{R}^6$), plus `scan`, `openGripper` and `closeGripper` actions, resulting in a discrete action space of size $2^n + 3$, where n is the robot's DOFs. The `scan` action takes $\sim$1.5s, while all other primitive actions take $\sim$0.7s to be executed. In subsection **Alleviating the Curse of History**, we discuss how this set of primitive actions is augmented with macro-actions, so as to reduce the effective planning horizon, thereby alleviating the curse of history issue.

The joint angles of the robot evolve linearly according to

$$\theta' = \theta + a_\theta + e_\theta \tag{1}$$

where $\theta, \theta' \in \Theta$ are the current and next joint angles, $a_\theta \in \mathcal{A}_\theta$ is a vector of joint angle increments/decrements and $e_\theta \sim N(0, \Sigma)$ is an additive control error drawn from a multivariate Gaussian distribution. The parameters are set through experiments with the physical system. Note that this component of the transition function is used throughout the subsequent POMDP problems described in the next subsections.

The actions `openGripper` and `closeGripper` are assumed to be perfect. To determine if the `closeGripper` action results in a successful grasp, we use a simple threshold based-method: Executing the `closeGripper` action corresponds to moving the finger-joint encoders from an opening angle of $0.0rad$ to a closing angle of $1.0rad$. If the resulting closing angles are less than $0.95rad$, we assume that the gripper was not fully closed, i.e. a grasp is established. Otherwise we assume that a grasp was unsuccessful.

The observation space is a joint product of four components. The first component is an estimate of the pose of the cup, which is computed by our perception system (discussed in section **Perception**) and perceived only when the `scan` action is performed. This component is continuous but, for the purpose of solving, we uniformly discretize the space, in the sense that two observations will be considered the same if their L2 distance is less than a pre-defined threshold. The second component comes from the joint-encoders, which measure the joint-angles of the arm. Similar to observation w.r.t. the cup's pose, we uniformly discretize observation regarding the joint-angles of the robot. The third component is a gripper-encoder that indicates whether the gripper is open or closed, and the last component is a grasp detector which observes whether the robot grasps the object or not.

For the observation function, we assume that the gripper-encoder sensor provides perfect information. For the grasping sensor we assume to get a correct reading 90% of the time. For the joint-angle sensor we assume that the encoder readings are disturbed by a small additive error drawn from a uniform distribution with support $[-0.05rad, 0.05rad]$. To accommodate for possible errors in the observations of the object pose, we assume that these pose estimates are drawn from a uniform distribution around the actual pose of the object. Note that the robot only receives an observation on the object pose when a *scan* action is performed.

The POMDP agent receives a reward of 1,750 for a successful grasp and a reward of 1,000 for successfully bringing the the cup to the goal region, which is located slightly above the table. We set a penalty of -250 for every self-collision or collision with the environment, i.e., the table, the cup, and obstacles. Collision with the cup happens whenever the robot touches the cup with any part other than the inside parts of its gripper. Each motion also incurs a small penalty of -3, to encourage the robot to accomplish the task sooner than later. Furthermore, a penalty of -100 is given to failed grasps, to ensure that the cup is firmly grasped, so that the subsequent problem (i.e., scooping) can be accomplished. To reflect the cost of scanning, the robot receives a penalty of -50 when it executes the `scan` action.

In addition to the above rewards, in this particular problem, we added the following three reward components to act as a heuristic to help guiding the search. In case the gripper is closed and a grasp has not been established, the robot receives a penalty of -700 if it does not execute the `openGripper` action. This forces the robot to immediately re-open the gripper after an unsuccessful grasp attempt. On the other hand, in case a grasp has been established, the robot receives a penalty of -250 if the gripper is opened. Additionally, the robot receives a penalty of -200 for states in which the object is "behind" the gripper.

Note that while our POMDP planner is insensitive to the exact values of the reward function, it is important to find a reasonable relation between the different reward components (e.g. large rewards for successful grasps and relatively small penalties for unsuccessful grasps) as this relation eventually determines the behaviour of the robot. This can require some hand-tuning. Automatically finding good reward functions for a particular task is an active field of research(Arora and Doshi 2018).

POMDP Formulation of the Pre-Scoop Problem

Here, the state, primitive action, and observation spaces, as well as the transition and observation functions are the same as for the picking problem. Since the curse of history of this problem is not as severe as the picking problem, the action space of this problems consists of only the primitive actions. During the initial scan, the robot obtains an estimate of the location of the box that contains the candy, as well as an estimate of the height of the candy surface. Based on the estimated location of the box, we construct a small goal-area above the box that must be reached by the robot, such that the cup is inside the goal-area and it faces the candy surface.

Furthermore, the reward function for this problem is simpler than the one used for the picking problem: The robot receives a penalty of -250 when it collides with itself or the environment, and a small penalty -3 for every step it takes. Additionally, the robot receives a large penalty of -250 when the `openGripper` action is executed, as this would result in loosing the grasp. If the robot successfully moves the cup to the goal-area such that the cup faces the candy surface, it receives a reward of 1,000.

POMDP Formulation of the Scooping Problem

In this problem, the robot has to move the cup beneath the candy surface, scoop the candy, and leave the candy box without spillage. The state space consists of the same state variables as the previous two problems, plus two additional variables. The first variable represents the height of the candy surface, so as to account for uncertainty in the candy surface estimate. In the initial belief, this variable is distributed according to a uniform distribution $[-3cm, 3cm]$ around the (point) estimate of the height of the candy surface provided by the perception system. Estimating the height of the candy surface is performed once, prior to execution. The second variable, `enteredCandyBox`, is a boolean that indicates whether the robot has moved the cup beneath the candy surface. The action and observation spaces, as well as the transition and observation functions are the same as the pre-scoop problem.

To achieve scooping we separate the reward function into two parts: The part where the cup hasn't entered the candy box yet and the part where the cup is beneath the candy surface. For the first part, the robot receives a reward of $1,000$ if the cup enters the candy box such that the opening of the cup faces the candy surface. For the second part, the robot receives a reward of $1,000$ once the cup emerges in a near upright orientation. Additionally we penalize backward-motions by -50 as this would prevent the robot from filling the cup with candy. With this simple reward function, the robot is encouraged to perform a scooping-like motion. The robot receives another reward of $1,000$ once it reaches a goal area located above the candy box and holds the cup at a near upright orientation. To avoid spillage, the robot receives a penalty of -500 if the orientation of the cup is more than a pre-defined threshold from an upright orientation.

POMDP Formulation of the Placement Problem

In this problem, we use the same state, action and observation spaces and the same transition and observation functions as in the pre-scoop problem. In terms of the reward function, similar to the scooping problem, we need to make sure that the cup faces upwards to avoid spillage. Hence the robot receives the same penalty of -500 if the orientation of the cup about the X and Y axis exceeds 10 degrees. This strategy proves effective (see Section **Results**). Then, once the robot has reached a state such that the cup is inside the goal area above the table, it receives a reward of 1000 for the `openGripper` action. At the same time, after performing the `openGripper` action, the robot enters a terminal state which concludes the task.

Planning

To solve the aforementioned POMDP problems, we use the Adaptive Belief Tree (ABT) (Kurniawati and Yadav 2013), a state-of-the-art general on-line solver that allows policy adaptation when the POMDP model changes. Similar to many on-line POMDP solvers (Silver and Veness 2010)(Somani et al. 2013), ABT approximates the optimal policy by constructing and maintaining a belief-tree, where nodes represent beliefs and edges represent action-observation pairs. The nodes of a belief tree is a sampled representation of the subset of the belief space that is reachable from the initial belief b_0. Two nodes representing beliefs b and b' are connected via an edge $(a, o) \in \mathcal{A} \times \mathcal{O}$ if $b' = \tau(b, a, o)$.

To construct the belief-tree, ABT samples episodes which are sequences of (s, a, o, r)-quadruples of state $s \in \mathcal{S}$, action $a \in \mathcal{A}$, observation $o \in \mathcal{O}$ and immediate reward $r = R(s, a)$. This is done by sampling a state s_0 from the current belief b_0, selecting an action a_0 based on the Upper Confidence Bounds1 (UCB1) (Auer, Cesa-Bianchi, and Fischer 2002), and using a generative model to sample the next state $s_1 \in \mathcal{S}$ from the transition function $T(s_0, a_0, \mathcal{S})$, an observation $o_0 \in \mathcal{O}$ from the observation function $Z(s_1, a, \mathcal{O})$ (we slightly abuse the notation to use the same notation for both the space and the random variable on sampling from the space), and an immediate reward r_0. The next state s_1 is then associated to the belief b_1 whose parent is b_0 via edge (a_0, o_0). This process iteratively repeats from s_1 until either a terminal state is reached or a state associated with a belief node with untried action. If the first terminating condition happens, the solver backpropagates the sampled reward-trajectory to update the estimates of $\widehat{Q}(b, a)$ for each belief associated to a state in the episode, where $\widehat{Q}(b, a)$ is the value of executing a from b and continuing optimally afterwards. Otherwise, the solver will first estimate the value of the last belief node by simulating a rollout strategy from the last node and then continue backpropagating the sampled reward-trajectory as described above. Furthermore, to help update the policy fast whenever the model changes, ABT maintains a record of the sampled episode and its associated belief tree path. Details on ABT is available in (Kurniawati and Yadav 2013).

The following describes some implementation tips in enabling ABT to generate a good strategy to solve the manip-

ulation problems set in this work.

Alleviating the Curse of History

The curse of history issue in this work is particularly severe in the picking problem. In this problem, the robot must cope with possible changes of the cup's position during run-time, which means it has to consider multiple additional sensing actions to re-localize the cup. However, such a re-localization becomes relevant only when the lookahead step performed reaches beliefs with substantial probability mass of being in states where the gripper is close enough to the cup, that attempting to grasp the cup is not futile, which in general, requires sufficiently long lookahead steps from when a scanning action should take place. To alleviate this curse of history issue, we propose macro actions, represented as finite state machines, that augment the action space of the POMDP problem. In particular, we propose two macro actions (the finite state machines representations are illustrated in Figure 1):

1. The `scan-and-approach` macro action, which consists of two sub-actions: A `scan` action to localize the cup, and an `approach` action, which is a sequence of primitive actions that attempt to bring the gripper close to the estimated location of the cup.

   For the `scan` action the robot queries the perception system to receive an estimated object pose, whereas the `approach` action utilizes a deterministic motion planner.

2. The `grasp-and-rescan` macro action. This macro action is designed to hedge failed grasps. A failed grasp could occur due to two reasons: There is too much uncertainty with respect to the object pose, or the object pose has changed during run-time.In both cases the robot must re-localize the object before attempting another grasp. Hence, if a grasping attempt during execution of the `grasp-and-rescan` fails, the robot immediately performs another `scan` action. However, if grasping attempt is successful, the robot continues with the next action.

In both macro actions, during a `scan` action, if the cup is occluded by the arm, such that the robot does not "see" the object, the robot performs a `retract` action, utilizing a deterministic motion planner to compute a sequence of primitive-actions that brings the arm to a configuration where the object is more likely to be occlusion-free. The transition function for the macro actions are derived from the transition functions of the primitive actions. Note that we designed the `retract` action such that it is performed repeatedly until the `scan` action results in a non-empty observation with respect to the pose of the cup. This is because we assume that the cup is within the viewing area of the robot, but might have been temporarily removed from the table.

To speed up planning, our solver does not run the deterministic planner when `approach` or `retract` action are selected at planning time. Rather, it computes a sampled set of various different joint angle values ($\subseteq \Theta$) and corresponding gripper pose, off-line. During on-line planning, it samples a joint angle value associated with a gripper pose nearest to the desired gripper pose, and assumes that this sampled joint angle is the result of executing the `approach` or `retract` action. During execution, a deterministic motion planner is used to find a sequence of primitive actions to move the robot towards the assumed resulting joint angles. This strategy performs well because we only require the `approach` or `retract` action brings the gripper reasonably close to the object pose estimate.

Rollout Strategy

ABT and most on-line solvers today use a rollout strategy as a heuristic to estimate the value of $Q(b, a)$, when the node b is first expanded via action $a \in \mathcal{A}$. In practice, this heuristic is critical for the performance of ABT, particularly when the belief tree is still shallow. A good heuristic helps ABT to focus on the most promising parts of the tree and therefore converge to a good policy faster.

A commonly used rollout strategy is a greedy approach, i.e., at each rollout step, the planner selects actions with the highest immediate reward. However, two issues arise with this strategy. First, it is too myopic for our problem. Second, it requires simulations to be run at each rollout step, which can be quite expensive. Therefore, we propose to use a heuristic that does not require repeated simulation runs to estimate the aforementioned Q-value. Suppose we need to estimate $Q(b, a)$ for the first time. We first sample a next state $s' \in \mathcal{S}$ according to $T(s, a, s')$. Let $\theta \in \Theta$ be the joint-angle component of s'. The heuristic estimate $\widehat{Q}(b, a)$ is:

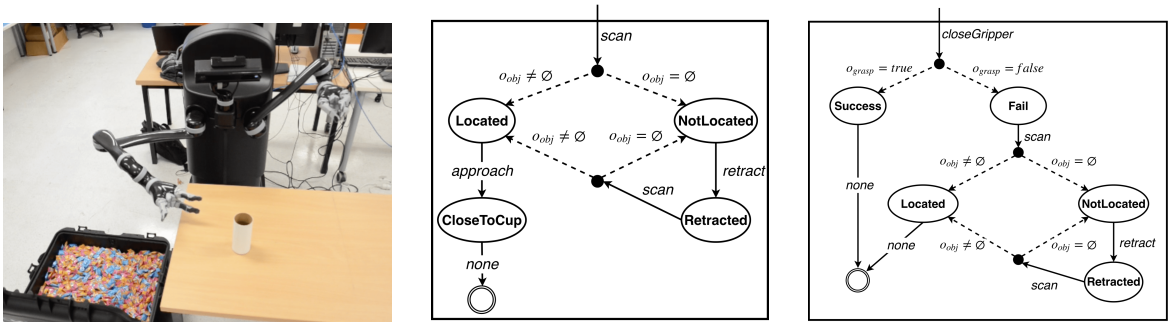
$$\widehat{Q}(b, a) = R_{max} * exp(-\lambda * d_{pos}(g, h(\theta))) \qquad (2)$$

where R_{max} is the maximum possible immediate reward, λ is a scaling factor, h is a function that maps the joint angles θ to a gripper pose in the robot's workspace and d_{pos} is the Euclidean distance function. The notation $g \in \mathbb{R}^3$ is a goal position that is different for each of the four POMDP problems. For the picking problem g refers to the position of the cup in the robot's workspace, whereas for the remaining problems g refers to the position of the goal areas. With this heuristic, we favor expanding nodes where the gripper is closer to the goal position of the respective problem, and therefore help the solver to quickly focus on parts of the belief tree where the gripper moves towards the goal.

Reducing Delay between Steps

Now, the naive implementation of ABT, or any other on-line POMDP solver, follows a strictly sequential order of execution, i.e., policy computation – policy execution – belief update. Such an implementation is likely to result in significant delays during execution, as the robot would have to wait for the solver to update the belief and compute a policy for the new belief. To substantially reduce such delays, we parallelize these tasks where possible by running two processes at the same time.

The first process is the belief-update process. Recall that in ABT, beliefs are represented by sets of particles that are updated using a particle filter. In our implementation we use the Sequential-Importance-Resampling (SIR) particle filter (Arulampalam et al. 2002). SIR particle filtering consists of



(a) Problem scenario (b) `scan-and-approach` (c) `grasp-and-rescan`

Figure 1: (a) Problem scenario. (b) and (c) Finite state machines of the macro-actions. Ovals are machine-states, labeled solid arrows are actions, dashed arrows are observations, and double circles are exit states. o_{obj} are observations with respect to the pose of the cup, whereas o_{grasp} are observation from the grasping sensor.

two steps. First, drawing samples from a proposal distribution, which in our case, $s'_k \sim T(s_k, a_k, \mathcal{S})$ where s_k are particles that are sampled from the current belief and $a_k \in \mathcal{A}$ is the action currently performed. Second, updating the importance weights of the samples s'_k up to a normalization constant based on the observation $o_k \in \mathcal{O}$ perceived, which in our case, $w'_k = w_k Z(s'_k, a_k, o_k)$. Generating samples from the proposal distribution is often quite expensive. However, we can start drawing these samples once the robot starts executing a_k. Then, once the robot receives an observation, all that remains for the belief update is to update the importance weights which can be done fast.

The second process is the policy-update process. Suppose the current belief maintained by the solver is b and $a_k \in \mathcal{A}$ is the action that the solver has estimated to be the best to perform from b. In our implementation, once the robot starts executing a_k, our implementation of ABT will start planning for the next step. The planning time is set to be 0.7s (the smallest execution time for a primitive action) or until the current action finishes execution, whichever is higher. During planning, ABT will sample additional episodes, starting from states sampled from b and performing action a_k as its first action, thereby improving the policy within the entire descendent of b via a_k in the belief tree. This strategy increases the chances that after the robot has executed a_k and the belief is updated based on the observation perceived, a good policy for the next belief is readily available.

Of course, there are cases where even with this strategy, the robot perceives observations that were not explored in the belief tree. In such cases, we restart planning from scratch. However, we found that the above parallel implementation of ABT is sufficient to reduce the delay between perceiving an observation and executing the subsequent action to be under 0.1 sec in all our experiments.

Perception

MOVO is equipped with a Kinect v2 sensor mounted on a pan and tilt actuator. This mechanism is only used to scan the entire scene (i.e., the table and candy box) at the beginning of the execution. The action `scan` uses a pre-defined position and orientation of the sensor, set to ensure that each `scan` action covers the entire workspace of the particular POMDP problem. The Kinect sensor provides both RGB and depth images.

Localizing the Objects

We use both the RGB image and point cloud obtained from the depth image to detect and localize the table where the cup and obstacles are located, the cup and obstacles themselves, the candy box location, and the height of the remaining candy in the candy box.

The localization pipeline for the table, cup, and obstacles is illustrated in Figure 2. To detect the cup, we first trained a CNN, for which we used the SSDLite-MobileNetV2 (Sandler et al. 2018) architecture, via Tensorflow (Abadi et al. 2016). The CNN was pre-trained on the COCO (Lin et al.

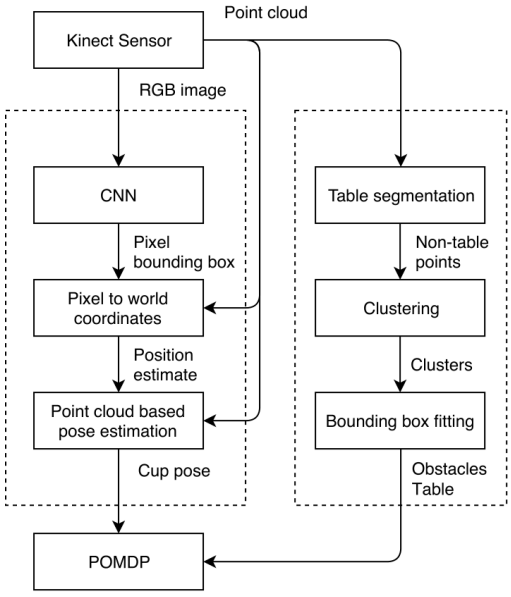


Figure 2: A flowchart of the perception pipeline.

2014) dataset and then fine-tuned for 200k iterations on 200

photos of the cup. However, this CNN only provides a rough estimate of the cup's position. To get a better estimate, we first crop the Kinect point could around the CNN pose estimate, which results in a much smaller point cloud. We then convert the cup's CAD model to an object point cloud, randomly sample object poses close to the CNN estimate and transform the object point cloud to each sample. The final pose estimate is the the sample whose transformed object point cloud is closest (in therms of the sum of pairwise distances) to the cropped Kinect point cloud. To detect obstacles on the table, we remove points corresponding to large planes (i.e. the table) and use a clustering algorithm on the remaining points. We then fit 3D bounding boxes around each cluster to estimate the poses and dimensions of the obstacles. For the localization of the candy box, we use a fiducial marker and information about the size of the box.

In an initial experiment we found that the average error with respect to the pose of the cup is approximately 3cm. Furthermore, the pose estimate can be increasingly inaccurate if the cup is too close to the Kinect sensor. Even factors such as sunlight, varying sensor temperature, sharp object curvature, and highly reflective surfaces decrease the quality of the pose estimate. In principle we could reduce the error by performing external calibration of the sensor. However, since this is often a tedious process, we instead chose to the let POMDP solver handle these errors.

Results

To evaluate our system, we implement our models and strategies using the OPPT framework (Hoerger, Kurniawati, and Elfes 2018) and ran three sets of experiments on the Kinova MOVO mobile manipulator platform, equipped with a 6-DOF arm and gripper. The first set evaluates the effectiveness of our system in picking up the cup when the position of the cup is changed during run-time. The second set evaluates our approach in increasingly cluttered environments. In the last set of experiments we evaluate our system for the entire scenario in a live demo setting.

At the start of each run, the user places the cup at an arbitrary position on the table that is unknown to the robot a-priori. For the first two sets of experiments, a run is considered successful if the robot picks-up the cup and unsuccessful if the robot pushes the cup off the table or pushes the cup to a position where it lies outside the observation range. Runs in which one or more pick-up attempts fail, but the robot is able to recover and eventually pick-up the cup are considered successful. For the third set of experiments, a run is considered successful if the robot is able to pick-up the cup, scoop candies from the candy box and deliver the cup back to the table without spillage or collisions with the environment.

Changing Cup Positions

In this set of experiments, we actively change the position of the cup during run-time by placing it at random positions on the table. For this we divided the experiments into 20 runs with 0, 2, 4 and 6 cup position changes. The occurrence of these cup position changes are random as well. Figure 3 shows snapshots of a typical run where we change the cup location twice during runtime.

Table 1 summarizes the results. In general, the robot showed robust behaviour in picking-up the cup. It successfully handled multiple initially unknown changes in the cup position, occlusions of the cup caused by the arm and imperfect information regarding the environment. Out of the 20 runs, only 2 were unsuccessful. In one of the failure cases, querying the perception system using the `scan` action resulted in a wildly incorrect cup pose estimation. Subsequently the robot attempted to pick-up the cup at a very different location. In doing so it pushed the cup off the table. In the second unsuccessful run, the cup slipped, toppled, and rolled off the table during a pick-up attempt.

Num cup position changes	% of successful runs
0 cup position changes	100
2 cup position changes	95
4 cup position changes	100
6 cup position changes	90

Table 1: Percentage of successful runs for scenarios with 0, 2, 4, and 6 cup position changes. For each scenario the robot performed 20 execution runs

To obtain a better understanding about the utility of the macro-actions for robustly picking up the cup, we ran an additional experiment without macro-actions.

We tested our system using 5 execution runs where for each run, we changed the position of the cup once during run-time. In none of the runs the robot was able to pick-up the cup. Due to due to the limited lookahead, the robot couldn't "see" the benefit of performing the `scan` action to re-localize the cup after an unsuccessful pick-up attempt, hence the `scan` action was never used. This indicates the importance of reducing the effective planning-horizon in this problem.

Cluttered Environments

Similar to the previous set of experiments, the task is to pick-up the cup, but now the robot additionally must avoid collisions with obstacles. This makes the problem significantly harder, since the robot has to plan how to negotiate the obstacles without compromising a successful grasp. For this experiment we do not manually change the cup position. We tested our system using two scenarios, a scenario with one obstacle and a scenario with three obstacles. Figure 4 shows the scenarios used for this set of experiments.

We tested the system in both scenarios using 20 execution runs. For the scenario with one obstacle, 19 out of 20 execution runs (i.e. 95%) were successful, i.e. the robot was able to pick-up the cup while simultaneously avoiding collisions with the environment. For the scenario with 3 obstacles, the robot managed to pick-up the cup in 19 out of 20 execution runs as well (95%). During the two failure cases the robot collided with the white carton (see Figure 4) causing it to tip over. This was caused by using a conservative approach in handling the uncertainties with respect to the

Figure 3: Snapshots of a successful run with two obstacle position changes. The robot attempts to pick-up the cup (first picture) but the cup position is changed to the left corner of the table (second picture). It performs a `scan` action, and attempts to pick up the cup again. But, the cup is again moved to a location close to the robot (third picture). A `scan` action is performed again. In the last picture the robot successfully picks-up the cup at its the new location.

(a) One obstacle (b) Three obstacles

Figure 4: The problem scenario used for the set of experiments with one (a) and three (b) obstacles. The task for the robot is pick-up the cylindrical cup while avoiding collisions with the two boxes and the black cup.

obstacle poses and dimensions. The high success rate indicates that the robot exhibited robust behaviour in scenarios with increasingly cluttered environments.

Live Demo

To test the entire planning process, from picking up an empty cup until delivering a cup of candy back to the table, we ran the entire system for 7 consecutive days in a live-demo setting. For this we let bystanders place the cup at random positions on the table ahead of every execution run. In approximately 150 runs we achieved a success rate of over 98%, i.e. the robot was able to pick up the cup, scoop candy and deliver the filled cup back to the table, showing the effectiveness of our approach in solving the entire planning problem in a non-sterile setting.

Comparison with Deterministic Motion Planner

For comparison, we apply a commonly used motion planner for high DOFs, RRTConnect(Jr. and Lavalle 2000), on the picking problem in an environment without obstacles where the cup's pose is stationary. Given the initial state of the robot, RRTConnect computes a trajectory in the robot's state space that moves the gripper towards the most-likely estimate (from the perception system) of the cup's pose, before attempting to pick-up the cup. Note that for this experiment, we are only interested in how effective RRTConnect is in achieving a successful grasp. Therefore we do not optimize or smoothen the computed trajectories. The RRTConnect-based picking is executed 20×. Among them, only 7 runs resulted in a successful pick-up (compared to a 100% success rate of our POMDP-based planner, see Table 4). While RRTConnect is reasonably effective in moving the gripper towards the estimated cup-pose, due to the perception errors, the resulting gripper pose where a grasp is attempted is often unsuitable to pick up the cup. In contrast, the policies computed by our POMDP-solver result in the robot to carefully approach the cup as to reduce uncertainty with respect to the pose of the cup, e.g. by slightly pushing the cup before attempting to pick it up.

Furthermore, a POMDP-based planner is able to automatically identify actions that can reduce uncertainty while accomplishing its goals. Such motion does not always have to be explicit scanning, especially when scanning is expensive (as in our problem). For instance, the accompanying video (http://rdl.cecs.anu.edu.au/videos/icaps19_candyScooper.mp4) at 00:13 and 00:43 shows that instead of moving straight to the most-likely location of the cup and attempt to grasp it, the robot carefully approaches the cup, "pushing" it slightly a few times before pick-up, hereby reducing uncertainty on the relative position of the cup w.r.t. the hand and enabling more robust grasps compared to RRTConnect-based approaches.

Summary

POMDP solving has advanced tremendously over the past decade. However, its application as the sole motion planner of a high DOFs manipulator is rare. This paper presents our experience in applying an on-line POMDP solver as the sole motion planner of a relatively long manipulation task. A 6-DOFs Jaco arm must pick-up a cup from a table, use the cup to scoop candies from the candy box, and place the cup back on the table, without spilling the candies, in the presence of perception errors affecting the localization of the cup, obstacles and the surface of the candy and "playful" users that keep moving the cup while the robot tries to pick it up. Our POMDP-based system was tested in a 7 days live demo and achieved a 98% success rate. Further experiments to test robustness demonstrate promising results too. We hope this work can be used as a guide for applying this robust method of decision making under uncertainty to physical robots.

References

Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; et al. 2016. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, 265–283.

Arora, S., and Doshi, P. 2018. A survey of inverse reinforcement learning: Challenges, methods and progress. *arXiv preprint arXiv:1806.06877*.

Arulampalam, M. S.; Maskell, S.; Gordon, N.; and Clapp, T. 2002. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on signal processing* 50(2):174–188.

Auer, P.; Cesa-Bianchi, N.; and Fischer, P. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine Learning* 47(2-3):235–256.

He, R.; Brunskill, E.; and Roy, N. 2010. Puma: Planning under uncertainty with macro-actions. In *AAAI*.

Hoerger, M.; Kurniawati, H.; and Elfes, A. 2018. A software framework for planning under partial observability. In *Proc. IEEE/RSJ Int. Conference on Intelligent Robots (IROS)*.

Jr., J. J. K., and Lavalle, S. M. 2000. Rrt-connect: An efficient approach to single-query path planning. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, 995–1001.

Kurniawati, H., and Yadav, V. 2013. An Online POMDP Solver for Uncertainty Planning in Dynamic Environment. In *Proc. Int. Symposium on Robotics Research (ISRR)*.

Kurniawati, H.; Du, Y.; Hsu, D.; and Lee, W. S. 2011. Motion planning under uncertainty for robotic tasks with long time horizons. *The International Journal of Robotics Research* 30(3):308–323.

Lin, T.-Y.; Maire, M.; Belongie, S.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; and Zitnick, C. L. 2014. Microsoft coco: Common objects in context. In *European conference on computer vision*, 740–755. Springer.

Papadimitriou, C., and Tsitsiklis, J. 1987. The complexity of Markov Decision Processes. *Math. of Operation Research* 12(3):441–450.

Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; and Chen, L.-C. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 4510–4520.

Seiler, K.; Kurniawati, H.; and Singh, S. 2015. An Online and Approximate Solver for POMDPs with Continuous Action Space. In *Proc. IEEE Int. Conference on Robotics and Automation (ICRA)*.

Silver, D., and Veness, J. 2010. Monte-carlo planning in large pomdps. In *Advances in neural information processing systems*, 2164–2172.

Somani, A.; Ye, N.; Hsu, D.; and Lee, W. 2013. DESPOT: Online POMDP Planning with Regularization. In *Advances in neural information processing systems (NIPS)*.

Sunberg, Z. N., and Kochenderfer, M. J. 2018. Online algorithms for pomdps with continuous state, action, and observation spaces. In *ICAPS*, 259–263.

Wang, E.; Kurniawati, H.; and Kroese, D. 2018. An online planner for pomdps with large discrete action space: A quantile-based approach. In *Proc. Int. Conference on Automated Planning and Scheduling (ICAPS)*.