Planning robot manipulation to clean planar surfaces

David Martínez, Guillem Alenyà and Carme Torras

Institut de Robotica i Informatica Industrial (CSIC-UPC), Llorens i Artigas 4-6, 08028 Barcelona, Spain {dmartinez,galenya,torras}@iri.upc.edu

Abstract

This paper presents a new approach to plan high-level manipulation actions for cleaning surfaces in household environments, like removing dirt from a table using a rag. Dragging actions can change the distribution of dirt in an unpredictable manner, and thus the planning becomes challenging. We propose to define the problem using explicitly uncertain actions, and then plan the most effective sequence of actions in terms of time. However, some issues have to be tackled to plan efficiently with stochastic actions. States become hard to predict after executing a few actions, so replanning every few actions with newer perceptions gives the best results, and the trade-off between planning time and plan quality is also important. Finally a learner is integrated to provide adaptation to changes, such as different rag grasps, robots, or cleaning surfaces.

We demonstrate experimentally, using two different robot platforms, that planning is more advantageous than simple reactive strategies for accomplishing complex tasks, while still providing a similar performance for easy tasks. We also performed experiments where the rag grasp was changed, and thus the behaviour of the dragging actions, showing that the learning capabilities allow the robot to double its performance with a new rag grasp after a few cleaning iterations.

Keywords: robotics, surface cleaning, AI planning, probabilistic planning, rule learning

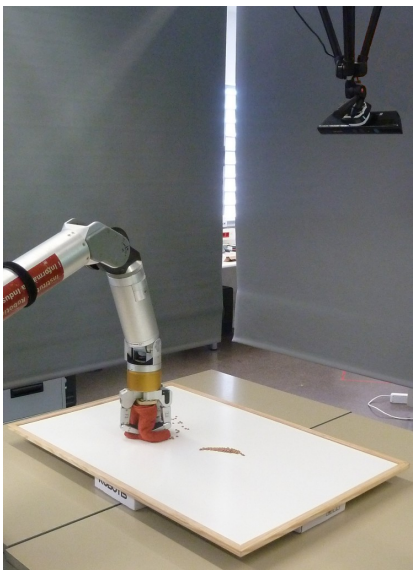
1. Introduction

Robots in household environments tend to move slowly to produce human safe actions, and therefore a lot of the time spent in a given task is devoted to carefully move the robot. The challenge is to produce sequences of actions that minimize the number of robot motions for a given task, and thus the overall time.

A symbolic planner can be used to select the best sequence of actions, the *plan*, in terms of some *metrics*. In this work we use the criterion of minimum time for the whole task, including the computing time and the execution time. Given a *state* (a representation of the environment), and a set of *rules* (definitions of the set of actions that can be executed), the planner computes the best plan to complete the task.

The robot interacts with a real environment by executing actions, and such interaction may change it in an unpredictable manner. Hence, it is desirable that the robot keeps replanning and adapts the plan to the unexpected changes that may arise.

In this paper we present a new approach to clean surfaces with plans of robot dragging actions. We have developed a system that continuously updates a state representing the



(a) WAM arm

(b) REEM service robot

Figure 1: The two robotic platforms used in the experiments performing cleaning tasks.

dirt on a planar surface, and has a set of actions designed to clean all distributions of dirt, including dirt *grouping* actions, and dirt *cleaning* actions. We show that although symbolic planners are usually tested in theoretical scenarios where problems have stricter constraints and states are well defined (Lang and Toussaint, 2010), they also provide good results in handling the uncertainty present in real problems, as the one addressed in this work. The experiments are performed in two platforms: a commercial REEM service robot, and a WAM manipulator cell, both equipped with a RGB-D camera (see Fig. 1), and the task is to clean lentils from a table.

Actions maintaining contact are usually implemented using force control, but force feedback is not available in the REEM robot. This paper shows the alternative of using depth information from RGB-D cameras and deformable tools like cloth, which provides some compliance. Although this alternative adds some extra uncertainty, it can be tackled with probabilistic planning without worsening much the results. Note that the skills for grasping a piece of cloth are already available (Ramisa et al., 2014).

Adaptation is another important skill in household robotics. When facing a new environment, a robot has to adapt to perform efficiently. We propose initializing a generic and optimistic rule model, and integrate a learning heuristic that updates the model to adapt to the features and peculiarities of each robot, surface, dirt or tool. After some learning iterations, the robot will have an accurate model to be used by the planner.

This paper is structured as follows. Section 2 presents some previous work. The proposed algorithm is introduced in Sec. 3, where perceptions, the set of actions (Sec. 3.2), and the planning strategy (Sec. 3.3) are presented. Section 4 explains the details of using a planner with uncertain actions. Section 5 presents the use of learning to improve the model and thus the planning results. Section 6 shows the experiments in the two robot

platforms, and presents a comparison between our approach and a very effective reactive strategy. Finally, Sec. 7 is devoted to draw some conclusions and future work.

2. Previous work

Recent work has already tackled the problem of surface cleaning as in Kormushev et al. (2011) and Sato et al. (2011), where robot skills to clean a whiteboard are presented. The robot is trained using imitation learning with hybrid position/force control to learn and execute trajectories while maintaining the force of the hand against the whiteboard. Gams et al. (2010) have also used force feedback to learn dynamic motion primitives that ensure that the robot maintains contact and applies the desired force in tasks such as wiping a table. Moreover, Nemec and Ude (2012) have proposed methodologies for sequencing motion primitives, allowing to perform more complex actions. However, in all these works, the surface cleaning strategy is fixed and thus the robot is unable to adapt to different distributions of dirt, some of which could be cleaned more efficiently with simpler trajectories.

For this, a perception system is necessary to analyze the scene and select actions accordingly. Bormann et al. (2013) developed an autonomous dirt detection system, but they do not tackle the problem of cleaning. Hess et al. (2011) formulate a table cleaning problem as a Markov Decision Process (MDP). The table surface is divided into cells, and a robot displaces a vacuum cleaner to the dirty cells. Interestingly, the initial perception can mark background textured points as dirty, and the algorithm learns to separate dirt from background, as it assumes that the vacuum cleaner removes all the dirt from a visited cell. The discount factor in the MDP is set to 0, converting planning into a 1-step lookahead problem. Hess et al. (2013) provide a more complete approach. They use a grid with Poisson processes to estimate the dirt and apply a TSP solver to generate the path used to clean the surface. These approaches work well when actions are accurate and grouping actions, that change the arrangement of dirt, are not considered.

Planning with actions that modify the configuration of the scene is challenging. Cambon et al. (2009) propose to link the symbolic description of a task with the geometric effects of the manipulation. They develop the aSyMov planner, and show the benefits of using actions to place the robot in better positions for object manipulation and to modify the environment by displacing objects. Unfortunately, aSyMov does not consider uncertainty issues, which are important in realistic environments.

Planning has also been used in manipulation domains with uncertain perceptions and stochastic actions (Kaelbling and Lozano-Pérez, 2012). Although stochastic actions are considered, planning takes into account only the most likely effect for every action, and re-planning is triggered whenever that effect fails to occur. In contrast, we want to take into account all possible effects when planning, as they are needed to obtain near optimal plans.

In probabilistic symbolic planning, one option is to *determinize* the problem and solve it using a deterministic planner, like FF-Replan (Yoon et al., 2007), that uses FF (Hoffmann and Nebel, 2001) as such deterministic planner. Since probabilities are ignored when *determinizing*, it is accepted that this option is not appropriate for domains in which the probabilities are important to get good plans (Little and Thiebaux, 2007). In our problem, cleaning actions may lead to diverse results, and although one action

may be the best for a given state, a different one may have a better overall performance in the long term.

MDPs are used to solve probabilistic symbolic planning, for which both exact and approximate solvers are available. When tackling large state spaces, a very common technique is UCT (Kocsis and Szepesvári, 2006), which finds near-optimal solutions in finite-horizon or discounted MDPs.

Lang and Toussaint (2010) propose a probabilistic planner that handles uncertainty by converting the rules into a dynamic Bayesian network for state representation, and predicts the effects of action sequences by using an approximate inference method. PRADA is an online procedure that is able to get a plan for every state, and has a better performance than classic UCT. Moreover it can cope with noise, which is usually present in robot actions, making it an appropriate planner for our task.

Finally, preliminary work on the proposed system focusing on the learning part can be found in Martínez et al. (2013). The current paper provides a more detailed and comprehensive description of the system, addresses the issues that appear when using probabilistic planning, and includes further experimentation and results.

3. Proposed approach

The method proposed in this paper is aimed at cleaning a surface using a calibrated RGB-D camera, a robot arm grasping a cloth, and an optional secondary robot arm holding a dustpan.

The RGB-D camera provides depth and colour data, which is used to extract information about the environment and create a symbolic state representing it.

The robot has a set of actions consisting of sequences of movements to clean or displace dirt. Every action is represented by at least one rule, which encodes the expected effects when a set of preconditions holds.

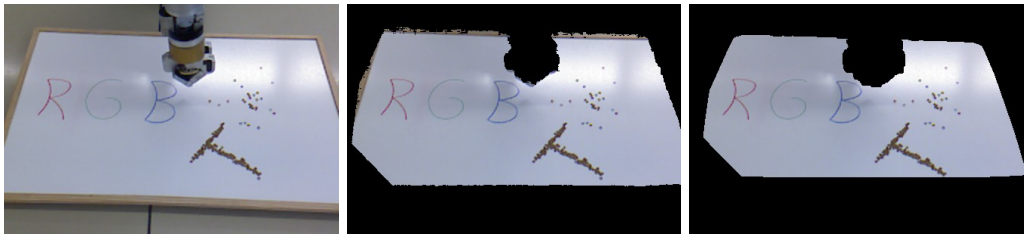
Given a symbolic state and a set of rules defining the available actions, the planner chooses a sequence of actions to clean the surface efficiently, which is then executed by the robot. Once the robot has finished, it will plan again with an updated state and execute new actions until all dirt is cleaned.

The actions generated by the planner have to be converted into motions. Depending on the action, a different type of movement will be created based on the dirty areas that it acts upon. The depth information from the camera is used to obtain the 3D positions corresponding to the dirty areas, which are then used to generate the 3D points where the robot will move to.

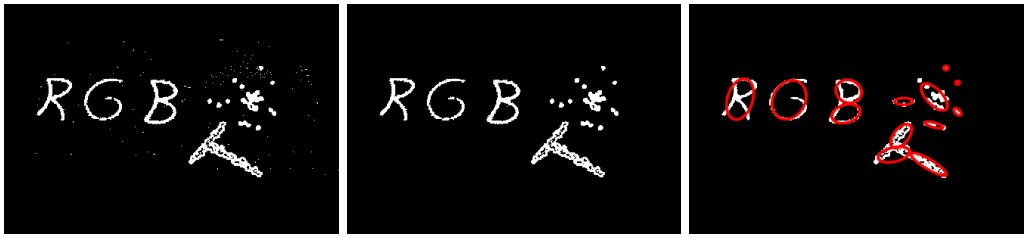
3.1. Perception

RGB-D cameras obtain observations containing both depth and colour information. Depth is used to segment the surface to be cleaned, while colour permits segmenting the dirty areas on the surface. Finally, to simplify the state, those dirty areas are represented by ellipses. The perception system is designed to be effective with different types of dirt, such as small particles like lentils or ink on a board. Figure 2 shows an example of the perception process, which performs the following steps:

- *Surface segmentation*: Depth information is used to segment the dirty surface, which may be located in any position and orientation. For simplicity the surface



(a) Original image (b) Surface segmentation (c) Edges removal



(d) Background subtraction (e) Noise removal (f) Ellipses representation

Figure 2: Perception process.

is assumed to be planar, allowing us to detect it by using the RANSAC algorithm. Once detected, all points lying outside the plane will be removed (see Fig. 2b).

- *Edges removal*: RGB-D cameras are not accurate in the colour and depth registration. This is crucial in the edges of a 3D object, where the colour given to a point may be the colour of a different object in its neighbourhood and detected as dirt (see Fig. 2c). Edges are therefore removed.

- *Background subtraction*: The image is divided in different areas through region growing using relative gradients. Areas bigger than a certain threshold are considered to be background and thus removed. As a result, the remaining areas will be the dirty ones (see Fig. 2d).

- *Noise removal*: A median filter is applied to remove spurious points produced by the background segmentation. Moreover, a belief state is maintained for every pixel, requiring a pixel to be perceived as dirty several times before considering it as actually dirty (see Fig. 2e).

- *Ellipse representation:* Planners cannot cope with large state spaces, a small state is recommended to plan efficiently. Therefore, a new representation based on ellipses that fit the dirty areas is created. These ellipses provide a compact representation of the dirty areas shape and size (see Fig. 2f).

Perception modules are initialized using a clean surface: the different parameters are adjusted to the minimum values for which the entire surface gets segmented as clean. Note that the subtraction thresholds would need to be adjusted again if either the surface or the lighting changes significantly, while all other modules can use the same parameters for a wide range of surfaces.

This paper focuses on selecting good cleaning strategies, and thus the perception system presented here is just a tool to test the cleaning skills. The proposed method is robust when cleaning a uniform surface with constant illumination, which was the only case tested in the experiments.

3.2. Actions

A set of actions is designed to clean a surface containing small objects like lentils, as shown in Fig. 3. The cleaning tool is always oriented perpendicular to the direction of motion to get effective moves. The robot starts and finishes actions a few centimeters away from the surface, so it does not push any lentils in between two actions. After reaching the starting point, the tool approaches the surface, performs the action while maintaining contact with it, and finally moves away from the surface as a preparation for the following action.

Actions are parametrized with the ellipses representing the dirty areas. To define the ellipses we use their positions and axes lengths and orientations.

3.2.1. One-arm cleaning actions

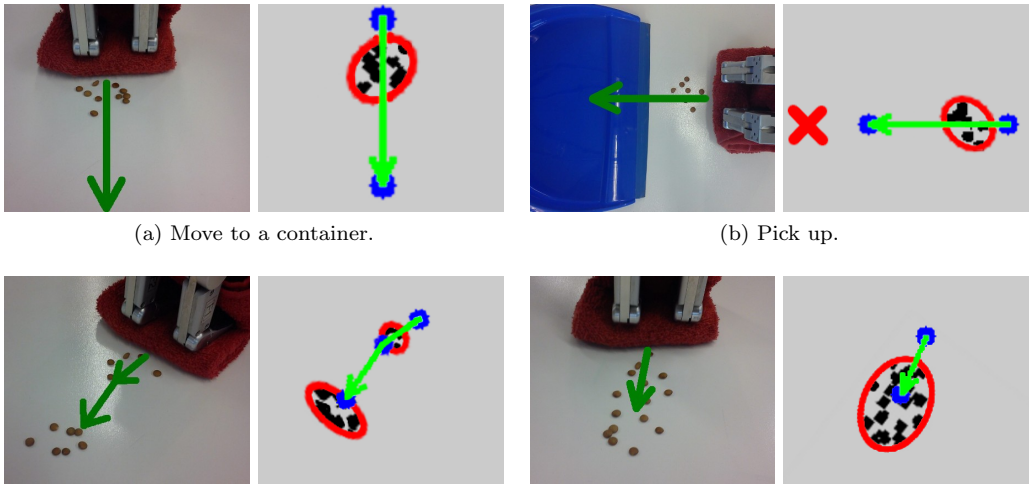
These actions move dirt to a container located close to an edge of the dirty surface. As they only have to push the dirt, one robotic arm is enough to execute them.

- `Fast move to container (ellipse)`:
  - *Starting point:* The point of the ellipse that is farthest from the container.
  - *Movement:* The robot moves the cloth to the container position, pushing the lentils towards it (Fig. 3a). It performs the shortest trajectory in joint space, which may not be completely straight.
- `Straight move to container (ellipse)`:
  - It is equivalent to *Fast move to container*, but uses path planning to ensure that the trajectory is straight. Although it is more precise, the addition of path planning makes it slower.

3.2.2. Two-arms cleaning actions

Having two arms allows the robot to execute pick up actions using a cloth and a dustpan. The arm with the cloth will push the dirt to the dustpan.

- `Pick up (ellipse)`: Assumes that the dustpan is grasped with the left hand, and the cloth with the right hand.
  - *Dustpan Starting Point:* Farthest point of the ellipse on the left.
  - *Cloth Starting Point:* Farthest point of the ellipse on the right.
  - *Movement:* The robot moves the cloth towards the dustpan position, pushing the lentils towards it (Fig. 3b).



(a) Move to a container.

(b) Pick up.

(c) Join 2 groups.

(d) Grouping scattered dirt.

Figure 3: Cleaning actions. On the left we show the movements of the robotic arm grabbing a cloth. On the right, the robot movements are shown in the segmented 2D images. The arm with the cloth moves along the green arrow. In (b), the arm places the dustpan on the red cross.

3.2.3. Grouping actions

These actions rearrange the dirty areas on the surface, so that they become easier to clean by means of future actions.

- `Join 2 groups(ellipse1, ellipse2)`: The movement pushes the dirt of ellipse1 to ellipse2 (Fig. 3c).
  - *Starting Point:* The point of ellipse1 that is farthest from ellipse2.
  - *Second Point:* The point of ellipse1 that is nearest to ellipse2.
  - *Ending Point:* The point of ellipse2 that is nearest to ellipse1.
- `Join 3 groups(ellipse1, ellipse2, ellipse3)`: Moves ellipse1 and ellipse2 to the position of ellipse3. The difference with executing twice the *Join 2 groups* action is that the robot arm continues smoothly the movement, not having to initialize the action again.
  - *Initial points:* The points of *Join 2 groups*.
  - *Continuation:* The point of ellipse2 that is nearest to ellipse3.
  - *Ending point:* The point of ellipse3 that is nearest to ellipse2.
- `Grouping scattered dirt(ellipse)`:
  - *Starting Point:* The ending point of the biggest axis of the ellipse.
  - *Movement:* The robot moves to the center of the ellipse, making a smaller and more manageable group of lentils in the process (Fig. 3d).

The important details to know about each action are its prerequisites, its effects and the time it takes to complete. These parameters can be obtained experimentally (Vaquero et al., 2013) or learned (Section 5). It is worth noting that all these actions are stochastic due to several factors:

Perception errors. Actions rely on the accuracy of the depth information provided by RGB-D cameras, which have a limited resolution. Although using a cloth provides some compliance, actions may fail to move the dirt as expected.

Manipulation errors. The same action may have different effects for similar dirty areas. For example, some lentils may spread during the trajectory in some cases, while they may move successfully in other similar situations.

Tools. Usual tools for cleaning, such us cloth, produce different results depending on the way they are grasped.

3.3. Planning

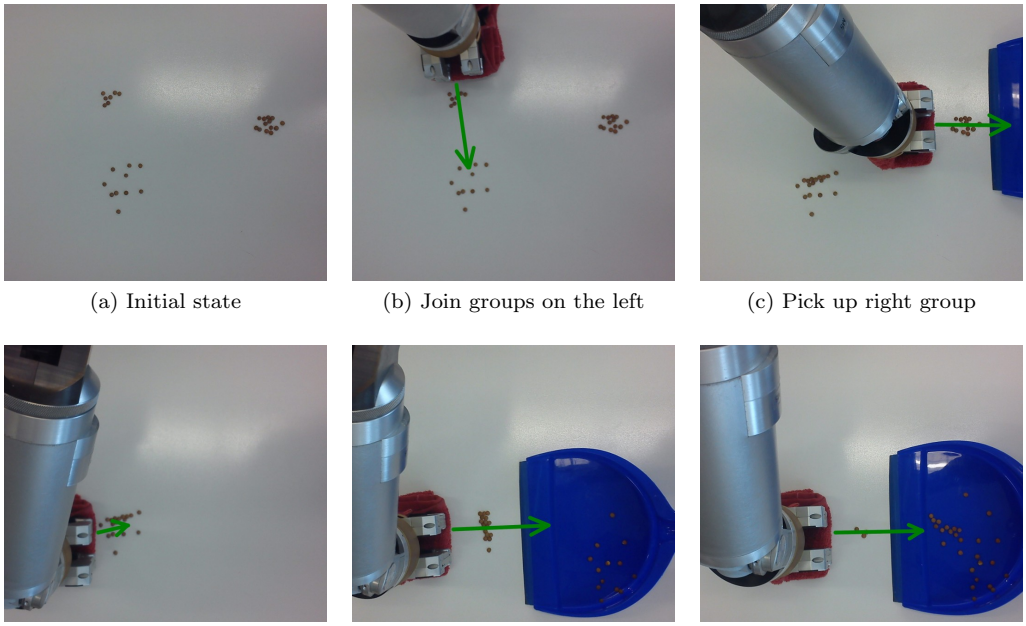
The planner selects the set of actions to execute based on the state and the rules. The task is quite complex, and selecting the fastest action sequence is challenging. For example, plans beginning with *grouping* actions may penalize in the beginning (remove no dirt) compared to *cleaning* actions, but they can provide the best results in the long run. Figure 4 shows a typical example of such behaviour. Moreover, actions are stochastic, which makes the system more complex.

The problem is formulated as a Markow Decision Process (MDP). A finite MDP is a five-tuple $\langle S, A, P, R, \gamma \rangle$ where

- S is a set of discrete states. States are obtained from perceptions.
- A is a set of actions that the robot can perform.
- $P(s'|s, a)$ is the transition function describing the probability of obtaining a successor state s' by executing an action a from a given state s. This transition function is obtained from a set of rules defining the expected results of the actions.
- $R : S \times A \to \mathbb{R}$ is the reward function.
- $\gamma \in [0, 1)$ is a discount factor measuring the degradation of future rewards.

The planner has to find a policy $\pi : S \to A$ that maximizes the value function $V^\pi(s)$ for a given state s. The value function is the sum of expected rewards: $V^\pi(s) = V(s, a|\pi) = E[\sum_t \alpha^t R(s_t, a_t)|s_0 = s, \pi]$.

The planner obtains a sequence of actions to maximize $V^\pi(s)$, and these actions are then converted into motions that the robot will perform to clean the dirty areas.



(a) Initial state (b) Join groups on the left (c) Pick up right group

(d) Group together scattered lentils (e) Pick up lentils (f) Pick up remaining lentils

Figure 4: Planning example. The plan begins by joining the groups on the left (b) so that the robot will be able to pick them up later with just one action. After that, it continues by picking up the group on the right (c). Seeing that the group on the left is scattered, the robot executes a grouping action (d) and finally picks up the lentils (e). Nevertheless, the last action failed to clean a few lentils, so another pick up action (f) is executed to finish.

3.3.1. State representing the scene

The state s is defined as

$$s = (d_1, ...d_n, near(d_i, d_j), ..., traversal(d_i, d_j, d_k), ...) \quad (1)$$

where d_i are the ellipses representing dirty areas, $near(d_i, d_j)$ are dirty areas whose positions are close to each other, and $traversal(d_i, d_j, d_k)$ indicates that d_k is positioned between d_i and d_j. The *traversal* predicate is used to know if actions would also push other lentils across their trajectories. With these predicates, the planner can select actions that clean aligned dirty areas with just one movement.

Each dirty area is defined as $d_i = (Id, s, \sigma)$

- Id: *Identifier*.
- s: *Size*, where $s \in \{big, medium, small\}$.
- σ: *Scattered*, where $\sigma \in \{true, false\}$ accounts for compact or scattered distributions.

Perceptions are acquired while the robot is cleaning, so there may be some occlusions in the scene. The planner considers the state to be fully observable, and we will tackle the problem of occlusions later in Section 4.2.3.

Action:
pickUp(X)
Preconditions:
dirt(X), mediumSize(X), ¬scattered(X)
Effects (Success probability: predicate changes):
0.4: ¬dirt(X), clean(X)
0.3: ¬mediumSize(X), smallSize(X)
0.2: ¬mediumSize(X), smallSize(X) sparse(X)
0.1: noise

Figure 5: NID rule example for picking up lentils.

3.3.2. Rules representing the actions

Cleaning actions are stochastic, i.e., they have different possible effects with associated probabilities. We model actions with noisy indeterministic deictic (NID) rules (Pasula et al., 2007) that define their preconditions and effects. The preconditions are a set of predicates that have to be satisfied for the rule to be applicable, and the effects are the list of predicates that change in the state with a certain probability when the rule is applied. The sum of the probabilities of the effects must be 1 so one effect is always applied. An example of a NID rule is shown in Fig 5.

A NID rule represents one action, while each action may be represented by several rules. All the rules defining one action have disjoint preconditions, and therefore, each state-action pair (s, a) is covered by just one rule r.

The rules can be obtained experimentally, executing each action a number of times until meaningful statistics are obtained to define the rules. However, the option of learning them autonomously is recommended, as it permits to adapt to possible changes in the robot or the environment. The learning method is described in Section 5.

4. Planning with uncertain actions

In this section we introduce the planner and some design considerations needed to tackle stochastic domains.

Using a probabilistic planner is important when actions have several possible effects with different probabilities. Deterministic planners and replanners only consider the most probable effect for each action, while a probabilistic planner takes into account all effects (Yoon et al., 2007). For example, consider that there are two scattered dirty areas and two cleaning actions.

- The first action cleans a scattered dirty area with probability 0.5 and a compact one with probability 0.8.
- The second action joins two dirty areas in a scattered group with probability 0.3, and joins them in a compact group with a probability 0.2.

The best plan joins the two groups and then cleans them. It has a $0.3 * 0.5(\mathit{scattered}) + 0.2 * 0.8(\mathit{compact}) = 0.31$ probability of cleaning both. In contrast, a deterministic planner would choose the first cleaning action twice as it considers just the best effect, and both groups would be cleaned with probability $0.5 * 0.5 = 0.25$.

4.1. Probabilistic planner

The system uses PRADA (Lang and Toussaint, 2010), a model-based planner for complex domains. This planner uses NID rules to plan action sequences that maximize the reward given a plan length.

Defining s^i as the state at time i, a^i as the action to be executed, and $\#dirt(s^i)$ as the number of dirty areas in state s^i, the sum of expected rewards $V(s, \overline{a})$ of a sequence of actions is obtained as follows:

$$V(s^0, a^{0:T-1}) = \sum_{t=0}^{T} \gamma^t P(\#dirt(s^t) - \#dirt(s^{t+1}) | a^{0:t-1}, s^0) \tag{2}$$

where $P(\#dirt(s^t) - \#dirt(s^{t+1}) | a^{0:t-1}, s^0)$ is the probability that a number of dirty areas are cleaned at time t.

The reward increases with the number of dirty areas cleaned, but also, as we want to clean as fast as possible, the discount factor γ makes the reward for cleaning an area decrease with the number of actions previously executed. This way, the best rewards will be obtained by plans that clean many areas fast, which are the best ones for our task.

4.2. Issues

Obtaining effective plans in real-world applications can be complex. The planner should take a limited amount of time while providing good results and adapting to all the contingencies that may appear.

4.2.1. Computational time

PRADA is a suboptimal planner, so the probability of finding good plans depends on the fraction of actions sampled. The maximum sampling number n should be proportional to the complexity of the problem, which depends on the number of dirty areas $\#dirt(s)$, the number of available actions $\#a$, and the plan length H, which is the maximum number of actions in the plan. The coverage of the state space is

$$coverage = \frac{n}{(\#dirt(s) \cdot \#a)^H} \tag{3}$$

and it should be at least significant enough to find good plans.

If the planner samples only a small portion of the state space there is a high probability that it will not find the best plans, but increasing the number of samples also increases the computational cost. For difficult problems two approaches can be taken to get good plans: increasing the number of samples, or reducing the problem complexity by ignoring some of the dirty areas.

It should be noted that it is not worth using a planner if more time is spent in planning than is saved by applying the plans. Therefore a maximum sampling number was set so the planner never spent more than a few seconds, and this number was reduced when few dirty areas were present on the surface.

4.2.2. Plan length

The planner maximizes the results obtained for a given plan length that has to be specified before planning. The plan length should be large enough to permit cleaning all the surface, but not so long that the last actions become useless while increasing the planning time. Good empirical results were obtained with a plan length of $1 + \#dirt(s)$, having one action for each dirty area and an extra action.

4.2.3. Replanning

Actions are stochastic, so plans will not always perform as expected. After a plan is executed, a new plan is generated based on the updated state containing only the remaining dirty areas. The robot will continue generating and executing new plans until the surface gets cleaned completely.

Moreover, to minimize cleaning time, the system is continuously taking perceptions in parallel while the robot cleans. The planner considers the state to be fully observable, but the robot arm may occlude some parts of the scene, so once the robot has cleaned the observable surface, it will take the arm out of the field of vision to check if there are remaining dirty areas.

4.2.4. Partial Plans

Executing only partial plans can also improve the results. Actions have many possible effects, which makes it hard to predict the state after executing a part of the plan. As later actions may be too uncertain, it is a good idea to execute only the first actions of a plan before generating a new plan with updated perceptions. Newer plans will be more precise and get better results. Experimental validation of this idea is presented in Section 6.

5. Learning

The planner uses a set of rules defining the actions that may be performed, and the quality of the plans will depend on the accuracy of these rules. However, it is difficult to define accurate rules for every surface, type of dirt, robot and grasping of cleaning tools. To solve this problem, the robot is initialized with a generic and inaccurate set of rules, and the actual effect probabilities are learned while performing the cleaning task. The learner updates the expected rule effects for every action that is executed to reflect the result of the execution. It also saves a record of previously executed actions and their results to get better estimates of the effects.

Following the principle of "optimism under uncertainty" (Brafman and Tennenholtz, 2003), the robot starts with an optimistic set of rules. This initial set of rules define the effects that the actions may yield. During the learning process, these rules will be updated until a set that represents the actual behaviour is obtained. After an action is executed, the robot takes a new perception to analyze the changes in the state, decide which rule effect happened, and update its probability. When no rule effect matches the result of the action, the learner considers it a rare effect and increases the noise effect probability instead.

5.1. Learning heuristics

We want the robot to rapidly refine the rule effects to adapt to new environments, but we also want to avoid wrong premature estimations to degrade its performance, as it is learning at the same time that it is solving the task.

Learning heuristics allow to obtain the rules modelling the task without requiring much computational time (Janssen and Fürnkranz, 2010). We are using a very fast heuristic, the decreasing-m-estimate to minimize the time taken to learn while preventing these premature wrong estimations. It is based on the m-estimate (Cestnik, 1990), which includes a parameter m to implement a trade-off between learned effects and a priori probabilities in rule effects

$$P = \frac{p + mP_0}{p + n + m}, \tag{4}$$

where P is the estimated probability, P_0 the a priori probability, p the number of positive examples and n the number of negative examples.

The problem with this heuristic is that small values of m may yield wrong estimates of rule effects, while a high value of m would entail the system taking too much time to converge to the learned estimates. We propose to use a different heuristic:

$$P = \frac{p + (m/\sqrt{p+n})P_0}{p + n + (m/\sqrt{p+n})}. \tag{5}$$

This decreasing-m-estimate is similar to the m-estimate when there are only a few examples, favouring a priori probabilities. But as the number of examples increases, their influence decreases, leading to better estimates that have little influence from a priori probabilities. The value of m should depend on the stochasticity of the task. In our experiments a value of 10 provided good estimates while being low enough to converge fast to the learned estimates.

5.2. Stop learning

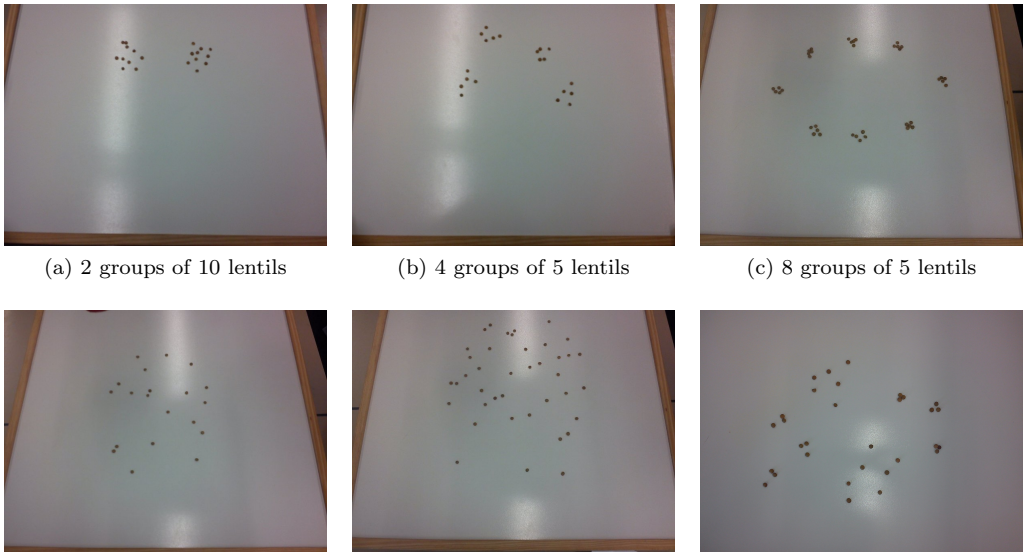
Although we are using a fast heuristic, learning adds some overhead to the system. After executing an action that is being learned, the arm has to leave the visual field of the camera to get a good perception of the surface and estimate the resulting effect correctly. Otherwise, planning would rely on partial perceptions that may have occlusions.

Therefore, we only learn actions until we have enough examples to consider that the learned estimate is quite accurate. Using the Hoeffding inequality, we can have a bound for having a high probability $(1 - \delta)$ that our estimate $\widehat{p}$ is accurate enough $|\widehat{p} - p| \leq \epsilon$. The number of trials T required for that bound is

$$T \geq \frac{1}{2\epsilon^2} \ln \frac{2}{\delta}. \tag{6}$$

6. Experimental results

The experiments were performed on a surface with many lentils arranged as seen in Fig. 6. The simpler layouts had just two groups of lentils, while more difficult ones had up to 40 lentils scattered all over the surface. The core of the experiments was carried



(a) 2 groups of 10 lentils (b) 4 groups of 5 lentils (c) 8 groups of 5 lentils

(d) 20 lentils scattered (e) 40 lentils scattered (f) 30 lentils scattered + groups

Figure 6: Experiments setup.

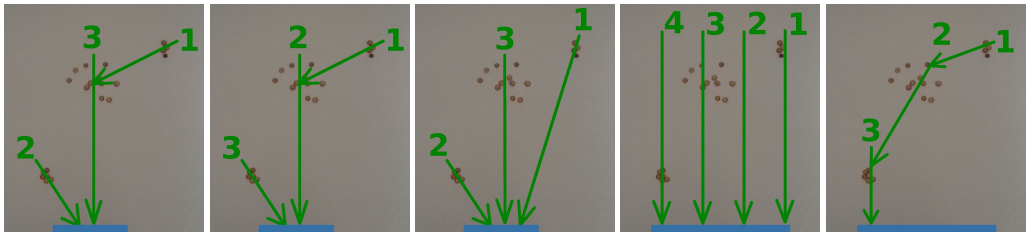
out using a WAM arm and a RGB-D camera attached to the environment (the hand-eye calibration parameters were obtained using standard methods). The REEM robot has been used to validate the results on a different robotic platform, but no systematic experiments were done that permit to obtain meaningful statistics about its performance. Videos of the experiments, including also the cleaning of a whiteboard, can be found at `http://www.iri.upc.edu/groups/perception/surfaceCleaning`.

Two different cleaning skills are validated here. The first one consists in moving the lentils with a cloth to a container positioned near an edge of the surface, and the second one consists in picking up lentils with the help of a dustpan. The γ parameter of the planner was set to 0.95 which led to good results in our tests. The sampling number n was set taking into account the most difficult scenarios, and a value of 10^4 was enough to obtain good results while spending at most ~ 4 seconds planning.

To the best of our knowledge, there are not approaches directly comparable to ours, because each tackles a slightly different cleaning problem. The closest to our purposes is that of Hess et al. (2013), which nevertheless uses an accurate vacuum cleaner instead of an inaccurate cleaning tool such as our rag. We compared different configurations of our approach with a reactive method, a fixed program and the approach by Hess et al. (2013), whose examples are shown in Fig.7.

- Planning and executing 1, 2 or 3 actions before replanning. Using 4 or more actions led to worse results as estimating the layout of the lentils becomes very difficult after a few actions.

- A reactive strategy that uses the segmentation into groups of lentils. The robot starts cleaning the dirty area that is farthest from the goal, and continues cleaning the dirty areas that are closest to the arm position.



(a) Plan, 1 action (b) Plan, 2 actions (c) Reactive (d) Complete wipe (e) Hess et al.

Figure 7: Examples of executions with different strategies in the task of moving lentils to a container. **Plan and execute 1 action:** the robot joins the two upper groups together. Then it replans while having the arm occluding the upper group, so it chooses to clean the dirty area on the bottom left. Finally it replans and cleans the upper group. **Planning and executing 2 actions:** The robot joins the two upper groups and cleans them. Then it replans and cleans the last group. **Reactive:** Cleans one group after another without planning. **Complete wipe:** The robot wipes in straight lines to clean the bounding box of the dirty areas without planning. **Hess et al. (2013):** The robot groups all dirty areas following the shortest path, and finally cleans them.

- A fixed program to wipe the bounding box of the dirty areas using a zig-zag like motion. The robot wipes in straight lines from right to left across the table.
- Hess et al. (2013) cleaning skills using a rag instead of a vacuum cleaner. This approach uses a TSP planner to obtain the shortest path that passes through all dirty areas (grouping them in a single dirty area) and finally the dirt is cleaned. This policy yielded good results when using a vacuum cleaner (Hess et al., 2013), but using a rag is more challenging since grouping all dirt together and then cleaning it involves a lot of uncertainty.

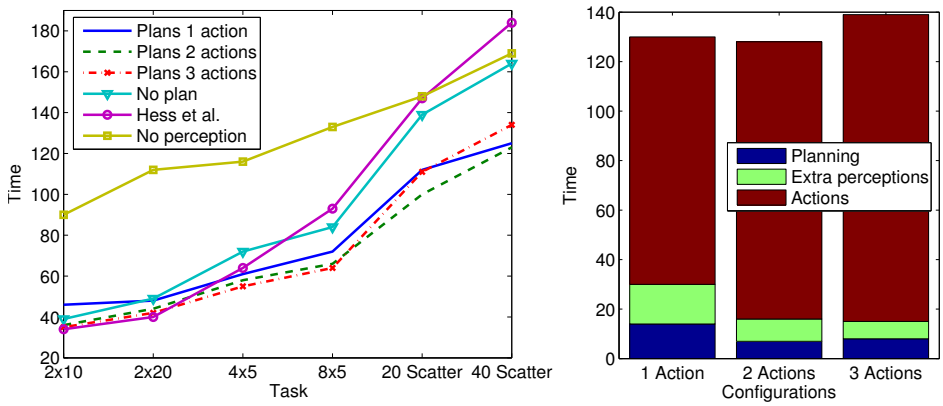
Finally, an experiment to assess the robot learning skills was performed. The robot learned using the layout in Fig. 6f where both grouped and scattered lentils were present.

6.1. Moving lentils to a container

This task consisted in dragging the lentils laying on a table to a container positioned near the edge of the table. The robotic arm used a cloth to push the lentils to the container.

In Figure 8a we show the results obtained when cleaning the different layouts of lentils shown in Fig. 6. The proposed approach was applied until all lentils were cleaned. In this experiment, planning gives similar or sightly better results in simpler tasks, and large improvements in difficult tasks. The reactive method scales very bad, as it cleans dirty areas one by one without grouping them. Hess et al. (2013) approach was devised to use an accurate vacuum cleaner, but adapts badly to unexpected problems that arise when using uncertain actions such as moving dirt with a rag. In contrast, planning adapts to failed actions, and avoids joining groups that are too big and far from each other. When planning, overall, executing two actions before re-planning usually yields the best results. A lot of re-planning is required when executing just one action, while the prediction of the state becomes too poor when using more than two actions in difficult problems.

Figure 8b shows the time spent in experiments entailing moving 40 scattered lentils to a container, which was the most difficult problem instance. Planning and perception



(a) Cleaning lentils (b) Cleaning 40 lentils

Figure 8: (a) Skill of moving lentils to a container. The results shown are the mean over 5 runs. (b) Time for the case of moving 40 scattered lentils to a container. Extra perceptions include the movements and perceptions needed to tackle occlusions by taking the arm out of the field of view (see Section 4.2.3).

time (which includes the time moving the arm away from the scene to tackle possible occlusions) constitute only a small overhead compared to cleaning actions. Perception time is also smaller when executing longer sequences, as the robot tries to clean more dirty areas before using newer perceptions.

6.2. Picking up lentils

In this experiment lentils are picked up from a table using two robotic arms grabbing a cloth and a dustpan. The arm with the cloth performs the grouping actions and also pushes lentils into the dustpan, while the arm with the dustpan just has to place it near the lentils that are going to be picked up.

In Figure 9 we show the results obtained when picking the different layouts of lentils. The robot kept cleaning until all lentils were picked up. In the simplest cases of cleaning

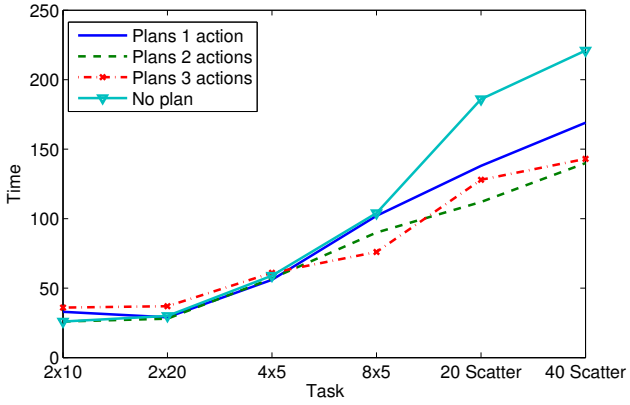


Figure 9: Skill of picking up lentils. The results shown are the mean over 3 runs.

two groups, the planner gives similar results to the reactive method, but as the complexity of the problem increases, it can be seen that using a planner improves the results significantly. The planner takes advantage of joining lentils in a few compact groups before picking them up. The action to pick up lentils is the slowest, so decreasing the number of times it has to be executed leads to better results. In general, executing 2 actions before replanning provides the best results, but adding a third action slightly enhances the results in some situations.

6.3. Learning

We have carried out two experiments to analyze the learner performance in the task of surface cleaning. Both experiments involved cleaning a surface with 30 lentils in small groups and spread over it as seen in Fig. 6f. The robot had to move the lentils to a container positioned near an edge of the surface. The task was repeated a number of episodes to analyze the learning process. The value of the m parameter was set to 10, and the number of action executions to consider an action as known T to 12. We measured the number of actions and time taken to clean with new grasps, which are shown in Fig. 10. As can be seen, the number of actions required to complete the tasks decrease as the rules improve. Also, the learner stops refining actions once it gets enough examples of them, reducing the learning time after a few iterations.

Finally, the same experiment was repeated using the original m-estimate to compare its performance with our proposal. Although the m-estimate also improves the rules, the decreasing m-estimate obtains better results with fewer iterations as shown in Fig. 10c.

7. Conclusions

In this work we have shown the use of a planner in a real robotic system, and the improvements that it provides over a reactive strategy and a fixed program. The planner was used in the task of cleaning surfaces with a set-up consisting of a RGB-D camera and robot arms without force control.

Real world tasks are usually stochastic and quite complex. Thus, selecting sequences of actions is very difficult in this kind of domains. Using a planner has been very useful to get good solutions to accomplish these tasks faster. The planner takes into account the different probabilities of the action effects and obtains good plans to optimize results. In simple scenarios the presented method performs similar to reactive strategies, but in complex scenarios the ability to plan proves to be crucial to improve results.

As we can tackle stochastic actions, we can also decrease the number of constraints of the system. In this work we have performed actions that maintain contact using a RGB-D camera instead of the common approach of force control. The depth information of RGB-D cameras and the compliance of using a cloth to clean, gives good results, but still adds some uncertainty in the actions. This extra uncertainty can be handled as we are planning with stochastic actions, and thus the constraint of having force sensors can be removed, allowing us to apply the method to the robot REEM.

Finally, a learning method is also integrated to adapt to different types of grasps, robots or surfaces, making the system more versatile. In our case, learning allows to use the same system after changing the cloth grasp or the robot used.

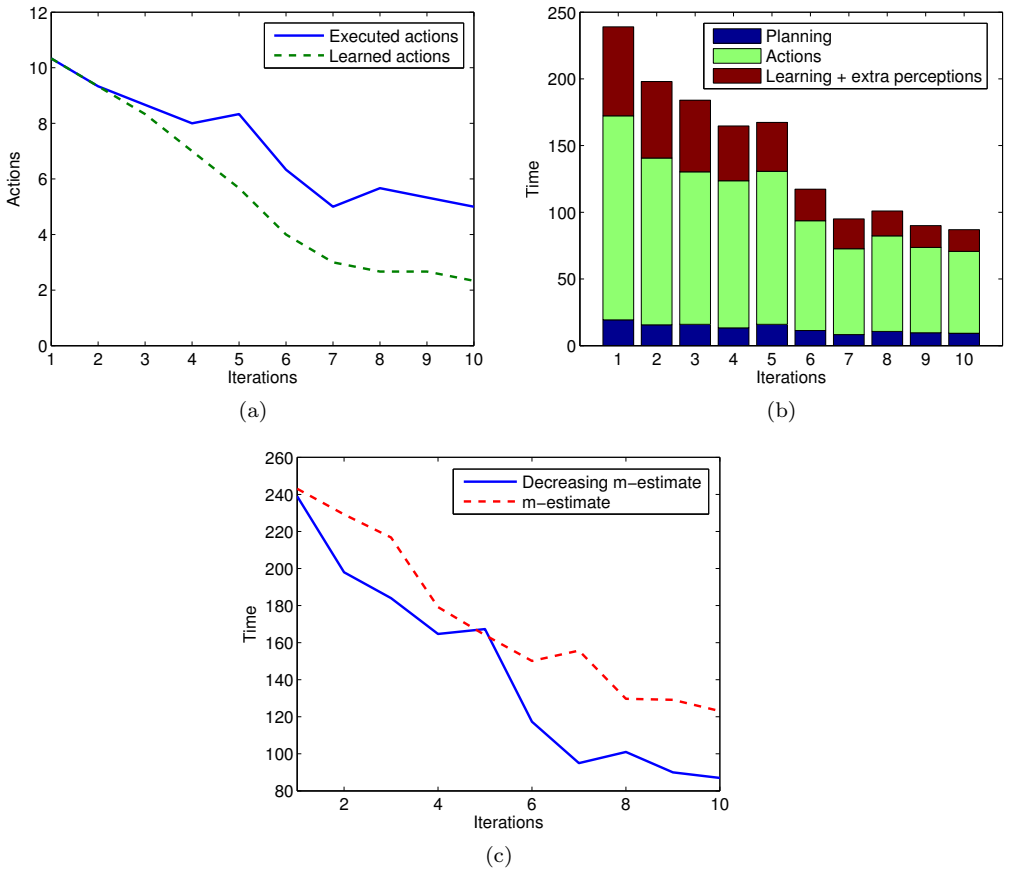


Figure 10: Improvements using the learner. **(a)** Number of actions executed and the number of them that required learning as they were considered unknown. **(b)** Distribution of time between planning, action execution and learning. **(c)** Time taken to clean the board using the proposed decreasing m-estimate and the original m-estimate.

7.1. Future work

An open line of research is learning the full action rules while performing the task. Having many records from executions, a learning optimization algorithm (Pasula et al., 2007) may be used to learn the rules while taking into account all preconditions and effects. This learner can be integrated within a reinforcement learning algorithm to have an autonomous learning robot (Martínez et al., 2014).

Acknowledgements

The authors would like to thank Ricardo Tellez and all the PAL Robotics staff for their help in programming and performing the experiments with the REEM robot.

This work was supported by EU Project IntellAct FP7-ICT2009-6-269959 and by the Spanish Ministry of Science and Innovation under project PAU+ DPI2011-27510. D.

Martínez is also supported by the Spanish Ministry of Education, Culture and Sport via a FPU doctoral grant (FPU12-04173).

References

Bormann, R., Weisshardt, F., Arbeiter, G., Fischer, J., 2013. Autonomous dirt detection for cleaning in office environments. In: Proc. of Int. Conf. on Robotics and Automation. pp. 1260–1267.

Brafman, R. I., Tennenholtz, M., 2003. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. Journal of Machine Learning Research 3, 213–231.

Cambon, S., Alami, R., Gravot, F., 2009. A hybrid approach to intricate motion, manipulation and task planning. Int. Journal of Robotic Research 28 (1), 104–126.

Cestnik, B., 1990. Estimating probabilities: A crucial task in machine learning. In: Proc. of European Conference on Artificial Intelligence. pp. 147–149.

Gams, A., Do, M., Ude, A., Asfour, T., Dillmann, R., 2010. On-line periodic movement and force-profile learning for adaptation to new surfaces. In: Proc. of Int. Conf. on Humanoid Robots. pp. 560–565.

Hess, J., Beinhofer, M., Kuhner, D., Ruchti, P., Burgard, W., 2013. Poisson-driven dirt maps for efficient robot cleaning. In: Proc. of Int. Conf. on Robotics and Automation. pp. 2245–2250.

Hess, J., Sturm, J., Burgard, W., May 2011. Learning the state transition model to efficiently clean surfaces with mobile manipulation robots. In: Proc. of ICRA Workshop on Manipulation under Uncertainty.

Hoffmann, J., Nebel, B., 2001. The FF planning system: Fast plan generation through heuristic search. Journal of Artificial Intelligence Research 14, 253–302.

Janssen, F., Fürnkranz, J., 2010. On the quest for optimal rule learning heuristics. Machine Learning 78 (3), 343–379.

Kaelbling, L. P., Lozano-Pérez, T., 2012. Unifying perception, estimation and action for mobile manipulation via belief space planning. In: Proc. of Int. Conf. on Robotics and Automation. pp. 2952–2959.

Kocsis, L., Szepesvári, C., 2006. Bandit based Monte-Carlo planning. In: Proc. of European Conference on Machine Learning. pp. 282–293.

Kormushev, P., Nenchev, D. N., Calinon, S., Caldwell, D. G., 2011. Upper-body kinesthetic teaching of a free-standing humanoid robot. In: Proc. of Int. Conf. on Robotics and Automation. pp. 3970–3975.

Lang, T., Toussaint, M., 2010. Planning with noisy probabilistic relational rules. Journal of Artificial Intelligence Research 39, 1–49.

Little, I., Thiebaux, S., 2007. Probabilistic planning vs. replanning. In: Proc. of ICAPS Workshop on IPC: Past, Present and Future.

Martínez, D., Alenyà, G., Jiménez, P., Torras, C., Rossmann, J., Wantia, N., Aksoy, E. E., Haller, S., Piater, J., 2014. Active learning of manipulation sequences. In: Proc. of Int. Conf. on Robotics and Automation. pp. 5671–5678.

Martínez, D., Alenyà, G., Torras, C., 2013. Planning surface cleaning tasks by learning uncertain drag actions outcomes. In: Proc. of ICAPS Workshop on Planning and Robotics. pp. 106–111.

Nemec, B., Ude, A., 2012. Action sequencing using dynamic movement primitives. Robotica 30 (5), 837.

Pasula, H. M., Zettlemoyer, L. S., Kaelbling, L. P., 2007. Learning symbolic models of stochastic domains. Journal of Artificial Intelligence Research 29 (1), 309–352.

Ramisa, A., Alenyà, G., Moreno-Noguer, F., Torras, C., 2014. Learning rgb-d descriptors of garment parts for informed robot grasping. Engineering Applications of Artificial Intelligence 35, 246–258.

Sato, F., Nishii, T., Takahashi, J., Yoshida, Y., Mitsuhashi, M., Nenchev, D., 2011. Experimental evaluation of a trajectory/force tracking controller for a humanoid robot cleaning a vertical surface. In: Proc. of Int. Conf. on Intelligent Robots and Systems. pp. 3179–3184.

Vaquero, T. S., Silva, J. R., Beck, J. C., 2013. Post-design analysis for building and refining ai planning systems. Engineering Applications of Artificial Intelligence 26 (8), 1967–1979.

Yoon, S. W., Fern, A., Givan, R., 2007. Ff-replan: A baseline for probabilistic planning. In: Proc. of Int. Conf. on Automated Planning and Scheduling. Vol. 7. pp. 352–359.