Adaptive Motion Planning in Bin-Picking with Object Uncertainties

Thomas Fridolin Iversen[1], Lars-Peter Ellekilde[1] and Jaime Valls Miró[2]

[1] The Maersk McKinney Moller Institute, University of Southern Denmark
Campusvej 55, 5230 Odense M, Denmark, {thfi,lpe}@mmmi.sdu.dk
[2]Centre for Autonomous Systems, Faculty of Engineering, University of Technology Sydney
NSW2007, Australia, Jaime.VallsMiro@uts.edu.au

Abstract: Doing motion planning for bin-picking with object uncertainties requires either a re-grasp of picked objects or an online sensor system. Using the latter is advantageous in terms of computational time, as no time is wasted doing an extra pick and place action. It does, however, put extra requirements on the motion planner, as the target position may change on-the-fly.
This paper solves that problem by using a state adjusting Partial Observable Markov Decision Process, where the state space is modified between runs, to better fit earlier solved problems. The approach relies on a set of waypoints, containing information about which parts of the state space may contain feasible solutions. Waypoints are pushed around the state space by observing which states in the neighborhood lead to successfully solved problems.
Two bin-picking scenarios are modeled with the proposed method. One scenario in which the system receives an object pose update while moving towards the place position. Another where the update includes the object type being grasped out of a fixed number of options, each class to be deposited in a different place. When an online POMDP solver is utilized, the state adjusting POMDP is improving performance by up to 28% on execution times compared to a not adjusted POMDP.

Keywords: Manipulation and Motion Planning, Industrial Robotics, Adaptive Motion Planning

1. INTRODUCTION

In industrial bin-picking, fast and efficient motions are necessary to keep production times low, but also high precision is crucial when placing objects to be used later in an automatic production line. Problems in bin-picking usually arise as sensor systems have a hard time pose estimating with a 100% accuracy. One solution is to re-pick objects in a place without other objects cluttering the scene views. This approach is unfortunately rather sub-optimal as a long time is spent on placing and re-picking objects. Finding the pose error while moving the robot will, however, suppress the need for re-picking. Thus, time is saved, but additional requirements are put on the robot motions.

To that end, this article is concerned with motion planning for two cases of bin-picking, where information is received on-the-fly. In one scenario an object pose error is acquired, and in the other, object type information is received.

Commonly, when doing motion planning for robots, motions are defined in a tree or graph structure where one single route from start to goal constitutes a motion in joint space. When using such motion planners, only the found path is guaranteed to be safe. If an object is grasped with a pose error or other information is received, the planned motion might give unexpected collisions due to wrong information of how the object is located in the gripper. This might be solved with large bounding volumes around the object, but that approach offers no immediate answer to whether a motion can be adapted when information is received.

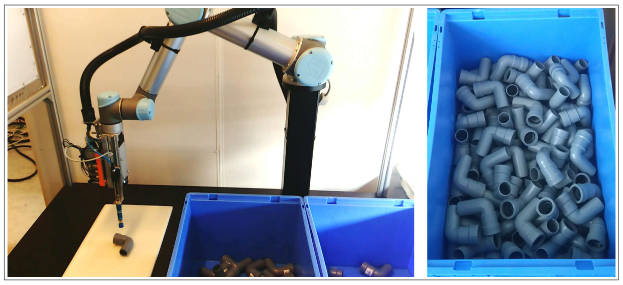


Fig. 1 Bin-picking scene which are used as basis for our POMDP model and for carrying out real world tests along with a picture showing the unsorted objects in the bin.

When planning a motion where new information is expected, e.g. pose corrections of a grasped object or object identification, it is important to ensure that no collisions occur and a quick and stable replacement motion is obtainable. The Partially Observable Markov Decision Process (POMDP) framework [1] offers solutions to both problems stated above; given a probabilistic start belief it can find the safest way through a set of states and when information is gained it can be tunneled into the system. Problems with POMDPs arise however as they scale exponentially with dimensions. Reducing problems to keep them solvable in real-time, can be done with an online POMDP solver [2]. On top of that, our proposed method concentrates a given maximum number of states in feasible places, thereby giving a higher density of states where needed.

The scene which is modeled as a POMDP (see Fig. 1)

contains a Universal Robots UR5 arm with a CB3 controller, a tool mounted sensor and suction cup gripper from Scape Technologies A/S, and two bins with randomly placed objects. The robot is doing pick-and-place operations, picking objects from one or both bins and placing them accurately for further operations in an industrial environment. Fig. 1 also shows an example of the cluttered objects in a bin, which combined with sensor inaccuracies and general grasp errors results in uncertainties of grasped objects. To compensate for those an in-hand recognition sensor is beneficial, as it can identify objects and update information about their poses while the robot is moving. The sensor itself is out of scope for this article which sole focus is adaptive motion planning able to receive information while moving.

The remainder of this paper is organized with Section 2 giving an overview of related work, Section 3 describing formalities of the POMDP framework and Section 4 presenting our proposed method. Sections 5 and 6 describe the tests carried out and the results while Sections 7 and 8 discuss and conclude the paper.

2. RELATED WORK

Motion planning for articulated robot arms is typically done with sampling based motion planners, Rapidly-Exploring Random Tree [3] being a classical approach. These planners can only guarantee a single path to be collision free, unless planning for a set of particles representing uncertainty in a starting pose [4]. Even though a number of attempts have been made to speed up path planning, including both tree structure optimization [5] and sampling methods [6], [7], it is still expensive to plan paths for many particles.

A significant amount of work has been put into the challenge of planning with uncertainties. The classic POMDP solution [1] is a framework for planning with imperfect information of a state using sensor inputs and actions, and has been applied to areas such as robotics, artificial intelligence, operation research and manipulation of objects [8]. Even though the classical approach only involves the discrete case, solving a full POMDP usually is intractable, due to challenges in dimensionality and history [9]. Point based methods have been shown to successfully solve approximations of POMDPs by using only a limited subset of the belief space [10]. Other approaches tries to sample beliefs only from reachable spaces by maintaining an upper and lower bound on the optimal value function to recognize whether a sampled belief is reachable [11].

As not all problems can be stated as discrete POMDPs, efforts have been made to solve the continuous case. Using Gaussian particle filters to represent beliefs [12] try to overcome the need of discretization, but when the environment becomes too complex the number of Gaussian components escalate. Thus instead of using a Gaussian particle filter, Van den Berg et al. [13] uses an extended Kalman filter to provide locally optimal solutions in polynomial time. Furthermore continuous POMDPs can be solved by Monte Carlo Value Iteration-based methods, which also use particles to represent beliefs and α-vectors to represent a policy graph [14].

Online planning algorithms omit precomputation entirely, finding only local policies for each step in the execution. Actions are chosen based on forward-search of the action and observation tree rooted at the current belief state [2][15]. These algorithms can produce solutions for very large POMDPs but also risk getting stuck at local minima.

3. POMDP PRELIMINARIES

To gain a better understanding of our proposed method, we here present a concise description of the general POMDP framework.
A POMDP is formally a tuple of $\{S, A, O, T, Z, R, \gamma\}$, where S represents a set of states, A is a set of actions manipulating the system to move between states and O is a set of observations. In each time step the system is in some state $s \in S$ and an action is performed, bringing the system in state s' with conditional probability $T(s, a, s') = p(s'|s, a)$. Afterwards an observation is made to gather information about its actual state, which is modeled as the conditional probability $Z(s, a, o) = p(o|s, a)$.

The objective of the system is to gain as high a reward as possible when traversing the states. It receives a real-valued reward by taking action a in state s, $R(s, a)$. When a POMDP has infinite horizon, i.e. an infinite amount of steps can be taken, the discount factor $\gamma \in [0, 1)$ is used to obtain a well-defined problem by having a finite total reward. To maximize its expected reward the system has to perform suitable sequences of actions given a belief b, or probability function over S, of the system.

The solution to a POMDP, a policy π, represents mappings from all beliefs to actions. The expected value of a belief b is present in the value function, which is found by following a policy π with initial belief b:

$$V_\pi(b) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(b_t)) \,\middle|\, b_0 = b\right] \tag{1}$$

One way of solving POMDPs online is to construct a belief tree with the current belief at the root and perform a look-ahead search for a policy π that maximizes $V_\pi(b_0)$. In such a tree, each node represents a belief, a node branches into $|A|$ action edges and further into $|O|$ observation edges. It is possible to find an optimal policy for POMDPs by using value iteration [11] or policy iteration [16], but when dealing with large dimensional spaces these methods are in the worst case intractable [9]. The Determinized Sparse Partially Observable Tree (DESPOT) [2] randomly samples a set of scenarios to be investigated instead of exploring the whole belief tree.

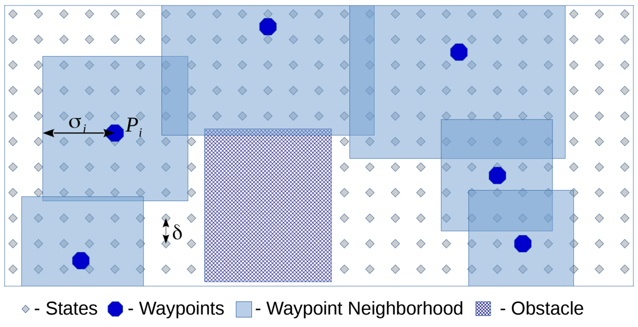


Fig. 2 Principal 2D example of the proposed method.

Each scenario contains a sequence $\Phi = \{s_0, \phi_1, \phi_2, ...\}$, where s_0 is a start state sampled from b and ϕ_i defines s' according to the transition probabilities $T(s, a, s')$. By simulating a finite number of steps an approximate value function is obtained, which can be used to determine the next action. After performing the action and receiving the observation a new belief tree is created and the process is repeated.

4. PROPOSED METHOD

This section describes our proposed method. Note that distances between states should be measurable and that the state space should be discretizable by a given stepsize. In some problems, e.g. decision-making problems, state spaces might not be discretizable at all, but for problems concerning robot control, state spaces are usually given as Cartesian or joint configuration spaces.

When using methods such as DESPOT, a trade-off between optimality and speed/tractability has to be made. The more transitions that are possible, the bigger parts of the belief tree have to be ignored to keep the problem tractable. Our state adjusting POMDP (SA-DESPOT) seeks to use prior knowledge of obtained solutions to concentrate the search in workspace areas with higher probability of containing solutions. The aim is to plan shorter paths which is strongly correlated with the moving time.

Our method utilizes a set of waypoints W guiding which subpart of the space has previously contained usable states. Each waypoint $w_i = \{P_i, \gamma_i\}$ contains a coordinate P_i, and a neighborhood around that coordinate, γ_i. When one or more problems have been solved, W is updated according to surrounding states usefulness and the POMDP model is updated only incorporating states within neighborhoods of waypoints, which means $||P_i - P_j||_\infty < \gamma_i$, where P_i is the position of the i'th waypoint, P_j is the position of the j'th state and γ_i is the neighborhood of the i'th waypoint. A 2D example is shown in Fig. 2, placing waypoints in state space and states within waypoint neighborhoods are used, while states outside are ignored.
After a POMDP solution has successfully been found, the following steps are taken.

State scores: Each POMDP state is assigned a score, labeling how useful it has been in previous solution attempts. Usefulness of a state depends on how close it has been to previous solutions and is defined as;

$$\lambda_j = \min_{\theta_i \in \Gamma} ||P_j - \theta_i|| \tag{2}$$

where θ_i is a state traversed in the found solution Γ. Scores are normalized to have a value between 0 and 1 for easier scaling later on:

$$\widehat{\lambda}_j = \frac{\lambda_j}{\max(\lambda_0, \lambda_1 ... \lambda_n)} \tag{3}$$

Waypoint neighborhood: The neighborhood of a given waypoint is dependent on its surrounding states scores. If a waypoint is surrounded by states with high scores, its neighborhood should be smaller, indicating high probability of feasibility in that area. Thus, the neighborhood of waypoint w_i is given as:

$$\gamma_i = 1 - \sum_{\widehat{\lambda}_j \in \Lambda_i} \frac{\widehat{\lambda}_j}{|\Lambda_i|} \tag{4}$$

where Λ_i is the scores for the set of states within a distance, d_i, of w_i. The value of d_i is chosen to balance between exploration versus exploitation and can be lowered as the algorithm advances.

Waypoint movement: Each waypoint is moved towards close states with high state scores:

$$\Delta x_i = \left[P_i - \left(\arg\min_{s \in S_p} ||P_i - s|| \right) \right] \cdot \epsilon \tag{5}$$

where Δx_i is the displacement of P_i, S_p is the set of states within distance $d_p = d_i$ and ϵ is a scaling factor defining how far to move towards the state with highest score.

Adding/Removing waypoints: After moving waypoints, all waypoints distances to all other waypoints are calculated and waypoints are added or removed based on distances between them. If a waypoint is contained in another waypoints neighborhood, it is removed. To guarantee that there always exists a way from the start belief to the end state, waypoints are added if the neighborhood of a waypoint does not reach halfway towards its nearest waypoint neighbor.

Stepsize alteration: As the computational bottleneck of POMDP solutions is the number of transitions, a fixed amount is wanted to ensure that the model is always tractable. As we do not adjust the number of actions per state the only variable determining the number of transitions is the number of states. The maximum number of states given a set of waypoints is bounded by

$$|S_{bound}| \leq \sum_{\{P_i, \gamma_i\} \in W} \left(\frac{\gamma_i}{\delta} \right)^n \tag{6}$$

where δ is the stepsize and n is the dimension of the state space. Distances between states are altered according to the expected amount of states included in waypoint neighborhoods, as follows:

$$\delta = \left(\frac{\sum\limits_{\{P_i,\gamma_i\}\in W} (\gamma_i)^n}{|S_{desired}|} \right)^{1/n} \tag{7}$$

where $|S_{desired}|$ is the desired number of states. Finding the stepsize according to (7) is based on an estimate of the number of states in (6) as some states may be counted twice if they are close enough to more than one waypoint, though the algorithm is not dependent on having an exact amount of states but rather an upper bound to secure tractability.

A pseudo-algorithm of the steps taken when updating waypoints and stepsize can be seen in Algorithm 1. The algorithm is executed after a number of successfully generated path and outputs a new set of waypoints with updated neighborhoods and a stepsize for discretizing the state space.

Algorithm 1 Update step of SA-DESPOT

```
procedure UpdateStateSpace(Γ, S, W)
▷ Set of traversed states in found solution Γ, discrete state space S, and waypoints W.
    for state scores $\lambda_j \in S$ do
        $\lambda_j = \min_{\theta_i \in \Gamma} ||P_j - \theta_i||$
    NormalizeStateScores, (Eq. 3)
    for waypoint $w_i = \{P_i, \gamma_i\} \in W$ do
        $\gamma_i \leftarrow$ UpdateNeighbourhood, (Eq. 4)
        $\Delta x_i \leftarrow$FindDisplacement, (Eq. 5)
        $P_i = P_i + \Delta x_i$
    for waypoint $w_i = \{P_i, \gamma_i\} \in W$ do
        for waypoint $w_j = \{P_j, \gamma_j\} \in W | j \neq i$ do
            if $\|P_i - P_j\|_\infty < \gamma_i$ then
                RemoveWaypoint($w_j$)
        $d_i = \min_{w_j \in W - w_i} \|P_i - P_j\|_\infty$
        if $\gamma_j + \gamma_i < d_i$ then
            AddWaypoint($\{\frac{P_i+P_j}{2}, d_i - \gamma_i - \gamma_j\}$)
    $\delta \leftarrow$UpdateStepsize, (Eq. 7)
    return $W$                ▷ Return updated waypoints
```

4.1 Discussion of Proposed Method

As the proposed method relies on adding, removing and adjusting state positions of a POMDP problem, an argument that the problem remains solvable and will generally converge towards a more feasible discretization of the workspace, is suggested.

The criterion for the problem to remain solvable is that there always exists a path from the pick place and to the end positions. When initializing, the path comes from RRT* and is therefore per definition valid. At a given update step, states will only get more concentrated around the previous found valid path, which therefore will not disappear after the update step, keeping the path solvable.

A problem may occur if one of multiple place positions rarely is chosen. The waypoint at that place position will remain untouched, but with a large neighborhood resulting in a sparse discretization. In such a scenario, if no solution can be found of the POMDP, an RRT* path can be generated and used in the same way as when initializing.

The local optimality of waypoint updates is inherited. The updates are done on the basis of previously found paths, which are locally optimal because of the POMDP structure always seeking to find the best solution to the problem. The local optimality of the suggested algorithm comes from the fact that waypoints are moved towards these optimal paths, reminiscent of a gradient descent method, making it locally optimal.

5. TEST SETUP

This section describes how the problem of bin-picking with grasp uncertainties is transformed into a POMDP model and how tests are performed. The general framework for defining and solving a POMDP is described in Section 3.

The problem to be solved, as described in Section 1, contains a pick-and-place scenario with a UR5 robot receiving information from an in-hand recognition sensor while moving between the pick and place positions. Though the sensor is out of scope of this article, worth noting is that information is transmitted once for each pick-and-place operation and we estimate the approximate recognition time as 1 second from when an object is picked up.

Two different bin-picking scenarios are modeled as POMDPs; one scenario receiving an object pose correction, and one receiving a discrete identification of the grasped object. The two scenarios are briefly described here:

Object Pose Correction (OPC)
When an object is picked in the bin, object pose uncertainties may arise from sensor inaccuracies, grasp errors or due to the cluttered environment in the bin. Instead of utilizing re-picking, as done in Fig. 1, the in-hand recognition sensor transmits an object pose correction, resulting in a slightly updated state of the system, the task is to place the picked object in a desired fixed location.

Object Identification (OI)
When an object is picked from the bin, the type of the object is not known. There are 4 different object types each having a unique place position, represented by coordinate systems in Fig. 4(c). At some point in time, the in-hand recognition sensor will notify of which object type has been grasped.

5.1 POMDP Model

Many of the POMDP elements described in the following sections are the same for both scenarios, and therefore unless stated otherwise, applies to both scenarios.

States: States are object poses defined as full 6D

Cartesian coordinates with an XYZ Euler angle rotation. For all states, the robots tool center point (TCP) is placed at the grasp position and checked for inverse kinematics solutions and collisions before they are added to the model. To make sure that inverse kinematics solutions will not result in the robot moving outside the linear path in Cartesian space between states, the joints of the robot are limited to make sure only one inverse kinematics solution exists. A state, s_f, indicating failure, collision or in other ways unwanted behavior is also added in the end of S.

For the OI scenario, each state furthermore contains a number indicating the object type.

Actions: An action is the movement between two neighboring states, meaning a positive or negative move along or around one axis, giving 12 actions. Furthermore are actions combined in pairs matching all with each other once, giving additional actions that can be e.g. both a rotation and a translation or two translations, resulting in a total 144 actions for each state. However, 12 combinations will result in no effective movement and 12 combinations will result in the same action being combined with itself, which is not desired. Furthermore half of the combinations are redundant as we do not distinguish between the order, giving 72 actions when all undesired actions are removed and the 12 basic actions are included.

Transitions: Transition probability for a given state and action is

$$p(s_i|s_j,a) = \begin{cases} 0, & \text{if } \nexists\, s_i \in S : s_i = s_j \cdot T_a \\ 0, & \text{if } collision \\ 1, & \text{otherwise} \end{cases} \tag{8}$$

where $collision$ is true if a collision occurs when doing a movement from s_j to s_i and T_a is the transformation representing action a.

Start belief: For the OPC scenario, the start belief is given by the initial uncertainties present when picking up an object. These are approximated with Gaussian distributions having a mean corresponding to that perceived of the sensor system. As states are multidimensional, the probability of starting in, or belief of, state s is

$$b(s) = \frac{1}{\sqrt{(2\pi)^n \det(\Sigma)}} e^{\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right)} \tag{9}$$

where Σ is the covariance matrix, μ is the object pose as observed by the sensor system and x the state pose.

In the OI scenario, the start belief is uniformly distributed over the 4 possible object types with a 25 % chance on each possibility.

Observations: Essentially there exist two kinds of observations in each step. Either no new information is available about the system, that is no pose correction or object identification is obtained, or new information is available which means the in-hand recognition has happened. If no new information is acquired, an observation o_0 is received meaning that

$$b(s_i|s_j,a,o_0) = b(s_j) \tag{10}$$

If new information is gained, an observation corresponding to the correct state, s_i, is received, meaning that

$$\forall_j b(s_i|s_j,a,o_i) = 1 \tag{11}$$

Rewards: When performing any action in any state a reward of $r_s = -1$ is received, with two exceptions; if an action leads to the end state a reward of $r_g = 1000$ is given and if the action leads to the fail state a reward of $r_f = -1000$ is given. It should be noted that the exact values of these rewards are not important as long as $r_f \ll r_s \ll r_g$.

5.2 Test Procedure

To solve the adjusted POMDPs DESPOT is used, other online solvers could also be used, but as DESPOT have proven useful on a wide range of problems especially those with a small optimal policy [2] and its asymptotical optimality, it is chosen. It is initialized with 400 particles, a search depth of 20 and a discount factor of 0.95, these values are empirically found to optimize both the state adjusting and general case. As the algorithm works in real-time two steps are planned ahead to allow the robot controller to blend through states, each planning step is limited to 0.1s, which in the given case is sufficient to ensure that the next action is determined before the previous has finished executing.

The algorithm is tested against plain DESPOT and to gain a fair comparison 500 paths are executed from the system shown in Fig. 1. To test whether the algorithm converges towards consistent results, 10 runs are made from scratch with different waypoint initializations.

The path needing planning starts in the right bin where an object has been picked and ends to the left in the back of the cell for the OPC scenario and in one of the four place positions for the OI scenario. Object pick positions vary in both translation and rotation and are in our tests all contained in the right bin. All other actions, e.g. image acquisition and moving back after placing an object, are not considered as these do not include any uncertainties. Actions are carried out as linear motions in Cartesian space and waypoints and states are updated for every 10 solved problems, giving a total of 50 waypoint updates.

As described earlier the in-hand recognition sensor is not available to us yet, thus we simulate corrections at a random time for testing purposes. The time of correction is determined by a normal distribution with $\mu_t = 1s$ and $\sigma_t = 0.05$, the pose correction in the OPC scenario is estimated with a normal distribution $\mu_p = 0$ and $\sigma_p = 0.01$ in translation and $\sigma_r = 0.15$ in Euler XYZ angles, where

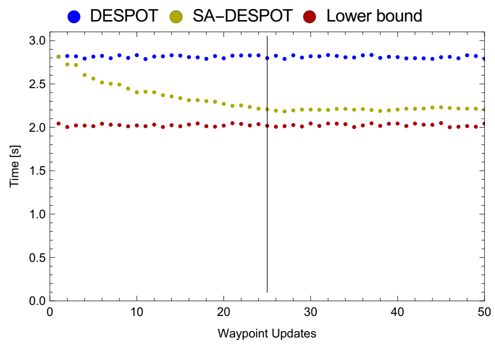
μ and σ are respectively the mean and standard deviation and translation is measured in meters and rotation is in radians. The object identification in the OI scenario, is chosen randomly with a uniform distribution over the four object types.

$|S_{desired}|$ is found empirically to 10000 being the maximum limit of tractable states with 60 actions, giving a total of $6 \cdot 10^5$ transitions, ϵ is set to 0.1 and $d_i = d_p = 2 \cdot \delta$. Tests are carried out on an Intel Core i7 CPU 4600U with 8 GB RAM running Ubuntu 16.04 and a Universal Robots UR5 robotic arm with the CB3 controller.

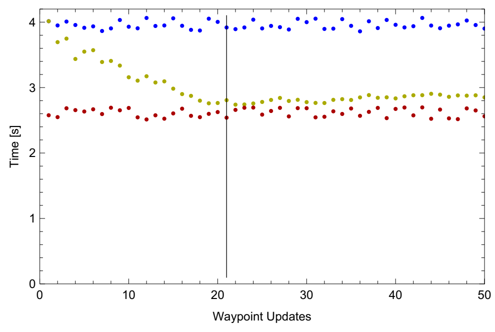
6. RESULTS

This section detail results obtained when carrying out the tests described in Section 5.

Fig. 3 shows the mean moving time of the robot for plain and SA-DESPOT. The state adjusting approach learns for approximately the first 20 to 25 waypoint updates in the two scenarios after which no significant change in moving times are observed. Plain DESPOTs moving time stays the same throughout the entire experiment.



(a)OPC



(b)OI

Fig. 3 Moving time on the two scenarios, for DESPOT, SA-DESPOT and the estimated lower bound.

Table 1 shows mean moving times on the two scenarios for times after SA-DESPOTs initial learning, represented by the vertical black lines in Fig. 3, which is approximated to be after 25 and 21 waypoint updates.

The difference between the two approaches is notable. SA-DESPOT improves the moving times by 20.58% and 28.89% on the two scenarios respectively, because paths are shorter due to the higher state density in critical areas. For comparison an estimated lower bound for each of the motion problems is made, also seen in Fig. 3. The bound is made by running the asymptotically optimal planner RRT* from scratch for 500 seconds for each task and executing the path on the robot. The bound is not necessarily the correct minimum execution time obtainable by POMDP, but it is a fair estimation. As 4 different goal positions are used in the OI scenario, the lower bound is varying as each position is not placed equally far away from the pick box.

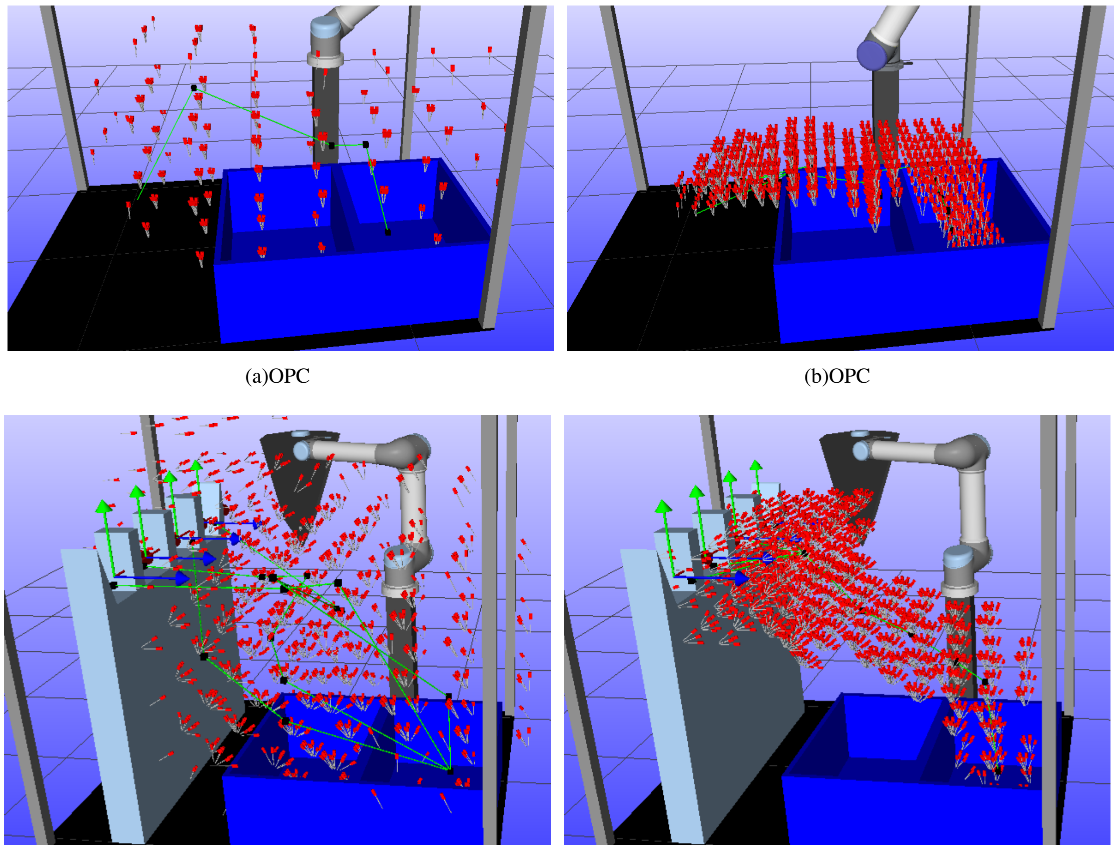
Table 1 Moving times for DESPOT and SA-DESPOT on the two scenes. Results are shown as mean $\pm$ 95% confidence interval along with the improvement.

	OPC	OI
Plain DESPOT	2.77±0.03	3.98±0.05
SA-DESPOT	2.2±0.02	2.83±0.05
Execution time reduction	20.58%	28.89%

Table 2 shows results from running SA-DESPOT from scratch with 10 different initializations on the two scenarios. The *Initialization* column shows the number of waypoints at initialization along with the length of RRT* paths and mean moving times of DESPOT. The *After Learning* column shows the number of waypoints after the initial learning period of SA-DESPOT along with the mean moving times after the learning period. As 4 initialization paths are made in the OI scenario, both path lengths and the number of waypoints are higher than in the OPC scenario. Variations are seen in both the number of waypoints and length of the found RRT* paths in both scenarios, demonstrating diversity in the different initializations. Noteworthy is the decrease of waypoints and standard deviation of these in both scenarios, indicating that different initializations of SA-DESPOT manage to converge towards similar results. The small deviations in moving times of SA-DESPOT come from paths being different in each iteration, as the information received is randomly distributed. A more qualitative result is shown in Fig. 4 where states and waypoints are shown before and after 50 waypoint updates. In Fig. 4(a) and 4(c) states are spread out in most of the workspace whereas states are concentrated in a band in Fig. 4(b) and 4(d). Noteworthy in Fig. 4(b) is that waypoint neighborhoods get smaller when moving towards the goal position, seen by the smaller area occupied by states because different starting positions are used but all of them are going to the same goal position. In Fig. 4(d) states are concentrated in a band going out of the pick box and spreading out when coming closer to the four place positions.

7. DISCUSSION

Results in Section 6 show that the proposed SA-DESPOT algorithm learns for the first 20 to 25 updates in the modeled application, and afterward flattens with


(a)OPC (b)OPC

(c)OI (d)OI

Fig. 4 States and waypoints when the two systems are initialized and after 50 updates. Black dots are waypoints while red markers each represent states/object poses with translation and rotation. The green lines are added for easier visualization.

an obtained improvement of 20.58% and 28.89% compared to plain DESPOT. While this is a fairly good result further improvement may be achievable.

Since linear motions in Cartesian space are used, the maximum speed limit is somewhat lower than that suggested, as de- and accelerations occur more frequently when performing the different actions compared to linear movements in joint space.

The distance measure to determine whether a state should be included or excluded is currently calculated at the center of each waypoint, resulting in states surrounding waypoints instead of the path they are spanning. A more proper approach is to find the distance to lines connecting waypoints, giving a more uniform distribution of states.

Another side-effect of the before mentioned distance measure is seen in Fig. 4. When initialized the neighborhoods of each waypoint are quite high because exploration of the workspace is wanted and avoiding getting stuck in local minima. As the precision of Eq. (6) depends on overlapping neighborhoods in waypoints, the initial guess with large neighborhoods raise the number of states being counted twice. When neighborhoods decrease, so does the number of overlapping states. Thus, Fig. 4(a) shows fewer states than Fig. 4(b). When adding and removing waypoints, the proposed method might remove a waypoint and afterward add another one close to or equal to the removed one. While this is not critical for the algorithm, it might be avoided, by only removing a waypoint if it is inside two other waypoint neighborhoods, instead of just one. Eq. (9) uses a multivariate Gaussian distribution for all dimensions to determine the starting probability of each state, while this is an appropriate approach for the translational dimensions, it is not optimal for rotational dimensions and a more appropriate probability function, e.g. von Mises-Fisher [17] would be preferred.

8. CONCLUSION

This paper has presented a state adjusting POMDP modeling approach (SA-DESPOT) for motion planning with incomplete information, represented as object pose and type uncertainties when doing pick-and-place operations. The method relies on a set of waypoints, which are updated online based on past problems and used to in- and exclude states of a POMDP model. Hereby the algorithm ensures that a modeled POMDP problem remains tractable, by limiting the number of states and actions to a given maximum.

The online solving algorithm DESPOT is used to solve

Table 2 Results of running SA-DESPOT with different initializations 10 times on the two scenes. WPs represent how many waypoints was created at initialization, RRT* is the Euclidean length of the found paths used for initialization and DESPOT and SA-DESPOT are the mean moving times for the two algorithms after SA-DESPOT has stopped learning.

	OPC					OI				
	Initialization			After Learning		Initialization			After Learning	
Run No.	WPs	RRT	DESPOT	WPs	SA-DESPOT	WPs	RRT	DESPOT	WPs	SA-DESPOT
1	6	3.64752	2.7990	5	2.1806	12	15.2983	3.9555	8	2.9024
2	7	3.75111	2.7574	5	2.1665	14	13.4688	3.9510	8	2.8601
3	5	3.43324	2.8025	5	2.2087	9	18.0224	3.9440	8	2.7502
4	5	3.69016	2.7235	5	2.1678	15	15.8694	3.9772	8	2.8357
5	4	3.25115	2.8200	5	2.2087	11	14.641	3.8983	8	2.9121
6	5	4.07882	2.8183	5	2.2347	12	15.9135	4.0350	9	2.8203
7	5	4.29201	2.7950	6	2.2103	12	14.7931	4.0394	9	2.8586
8	6	4.84673	2.7768	5	2.1718	11	17.9372	4.0442	8	2.8602
9	7	3.08367	2.7301	5	2.2365	18	16.1606	3.9661	8	2.7372
10	3	3.1682	2.7429	5	2.2335	15	15.4278	4.0559	8	2.8435
	5.3±1.25	3.72±0.55	2.77±0.03	5.1 ±0.31	2.2±0.02	12.9±2.61	15.75±1.406	3.98±0.05	8.2±0.42	2.83±0.05

the modeled POMDP in real time for a bin-picking scenario. Moving times on the robot are compared to a plain POMDP solved with DESPOT and shows a noteworthy improvement.

REFERENCES

[1] R. D. Smallwood and E. J. Sondik, "The optimal control of partially observable markov processes over a finite horizon," *Operations Research*, vol. 21, no. 5, pp. 1071–1088, 1973.

[2] A. Somani, N. Ye, D. Hsu, and W. S. Lee, "Despot: Online pomdp planning with regularization," in *Advances in neural information processing systems*, 2013, pp. 1772–1780.

[3] S. M. LaValle, "Rapidly-exploring random trees a new tool for path planning," Iowa State Univ., Comp Sci. Dept, Tech. Rep., 1998.

[4] Y. Huang and K. Gupta, "Collision-probability constrained prm for a manipulator with base pose uncertainty," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2009, pp. 1426–1432.

[5] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 2. IEEE, 2000, pp. 995–1001.

[6] T. F. Iversen and L.-P. Ellekilde, "Kernel density estimation based self-learning sampling strategy for motion planning of repetitive tasks," in *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ Intl. Conf. on*. IEEE, 2016, pp. 1380–1387.

[7] M. Zucker, J. Kuffner, and J. A. Bagnell, "Adaptive workspace biasing for sampling-based planners," in *Robotics and Automation, 2008. ICRA 2008. IEEE Int. Conf. on*. IEEE, 2008, pp. 3757–3762.

[8] A. R. Cassandra, "A survey of pomdp applications," in *Working notes of AAAI 1998 fall symposium on planning with partially observable Markov decision processes*, vol. 1724, 1998.

[9] O. Madani, S. Hanks, and A. Condon, "On the undecidability of probabilistic planning and infinite-horizon partially observable markov decision problems," in *AAAI/IAAI*, 1999, pp. 541–548.

[10] D. Hsu, W. S. Lee, and N. Rong, "A point-based pomdp planner for target tracking," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conf. on*. IEEE, 2008, pp. 2644–2650.

[11] J. Pineau, G. Gordon, S. Thrun *et al.*, "Point-based value iteration: An anytime algorithm for pomdps," in *IJCAI*, vol. 3, 2003, pp. 1025–1032.

[12] A. Brooks, A. Makarenko, S. Williams, and H. Durrant-Whyte, "Parametric pomdps for planning in continuous state spaces," *Robotics and Autonomous Systems*, vol. 54, no. 11, pp. 887–897, 2006.

[13] J. Van den Berg, S. Patil, and R. Alterovitz, "Motion planning under uncertainty using differential dynamic programming in belief space," in *Intl Symposium on Robotics Research*, 2011.

[14] S. Thrun, "Monte carlo pomdps." in *Advances in Neural Information Processing Systems*, vol. 12, 1999, pp. 1064–1070.

[15] H. Kurniawati and V. Yadav, "An online pomdp solver for uncertainty planning in dynamic environment," in *Robotics Research*. Springer, 2016, pp. 611–629.

[16] E. J. Sondik, "The optimal control of partially observable markov decision processes." *PhD thesis, Stanford University*, 1971.

[17] R. Fisher, "Dispersion on a sphere," in *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 217, no. 1130. The Royal Society, 1953, pp. 295–305.