2013 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS)
November 3-7, 2013. Tokyo, Japan

Improving Robot Plans for Information Gathering Tasks through Execution Monitoring

Minlue Wang[1] Sebastien Canu[1] Richard Dearden[1]

Abstract— **Recent advances in navigation and control of robots has increasingly led to systems where the actions are deterministic and the challenge is to collect information about the world using noisy sensors. Examples include search and rescue, Mars rover planning and robotic monitoring tasks. However, theoretical results show that in general these problems are as hard as solving partially observable Markov decision problems (POMDPs). We propose an approach where we build plans assuming both the actions and the observations are reliable, then monitor the execution of the plan and use a value of information calculation to add information gathering actions on-line. We describe two variants: one using a classical contingency planner to generate the initial plan, and the other using a Markov decision problem planner. We show how in both cases the addition of execution monitoring can considerably improve overall performance with lower computational cost than solving the original POMDP.**

I. Introduction

Recently there has been significant interest in a class of problems that are characterised by robots gathering information about the world in order to accomplish some task. In many of these problems the actions available to the robot can be divided into a set of deterministic actions that change the state of the world—we refer to these as *state changing actions*—and a set of actions which do not change the world state but stochastically provide information about it—*observation-making actions*. A good example of this kind of problem is the *RockSample* domain [1] in which a rover must collect a sample from a scientifically interesting rock, but does not know in advance which rocks are interesting. Other examples include robots in office assistant roles [2] and the problem of deciding what computer vision algorithms to apply to an image to answer a query [3].

Problems of this type can be modelled as partially observable Markov decision problems (POMDPs), but sine POMDPs are more general, allowing stochastic action effects, we might hope to gain computational advantage by studying this interesting subclass specifically. Following [4], we refer to these problems as *quasi-deterministic POMDPs* (for a formal definition, see Section II). They differ from another well-studied subclass of POMDPs, DET-POMDPS [5] in that they allow uncertainty in the observations (DET-POMDPs are entirely deterministic apart from the initial state).

An alternative way to think about quasi-deterministic POMDPs is as an extension of classical planning to allow *noisy* information gathering actions. Many classical approaches use modal logics of knowledge to represent known information (see for example [6]) but they assume this gathered information is always reliable.

Most existing approaches (see Section VII) to solving quasi-deterministic problems use POMDP solvers (the notable exception is [2]). However, POMDP algorithms do not scale to the size of problems we are interested in—problems with fewer than ten thousand states (equivalent to only 13 binary variables) take hours to solve.

The approach we take in this paper is to use much faster planning algorithms to produce plans that don't fully take the uncertainty in the observations into account and to use execution monitoring to "repair" the plans at runtime to improve their use of observations. Here we report on two approaches to the initial plan generation, one that uses a classical contingency planner (Section III), and the other that uses a Markov decision process (MDP) planner (in Section V). In each case, we automatically rewrite the original problem into a form that can be solved by the faster planner. Although the state changing actions are still deterministic, the observation-making actions introduce nondeterminism into the translated problem, hence the need for contingent branches and MDPs.

Plans created from the transformed problems are executable in the original problem, but are unlikely to be optimal because they assume the observation-making actions are perfectly reliable. During execution of the plan we monitor the actual belief state of the agent. When an observation-making action is reached, since the robot won't reliably know the true state even after the observation, we use a value of information (VoI) calculation to find an observation action that will change the belief state to improve the expected quality of the remainder of the plan. If such an action exists, we execute the one with the highest VoI. This repeats until no action has positive value. Execution monitoring for the contingent plan approach is described in Section IV, and for MDPs in Section VI. In Section VII we review related work, and in Section VIII we present our results in simulation and on a mobile robot.

II. Quasi-Deterministic Planning Problems

The POMDP model of planning problems is sufficiently general to capture all the complexities of the domains we consider here. Formally, a POMDP is a tuple $\langle S, A, T, \Omega, O, R\rangle$ where:

- S is the discrete state space of the problem.
- A is the set of actions available to the agent.

[1]M. Wang, S. Canu and R. Dearden are with the School of Computer Science, University of Birmingham, Edgbaston, Birmingham B15 2TT, UK. mxw765@cs.bham.ac.uk, sebcanu@wanadoo.fr, richard.dearden@gmail.com

- T is the transition function that describes the effects of the actions. We write $P(s, a, s')$ where $s, s' \in S, a \in A$ for the probability that executing action a in state s leaves the system in state s'.
- Ω is the set of possible observations.
- O is the observation function that describes what is observed when an action is performed. We write $P(s, a, s', o)$ where $s, s' \in S, a \in A, o \in \Omega$ for the probability that observation o is seen when action a is executed in state s resulting in state s'.
- R is the reward function that defines the value to the agent of particular activities. We write $R(s, a)$ where $s \in S, a \in A$ for the reward the agent receives for executing action a in state s.

Quasi-deterministic planning problems are POMDPs in which the actions are of two types: *state-changing actions* and *observation-making actions*, defined as follows:

Definition 1: A **state changing action** a is one such that:

$$\forall s \exists s' : P(s, a, s') = 1 \wedge \forall s'' \neq s' \; P(s, a, s'') = 0$$

$$\forall s, a, s' \exists o : P(s, a, s', o) = 1 \wedge \forall o' \neq o \; P(s, a, s', o') = 0$$

That is, for every state the action is performed in there is exactly one state it transitions to, and its observations are uninformative.

Definition 2: An **observation-making action** a has an unconstrained observation function O and:

$$\forall s : P(s, a, s) = 1 \wedge \forall s' \neq s \; P(s, a, s') = 0$$

Definition 3: A **quasi-deterministic POMDP** is a POMDP in which every action is either state changing or observation-making.

In practice, the problems we are interested in are unlikely to be specified as flat POMDPs. We assume they will be specified using state variables, for example using a dynamic Bayesian network as in symbolic Perseus [7]. Similarly, we use *factored-observable models* (see [4]), a variable decomposition of the observations, to simplify the representation of the observation space. We write $b(p_i)$ for the probability that some state variable p_i is true in the initial belief state, where $s = \langle p_1, p_2, ..., p_n \rangle$ is the state variable representation of POMDP state s.

We illustrate this using the *RockSample* domain [1], in which a robot can visit and collect samples from a number of rocks at locations in a rectangular grid. Some of the rocks are "good" (scientifically interesting) and the robot will get a reward for sampling them. Others are "bad" and the robot gets a penalty for sampling them. The robot's state-changing actions are to move in the grid and to sample a rock, and the observation actions are to check each rock, which returns a noisy observation of whether the rock is good. If the robot checks a "good" rock, the observation will be `good` with probability 0.8, but `bad` with probability 0.2, and the reverse for a "bad" rock.

III. Contingent Planning Approach

As we said in the introduction, the basis of our approach is to build approximate plans and then improve them at execution time by determining the number of times that observation-making actions will execute. To compute whether it is worth adding an observation-making action at any point, we need to know the effects that action will have on the quality of our future behaviour, and for that we need to know what future plans we might execute. We do this by generating a branching plan where every time the plan includes an observation-making action, there is a branch at that point for each possible observation the action could produce. This generating of a branching plan is done offline.

To use a contingency planner to solve a quasi-deterministic POMDP, we translate it into the probabilistic planning domain definition language (PPDDL) [8]. PPDDL is designed for problems that can be represented as completely observable Markov decision problems. To use it for a quasi-deterministic problem we need to represent the effects of the observation-making actions. We do this by adding a belief predicate *bel()* to indicate the agent's belief that a state variable has a particular value. For instance, in the *RockSample* problem we use *bel(rover0, rock0, good)* to reflect that *rover0* knows *rock0* has good scientific value. The belief predicate is included in the effects of the observation-making action and also appears in the preconditions of the state-changing actions that lead to the goal to ensure that the agent has to find out the value using the observation actions. The `checkRock` action is then transformed to the PPDDL action:

```
(:action checkRock
 :parameters
  (?r -rover ?rock -rocksample ?value -rockvalue)
 :preconditions
  (not (measured ?r ?rock))
 :effect
  (and (measured ?r ?rock)
       (when (and (rock_value ?rock ?value)
                  (= ?value good))
         (probabilistic
           0.8 (and (bel ?r ?rock good)))
           0.2 (and (bel ?r ?rock bad)))
       (when (and (rock_value ?rock ?value)
                  (= ?value bad))
         (probabilistic
           0.8 (and (bel ?r ?rock bad)))
           0.2 (and (bel ?r ?rock good)))))
```

where the *measured* predicate is used to prevent the same observation actions being selected multiple times.

Plan Generation

Given a quasi-deterministic planning problem we generate contingent plans where each branch point in a plan is associated with one possible outcome of an observation action. We take the approach of Kuter et al. [9] to generate contingent plans. First, we determinise the problem according to single-outcome determinisation [10]. For each observation action, only the most likely probabilistic effect is chosen. Similarly, only the most likely state from the initial belief state is used to define the initial state of the determinised problem. This converts a quasi-deterministic planning problem into a classical deterministic model. We use the classical planner FF

Algorithm 1 Generating the contingent plan using FF

plan=FF(*initial-state*,*goal*)
while *plan* contains observation actions without branches **do**
  Let o be an observation action without a branch such that all such actions preceding o have branches
  Let v be the variable observed at o and v_1 be the value obtained
  Let s be the belief state after executing all actions preceding o from the initial state
  for each value v_i, $i \neq 1$ of v with non-zero probability in s **do**
    branch = FF($s \cup (v = v_i)$,*goal*)
    Insert *branch* as a branch at o
  end for
end while

[11] to generate a plan from the determinised initial state[1].

Since each observation action in the plan could have outcomes other than the one selected in the determinisation, we then traverse the plan updating the initial state as we go until an observation action is encountered. This forms a branch point in the plan. For each value v_i of the observed state variable V apart from the one already planned for, we generate a new initial state for FF by adding $V = v_i$ to the state found by traversing the plan to this point, and call FF to generate a new branch as shown in Algorithm 1.This repeats until all observation-making actions have branches for every value of the observed variable.

IV. Execution Monitoring for Contingent Plans

Our approach for generating branching plans relies on relaxation of the uncertainty in the initial states and observation actions. The results of this are plans that account for every state the world might be in but do not account for the unreliable observations. To overcome this, we use execution monitoring. During execution, we update the agent's belief state after each selected action via Bayes rule. To select actions to perform when we reach an observation-making action in the plan, we utilise a VoI calculation [12]: Suppose the plan consists of state-changing action sequence a_1, followed by observation action o_1, which measures state variable c. If c is true, branch T_1 will be executed, and if c is false, branch T_2 will be executed. When execution reaches o_1, execution monitoring calculates the expected utility of the current best branch T^* based on the belief state $b(c)$ over the value of c after a_1 as follows:

$$U_b(T^*) = \max_{T_i} U(T_i, b) \tag{1}$$

where $U_b(T^*)$ represents the value, given belief state b, of making no observations and simply executing the best branch. $U(T_i, b)$ is the expected value of executing branch T_i in belief state b.

Next we examine the value of performing an observation-making action o (not necessarily the same o_1 as planned) that gives information about c. Performing o will change the belief state depending on the observation that is returned. Let B be the set of all possible belief states after executing o (one for each possible observation), and let $P(b')$ be the probability of getting an observation that produces belief state $b' \in B$. Let cost(o) be the cost[2] of performing action o. The VoI gained by performing o, is the value of the best branch to take in each b', weighted by the probability of b', less the cost of performing o and the value of the current best branch:

$$\text{VG}(o) = \sum_{b' \in B} P(b') U_{b'}(T^{b'}) - \text{cost}(o) - U_b(T^*) \tag{2}$$

Where $T^{b'}$ is the best branch to take given belief state b':

$$U_{b'}(T^{b'}) = \max_{T_i} U(T_i, b') \tag{3}$$

Equations 1 and 3 require computing the utility of executing a branch of the plan, $U(T, b)$. Building the complete contingent plan allows us to estimate this when selecting observation actions. We do this by a straightforward backup of the expected rewards of each plan branch given our current belief state. The value of $U(T, b)$ (the utility of branch T in belief state b is computed as follows:

- if T is an empty branch, then $U(T, b)$ is the reward achieved by that branch of the plan.
- if T consists of a state-changing action a followed by the rest of the branch T', then $U(T, b) = U(T', b) - \text{cost}(a)$.
- if T consists of an observation-making action o on some variable d where if $d = d_i$ then branch T_i will be executed (observation-making actions for each variable will appear at most once), then $U(T, b) = \sum_d b(d_i) U(T_i, b) - \text{cost}(o)$, that is, we weight the value of each branch at o by our current belief about d.

This ability to estimate the value of each branch is in contrast to the alternative approach of replanning (e.g. see [2], which we discuss in more detail in Section VII) where the utility of the future plan is impossible to determine since it hasn't yet been generated. Even in our case, we cannot compute this value exactly as we don't know what additional actions execution monitoring will add to the plan. However, since all the observation-making actions for some variable p look identical to the contingency planner (they only reveal the value of p) apart from their cost, the planner will choose the minimum cost one. This means that the cost we estimate for the tree will be an underestimate so execution monitoring will never perform fewer observational actions than are needed. The execution monitoring algorithm is given in Algorithm 2.

We select observation-making actions greedily so the sequence of actions may not be optimal. However, the unconstrained action selection problem satisfies the requirements for sub-modularity [13] which guarantees the greedy approach is close to optimal.

The description above assumes that there is at least one observation-making action associated with each uncertain variable. If this is not the case for some variable p, when

[1]If no plan is found for the given determinisation, we simply try a different one—all that is required is an initial plan we can construct the branching plan from.

[2]Costs are represented as negative rewards; we use cost here only to emphasise the contribution of the action to the VoI.

Algorithm 2 Execution monitoring at observation-making action o

```
Let c be the variable being observed by o
Let A be the set of actions that provide information about c
repeat
    Let VG(a) be the value gain for a ∈ A according to Eq. 2
    Let a* = arg max_a VG(a)
    if VG(a*) > 0 then
        execute a* and update the belief state b' based on the observation returned
    end if
until VG(a*) ≤ 0
Execute the best branch in the new belief state b' using Eq. 3
```

we create the domain for the contingency planner, we add a dummy action that reveals the value of p. At execution time, if a branch point on p is reached, the execution monitoring searches for any observation-making action that will provide information about p, so will substitute real actions for the dummy one if there are any available, and if not will simply execute the best branch in the current belief state. We do the same in the MDP-based approach described below.

V. MDP Planning Approach

In the standard *RockSample* problem, observations of the rocks get less reliable the further away from the rock they are made. Our approach of treating the observation actions as completely reliable ignores this, so the plans we find observe all the rocks from the starting position rather than driving closer to make better observations. This is an example of a common feature of many quasi-deterministic problems: an observation-making action that requires a state changing setup step. The approach we described in Section III performs particularly poorly for domains where these are present because execution monitoring cannot choose to do the setup action. This is because changing the state of the world might invalidate the rest of the plan. In this section we present an alternative that replaces the contingency planner with an MDP planner. This has the advantage that the policy specifies an action to perform in *every* state so we can execute state-changing actions at run-time and the policy still tells us what action to execute next. However, MDP planning does not scale as well as contingency planning.

A naive approach to using an MDP planner would be to convert the quasi-deterministic POMDP into an MDP by simply deleting the observations from the model. The problem with this is that it makes the observation-making actions into NOOPs, so they will never appear in the policy. In the case of the *RockSample* domain, this results in a plan where a good rock is immediately sampled and no others are examined. We would prefer a plan where each rock is investigated, and to do this the MDP planner needs a notion of what it does and does not know. To achieve this we treat observation actions as actions that switch their respective variables from an $unknown$ to a $known$ state, and set all the variables that are initially uncertain to be $unknown$ in the MDP initial state. So for each variable p with domain D, the corresponding variable in the MDP domain, p', has as its domain $D \cup unknown$, and in the initial state $p' = unknown$. To define the transition function from $unknown$ to each value $d_i \in D$ we use the probability of getting observation z_i that corresponds to value d_i of p if we performed the action in the initial state:

$$P(unknown, ., d_i) = \sum_D b(d_i) P(z_i|d_i) \quad (4)$$

Note that this translation assumes that the true value of a state variable is known after performing the corresponding observation action once. However, in reality the observation actions are still noisy, so execution monitoring is still required to improve the plan.

VI. Monitoring for MDP Policies

For the MDP case the execution monitoring is largely the same as in the contingency plan case described in Section III. The value of information calculation is exactly that given in Equations 1 to 3. The difference comes in the definition of the "branches" in the plan. For the MDP, the branches being chosen are the actions according to the MDP policy for the states corresponding to the possible values of the variable being observed. That is, when we use an observation-making action for a binary variable p with domain D_p (which must be *unknown* in the current MDP state or the policy wouldn't choose an observation-making action), the policies in all MDP states in $\{s - (p = unknown) \cup (p = x) : x \in D\}$ are evaluated. This means that to select actions correctly we maintain both the belief state according to the original quasi-deterministic POMDP and the current state according to the MDP.

Observation Actions with Setup Steps

As we said above, the major difference between the output of the contingent planner and the MDP planner is that the MDP planner provides an action for every possible state. This means that we can allow execution monitoring to perform state changing actions, and we will still know what policy to perform in the state that results. This is important because it allows us to make observations that require setup actions. To achieve this we allow execution monitoring to evaluate the information gain from macro actions consisting of a state changing action followed by an observation-making action. For example, in the *RockSample* domain this allows execution monitoring to calculate the value of information gained from moving one step towards a rock before observing it. We use Equation 2 as before, but the value of a macro action made up of a state changing action a followed by an observation-making action o becomes:

$$VG_{b'}(a + o) = R(b', a) + VG(o) \quad (5)$$

Once we decide the current best action, two situations need to be considered: If the current best action is a macro action, and hence the immediate action to execute is a state-change action, we execute the state changing action and repeat the above procedure. Since it's possible that a macro action will again be best, this allows us to execute a sequence of state changing actions before making an observation. If the current

Algorithm 3 Execution Monitoring with Macro Actions

Let the current MDP state be $s \in S$
Let o be the observation-making action which is the policy for s, where the observation is of variable V
Let b be the current belief state
repeat

$$a^* = \arg\max_a \begin{cases} VG(a) \text{ if } a \text{ is an observation action} \\ R(b,a) + VG(o') \text{ if } a \text{ is a macro action } (a, o') \end{cases}$$

  if a^* is a macro action (a, o) **then**
    Execute action a and update the MDP state s and the belief state b
  else if $VG(a^*) \geq 0$ **then**
    Execute action a, getting observation Z
    $b = \text{beliefUpdate}(b, a, Z)$
  end if
until $VG(a^*) < 0$
Let $\{s' : s' = s - (V = unknown) \cup (V = v)\}$
Let π_s be the policy at state $s \in S$
Let a^* be the first action in policy π^* where:

$$\pi^* = \arg\max_{\pi_s} \sum_c b(c) U(\pi_s, c)$$

where c is the POMDP state space
Execute action a^*
$b = \text{beliefUpdate}(b, a^*)$

best action is an observation action, this tells us that there is no better macro action. If this action has value gain greater than zero, we again execute it and repeat. Otherwise, we pick the best policy given our current belief state as described above. The algorithm is given in detail in Algorithm 3.

VII. Related Work

Many execution monitoring approaches [14], [15], [16] have been developed to detect and fix discrepancies between the actual world state and the agent's knowledge of world. In most cases the discrepancies these approaches are trying to detect result from exogenous events or action failures. In addition, most of these approaches are monitoring the execution of straight-line plans. A good survey can be found in [17]. However, as none of them are addressing the same problem of partial observability as we investigate, their approaches are not comparable with ours.

One related approach is [18], which also performs execution monitoring on MDP policies at run time. Fritz monitors plan optimality and identifies when a discrepancy in the plan is relevant to the quality of the plan. The other closely related work is [19], which similarly uses classical planning plus execution monitoring to solve problems that could be represented as POMDPs. In that work execution monitoring is used to check the preconditions of actions rather than to determine which branch to take. They also use value of information to measure whether monitoring is worthwhile, but formulate the monitoring decision problem as a set of POMDPs, rather than using the value of information directly.

An alternative to execution monitoring for solving quasi-deterministic problems efficiently is described in [2]. There they use a classical planner and a decision-theoretic (DT) planner to solve these problems, switching between them during planning. The approach is similar to ours in that they use FF to plan in a determinisation of the original problem augmented with *assumption actions* that set the value of unknown variables. However, they build linear plans with FF and switch to the DT planner to replace the assumption actions with small policies. The idea is to use the DT planner where the world is uncertain and the classical planner elsewhere.

Classical planners have also been applied in fully observable MDP domains—the best known is FF-replan [10], from which we have taken the determinisation approach. Because these approaches rely on being able to determine the state after each action, they cannot easily be applied in POMDPs.

VIII. Experimental Evaluation

We perform experiments in simulation on a modified version of the *HiPPo* domain [3] and on the *RockSample* domain [1]. We also tested a version of *RockSample* on a physical robot. For comparison in the simulations we use symbolic Perseus [7], a state-of-the-art point-based POMDP solver for structured representations. This is only approximately optimal but repeated trials suggest the policies reported are close to optimal. We use FF [11] and SPUDD [20] respectively to generate contingency plans and MDP policies. The standard *RockSample* problem allows observing any rock from any position with noise depending on the distance between the rover and the rock. Representing this in symbolic Perseus causes a blowup in the number of actions compared with PPDDL which makes comparison unfair so we only test contingent plan execution monitoring in the *HiPPo* domain and on the physical robot. Allowing macro actions does not change the policy in *HiPPo*, so we only report results from *RockSample*. We also compared with Q_{MDP} [21] in both domains. However, since the observation actions do not change the state, Q_{MDP} never includes them in plans, so performs much worse than the other approaches. To measure plan quality, we measure average total reward and discounted reward over multiple runs. On the *RockSample* domain, evaluation is done over 200 runs, each of 200 steps. Since *HiPPo* domains have a larger belief space and do not return to the initial state after reaching the goal, we performed 1000 runs of 20 steps each except for *HiPPo*(5,4) where only 100 runs were performed due to long run-times.

HiPPo

We first tested our approach in a modified *HiPPo* domain where a robot is able to answer queries, such as "where is the red triangular object". Each object in the domain has both colour and shape properties, and there are five different values for each. Two noisy observation actions, Colour and Shape, can be applied to collect information about the objects. The question is how to apply the colour and shape detecting actions before answering the user's queries. We extended the problem by putting the objects in a grid map, and making sensing actions usable only when the robot is at the same position as the object. Therefore, in order to decide which object is the answer to the query, the agent needs to move to the object's location and apply the observation actions. *HiPPo*(n, k) denotes a n by n grid with k objects,

TABLE I: Results for the *HiPPo* Domains comparing symbolic Perseus (POMDP) with the MDP and the contingency planning (FF) approaches.

Algorithm	Time (s)		Total Reward	Disc. Reward
	Plan	Exec.		
HiPPo (3,2)				
POMDP	196.25	0.26	-7.35 ± 29.9	-2.13 ± 12.0
MDP with EM	1.04	1.42	-7.45 ± 17.3	-2.28 ± 9.5
MDP, no EM	1.04	0.06	-6.57 ± 15.4	-2.83 ± 9.8
FF with EM	4.04	1.32	-7.04 ± 17.3	-3.04 ± 8.9
FF, no EM	4.04	0.06	-7.41 ± 16.9	-2.90 ± 9.3
HiPPo (4,3)				
POMDP	4059	9.95	-3.26 ± 11.5	-1.21 ± 10.6
MDP with EM	11.88	4.03	-3.75 ± 23.3	-1.21 ± 11.43
MDP, no EM	11.88	0.13	-5.78 ± 21.5	-1.69 ± 12.5
FF with EM	8.05	3.48	-5.79 ± 18.8	-1.87 ± 9.8
FF, no EM	8.05	0.12	-5.95 ± 18.5	-1.87 ± 9.7
HiPPo (5,4)				
POMDP	-	-	-	-
MDP with EM	207.65	56.74	-3.60 ± 27.5	-0.61 ± 11.1
MDP, no EM	207.65	0.46	-2.31 ± 9.8	-1.42 ± 22.6
FF with EM	16.15	37.38	-3.24 ± 22.2	-0.80 ± 10.4
FF, no EM	16.15	0.44	-3.30 ± 20.7	-2.01 ± 9.4

which has $n^2 \times 5^{2k}$ states in total. The state changing actions in this domain consist of four actions to move the agent around the grid and one termination action per object which is used to state that the corresponding object is the answer to the query. A typical plan in this domain is to move to an object, then apply observation actions until the agent is sufficiently confident that the object is or isn't the desired one. If the object is the one desired, then the termination action for that object is executed, and otherwise the agent moves to another object. The movement and observation actions all have small negative rewards associated with them; if the termination action for object a is executed and a is the correct object, the agent gets a large positive reward, but if a is not the correct object, it gets a large negative reward. More details can be found in [3].

Since *HiPPo* domains have a large observation space, problems larger than size (4,3) cannot be solved by symbolic Perseus in reasonable time (two hours) and memory. As Table I shows, although symbolic Perseus managed to achieve the best plan quality in terms of discounted reward, it requires orders of magnitude more time in generating policies. In this domain the MDP policies are quite good (they run the observation actions once per object, while the optimal policy runs them multiple times to ensure reliability), so there is less improvement from adding the execution monitoring. FF is much faster than the MDP planner, but the plans are of poorer quality. Execution monitoring helps, but again is not as effective as with the MDP policies.

RockSample

There are five state-changing actions in this domain: four moving actions and one sampling action. Each rock has an observation action which is not perfectly reliable. A reward of 20 will be given if the rover samples a good rock and goes to the exit and a reward of -40 if a bad rock is sampled. A large penalty is given if there is no rock at the position of the rover when sampling or if the rover moves out of grid except to go to the exit. *RockSample*(n, k) denotes a n by n grid with k rocks, which has $n^2 \times 2^k$ states in total. We use a version of *RockSample* with costs on the actions: Move actions have cost two while all others have cost one.

TABLE II: Results for the *RockSample* Domain comparing symbolic Perseus (POMDP) with the MDP approach.

Algorithm	Time (s)		Total Reward	Disc. Reward
	Plan	Exec.		
RS (4,4)				
POMDP	274	0.6	251.8 ± 69.1	6.9 ± 8.4
Macro Actions	13	3.5	161 ± 61.5	2.3 ± 9.5
EM	13	1.8	141 ± 61.3	1.9 ± 6.9
Without EM	13	1.1	41 ± 107.3	-3.5 ± 13.3
RS (5,5)				
POMDP	893	0.8	213 ± 66.8	4.5 ± 8.0
Macro Actions	99	4	179 ± 77.3	2.9 ± 10.9
EM	99	2.4	96 ± 70.2	-0.3 ± 8.0
Without EM	99	1.2	61 ± 100.5	-2.8 ± 13.3
RS (6,6)				
POMDP	1098	1.3	188 ± 55.5	3.4 ± 1.8
Macro Actions	476	5.0	150 ± 78.5	0.5 ± 9.3
EM	476	2.6	137 ± 73.3	1.9 ± 10.9
Without EM	476	1.7	58 ± 101.3	-2.1 ± 12.8
RS (7,7)				
POMDP	3520	2.4	154 ± 52.1	2.6 ± 8.5
Macro Actions	2096	7.2	147 ± 82.3	-0.2 ± 10.7
EM	2096	3.5	125 ± 75.5	-0.7 ± 10.7
Without EM	2096	4.5	78 ± 106.9	-2.2 ± 13.4

(a) (b)

Fig. 1: (a) The boxes used for the search task. The similarity of the centre two leads to many identification errors. (b) The turtlebot examining a box during execution.

As Table II shows, symbolic Perseus unsurprisingly again obtained the best plans in terms of total reward and discounted reward for all *RockSample* problems. The MDP solutions required much less time but only achieve 32% of optimal on average. Execution monitoring substantially improves plan quality to 65% of optimal, and macro actions further improve performance to an average of 81% of optimal. Although it requires more computation at run time, the overhead of execution monitoring with macro actions is still far less than solving the POMDP directly.

Mobile Robot

For the physical robot we used a TurtleBot as shown in Figure 1 (a), running ROS, with an ASUS Xtion depth sensor. The task for the robot was to find a particular object of interest in a grid similar to the *RockSample* domain. We used the boxes shown in Figure 1 b as the objects and SIFT [22] for recognition. Training and test images were collected using the robot, features extracted, and optimal thresholds to minimize errors determined. We then populated

the confusion matrix $P(\text{observation} = \text{box}_i | \text{ box}_j)$ between the objects.

We only tested the contingency planner on the robot. Plan generation time was less than 0.1 seconds (compared with around 60 seconds for Symbolic Pereus), and time for execution monitoring is insignificant compared with time spent running SIFT. The plan found is to visit each box in turn, use SIFT to see if it is the one of interest (as shown in Figure 1 (c)), and if so take an image and return to the start point. Execution monitoring then adds additional SIFT actions (the action includes taking a new image) to increase the probability of the box being correctly identified. As might be expected, more images are used to distinguish the centre two boxes in Figure 1 (b) than the others (the number of images varies depending on the reward model for the planner and the sequence of observations made—a false positive or negative observation leads to extra SIFT actions).

IX. Conclusion

We have presented an approach to approximately solve quasi-deterministic POMDPs by converting them into contingency planning problems or MDPs. At run time, we use execution monitoring to repair the plans in order to increase the plan quality by making the belief state more certain. A value of information approach is applied to monitor the belief state and choose observation actions to gain information about the true state. Our experimental results show that our approach is fairly close to optimal but is orders of magnitude faster than using even a state of the art POMDP solver.

Acknowledgements

This work was supported by the UK Engineering and Physical Sciences Research Council's Autonomous Intelligent Systems Programme project "Sustained Autonomy through Coupled Plan-based Control and World Modelling with Uncertainty".

References

[1] T. Smith and R. Simmons, "Heuristic search value iteration for POMDPs," in *Proceedings of the 20th conference on Uncertainty in artificial intelligence (UAI)*, 2004.

[2] M. Goebelbecker, C. Gretton, and R. Dearden, "A switching Planner for Combined Task and Observation Planning," in *Proceedings of the 25th Conference on Artificial Intelligence (AAAI)*, 2011.

[3] M. Sridharan, J. Wyatt, and R. Dearden, "HiPPo: Hierarchical POMDPs for Planning Information Processing and Sensing Actions on a Robot," in *the 18th International Conference on Automated Planning and Scheduling (ICAPS)*, 2008.

[4] C. Besse and B. Chaib-Draa, "Quasi-Deterministic Partially Observable Markov Decision Processes," in *Proceedings of the 16th International Conference on Neural Information Processing*, 2009.

[5] B. Bonet, "Deterministic POMDPs revisited," in *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence (UAI)* 2009.

[6] R. Petrick and F. Bacchus, "A Knowledge-Based Approach to Planning with Incomplete Information and Sensing," in *The International Conference on Artificial Intelligence Planning and Scheduling*, 2002.

[7] P. Poupart, "Exploiting structure to efficiently solve large scale partially observable markov decision processes," Ph.D. dissertation, Department of Computer Science, University of Toronto, 2005.

[8] H. L. S. Younes and M. Littman, "PPDDL1.0: The language for the probabilistic part of IPC-4," in *Proceedings of the International Planning Competition*, 2004.

[9] U. Kuter, D. S. Nau, E. Reisner, and R. P. Goldman, "Using classical planners to solve nondeterministic planning problems," in *Proceedings of the 18th International Conference on Automated Planning and Scheduling, ICAPS 2008*, 2008.

[10] S. W. Yoon, A. Fern, and R. Givan, "FF-Replan: A Baseline for Probabilistic Planning," in *Proceedings of the 14th International Conference on Automated Planning and Scheduling*, 2007.

[11] J. Hoffmann and B. Nebel, "The FF planning system: Fast plan generation through heuristic search," *Journal of Artificial Intelligence Research*, vol. 14, 2001.

[12] R. Howard, "Information value theory," *Systems Science and Cybernetics, IEEE Transactions on*, vol. 2, no. 1, 1966.

[13] A. Krause and C. Guestrin, "Near-optimal observation selection using submodular functions," in *National Conference on Artificial Intelligence (AAAI), Nectar track*, 2007.

[14] R. E. Fikes, P. E. Hart, and N. J. Nilsson, "Learning and executing generalized robot plans," *Artificial Intelligence*, vol. 3, 1972.

[15] G. D. Giacomo, R. Reiter, and M. Soutchanski, "Execution monitoring of high-level robot programs," in *6th International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 1998.

[16] M. M. Veloso, M. E. Pollack, and M. T. Cox, "Rationale-based monitoring for planning in dynamic environments," in *4th International Conference on Artificial Intelligence Planning Systems*, 1998.

[17] O. Pettersson, "Execution monitoring in robotics: A survey," *Robotics and Autonomous Systems*, vol. 53, no. 2, 2005.

[18] C. Fritz, "Monitoring the generation and execution of optimal plans," Ph.D. dissertation, University of Toronto, 2009.

[19] C. Boutilier, "Approximately optimal monitoring of plan preconditions," in *In Proceedings of the 16th Conference in Uncertainty in Artificial Intelligence (UAI)*, 2000.

[20] J. Hoey, R. St-Aaubin, A. Hu, and C. Boutilier, "SPUDD: Stochastic Planning using Decision Diagrams," in *In Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI)*, 1999.

[21] M. Littman, A. R. Cassandra, and L. P. Kaelbling, "Learning policies for partially observable environments: Scaling up," in *Proceedings of the 12th International Conference on Machine Learning*, 1995.

[22] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, 2004.