DYNAMIC PLANNING PROBLEM GENERATION IN A UAV DOMAIN

Per Nyblom [*]

[*] *Linköping University, Department of Computer and Information Science, Sweden, perny@ida.liu.se*

Abstract: One of the most successful methods for planning in large partially observable stochastic domains is depth-limited forward search from the current belief state together with a utility estimation. However, when the environment is continuous and the number of possible actions is practically infinite, then abstractions have to be made before any forward search planning can be performed. The paper presents a method to dynamically generate such planning problem abstractions for a domain that is inspired by our research with unmanned aerial vehicles (UAVs). The planning problems are created by first stating the selection of points to fly to as an optimization problem. When the points have been selected, a set of possible paths between them are then created with a pathplanner and then forward search in the belief state space is applied. The method has been implemented and tested in simulation and the experiments show the importance of modelling both the dynamics of the environment and the limited computational resources of the architecture when searching for suitable parameters in the planning problem formulation procedure.

Keywords: Unmanned Aerial Vehicles, Planning Problem Generation

1. INTRODUCTION

It is generally accepted within the AI community that continuous stochastic partially observable environments are very difficult to handle for autonomous agents. First of all, the representation of an agent's belief state that evolves when time passes by can generally not be represented exactly if very strong assumptions are not made (such as linearity and Gaussian representations of probability distributions (Kalman, 1960)). Secondly, any type of planning in such environments that involve solving the problem for *every* possible or reachable belief state is not applicable due to the continuous state variables.

Such findings have drawn researchers within the area to experiment with approximative representations of belief states such as particle filters (Doucet *et al.*, 2001) which has shown great success within the area of localization and simultaneous localization and mapping (SLAM) (Montemerlo *et al.*, 2002). For planning, forward search combined with good heuristics (Paquet *et al.*, 2005) seems to be a promising, but possibly computationally intensive, approach that assumes that the problem solving agent can perform the computation online.

For continuous domains, another problem arises when the agent performs planning. The number of possible actions in a certain situation could possibly be infinite and a number of simplifying assumptions must be made before such a problem can be stated as a forward search planning problem.

We will in this paper focus on how to dynamically formulate discrete planning problems suitable for forward search depending on an agent's current belief state in continuous domains. We have implemented a dynamic planning problem formulation procedure that is applied to a problem domain that is inspired by our research with unmanned aerial vehicles (UAVs) (Doherty, 2004). The main idea is to state parts of the planning problem formulation as an optimization problem in the space of possible sets of points to fly to from the agent's current position. The selected points are then transformed into a planning problem for a forward search planner that operates with belief states. The depth-limited forward search is combined with domain specific heuristics which is used at the cutoff depth.

2. PRELIMINARIES

In this section, we provide some preliminaries that briefly describe particle filters and forward search in belief states.

2.1 Particle Filters

Filters of different kinds are often used to represent an autonomous agent's belief state over time. In theory, a filter can use the Recursive Bayesian Estimation equations (Jazwinsky, 1970) to perform the necessary updates of the probability distributions. In practice, this update is not feasible in its pure form when the state contains non-linear and multi modal characteristics and some kind of approximation is necessary.

A common approximation is to represent the distribution as a set of so called *particles* which, in its most simple form, are full instantiations of the random variables in that distribution. The set of particles together with the update machinery is called a *particle filter*. This representation makes it possible to use any type of non-linear forward model to describe the dynamics of the domain.

The update rule for a particle filter can be implemented by resampling the particles depending on the current observation. The accuracy of a particle filter depends on the number of particles used and for probability distributions with many variables, a large number of particles may be required. More information about the different types of particle filters can be found in (Doucet *et al.*, 2001) and (Arulampalam *et al.*, 2002).

2.2 Planning and Forward Search

Automated planning is a general term for automatically generating plans or other types of solutions that determine what actions to perform, given some model of a task domain and a goal or preference measure. Automated planning is considered as one of the cornerstones of Artificial intelligence and is in general very difficult, especially in partially observable stochastic domains.

A common method to perform planning in such difficult domains is to use forward search from the current belief state (Paquet *et al.*, 2005). It is then possible to avoid generating solutions for every reachable belief state but with the extra cost of performing the planning online before each action execution. The search is cut off at a certain depth d where an approximative utility function U is used instead. At every search depth, all the applicable actions and possible observations are enumerated which is impossible for domains that contain continuous observations. Instead of enumerating all possible observations in continuous domains, it is possible to sample N_{obs} observations from the belief state. When the belief state is represented with a particle filter with normalized weights, the observation sampling can be performed by picking a particle at random and generating a random observation from it.

3. DOMAIN DESCRIPTION

Much of the work with this paper is inspired by our research with Vertical Takeoff and Landing (VTOL) UAVs and how efficient planning and execution can be combined in our system (Doherty, 2004). Our test domain is therefore a step towards a more autonomous system which can handle the stochastic and dynamic domains that we encounter during our different types of flight missions.

The problem domain consists of a freely moving agent in a continuous 2D environment (see Figure 1). The environment contains obstacles or zones where the agent is forbidden to fly. The agent can fly to any free point in the environment but must do so by first generating a stepwise linear path from its current position to the target. The path is generated by a roadmap based pathplanner, which is a simplified 2D version of the one we use in our current system (Wzorek and Doherty, 2006).

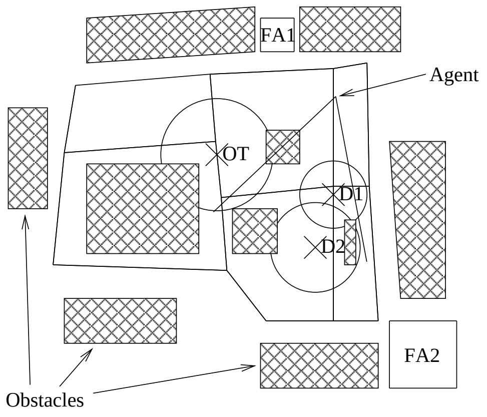

Fig. 1. A problem instance of the UAV domain. D1 and D2 are dangers and OT is an observation target. The circles around the external agents show the cost and classification radius. FA1 and FA2 are finish areas.

The environment can contain *finish areas* that are rectangular areas where the agent can safely stop its execution (corresponds to landing zones for our UAV).

Each finish area is associated with a certain reward that the agent receives if it chooses to stop there.

Other *external agents* can also move within the environment and they are partitioned into *observation targets* and *dangers*. These observation targets and dangers can either move freely or are restricted to move on certain paths that represent a road network. The main agent's task is to avoid the dangers and try to classify the observation targets, and this is done by specifying negative rewards for being close to dangers and positive rewards for classifying observation targets.

Each danger has a max cost C_{max} associated with it, which specifies the maximum negative reward that can be received during one second if the agent is visible from the danger and close to it. The cost decreases linearly with the distance until it reaches zero. That zero-cost distance is called the *cost radius* of the danger. Dangers with a negative C_{max} can be used to simulate targets that are supposed to be followed continuously by the agent.

The agent is equipped with a camera that can be used to track the external agents and perform classification. The camera is assumed to have a certain maximum view distance when it is used for such tasks.

The agent can try to classify an observation target. This operation is supposed to model a more detailed sensor action for a UAV that can be used to extract more information about a target than its position. The probability for success depends on the distance from the agent to the target and how long the classification is performed. The observation target must also be visible from the agent and within the camera's view area. Each observation target has a *classification radius* associated with it which determines the maximum range where the agent can possibly classify the target. The probability of a successful classification of an observation target ot from a distance d is modelled with a continuous-time Markov Process with only two possible states and with the intensity λ_d of going from "not classified" to "classified". The intensity decreases linearly from a maximum value to zero at the maximum classification distance.

The observation model that is used to update the belief state, is specified with likelihood functions. The likelihood for getting an observation from an object o at position p'_o when the true position is p_o and is visible from the agent's position, is assumed to be $max(\epsilon, P_{o,max} - \alpha d)$ where α is a positive constant, $P_{o,max}$ is the maximum probability of observing o (when the distance is zero) and d is the distance between p_o and p'_o. If p_o is not visible from the agent, the likelihood is set to ϵ which models spurious observations. When no observation is received, the likelihood function for visible states is a constant β and for invisible states γ where $\gamma > \beta$.

4. BELIEF STATE REPRESENTATION

The belief state for the agent is represented with particle filters, one for each external agent. Each particle for free flying objects is a tuple that contains the pose and velocity. Each particle for a road network bound object is a tuple containing the network link, the distance travelled on that link and the velocity. The agent's pose is assumed to be known.

The belief state update is performed by a sequential importance resampling (SIR) (described in (Arulampalam *et al.*, 2002)) together with the observation model described in Section 3.

5. DYNAMIC PLANNING PROBLEM GENERATION

In order to perform planning with the help of the output from the particle filter, a planning problem has to be constructed. This task has been divided into two parts. The first part is to select a set of "good" points to fly to given the current belief state. The second part consists of planning (with a path planner) the paths between the selected points to generate the planning problem specification. In this way it is possible to transform the extremely difficult problem of planning with the output from the particle filter directly into a tractable but simplified problem.

5.1 Point Selection

The first part of our problem generation in the UAV domain is to select the possible points that the agent should consider flying to. The point selection problem is stated as an iterative optimization problem by defining a utility measure for a point, given the current belief state and the previously selected points.

The utility of a point is assumed to depend on the distance to visible dangers, unclassified observation targets, whether the point is within a finish area and the possibility for the agent to move to that point. In our domain, where current state is partially observable, the *expected* utility of a point is used instead. When particle filters are used to represent the belief state of the agent, the expected utility can be considered to be the mean utility of the particles in the belief state used for point selection.

The utility $U(p)$ of a point p given a particle and a set of previously selected points, is divided into a sum of utility contributions from dangers, observation targets, finish areas and the previous points.

The utility contribution from a danger object, U_{do}, depends on the distance d from the particle to the agent:

$$U_{do} = min(-C_{max} + \frac{C_{max}}{C_R} d, 0) \qquad (1)$$

where C_R is the cost radius of the danger.

Similarly, the point utility for observation targets also depends on the distance but we also need to consider whether it has been classified previously or not:

$$U_{ot} = \begin{cases} R_{cl,ot} - \frac{R_{cl,ot}}{Cl_{R,ot}} d & d < Cl_{R,ot} \text{ and } \neg cl_{ot} \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

where $R_{cl,ot}$ is the reward for classifying the target ot, $Cl_{R,ot}$ is the classification radius and cl_{ot} the boolean variable that specifies whether ot has been classified previously or not. Notice that observation targets that have been previously classified provide nothing to the point utility.

The utility contribution U_{fa} from a finish area fa is the same as the corresponding finish reward if the point is within fa and the agent has not executed the finish action yet.

To provide a simple way to create diversity of the selected points, the point selection takes the previously selected points into account. We use a penalty function $U_{\mathbf{p}}$ for the set of previously selected points $\mathbf{p}$ that depends on the distance between the considered point with one exception: If the newly selected point is located within a finish area and no other point is, no penalty is given.

The sum of $\sum_{do} U_{do}$, $\sum_{ot} U_{ot}$, $\sum_{fa} U_{fa}$ and $U_{\mathbf{p}}$ is then used during optimization with a random restart hillclimbing search where the neighbourhood function generates a set of points that are within 5, 10 and 20 meters away. A point within the closest finish area is also added to a point's neighbourhood.

5.2 Problem Generation

When the set of points have been selected, paths are planned between every combination of distinct point pairs. This operation is very fast with our simplified 2D pathplanner, but it might be a bottleneck when our onboard pathplanner is used. The number of points also partly specifies the branching factor during forward search which means that there are two good reasons for keeping the set of selected points quite small.

Before the problem is considered ready, the number of paths are reduced to lower the branching factor for the forward search. Paths that contribute little or nothing are removed. In practice this filtering is performed by considering all triples of distinct points. If the length of a path between two points a and b is given by $l_{a,b}$, then if $l_{a,b} + l_{b,c} < \alpha \cdot l_{a,c}$, the path from a and c is removed from the problem model. α is in our implementation set to 1.1.

Figure 2 shows an example of how a dynamically generated problem can look like given the utility function calculated from the belief state of the agent.

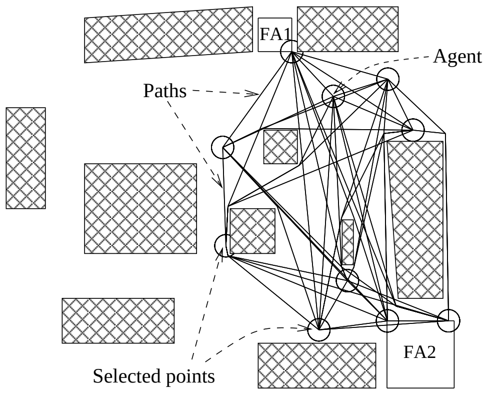

Fig. 2. A generated problem for forward search given a point value function calculated from a belief state of the agent. The selected points are drawn with circles. The road network is hidden for clarity.

6. PLANNING AND EXECUTION

This section describes how the planning and execution is performed given a problem model.

6.1 Forward Search

The planning in the problem model is done by forward search from the current belief state. The problem model is unfortunately not suitable for direct application of the forward search idea since by simply considering the flight from one point to another as a primitive action without modification ignores all the possible observations received *during* the execution.

We have choosen to use a fix step size τ_{plan} during planning and sample a number of possible observation sequences to approximate the enumeration of all observations. Another simplification during the search is that the camera's view area is a circle around the agent's position, but still taking obstacles into account. This simplification was done because considering all possible camera directions during search would increase the branching factor too much. The camera pointing actions can be planned in a more refined problem specification (assuming that the path is already given) or defined with a preprogrammed behavior module. The latter is the method we use in our current implementation (see Section 6.2).

6.2 Execution

Actions that are returned from the planner are executed by behaviors which emulate the behavior generating components used in our current robotic system (Doherty, 2004).

Paths, generated from the pathplanner, are followed in the straightforward way, assuming that the agent is a point with no dynamics. The camera movement during execution is performed by a greedy camera control behavior which points the camera towards a direction that maximizes the expected *relevance* of the visible particles in the belief state. The relevance $R_{p_{do}}$ for a visible danger object particle p_{do} is calculated by $C_{max} \cdot e^{-\alpha_{p_{do}} \cdot d}$ where d is the current distance from the agent, C_{max} is the maximum cost for the danger and $\alpha_{p_{do}}$ is set so that $R_{p_{do}}$ has 10 percent of its value at the cost radius.

The camera is assumed to be capable of pointing instantaneously towards a selected point, independent of its previous angle.

6.3 Replanning

During execution, the belief state of the agent changes and eventually the dynamically generated problem specification becomes outdated. In this paper, we have used a simple replanning strategy where a new problem specification is generated every T_r seconds.

7. EXPERIMENTS

We have performed a set of experiments with our implementation that are aimed towards finding suitable design parameters for our UAV domain. The following parameters are varied in the experiments:

- Number of points that are selected during problem generation, N_{pg}
- Search depth for the forward search, d
- Number of sampled observation sequences for the forward search, N_{obs}
- Replanning period T_r
- Number of particles used for belief state during forward search, N_{pfs}
- Number of particles used for belief state during point selection, N_{pps}
- Whether simulated dynamic mode is used, SD (see Section 7.1)

We believe that the parameters should ideally change dynamically during execution depending on the current situation. For example, it should be obvious that large numbers for d and N_{obs} (which would yield a long planning time) should be avoided when the agent is situated within a danger's cost radius. However, in this paper we only investigate fixed parameter settings and delay the experimentation of dynamic settings for future work.

7.1 Experimental Setup

Most of the experiments were performed in a simulated dynamic mode which means that the environment is evolving during the agent's deliberation. The deliberation time is estimated by counting the most frequently and costly operations that are performed during point selection and planning. The two operations that are used for deliberation time estimation are the utility calculations of a point during point selection and planning, and the simulation step function that is used for prediction during planning. The time for those operations were first measured in our implementation and then assumed to be fixed during the experiments.

The belief state is represented with 500 particles for each external agent which seems to be more than enough in this particular domain. Using the full belief state for planning and point selection was not successful due to the extra overhead during point selection and planning and it is therefore always subsampled before any of these operations.

Every test result is averaged over 500 randomly generated environments which all have two dangers, one observation target and two finish areas (see Figure 1).

7.2 Experimental Results

Since it is not feasable to generate results for every possible configuration of the parameters described previously, we tested some configurations that point out some interesting behavior of our implementation. We first created a *default* configuration, with some trial and error, which is shown in Table 1. This configuration was then used as basis for our experiments when we changed a subset of the parameters.

N_{pg}	d	N_{obs}	T_r	N_{pfs}	N_{pps}	SD
7	1	8	2.0	5	5	Yes

Table 1. The default configuration.

7.2.1. Number of Particles One of the experiments that we performed was to vary the number of particles used during forward search and point selection. Since the dynamics of the environment is simulated, deliberation time is penalized both by the cost of hovering during planning but also with increased response times in dangerous situations. The result of the experiment is shown in figure 3 which demonstrates the importance of taking the dynamics and available computational resources into account. The best result was obtained when N_{pfs} was set to 2 and N_{pps} to 4 which was much lower than we expected.

7.2.2. Search Depth and Observations The default configuration uses a search depth of 1, which is rather extreme. But the best results were in fact obtained when this setting was used. Table 2 shows the result of an experiment when the search depth and number of observation samples are varied simultaneously. The result clearly indicates that a search depth of 1 should be used for this problem specification when the computational resources are taken into account. We believe that the best search depth also highly depends on

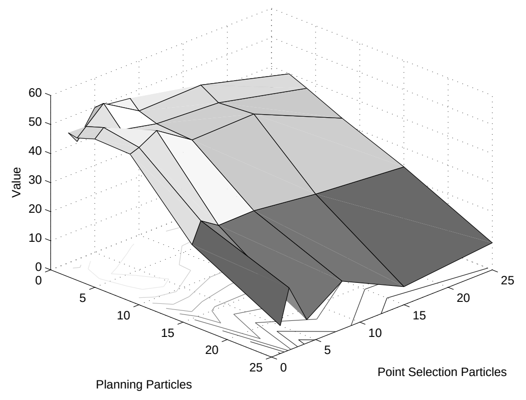

Fig. 3. The result when the number of particles for point selection and planning are varied.

		d			
		1	2	3	4
N_{obs}	1	53.58	18.88	-247.2	-2502
	2	53.71	-3.932	-2137	NA
	3	56.88	-64.09	NA	NA
	4	53.56	NA	NA	NA
	5	56.43	NA	NA	NA
	6	52.24	NA	NA	NA
	7	49.31	NA	NA	NA
	8	48.02	NA	NA	NA

Table 2. The results when the number of observations and search depth parameters are varied.

N_{pg}	d	N_{obs}	T_r	N_{pfs}	N_{pps}	SD	Value
10	1	15	1.0	50	50	No	59.39
20	1	15	1.0	50	50	No	62.46
20	1	20	1.0	200	200	No	64.30

Table 3. Three results when no simulated dynamics is used.

the problem formulation procedure, which in our case generates problems with very long temporal steps.

7.2.3. No Simulated Dynamics We also performed some tests when we disabled the simulated dynamics. Table 3 shows the three different configurations that we used together with their corresponding results. The results are clearly better than than the best result when simulated dynamics is used (56.88) but as the table shows, the number of points selected and particles used are much higher and it requires a lot more computation.

8. CONCLUSION

We have in this paper developed a method to dynamically create problem specifications depending on the current belief state in a continuous partially observable stochastic domain inspired by our research with unmanned aerial vehicles. Our experiments shows that the best results are obtained when a surprisingly inaccurate model (very few particles) is used during planning and problem generation. This result demonstrates the importance of modelling both the dynamics of the environment and the architecture's computational resources.

We will in our further work both try to make the domain more and more realistic and make it work for real in our current UAV system. An important step on the way is to move from a purely reactive camera control to a full camera motion planner. We also need to consider 3D environments and the resulting computational aspects when our real on-board path-planner is used instead. Another important extension is to perform the planning *during* execution instead of first braking and then doing it while hovering.

ACKNOWLEDGMENTS

This research has been supported by funding from a Swedish National Aeronautics Council grant (NFFP4-S4203) and a Swedish Research Council grant 621-2005-3642.

REFERENCES

Arulampalam, S., S. Maskell, N. Gordon and T. Clapp (2002). A tutorial on particle filters for on-line non-linear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing* **50**(2), 174–188.

Doherty, P. (2004). Advanced research with autonomous unmanned aerial vehicles. *Proceedings on the 9th International Conference on Principles of Knowledge Representation and Reasoning*.

Doucet, A., N. Freitas and N. Gordon (2001). *Sequential Monte Carlo Methods in Practice.*. Springer Verlag New York.

Jazwinsky, A.M. (1970). *Stochastic Processes and Filtering Theory.*. Academic, New York.

Kalman, R. (1960). A new approach to linear filtering and prediction problems.. *Journal of Basic Engineering* **82**, 35–46.

Montemerlo, M., S. Thrun, D. Koller and B. Wegbreit (2002). FastSLAM: A factored solution to the simultaneous localization and mapping problem. In: *Proceedings of the AAAI National Conference on Artificial Intelligence*. AAAI. Edmonton, Canada.

Paquet, S., L. Tobin and B. Chaib-draa (2005). Real-time decision making for large pomdps. In: *Proceedings of the Eighteenth Canadian Conference on Artificial Intelligence*.

Wzorek, M. and P. Doherty (2006). Reconfiguarable path planning for an autonomous unmanned aerial vehicle. In: *Proceedings on the 16th International Conference on Automated Planning and Scheduling*. pp. 438–441.