sensors

MDPI

Article

Sorting Objects from a Conveyor Belt Using POMDPs with Multiple-Object Observations and Information-Gain Rewards †

Ady-Daniel Mezei *, Levente Tamás and Lucian Buşoniu

Department of Automation, Technical University of Cluj-Napoca, Str. George Barițiu, Nr. 26-28, 400027 Cluj-Napoca, Romania; Levente.Tamas@aut.utcluj.ro (L.T.); Lucian.Busoniu@aut.utcluj.ro (L.B.)

* Correspondence: Daniel.Mezei@aut.utcluj.ro; Tel.: +40-0264-401-587

† This paper is an extended version of our paper published in Mezei, A.; Tamás, L.; Busoniu, L. Sorting objects from a conveyor belt using active perception with a POMDP model. In Proceedings of the 18th European Control Conference, Naples, Italy, 25–28 June 2019; pp. 2466–2471.

Received: 4 March 2020; Accepted: 22 April 2020; Published: 27 April 2020

Abstract: We consider a robot that must sort objects transported by a conveyor belt into different classes. Multiple observations must be performed before taking a decision on the class of each object, because the imperfect sensing sometimes detects the incorrect object class. The objective is to sort the sequence of objects in a minimal number of observation and decision steps. We describe this task in the framework of partially observable Markov decision processes, and we propose a reward function that explicitly takes into account the information gain of the viewpoint selection actions applied. The DESPOT algorithm is applied to solve the problem, automatically obtaining a sequence of observation viewpoints and class decision actions. Observations are made either only for the object on the first position of the conveyor belt or for multiple adjacent positions at once. The performance of the single- and multiple-position variants is compared, and the impact of including the information gain is analyzed. Real-life experiments with a Baxter robot and an industrial conveyor belt are provided.

Keywords: robotics; active perception; POMDP; information-gain rewards

1. Introduction

Robots in open environments, such as those that arise in the Industry 4.0 paradigm, collaborative, or domestic robotics [1–4], are affected by significant uncertainty in perceiving their environment. A way to handle this is to use active perception [5–9], which closes the loop between the sensing and control modules of the robot: control actions are chosen to maximise the information acquired from the sensor and thereby reduce uncertainty. Two leading paradigms in active perception are passive and active detection. In passive detection, the viewpoint is changed while leaving the state of the objects unaltered [10,11]. In active detection, the objects are also manipulated to improve performance [12].

We consider here the following active perception problem, which uses passive detection and is relevant for Industry 4.0. A robot working in a factory has the task of sorting differently shaped objects that are transported on a conveyor belt. The object classification is carried out from 3D scans. The conveyor belt is scanned from a set of poses (viewpoints), and the robot is able to move between the viewpoints. Due to an imperfect sensor and classification algorithm, multiple scans from various viewpoints are required to gain more information about the object so as to achieve a good classification.

In a similar problem, Patten et al. [10] propose a solution that uses RGB-D (Red, Green, Blue and Depth) data to detect objects from a cluttered environment. That solution can handle multiple objects at once by maintaining class and pose information in an occupancy grid. The occlusions are treated by choosing in a greedy manner from a set of viewpoints the one that is the most informative about the scene.

Cowley [13] considers a robot that uses 3D information to detect objects travelling on a conveyor belt in order further manipulate them. In [7,14], a 3D sensor is moved between viewpoints using a model that includes the class and pose of the objects. Compared to [10], our upcoming solution is similar to their choice of the next best action based on class and pose information, with the difference that in our case the action is chosen taking into account a longer horizon. Differently from our method, the solution in [13] does not handle uncertainty associated with the objects' detection. Compared to [7,14], a key difference in our scenario is the conveyor-belt structure, which allows us to propagate information and reduce the number of steps needed to take the decisions, as detailed later.

In order to achieve these advances over the state of the art, as a main contribution of this paper, we formalize and solve the sorting task as a partially observable Markov decision process (POMDP) that combines the deterministic, fully-observable motion dynamics of the robot between viewpoints with the stochastic, imperfect-sensor class observations. The specific structure of the task (advancing conveyor belt) will allow us to adapt the model and solution algorithm in some particular ways described below, and to provide some theoretical insight—which are additional contributions. In the POMDP model, object classes belong to the true, underlying state signal, and information about them is obtained via uncertain class observations, through an experimentally identified sensor model. At each step, the robot may either choose a new viewpoint from which to observe the objects, or decide on the object class. The tradeoff between all these options is solved automatically, via the reward function of the POMDP, which initially includes a correct-decision reward, an incorrect-decision penalty, and a step cost. A state-of-the-art planner called DESPOT (Determinized Sparse Partially Observable Tree) [15] is used to obtain a long-horizon sequence of actions that (near-optimally) maximize the expected cumulative reward.

In a first version of the technique, rewards only assess the quality of sorting decisions, as already explained. However, the belief tree structure exploited by DESPOT allows us to propose a second, key contribution. The algorithm is modified so that the rewards contain the information gain achieved by refining the belief as a result of each sensor movement action. The idea is to directly reward actions that better disambiguate between object classes, with the goal of reducing the time required to sort the objects. The related approaches in [16,17] provide a way to compute rewards that behave similarly to the information gain, but can be expressed in terms of the original POMDP reward function, and thereby facilitate solving the POMDP. The technique is used to balance exploration with exploitation in a networked robot system, where fixed and mobile robots have to understand the environment. In our case, the structure of the belief tree used in the DESPOT planner allows us to directly employ the information gain. Mafi et al. [18] also propose adding a so-called "intrinsic" reward based on the entropy to POMDPs, and apply the method in a market scenario. The solution is found in [18] with a reinforcement learning strategy, whereas here we use DESPOT and integrate it tightly with the information-gain rewards.

Another key feature of our method is that it permits the robot to observe from one scan objects at multiple positions on the conveyor belt. This allows the accumulation of information about upcoming objects on the belt, which is then propagated when the conveyor belt advances. Thus, the robot already has an informative estimate of the new class at the end of the belt, instead of starting from scratch. The goal is again to reduce sorting time without sacrificing accuracy.

For a simple case, we provide some insight into the structure of the tree explored in the active perception problem, and we empirically determine the branching factor of DESPOT. Both have implications on complexity.

To evaluate our method, we consider a specific scenario involving a Baxter robot equipped with an Asus Xtion 3D sensor mounted on one of its wrists. The robot must sort differently shaped light bulbs that travel on a conveyor belt, via classification from point clouds acquired by the 3D sensor. We start with a batch of simulation results, in which we evaluate both classification accuracy and the number of steps required to sort a given number of light bulbs. We first compare the variant where only one position on the belt is observed, with a second variant in which two adjacent positions are seen, studying the impact of a key tuning parameter of the algorithm: the incorrect classification

penalty. For the single-position version, we also compare to a simple baseline that alternates between two nearly opposite viewpoints. Then, we analyze the impact of introducing the information gain, and of its weight in the reward function, for both the single- and two- position variants. We close with a real-life batch of experiments in which we evaluate whether the advantages of observing two positions are maintained.

A preliminary version of this work was published in [19]. Compared to that work, the key methodological contributions here are including rewards based on the information gain; and providing insight into the complexity of the problem. Moreover, the comparison between observing single and multiple positions has been extended: e.g., we now use four classes of objects compared to two in [19], and report confidence intervals. Some parts of the approach are presented in more detail here, including the hardware and software setup in Section 2.4 and DESPOT in Section 2.2.

The outline of the paper is the following. Section 2 gives our methodology, as follows: Section 2.1 gives the mathematical formalism related to POMDPs; Section 2.2 explains our approach to adding information-gain rewards; some insight on complexity is provided in Section 2.3; and the hardware and software setup is described in Section 2.4. The results are presented and discussed in Section 3, as follows: for single versus multiple observations in Section 3.1; for the information-gain rewards in Section 3.2; and for the real robot in Section 3.3. Section 4 concludes the paper.

2. Methodology

2.1. POMDP Model of the Sorting Task

Here, we explain the basic PODMP model of the sorting task, initially presented in [19]. In general, a partially observable Markov decision process (POMDP) is a tuple $(S, A, T, R, Z, O, \gamma)$ [20], where the elements are the following:

- S is a set of states, taken discrete and finite for classical PODMPs. Individual states are denoted by $s \in S$.
- A is a set of actions available to the robot, again discrete and finite. Actions are $a \in A$.
- $T : S \times A \times S \to [0, 1]$ is a stochastic state transition function. Each function value $T(s, a, s')$ gives the probability that the next state is s' after executing action a in current state s.
- $R : S \times A \to \mathbb{R}$ is a reward function, where $r = R(s, a)$ is the reward obtained by executing a in s. Note that sometimes rewards may also depend on the next state; in that case, $R(s, a)$ is the expectation taken over the value of the next state. Moreover, rewards are classically assumed to be bounded.
- Z is a set of observations, discrete and finite. The robot does not have access to the underlying state s. Instead, it observes the state through imperfect sensors, which read an observation $z \in Z$ at each step.
- $O : S \times A \times Z \to [0, 1]$ is a stochastic observation function, which defines how observations are seen as a function of the underlying states and actions. Specifically, $O(s', a, z)$ is the probability of observing value z when reaching state s' after executing action a.
- $\gamma \in [0, 1)$ is a discount factor.

The objective in the POMDP is to maximize the expected sum of discounted rewards along the trajectory.

A central concept in POMDPs is the belief state, which summarizes information about the underlying state s, as gleaned from the sequence of actions and observations known so far. The robot is uncertain about s, so the belief state is a probability distribution over S, $b \in [0, 1]^{|S|}$, where $|S|$ denotes the cardinality of S. The belief state is initially chosen equal to b_0 (uniform if no prior information is available), and then updated based on the actions a and observations z with the following formula:

$$b'(s') = \frac{O(s', a, z)}{P(z|s, a)} \sum_s T(s, a, s') b(s) \tag{1}$$

Here, $P(z|s,a)$ is a normalization factor, equal to the probability of observing z when a is executed in s; this can be easily computed from O.

Now that the general POMDP concepts are in place, we are ready to describe the sorting task. There are two main components to this task: a deterministic, fully-observable component relating to robot motion among the viewpoints, and a stochastic, partially-observable component relating to the object classes. The two components run largely in parallel, and they are connected mainly through the rewards for the class-decision actions of the robot. We first present the motion component, as it is rather simple, and then turn our attention to the more interesting, class-observation component.

The motion component is defined as follows:

- Motion state $p \in P = \{p_1, p_2, p_3, \ldots, p_K\}$, meaning simply the viewpoint of the robot. There are K such viewpoints.
- Motion action $m \in M \in \{m_1, m_2, \ldots, m_K\}$, meaning the choice of next viewpoint.
- Motion transition function:

$$T_p(p, m, p') = \begin{cases} 1 & \text{if } p' = p_i \text{ and } m = m_i \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

which simply says that the robot always moves deterministically to the chosen viewpoint.

Motion states are fully observable, so we can define observations $z_p \in Z_p = P$ and the observation function $O_p(p', m, z_p) = 1$ if and only if $p' = z_p$ (and 0 otherwise).

Consider now the class observation component. There are $L \geq 2$ object classes $c_1, c_2, \ldots, c_L$, and the robot simultaneously observes $H \geq 1$ positions from the conveyor belt. Thus, the **state** contains the object class c^j at each such position j: $c^j \in C = \{c_1, c_2, c_3, \ldots, c_L\}$. Note that the subscript of c indexes the class values, while the superscript indexes positions. The **action** for this component is a decision on the class of the object at the start of the belt: $d \in D = \{d_1, d_2, \ldots, d_L\}$. The robot is expected to issue such an action only when it is sufficiently certain about this class; this will be controlled via the reward function, to be defined later.

To define the **transition function** T_c, we first need to give the overall action space available to the robot, which consists of all the motion and class decision actions $A = M \cup D$. Note that at a given step, the robot may either move between viewpoints, or make a class decision. Then:

$$T_c^j(c_i'^j, a, c) = \begin{cases} 1, & \text{if } a \in M \text{ and } c'^j = c^j \\ 1 & \text{if } a \in D \text{ and } j \leq H-1 \text{ and } c'^j = c^{j+1} \\ \frac{1}{L}, & \text{if } a \in D \text{ and } j = H \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

In order, the branches of this transition function have the following meaning. The first branch encodes that, if the robot moved between viewpoints (so the belt did not advance), then the classes remain the same on all H positions of the belt. The second branch, on the other hand, says that the classes move after a decision action (which automatically places the object in the right bin and advances the belt): the new class on position 1 is the old one on position 2, and so on. The third branch also applies for a decision action, and its role is to initialize the class value at the last position H. Note that in reality the class will be given by the true subsequent object, but since the POMDP transition function is time-invariant, this cannot be encoded and we use a uniform distribution over the classes instead. The fourth branch simply assigns 0 probability to the transitions not seen on the first three branches. To better understand what is going on, see Figure 1.

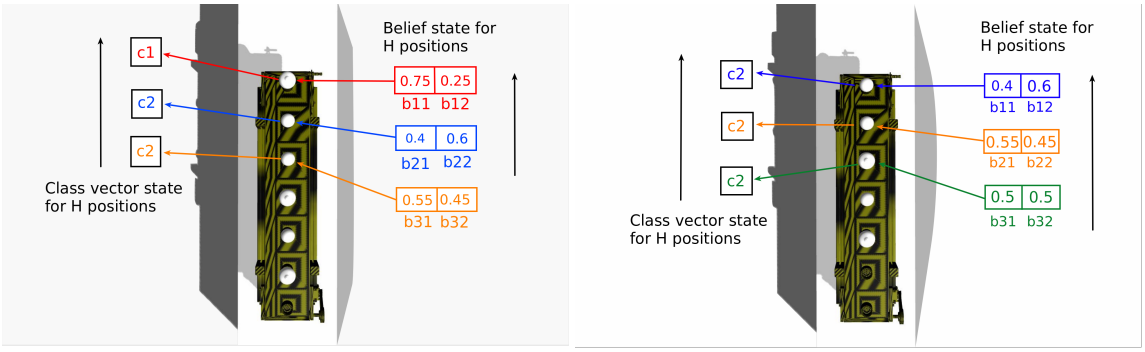

Figure 1. Belief state and class vector state before propagation (**left**) and after propagation (**right**), shown for an top view of the conveyor belt. Beliefs and classes are maintained maintained at each position, and whenever a decision is taken and the belt moves, these values are propagated in the direction indicated by the arrows.

The classes are of course not accurately observable, so we need to extract information about them via observations, and maintain a belief over their values. We will do this in a factored fashion, separately for each observed position on the belt.

The robot makes an **observation** $z^j \in \{z_1, z_2, \ldots, z_L\}$ about each position j, where z_i^j means that the object at position j is seen to have class i (which may or may not be the true class c_i^j).

Observations at each position j are made according to the **observation function** o^j:

$$o^j(s', a, z^j) = P(z^j | p', c^j), j \leq H \tag{4}$$

where $P(z^j | p', c_i^j)$ is the probability of making observation z^j from the viewpoint p' just reached, when the underlying class of object is c^j. These probabilities are application-dependent: they are given among others by the sensor properties, classification algorithm accuracy, actual viewpoint positions, etc. If a good *a priori* sensor model is available, it can inform the choice of o^j. However, the only generally applicable way of obtaining the observation function is experimental. For each viewpoint p', position j, and underlying class c^j, a number n of independent observations are performed, and the classes observed are recorded. Then, $o^j(s', a, z^j)$ is computed as the ratio between the number of observations resulting in class z^j, and n. Note that our approach is thus independent of the details of the classifier, which can be chosen given the constraints in the particular application at hand. Any classifier will benefit from our approach in challenging problems where the object shape is ambiguous from some viewpoints.

The overall state signal of the POMDP is $s = (c^1, c^2, \ldots, c^H, p)$, with state space $S = C^H \times P$. We have already defined the action space A, and the overall observation z is $(z^1, \ldots, z^j, z_p)$. We will not explicitly define the joint transition and observation functions T and O as the equations are overly complicated and do not really provide additional insight; nevertheless, the procedure to attain them follows directly.

Instead, let us focus now on the belief state. There is one such belief state $b^j \in [0,1]^L$ at each position j, which maintains the probabilities of each possible class value c^j at that position. Note that b_i^j is the belief that c^j is equal to c_i. Then, at any motion action, observations are performed according to O and the belief state is updated per the usual Formula (1). After decision actions however, there is a special behavior:

$$\begin{aligned} &\forall i \leq L\,,\, j \leq H \\ &{b'}_i^j = \begin{cases} b_i^{j+1}, & \text{if } j \leq H-1 \\ \frac{1}{L}, & \text{if } j = H \end{cases} \end{aligned} \tag{5}$$

What is happening is that the old belief state at $j+1$ is moved to j, and the belief for the last position is initialized to be uniform, as there is no prior information about the object (if a prior is available, then it should be used here). Figure 1 also provides an example for the propagation process of the beliefs.

The overall **reward function** is initially defined as follows:

$$R(s,a) = \begin{cases} r_{\max} & \text{if } a = d_i \text{ and } c^1 = c_i \\ -r_{\min} & \text{if } a = d_i \text{ and } c^1 \neq c_i \\ -1, & \text{otherwise} \end{cases} \quad (6)$$

At each motion action, a constant reward of -1 is received, which encodes time or energy consumption required to move the robot arm. When a decision is made, a reward $r_{\max}$ is obtained if the decision was correct (the class was well identified), and the incorrect-decision penalty $r_{\min}$ is assigned otherwise.

2.2. Adding Rewards Based on the Information Gain

The reward function (6) is based only on performance in the task (correct or incorrect decisions, and a time/energy penalty). In our active perception problem, it is nevertheless essential that before taking a decision, the algorithm is sufficiently confident about the object class. Of course, the incorrect decision penalty indirectly informs the algorithm if the class information was too ambiguous. We propose however to include more direct feedback on the quality of the information about the object class in the reward function. This is a novel contribution compared to [19].

Specifically, since in our problem the belief is a distribution over object classes (or over combinations of classes, for the multiple-position variant), we will characterize the amount of extra information provided by an action by using the information gain—or Kullback-Leibler divergence—between the current belief state and a possible future one:

$$IG(b,b') = \sum_s b(s) log_2 \frac{b(s)}{b'(s)} \quad (7)$$

Informally, we expect the information gain to be large when distribution b' is significantly "peakier" than b, i.e., the object class is significantly less ambiguous in b' than in b.

We will also need the entropy of the belief state, defined as follows:

$$\mathcal{H}(b) = \sum_s b(s) log_2 \frac{1}{b(s)} \quad (8)$$

To understand how the information gain is exploited in our approach, we must delve into the planning module, see also Section 2.4. This module has the role of finding a good sequence of actions that maximizes the amount of reward—in our case, a sequence of viewpoints, which improves the likelihood of a proper sorting for candidate objects; and of decisions on the classes of these objects. The planning module solves the POMDP problem using the DESPOT algorithm of [15], using an online approach that interleaves planning and plan execution stages. We will explain a few details about DESPOT, to the extent required to understand our method; the complete algorithm is rather intricate and outside the scope of this paper.

DESPOT constructs a tree of belief states, actions, and observations. Figure 2 gives an example of such a tree. Each square node is labeled by a belief state b, and may have a round child for each action a; in turn, each such action node may have a square, belief child for each observation z, labeled by the belief b' resulting from a and z. A tree represents many possible stochastic evolutions of the system, e.g., for the sequence of actions $[a_2, a_1]$ there are four possible belief trajectories in the tree of Figure 2: those ending in the 9th, 10th, 13th and 14th leaves at depth 2.

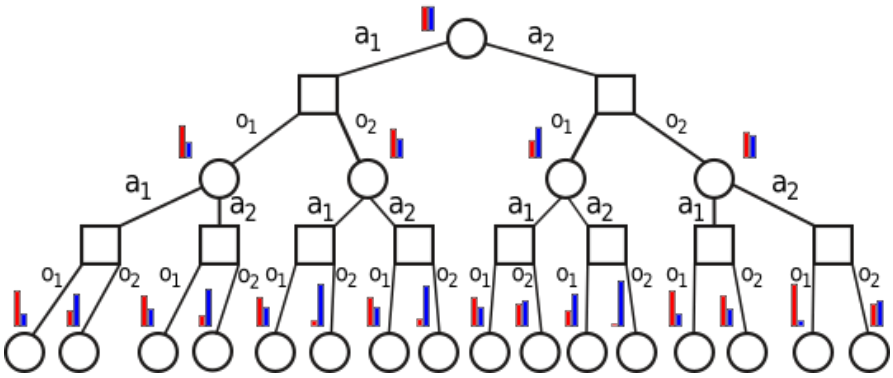

Figure 2. An example of a belief tree of the type constructed by DESPOT. Each circle represents a belief state node, and the squares are action nodes. The bar graph is the belief state associated to a node (only two classes are considered for readability).

We will work with a reward function $\rho(b, a, b')$ that is defined on transitions between belief nodes of this tree. For the original task-based POMDP reward function R in the section above, the corresponding belief-based reward would be:

$$\rho_t(b, a, b') = \sum_s b(s)R(s, a), \quad \forall b'$$

where the subscript t indicates this is the direct task reward. Note that in DESPOT, beliefs are approximately represented in the form of a set of particles, and belief rewards are similarly approximated based on these particles. We implicitly work with these approximate belief versions, both in the equation above and in the sequel.

We include the information gain by using a modified reward, as follows:

$$\rho(b, a, b') = \rho_t(b, a, b') + \alpha IG(b, b') \tag{9}$$

Thus, larger rewards are assigned to actions that help disambiguate better between object classes. Here, $\alpha \geq 0$ is a tuning parameter that adjusts the relative importance of the information-gain reward. Later on, we study the impact of α on performance.

To choose which nodes to create in developing the tree, DESPOT requires upper and lower bounds on the values (long-term expected rewards) of beliefs. It computes lower and upper bounds $\mathcal{L}_t$ and $\mathcal{U}_t$ of the task-reward values with well-known procedures in the PODMP literature [20]. To include the information gain, we leave the original lower bounds unchanged; since information gains are always positive, the lower bounds computed for the task rewards remain valid for the new rewards. For the upper bounds, we add α times the entropy of b as an estimate of the upper bound of any sequence of information gains:

$$\mathcal{U}(b) = \mathcal{U}_t(b) + \alpha \mathcal{H}(b) \tag{10}$$

2.3. Complexity Insight

A key factor dictating the complexity of the problem is the branching factor of the tree explored by the planning algorithm. To gain some more insight into this, let us examine a simple case where there are two viewpoints labeled L (for Left) and R (for Right), two classes labeled 1 and 2, and the observation function given in Table 1. Thus, if q is close to 1, then from viewpoint L class 1 is seen more accurately, and from viewpoint R class 2 is seen more accurately.

Table 1. A simple observation function. The left side of the table shows the probabilities of class observations z when the true class is $c = 1$; and the right side shows the case when $c = 2$. For instance, when $c = 1$ and the viewpoint is R, the robot observes the correct class with probability $1 - q$.

$P(z\|p, c = 1)$			$P(z\|p, c = 2)$		
$z \backslash p$	L	R	$z \backslash p$	L	R
1	q	$1 - q$	1	$1 - q$	q
2	$1 - q$	q	2	q	$1 - q$

Take a uniform initial belief, $b_0 = [0.5, 0.5]$. For this case, if we define the probability $P(b)$ of a belief (round) node in Figure 2 as the product of all observation probabilities from the root to that node, we can describe the tree explicitly. In particular, at depth d we will have only nodes with the following structures:

$$b = \left[\frac{q^k}{t_k}, \frac{(1-q)^k}{t_k}\right] \text{ or } \left[\frac{(1-q)^k}{t_k}, \frac{q^k}{t_k}\right], \qquad P(b) = \frac{t_k}{2}[q(1-q)]^{\frac{d-k}{2}}$$

where $t_k = q^k + (1-q)^k$, and k decreases in steps of 2 from d down to 0 when d is even, or to 1 when d is odd. The proof is an intricate induction, which we skip for space reasons. Instead, we plot in Figure 3 an example evolution of the probabilities $P(b)$ as a function of d (up to 100) and of the resulting values of k, for the particular case when $q = 0.9$. These results say that at each depth d, when q is large (i.e., when sensing is good) there are only a few classes with large probabilities: probabilities drop exponentially as k decreases. This is encouraging, because results in [4] suggest that complexity is small when node probabilities are skewed in this way (results there were for a different algorithm, AEMS2 [21], but we believe this principle is generally applicable to any belief-tree exploration algorithm; see also the related concept of covering number [22,23]).

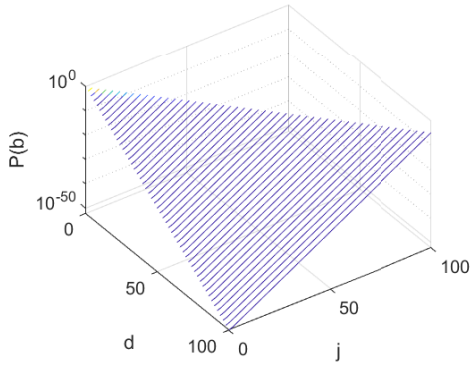

Figure 3. Evolution of probabilities with d and k. Note the logarithmic scale on the Z axis.

Obtaining a full analytical statement of this insight seems difficult. Instead, next we study empirically the effective branching factor of DESPOT with information-gain rewards, for $\alpha = 5$, and for a slightly more complicated version of the problem with 4 classes (the case of 2 classes is not informative as the algorithm only develops very shallow trees). The branching factor is estimated by letting the algorithm run for a long time from a uniform initial belief, and dividing the number of belief nodes (round in Figure 2) at depth $d + 1$ by the number at d. We obtain a value of 5.92 for the largest branching factor across all depths d. Note that the largest possible branching factor is 32, so the effective branching factor is significantly smaller, suggesting that the problem is not overly difficult to solve.

2.4. Hardware, Software, and Experimental Setup

2.4.1. Hardware and Software Base

The Baxter research robot developed by Rethink Robotics has been used for both simulated and real experiments. An Asus Xtion 3D sensor was mounted on one of the robot's wrists, while a conveyor belt, transporting different models of light bulbs, was placed in front of the robot. The motion planning tasks for the arms were carried out by solvers specific to the robot platform. The functionalities of PCL (Point Cloud Library) were used to construct the modules that handle all aspects regarding the point clouds acquired by the sensor. The active perception pipeline was developed in C++ and Python and was integrated into ROS (Robot Operating System), while Gazebo was used for simulation purposes.

2.4.2. Active Perception Pipeline

The pipeline consists of two high-level modules: detection and planning. In turn, detection includes acquisition, preprocessing and classification steps. Several components of the pipeline are similar to the ones presented in [24], what is different is the classification algorithm and module used, and most importantly, the presence of a planning module in the pipeline.

The pipeline approach was preferred because of its flexibility, as modifications to the underlying submodules can be performed without affecting the overall workflow of the pipeline. Figure 4 provides a graphical overview of the proposed pipeline, where the arrows show the flow of the information in the pipeline.

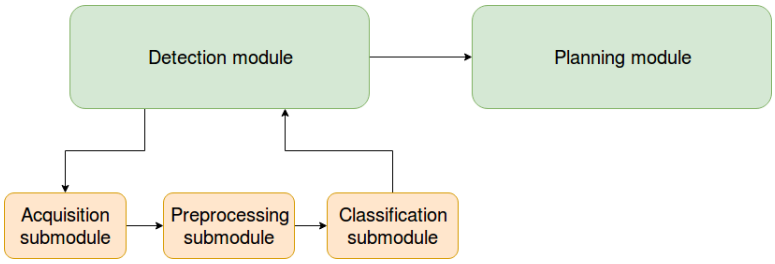

Figure 4. Structure of the pipeline.

The robot performs 3D scans of the conveyor belt in order to detect the objects that are being transported, which for our particular application are light bulbs of different shapes. The data acquired is in form of point clouds and tasks such as saving and loading are handled by the acquisition submodule [25].

The raw sensor data from the depth sensor is corrupted by noise, especially for translucent or highly reflective objects such as glass or shiny metal surfaces. Our setup includes such objects nearby the conveyor-belt, thus a pre-processing pipeline was carefully designed in order to mitigate the effect of measurement noise. This pipeline starts by cleaning up the data of any point that has NaN (not a number) valued coordinates, which were affected by noise. Next, the data is segmented using a pass-through filter: the points in the neighbourhood of the positions of interest on the conveyor belt are the ones that remain and all the others are removed. The remaining points are clustered using Euclidean clustering, with each cluster forming a candidate for classification. Because the point clouds scanned from different viewpoints have different numbers of points, which can affect the classification, each extracted cluster is uniformly sampled using a 3D voxel filter. Based on the robustness analysis of the depth sensor [26] the tuning of the 3D processing pipeline was performed for the specific indoor scenario with objects in close proximity.

The classification module receives as input a prepared point cloud, corresponding to a candidate light bulb, and has the role of classifying it. A prior training step is necessary, in which the uniformly sampled point clouds for known light bulb classes are used. The Viewpoint Feature Histogram (VFH) [27] provides descriptors containing information about the shape of the cloud and also viewpoint information.

During training, a k-d tree is built from the clouds taken from each observation point corresponding to each class of light bulb. The classification becomes a nearest-neighbour search problem, in which for a candidate cloud a list of the trained clouds is returned sorted by the shortest distance to the candidate cloud.

The crucial component of the pipeline is the planning module, which has the role of finding a good sequence of observations and class decisions. As previously stated, the planning module solves the POMDP problem using DESPOT [15]. The belief state is updated with the results coming from the detection module. The planning module returns an action, either of motion or decision type, which is processed and further transmitted to the motion planning and execution modules specific to the robot.

2.4.3. Workflow

The workflow of the application is presented in Figure 5. The robot starts the sorting task by moving to an initial viewpoint. From there it performs and processes an observation, which consists of the acquisition of point clouds; their preprocessing and the classification of the extracted candidates; as well as the update of the belief state. After that it computes the upcoming best action. If it is a motion action, it moves to the next viewpoint and proceeds with the steps already presented. For a decision action, the light bulb is sorted and the conveyor belt advances.

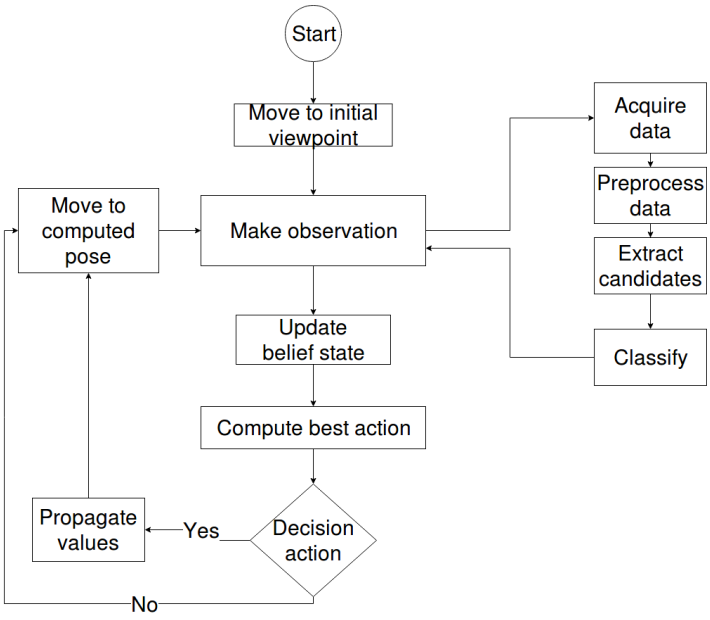

Figure 5. The workflow of the application.

2.4.4. Experimental Setup

To construct the viewpoints from which observations can be performed, we started by uniformly sampling the upper half of a sphere [28] that has a radius of 70 cm and is centered on the conveyor belt. The lower half of the sphere was eliminated to avoid occlusions with the belt. Points that could not be reached by the arm due to kinematic constraints were eliminated, and the remaining points were connected into a graph. Figure 6 shows the set of sampled points that remained. To obtain the graph, our procedure finds the nearest neighbors of each point along the cardinal directions (North, South, East, West). Thus, the robot is not able to travel between any two points freely, but is restricted to travel first through the neighbouring points.

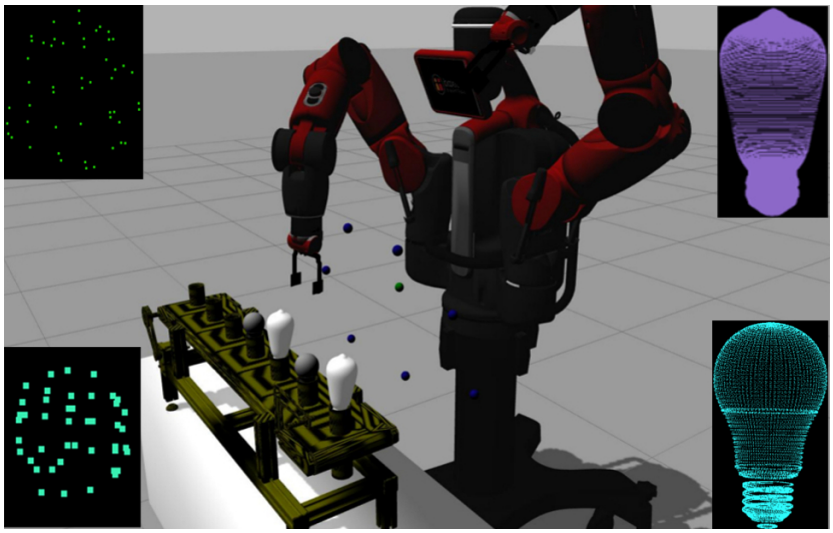

Figure 6. The robot sampling a set of points and testing their rechability in a simulated environment. The models for livarno and elongated light bulbs (**right side**) and the corresponding sampled point clouds used for classification of the light bulbs (**left side**).

In Figure 7, an example graph can be seen. The sampled points and their closest neighbours along the cardinal directions are plotted. In this figure, green is North, red is East, yellow is South and blue is West. e.g., a green line between two points means that the top point is reachable from the bottom one by travelling north.

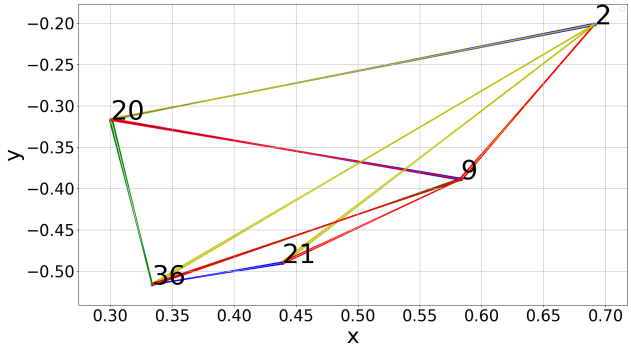

Figure 7. Graph constructed from a set of sampled reachable points.

We run several batches of experiments, varying two key parameters: the number of positions observed simultaneously (one or two, with belief propagation); and whether the rewards include or not the information gain. Note that the third and further positions are too far for the point clouds to offer useful information. In all experiments, the robot must sort 10 light bulbs. "Sorting" a light bulb is defined in the POMDP framework as equivalent to taking a decision action; once the class is decided, the remainder of the task (picking up the bulb, placing it in the right bin) is executed in a preprogrammed fashion. The number of correct and incorrect classifications among the 10 bulbs is reported. For all simulation experiments, we repeated the experiments 50 times and reported the means and 95 % confidence intervals on these means.

Observation probability distributions were computed experimentally beforehand for every vertex of the graph. for both the single- and two-position cases. To do this, from each vertex, several scans were performed for each true class of light bulb. After preprocessing each scan, the segmented bulbs were classified using the method explained above, and the observation probability distribution was computed as the fraction of the full set of experiments in which each class (correct or incorrect) was observed; see also Section 2.1. Table 2 exemplifies the observation probability distribution for the elongated bulb

(true class), when observing the first position, for a few representative viewpoints. It shows how likely is to observe the respective class when looking at it and how likely is to misclassify it as one of the other classes, from these viewpoints. In the two-position case, Table 3 shows an example of how likely is to observe the different classes of bulbs on the first two positions of the belt, from the same viewpoints as in Table 2. In both tables, the labels "elongated", "livarno", "mushroom", and "standard" refer to the shape of the light bulb, which directly corresponds to its class.

Table 2. Observation probability distribution for the single-position case when the underlying object is of the elongated class.

Pr(o) / Viewpoint	elongated	livarno	mushroom	standard
64	0.6	0.2	0	0.2
43	0.5	0.3	0.1	0.1
87	0.8	0.1	0.1	0

Table 3. Observation probability distribution in the multiple-position case. Top: position 1, bottom: position 2. The underlying object is of the livarno class on position 1, and elongated on position 2.

Pr(o) / Viewpoint	elongated p1	livarno p1	mushroom p1	standard p1
64	0.3	0.6	0.1	0
43	0.1	0.8	0	0.1
87	0.2	0.7	0	0.1
Pr(o) / Viewoint	**elongated p2**	**livarno p2**	**mushroom p2**	**standard p2**
64	0.4	0.1	0.4	0.1
43	0.5	0.2	0.2	0.1
87	0.2	0.2	0.6	0

3. Results and Discussion

3.1. Effect of Decision Penalty and Multiple-Position Observations

In all the experiments of this section, we study the impact of the incorrect classification penalty, r_{min}, as it varies in absolute value from 5 to 1000 (i.e., small to heavy penalty for incorrect decisions). This is always done for the task of sorting a sequence of 10 light bulbs, and for observations of either the first object on the conveyor belt, or of the first two objects. Compared to [19], where only two object classes were considered, here we use four classes; we compare to a simple baseline algorithm; and we additionally report confidence intervals for all the results, ensuring that they are statistically significant.

3.1.1. Single-Position Observations. Comparison to Baseline

We begin with the experiments in which only the first position on the belt is observed. Figure 8 shows the belief state evolution for one such experiment, to get insight into how well the algorithm performs. The real types of the 10 light bulbs are chosen randomly by the simulator, the letters being the first letters of the four classes defined (thus, *e* means elongated, *l* livarno, *m* mushroom, and *s* standard). The steps at which decisions are taken are also plotted (*d_e* for elongated decision, *d_l* for livarno decision, *d_m* for mushroom decision and *d_s* for standard decision). The algorithm waits until the probability of a class reaches a large enough value, and then issues a decision action. Because only the light bulb from the end of the conveyor belt is observed, there is no belief propagation and after each decision the belief state is reinitialized to uniform values.

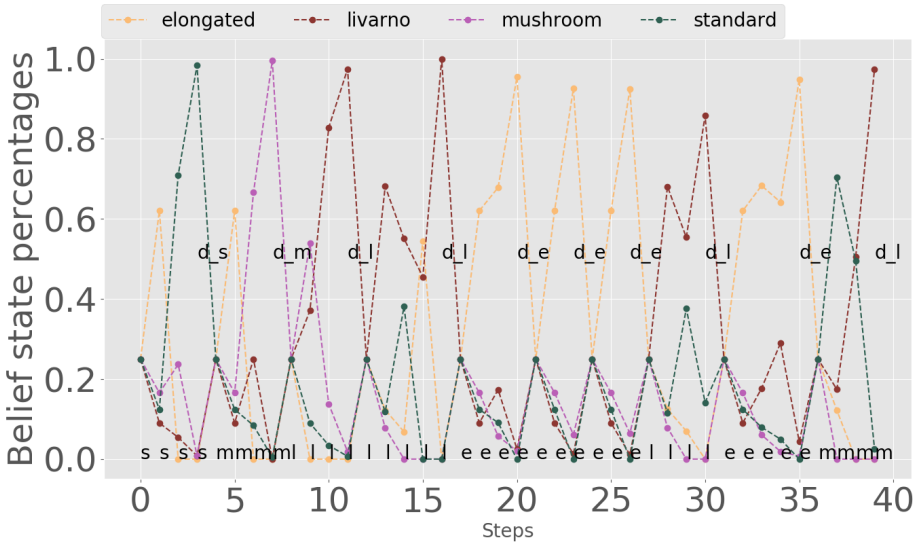

Figure 8. Belief state evolution for single-position observation experiments.

Figure 9 (left) shows the ratio of positives (that is, correct classifications) to negatives (incorrect classifications) among the 10 light bulbs, as r_{min} varies from 5 to 1000. Figure 9 (right) similarly shows the number of steps needed to sort the light bulbs. Here, the steps counted consist of both the motion and decision actions. Each figure shows a mean value and a 95% confidence interval on the mean as an error bar; for the left graph, the bar should be interpreted as saying that the mean split between the number of positives and negatives is within the bar with 95% probability.

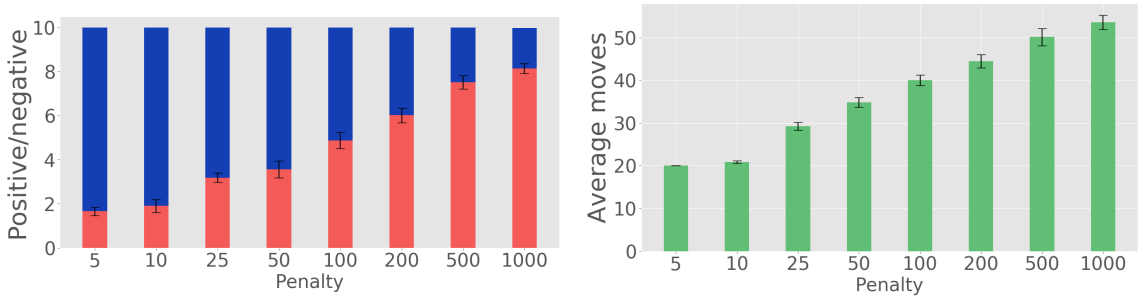

Figure 9. Single-position observations: Positives–negatives counts (**left**) and number of steps needed for sorting (**right**).

The penalty r_{min} has a large impact on the performance. When r_{min} is too small, the task is finished in a small number of steps, but the robot decides prematurely on the class of the object, before having enough information, so the object is often misclassified. On the other hand, when r_{min} is too large, a good classification accuracy is obtained, but the number of steps is very large. Overall, r_{min} must be selected to achieve a good balance between the quality of the classification and the total number of steps required.

To verify whether our approach of judiciously choosing the sequence of viewpoints is in fact useful, we compare here to a simple baseline. We select two viewpoints in the set constructed that are furthest away from each other (indices 43 and 64), and hard-code a simple solution that observes the object alternately from the two viewpoints. We allocate 5 such observation steps to each lightbulb (leading to 50 steps in total), and after these steps we select the class associated with the largest probability in the belief state. We repeat this experiment 50 times, and the number of lightbulbs classified correctly is 3.36 ± 0.52 (mean and 95% confidence interval half-width). Compare e.g., to the planning solution for penalty 500, which in around 50 steps works classifies 7.52 lightbulbs correctly on average, see again Figure 9. Clearly, planning works much better.

3.1.2. Two-Position Observations

In this type of experiment, two positions of the conveyor belt are observed, and belief state values and the vector of classes are propagated after each decision, as explained in Section 2.1. As before, Figure 10 shows the evolution of the belief state, with the first position on top and the second on the bottom. The underlying states are plotted for each position. Note that if the belief state of the first position after propagation already has a clear candidate class, decision actions may follow one after the other, which already shows the advantage of observing two positions.

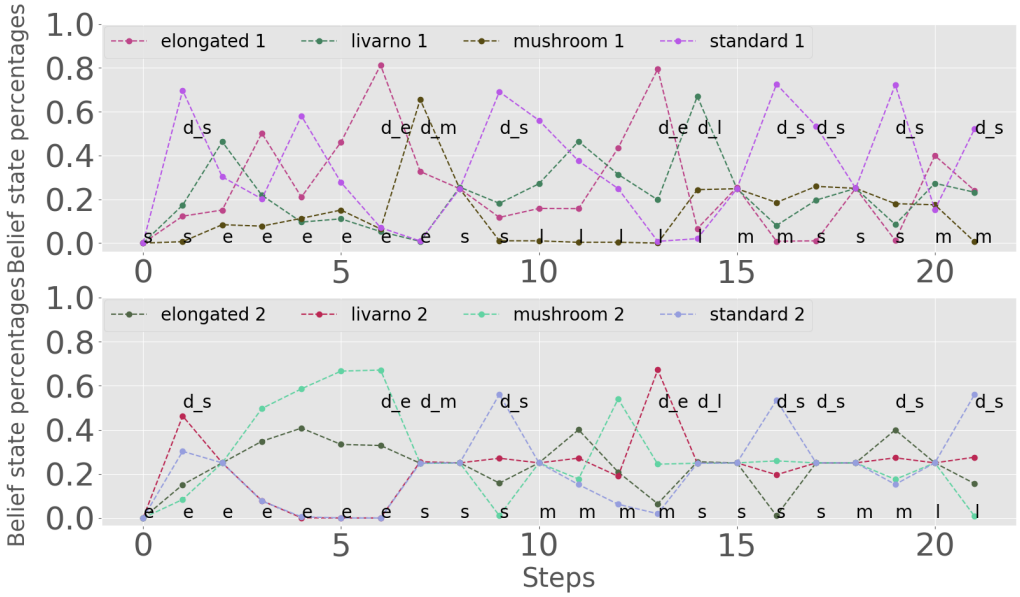

Figure 10. Belief state evolution when observing two positions from the conveyor belt.

The graph on the left of Figure 11 presents the ratio of positives to negatives, for the same range of rewards as above, but now for the multiple-observation experiment; and the graph on the right gives the number of steps. As for the single-position experiments, the choice of $r_{\min}$ must trade off classification quality (better when $r_{\min}$ is large) with the time/number of observations required (smaller when $r_{\min}$ is small).

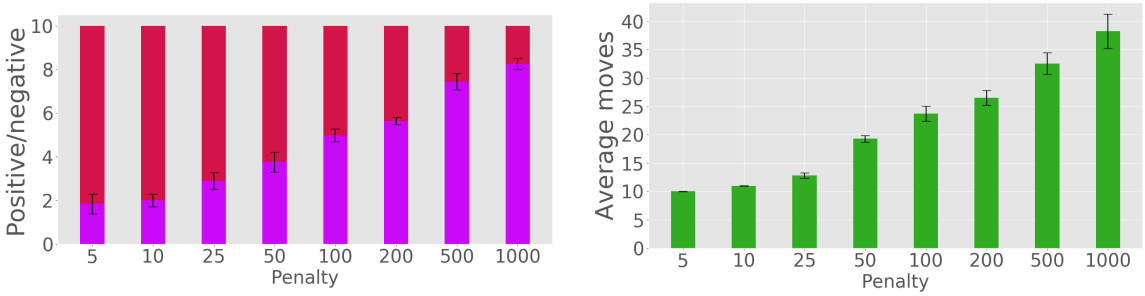

Figure 11. Two-position observations: Positive–negative counts (**left**) and number of steps needed for sorting (**right**).

3.1.3. Single-Position Versus Two-Position Observations

Next, we compare the case where one position is observed, with the multiple-position case. Figure 12 is a synthesis of the figures above, comparing the classification quality in the graph on the left, and the number of steps taken by sorting in the graph on the right. The evolution of classification quality with the reward value is similar for the two cases. However, a difference arises between the

total number of steps needed to finish sorting: the multiple-observation version requires a significantly smaller number of steps to achieve the same performance as its single-observation counterpart.

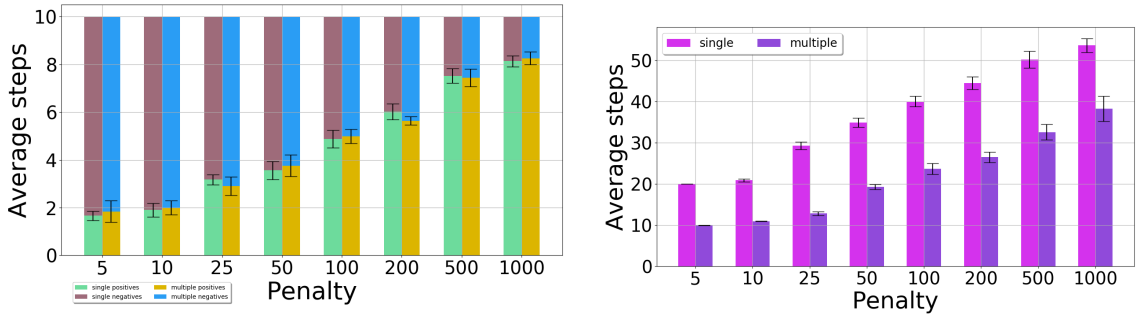

Figure 12. Comparison between the single-and multiple-observation experiments: Count of positive/negative classifications (**left**) and number of steps needed for sorting (**right**).

3.2. Effect of Including the Information Gain

Next, we focus on the case in which the rewards are computed taking into account the information gain; these results are novel compared to [19]. We study the effect of the weight α of the information gain in the reward, by measuring the number of steps needed to sort the 10 light bulbs. The value of α was varied from 0 to 500. The penalty r_{min} was kept equal to 50. To make the comparison fair, in all the experiments the DESPOT planner was configured to run for 0.5 s at each step.

The results for the case when only the end position of the belt is observed are given in Figure 13: mean number of steps and 95% confidence interval on the mean. We do not report the classification quality because it is roughly the same for all cases. The base case is for $\alpha = 0$ (task rewards only, no information gain). For values of α between 1 and 20 the algorithm performs considerably better, so the inclusion of the information gain clearly pays off. When α is too large, performance suffers, likely because the actual task rewards are almost entirely disregarded and the algorithm focuses solely on the information gain, which does not lead to a good solution.

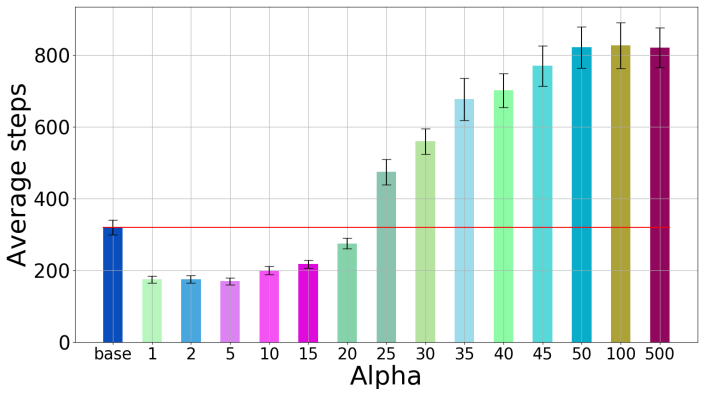

Figure 13. Single-position observations: Number of steps when varying the α parameter.

Next, Figure 14 shows the results when the first two positions from the conveyor belt are observed. The same values of r_{min} and α are used as before. Performance remains roughly the same when information-gain rewards are added. Thus, either of the two improvements proposed (two-position observations and the information gain) seems to be sufficient on its own. In more challenging problems, it may however become necessary to use them both.

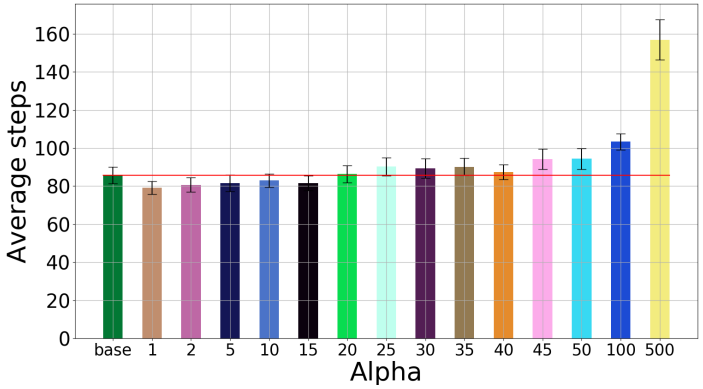

Figure 14. Two-position observations: Number of steps when varying the α parameter.

3.3. Real-Robot Experiments

An illustration of the real setup and experiment is given in Figure 15. In contrast to simulation, for simplicity here we only used task rewards, without the information gain ($\alpha = 0$); and performed experiments for only two values of the penalty: 25 and 50.

Figure 16 shows respectively the quality of the classification and the number of required steps. The real-robot results are similar to those in simulation: classification quality and the number of steps required both grow with the magnitude of the penalty. Compared to simulation, the ratio of correct classifications is lower, since the acquired data has a higher degree of noise corruption. A video of the robot, together with code for the experiments, can be found at http://community.clujit.ro/display/TEAM/Active+perception.

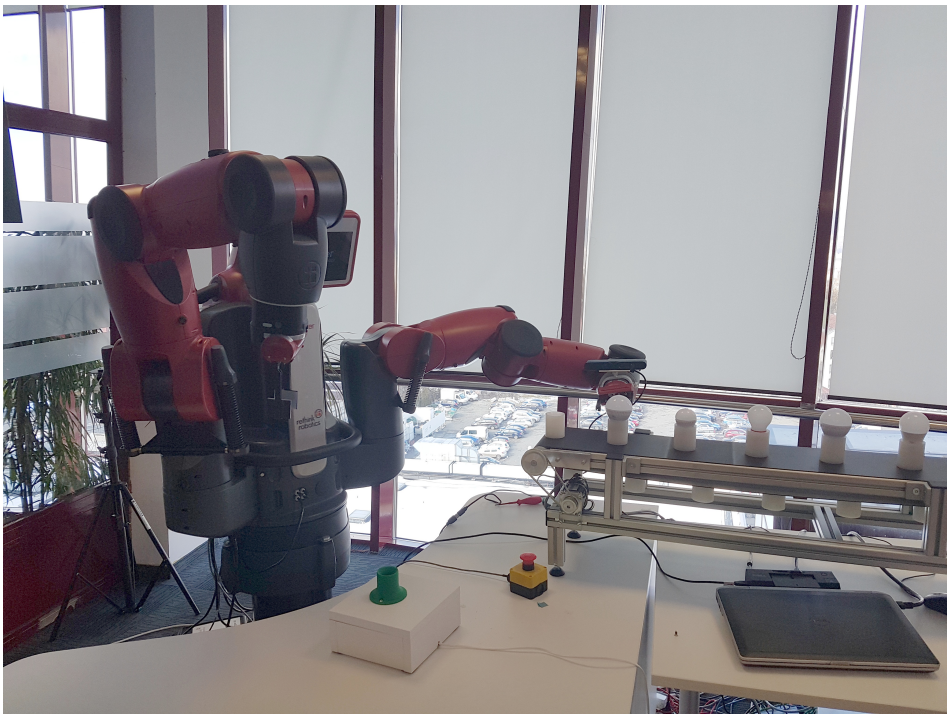

Figure 15. A real robot performing the sorting of light bulbs.

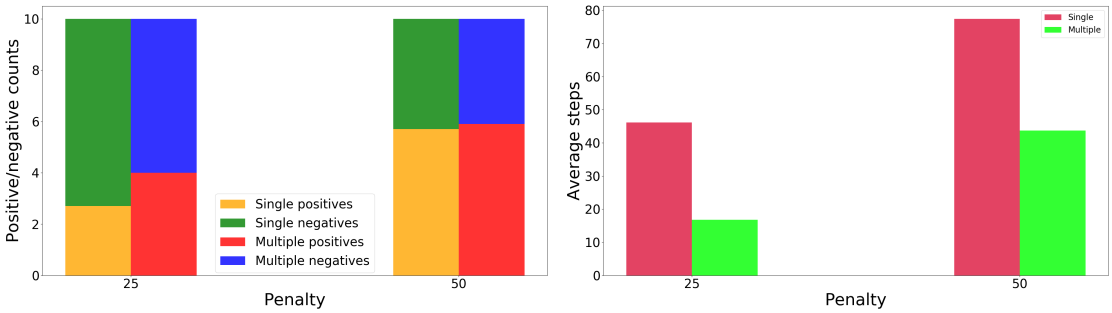

Figure 16. Real experiment, comparison between the single- and multiple-position observations: Count of positive/negative classifications (**left**) and total number of steps needed to sort 10 light bulbs (**right**).

4. Conclusions

In this work, the task of sorting objects transported using a conveyor belt was handled using an active perception approach, to mitigate imperfect detections and classifications. The approach describes the problem as a partially observable Markov decision process, uses 3D data to classify the objects, and computes the actions using a planner. The method was tested both in simulation and on a real Baxter robot equipped with an Asus 3D camera. One key feature of the approach is that it can computes rewards using the information gain to promote actions that better disambiguate between object classes. A second key feature is that the method allows observing several conveyor belt positions at once, and information is propagated across these positions. Either of these improvements reduces the total total number of steps required to sort a required number of light bulbs, without sacrificing classification accuracy.

Future work may include approaches where the robot collects images along its entire trajectory, instead of just at the viewpoints, using a better sensor since the Asus 3D camera provides less accurate data in this regime. Better classifiers could be investigated in applications where computational resources allow them. For example, in separate tests CNN classifiers [29] provided single-image accuracy that is a few percent better than the method we used above.

Author Contributions: Conceptualization, L.T.; Formal analysis, L.B.; Funding acquisition, L.B.; Investigation, L.T.; Resources, L.T.; Software, A.-D.M.; Supervision, L.B.; Validation, A.-D.M; Visualization, A.-D.M; Writing—original draft, A.-D.M; Writing review & editing, L.B.; All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Young Teams program of the Romanian Authority for Scientific Research via UEFISCDI grant number PN-III-P1-1.1-TE-2016-0670, contract number 9/2018, and by HAS Bolyai scholarship.

Conflicts of Interest: The author declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript (in alphabetical order):

DESPOT	Determinized Sparse Partially Observable Tree
PCL	Point Cloud Library
POMDP	Partially Observable Markov Decision Process
RGB-D	Red-Green-Blue-Depth
ROS	Robot Operating System
VFH	Viewpoing Feature Histogram

References

1. Militaru, C.; Mezei, A.D.; Tamas, L. Object handling in cluttered indoor environment with a mobile manipulator. In Proceedings of the 2016 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR), Cluj-Napoca, Romania, 19–21 May 2016; pp. 1–6.
2. Militaru, C.; Mezei, A.D.; Tamas, L. Lessons Learned from a Cobot Integration into MES. In Proceedings of the ICRA—Recent Advances in Dynamics for Industrial Applications Workshop, Singapore, 24–28 May 2017.

3. Militaru, C.; Mezei, A.D.; Tamas, L. Industry 4.0 – MES Vertical Integration Use-case with a Cobot. In Proceedings of the ICRA—Recent Advances in Dynamics for Industrial Applications Workshop, Singapore, 24–28 May 2017.
4. Páll, E.; Tamás, L.; Buşoniu, L. Analysis and a home assistance application of online AEMS2 planning. In Proceedings of the 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Daejeon, Korea, 9–14 October 2016; pp. 5013–5019.
5. Bajcsy, R. Active perception. *Proc. IEEE* **1988**, *76*, 966–1005. [CrossRef]
6. Aloimonos, J.; Weiss, I.; Bandopadhay, A. Active vision. *Int. J. Comput. Vis.* **1988**, *1*, 333–356. [CrossRef]
7. Eidenberger, R.; Scharinger, J. Active perception and scene modeling by planning with probabilistic 6d object poses. In Proceedings of the 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, Taipei, Taiwan, 18–22 October 2010; pp. 1036–1043.
8. Velez, J.; Hemann, G.; Huang, A.S.; Posner, I.; Roy, N. Planning to perceive: Exploiting mobility for robust object detection. In Proceedings of the Twenty-First International Conference on Automated Planning and Scheduling, Freiburg, Germany, 11–16 June 2011.
9. Holz, D.; Nieuwenhuisen, M.; Droeschel, D.; Stückler, J.; Berner, A.; Li, J.; Klein, R.; Behnke, S. Active recognition and manipulation for mobile robot bin picking. In *Gearing Up and Accelerating Cross-fertilization between Academic and Industrial Robotics Research in Europe*; Springer: Berlin/Heidelberg, Germany, 2014; pp. 133–153.
10. Patten, T.; Zillich, M.; Fitch, R.; Vincze, M.; Sukkarieh, S. Viewpoint Evaluation for Online 3-D Active Object Classification. *IEEE Robot. Autom. Lett.* **2016**, *1*, 73–81. [CrossRef]
11. Atanasov, N.; Le Ny, J.; Daniilidis, K.; Pappas, G.J. Decentralized active information acquisition: Theory and application to multi-robot SLAM. In Proceedings of the 2015 IEEE International Conference on Robotics and Automation (ICRA), Seattle, WA, USA, 26–30 May 2015; pp. 4775–4782.
12. Aleotti, J.; Lodi Rizzini, D.; Caselli, S. Perception and Grasping of Object Parts from Active Robot Exploration. *J. Intell. Robot. Syst. Theory Appl.* **2014**, *76*, 401–425. [CrossRef]
13. Cowley, A.; Cohen, B.; Marshall, W.; Taylor, C.; Likhachev, M. Perception and motion planning for pick-and-place of dynamic objects. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2013), Tokyo, Japan, 3–7 November 2013; pp. 816–823.
14. Atanasov, N.; Sankaran, B.; Le Ny, J.; Pappas, G.J.; Daniilidis, K. Nonmyopic view planning for active object classification and pose estimation. *IEEE Trans. Robot.* **2014**, *30*, 1078–1090. [CrossRef]
15. Somani, A.; Ye, N.; Hsu, D.; Lee, W.S. DESPOT: Online POMDP Planning with Regularization. In Proceedings of the Advances in Neural Information Processing Systems, Lake Tahoe, NV, USA, 5–10 December 2013; pp. 1772–1780.
16. Spaan, M.T.; Veiga, T.S.; Lima, P.U. Decision-theoretic planning under uncertainty with information rewards for active cooperative perception. *Auton. Agents Multi-Agent Syst.* **2014**, *29*, 1–29. [CrossRef]
17. Spaan, M.T.J.; Veiga, T.S.; Lima, P.U. Active Cooperative Perception in Network Robot Systems Using POMDPs. In Proceedings of the International Conference on Intelligent Robots and Systems (IROS), Taipei, Taiwan, 18–22 October 2010; pp. 4800–4805.
18. Mafi, N.; Abtahi, F.; Fasel, I. Information theoretic reward shaping for curiosity driven learning in POMDPs. In Proceedings of the 2011 IEEE International Conference on Development and Learning (ICDL), Frankfurt am Main, Germany, 24–27 August 2011; Volume 2, pp. 1–7.
19. Mezei, A.; Tamás, L.; Busoniu, L. Sorting objects from a conveyor belt using active perception with a POMDP model. In Proceedings of the 18th European Control Conference, Naples, Italy, 25–28 June 2019; pp. 2466–2471.
20. Ross, S.; Pineau, J.; Paquet, S.; Chaib-draa, B. Online Planning Algorithms for POMDPs. *J. Artif. Int. Res.* **2008**, *32*, 663–704. [CrossRef]
21. Ross, S.; Chaib-draa, B. AEMS: An Anytime Online Search Algorithm for Approximate Policy Refinement in Large POMDPs. In Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07), Hyderabad, India, 6–12 January 2007.
22. Zhang, Z.; Littman, M.; Chen, X. Covering number as a complexity measure for POMDP planning and learning. In Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, Toronto, ON, Canada, 22–26 July 2012.

23. Zhang, Z.; Hsu, D.; Lee, W.S. Covering Number for Efficient Heuristic-based POMDP Planning. In Proceedings of the 31th International Conference on Machine Learning, Beijing, China, 21–26 June 2014; pp. 28–36.
24. Mezei, A.; Tamas, L. Active perception for object manipulation. In Proceedings of the IEEE 12th International Conference on Intelligent Computer Communication and Processing, Cluj-Napoca, Romania, 8–10 September 2016; pp. 269–274.
25. Rusu, R.B.; Cousins, S. 3D is here: Point Cloud Library (PCL). In Proceedings of the 2011 IEEE International Conference on Robotics and Automation, Shanghai, China, 9–13 May 2011; pp. 1–4.
26. Tamas, L.; Jensen, B. Robustness analysis of 3d feature descriptors for object recognition using a time-of-flight camera. In Proceedings of the 22nd Mediterranean Conference on Control and Automation, Palermo, Italy, 16–19 June 2014; pp. 1020–1025.
27. Rusu, R.B.; Bradski, G.; Thibaux, R.; Hsu, J. Fast 3D recognition and pose using the Viewpoint Feature Histogram. In Proceedings of the 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, Taipei, Taiwan, 18–22 October 2010; pp. 2155–2162.
28. Deserno, M. *How to Generate Equidistributed Points on the Surface of a Sphere*; Technical report; Max-Planck Institut fur Polymerforschung: Mainz, Germany, 2004.
29. Qi, C.R.; Su, H.; Mo, K.; Guibas, L.J. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. *arXiv* **2017**, arXiv:1612.00593.