IJCAI-05 Workshop

Reasoning with Uncertainty in Robotics (RUR-05)

Edinburgh, Scotland, 30 July 2005

Workshop Notes

Preface

This volume contains the workshop notes of the *Reasoning with Uncertainty in Robotics* (RUR-05) workshop, held at the IJCAI-05 International Joint Conference on Artificial Intelligence, in Edinburgh, Scotland, on 30 July 2005. This RUR-05 workshop is the 5th in the series of RUR workshops (four of them at IJCAI), bringing together researchers from the AI and the robotics community to discuss issues related to robotics and artificial intelligence, and in particular on ways to deal with uncertainty. This year RUR-05 has focussed on the following topics:

– Representations for dealing with uncertainty in robotics.
– Planning in continuous state/action/observation spaces.
– Simultaneous localization and map building (SLAM).
– Single- and multi-robot planning under motion and sensor noise.
– Simultaneous planning, localization, and map building (SPLAM).
– Successful applications.

We would like to thank the authors for their interest in the workshop, as well as the members of the program committee for their help in reviewing papers and providing useful comments:

Henrik Christensen (Royal Institute of Technology, Sweden)
Gregory Dudek (McGill University, Canada)
Eric Hansen (Mississippi State University, USA)
Joachim Hertzberg (University of Osnabrueck, Germany)
Ronald Parr (Duke University, USA)
Pascal Poupart (University of Waterloo, Canada)
Stergios Roumeliotis (University of Minnesota, USA)
Nicholas Roy (MIT, USA)
Alessandro Saffiotti (Orebro University, Sweden)
Luis Enrique Sucar (ITESM, Mexico)
Sebastian Thrun (Stanford, USA)

We would also like to thank Aristeidis Diplaros for his help in preparing these notes.

The RUR-05 organizers:
Nikos Vlassis, University of Amsterdam, The Netherlands
Geoff Gordon, Carnegie Mellon University, Pittsburgh PA, USA
Joelle Pineau, McGill University, Montreal, Quebec, Canada

Contents

A Bayesian Approach for Place Recognition

Fabio T. Ramos Ben Upcroft Suresh Kumar Hugh F. Durrant-Whyte

ARC Centre of Excellence for Autonomous Systems
Australian Centre for Field Robotics
The University of Sydney
Sydney, NSW 2006, Australia
{f.ramos,b.upcroft,suresh,hugh}@acfr.usyd.edu.au

Abstract

This paper presents a robust place recognition algorithm for mobile robots. The framework proposed combines nonlinear dimensionality reduction, nonlinear regression under noise, and variational Bayesian learning to create consistent probabilistic representations of places from images. These generative models are learnt from a few images and used for multi-class place recognition where classification is computed from a set of feature-vectors. Recognition can be performed in near real-time and accounts for complexity such as changes in illumination, occlusions and blurring. The algorithm was tested with a mobile robot in indoor and outdoor environments with sequences of 1579 and 3820 images respectively. This framework has several potential applications such as map building, autonomous navigation, search-rescue tasks and context recognition.

1 Introduction

Localisation in complex environments is one of the main challenges for autonomous navigation. Currently, this task is performed by solving the simultaneous localisation and map building problem (SLAM) [7]. The main issue of this approach is the identification of landmarks in unstructured environments - they have to facilitate detection and association. On the other hand, rather than navigating using relative coordinates, humans have an abstract notion of distance but are still able to recognise where they are in the space. This ability is provided mainly from visual information associated with an internal (map) representation that, rather than localising the person with distances from objects, localises it based on the appearance of that scene. The same approach is used here for robot localisation in two different situations, indoor and outdoor environments. Places are learnt and identified from images obtained with a camera fixed in a mobile robot. The aim is thus to learn a multi-class classifier to label new images as the robot navigates. No other extra information such as a topological map is provided.

Place recognition and localisation from images are not new in robotics. Some previous approaches use image histograms and topological maps for classification [21, 14]. Others use invariant features such as [11, 22]. Place recognition was also shown to improve the performance of object recognition when both are performed simultaneously [20]. In another approach, image features were used to estimate the position of the robot for map building problems [17, 18, 5]. The novelty of this work compared to the others lies on the proposed method that allows robust recognition from few training images (usually 3 to 10 per place) and without the need of a map. Because the robot needs fewer training images, a quick learning exploration is enough for further localisation tasks. In addition, the framework was tested in both indoor and outdoor environments proving to be robust for practical applications.

In the solution proposed, the world is interpreted as a set of places. Each place has a probabilistic representation learnt from images and localisation is performed in near real-time by evaluating the responses of each model given a new image. The place recognition task is treated as a Bayesian learning problem in a space of *essential features*. Initially, training images are divided into small patches that constitute a high dimensional set. The dimensionality of this set is then reduced with nonlinear and neighbourhood

preserving techniques to create a low dimensional set. These two sets are used to learn a mixture of linear models for the nonlinear regression, from points in the high to the low dimensional space. Points is this low dimensional space constitute the set of *essential features* and are used in the next step where the variational approximation for Bayesian learning is computed to create a probabilistic density for each place. Recognition is performed by computing the log-likelihood of an entire image over each place model. This approach was tested with sequences of images obtained by a mobile robot operating under different conditions - moving objects, changes in illumination, different viewpoints, occlusions, outdoor and indoor environments - and proved to be robust for localisation.

2 Approach

This paper focuses on a classification procedure to map images to labels. Each label corresponds to a place learnt from a set of images. The learning algorithm is supervised as every image in the training set has an assigned label. Thus, given a training set of n pairs $(\mathbf{I}_i, p_i)$, where $\mathbf{I}_i$ is the ith image and p_i is the label of that particular image, the algorithm has to generate a model able to classify new images regardless whether they come from different view points or are partially occluded.

Images are represented as sets of patches of the same size that, when combined, recover the original image. Thus an image $\mathbf{I}_i$ is represented as a set of m patches $\{I_{i,1}, \ldots, I_{i,m}\}$. Each patch is convolved with a sequence of Gabor wavelets to quantify texture. This convolution has also a natural interpretation as it provides a good approximation of natural processes for spectral decomposition that occurs in the primary visual cortex. Each patch now has a feature-vector representation $x_{i,j} = \phi(I_{i,j}) \in \mathbb{R}^D$. The dimension D is usually intractable for direct density estimation. Hence, dimensionality reduction techniques are applied to extract the *essential* information of each patch and represent them in a lower dimensional space. However, this reduction should preserve important characteristics of the original space such as keeping the neighbourhood of points unchanged. This ensures that patches representing trees and grass for example are situated nearby and not mixed up with other patches even when lying in a nonlinear surface.

2.1 Neighbourhood-Preserving Dimensionality Reduction

Dimensionality reduction is one of the techniques that can manage the amount of information robotics application face. In this work, a nonlinear technique, Isomap [19], is applied to reduce the dimensionality of image patches into a feasible number where further statistical learning methods can be used. As opposed to principal components analysis (PCA) [8] and multidimensional scaling (MDS) [4], Isomap has the desired property of preserving the neighbourhood of points in the low dimensional manifold.

2.2 Noisy Non-Linear Regression

Isomap and indeed most nonlinear dimensionality reduction algorithms are inherently deterministic algorithms that do not provide a measure of uncertainty of underlying states of high dimensional observations. In addition, Isomap does not output a model or function to directly compute the low dimensional coordinates of new observations, thus requiring k-neighbours based algorithms that can be cumbersome in real-time applications.

An alternative solution is to learn a generative model $p(x|y)$, where x is the feature-vector in the high dimensional space and y is its low dimensional representation. This model encapsulates the uncertainties inherent in the inference of low dimensional points from noisy high dimensional observations. It can be learnt in a supervised manner to derive compact mappings that generalise over large portions of the input and embedding space. The input-output pairs of Isomap can serve as training data for an invertible function approximator in order to learn a parametric mapping between the two spaces. Once the model is learnt, patches of new images can have their low dimensional representation according to the manifold of the training set. This is the key point to make real-time recognition since the essential features of a new image can be quickly computed from the model by making probabilistic inferences.

Given the results of Isomap, the parameters of a mixture of linear models $p(x, y, s)$ can be learnt through Expectation Maximisation (EM) [6], where s is a hidden discrete variable representing the weights

of the components. Mixture of linear models are similar to mixtures of factor analysers, that are commonly used to perform simultaneous clustering and local dimensionality reduction [9]. The only differences are that the low dimensional variable y is observed (through Isomap), not hidden, and the Gaussian distributions $p(y|s)$ have nonzero mean vectors ν_s and full covariance matrices Σ_s. Learning when the variable y is observed seems to discover a solution of better quality than in the opposite situation, the conventional mixture of factor analysers [16]. The discrete hidden variable s introduced in the model physically represents a specific neighbourhood on the manifold over which a mixture component is representative. This representation conveniently handles highly nonlinear manifolds through the capability to model the local covariance structure of the data in different areas of the manifold.

The result of the inference process in this model is a mixture of Gaussians with means μ, covariances Σ and weights s. To make it feasible for Bayesian learning this mixture is collapsed so as to have a single mean, which will be used as a training point, and a covariance matrix which will be used in the initialisation of the hyperparameters.

2.3 VBEM for Mixtures of Gaussians

The training data now represented with its essential features in the low dimensional space is used to learn a generative model for each place. This problem is formulated in a Bayesian framework where the model selection task consists of calculating the posterior distribution over a set of models (which in this case will be mixtures of Gaussians with different numbers of components) given the prior knowledge and the dataset. Denoting $\mathbf{s}$ for the hidden variable representing the weights, $\mathbf{y}_i$ for the observations of a place i, θ for the parameters of a model M, the *marginal likelihood* $p(\mathbf{y}_i|M)$ is the key expression in the Bayesian formulation for model selection. It represents an average of how good a particular model fits the observations over all possible parametrisation, convoluted by the prior. This quantity permits the comparison of different models given the data by having the Occam's Razor property[12]. However, analytical solutions are intractable so that the idea of the variational Bayesian approach is to approximate the marginal likelihood with a lower bound by using variational calculus techniques [10, 13, 2].

Introducing a free distribution q over $\mathbf{s}$ and θ, with $\int \sum_{\mathbf{s}} q(\mathbf{s}, \theta)\, d\theta = 1$, and applying the Jensen's inequality [3], it is possible to compute a lower bound on the log of the marginal likelihood. Maximising this lower bound with respect to the free distribution $q(\mathbf{s}, \theta)$ is analytically intractable. A better strategy is to factorise this free distribution to yield a variational approximation in which $q(\mathbf{s}, \theta) \approx q_{\mathbf{s}}(\mathbf{s})\, q_\theta(\theta)$:

$$\ln p(\mathbf{y}_i \mid M) \geq \int \sum_{\mathbf{s}} q_{\mathbf{s}}(\mathbf{s})\, q_\theta(\theta) \ln \frac{p(\mathbf{s}, \mathbf{y}_i, \theta \mid M)}{q_{\mathbf{s}}(\mathbf{s})\, q_\theta(\theta)} d\theta \tag{1}$$

$$= \mathcal{F}_M(q_{\mathbf{s}}(\mathbf{s}), q_\theta(\theta), \mathbf{y}_i) \tag{2}$$

The quantity $\mathcal{F}_M$ is a functional of the free distributions $q_{\mathbf{s}}(\mathbf{s})$ and $q_\theta(\theta)$ and is known as the negative free energy. The variational Bayesian algorithm iteratively maximises $\mathcal{F}_M$ with respect to the free distributions until the function reaches a stationary value.

An interesting implementation of VBEM can use conjugate priors that are analytically tractable and easy to interpret. Thus, Dirichlet, Normal and Wishart multivariate distributions [15] are used as priors over weights, means and covariances. They are denoted as $\mathcal{D}(\pi; \lambda)$, $\mathcal{N}\left(\mathbf{x}; \mu, \mathbf{\Sigma}^{-1}\right)$ and $\mathcal{W}(\Gamma; \alpha, \mathbf{B})$ and are functions of their hyperparameters. Also, a multivariate Student-t distribution $\mathcal{S}(\mathbf{x}; \rho, \mathbf{\Lambda}, \omega)$ is used to represent the predicted density.

Assuming a particular model M, the Gaussian mixture model has S components, where each component has weight given by π_s, mean μ_s and covariance $\mathbf{\Gamma}_s$. The set of parameters can be written as $\theta = \{\pi, \mu, \mathbf{\Gamma}\}$ where $\pi = \{\pi_1, \pi_2, \ldots, \pi_S\}$, $\mu = \{\mu_1, \mu_2, \ldots, \mu_S\}$ and $\mathbf{\Gamma} = \{\mathbf{\Gamma}_1, \mathbf{\Gamma}_2, \ldots, \mathbf{\Gamma}_S\}$.

Given these parameters and the model, the likelihood of an observation $y_{i,j}$ in a d-dimensional space can be written as

$$p(y_{i,j} \mid \theta, M) = \sum_{s=1}^{S} p(s_n = s \mid \pi)\, p(y_{i,j} \mid \mu_s, \mathbf{\Gamma}_s), \tag{3}$$

where each component is a Gaussian with $p\left(y_{i,j} \mid \mu_s, \mathbf{\Gamma}_s\right) = \mathcal{N}\left(y_{i,j}; \mu_s, \mathbf{\Gamma}_s\right)$ and $p\left(s_n = s \mid \pi\right)$ is a multinomial distribution representing the probability of the observation $y_{i,j}$ be associated with component s.

The prior over the parameters is given by

$$p\left(\theta \mid M\right) = p\left(\pi\right) \prod_s p\left(\mathbf{\Gamma}_s\right) p\left(\mu_s \mid \mathbf{\Gamma}_s\right) \tag{4}$$

where the weight prior is a symmetric Dirichlet $p\left(\pi\right) = \mathcal{D}\left(\pi; \lambda_0 \mathbf{I}\right)$, the prior over each covariance matrix is a Wishart $p\left(\mathbf{\Gamma}_s\right) = \mathcal{W}\left(\Gamma; \alpha_0, \mathbf{B}_0\right)$ and the prior over the means given the covariance matrices is a multivariate normal $p\left(\mu_s \mid \mathbf{\Gamma}_s\right) = \mathcal{N}\left(\mu_s; \mathbf{m}_0, \beta_0 \mathbf{\Gamma}_s\right)$. The joint likelihood of the data, assuming the samples are i.i.d., can be computed as

$$p\left(\mathbf{y_i}, \mathbf{s} \mid \theta, M\right) = \prod_{n=1}^{N} p\left(s_n = s \mid \pi\right) p\left(y_{i,n} \mid \mu_s, \mathbf{\Gamma}_s\right) \tag{5}$$

where $\mathbf{y}_i = \{y_{i,1}, y_{i,2}, \ldots, y_{i,n}\}$ and $\mathbf{s} = \{s_1, s_2, \ldots, s_S\}$.

The variational approximation for the log marginal likelihood leads to the following free densities q:

- For the covariance matrices, $q\left(\mathbf{\Gamma}\right) = \prod_s q\left(\mathbf{\Gamma}_s\right)$ with $q\left(\mathbf{\Gamma}_s\right) = \mathcal{W}\left(\Gamma; \alpha_s, \mathbf{B}_s\right)$;
- For the means, $q\left(\mu \mid \mathbf{\Gamma}\right) = \prod_s q\left(\mu_s \mid \mathbf{\Gamma}_s\right)$ with $q\left(\mu_s \mid \mathbf{\Gamma}_s\right) = \mathcal{N}\left(\mathbf{x}; \mathbf{m}_0, \beta_s \mathbf{\Gamma}_s\right)$;
- For the mixing coefficients, $q\left(\pi\right) = \mathcal{D}\left(\pi; \lambda\right)$, where $\lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_s\}$;
- For the hidden variable $\mathbf{s}$, $q\left(\mathbf{s}\right) = \prod_s q\left(s_s\right)$.

Taking the functional derivatives of the free energy with respect to the free densities q produces the update rules of VBEM for the mixture of Gaussians case. In the VBE-Step the weights of the hidden variable are calculated and in the VBM-Step parameters and hyperparameters are updated. These rules are omitted here for brevity but can be found in [1].

Once parameters and the model were obtained, the predictive density for a particular patch $p\left(y' \mid \mathbf{y_i}, M\right)$ has a close-form solution of a mixture of Student-t distributions,

$$p\left(y' \mid \mathbf{y_i}, M\right) = \sum_{s=1}^{S} \bar{\pi}_s \mathcal{S}\left(\mathbf{y}'; \rho_s, \mathbf{\Lambda}_s, \omega_s\right), \tag{6}$$

with $\omega_s = \alpha_s + 1 - d$ degrees of freedom, where the means are$\rho_s = \mathbf{m}_s$ and the covariances are $\mathbf{\Lambda}_s = \left(\left(\beta_s + 1\right) / \beta_s \omega_s\right) \mathbf{B}_s$. The weights are computed based on the hyperparameters of the Dirichlet distribution with $\bar{\pi}_s = \lambda_s / \sum_{s'} \lambda_{s'}$.

2.3.1 Heuristic for Searching

VBEM allows direct model comparison by evaluating the free energy function of different models. In the case of mixtures of Gaussians this model can be a single component or a mixture with hundreds of components. Theoretically, there is no limit for the number of components and the search for the best model can be cumbersome. To cope with this problem, an heuristic based on birth and death of components is used. This heuristic seems to be appropriate for robotics problems since simpler models are evaluated before more complex ones, which decreases the computational complexity.

The heuristic used here has the same stopping and splitting criteria as in [2] for mixtures of factor analysers. The selection for splitting is based on the component with the smallest individual free energy. The search ends when all existent components were divided but none of those divisions resulted in free energy improvement.

2.4 Multi-Class Classification

As opposed to most classification problems where the input is a single feature-vector, in this approach the whole set of patches of an image is used. Each patch has equal contribution to the final classification decision and it is evaluated under the different models representing the places. The idea is to compute the log-likelihood of a set of image patches for every model learnt. The log-likelihood with the largest value is the final decision of the classification. Thus, the label of an image i is the label corresponding to the place model that maximises the expression:

$$M^* = \arg\max_M \sum_{j=1}^{m} \log p\left(y_{i,j} \mid M\right) \tag{7}$$

The computation for the log-likelihood in selecting the model that best explains the set of patches can be quickly computed. Also, it is possible to include more models, allowing sequential learning implementations. This is one of the demands for autonomous navigation as the robot visits new places, representations of them should be incorporated and correlated with the current knowledge.

3 Implementation

The whole framework was implemented in Matlab and tested with a Pioneer 2-AT. Images were obtained with 320x240 of resolution in 24-bits colour. Patches have a size of 5x5 pixels and were convolved by 4 Gabor wavelets resulting in a input space of 175 dimensions. Each image is thus a set of 3072 patches equally distributed, and of the same size. Isomap is then used to reduce the dimensionality to 5 and create the training data. The intrinsic dimensionality was also estimated by Isomap through the evaluation of the residual variance.

Learning is performed offline with labelled images from the above set. The training images were selected to give a multi-view perspective of the place. They were, however, taken from a particular position inside the place so as to verify how the algorithm generalises them to multiple positions. In the indoor experiment for example, if an office has a rectangular shape, 4 training images are taken close to the walls but the algorithm should still be able to recognise the place when observing it from the centre.

The variational Bayesian learning starts searching for the best model from a single-component model and follows the heuristic of birth and death as described before. The covariance obtained from the mixture of linear models is used to initialised the parameters α_0 and $\mathbf{B}_0$ of the Wishart distribution.

When testing the algorithm, the whole set of 3072 patches of an image is used. The process takes about 1 second per image in a Pentium M 1.7GHz which comprises Gabor convolutions, inference in the mixture of linear models and log-likelihood computation for each model learnt. Further implementations may use a subset of patches sampled from the original set to reduce the classification time. This, however, may decrease the accuracy which characterises an accuracy-time trade-off.

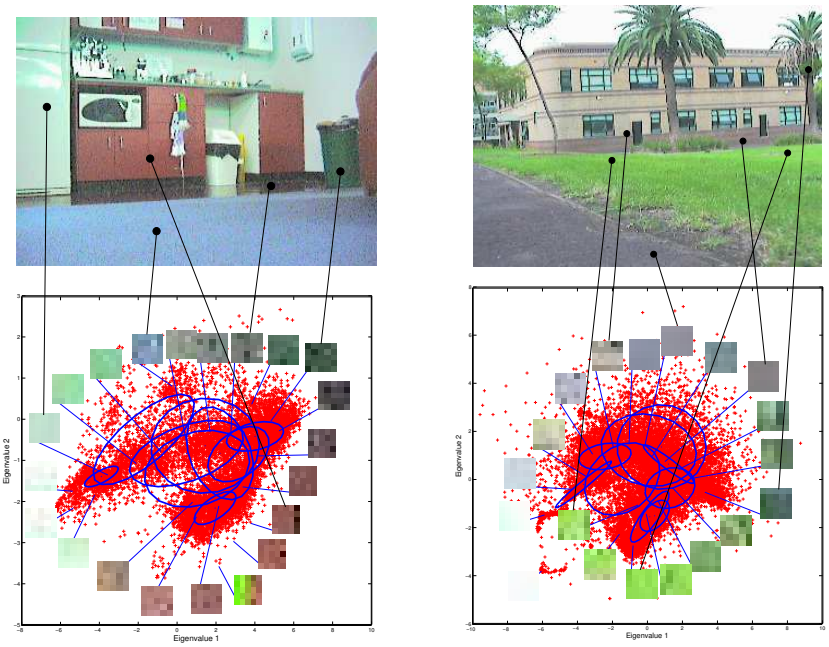
4 Experiments[1]

Two different experiments were performed to evaluate the algorithm in different conditions - indoor and outdoor environments. In both experiments, there were people walking by the places and sometimes occluding the robot's view. In the outdoor experiment, there were also cars and bicycles moving in some places which make the problem more difficult since the environment is dynamic.

4.1 Indoor dataset

The indoor dataset consists of 55 training images of 9 different places - each place has 5 to 9 training images. The test set has 1579 sequence images obtained by the robot when navigating inside the lab. The classes are {kitchen, seminar room, student cubicle 1, corridor 1, student cubicle 2, corridor 2, corridor

[1] Videos with the experiments are available at: http://www.acfr.usyd.edu.au/people/postgrads/ftozeto



a) Kitchen model. b) ACFR Park model.

Figure 1: Generative models learnt for kitchen (indoor) and ACFR Park (outdoor). Points are plotted on the direction of the two largest eigenvalues of the essential features (the total dimensionality of this space is 5). Ellipses correspond to the covariance matrices of the components learnt with VBEM. The association between the patches and their location in the real scene is also indicated.

3, research office and professor office}. The generative model for the kitchen is depicted in Figure 1a. It shows the covariance matrices learnt through VBEM from the essential features. The correlation between the patches and their real position is also indicated.

Table 1a shows the precision and recall results for this 9-class problem. The classification was accurate considering that in many occasions the robot was very close to walls or other objects, in situations where even for a human, the classification would be difficult. Also, some places are very alike, for example, the corridors and the student cubicles. These places can be distinguished from few objects such as paintings in the case of the corridors and books, computers on the desks of the student cubicles. However, both objects were not observed by the robot since they are in a higher position, away from the robot view. Figure 2 shows the training images for the kitchen model. Even not having a training image with a wider view of the kitchen, the classifier was able to recognise the kitchen from the image of Figure 1a. This generalisation is one of the main properties of the algorithm.

4.2 Outdoor dataset

The outdoor dataset consists of 57 training images of 11 different places at University of Sydney, with each place having 3 to 8 training images. The test set has 3820 images obtained from a half-kilometre journey around the University of Sydney. The classes are {ACFR front, ACFR park, Eng. Building, Eng. Road, Eng. Carpark, Mech. Building, ACFR carpark, ACFR road, garage(indoor) and office(indoor)}. The generative model for the class ACFR park is shown in Figure 1b. Also annotated is the correlation between patches and their real location.

Table 1b presents the precision and recall results which are in general better than the indoor dataset. The most difficult problem of the outdoor dataset was to distinguish between the two carparks. When the robot was very close to a car, it was not able to have a more general view of the place which resulted in

Figure 2: The training images used for the kitchen model. The model learnt from them was able to recognise wider views of the place such as the image in Figure 1a.

Place Name	Precision	Recall
Kitchen	81.82	76.60
Seminar Room	68.07	81.82
Student Cubicle 1	52.86	48.05
Corridor 1	74.96	84.09
Student Cubicle 2	62.96	46.12
Corridor 2	40.00	62.50
Corridor 3	100	15.13
Researcher Office	73.01	90.16
Professor Office	62.96	71.83

a) Indoor results.

Place Name	Precision	Recall
ACFR Front	71.04	87.44
ACFR Park	83.19	71.05
Eng. Building	65.40	77.31
Eng. Road	91.43	45.43
Eng. Carpark	26.17	87.10
Mech. Building	64.47	17.63
Mech. Corridor	90.68	65.54
ACFR Carpark	55.24	76.40
ACFR Road	87.00	74.34
Garage	23.94	87.18
Office	99.47	61.84

b) Outdoor results.

Table 1: Precision and recall results for the indoor and outdoor datasets.

classifying the image as the other carpark. Also, “Mech. Building” and “Eng. Building” are physically in the same building and the limits of where one starts and the other finish are not very clear.

5 Conclusions

The framework here proposed has three main contributions: it shows that mixture of linear models can be used as a tool for nonlinear regression problems with noise such as Isomap mappings; it demonstrates how variational Bayesian learning with a free-energy heuristic can choose the right number of components of a mixture of Gaussians; it shows how the log-likelihood can be applied to multi-class problems where classification is given from a set of samples rather than from a single point.

Patches of images and images themselves are treated here as independent and identically distributed. In the case of patches, further implementations can include the positions of each patch as additional dimensions in the feature-vector. Also, spatial relations among them can be included in more sophisticated relational statistical models. This, however, should preserve the main benefits of the model such as learning from few images and real-time classification.

Most of the wrong classifications took place when the robot was close to a wall or object that occluded a wider view of the scene. This problem could be avoided if a topological map of the environment were encoded in a hidden Markov model to constrain the search to fewer places. In future works, algorithms for learning this topological map will be investigated as well as how to integrate them in the framework.

Acknowledgements

This work is supported by the ARC Centre of Excellence programme, funded by the Australian Research Council (ARC) and the New South Wales (NSW) State Government.

References

[1] H. Attias. A variational bayesian framework for graphical models. In *Proceedings of Neural Information Processing Systems 12*, Cambridge, MA, USA, 2000. MIT Press.

[2] M. J. Beal. *Variational Algorithms for Approximate Bayesian inference*. PhD thesis, The Gatsby Computational Neuroscience Unit, University College London, May 2003.

[3] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. John Wiley & Sons, Inc, New York, 1991.

[4] T. Cox and M. Cox. *Multidimensional Scaling*. Chapman and Hall, 1994.

[5] A. Davison and D. Murray. Simultaneous localisation and map-building using active vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:865–880, 2002.

[6] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society B*, 39:1–37, 1977.

[7] M. G. Dissanayake, P. Newman, S. Clark, and H. F. Durrant-Whyte. A solution to the simultaneous localization and map building (SLAM) problem. *IEEE Transactions on Robotics and Automation*, 17:229–241, 2001.

[8] R. Duda, P. Hart, and D. Stork. *Pattern Classification*. Wiley-Interscience, New York, second edition, 2001.

[9] Z. Ghahramani and G. E. Hinton. The EM algorithm for mixtures of factor analyzers. Technical report, Department of Computer Science, University of Toronto CRG-TR-96-1, 1996.

[10] M. I. Jordan, Z. Ghahramani, T. Jaakkola, and L. K. Saul. An introduction to variational methods for graphical models. *Machine Learning*, 37(2):183–233, 1999.

[11] J. Kosecka and F. Li. Vision based topological Markov localization. In *Proc. of IEEE International Conference on Robotics and Automation (ICRA)*, 2004.

[12] D. J. C. Mackay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, Cambridge, UK, 2003.

[13] T. P. Minka. Using lower bounds to approximate integrals. *Informal notes available at http://www.stat.cmu.edu/minka/papers/learning.html*, 2001.

[14] T. Mitchell and F. Labrosse. Visual homing: a purely appearance-based approach. In *In Proc. of Towards Autonomous Robotic Systems*, Colchester, UK, 2004.

[15] R. J. Muirhead. *Aspects of Multivariate Statistical Theory*. Wiley, New York, USA, 1982.

[16] L. K. Saul and S. T. Roweis. Think globally, fit locally: Unsupervised learning of low dimensional manifolds. *Journal of Machine Learning Research*, 4(Jun):119–155, 2003.

[17] S. Se, D. Lowe, and J. Little. Global localization using distintive visual features. In *Proc. of International Conference on Intelligent Robots and Systems*, pages 226–231, Lausanne, Switzerland, 2002.

[18] R. Sim and G. Dudek. Learning environmental features for pose estimation. *Image and Vision Computing*, 19(11):733–739, 2001.

[19] J. Tenenbaum, V. DeSilva, and J. C. Langford. A global geometric framework for nonlinear dimensionality redution. *Science*, 290:2319–2323, 2000.

[20] A. Torralba, K. Murphy, W. Freeman, and M. Rubin. Context-based vision system for place and object recognition. In *Proceedings of International Conference of Computer Vision*, 2003.

[21] I. Ulrich and I. Nourbakhsh. Appearance-based place recognition for topological localization. In *Proceedings of ICRA 2000*, volume 2, pages 1023–1029, April 2000.

[22] J. Wolf, W. Burgard, and H. Burkhardt. Robust vision-based localization for mobile robots using an image retrieval system based on invariant features. In *Proc. of the IEEE International Conference on Robotics & Automation (ICRA)*, 2002.

Vision-based SLAM using the Rao-Blackwellised Particle Filter

Robert Sim, Pantelis Elinas, Matt Griffin and James J. Little

Laboratory for Computational Intelligence
Computer Science Department
University of British Columbia
Vancouver, BC, V6T 1Z4
Canada

Abstract

We consider the problem of Simultaneous Localization and Mapping (SLAM) from a Bayesian point of view using the Rao-Blackwellised Particle Filter (RBPF). We focus on the class of indoor mobile robots equipped with only a stereo vision sensor. Our goal is to construct dense metric maps of natural 3D point landmarks for large cyclic environments in the absence of accurate landmark position measurements and reliable motion estimates. Landmark estimates are derived from stereo vision and motion estimates are based on visual odometry. We distinguish between landmarks using the Scale Invariant Feature Transform (SIFT). Our work differs from current popular approaches that rely on reliable motion models derived from odometric hardware and accurate landmark measurements obtained with laser sensors. We present results that show that our model is a successful approach for vision-based SLAM, even in large environments. We validate our approach experimentally for long camera trajectories. We identify the areas where future research should focus in order to further increase its accuracy and scalability to significantly larger environments.

1 Introduction

In robotics, the problem of *Simultaneous Localization and Mapping* (SLAM) is that of estimating both a robot's location and a map of its surrounding environment. It is an inherently hard problem because noise in the estimate of the robot's pose leads to noise in the estimate of the map and vice versa. In general, SLAM algorithms must address issues regarding sensors, map representation and robot/environment dynamics. A probabilistic framework is necessary for combining over time the incoming sensor measurements and robot control signals.

Previous work on SLAM has focused predominantly on planar robots that construct $2D$ occupancy grid maps [15] using sonar or laser sensors [2, 6, 14]. Laser sensors have high depth resolution providing accurate measurements of landmark positions but suffer from the perceptual aliasing problem. We focus on using vision as the sensing modality. Vision can be used to construct $2D$ [20] and $3D$ occupancy grids or maps of $3D$ natural landmarks [17].

In order to integrate sensor measurements and robot control signals over time, the Extended Kalman Filter (EKF) has been the most common approach since its application by Smith et. al. [18]. The complexity of the EKF grows quadratically with the number of landmarks added to the map making its use problematic for learning maps of large environments. The EKF is also very sensitive to outliers in landmark detection. A single outlier measurement once incorporated into the covariance matrix cannot be corrected at a later time if more information becomes available. Another approach is the use of Particle Filters (PFs) to approximate the posterior distribution over robot poses and maps. PFs can handle outliers better than the EKF but scale poorly with respect to the dimensionality of the state. The Rao-Blackwellised Particle Filter (RBPF) is a method for handling this dimensionality problem [5]. Murphy [16] was the first to study the application of RBPFs to SLAM and others followed [2]. Sampling over robot poses allowed him to independently

estimate each landmark using an EKF. A naive implementation yields a complexity of $O(MN)$, where M is the number of new particles at each step. Montemerlo et al. [14] present FastSLAM, a variant of RBPF-based SLAM that introduces a tree-based structure which refines this complexity to $O(M \log N)$ by sharing landmarks between particles. Similarly, Eliazar et. al. have constructed an efficient $2D$ occupancy grid representation for particle-based SLAM [6].

The approach we advocate here depends on a motion model based on visual odometry and an observation model based on $3D$ landmarks from stereo vision coupled with the Scale-Invariant Feature Transform (SIFT) detector [12]. SIFT is used for robust data association. These features are desirable as landmarks because they are invariant to image scale, rotation and translation as well as partially invariant to illumination changes and affine or 3D projection. This combination can result in many viable landmarks from an unaltered environment and in fact SIFT has been shown to outperform other leading edge image descriptors in matching accuracy [13].

The major contributions of this paper are two-fold. First, we present RBPF-based SLAM utilizing vision-based sensing, rather than traditional range sensing with a laser. Our motion model depends on visual odometry that generalizes to unconstrained $3D$ motion. That is, we assume no prior knowledge of the control actions that drive the camera through the world. Furthermore, where previous implementations of the SLAM algorithm have generally employed sensors with a wide field of view, our experimentation demonstrates the performance of the algorithm using sensors with a narrow field of view. We leverage the strengths of particle filter-based methods for uncertainty estimation (such as the possibility of multi-modal estimates), with data association techniques that were previously only applied to Kalman-filter based estimators [1, 9, 17].

Our works differs from [10] because their approach learns topological maps. The maps we learn are similar to those first introduced by [17] but our approach can handle longer trajectories. In addition, our work does not depend on mechanical odometry measurements which are used in both [10] and [17].

This paper is structured as follows. We first present an overview of Bayesian filtering applied to SLAM and its RBPF approximation. We then focus on the details of our vision-based SLAM presenting our map representation, observation and motion models. We provide experimental results to support the validity of our approach and conclude by discussing scalability issues and implementation pitfalls along with directions for future work.

2 SLAM using the Bayes Filter

Formally, and in accordance with popular SLAM literature, let s_t denote the robot's pose at time t, m_t the map learned thus far and $x_t = \{s_t, m_t\}$ be the complete *state*. Also, let u_t denote a control signal or a measurement of the robot's motion from time $t-1$ to time t and z_t be the current observation. The set of observations and controls from time 0 to t are denoted by z^t and u^t respectively. Our goal is to estimate the density

$$p(s_t, m_t | z^t, u^t) = p(x_t | z^t, u^t) \tag{1}$$

That is we must integrate the set of observations and controls as they arrive over time in order to compute the posterior probability over the unknown *state*. Applying Bayes rule on 1 we get [19]

$$p(x_t | z^t, u^t) = Bel(x_t) = \eta p(z_t | x_t) \int p(x_t | u_t, x_{t-1}) p(x_{t-1} | z^{t-1}, u^{t-1}) dx_{t-1} = \\ \eta p(z_t | x_t) \int p(x_t | u_t, x_{t-1}) Bel(x_{t-1}) dx_{t-1} \tag{2}$$

where η is a normalizing constant.

Equation 2 allow us to recursively estimate the posterior probability of maps and robot poses if the two distributions $p(z_t|x_t)$ and $p(x_t|u_t, x_{t-1})$ are given. These distributions model the observations and robot motion respectively. For SLAM, an analytical form for $Bel(x_t)$ is hard to obtain and as such the Bayes filter is not directly applicable. Instead we approximate it using a Particle Filter as we describe next.

2.1 Rao-Blackwellised Particle Filters

In the previous section we showed how we can recursively estimate the posterior density $Bel(x_t)$ using the Bayes filter. As discussed we cannot directly implement this filter and so we employ a common approximation technique known as Particle Filtering. PF is a general method for approximating $Bel(x_t)$ using a set of m weighted particles, $Bel(x_t) = \{x^{(i)}, w^{(i)}\}_{i=1,\cdots,m}$. The system is initialized according to $p(x_0)$ and the recursive update of the Bayes filter proceeds using a procedure known as *sampling-importance-resampling* [3]. A major caveat of the standard PF is that it requires a very large number of particles as the size of the *state* increases. Since for SLAM the *state* of the system includes the map that often has tens of thousands of landmarks, the PF is not applicable from a practical point of view. The method of Rao-Blackwellization can be used to reduce the complexity of the PF [4]. In the case of SLAM, we sample over possible robot poses, s_t, and then marginalize out the map [16]. The posterior then is factored as:

$$Bel(x_t) = Bel(s_t, m_t) = p(s_t, m_t|z^t, u^t) = p(s_t|z^t, u^t) \prod_k p(m(k)|s^t, z^t, u^t) \tag{3}$$

where $m(k)$ denotes the k-th landmark in the map. We use a standard PF to estimate $p(s_t|z^t, u^t)$ and an EKF for each of the k landmarks.

3 Vision-based SLAM

In this section we present our RBPF model for vision-based SLAM. We first describe how we represent maps that are central to our method. Next we define observations and how we compute the observation likelihood followed with a description of our motion model based on visual odometry.

3.1 Map Representation

We construct maps of naturally occurring $3D$ landmarks similar to those proposed in [17]. Each landmark is a vector $l = \{P, C^G, \alpha, s, f\}$ such that $P = \{X^G, Y^G, Z^G\}$ is a 3-dimensional position vector in the map's global coordinate frame, C^G is the 3×3 covariance matrix for P, and α, s, f describe an invariant feature based on the Scale Invariant Feature Transform [12]. Parameter α is the orientation of the feature, s is its scale and f is the 36-dimensional key vector which represents the histogram of local edge orientations.

3.2 Observation Model

Let I_t^R and I_t^L denote the right and left gray scale images captured using the stereo camera at time t. We compute image points of interest from both images by selecting maximal points in the scale space pyramid of a Difference of Gaussians [12]. For each such point, we compute the SIFT descriptor and record its scale and orientation. We then match the points in the left and right images in order to compute the points' $3D$ positions in the camera coordinate frame. Matching is constrained by the stereo camera's known epipolar geometry and the Euclidean distance of their SIFT keys. Thus, we obtain a set $O_C = \{o_1, o_2, \cdots, o_n\}$ of n local landmarks such that $o_j = \{P_{o_j} = \{X_{o_j}^L, Y_{o_j}^L, Z_{o_j}^L\}, p_{o_j} = \{r_{o_j}, c_{o_j}, 1\}, C^L, \alpha, s, f\}$ where $p_{o_j} = \{r_{o_j}, c_{o_j}, 1\}$ is the image coordinates of the point and $j \in [1 \cdots n]$.

An observation is defined as a set of k correspondences between landmarks in the map and the current view, $z_t = \cup_{1 \cdots k}\{l_i \leftrightarrow o_j\}$ such that $i \in [1..m]$ and $j \in [1..n]$ where m is the number of landmarks in the map and n is the number of landmarks in the current view. Each local landmark either corresponds to a mapped landmark l_k, or has no corresponding landmark, denoted by the null correspondence $l_\emptyset$. We compare the landmarks' SIFT keys in order to obtain these correspondences just as we did before during stereo matching. There are no guarantees that all correspondences are correct but the high specificity of SIFT results in a reduced number of incorrect matches.

A pose of the camera, s_t, defines a transformation $[R, T]_{s_t}$ from the camera to the global coordinate frame. Specifically, R is a 3×3 rotation matrix and T is a 3×1 translation vector. Each landmark in the current view can be transformed to global coordinates using the well known equation

$$P_{o_j}^G = R_{s_t} P_{o_j} + T_{s_t} \tag{4}$$

Using Equation 4 and the Mahalanobis distance metric we can define the observation log-likelihood, $\log p(z_t|m_t^i)$. Special consideration must be taken when computing this quantity, particularly where large numbers of feature observations, with significant potential for outlier correspondences, are present. We compute it by summing over the feature correspondences:

$$\log p(z_t|m_t^i) = \sum_k \log p(o_k|l_k^i) \tag{5}$$

The log-likelihood of the k-th observation is given by

$$\log p(o_k|l_k^i) = -0.5 \min(T_l, (P_{o_k}^G - P_k^G)^T S^{-1} (P_{o_k}^G - P_k^G)) \tag{6}$$

where the correspondence covariance S is given by the sum of the transformed observation covariance $C_{o_k}^L$ and the landmark covariance C_k^G:

$$S = R_{s_t} C_{o_k}^L R_{s_t}^T + C_k^G. \tag{7}$$

For the null correspondence, S is assumed to be zero.

The maximum observation innovation T_l is selected so as to prevent outlier observations from significantly affecting the observation likelihood. However, given the potentially large numbers of correspondences, even with a reasonable setting for T_l (in our case, 4.0), the magnitude of $\log p(z_t|m_t^i)$ can be such that raising it to the exponential to evaluate the i-th particle weight:

$$w_i = \frac{p(z_t|m_t^i)}{\sum_{j=1}^N p(z_t|m_t^j)} \tag{8}$$

results in zero weights. In order to preserve numerical accuracy, we note the following simplification. Let $H_i = -\log p(z_t|m_t^i)$. Without loss of generality, assume that m_t^0 is the particle that minimizes H_i. Then for all particles:

$$\log p(z_t|m_t^i) = -(H_0 + H_i'). \tag{9}$$

where $H_i' = H_i - H_0$. Substituting into Equation 8:

$$w_i = \frac{\exp(-(H_0 + H_i'))}{\sum_{j=1}^N \exp(-(H_0 + H_i'))} = \frac{\exp(-H_0)\exp(-H_i')}{\exp(-H_0)\sum_{j=1}^N \exp(-H_i')} = \frac{\exp(-H_i')}{\sum_{j=1}^N \exp(-H_i')} \tag{10}$$

Note that for m_t^0, $H_i' = 0$, so we guarantee that at least one particle has a numerator of 1, above, and the denominator is at least 1.0. This approach effectively eliminates the probability mass associated with outliers that is common to all particles. It is also important to note that using this approach assures that all particles have comparable weights – every particle has the same number of input observations, and outliers are represented in the model on a per-particle basis. Hence, a particle with more outlier matches will have a lower weight than a particle with better data association.

3.3 Motion Model

An essential component to the implementation of RPBF is the specification of the robot's motion model, u_t. In all previous work, this has been a function of the robot's odometry, i.e., wheel encoders that measure the amount the robot's wheels rotate that can be mapped to a metric value of displacement and rotation. Noise drawn from a Gaussian is then added to this measurement to take into account slippage as the wheels rotate. Odometric measurements of this type are limited to robots moving on planar surfaces. We want to establish a more general solution. Thus, we obtain u_t measurements by taking advantage of the vast amount of research in multiple view geometry [8]. Specifically, it is possible to compute the robot's displacement directly from the available image data including an estimate of the uncertainty in that measurement.

Let I_t and I_{t-1} represent the pairs of stereo images taken with the robot's camera at two consecutive intervals with the robot moving between the two. For each pair of images we detect points of interest, compute SIFT descriptors for them and perform stereo matching, as described earlier in section 3.2, resulting in 2 sets of landmarks L_{t-1} and L_t. We compute the camera motion using a non-linear optimization

Figure 1: Sample images from the 4000 frame sequence.

algorithm minimizing the re-projection error of the $3D$ coordinates of the landmarks. We employ the Levenberg-Marquardt (LM) non-linear optimization algorithm [8]. We utilize the $3D$ coordinates of our landmarks and use the LM algorithm to minimize their re-projection error. Let $\tilde{s}_t$ be the 6-dimensional vector $\tilde{s}_t = [roll, pitch, yaw, T_{11}, T_{21}, T_{31}]$ corresponding to a given $[R, T]$. Our goal is to iteratively compute a correction term χ

$$\tilde{s}_t^{i+1} = \tilde{s}_t^i - \chi \tag{11}$$

such as to minimize the vector of error measurement ϵ, i.e., the re-projection error of our $3D$ points. For a known camera calibration matrix K, ϵ is defined as

$$\epsilon = \begin{bmatrix} \epsilon_0^T \\ \epsilon_1^T \\ \vdots \\ \epsilon_k^T \end{bmatrix} = \begin{bmatrix} p_t^0 - K(RP_{t-1}^0 + T) \\ p_t^1 - K(RP_{t-1}^1 + T) \\ \vdots \\ p_t^k - K(RP_{t-1}^k + T) \end{bmatrix} \tag{12}$$

Given an initial estimate for the parameters, we wish to solve for χ that minimizes ϵ, i.e.,

$$\begin{bmatrix} J \\ \lambda I \end{bmatrix} \chi = \begin{bmatrix} \epsilon \\ \lambda d \end{bmatrix} \Leftrightarrow (J^T J + \lambda I)\chi = J^T \epsilon + \lambda d \tag{13}$$

where $J = [\frac{\partial \epsilon_0}{\partial \chi}, \cdots, \frac{\partial \epsilon_k}{\partial \chi}]^T$, is the Jacobian matrix, I is the identity matrix and d is an initial solution that in this case is set to zero rotation and translation. The LM algorithm introduces the variable λ that controls the convergence of the solution by switching between pure gradient descent and Newton's method. As discussed in [11] solving Equation 13, i.e., the normal equations, minimizes

$$||J\chi - \epsilon||^2 + \lambda^2 ||\chi - d||^2 \tag{14}$$

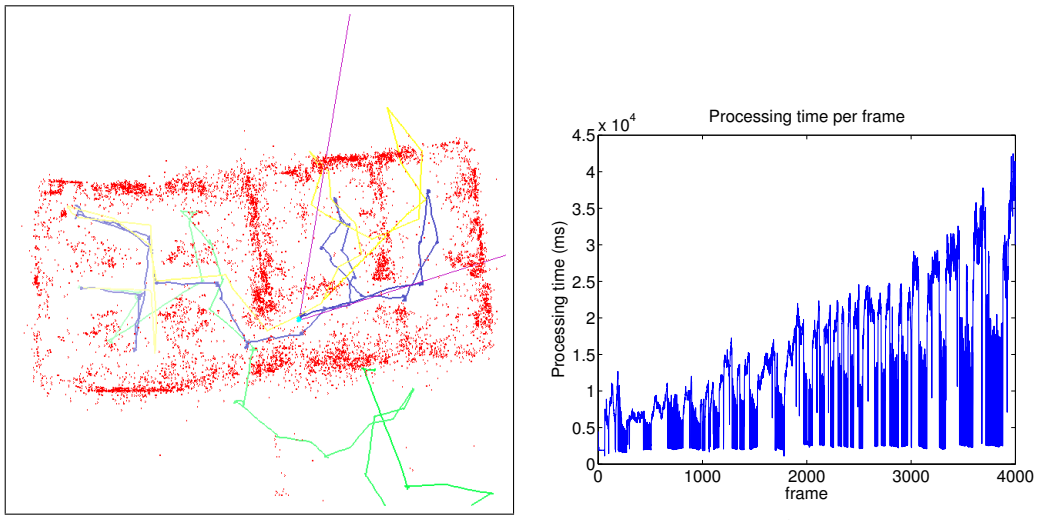
The normal equations can be solved efficiently using the SVD algorithm. A byproduct from solving Equation 14 is that we also get the covariance of the solution in the inverse of $J^T J$.

4 Experimental Results

For the purposes of our experiments, we used an RWI B14 robot with a BumbleBee stereo head from Point Grey Research. We manually drove it through two connecting rooms in a laboratory environment, and we collected 4000 images along a trajectory of approximately 67.5m. Figure 1 displays a subset of the collected images. While the visual odometry produces 6-DOF motion estimates, we chose to estimate only three parameters in constructing s_t from $\tilde{s}_t$. While for this particular experiment, this assumption was reasonable, we have preliminary results suggesting that relaxing the assumption altogether will be successful [7].

As a summary of the map construction process, Table 1 describes at 200 frame intervals the mean number of landmarks per particle (SIFT features observed more than three times), the total distance traveled according to the robot's odometer, and the total number of SIFT features (landmarks have been observed at least three times, whereas SIFT features have been observed at least once, and are removed if unobserved for a second time within three frames).

Part (a) of Figure 2 depicts the map constructed for the maximum-likelihood particle at the end of exploration. This map is not post-processed to remove noise or perform any global optimization. The



(a) (b)

Figure 2: (a) The constructed map for the best sample at the end of exploration. The blue trajectory indicates the trajectory of the best sample and the green trajectory indicates the visual odometry measurements. The robot odometer (not used for map estimation) is plotted as a yellow trajectory. Landmark positions are marked with red 'X's. The set of particles is shown by the cyan blob near the center. The width of the map is approximately 18m. (b) Processing time per frame. The mean is 11.9s.

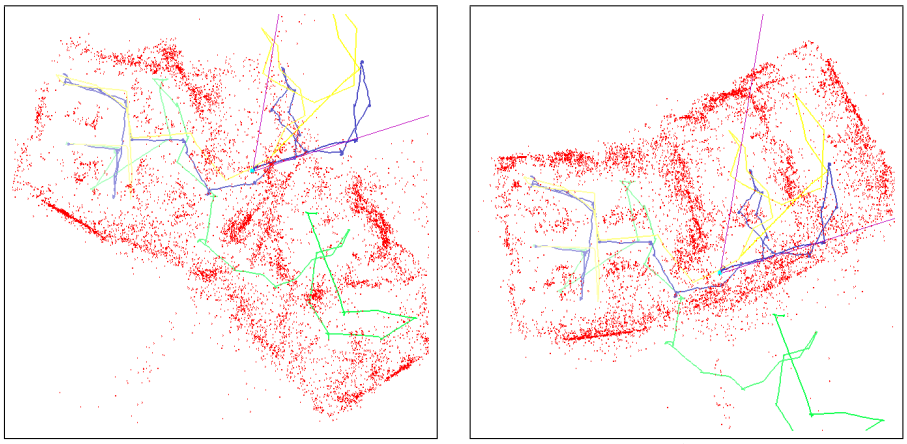
blue trajectory indicates the trajectory of the best sample and the green trail indicates the visual odometry measurements. The robot odometer (not used for map estimation) is plotted as a yellow trajectory. All three trajectories begin from the origin, on the left side of the image. Landmark positions are marked with red 'X's. The set of particles is shown by the cyan blob near the center. Figure 4 depicts the maps as constructed using only visual odometry and the robot's odometry, respectively. Clearly, the filter outperforms both kinds of odometry.

Part (b) of Figure 2 depicts the computation time for each frame of the sequence on a 2.6GHz Pentium 4 CPU. The mean compute time per frame is 11.9s. The base-line cost (horizontal line near about 2s) corresponds to the motion estimation, whereas the larger costs correspond to RBPF updates (which are triggered only when sufficient motion is detected). A major contributor to the increased cost over time is the cost of matching SIFT features. For this experiment, to ensure robustness in data association, we employed a linear-time comparison of image features with SIFT features in the map ($O(MN)$ where M is the number of observed features and N is the number in the map). There are a variety of fast methods for improving this result, particularly kd-trees. We have found that there is some degradation in data association quality using kd-trees, and that the kd-tree can become overcrowded over time as a result. Future work will address these issues.

5 Scalability Issues and Implementation Pitfalls

In this paper, we have presented experimental results which push the envelope for what can be accomplished using vision and no prior knowledge of the camera's motion. In particular, we are successfully building accurate maps over long-range motion. However, there are several considerations that were taken into account in order to compute an accurate result in a reasonable amount of time.

There are two barriers to full frame-rate operation. First, the number of particles must be small in order to update the maps in a reasonable amount of time. While some papers have argued that a proposal distribution conditioned on the observation can lead to a filter that converges with only one particle, we would argue that this distribution is highly over-confident and somewhat biased, necessitating the injection of noise into the distribution, and also necessitating a reasonably large number of particles to ensure diversity in the filter (particularly important for loop closing). For these experiments, we used 400 particles, and we



(a) (b)

Figure 3: The constructed map (a) based solely on the visual odometry and (b) based solely on the robot's odometer (which was not used for constructing the map in Figure 2).

believe the loop can be reliably closed over reasonable distances We have not experimented significantly with fewer particles, or the level of noise that must be injected.

The second barrier is the management and correspondence of SIFT features. We use 36-element SIFT feature vectors, but perform a list traversal to match each feature. As mentioned above, matching can be improved by using a kd-tree, but this can present additional complications for key maintenance (for example, deleting unmatched keys from the tree after a few frames). Without sophisticated key maintenance, the tree can become over-populated, making it very difficult to verify good matches. The rate at which SIFT keys are added is another consideration, and we insert a limited number of keys into the database at each frame (10-15). These insertions are predicated on the new keys being sufficiently distinct from the keys already in the database. Without these limits, the number of SIFT keys can grow by up to 500 keys per frame.

6 Conclusions

In this paper we have presented our model for vision-based SLAM from a Bayesian point of view using the RBPF. We show that we can successfully construct dense metric maps of more than 11,000 $3D$ point landmarks for long camera trajectories in the order of 68 meters and 4000 image frames. We have utilized SIFT for identifying landmarks and defining the observation function of our model. We diverged from popular SLAM literature by not relying on motion estimates based on odometric hardware but only on visual odometry. We have identified a number of areas that need further work to increase the computational efficiency, and representational power of our method, in order to build accurate maps of even larger environments.

References

[1] A. J. Davison. Real-time simultaneous localisation and mapping with a single camera. In *Proc. International Conference on Computer Vision, Nice*, Oct. 2003.

[2] F. Dellaert, W. Burgard, D. Fox, and S. Thrun. Using the CONDENSATION algorithm for robust, vision-based mobile robot localization. In *IEEE Computer Society Conf. on Computer Vision and Pattern Recognition*, pages 2588–2593, Ft. Collins, CO, June 1999. IEEE Press.

[3] A. Doucet, N. de Freitas, and N. Gordon. *Sequential Monte Carlo in Practice*. Springer-Verlag, 2001.

[4] A. Doucet, N. de Freitas, K. Murphy, and S. Russell. Rao-blackwellised particle filtering for dynamic bayesian networks. In *Uncertainty in Artificial Intelligence*. 2000.

Frame Number	Mean Landmarks per particle	Distance traveled (m)	SIFT Features
4000	11085	65.71	38394
3600	10056	58.82	34404
3199	9065	52.05	30639
2797	8355	43.58	27369
2400	7155	38.07	23259
1998	6213	31.11	19869
1600	4857	26.01	15264
1200	3773	18.69	11424
800	2625	11.48	7674
400	1542	4.02	4194

Table 1: Summary of map construction statistics at 400 frame intervals.

[5] A. Doucet, S. Godsill, and C. Andrieu. On sequential Monte Carlo sampling methods for Bayesian filtering. *Statistics and Computing*, 10(3):197–208, 2000.

[6] A. I. Eliazar and R. Parr. DP-slam 2.0. In *Proc. ICRA 2004*, New Orleans, LA, 2004. IEEE Press.

[7] P. Elinas and J. Little. σMCL: Monte-Carlo localization for mobile robots with stereo vision. In *Proceedings of Robotics: Science and Systems*, Cambridge, MA, USA, 2005.

[8] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision.* Cambridge Univ. Pr., Cambridge, UK, 2000.

[9] M.-H. Y. Jason Meltzer, Rakesh Gupta and S. Soatto. Simultaneous localization and mapping using multiple view feature descriptors. Sendai, Japan, September 2004.

[10] N. Karlsson, E. D. Bernardo, J. Ostrowski, L. Goncalves, P. Pirjanian, and M. E. Munich. The vSLAM algorithm for robust localization and mapping. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 24–29, Barcelona, Spain, April 2005.

[11] D. Lowe. Fitting parameterized three-dimensional models to images. *IEEE Trans. Pattern Analysis Mach. Intell. (PAMI)*, 13(5):441–450, May 1991.

[12] D. G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the Int. Conf. on Computer Vision*, pages 1150–1157, Corfu, Greece, September 1999. IEEE Press.

[13] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. In *International Conference on Computer Vision & Pattern Recognition*, volume 2, pages 257–263, June 2003.

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

[15] H. P. Moravec and A. Elfes. High resolution maps from wide angle sonar. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 116–121, 1985.

[16] K. Murphy. Bayesian map learning in dynamic environments. In *1999 Neural Information Processing Systems (NIPS)*, 1999.

[17] S. Se, D. G. Lowe, and J. Little. Mobile robot localization and mapping with uncertainty using scale-invariant visual landmarks. *Int. J. Robotics Research*, 21(8):735–758, 2002.

[18] R. Smith, M. Self, and P. Cheeseman. A stochastic map for uncertain spatial relationships. In *Workshop on Spatial Reasoning and Multisensor Fusion*, 1987.

[19] S. Thrun. Robot mapping: A survey. Technical Report CMU-CS-02-11, Carnegie Mellon university, February 2002.

[20] V. Tucakov, M. Sahota, D. Murray, A. Mackworth, J. Little, S. Kingdon, C. Jennings, and R. Barman. A stereoscopic visually guided mobile robot. In *Proc. of Hawaii International Conference on Systems Sciences*, 1997.

Towards Solving Large-Scale POMDP Problems Via Spatio-Temporal Belief State Clustering

Xin Li, William K. Cheung, Jiming Liu
{lixin,william,jiming}@comp.hkbu.edu.hk

Department of Computer Science of Hong Kong Baptist University

Abstract

Markov decision process (MDP) is commonly used to model a stochastic environment for supporting optimal decision making. However, solving a large-scale MDP problem under the partially observable condition (also called POMDP) is known to be computationally intractable. Belief compression by reducing belief state dimension has recently been shown to be an effective way for making the problem tractable. With the conjecture that temporally close belief states should possess a low intrinsic degree of freedom due to problem regularity, this paper proposes to cluster the belief states based on a criterion function measuring the belief states spatial and temporal differences. Further reduction of the belief state dimension can then result in a more efficient POMDP solver. The proposed method has been tested using a synthesized navigation problem (Hallway2) and empirically shown to be a promising direction towards solving large-scale POMDP problems. Some future research directions are also included.

1 Introduction

Markov decision process (MDP) is commonly used to model a stochastic environment for supporting optimal decision making. An MDP model is characterized by a finite set of states $\mathcal{S}$, a finite set of actions $\mathcal{A}$, a set of corresponding state transition probabilities $T : \mathcal{S} \times \mathcal{A} \rightarrow \Pi(\mathcal{S})$ and a reward function $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. Solving an MDP problem means finding an optimal policy which maps each state to an action so as to achieve the best long-term reward. One of the most important assumptions in MDP is that the state of the environment is fully observable. This, however, is unfit to a lot of real-world problems. Partially observable Markov decision process (POMDP) generalizes MDP in which the decision process is based on incomplete information about the state. A POMDP model is essentially equivalent to that of MDP with the addition of a finite set of observations $\mathcal{Z}$ and a set of corresponding observation probabilities $\mathrm{O} : \mathcal{S} \times \mathcal{A} \rightarrow \Pi(\mathcal{Z})$. Thus, a standard POMDP model is described by a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{Z}, \mathrm{T}, \mathrm{O}, \mathrm{R} >$, usually companied with two parameters the discount factor $\gamma \in [0, 1]$ and the initial belief. The policy of a POMDP can then be understood as a mapping from histories of observations to actions.

The MDP problem can be solved with $O(|\mathcal{S}|^2)$ complexity where $|\mathcal{S}|$ denotes the number of states with value iteration or policy iteration the popular approaches. For POMDPs, the policy is defined over the *belief* state which can be taken as a probability distribution over the unobservable real states as an effective summary of the observation history. The belief state is updated based on the last action, the current observation, and the previous belief state using the Bayes rule. The policy of a POMDP is thus a mapping from a belief state to an action. The space of belief state is continuous. Obviously, the complexity of computing the optimal policy for a POMDP is much higher than that of an MDP. The best bound for obtaining the exact solution is doubly exponential in the horizon [3]. For large-scale POMDP problems, it is computationally infeasible even though it is known that the value function can be proven piecewise linear and convex (PWLC) over the internal state space [1].

With the conjecture that temporally close belief states for navigation related problems should possess a low intrinsic degree of freedom, this paper proposes to cluster belief states by considering belief states' spatio similarity and their temporal order in a given belief state sample. The expected outputs are clusters of belief states, each characterized by a much lower dimensional space. The proposed method has been

tested using a synthesized navigation problem and empirically shown to be a promising direction towards solving large-scale POMDP problems.

The remaining of this paper is organized as follows. Section 2 provides the background on belief compression. Section 3 describes the proposed belief state clustering technique. Experimental results are reported in Section 4 with possible extensions found in Section 5. Section 6 concludes the paper.

2 Belief Compression

In the literature, there exist a number of different methods proposed to solve large-scale POMDP problems efficiently. They include the witness algorithm [1], VDC algorithm [11], BFSC alogrithm [12], etc. Belief compression is one recently proposed paradigm [8] which reduces the sparse high-dimensional belief space to a low-dimensional one via projection. The principle behind is to explore the redundancy in computing the optimal policy for the entire belief space which is typically sparse. Using a sample of belief states of a specific problemn computed based on the consecutive observations, data analysis techniques like exponential principal component analysis (EPCA) can be adopted for characterizing the belief states and thus the corresponding dimension reduction. This paradigm has been found to be effective in making POMDP problems much more tractable.

Let S denote the set of true states, $\mathbb{B}$ denote the belief state space of dimension $|S|$, $b_i \in \mathbb{B}$ denote the belief state where its j^{th} element $b_i(j) \geq 0$ and $\sum_{j=0}^{|S|} b_i(j) = 1$, B denote a $|S| \times n$ matrix defined as $[b_1|b_2|...|b_n]$ where n is the number of belief states in the training sample.

According to [8], one can apply EPCA and obtain a $|S| \times l$ transformation matrix U which factors B into the matrices U and $\widetilde{B}$ such that

$$B \approx e^{U\widetilde{B}} \tag{1}$$

where each column of B equals $b \approx b^r = e^{U\widetilde{b}}$ and the dimension of $\widetilde{B}$ is $l \times n$. As the main objective of U is for dimension reduction, it is typical that $l << |S|$.

For computing the policy, an MDP using the sampled belief states as its true states is created. The corresponding transition probabilities have to be estimated based on those of the original POMDP. Then, the value iteration method can be used to compute the optimal policy for the MDP which then becomes the policy for the POMDP. This way of approximation has been proven to be bounded-error [4]. It is to noted that due to the nonlinear projection using EPCA, the PWLC property of the POMDP value function is lost and thus some existing efficient exact algorithms no longer fit to solving the POMDP problem after belief compression.

3 Clustering Belief States for Problem Decomposition

3.1 Motivation

We believe that the contribution of belief compression is not merely yet another technique to address the POMDP's scalability issue. It in fact opens up a new dimension for tackling the problem and shows the possibility to apply data analysis techniques to the belief state space to result in more elegant problem solving tricks. As mentioned in [8], the efficiency of belief compression is owing to its strategy of "compression" the high-dimensional belief state which is one of the main causes for the exponential complexity.

To further exploit the dimension reduction paradigm, we propose to partition the entire belief space by analyzing the manifold of a sample of belief states instances. By clustering the belief states sample, dimension reduction techniques can be applied to individual clusters of belief states instead of the overall belief state sample. With the assumption that the instrinsic dimensions of the belief state clusters are much lower than that of the original sample owing to problem regularities found in a lot of real applications, it is anticipated that the dimensional reduction per cluster will be more significant, and thus the subsequent complexity for solving the POMDP problem can be further reduced. This idea can intuitively be explained as exploitation of the structural modularization of the belief state domain.

For related work, it is to be noted that the proposed belief state clustering has some analogy with POMDP decomposition. However, in the literature, most of the proposed POMDP decomposition techniques [10] focus on analyzing the original states of the POMDP, instead of based on the statistical properties of belief state occurrence as what being proposed in this paper. In [13], an MDP state aggregation algorithm has been proposed, which however focus on clustering the true states with similar features. For POMDPs, we perform clustering in the belief state space. Also to contrast with the standard clustering problem where the typical objective is to identify data clusters with large inter-cluster distances and small intra-cluster distances, the objective here is to cluster the belief states in such a way that at least the resultant clusters of belief states can further be compressed when compared with the compressing all the belief states. In this paper, with the assumption some regularities should exist for temporally close belief states, we propose to cluster the belief states based on both the euclidean distance between them as well as their temporal difference in the belief state sample.

3.2 A Spatio-Temporal Criterion Function for Clustering

Among all the clustering algorithms, the k-means algorithm [7] is used here for the simplicity reason, where a distance function between the cluster means and the data has first to be defined. Then, data found to be closest to one of the cluster means will contribute to the update of that mean in the next iteration. The whole process will repeat until it converges. For clustering belief states with the dimension-reduction objective, we propose to achieve that implicitly by defining a spatio-temporal distance function between two belief states, given as

$$dist(b_i, b_j) = \sqrt{\|b_i - b_j\|^2 + \lambda \|\frac{i-j}{n|S|}\|^2} \tag{2}$$

where λ is a trade-off parameter for controlling the relative contribution of the first (spatio) term and the second (temporal) one. For navigational problems, a belief state provides a good indication of the spatial location of the agent. Therefore, we use "spatio" to describe the first term which measures the distance in the belief state space. The second term is essentially measuring the terporal steps apart between two belief states as indicated by their indices. $n|S|$ is introduced to normalize the second term to be within $[0, \frac{1}{n}]$. If λ is too large, it will dominate the first term and the k-means clustering results will essentially be cutting the belief state sample into some consecutive parts according to the state appearance sequence in the sample. Also, the value of k, *i.e.*, the number of clusters, is another parameter that one can tune for optimal belief state dimension reduction. To determine the values of λ and k, we only used an empirical procedure in this paper to be explained in the following. It is in fact possible to replace the k-means clustering with mixture of Gaussian so that the data partition becomes soft instead of hard and the analytical derivation of optimal λ could be possible. This part will further be pursued in the future due to the promising empirical results we obtained.

The optimality of k and λ should be defined based on some criterion function which measures the difference between the original belief states and the reconstructed belief states after belief compression is applied. As each belief state is a probability distribution, Kullback-Leibler (KL) divergence (KL-divergence has been shown that this is asymmetry, and there are alternative measures that could be used for instance the Kontorovich metric and the total variation metric, we will extend this part in future work)could be used for evaluating the discrepancy between the original belief states and the reconstructed belief states, given as

$$\overline{KL}(B) = \frac{\sum_{i=1}^{n} KL(b_i \| b_i^r)}{n} \tag{3}$$

$$KL(b_i \| b_i^r) = \sum_{j=1}^{|S|} b_i(j) \ln \left(\frac{b_i(j)}{b_i^r(j)} \right). \tag{4}$$

3.3 Multiple Transformation Matrices

For the original belief compression, the compression is based on primarily one transformation matrix U as described in Section 2. Now, as the belief states are clustered, there will be several transformation matrices, each corresponding to a particular cluster. Let the belief state sample be partitioned into P clusters $\{C_1, C_2, ..., C_P\}$ and the transformation matrix of the p^{th} cluster C_p to be U_p. The reconstructed belief states associated to C_p can then be approximated as $b^{r,C_p} = e^{U_p \widetilde{b}}$. To measure the dimension reduction effectiveness via the clustering, the KL divergence per cluster is to be computed, given as

$$\overline{KL}(C_p) = \frac{\sum_{b_j \in C_p} KL(b_j \| b_j^{r,C_p})}{|C_p|}. \tag{5}$$

Before proceeding to the section on experimental results, we would like to highlight the fact that clustering the belief states can result not only in reducing the overall complexity for solving the original POMDP problem, but also that for performing the EPCA for belief compression and that for computing of the transition probabilities for the belief state pairs. This computational gain is achieved at the expense of the clustering overhead as well as the optimality of the resulting policy that we may sacrify after the problem decomposition. Fortunately, the clustering overhead is found to be not significant when compared with the overall complexity. For the resulting policy's optimalty, further evaluation will be needed.

4 Experimental Results

4.1 The Hallway2 Problem

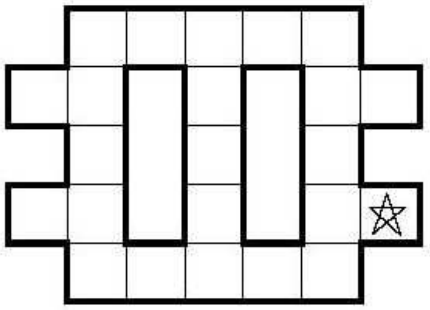


Figure 1: Navigation environment with 92 states.

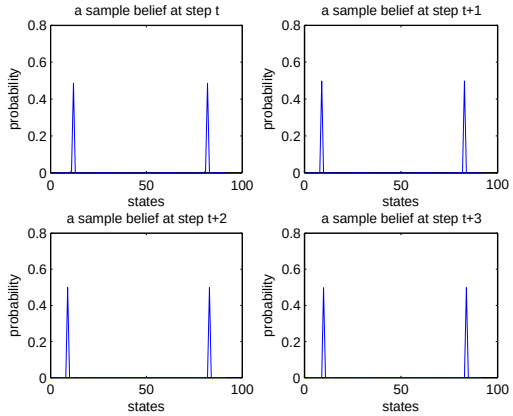


Figure 2: Belief states sampled in consecutive time steps.

The Hallway2 Problem which is constructed by [6] to model a robot navigation domain with a specific maze is commonly used to test the scalability of algorithms for solving POMDP problems (see Figure 1). The problem is to find the goals in the maze with 92 states (4 possible orientations in each of 22 rooms, and 4 being the goal states which are 4 possible orientations in the star room), and contains 5 actions (stay in place, move forward, turn right, turn left, turn around) and 17 types of observations (all combinations of walls, plus "reaching goal"). Reaching one of the goal states will yield a $+1$ reward and then the next state will be set to a random non-goal state. In addition, it is assumed that all the non-goal states of the problem are equally likely to be the initial state location and thus the starting belief state is $b_1 = (\frac{1}{88}, ..., \frac{1}{88}, 0.0, 0.0, 0.0, 0.0, \frac{1}{88}, ..., \frac{1}{88})^T$. The zeros are corresponding to four goal states. Also, the discount factor used is 0.95. In this paper, all the experimental results reported are based on this problem setting.

4.2 Belief State Sampling

The process of belief compression is operated on a belief state sample generated via simulation. During the simulation for sample generation, two levels of random numbers are used to select an action (one level is used to determine when to call MDP program, another level is used to determine the random action), and the Bayes rules are used to evolve the belief states. When one random number is found to be less than the threshold defined as 0.5, another random number will be generated to decide the next action. Otherwise, it will sum up all the beliefs generated so far and take the state with the maximal sum of probabilities as the current state. Then, an MDP solver will be called to get the corresponding policy table to choose the next action for its 'current state'.

The belief states in consecutive time steps often have similar shape with the same number of modes (see Figure 2). Obviously, these "structurally" similar belief states could have them represented with a much lower dimension. That's why the belief state space is often considered to be sparse.

4.3 Performance of Belief State Clustering

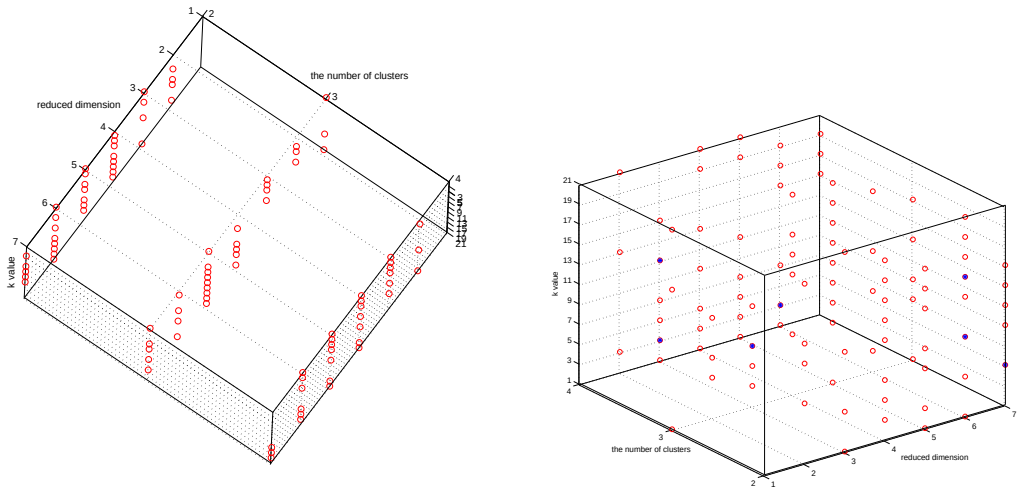
The first experiment focuses on evaluating the effectiveness of the proposed spatio-temporal clustering scheme for overall dimension reduction. We enumerated a set of different values for the trade-off parameter λ and evaluated the corresponding dimension reduction performance. For performance measurement, we contrasted the values of the KL-divergence between the set of original belief states and the ones reconstructed based on the conventional belief state compression (*i.e.*, without clustering) and the one we proposed in this paper with belief state clustering. Figure 3(a) shows a lattice with the three axes being the reduced dimension, the number of clusters and k value. We only plot the lattice points which correspond to the parameter setting resulting in the averaged KL divergence of the clusters being less than the averaged KL divergence using the conventional belief compression. According to Figure 3(a), it is noted that a number of settings can result in better overall dimension reduction. Among those settings, we set a filter and highlight those with high reduction as shown in Figure 3(b). The filtering is based on a ratio R which illustrates the degree of the KL-divergence's reduction, defined as

$$R(\lambda, l, P) = 1/P * \sum_{p=1}^{P} \frac{(KL_U(\lambda, l, C_p) - KL_{U_p}(\lambda, l, C_p))}{KL_U(\lambda, l, C_p)} \tag{6}$$

where $KL_U(\lambda, l, C_p)$ stands for the KL-divergence between the original belief states in the p^{th} cluster and the corresponding reconstructed belief states based on only U (original EPCA), and $KL_{\{U_p\}}(\lambda, l, C_p)$ stands for the KL-divergence between the original belief states in the p^{th} cluster and the corresponding reconstructed belief states based on U_p (resulted from applying EPCA to the cluster).

In Figure 3(b), the filled points show the parameter settings with $R > 0.95$. Among them, the operation point $R(3, 4, 3)$ was chosen for the subsequent experiments. This point is equivalent to the situation that the belief state sample is partitioned into 4 clusters, its dimension is reduced to 3, and the trade-off parameter λ is 3. Figure 4 shows the reconstructions of a particular belief state using the conventional EPCA (Figure 4(a)) and the proposed clustered EPCA (Figure 4(b)). The latter one's reconstruction can almost completely recover the original belief state and is more accurate when compared with the former one. Also, Figure 5 shows the temporal sequence of the belief states in each cluster. It is noted that in some clusters (e.g., Cluster 1 and 3), the belief states under them are only partially ordered, which is consistent to the spatio-temporal criterion function used.

Table 1 tabulates the performance comparison using the KL divergence given the belief state dimension is reduced to three. Obviously, the values of the KL divergence obtained using the proposed spatio-temporal clustering were much lower than the case using only EPCA. Figure 6 shows the comparison of the KL-divergence averaged over all sampled beliefs at different reduced dimensions using EPCA and the KL-divergence using proposed method. Note that our proposed method achieves more accurate reconstruction than that of using conventional EPCA. In addition, our proposed method is much more timesaving than the conventional EPCA. As reported in the rightmost column of Table 1, our proposed method took 71.94 seconds while the conventional EPCA took 153.08 seconds.



(a) Parameter settings with better overall dimension reduction.

(b) The best parameter settings with $R > 0.95$.

Figure 3: Parameter settings

5 Discussion and Future Works

This paper mainly demonstrates the possibility of clustering the belief states in a spatio-temporal manner for achieving further belief state compression. We are currently working on the evaluation of the optimality of the policy computed based on the proposed belief state clustering. Some experiments on evaluating the optimal policy based on the proposed approach has been accomplished with positive results(more details can be found at http://www.comp.hkbu.edu.hk/tech-report/tr05006f.pdf). Some future extensions of this work can include at least the following two directions.

5.1 Hierarchical POMDP Decomposition

Hierarchical POMDP (HPOMDP) Decomposition has recently been proposed for decomposing a POMDP problem into a hierarchy of related POMDP models of reduced size. PolCA [10] is a representative example where the decomposition is resulting from the decomposability of the action domain. The limitation of HPOMDP is that the decomposition is not fully automatic, where the underlying hierarchy of actions requires knowledge of domain experts. In other words, this is domain-specific. Also, the decomposition is not based on the belief states. However, it would be interested to see if the notion of hierarchical decomposition can be incorporated into the proposed spatio-temporal clustering framework.

5.2 The Multi-Agent Consideration

Given a POMDP decomposition, one can model each sub-POMDP as a problem solving agent for the specific sub-problem. As the decomposition based on the proposed belief state clustering may not result in a set of sub-POMDP problems which are equivalent to the original POMDP problems, interaction between those multiple agents for achieving the overall optimal policy is an important research issue to look at. Nash Equilibrium is an important concept commonly used in multi-agent learning [5] for solving decentralized MDP [2] and POMDP problems [9]. Our research agenda is to investigate how this paradigm can be applied to our decomposition scheme. The basic idea is that every agent would conjecture other agents' behaviors and give the best response to other agents from its local view. A Nash equilibrium usually would not deduce the optimal policy. However, it should be able to guarantee a not-too-bad sub-optimal one.

What being described so far assumes that the whole model of the decision process is known. That is, we have the perfect knowledge about the reward function, transition function and observation function. Solving the corresponding POMDP problems is an off-line process. It is also interesting to see how the multi-agent approach can be extend to support online learning (e.g., Q-learning [14]) for POMDP under partial observation scenarios.

	# belief states	Original EPCA	Proposed Method	Comp.Cl Cost (sec.)
$Cluster1$	96	1.3997	0.0024	1.84
$Cluster2$	16	0.4998	0.0004	0.34
$Cluster3$	36	0.1893	0.0003	0.49
$Cluster4$	352	4.2596	0.4938	69.27

Table 1: Performance comparison between the conventional EPCA for belief compression and the proposed method, where the number of clusters is 4, the reduced dimension is 3 and $\lambda = 3$.

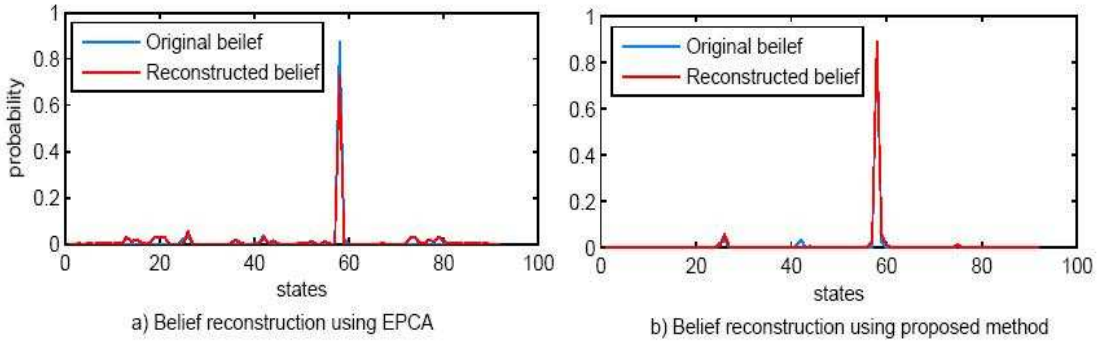


Figure 4: Reconstructed belief states using the conventional EPCA and the proposed method.

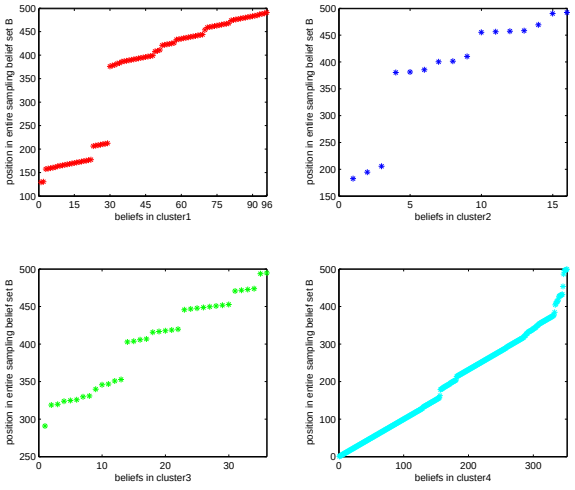


Figure 5: Temporal sequence of the belief states in the four clusters.

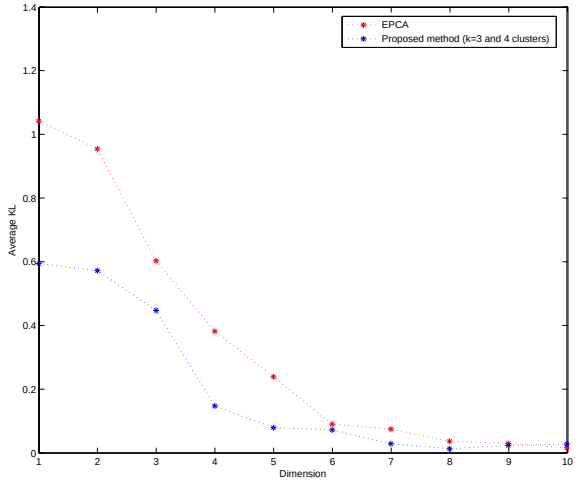


Figure 6: Comparison of average KL Divergences using conventional EPCA and the proposed method.

6 Conclusion

This paper extends the recently proposed belief compression by introducing a spatio-temporal belief state clustering for addressing large-scale POMDP problems. It was found that the proposed the spatio-temporal method can further compress the belief states in each cluster to a much lower dimension with similar belief state reconstruction accuracy. Future research directions include at least hierarchical clustering of the belief states and investigating the integration of the proposed belief state clustering and the multi-agent paradigm as a complete solution for solving large-scale POMDP problems.

Acknowledgements

We would like to thank the anonymous reviewers' valuable comments and insight into this work. This work has been partially supported by RGC Central Allocation Group Research Grant (HKBU 2/03/C).

References

[1] A.Cassandra. *Exact and approximate algorithms for partially observable Markov decision processes*. U.Brown, 1998.

[2] R. Becker, S. Zilberstein, V. Lesser, and C. V. Goldman. Transition-Independent Decentralized Markov Decision Processes. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi Agent Systems*, pages 41–48, Melbourne, Australia, July 2003. ACM Press.

[3] D. Burago, M. de Rougemont, and A. Slissenko. On the complexity of partially observed Markov decision processes. *Theoretical Computer Science*, 157(2):161–183, 1996.

[4] G. J. Gordon. Stable function approximation in dynamic programming. In A. Prieditis and S. Russell, editors, *Proceedings of the Twelfth International Conference on Machine Learning*, pages 261–268, San Francisco, CA, 1995. Morgan Kaufmann.

[5] M. P. W. Junling Hu. Nash q-learning for general-sum stochastic games. *Journal of Machine Learning Research*, 4:1039–1069, 2003.

[6] M. L. Littman, A. R. Cassandra, and L. P. Kaelbling. Learning policies for partially observable environments: Scaling up. pages 495–503. 1997. (Reprinted from *Proceedings of the 12th International Conference on Machine Learning, 1995*).

[7] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *5th Berkley Symposium on Mathematics and Probability*, pages 281–297, 1967.

[8] N. Roy, G. Gordon and S. Thrun. Finding approximate POMDP solutions through belief compressions. *Journal of Artificial Intelligence Research*, 23:1–40, 2005.

[9] R. Nair, M. Tambe, M. Yokoo, D. Pynadath, and S. Marsella. Taming decentralized pomdps: Towards efficient policy computation for multiagent settings. 2003.

[10] J. Pineau and S. Thrun. An integrated approach to hierarchy and abstraction for pomdps. Technical Report CMU-RI-TR-02-21, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, August 2002.

[11] P. Poupart and C. Boutilier. Value-directed compression of pomdps. In S. T. S. Becker and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 1547–1554. MIT Press, Cambridge, MA, 2003.

[12] P. Poupart and C. Boutilier. Bounded finite state controllers. In S. Thrun, L. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

[13] S. P. Singh, T. S. Jaakkola, and M. I. Jordan. Reinforcement learning with soft state aggregation. In G. Tesauro, D. S. Touretzky and T. K. Leen, (Eds.), ¡i¿Advances in Neural Information Processing Systems (NIPS) 7¡/i¿, Cambridge, MA: MIT Press, 1995., 1995.

[14] C. Watkins. *Learning from Delayed Rewards*. PhD thesis, Cambridge Univ., Cambridge England, 1989.

Human Sensor Model for Range Observations

Tobias Kaupp, Alexei Makarenko, Fabio Ramos and Hugh Durrant-Whyte

ARC Centre of Excellence for Autonomous Systems (CAS)
The University of Sydney, Australia
www.cas.edu.au
{t.kaupp, alexei, f.ramos, hugh}@cas.edu.au

Abstract

This paper presents the design of a probabilistic model of human perception as an integral part of a sensor network. Human operators are regarded as information sources cooperating with robotic platforms, together forming a heterogeneous team. A human sensor model converts raw human observations into a probabilistic representation suitable for fusion with the belief maintained by the sensor network. The initial model is built offline based on calibration experiments involving multiple human subjects. At runtime, this averaged model is used to convert raw human observations into a probabilistic form. Due to the possibly wide variation in performance levels of individual operators, the human sensor model needs to be adapted at runtime. Since the true feature state is unknown, the network estimate is used for online model adaptation. Results of an outdoor calibration experiment using range and bearing observations are presented. This data is used to build a conditional Gaussian sensor model with linearly increasing standard deviation. Simulations show the online adaptation of the human sensor model when the operator's estimates include a bias.

1 Introduction

This paper considers the problem of incorporating information perceived by human operators into a probabilistic feature description produced by a Sensor Network (SN). The SN under consideration represents a solution to the problem of Distributed Information Gathering (DIG) [7]. The SN's goal is to maintain a common estimate of a distributed phenomenon which can be described by a state vector x. Typically, robotic platforms collect information through observations with their on-board sensors. This approach is extended by incorporating human operators observing the phenomenon through their senses.

This problem is motivated by many automated applications where human operators remain the center of the system, mostly at a supervisory or decision making level. These areas would benefit from incorporating additional information perceived by humans, e.g. in search and rescue, bush fire fighting, and defence. When observing complex environments, the observations made by robots and humans may be complementary. Robotic sensors perform well with low-level representations such as geometric properties. In contrast, human operators can be more valuable in identifying more abstract properties such as class labels.

The SN considered here uses Decentralized Data Fusion (DDF) to build a common representation of the environment [7]. Physically, the network is formed by a heterogeneous team of outdoor robotic platforms and human operators. All members of the team must communicate using a common language which is defined in terms of probability distributions on the state of the environment.

The focus of this paper is on human operators contributing information to the global belief which requires the SN to have a probabilistic model of human perception. This *Human Sensor Model* (HSM) converts raw human observations into likelihoods using probabilistic modeling techniques. The HSM is built offline based on experiments involving several human subjects. Since operators are likely to differ in their performance, an online adaptation of the model to *individual* operators is required. In the online phase, however, the true feature state is unknown. Instead, the DDF belief is used for model adaptation.

The proposed methodology is implemented for static point features with location uncertainty. Both robotic sensors and human operators make raw range and bearing observations of the features. Estimating

feature locations in a mixed human/robotic network is considered in this work because it is a prerequisite for contributions of any other feature property human operators may observe. The implementation is part of the ANSER II project which aims at demonstrating DDF techniques for natural features using a heterogeneous outdoor SN.

2 Operators in Sensor Networks

The architecture of the SN used in our experiments is called Active Sensor Network (ASN) which is a realization of a *macro* SN (MSN) [4]. Typical MSN's are comprised of a number of platforms capable of executing complex algorithms such as data fusion. ASN's decentralized architecture results in the system properties of *scalability*, *robustness* and *modularity*.

Conceptually, the ASN architecture is composed of interacting components. Several component types are identified which realize certain interfaces. The design of the ASN architecture emphasizes interaction between components to achieve extensibility in component types. ASN components are capitalized throughout the remainder of this paper.

The ASN architecture favors *peer-to-peer* interaction between human operators and other SN components [8]. Operators communicate with the SN through the USER INTERFACE. It is used to fulfill two main objectives of human-SN interaction defined in terms of information flow: (1) to present the human operator with the global belief state (network-to-human, operator as information *sink*), and (2) to allow human operators to contribute information to the network (human-to-network, operator as information *source*).

This paper discusses the human-to-network information flow. Human observations are given in a raw form which requires the translation into a language the fusion NODE's of the ASN understand. The NODE's perform Bayesian filtering according to the following equation:

$$P(x \mid Z^k) = \alpha P(z_k \mid x; \Theta) P(x \mid Z^{k-1}) \tag{1}$$

where Z^k represents all raw observations up to time step k, α is a normalizing constant, and z_k is a raw observation obtained at t_k. $P(z_k \mid x)$ represents a distribution over the values of the true state x which is referred to as *likelihood*. It is generated using the sensor model's parameters Θ. Likelihoods are generated by robotic SENSOR's such as lasers or by the USER INTERFACE which converts human raw observations. Likelihoods are subsequently combined with the belief from the previous time step $P(x \mid Z^{k-1})$.

3 Offline Creation of the Human Sensor Model

In order to find a model of human perception, experiments with multiple human subjects are required. Other options include using expert opinions or results from experiments reported in the literature [6]. Since we are interested in a probabilistic model, this offline phase determines the type of probabilistic representation and the initial model parameters adequate for the particular problem.

The example we have implemented is location estimation of point features in an outdoor environment. The HSM is used in the ANSER II project where human operators, ground, and air vehicles cooperatively build a map of rich features. Humans also observe more complex feature properties but in order to do this, they need to specify the location of the features.

3.1 Calibration Experiment

In a calibration experiment eight poles were placed in an open space with known coordinates. Twenty-one human subjects were asked to estimate range and bearing to each pole used in the experiment. Fig. 1(a) shows the true location of the poles, mean estimates and single observations of all subjects. The ellipses in the graph indicate a 2-σ standard deviation assuming a Gaussian distribution.

The evaluation of the experiment focuses on the range estimates. Decoupling range and bearing estimates implies the assumption that there is no correlation between them. The results, both qualitatively and quantitatively, can be summarized as follows:

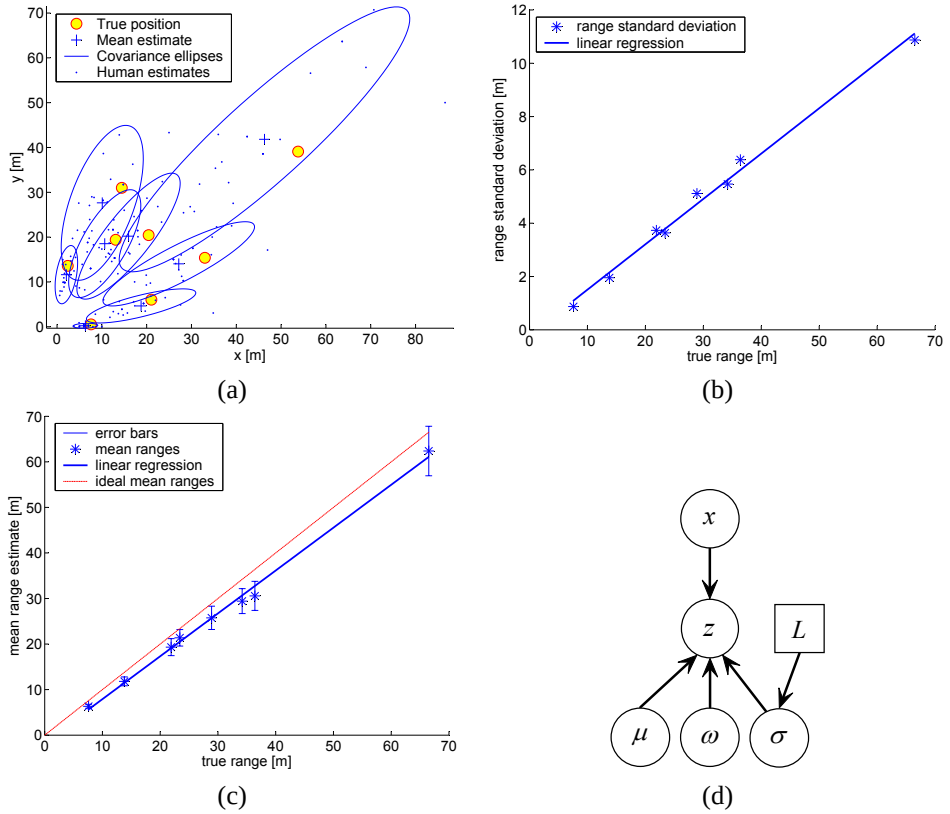


Figure 1: Results of the calibration experiment: (a) all range/bearing estimates of poles, Gaussian uncertainty ellipses around means; (b) range standard deviation increases linearly with range; (c) mean range estimates are linear to true ranges, ranges are underestimated, error bars indicate increasing standard deviations; (d) a BN representation of the HSM.

(1) The standard deviations of the range estimates can be approximated as a linear function of range, i.e. uncertainty increases with range as shown in Fig. 1(b). (2) The mean range error can be approximated as a linear function of range. On the average, the range was underestimated for all the poles as shown in Fig. 1(c). (3) Range estimates can be approximated by a Gaussian distribution as verified through a normal probability plot of all range measurements.

Researchers in the field of psychology have also investigated the topic of human distance estimation. Baird *et al.* demonstrated that the relationship of perceived distance and actual distance, on average, can be described by a power function with an exponent of approximately 1.0 and thus very close to being linear as our results suggest [1]. Da Silva *et al.* found that it is slightly greater than 1.0 with indoor observations and generally slightly less than 1.0 with outdoor observations [5].

3.2 Probabilistic Human Sensor Model

Based on our experimental data, it is possible to build a probabilistic sensor model with initial model parameters. For a HSM, the goal is to produce a likelihood of x given a raw human observation z.

The HSM for range is graphically represented as a Bayesian Network (BN) as shown in Fig. 1(d). This representation is adopted to show the conditional independencies of the random variables and their parameters. It is common practice to include the nodes' parameters as additional random variables in the graph if they are learned offline or adapted at runtime which both apply to our problem.

The BN consists of the following nodes: node x represents the range to the target and is encoded as a Gaussian *pdf*. Node z represents the human range observation and is encoded as a conditional Gaussian

density function. Nodes μ, σ and ω are mean, standard deviation and regression coefficient of the conditional Gaussian sensor model. Node *L* stands for *Location* and is required to encode the range-dependent uncertainties of the range observations. Although this variable is continuous it is discretized into several bins for implementation purposes.

The initial parameter estimates based on the results of the calibration experiment are: $\hat{\mu}$=0, $\hat{\sigma}$=[2 4 6] (dependent on value of *L*), and $\hat{\omega}$=0.9. The model can now be used for inference of range at runtime.

4 Online Usage of the Human Sensor Model

After the BN is initialized, it can be used in the *online phase* to yield estimates of the state x. However, individual operators differ in their performance and it is desirable to adapt the HSM to the particular operator.

4.1 Online Learning Algorithm

We suggest that the model parameters Θ should be updated online. The algorithm consists of two steps for each time slice:

(1) *Belief update on x:* Equation 1 is used to yield a posterior over x incorporating a local likelihood.

(2) *Adapting Θ:* A sample pair $\langle x_k, z_k \rangle$ is used to update a distribution on Θ at time step k:

$$P(\Theta \mid X^k, Z^k) = \frac{P(x_k, z_k \mid \Theta) P(\Theta \mid X^{k-1}, Z^{k-1})}{\int P(x_k, z_k \mid \Theta) P(\Theta \mid X^{k-1}, Z^{k-1}) d\Theta} \tag{2}$$

where X^k represents all samples of x up to time step k.

Equation 2 refers to the case of parameter learning for Θ in a Bayesian approach. Computationally cheaper methods can be used such as *Maximum Likelihood* (ML) or *Maximum A Posteriori* (MAP). These methods choose a particular parameter set Θ at each time step rather than keeping a distribution on Θ:

$$\Theta^{ML} = \arg\max_{\Theta} P(x_k, z_k \mid \Theta) \tag{3}$$

$$\Theta^{MAP} = \arg\max_{\Theta} P(x_k, z_k \mid \Theta) P(\Theta \mid X^{k-1}, Z^{k-1}) \tag{4}$$

The ML method only uses online data to yield an estimate of the parameters. The MAP method, however, takes a prior belief about the parameters into account. Particularly well suited priors over parameters are called *conjugate priors* [2]. Since the offline phase determines initial parameters, it is possible to define a reasonable prior distribution on Θ.

The online phase yields likelihoods on Θ produced by data samples $\langle x_k, z_k \rangle$. However, unlike in the offline phase, the true state is unknown. Instead, a posterior belief of x is used to obtain values for x_k as further described in Sec. 4.3.

4.2 Parameter Learning for a Gaussian Density

This section presents the derivation of Eq. 3 and 4 for the special case of a Gaussian *pdf*. The conditional Gaussian model from Fig. 1(d) can be represented as a single two-dimensional Gaussian variable $y = \begin{pmatrix} x \\ z \end{pmatrix}$. The *pdf* including its parameters μ_y and Σ_y is:

$$P(y; \mu_y, \Sigma_y) = \frac{1}{\sqrt{2\pi}\,|\Sigma_y|^{\frac{1}{2}}} e^{\frac{1}{2}(y-\mu_y)^T \Sigma_y^{-1} (y-\mu_y)} \tag{5}$$

The conjugate prior of a Gaussian distribution is a Normal-Wishart *pdf* [2]. The parameters of this distribution are (m, τ) for the Normal and (Λ, α) for the Wishart. They are referred to as the *hyperparameters* of the two-dimensional Gaussian y. The parameters' prior can be written as:

$$P(\mu_y, \Sigma_y) = P(\mu_y) P(\Sigma_y \mid \mu_y) = \mathcal{N}(\mu_y; m, \tau^{-1} I_2) \mathcal{W}(\Sigma_y | \mu_y; \Lambda, \alpha) \tag{6}$$

The log likelihood (ML) and unnormalized log posterior (MAP) are given by:

$$L^{ML} = \log(\mathcal{N}(y; \mu_y, \Sigma_y)) \tag{7}$$

$$L^{MAP} = L^{ML} + \log(\mathcal{N}(\mu_y; m, \tau^{-1} I_2)) + \log(\mathcal{W}(\Sigma_y | \mu_y; \Lambda, \alpha)) \tag{8}$$

The parameter estimates can be computed by setting the derivative of L^{ML} and L^{MAP} with respect to μ_y and Σ_y to zero. The final results are:

$$\hat{\mu}_y^{MAP} = \frac{\sum_t E_t[y] + \tau m}{\tau + N} \quad ; \quad \hat{\mu}_y^{ML} = \frac{\sum_t E_t[y]}{N} \tag{9}$$

$$\hat{\Sigma}_y^{MAP} = \frac{\Lambda + \tau(\mu_y - m)(\mu_y - m)^T + \sum_t E_t[(y-\mu_y)(y-\mu_y)^T]}{\alpha - 2 + N} \quad ; \quad \hat{\Sigma}_y^{ML} = \frac{\sum_t E_t[(y-\mu_y)(y-\mu_y)^T]}{N} \tag{10}$$

where N is the number of data samples. $E_t[y]$ represents the expectation and $E_t[(y-\mu_y)(y-\mu_y)^T]$ the covariance of y at time step t.

It remains to calculate the parameters of the conditional Gaussian distribution:

$$\hat{\mu} = \mu_{y,z} \quad ; \quad \hat{\sigma} = \sqrt{\Sigma_{zz} - \frac{\Sigma_{xz}^2}{\Sigma_{xx}}} \quad ; \quad \hat{\omega} = \frac{\Sigma_{zx}}{\Sigma_{xx}} \tag{11}$$

4.3 Efficient Adaptation

As stated in Sec. 4.1, the true belief is not available for online learning. However, the DDF network provides a common estimate generated from multiple sensors. We propose to use the common DDF network's estimate to adapt the HSM. However, the common state cannot be used naively since it includes *a priori* fused human (biased) information.

The DDF network maintains different estimates of the state of the world depending on the source of information: local belief x_i based on local sensor observations; global DDF belief x based on all sensor observations; global DDF belief excluding local information x_{-i}.

We suggest to use all DDF network information *excluding* local information to find $\langle x_k, z_k \rangle$ sample pairs to efficiently adapt the model parameters. This approach assumes that human operators and other sensors observe the same states.

Fig. 2 shows the collaboration of the USER INTERFACE and NODE i. Arrows indicate the information flow. An incoming message from the NODE's point of view consists of a likelihood and triggers the execution of step (1) of the algorithm from Sec. 4.1. An incoming message from the USER INTERFACE's point of view consists of the combined DDF posterior and triggers the execution of step (2). Local information needs to be removed from the posterior.

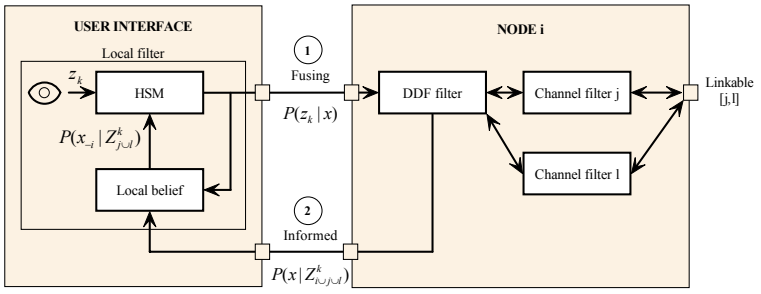


Figure 2: Internal functionality of NODE and USER INTERFACE. Information flow is indicated by arrows. Incoming messages trigger the steps of the algorithm from Sec. 4.1.

To demonstrate the proposed algorithm described in Sec. 4.1 using efficient adaptation, a one-dimensional simulation is presented. A DDF network with two NODE's (one SENSOR each) is created. One SENSOR has

a correct model and observations are generated according to that model. The other SENSOR has a constant bias which means its sensor model has an additive error. It is also wrong in the standard deviation, i.e. the sensor's uncertainty is larger than the sensor model assumes. Both sensor model parameter errors need to be corrected. The *pdf* over x is a Gaussian. The density function over z is a conditional Gaussian with parameters μ, σ and ω. The biased SENSOR adapts its sensor model's mean μ and standard deviation σ at each time step to learn the constant additive bias and the larger standard deviation using Eq. 9-11.

Fig. 3 shows the progress of learning a bias of absolute value 3 and an uncertainty of absolute value 4 (the initial value is 2). Each plot displays the adaptation using three different estimates of x: the local belief, the global DDF belief naively and the global belief excluding local information. The results show that the best way for model adaptation is to use the suggested method of using the global belief excluding local information.

The top two rows of Fig. 3 show learning using the MAP method. The behaviour of the learning curve depends on the choice of the hyperparameters defined in Eq. 6. The graphs on the top row show faster learning compared to the graphs on the second row since we "trust" the initial (offline) parameters more by defining them as more certain. The bottom row shows the ML graphs which are less smooth than the MAP graphs because they only take the online data into account.

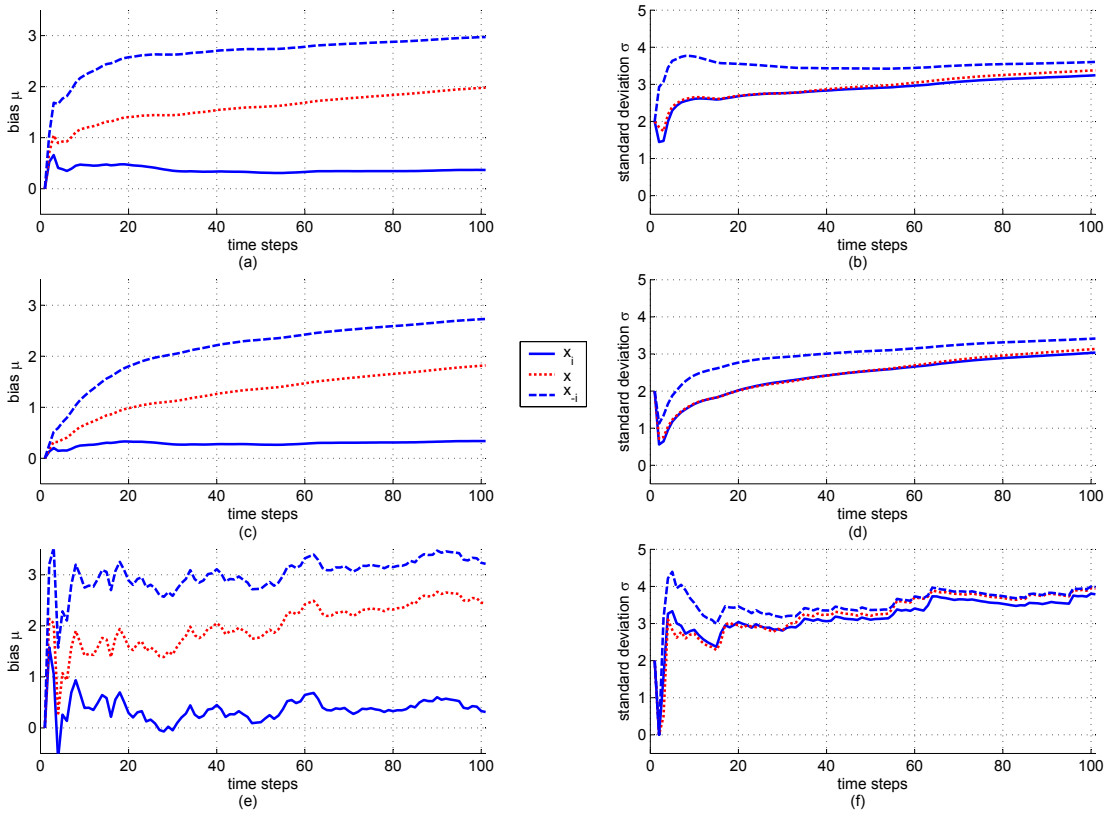


Figure 3: Progress of learning an additive bias of absolute value 3 and a standard deviation of absolute value 4 using different beliefs: x_i, x and x_{-i}. (a),(b): MAP method; (c),(d): MAP method using "stronger" hyperparameters; (e),(f): ML method.

Fig. 4 once more illustrates efficient adaptation. It shows that the errors of the system estimates get within the expected (fused) variance bounds σ faster when the global belief excluding local information is used for adaptation.

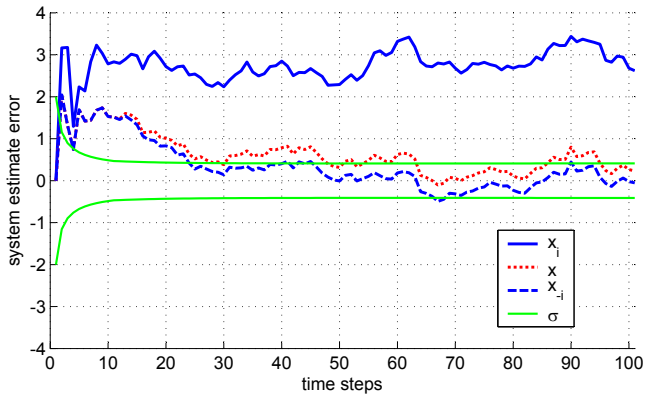


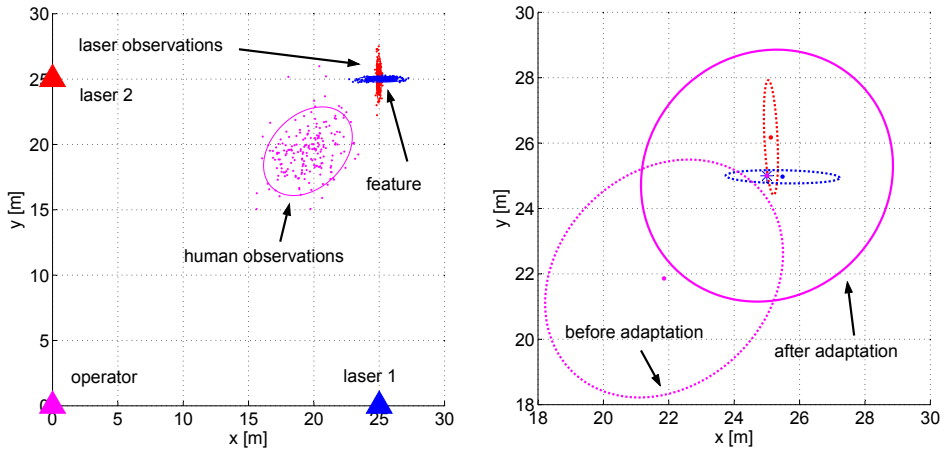
Figure 4: Errors in the system estimate when the HSM is adapted using beliefs x_i, x and x_{-i}. The errors get within the expected variance bounds σ faster when x_{-i} is used.

4.4 Model Adaptation of the HSM for Range

This section shows a practical example for online adaptation of the HSM for range which has been learned offline as presented in Sec. 3.2. In this scenario, a mixed human/robotic team observes a point feature location in a global coordinate system.

The DDF network consists of two NODE's. Two laser SENSOR's submit their likelihoods to NODE 1. It is connected to NODE 2 which receives likelihoods from a USER INTERFACE. This example only adapts parameter μ, i.e. ω and σ are kept according to the offline model.

Fig. 5(a) shows the deployment of the scenario. The two lasers are located at position (0,25) and (25,0). The human operator is positioned at (0,0). Both lasers and the operator observe a feature at position (25,25) by making range/bearing observations for 200 time steps. Laser observations are generated according to their fixed sensor model. The human range observations are generated according to the offline HSM but additionally include an additive bias of -4 m. This bias is not accounted for in the model and needs to be learned. Human bearing information is assumed to have constant uncertainty (compass readings). All observations are plotted and Gaussian ellipses are fit to the observations.



(a) (b)

Figure 5: (a) Physical deployment and raw observations; (b) likelihoods from lasers (small ellipses) and human operator (large ellipses). The two operator likelihoods correspond to before and after adaptation (dotted and solid line).

Fig. 5(b) shows likelihoods generated from raw observations. The two small ellipses are likelihoods generated by the lasers at a random time step. The bigger ellipses show two operator likelihoods: the dotted ellipse is generated before adaptation, the solid ellipse is generated after adaptation. It is clear that the bias has been learned. It is incorporated in the HSM of the particular operator as $\mu \approx -4$.

5 Conclusions and Future Work

This paper presented the design of a probabilistic model of human perception as an integral part of an outdoor SN. Human operators and robotic platforms are considered entities of a heterogenous team which cooperatively builds a representation of a distributed phenomenon. A DDF algorithm ensures that each platform has the global belief. Humans contribute information the same way as their robotic teammates. Raw observations require the conversion into a common language defined in state space which is performed by a HSM. The creation of the model involves two phases: offline and online. The initial model is built offline based on an average of many human subjects. At runtime, the model is applied to raw human observations to convert them into likelihoods. Furthermore, the common DDF belief is used to adapt the parameters of the model to individual operators since the true feature state is unknown at runtime. We investigated the feasibility of this theory for range estimation with experiments and in simulation. Specifying feature location is a prerequisite for any other feature property operators may observe.

This work has dealt with humans and robotic sensors observing the same low-level property. However, human operators perform better at observing more abstract properties. Future work will include the design of a state space with several abstraction layers which depend on each other in a hierarchical way. Human operators and different types of robotic sensors will be able to contribute information at a level which best fits their capabilities. This concept leads to the "multi-level data fusion" paradigm [3]. The BN representation is considered to be suitable for offline learning and online inference and adaptation. The work presented here will be integrated into these hierarchical models.

Acknowledgment

This work is supported by the ARC Centre of Excellence programme, funded by the Australian Research Council (ARC) and the New South Wales State Government. ANSER II is supported by BAE Systems, Bristol, UK.

References

[1] J. C. Baird and W. R. Biersdorf. Quantitative functions for size and distance judgments. *Perception and Psychophysics*, 2:891–902, 1967.

[2] J. Bernardo and A. Smith. *Bayesian Theory*. John Wiley and Sons, New York, 1994.

[3] E. Blasch and S. Plano. JDL level 5 fusion model: User refinement issues and applications in group tracking. *Aerosense*, 4729:270–279, 2002.

[4] A. Brooks, A. Makarenko, T. Kaupp, S. Williams, and H. Durrant-Whyte. Implementation of an indoor active sensor network. In *9th Int. Symp. on Experimental Robotics (ISER '04)*, Singapore, 2004.

[5] J. Da Silva and S. Fukusima. Stability of individual psychophysical functions for perceived distance in natural indoor and outdoor settings. *Perceptual and Motor Skills*, 63:891–902, 1986.

[6] M. Druzdzel and L. van der Gaag. Building probabilistic networks: "Where do the numbers come from?" Guest editors' introduction. *IEEE Trans. on Knowledge and Data Engineering*, 12(4), 2000.

[7] A. Makarenko and H. Durrant-Whyte. Decentralized data fusion and control in active sensor networks. In *7th Int. Conf. on Info. Fusion (Fusion'04)*, Stockholm, 2004.

[8] A. Makarenko, T. Kaupp, B. Grocholsky, and H. Durrant-Whyte. Human-robot interactions in active sensor networks. In *IEEE Int. Symp. on Computational Intelligence in Robotics and Automation (CIRA'03)*, 2003.

Symbolic Probabilistic-Conditional Plans Execution by a Mobile Robot

Abdelbaki BOUGUERRA and Lars KARLSSON

Department of Technology
Örebro University, SE-70182
Örebro; Sweden
http://aass.oru.se

Abstract

In this paper we report on the integration of a high-level plan executor with a behavior-based architecture. The executor is designed to execute plans that solve problems in partially observable domains. We discuss the different modules of the overall architecture and how we made the different modules interact using a shared representation. We also give a detailed description of the hierarchical architecture of the executor and how execution-time failures are handled.

1 Introduction

Carrying tasks through in real world environments presents a multitude of challenges to contemporary mobile robots. Most importantly is how to deal with the uncertainty inherent in on-board sensing, acting and lack of information. Acknowledging that classical planning, i.e. planning that assumes that the environment is static and the robot is omniscient, is not the best choice for reasoning in such environments, research has focused on developing planning approaches capable of reasoning under uncertainty and partial observability [6][2][8]. However, most developed planning approaches forget about plan execution and monitoring. Even with plans that incorporate contingencies of execution, there is still an uncountable number of unexpected events that might prevent executing the plan reliably or even invalidate it.

In this paper we report on the successful implementation of a high-level plan executor on top of a behavior-based mobile-robot control architecture. The executor is designed to handle probabilistic conditional plans and is not constrained by using a specific planner. In fact the executor can execute any plans fulfilling some representational constraints, mainly the syntax of the plans, and the specification of how to execute the plans actions. The execution system uses three hierarchical layers each with a specialized process. The top layer manages high level plans, user requests, and recovery when necessary. The middle layer has a more specialized process whose task is to execute the actions of the plan selected by the upper layer, whereas the third layer is responsible of low-level execution and monitoring. The overall system has been successfully used for research on sensor-based planning for mobile robots, most notably in the areas of perceptual anchoring [5] and active smelling [11]. One of the main strengths of our execution framework is the ability to act in partially observable domains as a result of using reasoning with symbolic planning under uncertainty. POMDP:s are by far the most used paradigm for decision making in partially observable domains, however the kind of symbolic planning that we use has certain advantages over using POMDP:s. First, symbolic planners are much faster then POMDP solving algorithms which makes replanning possible at execution-time, that is not the case with POMDP:s because policies have to be found to cover an exponential number of continuous belief states. Second, at execution time, no belief-state tracking is required, since belief states used by symbolic planners are distinguishable by their set of observations as opposed to implemented mobile-robot architectures executing POMDP policies which requires updating belief states online [15] [13], although in theory it is possible to approximate policies using finite state machines [7][12]. For an overview of planning under uncertainty including decision-theoretic planning and

symbolic planning, the reader is referred to [3]. A survey of approximate methods used to solve POMDP:s can be found in [1].

One central issue that we address is how to respond to unexpected events at execution time. Our framework provides some degree of flexibility regarding failure recovery: with each low-level executable action, a very quick recovery strategy can be specified. If low-level execution is not recoverable, a more high-level deliberate recovery is launched to cope with the unexpected situation.

2 Architecture Overview

The plan executor and high-level planning were added as a deliberation layer on the top of the ThinkingCap behavior-based control architecture [14]. Figure 1 gives an overview of the overall architecture and how both layers are integrated. In the following subsections we briefly outline the different entities forming the complete system.

2.1 Behavior-based Architecture

The ThinkingCap (TC) robot-control architecture is composed of a fuzzy-logic controller and a navigational planner called B-Planner. TC controls the mobile robot using fuzzy behaviors expressed as sets of control rules. To generate navigation plans, B-Planner (for Behavior Planner) computes a set of context-behavior rules having the form `IF context THEN behavior`, where `context` is a formula of fuzzy predicates evaluated on the current world model. The context-behavior rules of a B-Plan are evaluated in parallel, influencing the overall robot behavior according to the blended value of their respective context.

TC uses a local perceptual space LPS to store information about the world around the robot expressed as object descriptors and perceptual data. The LPS provides data to the self-localization module used to compute the robot position in the different sectors of the map. The controller produces crisp control values (steering and velocity) through defuzzification of the result of the blended active fuzzy behaviors (according to their context calculated from the LPS).

2.2 High-level Planning

The kind of symbolic planning used within the overall system solves problems in partially observable domains i.e. observations reveal only part of the real state. Both planners PTLPLAN [9] and C-SHOP [4] that we used in our experiments use the same formalism for actions and world model. The action model makes it possible to reason about conditional non-deterministic effects with probability distribution over them. The description of effects might also include making observations. To represent uncertainty about the state of the world belief states are used. Both Planners take an initial belief state and a goal formula and return a plan with a certain probability of success.

2.3 Anchoring and Perception

The anchoring module provides an interface to perceptual information from the sensor systems of the mobile robot. It contains a number of functionalities for establishing the connection between the high-level symbolic representation and low-level perceptual representations such as video camera images. Typically, executing an action such as "(move-near gasbottle1)" requires the identification of what perceptual data correspond to the symbol "gasbottle1". On the symbolic level "gasbottle1" should have a symbolic description such as (shape gasbottle1 = bottle), (color gasbottle1 = red), which is matched against the available perceptual data. If a matching percept is found, the symbol is anchored to that percept. It sometimes happens that the anchoring module fails to find a specific object, either because no match is found, or there are several but partial matches. This is one important class of situations where recovery is needed [5]. The anchoring module can also provide information about already perceived objects, such as position and visual features extracted from the LPS.

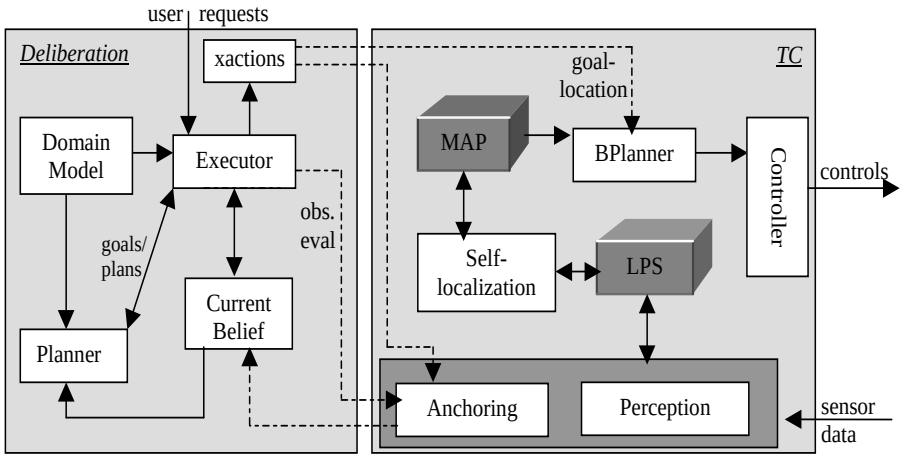


Figure 1: Global architecture

3 Shared Representation

Making the planner, the plan executor, and TC work together implies representing the different entities at the borderline between the different layers with a notation that all of them understand. More specifically is how to represent the robot's belief about the world, syntax and semantics of plans, the process of making observations and how plan actions should be executed by TC.

3.1 Belief States

The high-level plan executor is designed to execute and monitor plans that solve partially observable problems. Belief states are used to model partial observability about the state of the environment. The representation of belief states we use is similar to the one used by many symbolic planners such as C-SHOP [4], PTLPLAN [9], and C-BURIDAN [6], where a belief state is a probability distribution over elementary states that share the same set of observation fluents. For instance, the following belief state represents that, after making the observation of noticing a red light inside a room, the robot is either in room r_1 with 70% or in room r_2 with 30%.

$$\texttt{belief} = \langle \texttt{obs} = \{redlight\}; \{0.3 : \texttt{(robot-in } r_2\texttt{)}; 0.7 : \texttt{(robot-in } r_1\texttt{)}\}\rangle$$

To be able to evaluate observation fluents at execution-time, a procedure must be defined and associated with each observation fluent. When executing a plan, the executor uses the observation fluent to determine the evaluation procedure associated with it. The procedure specifies how to make the observation by calling TC's fuzzy observation predicates such as `(open d)` that refers to the degree to which the door `d` is open. The procedure might also use the data stored in the LPS to evaluate observation fluents not computed by TC such as `(near obj1 obj2)` for evaluating whether `obj1` is near `obj2` based on metric data in the LPS about both objects.

3.2 Conditional Plans

The symbolic planning system is required to generate plans with following syntax that the executor adopts for plan representation:

plan ::= `(` *action** *end-step* `)`
end-step ::= `:success` | `:fail` | `(cond` *branch** `)`
branch ::= `(`*obs-cond* *action** *end-step* `)`
action ::= `(`*action-name* *term**`)`
obs-cond ::= *fluent-literal* | `(and` *fluent-literal**`)`

action represents an instantiated domain action, *obs-cond* is a conjunction of fluent-value formulae defined over observations, and `:success`, `:fail` are used to denote predicted plan success and failure respectively. This grammar accepts plans that have the structure of a tree where nodes with one successor represent actions, and nodes with more than successor represent a conditional branching. The following plan is an example of plans accepted by the executor:

```
((go-near d1)(check d1)(cond (open d1) :  (enter r1) :success)
                             (not (open d1)) :  :fail))
```

Besides the syntax, the executor and the planner must agree on the semantics of the conditional plans. In our case, the semantics of a conditional plan can be interpreted as applying the first action of the plan in the initial belief state. The second action is applied in the resulting belief state of the first action, and repeatedly applying an action in the resulting belief state of its preceding action. If the application of an action results in more than one belief state, then the subsequent action must be a conditional plan `(cond` $(c_1\ p_1) \ldots (c_m\ p_m)$ `)` with as many branches as resulting belief states. Each branch $(c_i\ p_i)$ represents a contingency plan p_i to be executed in the belief state whose observations satisfy the branch condition c_i. As mentioned in the introduction, since belief states at a certain execution time are uniquely identified by their observations, execution-time belief update is no longer required. Only the the observations fluents are evaluated and the branch that has its observation fluent formula verified in the real world is selected for subsequent execution. Of course at execution time, there must be only one belief state whose observations are verified in the real world.

3.3 Executable Actions

To be able to execute the actions of a high-level plan, the domain creator must specify with each high-level action the different executable actions `xactions` in terms of the robot control-architecture (in this case TC) functionalities. Typically, an executable action defines a procedure that calls TC's functions to produce behaviors that would achieve a specific low-level goal. The procedure also defines the monitoring process to be associated with the execution of the behaviors in order to make sure to respond to unexpected events and apply local recovery strategies if possible. The monitoring process must also give an indication of whether the `xaction` has been executed successfully or with failure, so that a deliberate recovery would be considered for the high-level action.

Example 1 In order to execute the high-level action `(enter` r_1`)` to enter room r_1, the execution part may consist of a procedure "`execute-enter (room)`" that 1) calls the B-Planner with the goal `(robot-in` r_1`)`, where the goal represents a fuzzy predicate, and 2) installs a monitor process for the generated B-Plan. In case of failure to achieve the b-plan goal, the monitor may call the B-Planner to replan for another navigation path to enter room r_1.

4 High-Level Execution

In this section we outline the main processes involved in executing a high-level conditional plan and its actions. The executor uses different data structures to manage the execution of multiple plans that can arrive asynchronously. After having generated a plan, the planner creates an execution context that includes the initial belief state, the goal, the plan itself, the last action executed, and the priority of the plan. The execution context is then placed in one of three queues waiting for execution. The queues are associated with classes of plans identified by their priorities. A plan can have either low, medium, or high priority.

Plan execution is performed hierarchically. At the top-level, there is a process responsible for selecting the plan with the highest priority for execution. It is also responsible of launching the recovery of plans when one of their actions fails to execute. At the second level, a more specialized process is used to control the execution of the actions of conditional plans, reporting the outcome of the action to the high-level process. The action execution process is mainly responsible of extracting the executable actions of the current plan action considered for execution, and launching the appropriate processes to achieve the executable actions and monitor their progress, see Fig 2.

4.1 Plan Execution Process

The plan-execution process is launched upon starting up the robot. While in state INIT, the process checks periodically for waiting plans, proceeding with the execution of the plan with the highest priority. The actual execution of a plan starts in state NEXT-ACTION, where the executor checks the type of the current action selected for execution. An action can be `:success`, `:fail`, a conditional plan, or a domain action. It is also in this state where plans with higher priority can interrupt the execution of the plan in execution. As mentioned earlier, the two special actions `:success` and `:fail` reflect predicted success and failure by the planner of the plan, respectively. If the plan reaches a predicted failure, then the process simply drops the plan. Reaching a predicted success state means that the plan has achieved its goals with success. Because the currently dropped or succeeded plan might have interrupted the execution of another plan with lower priority, the execution process checks subsequently whether there is an interrupted plan waiting for execution in order to restore its execution context and start it again (state RESUME).

One issue in restarting the execution of an interrupted plan that might arise is the possibility not being able to start the execution of the interrupted plan, because its action to resume is not applicable in the the current real world situation resulting from the execution of an interrupting plan. To remedy partly to this problem, the process does not interrupt an executing plan unless it can find a chaining plan that ensures that the interrupted plan can be resumed when the interrupting plan finishes execution with success. It is worth noting that finding a chaining plan might be problematic since the interrupting plan can have more than one branch that leads to success. Generating the chaining plan would take into account this issue which results in a chaining plan with a possible branch for each possible belief state where the goals of the interrupted plan are satisfied. Upon resuming the execution of an interrupted plan, the process executes the chaining plan first, and then the rest of the interrupted plan. Obviously, this works only when the interrupting plan has successfully been executed, in case of failure, the process has to launch the planner to find a chaining plan that reaches the preconditions of the first action of the rest of the interrupted plan starting from the current situation.

If the current action is a conditional plan, the process checks the contingency condition for every branch in the real world, and chooses the branch whose condition is verified in the real world observations. Checking the branching conditions is done through calls to the procedures associated with the observation fluents that evaluate the condition fluents using the perceptual data provided by the anchoring module. If, on the other hand, the current action is an instantiated domain action, the process checks its preconditions in the current belief state. In case the preconditions are satisfied, another process is launched to execute the action as described in the next subsection. In case of discrepancy, the process calls special functions that build the current belief state so that more information is included about the current situation and then calling the planner to find a plan that achieves the preconditions of the failing action (state RECOVERY).

4.2 Action Execution Process

The execution of high-level actions is performed by a more specialized process whose states are outlined in Fig 2. Activating action execution at this level involves blocking the launching process i.e. the plan-execution process. High-level action execution starts by retrieving the `xactions` one at a time (state `xaction`). As we outlined before, there are specialized procedures defined with each `xaction` specifying the necessary steps to perform along with a monitoring process. Calling the specialized procedure of an `xaction`, results in blocking the action-execution process and launching the monitoring process of the `xaction`.

The monitoring process of an `xaction` can respond to failures by calling precomputed procedures or by calling the B-Planner to find another local B-plan. The implementation of an `xaction` monitoring process has also to to guarantee that the blocked action-execution process is notified about the outcome of the execution of the `xaction`. If the execution of the `xaction` is successful, then the action-execution process is awaken in the state XOK, otherwise it is awaken in the state XFAIL.

Awaking the action-execution process in state XFAIL is an indication of the inability of the robot to execute the `xaction` with success which leads to the failure of the high-level action. Therefore the plan-execution process is notified in turn that the execution of the current action has failed (state DISCREPANCY/FAIL). If, on the other hand, the monitor of the `xaction` reports to have executed `xaction`

successfully, the action-execution process repeats the same steps with the remaining `xactions`. When all the `xactions` have been successfully executed (state SUCCESS), the action-execution process awakes the plan-execution process in the state NEXT-ACTION, so that the same steps can be performed with the next high-level action of the plan.

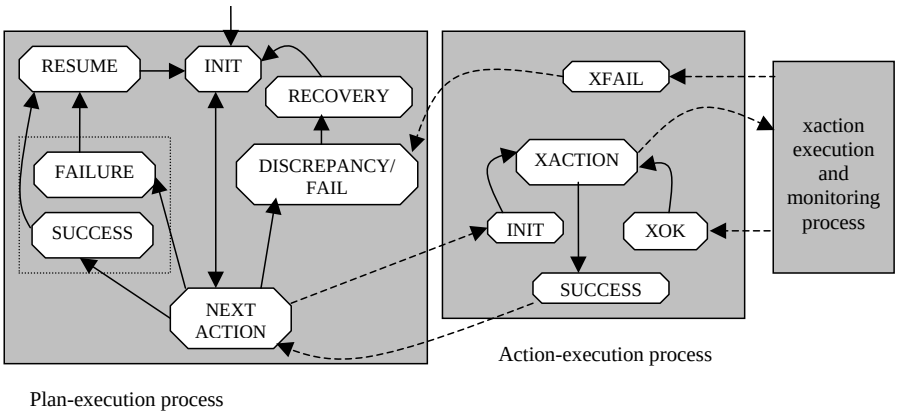


Figure 2: The different processes of the plan executor

5 Experiments

The system reported in this paper has been used on two Magellan Pro mobile robots for a number of experiments, in particular relating to the problems of anchoring with ambiguities, and the use of an electronic nose on a mobile robot. In the following, we briefly present some of those experiments. Although they were designed to test other specific capabilities of our system, they also serve well to illustrate what the executor can achieve.

The visual anchoring experiments, some of which were reported in [5], consisted of recovery planning in situations where the robot was supposed to find and approach an object but could not immediately determine which of several objects was the correct one. One series of experiments involved a number of gas bottles. The robot had the task to approach the marked gas bottle but the mark was not visible from the robot's current position. This resulted in a recovery plan where the robot inspected the different gas bottles from different angles and eventually detected the correct one. The success rates for these experiments were 87% of 45 runs, with setups of 2, 3 or 4 gas bottles. The maximal time required for the recovery planning was below 1.5 s for these experiments. More recent experiments involved using relational information to find objects (" the green can near to the red ball with a mark") with similar levels of success (80–93%) and performing recoveries from a sequence of problems occurring when the robot was to approach several different objects located in different corridors.

A series of e-nose experiments [11] involved the e-nose as a complementary sensor modality. The experiments were performed by using a number of cups containing different substances and in order to find the correct cup, the robot first needed to visually search for candidate objects and then use the electronic nose to discriminate between these candidates (success rates 82 – 76% for 2–5 candidates). In another series of experiments [10] the robot patrolled a number of corridors, traveling in total more than 1.2 km, while maintaining and updating information about cans it encountered. Cans were sometimes added, removed or displaced. In addition, the robot was occasionally given odour samples and had to find cans with similar odours.

6 Conclusion

We have presented a hierarchical system to execute probabilistic conditional plans that solve problems in partially observable domains. One major advantage of using hierarchy of processes is the ability to

reason about failure and recovery at different levels of detail and complexity. We also demonstrated how we integrated the proposed execution system on top of a behavior-based control architecture. Among the issues that were encountered in the process of integration, was the common representation that should be used to interface the executor and the planning system on one hand, and the executor and the control architecture on the other hand. We tackled this issue by adopting a simple set of rules that do not require too much effort from the planning domain writer.

References

[1] D. Aberdeen. A (revised) survey of approximate methods for solving partially observable markov decision processes. Technical report, National ICT Australia, Canberra, Austalia, 2003.

[2] P. Bertoli, A. Cimatti, M. Roveri, and P. Traverso. Planning in non-deterministic domains under partial observability via symbolic model checking. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence. (IJCAI)*, pages 473–478, 2001.

[3] J. Blythe. An overview of planning under uncertainty. *AI Magazine*, 20.

[4] A. Bouguerra and L. Karlsson. Hierarchical task planning under uncertainty. In *3rd Italian Workshop on Planning and Scheduling (AI*IA 2004)*. Perugia, Italy, 2004.

[5] M. Broxvall, L. Karlsson, and A. Saffiotti. Have another look: On failures and recovery planning in perceptual anchoring. In *Proceedings of the 4th International Cognitive Robotics Workshop (CogRob-2004)*, 2004.

[6] D. Draper, S. Hanks, and D. Weld. Probabilistic planning with information gathering and contingent execution. In *Proceedings of the 2nd International Conference on Artificial Intelligence Planning Systems (AIPS)*, pages 31–63, 1994.

[7] E. Hansen. Solving pomdps by searching in policy space. In *Proceedings of the 14th Annual Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 211–219, San Francisco, CA, 1998. Morgan Kaufmann Publishers.

[8] L. Kaelbling, M. Littman, and A. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

[9] L. Karlsson. Conditional progressive planning under uncertainty. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 431–438, 2001.

[10] A. Loutfi and S. Coradeschi. Maintaining coherent perceptual information using anchoring. In *Nineteenth International Joint Conference on Artificial Intelligence (IJCAI05, to appear)*, 2005.

[11] A. Loutfi, S. Coradeschi, L. Karlsson, and M. Broxvall. Putting olfaction into action: Using an electronic nose on an multi-sensing mobile robot. In *Proceedings of IEEE/RSJ Int. Conference on Intelligent Robots and Systems, IROS04*, 2004.

[12] N. Meuleau, K. E. Kim, L. Kaelbling, and A. Cassandra. Solving pomdps by searching the space of finite policies. In *Proceedings of the 15th Annual Conference on Uncertainty in Artificial Intelligence (UAI-99)*, pages 417–426, San Francisco, CA, 1999. Morgan Kaufmann Publishers.

[13] J. Pineau, N. Roy, and S. Thrun. A hierarchical approach to pomdp planning and execution. In *Workshop on Hierarchy and Memory in Reinforcement Learning (ICML)*, June 2001.

[14] A. Saffiotti. *Autonomous Robot Navigation: a fuzzy logic approach*. PhD thesis, Faculté de Sciences Appliqueés, Université Libre de Bruxelles, 1998.

[15] V. Verma, J. Fernandez, and R. Simmons. Probabilistic models for monitoring and fault diagnosis. In R. Chatila, editor, *The Second IARP and IEEE/RAS Joint Workshop on Technical Challenges for Dependable Robots in Human Environments.* October 2002.

Planning in Continuous State Spaces with Parametric POMDPs

Alex Brooks, Alexei Makarenko, Stefan Williams, Hugh Durrant-Whyte

ARC Centre of Excellence for Autonomous Systems
Australian Centre for Field Robotics
The University of Sydney
Sydney, NSW 2006, Australia
{a.brooks, a.makarenko, s.williams, hugh}@cas.edu.au

Abstract

This paper presents a novel approach to efficiently applying the POMDP formulation to problems with continuous states, typical in robotics. Continuous distributions over state-space are stored in parametric form, using a vector of sufficient statistics. The value function over continuous sufficient-statistic space is represented using a grid-based approximation, and Dynamic Programming is used to estimate this value function. The algorithm is applied to a simulated robot navigation problem.

1 Introduction

Traditional MDP-based approaches to planning are well-studied and effective in solving problems where perfect knowledge of the agent's state is available. Unfortunately, they are less effective in problems where the agent's state is uncertain, a condition which prevails in many real-world problems. Partially-Observable Markov Decision Processes (POMDPs) are promising as a method for dealing with this uncertainty. Instead of considering the agent's state to be perfectly known, POMDPs operate on the agent's belief, which is a probabilistic distribution over possible states of the world. The POMDP is essentially transformed into an MDP, using the belief distribution as the known state.

POMDPs are often cited as potential solutions for robot navigation problems, with the motivation that localization is imperfect and MDP-based approaches do not account for this uncertainty. The POMDP solution explicitly models the robot's position uncertainty, making decisions based on the probabilistic distribution over pose space. This naturally imparts the useful property that the robot will trade off actions that move the robot towards its goal with actions that reduce the robot's uncertainty, in a principled way.

POMDP solution methods can be divided into two approaches: policy-based methods and value-based methods [1]. This paper deals with the latter. The majority of mainstream value-based POMDP research has focussed on the discrete case, requiring that the state space be a finite set of cells. Robot navigation, however, is a fundamentally continuous problem that is poorly represented in the discrete domain unless the discritization is sufficiently fine. POMDP solution methods have problems with fine discretizations because the computational complexity increases rapidly with the size of the state space.

This paper presents an approach to solving robot-navigation POMDP problems in a continuous state-space. By constraining distributions over state space to a parametric family, points in the infinite-dimensional continuous belief space can be represented by finite vectors of sufficient statistics. For an appropriate choice of parametric family and MDP model, the parametric form of beliefs is closed under the transition and observation functions. Under these conditions value iteration can be performed efficiently, directly in the space of sufficient statistics. Since the value function is not likely to be piecewise-linear and convex (PWLC) in sufficient-statistic space, a grid-based approximation is used to solve the POMDP.

Section 2 discusses related approaches, and Section 3 formulates the DP equations on which the POMDP solution is founded. Section 4 provides a brief overview of POMDP representations based on

a discretization of the state-space and Section 5 describes the equivalent continuous representations. Section 6 describes a solution using this representation, Section 7 applies this solution to a robot navigation problem and Section 8 concludes.

2 Related Work

To the authors' knowledge there have only been three cases where POMDPs have been solved using a value-based approach in a continuous state space. In the case of linear systems with quadratic cost, a closed-form solution is available [2]. Unfortunately systems that are interesting from a planning point of view are generally not linear-quadratic.

Thrun uses a set of particles to represent beliefs, and represents the value function using a set of stored belief points [10]. Looking up the value of a belief that has not been stored involves applying k-nearest-neighbours, using KL divergence to measure the similarity between beliefs. The approach proposed in this paper is to represent beliefs more compactly, using an appropriate parametric form.

More recently, Porta et al extend the Perseus algorithm to continuous-state POMDPs [9][6]. The idea of representing a PWLC value function as the supremum of a set of hyperplanes is generalized, for continuous domains, to the supremum of a set of mixtures of gaussians (MoGs). Modelling beliefs and reward and observation functions with MoGs allows closed-form solutions to the integrations required for value iteration. To prevent an explosion in the number of gaussian components required to represent the value and belief functions, the number of components in each function is kept constant by approximating the function, at each iteration, with a MoG of fewer components. In comparision to grid-based solutions this approach has the advantage that values calculated at each belief point will generalize better over the belief space, at a cost of higher computational requirements per update. The combined effect for real-world problems is yet to be determined.

The algorithm presented in this paper bears similarities with the algorithms proposed by Roy [7]. Roy uses a high-dimensional discrete representation of the state space, but estimates a value function (using a grid-based approximation) over a lower-dimensional space. The first algorithm, AMDP, posits a set of 'belief features' such as maximum likelihood and entropy of the belief. The second applies non-linear dimensionality reduction (using E-PCA) to project beliefs to the low-dimensional space. To evaluate the low-dimensional transition function at each belief point, both algorithms require a mapping from the low-dimensional to high-dimensional space and an evaluation of the transition and observation functions in the high-dimensional discrete space, followed by a mapping back to the low-dimensional space.

In comparison with the algorithm presented here, Roy's can model arbitrary distributions (or rather compressed versions of discrete approximations thereof). This is an advantage when used in conjunction with a state estimator capable of producing arbitrary distributions, such as a particle filter. We expect our algorithm to be particularly effective when used in conjunction with an estimator that produces precisely the same parametric family of distributions which was used for planning. In addition, since our algorithm does not require a transition function in a discrete space, the transition function used for planning can exactly match the transition function used for online state estimation.

3 Formulation of the POMDP

The aim in a POMDP problem is to calculate a policy that optimizes a discounted set of future rewards in a stationary, partially-observable environment with known dynamics. This section begins by establishing some terminology.

At each discrete time interval k, a POMDP agent is in an unknown state $\mathbf{x}_k$. The agent chooses an action u_k, receives a reward $r(\mathbf{x}_k, u_k)$, and arrives in state $\mathbf{x}_{k+1}$. It then receives an observation z_{k+1} from the new state.

It is useful to define an information vector of all information up to time k as

$$I_k = (z_0, z_1, ..., z_k, u_0, u_1, ..., u_{k-1}), \qquad I_0 = z_0, \tag{1}$$

containing all the observations up to and including time k, but only the control decisions up to time $k-1$. Therefore I_k is the information available after making an observation but before taking an action. Since

the agent cannot observe the state directly it maintains a probability distribution over the state-space given all available information, denoted $b_k = p(\mathbf{x}_k|I_k)$, where b_k is known as the belief. Due to the Markov assumption it is a sufficient statistic for the entire history, so

$$b_{k+1} = p(\mathbf{x}_{k+1}|I_k, u_k, z_{k+1}) = p(\mathbf{x}_{k+1}|b_k, u_k, z_{k+1}) = \Phi(b_k, u_k, z_{k+1}), \tag{2}$$

where Φ is the belief transition function. Note that while state transitions are stochastic, Φ is a *deterministic* function of action, observation and prior belief. By maintaining a consistent belief, the POMDP over unknown states is transformed into an MDP over known beliefs.

In order to maintain a consistent belief, the agent must know the following *a priori*:

1. the initial belief: $b_0 = p(\mathbf{x}_0|I_0)$;
2. the probability distribution governing state transitions: $p(\mathbf{x}_{k+1}|\mathbf{x}_k, u_k)$; and
3. the observation likelihood function: $p(z_{k+1}|\mathbf{x}_{k+1})$.

Note that $p(\mathbf{x}_{k+1}|\mathbf{x}_k, u_k)$ is not a belief; it is a distribution over state at time $k+1$ *given perfect state knowledge* from time k.

Over an episode, the agent executes a policy $u_k = \pi(I_k)$. Since the probability distribution over state is a sufficient statistic for the history, the policy is a function of that belief, thus $u_k = \pi(b_k)$. The value of executing a policy π starting from a belief b_k is equal to the discounted sum of expected future rewards:

$$V_\pi(b_k) = \sum_{j=0}^{\infty} E[\gamma^j r(\mathbf{x}_{k+j}, u_{k+j})] \tag{3}$$

where γ is a discount factor ≤ 1. The aim of the POMDP agent is to find the optimal policy

$$\pi^*(b_k) = \arg\max_\pi V_\pi(b_k). \tag{4}$$

A standard approach to finding this optimal policy is to use Dynamic Programming [2]. The value function at time k is defined recursively in terms of the value function at time $k+1$,

$$V_{\pi^*}(b_k) = \max_{u_k} \left[\underset{\mathbf{x}_k}{E}\big[r(\mathbf{x}_k, u_k)\big] + \gamma \underset{z_{k+1}}{E}\big[V_{\pi^*}(\Phi(b_k, u_k, z_{k+1}))\big] \right]. \tag{5}$$

Two issues need to be resolved before Equation 5 can be implemented. Firstly one needs a representation for probability distributions over state-space. Secondly one needs a representation for V, which is a continuous function of probability distributions (not states). To see why the latter issue is problematic, consider a two-dimensional continuous state-space: $V(b_k)$ needs to assign a cost to every possible surface (that integrates to one) over that two-dimensional space. There is an uncountable number of possible surfaces: certainly too many to represent with a digital computer. The standard solution to this problem is to discretize the state, action and observation spaces.

4 Representing Value over Discrete States

When the state-space is discretized into a set of cells S a belief can be represented by a vector $b \in \Re^{|S|}$, where $|S|$ is the number of cells into which the state space is divided and $b(s)$ represents the agent's belief that it is in state s. Approaches to representing the value function fall into one of two categories.

4.1 Grid-Based Representations

Grid-based approaches define a grid G containing a finite set of belief states $\{b_1^G ... b_{|G|}^G\}$, $b_n^G \in \Re^{|S|}$. An interpolation-extrapolation function estimates the value at any point in belief space using the values at the grid points. The main problem with grid-based representations is that the number of required points increases rapidly with both the size of the state space and the resolution of the grid. Regular [4], variable-resolution [12] and arbitrary [3] grids have been proposed.

4.2 Gradient-Based Representations

Sondik shows that in the case where the state, action and observation spaces are all discrete, the value function is piecewise-linear and concave (PWLC) [8]. In this case V can be represented by the supremum of a finite set of hyperplanes over the belief space, each defined by a vector $\alpha \in \Re^{|S|}$.

Exact solution methods manipulate these α vectors directly. The computational cost is usually dominated by the *curse of history*: in the worst case the number of hyperplanes after the i'th iteration is $O(|A|^{|O|^{i-1}})$, where $|A|$ and $|O|$ denote the number of discrete actions and observations, respectively [5] [3]. Various techniques have been proposed to prune the redundant hyperplanes, however the pruning steps are usually expensive and effect the constant factors rather than the order of the growth [5].

Point-based approximations perform updates by finding hyperplanes that maximize the value function at a finite set of belief points B, where $|B|$ is either fixed [9] or increasing [5]. Point-based approximations claim to generalize better than grid-based approximations because they calculate both the value *and the value gradient* at each belief point.

5 Representing Value over Continuous States

It is possible to represent the value function over belief space if there is a finite set of scalars representing each belief point. The value function is then a function of those scalars. Section 4 showed how this is possible for discrete state spaces, this section examines the possibilities for continuous state spaces.

A requirement for the use of a parametric representation for the distribution b_k is that the belief transition function $\Phi(b_k, u_k, z_{k+1})$ maintains the parametric form. This paper focusses on the use of a Gaussian representation. While it is not the only choice, it has the advantage that there are well-known process models and observation functions under which the parametric form of the belief distribution is closed. The result of using a parametric representation is that the value function can be represented by a function over the sufficient statistics of the distribution.

Consider a one-dimensional toy POMDP problem. The traditional approach is to partition the space into $|S|$ cells. One must then evaluate the value function over the $|S|$-dimensional continuous space of distributions over those cells. This becomes expensive for large $|S|$. Instead, one could represent the distribution as a Gaussian with parameters (μ, σ). The problem is reduced to computing a value function over a two-dimensional space.

This reduction in dimensionality introduces several potential problems. From an implementation standpoint, the value function over the sufficient statistics is not likely to be PWLC, and thus cannot be represented by a set of hyperplanes. Therefore evaluating the expectations in Equation 5 will involve integrating over the state space, and brute-force numerical integration becomes exponentially more expensive as the dimensionality of the state-space increases. In addition, the gradient of the value function is not available.

A more fundamental issue is that the ability to represent arbitrary distributions is lost. This is problematic only when one is unable to represent distributions that are likely to occur in practice. As Roy points out, the space of distributions commonly encountered in robot navigation problems is extremely constrained [7]. Gaussians have certainly proven to be a sufficiently useful representation in many robot mapping and localization problems [11]. They enforce the fact that nearby regions of the state space are related to one another, which arbitrary distributions over a discretized state space do not. Constraining oneself to precisely those distributions which are likely to occur offers obvious potential advantages.

If a single Gaussian is inadequate, more complex functions can be approximated arbitrarily accurately by a mixture of Gaussians, the only problem being that the dimensionality of the sufficient statistics increases linearly with the number of Gaussians.

6 Solving Parametric POMDPs

Given a parametric form, the continuous distribution over state-space b_k can be written in terms of a vector of sufficient statistics $\mathbf{\Theta}_{k|I_k}$. Maintaining a consistent belief in this parametric form requires an initial belief state, $\mathbf{\Theta}_{0|I_0}$, plus an update function:

$$\mathbf{\Theta}_{k+1|I_{k+1}} = \bar{\Phi}(\mathbf{\Theta}_{k|I_k}, u_k, z_{k+1}) \tag{6}$$

The instantaneous reward function and the observation likelihood function are also functions of the sufficient statistics, defined in terms of integrals over the state space:

$$\bar{r}(\mathbf{\Theta}_{k|I_k}, u_k) = \underset{\mathbf{x}_k}{E}\big[r(\mathbf{x}_k|\mathbf{\Theta}_{k|I_k}, u_k)\big] = \int r(\mathbf{x}_k, u_k) p(\mathbf{x}_k|\mathbf{\Theta}_{k|I_k}) d\mathbf{x}_k \tag{7}$$

$$p(z_{k+1}|\mathbf{\Theta}_{k|I_k}, u_k) = \int p(z_{k+1}|\mathbf{x}_{k+1}) p(\mathbf{x}_{k+1}|\mathbf{\Theta}_{k|I_k}, u_k) d\mathbf{x}_{k+1} \tag{8}$$

Given an appropriate belief update function $\bar{\Phi}$, the DP equation can be written, analogous to Equation 5, in terms of sufficient statistics,

$$\bar{V}_{\pi^*}(\mathbf{\Theta}_{k|I_k}) = \max_{u_k}\left[\bar{r}(\mathbf{\Theta}_{k|I_k}, u_k) + \gamma \underset{z_{k+1}}{E}\left[\bar{V}_{\pi^*}(\bar{\Phi}(\mathbf{\Theta}_{k|I_k}, u_k, z_{k+1}))\right]\right]. \tag{9}$$

Since the value function is not PWLC (and thus no gradient information is available) a grid-based approximation is used, as in Section 4.1. The parametric grid-based solution is not expected to present the same scalability problems since the dimensionality of the state-space is much lower.

Let G be a set of belief points in sufficient-statistic space, $G = \{\mathbf{\Theta}_1^G, \ldots \mathbf{\Theta}_{|G|}^G\}$, and let Ψ be a set of point-value pairs, $\Psi^G = \{(\mathbf{\Theta}_1^G, \psi(\mathbf{\Theta}_1^G)), \ldots (\mathbf{\Theta}_{|G|}^G, \psi(\mathbf{\Theta}_{|G|}^G))\}$. Let $\hat{V}_k^G$ be the grid-based value function approximation at time k. Then the approximation at time $k+1$ is $\hat{V}_{k+1}^G(\mathbf{\Theta}) = R_G(\mathbf{\Theta}, \Psi_{k+1}^G)$, where R_G is an interpolation-extrapolation rule that estimates $\hat{V}^G$ at any point in sufficient-statistic space. The values associated with each grid point can therefore be updated using

$$\psi_k(\mathbf{\Theta}_j^G) = \max_{u_k}\left[\bar{r}(\mathbf{\Theta}_j^G, u_k) + \gamma \underset{z_{k+1}}{E}\left[\hat{V}_{k+1}^G(\bar{\Phi}(\mathbf{\Theta}_j^G, u_k, z_{k+1}))\right]\right] \tag{10}$$

For a more detailed discussion of grid-based approximations, the reader is directed to [3].

The entire parametric POMDP algorithm can now be outlined:

1 Select a set of belief points G
2 $k \leftarrow MAX\_ITERATIONS$
3 initialize $\psi_k(\mathbf{\Theta}_j^G)$ to zero, $\forall j \in |G|$
4 *while not* (**converged**)
5 $k \leftarrow k-1$
6 *foreach* $\mathbf{\Theta}^G \in G$
7 *foreach* $u_k \in U$
8 calculate $\rho_{u_k} \leftarrow \bar{r}(\mathbf{\Theta}^G, u_k)$
9 $v_{u_k} \leftarrow 0$
10 *for each* $z_{k+1} \in$ Z
11 calculate the likelihood $l \leftarrow p(z_{k+1}|\mathbf{\Theta}^G, u_k)$
12 calculate $\mathbf{\Theta}' \leftarrow \bar{\Phi}(\mathbf{\Theta}^G, u_k, z_{k+1})$
13 interpolate/extrapolate to find $v \leftarrow \hat{V}_{k+1}(\mathbf{\Theta}')$
14 $v_{u_k} \leftarrow v_{u_k} + lv$
15 *end foreach* z_{k+1}
15 *end foreach* u_k
16 $\psi_k(\mathbf{\Theta}^G) \leftarrow \max_{u_k}[\rho_{u_k} + \gamma v_{u_k}]$
17 *end foreach* $\mathbf{\Theta}^G$
18 *end while*

The convergence criterion at step 4 remains to be defined. An acceptable choice of criterion would be when the maximum change in value of a belief point in G drops below a threshold. It should also be noted that the algorithm iterates over a set of actions and observations. While this works well when they are defined as discrete sets, continuous actions and observations can also be dealt with by sampling from their distributions, as suggested elsewhere [10] [9].

For discrete actions and observations, the complexity of each iteration can be kept low by pre-calculating most of the steps. If the instantaneous rewards, observation likelihoods and belief transitions, calculated in steps 8, 11 and 12 respectively, were all pre-computed then each iteration could be performed efficiently. The complexity of the algorithm is $O(|G||A||O|C_{EVAL}(R_B, |G|)$, where $C_{EVAL}(R_G, |G|)$ is the complexity of applying interpolation-extrapolation rule R_B for $|G|$ belief points. In addition, Hauskrecht shows that under certain conditions the POMDP can be converted to a grid-based MDP such that the interpolation-extrapolation rule need not be calculated at every step [3].

The question of how to select the set of belief points G has not been addressed so far. If the state-space is discrete, the value function V is convex. For a convex interpolation rule R_G, the grid-based approximation to the value function, $\hat{V}$, is also convex. This guarantees that $\hat{V}$ is an upper bound on V [3]. In the case of a continuous state-space, no such guarantee can be made, so it is important to select an appropriate set of belief points, maintaining a balance between accuracy and computational complexity. In general, belief points should be concentrated in areas where the second derivative of the value function is high. These areas are where a linear approximation between belief points is worst and the optimal actions vary most. Belief points should also be concentrated in areas of sufficient-statistic space that are likely to occur in practice. While the shape of distributions has been constrained to those likely to occur, there are still unlikely belief points (such as being well-localized in areas devoid of landmarks). Methods have been proposed in the literature for adaptively improving grids [3], or finding likely beliefs by having a robot randomly explore the environment during a training phase [9].

7 Robot Navigation Example

As an example application, consider the familiar robot navigation problem to which POMDPs are often applied: a robot moves in a known world towards a goal point. The state-space is continuous and two-dimensional rather than discrete with one dimension per cell. The distribution over state-space is constrained to being a circular Gaussian, thus beliefs can be represented by the three-dimensional vector of sufficient statistics $\mathbf{\Theta}$ whose components are (μ_x, μ_y, σ).

The robot relies on odometry for localization and has to avoid a known set of hazards. In addition, there is a set of electric beacons on the floor which, should the robot run over them, will localize the robot extremely well. In the absence of beacons, localization uncertainty increases linearly with distance travelled. The robot's control actions are discrete, limited to movements of 0.2m along the four compass directions. Any movement shifts the mean of the robot's distribution, as well as increasing the standard deviation by 0.02m. If the robot attempts to shift its mean off the edge off the world, the mean does not change. The observations are also discrete: there is one possible observation per beacon, plus one observation representing 'no beacon observed'. The observation is a deterministic function of state: if the robot is over a beacon then it will observe it and become well-localized at that beacon. The probability of each observation is given by Equation 8. For observations of beacons, the integral is simple to evaluate since it is non-zero only at the beacons. The probability of the 'no beacon' observation can be calculated by taking the complement of the sum of the probabilities of the beacon observations. Together, the effects of actions and observations define the deterministic belief transition function, $\Phi(\mathbf{\Theta}_{k|I_k}, u_k, z_{k+1})$.

The instantaneous reward is a simple function of state alone: the reward is $+1$ at the goal, -1 at hazards, and zero otherwise. For a given expected position and uncertainty in position there is some probability that either of the above has occurred. Therefore a reward function can be constructed using Equation 8. This integral is not difficult to calculate because $r(\mathbf{x})$ is non-zero only at hazards and goals.

The set of belief points G have been chosen in a regular grid, spaced 0.2m apart in space and 0.02m apart in σ. This fairly naive placement leads to a large number of belief points but a trivial interpolation-extrapolation rule, since the dynamics of the world and the placement of beacons constrain $\mathbf{\Theta}_{k+1|I_{k+1}}$ to remaining on the grid if $\mathbf{\Theta}_{0|I_0}$ was on the grid. It would certainly be possible to reduce the number of belief points without affecting solution quality.

Figure 1 shows the rewards, the final value function and the corresponding best actions after convergence. The rewards show that the goal is only attractive when uncertainty is low. Note that beacons have no effect on the *instantaneous* reward. The actions demonstrate how the robot accounts for uncertainty when making decisions: the probability of an uncertain robot finding the goal without encountering a hazard is

low, so it heads toward the beacons. When the robot is well-localized it is capable of avoiding hazards, so it approaches the goal. The plots of the final value function in Figure 1 are on the same scale. It can be seen that the value surface corresponding to uncertain beliefs lies entirely below the surface corresponding to a small σ. Thus the agent always values certainty, regardless of the mean of a belief.

8 Conclusion

The contribution of this paper is a novel approach to efficiently applying the POMDP formulation to problems with continuous states, typical in robotics. Choosing a parametric form provides a compact representation and restricts the possible beliefs to a set that approximates those that are likely to occur in practice. An approximate algorithm was presented for solving POMDPs under this formulation. While no exact solution is available, it should be noted that discretization of the state space also involves an approximation. Finally, the algorithm was applied to a simple simulated robot navigation problem.

POMDP solutions are generally considered to be incapable of scaling to real-world problems. The limiting factors are the curses of history and dimensionality. Point-based and grid-based approximations avoid the curse of history, however the curse of dimensionality remains: the number of samples required to achieve a given sample density is exponential in the number of states. A parametric approach limits the dimensionality of the state space, however the number of grid points required still increases with the size of the map. A parametric approach is likely to be superior in domains where a fine discretization is required to approximate continuous values, and where the number of sufficient statistics is small. A more realistic robot navigation problem would involve modelling heading in addition to location and allowing arbitrary uncertainty ellipses rather than only circular Gaussians. This would involve nine sufficient statistics (three for the mean, six for the covariance). Whether this is feasible remains an open question.

Acknowledgement

This work is supported by the ARC Centre of Excellence programme, funded by the Australian Research Council (ARC) and the New South Wales State Government.

References

[1] D. Aberdeen and J. Baxter. scaling internal-state policy-gradient methods for pomdps. In *Proceedings of the International Conference on Machine Learning*, 2002.

[2] D. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, Belmont, Massachusetts, 2000.

[3] M. Hauskrecht. Value-function approximations for partially observable Markov decision processes. *Journal of Artificial Intelligence Research*, 13:33–94, 2000.

[4] W. Lovejoy. Computationally feasible bounds for partially observed Markov decision processes. *Operations Research*, 1991.

[5] J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *Int. Joint Conf. on Artificial Intelligence*, 2003.

[6] J. Porta, M. Spaan, and N. Vlassis. Robot planning in partially observable continuous domains. In *Robotics: Science and Systems*, 2005 (to appear).

[7] N. Roy. *Finding Approximate POMDP Solutions Through Belief Compression*. PhD thesis, MIT, 2003.

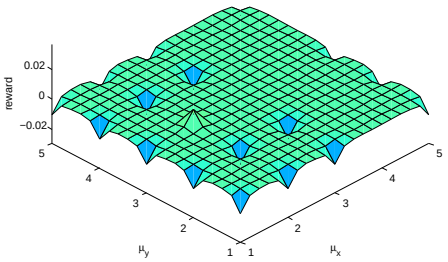
[8] E. J. Sondik. *The Optimal Control of Partially Observable Markov Processes*. PhD thesis, Stanford University, 1971.

[9] M. Spaan and N. Vlassis. Perseus: Randomized point-based value iteration for POMDPs. Technical report, Informatics Institute, U. of Amsterdam, 2004.

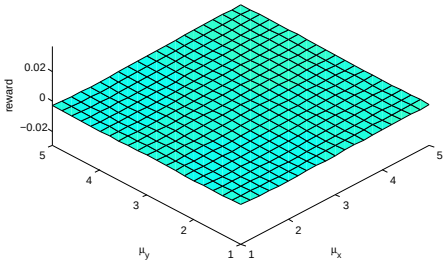
[10] S. Thrun. Monte carlo POMDPs. In S. Solla, T. Leen, and K.-R. Müller, editors, *NIPS 12*, pages 1064–1070. MIT Press, 2000.

[11] S. Williams, G. Dissanayake, and H. Durrant-Whyte. Field deployment of the simultaneous localisation and mapping algorithm. In *15th IFAC World Congress on Automatic Control*, 2002.

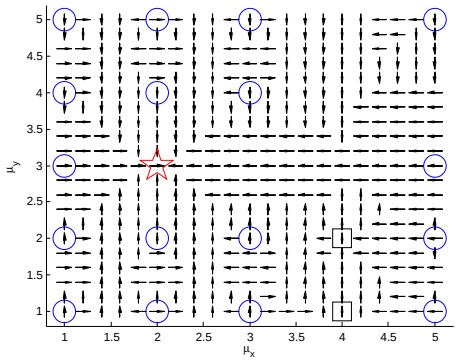
[12] R. Zhou and E. Hansen. An improved grid-based approximation algorithm for POMDPs. In *IJCAI*, pages 707–716, 2001.



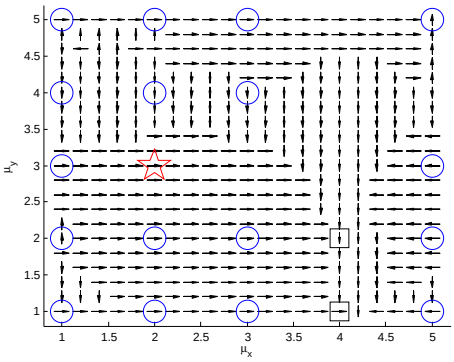
(a) Rewards, $\sigma = 0.1$



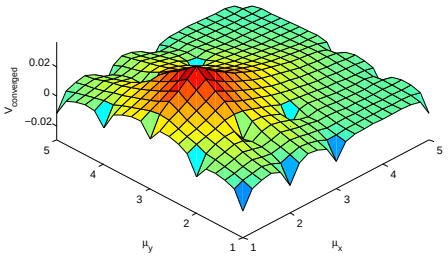
(a) Rewards, $\sigma = 1.0$



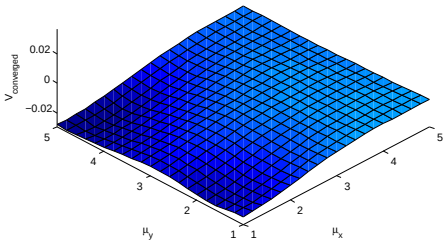
(a) Actions, $\sigma = 0.1$



(a) Actions, $\sigma = 1.0$



(a) Values, $\sigma = 0.1$



(a) Values, $\sigma = 1.0$

Figure 1: The (a-b) rewards, (c-d) actions and (e-f) values as functions of belief (not state), plotted for all μ_x and μ_y, for $\sigma = 0.1$ and $\sigma = 1.0$. The arrows in (c) and (d) indicate the direction the robot will move from a given belief. The known positions of hazards (circles), beacons (squares) and the goal (pentagram) are overlaid over the actions.

Navigation and planning in an unknown environment using vision and a cognitive map

Nicolas Cuperlier, Mathias Quoy, Philippe Gaussier, Christophe Giovanangelli

ETIS-UMR 8051
Université de Cergy-Pontoise - ENSEA
6, Avenue du Ponceau
95014 Cergy-Pontoise France
cuperlier@ensea.fr

Abstract

We present a navigation and planning system using vision for extracting unpredefined landmarks. The set of landmarks and their azimuth relative to the north given by a compass defines a particular location. The transitions between two locations are coded in a graph (our cognitive map) where the links are reinforced when the path is used. The transitions are linked with the integrated movement used for going from one place to the other. Planning corresponds to the diffusion of an activation in the graph from the location to reach to the possible transitions from the current location. The proposed transitions are merged in a neural field so that the direction taken is the stable solution of a dynamical system.

1 Introduction

Various SPLAM methods have proved to be effi cient in different indoor and outdoor environments. They rely on the combination of different algorithms that have to be triggered appropriately (and concurrently) when necessary. For instance, mapping may be linked with different sensors (laser, ultra-sound, visual feature recognition ...) that have to be chosen appropriately. The main drawback of this approach is the linking of ad hoc algorithms. We propose here a unifi ed neuronal framework based on an hippocampal and prefrontal model where vision, place recognition and planning are fully integrated. Assets of this model are:

- autonomous landmark extraction based on characteristic points (section 2)
- autonomous place building: there are no a priori predefi ned squares, or world model (section 3)
- autonomous cognitive map building associating transition between places with the integrated movement given by the robot's odometry (section 4)
- autonomous planning using the cognitive map (section 5)
- stable movement proposed given by the stable fi xed point solution of a dynamical system (section 6)

Drawbacks will be left for conclusion.

2 Autonomous landmark extraction based on characteristic points

Images are taken by a panoramic camera. Curvature points are extracted from the gradient image (at low resolution) by DOG fi ltering. These points are the center of a 32x32 pixels small image corresponding to a landmark. This image is binarized through a log-polar transform [10]. Each landmark is linked with its angular position relative to the north given by a compass [16, 11]. In a panoramic image, 30 (landmark, azimuth) pairs are extracted (see fi gure 1).

The log-polar transform gives some rotation and depth robustness.

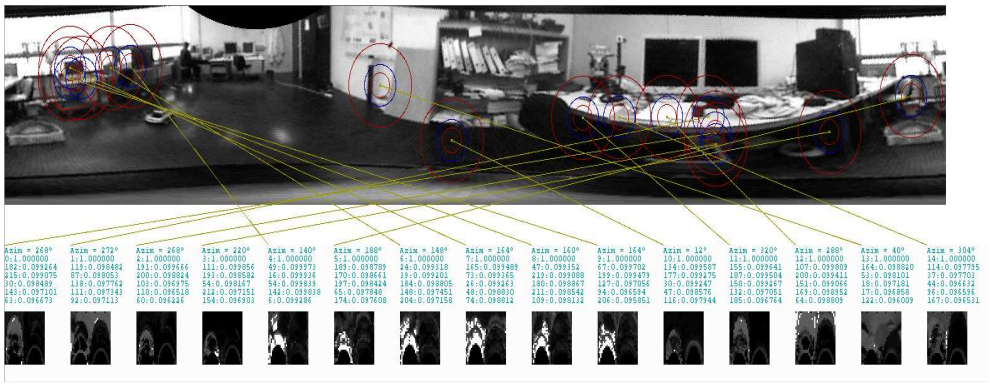


Figure 1: Image taken from a panoramic camera. Below are 15 examples of 32×32 log-polar transforms taken as landmarks and their corresponding position in the image.

3 Autonomous place building

Each set of (landmark, azimuth) pairs is learned and thus characterizes one location. The neuron coding for this location is called a "place cell" as the one found in the rat's hippocampus [11]. Our exact neural model will not be described here, but may be found in [9] (see fig. 2).
A matching function computes the distance between the learned sets and the current set. If the result of this function is below a *given* recognition threshold, then a new neuron is recruited for coding this new location. Hence, the density of location learned depends on the level of this threshold, but also on the position in the environment. Namely, more location are learned near walls or doors because there is a fast change in the angular position of near landmarks, or in the (dis)appearance of landmarks.
Two successively reached places are coded by a *transition cell* (see fig. 2). Each of these cells is linked with the direction used to go from the starting location to the ending location. For instance, going from place A to place B creates a transition cell AB linked with the direction (relative to the north) for going from A to B. This direction integrates all direction changes performed between A and the creation of B. It is reset when a new place cell is recruited.

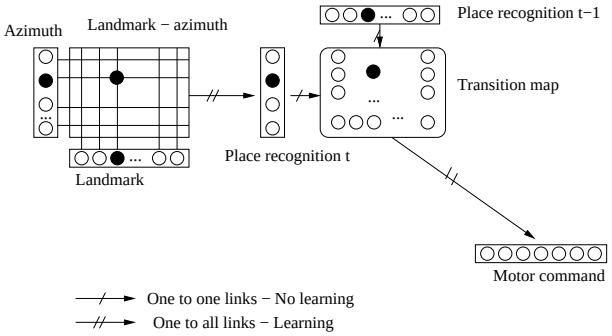


Figure 2: Sketch of the model. From left to the right: merging landmarks and their azimuth, then learning of the corresponding set on a place cell. Two successive place cells define a transition cell which is linked with the integrated movement performed. The cognitive map is not shown here.

4 Autonomous cognitive map building

Experiments carried out on rats have led to the definition of cognitive maps used for path planning [17]. Most of cognitive maps models are based on graphs showing how to go from one place to an other [2, 3]. They mainly differ in the way they use the map in order to find the shortest path, in the way they react to

dynamical environment changes, and in the way they achieve contradictory goal satisfactions. Other works use ruled-based algorithms, classical functional approach, that can exhibit the desired behaviors, we will not discuss them in this paper, but one can refer to [8].
Each time a transition is used, a link is created in the cognitive map. This link links the transition used with the previous transition. After some time, exploring the environment leads to the creation of the cognitive map (see fi g. 3). This map may be seen as a graph where each node is a transition and the arcs the fact that the path between these two transitions was used. We can give a value to each link. This value is increased if the link is used, and decreased if it is not. After some time is the environment, some links are reinforced. These links correspond to paths that are often used. In particular, this is the case when some particular locations have to be reached more often than others (see section 5) [12].

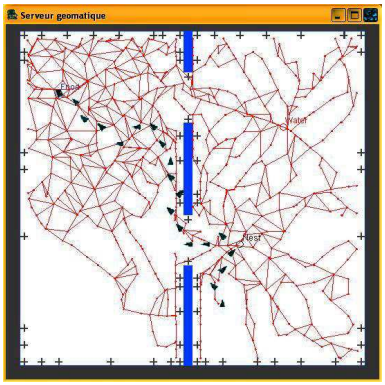


Figure 3: Cognitive map build by exploration of the environment. The triangles give the successive robot position starting from the right to the goal (on the left).

5 Autonomous planning using the cognitive map

Some places are more important because they may be some goals to reach when necessary. When a goal has to be reached, the transitions leading to it are activated. This activation is then diffused on the cognitive map graph, each node taking the maximal incoming value which is the product between the weight on the link and the activity of the node sending the link. After stabilization, this diffusion process gives the shortest path between all nodes and the goal nodes. This is a neural version of the Bellman-Ford algorithm [5, 14] (see fi g. 4).
When the robot is in a particular location A, all possible transitions beginning with A are possible. The top-down effect of the cognitive map is to bias the possible transitions such that the one chosen by the cognitive map has a higher value. A competition mechanism makes this transition win (however see section 6). So, the corresponding movement is triggered and thus the robot realizes the planned movement.

6 Movement and neural field dynamics

Obstacles are detected by 12 infra-red sensors. A reflex behavior is triggered by a Braitenberg-like architecture [6]. The direction given by this reflex behavior takes over the direction given by random exploration or by planning. Nevertheless, these obstacle avoidance movements are integrated in the computation of the overall direction used between two places.
After planning, different movements are proposed. One has a higher value. As seen in section 5, a simple competition mechanism selects the neuron with the higher value. However, this solution is not stable enough, in the sense that the direction to use be brutally change from one step to the other. There are also some deadlock cases [7].
The solution used for having a stable continuous direction to follow is to defi ne a dynamical system where the stable fi xed point solution is the direction to follow. This is achieved using a neural fi eld [1, 15, 13].

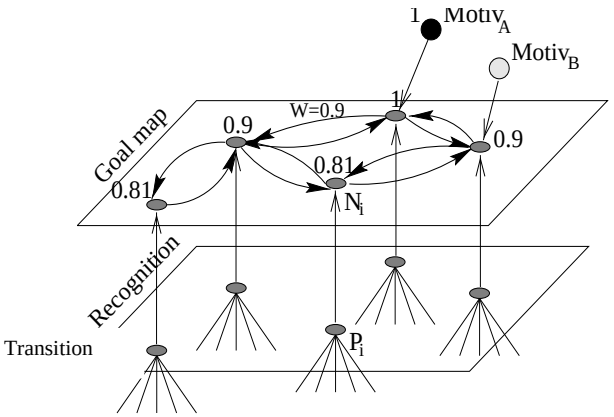


Figure 4: Diffusion of the activity n the graph corresponding to the cognitive map. Diffusion is starting from the goal. Each node keeps the maximal activity coming from its neighbors. The activity is the product between the value of the link and the activity of the sending node.

$$\tau.\frac{f(x,t)}{dt} = -f(x,t) + I(x,t) + h + \int_{z \in V_x} w(z).f(x-z,t)dz \quad (1)$$

Where $f(x,t)$ is the activity of neuron x, at time t. $I(x,t)$ is the input to the system. h is a negative constant. τ is the relaxation rate of the system. w is the interaction kernel in the neural fi eld activation. A difference of Gaussian (DOG) models these lateral interactions that can be excitatory or inhibitory. V_x is the lateral interaction interval that defi nes the neighborhood. Without inputs the constant h ensures the stability of the neural fi eld homogeneous pattern since $f(x,t) = h$. In the following, the x dimension will by an angle (direction to follow), 0 corresponding to go straight forward.

The properties of this equation allow the computation of attractors corresponding to fi xed points of the dynamics and to local maxima of the neural fi eld activity. Repellors may appear too, depending on the inputs. A stable direction to follow is reached when the system is on any of the attractors.

The angle of a candidate transition is used as input. The intensity of this input depends on the corresponding goal transition activity, but also on its origin place cell recognition activity. If only one transition is proposed, there will be only one input with an angle $x_{targ} = x^*$ and it erects only one attractor $x^* = x_{targ}$ on the neural fi eld. If x_c is the current orientation of the animat, the animat rotation speed will be $w = \dot{x} = F(x_c)$.

Fusion of several transition information depends on the distance between them. Indeed the neural fi eld equation allows cooperation for coherent inputs associated with spatially separated goals (for us different angles proposed). If the inputs are spatially close, the dynamics give rise to a single attractor corresponding to the average of them. Otherwise, if we progressively amplify the distance between inputs, a bifurcation point appears for a critical distance, and the previous attractor becomes a repellor and two new attractors emerge.

Oscillations between two possible directions are avoided by the hysteresis property of this input competition/cooperation mechanism. It is possible to adjust this distance to a correct value by calibrating the two elements responsible for this effect: spatial fi ltering is obtained by convoluting the dirac like signal coming from transition information with a Gaussian and taking it as the input to the system. This combined with the lateral interactions allows the fusion of distinct input as a same attractor. The larger the curve, the more fusion there will be.

7 Conclusion

Our model currently running on robots (Koala robots and Labo3 robots) has interesting properties in terms of autonomous behavior. However, this autonomy has some drawbacks:

- we are not able to build a cartesian map of the environment because all location learned are robot centered. However, the places in the cognitive map and the direction used give a squeletton of the environment.
- we have no information about the *exact* size of the rooms or corridors. Again, the cognitive map only gives a sketch of the environment.
- some parameters have to be set: the recognition threshold (section 3) and the diffusion size of the interaction kernel of the neural field (section 6). The first one determines the density of build places. The higher the threshold, the more places are created. The second parameter has to be tuned for each robot depending on its size and on the position of the infra-red sensors for obstacle avoidance. For instance, a too high diffusion value prevents the robot from going through the doors.

The transition used in this model may also be the elementary block of a sequence learning process. Thus, we are able to propose a unified vision of the spatial (navigation) and temporal (memory) functions of the hippocampus [4].

Acknowledgments This work is supported by two french ACI programs. The first one on the modeling of the interactions between hippocampus, prefontal cortex and basal ganglia in collaboration with B. Poucet (CRNC, Marseille) JP. Banquet (INSERM U483) and R. Chatila (LAAS, Toulouse). The second one (neurosciences intégratives et computationnelles) on the dynamics of biologically plausible neural networks in collaboration with M. Samuelides (SupAéro, Toulouse), G. Beslon (INSA, Lyon), and E. Daucé (Perception et mouvement, Marseille).

References

[1] S. Amari. Dynamics of pattern formation in lateral-inhibition type neural fields. *Biological Cybernetics*, 27:77–87, 1977.

[2] M. Arbib and I. Lieblich. Motivational learning of spatial behavior. In J. Metzler, editor, *Systems Neuroscience*, pages 221–239. Academic Press, 1977.

[3] I. Bachelder and A. Waxman. Mobile robot visual mapping and localization: A view-based neurocomputational architecture that emulates hippocampal place learning. *Neural Networks*, 7(6/7):1083–1099, 1994.

[4] J. Banquet, P. Gaussier, M. Quoy, A. Revel, and Y. Burnod. A hierarchy of associations in hippocampo-cortical systems: cognitive maps and navigation strategies. *Neural Computation*, 17(6), 2005.

[5] R. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958.

[6] V. Braitenberg. *Vehicles : Experiments in Synthetic Psychology*. Bradford Books, Cambridge, 1984.

[7] N. Cuperlier, P. Laroque, M. Quoy, and P. Gaussier. Learning to plan and build experience via imitation in a social environment. Marbella (spain), sept 1-3, 2004. Artificial Intelligence and Soft Computing, IASTED 2004.

[8] J. Donnart and J. Meyer. Learning reactive and planning rules in a motivationnally autonomous animat. *IEEE Transactions on Systems, Man and Cybernetics-Part B*, 26(3):381–395, 1996.

[9] P. Gaussier, A. Revel, J. Banquet, and V. Babeau. From view cells and place cells to cognitive map learning: processing stages of the hippocampal system. *Biological Cybernetics*, 86:15–28, 2002.

[10] C. Joulain, P. Gaussier, and A. Revel. Learning to build categories from perception-action associations. In *International Conference on Intelligent Robots and Systems - IROS'97*, pages 857–864, Grenoble, France, September 1997. IEEE/RSJ.

[11] J. O'Keefe and N. Nadel. *The hippocampus as a cognitive map*. Clarendon Press, Oxford, 1978.

[12] M. Quoy, P. Gaussier, S. Leprêtre, A. Revel, C. Joulain, and J. Banquet. *Lecture Notes in Artificial Intelligence Series, 1812*, chapter A planning map for mobile robots: speed control and paths finding in a changing environment, pages 103–119. Springer, ISBN 3-540-41162-3, 2000.

[13] M. Quoy, S. Moga, and P. Gaussier. Dynamical neural networks for top-down robot control. *IEEE transactions on Man, Systems and Cybernetics, Part A*, 33(4):523–532, 2003.

[14] A. Revel, P. Gaussier, S. Leprêtre, and J. Banquet. Planification versus sensory-motor conditioning: what are the issues ? In *From Animals to Animats : Simulation of Adaptive Behavior SAB'98*, pages 129–138, 1998.

[15] G. Schöner, M. Dose, and C. Engels. Dynamics of behavior: theory and applications for autonomous robot architectures. *Robotics and Autonomous System*, 16(2-4):213–245, December 1995.

[16] N. Tinbergen. *The study of instinct*. Oxford University Press, London, 1951.

[17] E. Tolman. Cognitive maps in rats and men. *The Psychological Review*, 55(4), 1948.

Real-Time Hierarchical POMDPs for Autonomous Robot Navigation

Amalia Foka Panos Trahanias

Institute of Computer Science
Foundation for Research and Technology – Hellas (FORTH)
P.O.Box 1385, Heraklion, 711 10 Crete, Greece
and
Department of Computer Science, University of Crete
P.O.Box 1470, Heraklion, 714 09 Crete, Greece
e-mail:{foka,trahania}@ics.forth.gr

Abstract

This paper proposes a novel hierarchical representation of POMDPs that for the first time is amenable to real-time solution. It will be referred to in this paper as the Robot Navigation - Hierarchical POMDP (RN-HPOMDP). The RN-HPOMDP is utilized as a unified framework for autonomous robot navigation in dynamic environments. As such, it is used for localization, planning and local obstacle avoidance. Hence, the RN-HPOMDP decides at each time step the actions the robot should execute, without the intervention of any other external module. Our approach employs state space and action space hierarchy, and can effectively model large environments at a fine resolution. Finally, the notion of the *reference* POMDP, that holds all the information regarding motion and sensor uncertainty is introduced, which makes our hierarchical structure memory efficient and enables fast learning. The RN-HPOMDP has been tested extensively in a real-world environment.

1 Introduction

The autonomous robot navigation problem has been studied thoroughly by the robotics research community over the last years. The navigation problem involves the three main tasks of mapping, localization and path planning. Incorporating uncertainty in methods for navigation is crucial to their performance due to the the robot motion uncertainty and sensor uncertainty. Hence, probabilistic methods dominate the proposed approaches present in the literature. However, probabilistic methods, that integrate uncertainty, for motion planning have not been well studied until now in contrast to probabilistic methods for mapping and localization. Contemporary methods for robot motion planning [6] do not take into account the involved uncertainty. The probabilistic path planning methods present in the literature so far are dominated by methods based on Partially Observable Markov Decision Processes (POMDPs) [8, 13, 16, 15, 10, 9, 14] but they are mainly utilized only as high level path planners due to the computational complexity involved and require a lower level path planner, that most commonly is not probabilistic, to drive the robot between intermediate points and also perform obstacle avoidance. In this paper a Hierarchical POMDP (HPOMDP) is employed that facilitates probabilistic navigation where the probabilistic nature of POMDPs is exploited in all aspects of navigation tasks. The proposed HPOMDP solves in a unified manner the navigation tasks of localization, path planning and obstacle avoidance.

POMDPs provide the mathematical framework for probabilistic planning. POMDPs model the hidden state of the robot that is not completely observable and maintain a belief distribution of the robot's state. Planning with POMDPs is performed according to the belief distribution. Therefore, actions dictated by a POMDP drive the robot to its goal but also implicitly reduce the uncertainty of its belief.

Although POMDPs successfully meet their purpose of use, they are intractable to solve with exact methods when applied to real-world environments modelled at a fine resolution. Many approximation methods for solving POMDPs are present in the literature that have also been applied to robotics problems [1, 12, 4, 16, 9, 14]. The approximation methods presented in the literature so far can only deal with problems where the size of the state space is limited to at most a few thousands states. As a result, approximation methods known so far cannot model large real world environments at a fine resolution and hence POMDPs are used as high level mission planners. Furthermore, even when POMDPs are able to model large environments [12] they have to be amenable to real time solution to be applied as unified navigation model that can perform the navigation tasks of localization, path planning and obstacle avoidance since the POMDP will have to be solved at each time step.

In this paper, we propose a hierarchical representation of POMDPs for autonomous robot navigation, termed as the Robot Navigation-HPOMDP (RN-HPOMDP) that can efficiently model large real world environments at a fine resolution. The RN-HPOMDP provides a representation that for the first time enables real-time POMDP solution even when the state space size is extremely large. In effect, the RN-HPOMDP is solved on-line at each time step and decides the actual actions the robot performs, without the intervention of any other external modules. Hence, the RN-HPOMDP is utilized as a unified framework for the autonomous robot navigation problem, that integrates the modules for localization, planning and local obstacle avoidance.

Two other HPOMDP approaches are currently present in the literature that employ either state space hierarchy [15], applied as a high level mission planner, or action and state space hierarchy [11], applied for high level robot control and dialogue management. Independently and concurrently with these works we have come up with the RN-HPOMDP[1] that applies both state space and action space hierarchy. It is specifically designed for the autonomous robot navigation problem and offers specific advantages over the two approaches mentioned above. A comparison between the RN-HPOMDP and the mentioned approaches can be found in Section 4.

Experimental results have shown the applicability of the RN-HPOMDP for autonomous robot navigation in large real world and dynamic environments where humans and moving objects are effectively avoided and the robot follows optimal paths to reach its destination.

2 Partially Observable Markov Decision Processes (POMDPs)

POMDPs are a model for planning under uncertainty [5]. A POMDP is a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{Z}, \mathcal{O} \rangle$, where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$, is a finite set of actions, $\mathcal{T}$ is the *state transition function*, $\mathcal{Z}$, is a finite set of observations, $\mathcal{O}$ is the *observation function* and $\mathcal{R}$ is the *reward function*, giving the expected immediate reward gained by the agent for taking each action in each state. The robot maintains a belief distribution at all times, b_t, over the set of environment states, $\mathcal{S}$.

Each state s represents the location (x, y) of the robot and its orientation θ, termed as the *orientation angle*. The set of actions is composed of all possible rotation actions a from 0° to 360° that are termed as *action angles*. The set of observations is instantiated in our approach as the output of the *iterative dual correspondence* (IDC) [7] algorithm for scan matching. Therefore, the output of the IDC algorithm, that is the dx, dy and $d\theta$ from the estimated location provided, is discretized and the observation set if formed.

The RN-HPOMDP provides the actual actions that are executed by the robot and also carries out obstacle avoidance for moving objects. Therefore, the reward function is built and updated at each time step according to two reward grid maps (RGMs): a *static* and a *dynamic*. The RGM is defined as a grid map of the environment in analogy with the OGM. Each of the RGM cells corresponds to a specific area of the environment with the same discretization of the OGM, only that the value associated with each cell in the RGM represents the reward that will be assigned to the robot for ending up in the specific cell. The static RGM is built once by calculating the distance of each cell to the goal position and by incorporating information about cells belonging to static obstacles. The dynamic RGM is responsible for incorporating into the model information on whether there are objects moving within it or other unmapped objects. Superimposing the static and dynamic RGMs provides the reward function that is updated at each time step. The use of the static and dynamic RGM alleviates the need for modelling moving objects as observations.

3 The RN-HPOMDP

The RN-HPOMDP is built through an automated procedure using as input a map of the environment and the desired discretization of the state and action space. The map of the environment can be either a probabilistic grid map obtained at the desired discretization or a CAD map.

The RN-HPOMDP structure is built by decomposing a POMDP with large state and action space into multiple POMDPs with significantly smaller state and action spaces. The process of building the hierarchical structure is performed in a top-down approach. The number of levels of the hierarchical structure is determined by the desired discretization of the action angles or the orientation angles, since their discretization is the same in the RN-HPOMDP. Thus, if the desired discretization of the action angles or the orientation angles is ϕ, the number of levels of the RN-HPOMDP, L, will be $L = \log_2(90^\circ/\phi) + 1$.

The number of levels of the RN-HPOMDP in conjunction with the desired discretization of the state space affects the size of the top-level POMDP and in effect the performance of the RN-HPOMDP regarding the time complexity of solving it.

[1] Preliminary versions of the RN-HPOMDP are presented in [2, 3]

Table 1: Properties of the RN-HPOMDP with L levels.

	Top Level	**Intermediate Level l**
No of POMDPs	1	$\lvert\mathcal{A}^{l-1}\rvert \times \lvert\mathcal{S}^{l-1}\rvert$
$\lvert\mathcal{S}\rvert$	$\lvert\mathcal{S}^0\rvert/2^{2(L-1)}$	20 except when $l = L$ where $\lvert\mathcal{S}\rvert = 5 \times (2+r)^2$
θ and a range	$[0°, 360°]$	$(\theta_p, \alpha_p) \pm (90°/2^{l-1})$
θ and a resolution	$90°$	$90°/2^{l-1}$
$\lvert\mathcal{A}\rvert$	4	5

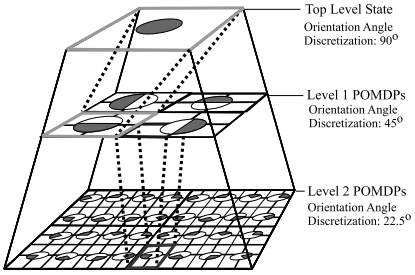


Figure 1: State space hierarchy decomposition. Orientation angle range is denoted by the shaded region of the circles for each POMDP.

The top level of the hierarchical structure is composed of a single POMDP with very coarse resolution so it can represent the whole environment in a small number of states. The grid resolution of the top level states is equal to $d \times 2^{L-1}$, where d is the desired discretization of the corresponding flat POMDP. The orientation angle of the robot and the action angles are also discretized in a very coarse resolution of $90°$ and thus represent the basic four directions $[0°, 90°, 180°, 270°]$.

To summarize, the top-level is always composed of a single POMDP with predefined discretization of the orientation and action angles at $90°$. The state space size of the top-level POMDP is variable and dependent on the discretization of the corresponding flat POMDP and the number of levels of the hierarchical structure. Hence, the number of levels of the RN-HPOMDP, L, should be such that it ensures that the size of the top-level POMDP remains small.

Subsequent levels of the RN-HPOMDP are composed of multiple POMDPs, each one representing a small area of the environment and a specific range of orientation angles. The actions of an intermediate level POMDP are also a subset of the actions of the corresponding flat POMDP.

In detail, each state of the top level POMDP corresponds to a POMDP at the immediate next level, as we go down the hierarchical structure. A POMDP at an intermediate level l, has states that represent grid locations of the environment at a resolution of $d \times 2^{(L-l)}$. Thus, by going down the hierarchical structure the grid resolution of a level's POMDPs is always twice the resolution of the previous level. Therefore, when a top level state, that corresponds to a specific grid location, is decomposed it will be represented in the immediate next level POMDP by an area of 2×2 cells with double grid resolution than the top level's grid resolution.

Going down the hierarchical structure, the resolution of the orientation angle is also doubled. Since the resolution of the orientation angle is increased as we go down the hierarchical structure, the whole range of possible orientation angles, $[0°, 360°]$, cannot be represented in every intermediate level POMDPs. This would dramatically increase the size of the state space and therefore we choose to have many POMDPs that represent the same grid location but with a different range of orientation angles.

The range of orientation angles that is represented within each intermediate level POMDP is expressed in terms of the orientation angle, θ_p, of the previous level state that is decomposed, and is equal to $\left[\theta_p \pm \left(90°/2^{l-2}\right)\right]$, where l is the current intermediate level. By the above expression of the range of orientation angles, every intermediate level POMDP will always have five distinct orientation angles. For example, if the state of the top level POMDP, $l = 1$, has orientation angle $\theta_p = 90°$, the range of orientation angles at the next level, $l = 2$, will be equal to $[0°, 180°]$. As mentioned earlier the angle resolution of the top level is always equal to $90°$ and the next level will have double resolution, i.e. $45°$. Therefore, the range of orientation angles $[0°, 180°]$ will be represented by five distinct orientation angles. Consequently, the size of the state space for every intermediate level POMDP is constant and equal to 20, since it always has five possible orientation angles and it represents a 2×2 area of grid locations.

Action angles are decomposed from the top level POMDP to the next intermediate level in the same manner as with the orientation angles. The resolution of the action angles at each level is the same as the resolution of the orientation angle. Hence, it is equal to $90°/2^{l-1}$. As a result, a top level state is also decomposed into multiple POMDPs, each one with a different range of orientation angles but also with a different range of action angles. The range of an action set is equal to $\left[a_p \pm \left(90°/2^{l-2}\right)\right]$, where a_p is the previous level action and l is the current intermediate level. The action angles set is also always composed of five distinct actions according to the above expression.

The procedure described is used to built all intermediate levels of the hierarchical structure until the bottom level is reached. Bottom level POMDPs' state and action space is discretized at the desired resolution as a flat POMDP would be discretized. The bottom level is composed of multiple POMDPs having the same properties as all other intermediate levels' POMDPs, only that

while not reached the goal state
    `compressTopBelief`(*top level*)
    a_p = `solveTopLevel`(*top level*)
    for $l = 2$ to L
        $whichPOMDP$ = `selectPOMDP`(l, a_p)
        `compressBelief`$(l, whichPOMDP)$
        a_p = `solveLevel`$(l, whichPOMDP)$
    end
    `executeAction`(a_p)
    z = `getObservation`()
    $belief_{l,whichPOMDP}$ = `updateBelief`$(l, whichPOMDP, a_p, z)$
    `updateFullBelief`$(belief_{l,whichPOMDP}, l, whichPOMDP)$
end

Table 2: RN-HPOMDP planning.

the grid location the bottom level POMDPs represent is overlapping by a region r. Overlapping regions are required to be able to solve the bottom level POMDPs for border location states.

The properties of the RN-HPOMDP are summarized in Table 1.

3.1 Planning with the RN-HPOMDP

Solving the RN-HPOMDP to obtain the action the robot should perform, involves solving a POMDP at each level. The intuition of the RN-HPOMDP solution is to obtain at first a coarse path that the robot should follow to reach a goal position, and then refine this path at each subsequent level in the area that the robot's current position lies. In Table 2 the algorithm for the RN-HPOMDP planning procedure is detailed.

During the RN-HPOMDP planning procedure the belief distribution of the corresponding flat POMDP is maintained at all times. This distribution will be denoted as the *full belief*. Before solving any POMDP at a level, the *full belief* is compressed, by the functions `compressTopBelief()` and `compressBelief()`, to obtain the belief distribution of the POMDP to be solved. Belief compression is performed according to the state abstraction present at each level of the RN-HPOMDP structure. Therefore, the belief assigned to an abstract state will correspond to the average belief of all the corresponding flat POMDP states that has integrated. The belief distribution obtained for any POMDP is normalized before solving it.

The top level POMDP is solved, by the function `solveTopLevel()`, at an infinite horizon, until the goal state is reached. The immediate abstract action to be executed, a_p, as dictated by the top level POMDP solution determines which POMDP at the immediate next level of the hierarchical structure will be solved to obtain a new refined abstract action.

The POMDP to be solved at the next level is determined by the function `selectPOMDP()`. This function searches a level l for the POMDP that satisfies the following two criteria:

- The zero moment of the full belief distribution over the area that is defined by the candidate POMDP states is maximum.
- The set of actions of the candidate POMDP contains an action that has minimum distance from the the previous level solution's action, a_p.

The structure of the RN-HPOMDP, as described in Section 3, ensures that when solving an intermediate level POMDP the action obtained from the previous level will be refined to a new action since the action subset range is equal to $\left[a_p \pm \left(90^\circ/2^{l-2}\right)\right]$. Therefore the solution of an intermediate level POMDP is bounded according to the previous level solution.

The described procedure continues until the bottom level is reached where an abstract action will be refined to an actual action, that is the action the robot will perform.

When the robot executes the action obtained by the bottom level POMDP solution, an observation, z, is obtained and the belief distribution of this bottom level POMDP is updated by `updateBelief`(). Bottom level POMDPs are composed of actual states and actions, i.e. subsets of states and actions that compose the corresponding flat POMDP. Hence, updating the belief of a bottom level POMDP amounts to updating a specific region of the *full belief*. Therefore, the belief distribution of the bottom level POMDP that was solved is transferred to the *full belief* by the function `updateFullBelief`().

All POMDPs at all levels are solved in our current implementation using the Voting heuristic, that is an MDP-based approximation method. However, this is not an inherent feature of the RN-HPOMDP structure, as any other POMDP solution method can be used. Furthermore, the POMDP solution method used can also be different for each level of the hierarchical structure.

3.2 Complexity Analysis of the RN-HPOMDP Solution

In the complexity analysis that follows, computation time complexities are evaluated for the RN-HPOMDP solution using exact methods and heuristics.

The flat POMDP solution has time complexity, for a single step, $\mathbf{O}\left(|\mathcal{S}|^2|\mathcal{A}|\right)$ when solved with the MLS or Voting heuristic. Referring to Table 1, where the properties of the RN-HPOMDP structure are detailed, the solution of the top level POMDP requires $\mathbf{O}\left(\left(|\mathcal{S}|/2^{2(L-1)}\right)^2\right)$ time, where L is the number of levels of the hierarchical structure.

The solution of all intermediate levels POMDPs requires $\mathbf{O}(C_1)$ time, since the size of the state space and action space is constant and predefined. The bottom level POMDP solution is $\mathbf{O}(C_2)$, since the state space and action space is again constant and predefined. Therefore, the total time required to solve the RN-HPOMDP reduces to actually the complexity of the top level POMDP. The top-level POMDP state and action space size can remain small regardless of the size of the whole environment by increasing the number of levels, L, of the hierarchical structure.

When solving a flat POMDP exactly for a single step in time t, the time complexity is $\mathbf{O}\left(|\mathcal{S}|^2|\mathcal{A}||\Gamma_{t-1}|^{|\mathcal{Z}|}\right)$, where $|\Gamma_{t-1}|$ is the number of linear components required to represent the value function at time $t-1$. The size of Γ at any time t is equal to $|\Gamma_t| = |\mathcal{A}||\Gamma_{t-1}|^{|\mathcal{Z}|}$.

The time complexity and size of the RN-HPOMDP when solved exactly is $\mathbf{O}\left(\left(\left(|\mathcal{S}|/2^{2(L-1)}\right)\right)^2|\Gamma_{t-1}|^{|\mathcal{Z}|}\right)$ and $|\Gamma_t| = |\Gamma_{t-1}|^{|\mathcal{Z}|}$, respectively.

Apart from the notable reduction in computation time due to the reduced size of the state and action space, it should be noted that the above mentioned times are for a single time step. The infinite horizon solution of a flat POMDP would require these computations to be repeated for a number N of time steps until the goal point is reached, that is dependent on the number of states of the flat POMDP, $|\mathcal{S}|$. In the RN-HPOMDP case, only the top level POMDP is solved at an infinite horizon, and the number of time steps N' until the goal point is reached, is now dependent on the number of states of the top level POMDP, $(|\mathcal{S}|/2^{2(L-1)})$.

From this short complexity analysis, we may conclude that the particular POMDP formulation in our approach takes care of the "curse of dimensionality" [5] and also the "curse of history" [9].

3.3 The Reference POMDP

The RN-HPOMDP described in the previous section, can cope with the computational time requirements but cannot address the memory requirements. A flat POMDP would require to hold a transition matrix of size $(|\mathcal{S}|^2 \times |\mathcal{A}|)$ and an observation matrix of size $(|\mathcal{S}| \times |\mathcal{A}| \times |Z|)$.

The RN-HPOMDP structure requires to hold the transition and observation matrices for all the POMDPs at all levels. As it can be seen in Table 1 the number of POMDPs at each level is large and dependent on the size of action space and state space. Consequently, even thought each POMDP's observation and translation matrix is small the total memory requirements would be extremely large. The RN-HPOMDP has larger memory requirements than the flat POMDP, although the flat POMDP memory requirements are already very hard to manage for large environments. For this reason, the notion of the *reference* POMDP (rPOMDP) is introduced.

The transition and observation matrices hold probabilities that carry information regarding the motion and sensor uncertainty. In the formulation of the autonomous robot navigation problem with POMDPs, as described in Section 2, transition and observation probabilities for a given action, a, and an observation, z, depend actually only on the relative position and orientation of the robot. This is due to the design choice to model the environment structure and state in the reward function instead of the transition and observation matrices as commonly used in the POMDP literature. Therefore, the transition and observation probabilities are dependent only on the robot motion model.

The transition probability of a robot from a state s to a new state s', when it has performed an action a is only dependent on the action a. Therefore when the robot is executing an action a, the transition probability will be the same for any state s when the resulting state s' is defined relatively to the initial state s.

The probability that the robot observes a feature z, when it is in a state s and performs an action a, can also be defined in the same manner as with the transition probabilities, since the set of features $\mathcal{Z}$ has been defined in Section 2 to be the result of the scan matching algorithm when feeded with a reference laser scan and the actual scan the robot perceived. Therefore, perceived features are dependent on the motion of the robot, i.e. the action a it performed.

The rPOMDP is built by defining a very small state space, defined as an $R \times R$ square grid (in our implementation $R = 7$) representing a subset of possible locations of the robot and all the orientation angles of the robot that would be assigned in the flat POMDP. The size of the grid that defines the possible robot locations in the rPOMDP is established by determining the largest possible location transition when a single action a is executed. This is due to the fact that the rPOMDP conveys the transition and observation probabilities based only on the actual robot motion independently of the exact location of the robot in the environment. The center location of the state space represents the invariant state s_r of the robot. The action and observation spaces are defined in the same manner they would be defined for the original POMDP. This rPOMDP requires to hold transition and observation matrices of size $((R \times 2^{2+L})^2 \times |A|)$ and $((R \times 2^{2+L})^2 \times |A| \times |Z|)$, respectively. The size of the matrices is only dependent on the size of the set of actions and observations and the number of levels of hierarchy, L, since the number of levels defines the discretization of the robot's orientation angle.

Transition and observation probabilities for each POMDP in the hierarchical structure are obtained by translating and rotating the reference transition and observation probability distributions over the current POMDP state space. The transfer of probabilities is performed on-line while a POMDP is solved or the robot's belief is updated.

The transition probability for any POMDP of the hierarchical structure, $\mathcal{T}(s, s', a)$, is equivalent to the transition probability of the rPOMDP, $\mathcal{T}_r(s_r, s'_r, a_r)$. The reference result state, s'_r, is determined by the following equation:

$$\begin{bmatrix} x'_r \\ y'_r \\ f'_r \end{bmatrix} = \begin{bmatrix} x_r \\ y_r \\ f_r \end{bmatrix} + \begin{bmatrix} x' - x \\ y' - y \\ f' - f \end{bmatrix},$$

where, the states s, s', s_r and s'_r are decomposed to the location and orientation triplets $(x, y, f), (x', y', f'), (x_r, y_r, f_r)$ and (x'_r, y'_r, f'_r), respectively. The reference action is determined by $a_r = a + f - f_r$.

In the same manner, the observation probability for any POMDP of the hierarchical structure, $\mathcal{O}(s, z, a)$, is equivalent to the observation probability of the rPOMDP, $\mathcal{O}_r(s_r, z_r, a_r)$. The reference observation, z'_r, is now determined as:

$$\begin{bmatrix} dx_r \\ dy_r \\ df_r \end{bmatrix} = \begin{bmatrix} d\cos(f_r + a_r) \\ d\sin(f_r + a_r) \\ df \end{bmatrix},$$

where the observations z and z_r are decomposed into (dx, dy, df) and (dx_r, dy_r, df_r), respectively, as observations are defined as the position and angle difference between laser scans, and d is the distance $d = \sqrt{dx^2 + dy^2}$.

4 Comparison with other HPOMDPs

4.1 Comparison with the Theocharous approach

The Theocharous [15] approach uses a topological map of the environment where state abstraction in high levels of the HPOMDP, has a physical meaning based on the environment. Thus, abstract states are manually defined such that they represent a corridor or a junction.

The Theocharous HPOMDP has been used as a high-level planner where the POMDP is solved once to obtain the shortest path to the goal position. As a result, the state space resolution is set to $2m^2$ and the action space is discritized at a resolution of $90°$.

The Theocharous approach, uses the MLS heuristic and has time complexity[2] between $\mathbf{O}(|\mathcal{S}|^{\frac{2}{d}} N|\mathcal{A}|)$ and $\mathbf{O}(|\mathcal{S}|^2|\mathcal{A}|)$, based on how well the HPOMDP was constructed. The time required to solve the proposed RN-HPOMDP, with the MLS heuristic, is $\mathbf{O}((|\mathcal{S}|/2^{2(L-1)})^2)$, hence the complexity reduction of our approach is significantly greater and also is not dependent on any quality measure of the hierarchical structure.

4.2 Comparison with the Pineau approach

In the Pineau HPOMDP approach [11], actions are grouped into abstract actions called subtasks. Subtasks are defined manually and according to them state abstraction is performed automatically. States that have the same reward value for executing any action that belongs to a predefined subtask are clustered. Observation abstraction is performed by eliminating the observations that have zero probability over all state clusters for that actions belonging to a specific subtask.

[2] d is the depth of the tree and N is the maximum number of entry states for an abstract state.

Planning with the Pineau HPOMDP involves solving the POMDP defined for each action subtask, that are solved using the exact POMDP solution method.

The HPOMDP proposed by Pineau does not have a guaranteed reduction of the action space and state space since it is dependent on the action abstraction that is defined manually. The authors have performed experiments (real and simulated) only for problems of high level behavior control. Hence it is not clear whether their approach of state abstraction could be applied to the problem of the autonomous robot navigation in the context that we have defined or more importantly if it would perform as well as the RN-HPOMDP does, since it has a guaranteed reduction of the state space that is equal to $\left(|\mathcal{S}|/2^{2(L-1)}\right)$. On the other hand, the authors in [11] do not state how well their approach performs in terms of state space abstraction.

4.3 Approximation methods for solving flat POMDPs

Reference [4] presents a review of approximation methods for solving POMDPs. The complexity of the methods reviewed there is in the best case polynomial to the POMDP size. Furthermore, one of the most recent methods for approximation is the Point Based Value Iteration (PBVI) [9] method, where its complexity is again polynomial to the size of the POMDP.

All the above mentioned methods have been applied to problems where in the best case the POMDP comprised of a few thousand states with an exception of the work in [12] where the POMDP is comprised of millions of states as with our approach but it cannot be solved in real time. The problem we consider consists of many orders of magnitude larger state space. As a result the reduction of the state space that the RN-HPOMDP offers and also the reduction of the action space is crucial to its performance. Furthermore, since the proposed RN-HPOMDP is not restricted to a specific method for solving the underlying POMDPs, a combination of an approximation method for solving a flat POMDP with the proposed hierarchical structure improves dramatically its performance.

5 Experimental Results

In Table 3, the CPU time required to solve the proposed HPOMDP structure using the Voting heuristic for varying grid size and number of levels is given, where as it can be observed with appropriate choice of the number of levels real time POMDP solution is possible. It is evident from the results presented in Table 3 that the RN-HPOMDP is amenable to real-time solution in problems with extremely large state and action spaces.

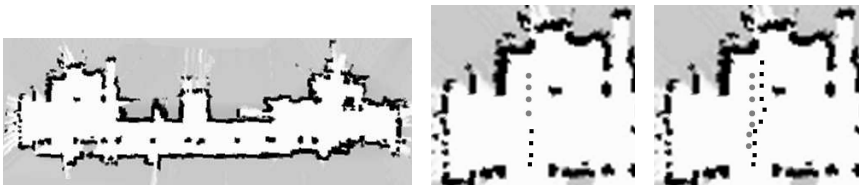
Table 3: Computation time required to solve the proposed HPOMDP.

No. of Levels	Grid size	POMDP size		time (sec)	No. of Levels	Grid size	POMDP size		time (sec)
5	$5cm^2$	$\|S\| = 18,411,520$	$\|A\| = 64$	18.520	3	$10cm^2$	$\|S\| = 1,150,720$	$\|A\| = 16$	201.210
5	$10cm^2$	$\|S\| = 4,602,880$	$\|A\| = 64$	0.911	4	$10cm^2$	$\|S\| = 2,301,440$	$\|A\| = 32$	16.986
5	$15cm^2$	$\|S\| = 2,038,080$	$\|A\| = 64$	0.426	5	$10cm^2$	$\|S\| = 4,602,880$	$\|A\| = 64$	0.911
5	$20cm^2$	$\|S\| = 1,150,720$	$\|A\| = 64$	0.257	6	$10cm^2$	$\|S\| = 9,205,760$	$\|A\| = 128$	0.460
5	$25cm^2$	$\|S\| = 734,976$	$\|A\| = 64$	0.262	7	$10cm^2$	$\|S\| = 18,411,520$	$\|A\| = 256$	0.411
5	$30cm^2$	$\|S\| = 503,808$	$\|A\| = 64$	0.251					

The RN-HPOMDP has been tested extensively in a real world environment. The robot was set to operate for more than 70 hours in the FORTH main entrance hall shown in Figure 2. The environment was modeled with a RN-HPOMDP of size $|S| = 18,411,520$, $|A| = 256$ and $|Z| = 24$, built with 7 levels. Experiments were performed in a dynamic environment where people were moving within it. A sample path the robot followed to reach its goal and also performed local obstacle avoidance to avoid a human is shown in Figure 2.

6 Conclusions and Future Work

In this paper, a novel hierarchical representation of POMDPs for autonomous robot navigation has been proposed that can be solved for the first time in real-time when an extremely large state space is involved and is memory efficient. Hence, the RN-HPOMDP provides a unified framework for robot navigation that is able to provide the actual actions the robot executes without the intervention of any other external modules. Our proposed hierarchical structure employs state space and action space



(a) (b) (c)

Figure 2: The FORTH main entrance hall and avoiding a human to reach the goal position. The robot track is marked with the black dots (•) and the human track is marked with the grey dots (•).

hierarchy. Memory efficiency is achieved by introducing the *reference* POMDP that holds all the information regarding motion and sensor uncertainty. Our comparative experiments have indicated that our approach results in very efficient computation times and manageable memory requirements for realistic environments. The RN-HPOMDP provides an approximate solution suited for the robot navigation problem. Preliminary versions of the proposed RN-HPOMDP structure have already been applied for predictive obstacle avoidance [2] and robot velocity control in dynamic environments [3]. Future work involves applying the RN-HPOMDP into the multi-robot navigation problem.

References

[1] A. R. Cassandra, L. P. Kaelbling, and J. A. Kurien. Acting under uncertainty: Discrete bayesian models for mobile-robot navigation. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1996.

[2] A. Foka and P. Trahanias. Predictive autonomous robot navigation. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots & Systems (IROS)*, 2002.

[3] A. Foka and P. Trahanias. Predictive control of robot velocity to avoid obstacles in dynamic environments. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots & Systems (IROS)*, 2003.

[4] M. Hauskrecht. Value function approximations for Partially Observable Markov Decision Processes. *Journal of Artificial Intelligence Research*, 13:33–95, 2000.

[5] L. P. Kaebling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, 1998.

[6] J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[7] F. Lu and E. Milios. Robot pose estimation in unknown environments by matching 2d range scans. *Journal of Intelligent and Robotic Systems*, 18:249–275, 1998.

[8] I. Nourbakhsh, R. Powers, and S. Birchfield. Dervish an office-navigating robot. *AI Magazine*, 16(2):53–60, 1995.

[9] J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *Proc. Int. Joint Conf. on Artificial Intelligence (IJCAI)*, 2003.

[10] J. Pineau, N. Roy, and S. Thrun. A hierarchical approach to POMDP planning and execution. *Workshop on Hierarchy and Memory in Reinforcement Learning (ICML)*, 2001.

[11] J. Pineau and S. Thrun. An integrated approach to hierarchy and abstraction for POMDPs. Technical Report CMU-RI-TR-02-21, Carnegie Mellon University, 2002.

[12] P. Poupart and C. Boutilier. VDCBPI: an approximate scalable algorithm for large scale POMDPs. In *Neural Information Systems (NIPS)*, 2004.

[13] R. Simmons, J. Fernandez, R. Goodwin, S. Koenig, and J. O'Sullivan. Xavier: An autonomous mobile robot on the web. *Robotics and Automation Magazine*, 1999.

[14] M. Spaan and N. Vlassis. A point-based POMDP algorithm for robot planning. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, 2004.

[15] G. Theocharous. *Hierarchical Learning and Planning in Partially Observable Markov Decision Processes*. PhD thesis, Michigan State University, 2002.

[16] S. Thrun, M. Beetz, M. Bennewitz, W. Burgard, A. Cremers, F. Dellaert, D. Fox, D. Hahnel, C. Rosenberg, N. Roy, J. Schulte, and D. Schulz. Probabilistic algorithms and the interactive musuem tour-guide robot minerva. *International Journal of Robotics Research*, 19(11):972–999, 2000.

Speeding up Reinforcement Learning using Manifold Representations: Preliminary Results

Robert Glaubius, Motoi Namihira, and William D. Smart

Department of Computer Science and Engineering
Washington University in St. Louis
St. Louis, MO 63130
United States
rlg1@cse.wustl.edu, mjn1@cec.wustl.edu, wds@cse.wustl.edu

Abstract

Reinforcement Learning (RL) has proven to be a useful set of techniques for planning under uncertainty in robot systems. Effective RL algorithms for this domain need to be able to deal with large, continuous state spaces, and must make efficient use of experience. In this paper, we present methods to better leverage observed experience by reusing experience across parts of the problem state space that are known to be similar. We present experimental results in a navigational, goal-based domain. We develop an approach to identifying portions of the world that appear similar based on observed transition samples.

1 Introduction

Reinforcement Learning (RL) has proven to be a useful set of techniques for planning under uncertainty in robot systems. Effective RL algorithms for this domain need to be able to deal with large, continuous state spaces, and must make efficient use of experience. In previous work [5, 4] we have shown that a value-function representation (VFA) scheme based on manifolds can effectively deal with continuous state problems. In this paper, we briefly summarize this approach, and show how it can be used to make efficient use of experience in large domains of the type typically encountered in robotics applications.

One particular hurdle in real-world domains in general, and robotics in particular, is the expense of acquiring experience. Physical agents are subject to the cost of moving about in a real environment. These experiences are also constrained to occur around the agent's trajectory through the state space – the agent may not arbitrarily sample the space without intervention.

However, one observation is that, particularly in navigation domains, much of the world behaves similarly. Traveling down one corridor is much the same as traveling down another, and hitting a wall should generalize between most walls. Continuous-state reinforcement learning does not handle these similarities between topologically similar states, and the agent must learn a local policy from scratch for each.

In this paper, we use local features of the manifold representation to allow the reuse across many states of experiences observed elsewhere during training. This extends our work on manifold-based value-function approximation by incorporating synthesized experience into the training data. By "replaying" experiences gained in one part of the world in other parts of the world that are similar, we expect to approach a good policy more quickly, and to require fewer actual experiences.

2 Background

In the RL framework, an agent interacts with the environment by observing the current environment state s, taking an action a, and observing the new state of the environment s' and a real-valued reward signal r. The evolution of states is governed by the transition function $T(s, a)$, which maps a state and action to a distribution over possible next states.

One popular approach to solving RL problems is to estimate the Q-value of each state-action (s, a) pair. The optimal state-action value function $Q^*(s, a)$ is the expected sum of rewards observed when taking action a from state s and behaving optimally from then on. To obtain an optimal policy, the agent simply chooses the action that maximizes Q^* for its current state.

The update equation shown in equation 1 is the Q-learning rule [17]. Iteratively applying this rule to an initial estimate of the state action value function is guaranteed to converge to the optimal value function. Each update is based on a sample (s, a, r, s'), where s is some initial state, a is the action taken, and r and s' are the observed reward and resulting state, respectively. The learning rate α controls the contribution of new experiences to the current estimate. The discount factor $\gamma \in [0, 1]$ weights the effect of future rewards. One interpretation of γ is that it is the prior probability of surviving to the next time step.

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha[r + \max_{a \in A} Q(s', a)] \tag{1}$$

2.1 Related Work

Q-learning is guaranteed to converge only in the case of an exhaustive tabular representation of the Q function, making the application problems with continuous state a non-trivial problem. Many real-world problems are most naturally modeled using a continuous state space; in this work we assume that the state space S is a connected subset of $\mathbb{R}^d$.

Although VFA has is not guaranteed to converge in the general case [3], there have been several successful applications reported, using a variety of function approximators. Tesauro used artificial neural networks to represent the state value function for the game of backgammon, resulting in an expert level of play [15]. Smart and Kaelbling used instance-based learning to represent the state-action value function for robot control tasks [11]. Munos and Moore used an informed decomposition of the state space to approximate the state value function with a combination of simple function approximators [10]. Sutton has successfully used tile coding approximators (often called CMACs [1]) in a variety of applications [12, 13].

Our VFA representation is most closely related to the tile coding approaches, in that it covers the domain of the value function with overlapping patches, and learns a partial value function on each of them. However, our approach takes the system dynamics into account when performing this covering, where tile coding typically does not. The work by Munos and Moore is also similar, in that it divides up the domain based on the system dynamics. They use non-overlapping states, however, and cannot make the strong continuity guarantees that our manifold-based approach can make.

In previous work we have argued that a manifold representation is sufficiently general to describe much of the existing corpus of methods for continuous-state value-function approximation in reinforcement learning. In this paper we present a method for reusing experience in portions of the state space that are similar, in the context of a manifold representation of the state space.

One way to view experience reuse is as a means to directly update the value of multiple states from one experience. Boutilier and Dearden [2] aggregate states during the construction of decision tree representations of the value function. In this sense, two states are equivalent if they have the same, or nearly the same, value. Our work differs in that it allows generalization between states where the system dynamics are the same, but values may be different.

Other researchers' work on exploiting the structure of the world has primarily focused on learning temporally extended actions to speed up learning [16, 8]. Lane and Wilson [7] derived conditions under which policies could be relocated in navigation tasks with relational domains. Our work differs in that we exploit knowledge about parts of the world that behave in a similar fashion to more efficiently use the experiences we have obtained.

2.2 Manifold Representations

In this section, we define precisely what we mean by "manifold". Before giving the formal definition of a manifold, we provide an intuition about the structure, and how it can be used constructively. Consider an atlas of the world. Each page in the atlas has a partial map on it, usually a single country. Each of these single-page maps can have a different scale, depending on the size of the country. This allows more

detail to be shown where it is necessary, without committing to representing the entire atlas at a particular resolution.

The pages in the atlas also overlap with each other at the edges. For example, the map for France has part of northern Spain on it. This provides a well-defined way of moving from one map page to another. Notice that the maps may be at different scales, and may not line up perfectly. However, we can still establish a correspondence between points in the overlap regions.

This method of using overlapping partial maps allows us to cover a complex surface (the surface of the Earth) with a set of simpler surfaces (the individual maps). Each local map can be appropriate to the local features, such as scale, and be topologically simpler than the global map. In the case of an atlas, we are covering a sphere with a set of (topological) disks. We can define global features, such as distance, by composing their local versions on each page, and dealing appropriately with the overlap regions.

We now make some definitions. Each map page in the above example is a *chart*. The collection of all charts is the *atlas*. The area shared by two adjacent pages is the *overlap region*. The function on each chart (for example, the elevation, or value function) is the *embedding function*.

Chart A homeomorphism φ from $U \subseteq \mathcal{S}$ to a disk in $\mathbb{R}^d$. A chart can be thought of as a local coordinate system on U.

Atlas A set Φ of charts. The manifold is defined as the union of the chart domains, $\mathcal{M} = \bigcup_{\varphi \in \Phi} \mathrm{dom}(\varphi)$ In this work, Φ will always be finite.

A manifold $\mathcal{M}$ is constructed by covering the state space $\mathcal{S}$ with an atlas. Assuming that the chart domains are selected appropriately, $\mathcal{M}$ will represent the manifold structure of the state space. We refer the reader to our previous work on methods used to construct a manifold representation given an RL problem instance [5].

2.3 Manifolds for Value Function Approximation

The basic idea of using a manifold representation for value-function approximation (VFA) is to model small, local value functions on each of the charts, then combine these local models into a larger global model. This has a number of significant benefits, including the ability to explicitly model the topology of the problem domain. Again, since VFA is not the focus of this paper, we will forego a detailed explanation here, and refer the interested reader to our previous work [5] for more details.

It is important to note that our use of the term "manifold" is somewhat different from the one currently in vogue in the machine learning community. Commonly in this literature, a "manifold" is the intrinsic space that a collection of data points are drawn from. Usually this manifold has a lower dimension than the space it's embedded in. Estimating the intrinsic dimension of the state space is not the focus of our manifold representations. Our usage is similar to the application of manifolds to surface modeling in computer graphics [6].

3 Speeding up Reinforcement Learning

The most relevant aspect of our value-function approximation scheme is the construction of a manifold representation. This models the problem state space as a set of smaller, overlapping chart domains. A local model, i.e., a function approximator, is embedded on each chart, and these local models are combined to give a global representation of the value function.

We can use this manifold structure to make more efficient use of the experiences gathered during training. Each of the charts covers an area of the state space that is locally self-similar. For many problem domains, these charts will form equivalence classes, where members of the class cover parts of the state space that are similar. For example, consider an empty room, with the robot state represented as its (x, y) position. If we cover this domain with relatively small (compared to the size of the room) charts, there will be intuitively three equivalence classes: "open space", "wall", and "corner". Each chart is a member of exactly one of these classes.

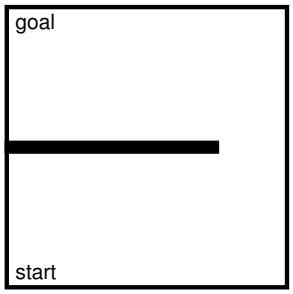


Figure 1: The experimental navigation domain.

We can define these equivalence classes according to the RL transition function over the charts. Two charts φ_i and φ_j are equivalent if the RL transition function, in their local coordinate frames, is "the same" in both. What do we mean when we say that the transition function is the same across two charts? We mean that the result of taking any action in $\mathrm{dom}(\varphi_i)$ moves the agent in the same direction, and the same distance, as that same action in φ_j. If we define equivalence in this way, we can reuse experience gathered in one chart in any chart that is equivalent to it.

Chart equivalence with respect to the transition function alone does ignore one important detail. In goal-based reinforcement learning, the reward function is tied to a region, or a set of regions. Any chart domain intersecting one of these regions can not be equivalent to one that is not, since generalizing a reward-bearing experience to a part of the domain that does not supply such a reward alters the problem that we are trying to solve. In the rest of this work, it is assumed that each chart intersecting a goal region is in its own singleton equivalence class.

Given this definition of equivalence does mean we have to revise our empty room example to include more equivalence classes. Charts that do not overlap a wall may still be close enough to a wall that an agent taking an action from inside that neighborhood may still strike a wall. This differentiates that particular chart from, say, an "open space" chart that is farther from walls. For now, we can work around this issue by restricting reuse only to experiences (s, a, r, s'), where s and s' have at least one chart domain in common, i.e., if there exists a chart φ such that s and s' are both in $\mathrm{dom}(\varphi)$, then that experience generalizes to charts equivalent to φ.

If we assume that all charts are the same size, and that there is a single source of non-default reward, experience reuse is straightforward. Suppose we observe the experience (s, a, r, s'), where $s, s' \in \mathrm{dom}(\varphi_i)$. Suppose that we also have a function ψ_{ij} that maps experiences from $\mathrm{dom}(\varphi_i)$ to $\mathrm{dom}(\varphi_j)$ (in the case of our uniform-sized charts, this is just a translation). Then, for each chart φ_j equivalent to φ_i, we synthesize the experience $\psi_{ij}(s, a, r, s')$, and update the local approximator on φ_j according to the synthesized experience. If there are many charts in a particular equivalence class, this will lead to a dramatic increase in learning speed.

4 Results

In this section, we present some experimental results to show the effectiveness of the techniques described above. Our experiments were performed in a simple navigation domain. The agent starts in the lower right corner of the world, and gets a reward of +10 for reaching the upper right corner, as shown in Figure 4. The agent can move in the four cardinal directions, with a reward of -1 on every step that does not end at the goal state. The state space of the problem is continuous, with two dimensions corresponding to the agent's (x, y) position.

In these experiments, the world is a $(0, 1)^2$ room with a wall with vertical width 0.05 dividing the center. 0.05×0.05 charts were placed uniformly across the environment, with adjacent charts overlapping by 0.02. The local model on each chart is a single scalar value for each action. This chart allocation is similar to a tabular discretization, except that adjacent cells overlap.

The chart equivalence classes are precomputed; in the next section we will discuss methods for computing equivalence classes online. The only class of charts we consider equivalent in these experiments are

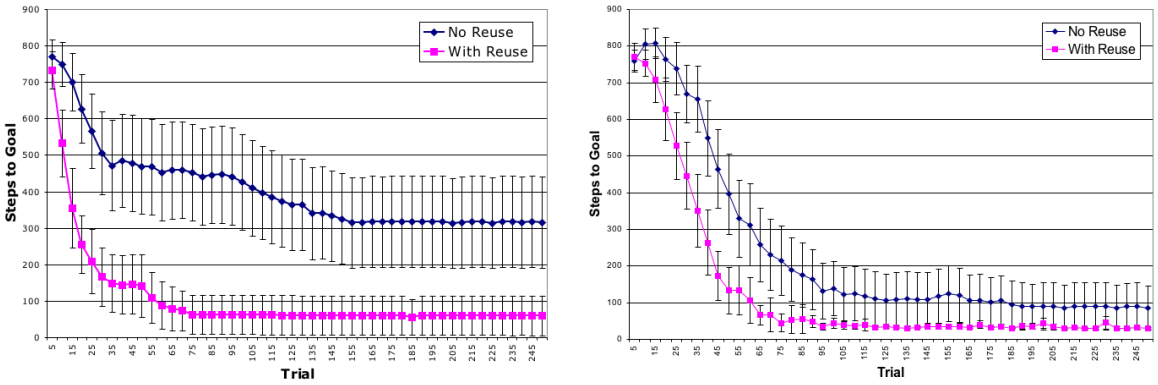


Figure 2: Comparison of performance with and without experience reuse on 2-d (left) and 3-d (right) navigation domains. Results are the mean of fifty experiments; 95% confidence intervals are shown.

the open space charts, i.e., charts that do not intersect any walls. There were 1225 total charts in the atlas, 979 of which were "open space" charts.

Experiments consist of a series of 250 trials. Each trial is terminated once the agent reached the goal or after 1000 steps. The agent starts each trial from a random point in a small disc, as shown in Figure 4. We compare the performance of an agent using experience reuse to one which does not. Value backups for experiences observed along the agent's trajectory were performed according to the SARSA update rule [14], while synthesized experiences were backed up using the Q-learning update from Equation 1.

Figure 4 provides the results of our experiments. Experience reuse results in significantly better performance than is observed when experiences are not reused. Two of the experiments with experience reuse never found the goal; with these two outliers removed the performance in later trials improves from about 60 steps to to about 20 steps to reach the goal. In the no-reuse case, performance reached an expected 316 steps to goal about trial 250; 15 experiments without experience reuse failed to learn a good policy.

In our next set of experiments we increase the complexity of the navigational domain by incorporating the robot pose into the world state. The agent can drive forward or backward, or turn left or right $frac\pi 4$ radians then drive forward. The robot moves a maximum distance of 0.03 on each time step. The goal and start positions are changed in order to increase the chance that the agent will find the goal. The goal region is centered about $x = 0.5$, $y = 0.8$, and starting points are selected from a disc about $(0.2, 0.2, \frac{pi}{2})$.

The world is covered with $0.08 \times 0.08 \times \frac{5\pi}{18}$ charts. Adjacent charts overlap by an extent of 0.04 in the x and y directions, and $\frac{\pi}{10}$ radians along the θ-axis. The total number of charts in the covering is 5013, 3437 of which do not overlap walls.

In these experiments we restrict ourselves again to experience reuse by translation of sample endpoints only. Since two charts open space with disjoint θ intervals will not observe transitions with the same orientation between initial and end point, we limited experience reuse to open space charts with the same θ interval. We could clearly enhance the amount of reuse by taking into account that some charts many charts in this domain are related by rotation as well as translation.

As the results in Figure 4 show, even this relatively naive implementation of reuse continues to significantly outperform the no-reuse case. The performance of each is much closer in the later trials in these experiments. It is likely that admitting more extensive reuse would increase this margin.

5 Detecting similar charts

The results with experience reuse shown in the previous section illustrates the merit of this approach. However, in practice it is unreasonable to expect that the agent has access to enough information about the world to determine equivalence classes *a priori*. The desired solution is to determine these equivalence classes based on the agent's interaction with the environment. In order to achieve this goal, we present a chart similarity measure based on comparison of vector fields. In deterministic domains with continuous

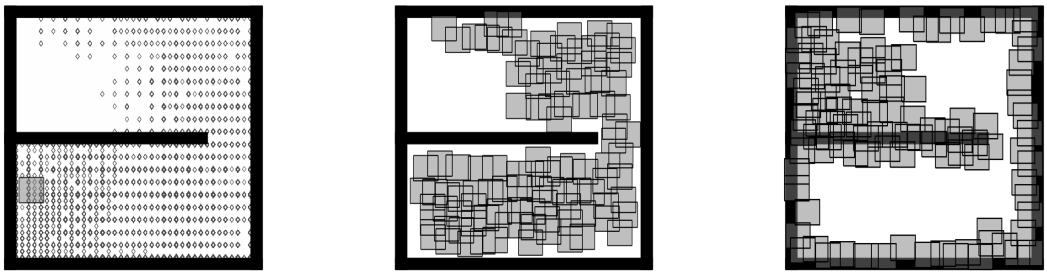


Figure 3: Automatically detected charts similar to a selected chart. Left: the collection of stored samples' initial points, and the selected chart. Center: Charts similar to the selected chart. Right: Charts dissimilar to the selected chart.

dynamics, the transition function $T(s, a)$ for any choice of a induces a vector field on S.

Recall that the divergence $\nabla F(\mathbf{x})$ for vector field $F(\mathbf{x})$ is $\sum_{i=1}^{d} \frac{\partial F(\mathbf{x})}{\partial x_i}$. Based on this notion, one measure of difference between charts φ_i and φ_j related by ψ_{ij} is the squared difference in divergences, where A is the set of actions:

$$\int_{a \in A} \int_{s \in \mathrm{dom}(\varphi_i)} [\nabla T(s, a) - \nabla T(\psi_{ij}(s), a)]^2 ds da \tag{2}$$

This gives us a measure that is rotation invariant. If we were to consider a more realistic version of the navigation environment in the previous section, the robot might have available actions "turn right", "turn left", and "go forward", and state will be the pose $\rho = (x, y, \theta)$. In this case, two open space charts may be related by a rotation as well as translation. For now we restrict ourselves to similar charts that are related by translation, so we must amend our similarity measure to avoid rotational invariance.

$$\int_{a \in A} \int_{s \in \mathrm{dom}(\varphi_i)} (\nabla [T(s, a) - T(\psi_{ij}(s), a)])^2 ds da \tag{3}$$

In practice we do not have access to the system dynamics. We do have a set of transition samples $\Xi_\varphi = \{\xi = (s_\xi, a_\xi, r_\xi, s'_\xi) : s \in \mathrm{dom}(\varphi)\}$. At each sample transition $\xi = (s_\xi, a_\xi, r_\xi, s'_\xi)$, we can compute the divergence $T(s_\xi, a_\xi)$ directly in terms of $s'_\xi - s_\xi$. Recasting Equation 3 for the sampled case, we get

$$D(\varphi_i, \varphi_j) = \sum_{\xi \in \Xi_{\varphi_i}} (\nabla [T(s_\xi, a_\xi) - T(\psi_{ij}(s_\xi), a_\xi)])^2 + \sum_{\xi \in \Xi_{\varphi_j}} (\nabla [T(\psi_{ij}^{-1}(s_\xi), a_\xi) - T(s_\xi, a_\xi)])^2 \tag{4}$$

This does present one problem, however; if we have a sample $\xi \in \Xi_{\varphi_i}$, it is likely that we do not have a sample rooted at $\psi_{ij}(s_\xi)$, much less with the same action a_ξ. In this case, some approximation is needed; for this paper we use the sample $\psi' \in \Xi_{\varphi_j}$ such that $a_{\xi'} = a_\xi$ that minimizes the distance $d(s_\xi, \psi_{ij}(s_{\xi'}))$. More sophisticated approximations than this nearest neighbor approach are possible, but this straightforward approach appears to be sufficient in practice.

Figure 5 shows the set of charts similar to a selected chart in a manifold constructed from randomly-placed fixed-size charts. The sample sets used for each chart were obtained by executing 30 800-step random walks from a fixed starting point. In order to obtain an unbiased sample, the resulting sample sets were subsampled on each chart by generating a set of 30 points uniformly at random on each; the nearest sample to each point was retained. This was performed once for each action on each chart. Charts φ_i and φ_j were considered similar if $D(\varphi_i, \varphi_j) < \frac{\sum_{\varphi_k \in \Phi} D(\varphi_i, \varphi_k)}{2|\Phi|}$.

The figure demonstrates that, when the charts are well-sampled, the set of similar charts closely matches our intuition. However, errors may occur when charts are under-sampled. For instance, in Figure fig:equivs,

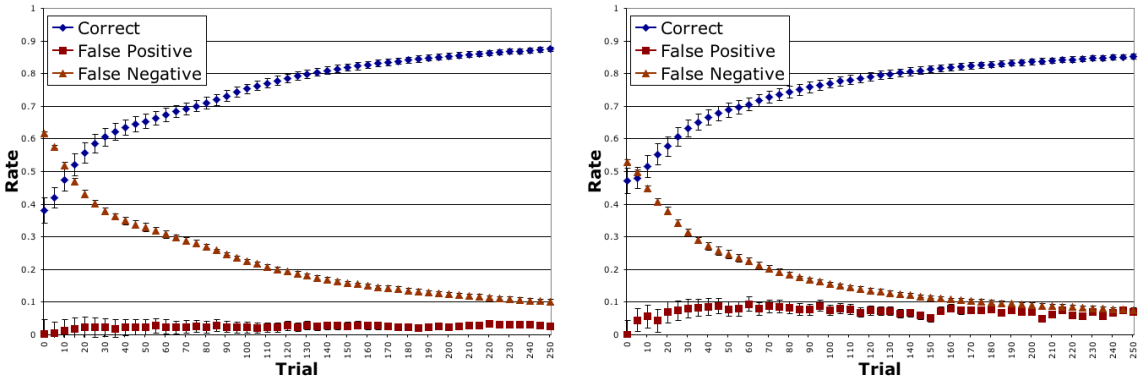


Figure 4: Classification performance on two example chart classes with respect to a hand-coded algorithm. Chart classes are open space (left) and bounded on the left by a wall (right).

we apply our similarity measure to an open space chart. However, some charts overlapping the upper wall are found to be similar. This may be due to the fact that any observed samples with action "up" were from an initial state near the bottom of the chart. Since such an action would not carry the agent all of the way to the wall, the chart appears to be open based on the observed samples. As more samples are collected, these misclassification errors should become less frequent.

Figure 5 compares the classification performance of this similarity metric against a hand-coded algorithm for determining equivalence classes. Performance is shown for two chart types in the 2-dimensional navigation domain – open space charts, and charts bounded on the left by a wall. Samples were collected by running a series of random walk trials. In each trial, the agent is initially placed in a small disc in the lower right-hand corner. The agent then performs a 1000-step random walk by selecting random actions. At the end of each trial, the sample set on each chart was subsampled as described above, so that as many as thirty samples per chart per action are retained.

The hand-coded algorithm for selecting equivalence classes groups charts that overlap a wall in the same way, i.e., do not intersect a wall, or all charts that are bounded on the left and above by walls, etc. This gives defines all of the obvious equivalence classes under translation. We do not expect the divergence-based similarity measure to achieve perfect accuracy with respect to the hand-coded equivalence measure, however. As was mentioned above, two charts that do not overlap a wall are still not necessarily equivalent with respect to the transition function, as it may be possible to take an action that results in the agent hitting a wall from one of the charts but not the other. The divergence measure is sensitive to this distinction, while the hand-coded policy is not.

6 Conclusions

The results shown in Section 4 demonstrate that a substantial increase in learning speed can be obtained from even a straightforward approach to experience reuse. Admitting a richer set of equivalence classes, such as "wall" and "corner" classes would present more opportunities for experience synthesis. Of course, as more charts are eligible for reuse, the cost of reuse increases. We have not discussed this cost up to this point, as we expect the largest expense to be acquisition of the training data from the world.

A potential improvement that we have not considered in this work is the order in which chart models are updated based on synthesized experiences. In the context of goal-based reinforcement learning, there is a natural extension to the manifold representation that will allow appropriate ordering. By constructing a graph with vertices corresponding to charts, with edges between overlapping charts, we can perform updates on synthesized data backwards from the goal in a breadth-first fashion. This should speed up the propagation of value back from charts containing the goal, and is similar to prioritized sweeping [9].

One limitation in our current experiments was the requirement that experience reuse only occurs for samples whose start and endpoint share a chart. In Section 5 we described one approach to determining

more robust chart similarity that will allow the reuse of samples that do not satisfy this constraint. By admitting a more complex relationship between samples on charts, we are likely to further increase the potential for experience reuse, but will likely require a more complex similarity measure.

Finally, we have alluded to the possibility of allowing more general sample transformations between charts. Thus far we have limited reuse to charts that are related by translation. We have seen a domain, the 3-dimensional navigation domain, charts are naturally related by rotation. Extending our methods to handle more general transforms will lead to more efficient reuse of experiences.

References

[1] J. S. Albus. A new approach to manipulator control: The cerebellar model articulation controller (CMAC). *Journal of Dynamic Systems, Measurement and Control*, pages 220–227, 1975.

[2] C. Boutilier, R. Dearden, and M. Goldszmidt. Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 121(1-2):49–107, 2000.

[3] J. A. Boyan and A. W. Moore. Generalization in reinforcement learning: Safely approximating the value function. In G. Tesauro, D. S. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 369–376, 1995.

[4] R. Glaubius and W. D. Smart. Manifold representations for value-function approximation. In *Working Notes of the Workshop on Markov Decision Processes, AAAI 2004*, San Jose, California, USA, 2004.

[5] R. Glaubius and W. D. Smart. Manifold representations for continuous-state reinforcement learning. Technical Report WUCSE-2005-19, Department of Computer Science and Engineering, Washington University in St. Louis, 2005.

[6] C. M. Grimm and J. F. Hughes. Modeling surfaces of arbitrary topology using manifolds. *Computer Graphics*, 29(2), 1995. Proceedings of SIGGRAPH '95.

[7] T. Lane and A. Wilson. Toward a topological theory of relational reinforcement learning for navigation tasks. In *Proceedings of the Eighteenth International Florida Artificial Intelligence Research Society Conference (FLAIRS-2005)*, 2005.

[8] A. McGovern, D. Precup, B. Ravindran, S. Singh, and R. S. Sutton. Hierarchical optimal control of MDPs. In *Proceedings of the Tenth Yale Workshop on Adaptive and Learning Systems*, pages 186–191, 1998.

[9] A. Moore and C. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less real time. *Machine Learning*, 13:103–130, 1993.

[10] R. Munos and A. Moore. Variable resolution discretization for high-accuracy solutions of optimal control problems. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI '99)*, pages 1348–1355, 1999.

[11] W. D. Smart and L. P. Kaelbling. Practical reinforcement learning in continuous spaces. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML-2000)*, pages 903–910, 2000.

[12] P. Stone and R. S. Sutton. Scaling reinforcement learning toward robocup soccer. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML-2001)*, 2001.

[13] R. S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 1038–1044, 1996.

[14] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. Adaptive Computations and Machine Learning. The MIT Press, Cambridge, MA, 1998.

[15] G. J. Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8(3/4):257–277, 1992.

[16] S. Thrun and A. Schwartz. Finding structure in reinforcement learning. In *Advances in Neural Information Processing Systems 7*. MIT Press., 1995.

[17] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.

Symbolic Focused Dynamic Programming for Planning under Uncertainty

P. Fabiani and **F. Teichteil-Königsbuch**
ONERA/DCSD, 2 Avenue Édouard-Belin 31055 Toulouse, FRANCE
(patrick.fabiani,florent.teichteil)@cert.fr

Abstract

This research addresses decision-theoretic planning with structured MDPs for the *ReSSAC* autonomous exploration rotorcraft. We developped symbolic dynamic programming algorithms for planning in large state spaces. The requirements are to control the optimization process, to enable reasoning at a higher level of abstraction and to deal with both uncertainty and non-Markovian historic dependent rewards or goals. Decomposition and factorization techniques are applied to exploration-like Markov Decision Processes (MDPs), which are structured in two components. A graph of enumerated states represents the navigation component of the problem, as in gridworld MDPs. A set of state variables describes, in a compact and implicit way, the other features of the problem, including the intermediate goals to be achieved in sequence. A family of such problems, of small and large sizes, is used to compare Heuristic Search Dynamic Programming algorithms. The solution of an academic gridworld exploration instance is explained. We propose an original *Symbolic Focused Dynamic Programming* scheme *SFDP* in which the optimization time and the solution quality are controlled by planning for a partial subset of selected planning goals. Experimental results are presented and optimality issues discussed.

1 Introduction

This paper deals with planning under uncertainty for the *ReSSAC* (http://www.cert.fr/dcsd/RESSAC) autonomous exploration rotorcraft. A number of preliminary computations are necessary before addressing the planning problem. First the aircraft must be able of autonomous flight and navigation. Next, navigation waypoints must be automatically generated from an *a priori* map like on Figure 1.a. Crests and valleys can be computed on the *a priori* numerical terrain model, thus leading to a partition into interest regions. Waypoints are remarquable nodes of this partition.

Transition trajectories between the way-points are computed, optimizing time, energy consumption or perception capabilities, between the way-points. A stochastic exploration mission comprises both a problem of navigation (a navigation graph and regions as in Figure 1) in a partially known environment, and a problem of online information acquisition.

Our algorithms are developped for planning in large state spaces, with repeated- or single-achievement goals, history-dependent rewards and costs. Several final, alternative or intermediate goals may be given to the agent. Some final goals, such as landing in a safe area, must be achieved in all possible plans. Some goals, such as exploring or searching a region, are the pre-conditions to be achieved before seeking to realize further goals. Some goals can only be achieved once, and other rewards can be claimed repeatedly (e.g. refueling actions). A set of ordering constraints between the goals $O_j \leq O_{j'}$ can be imposed: e.g. the agent must take information in region R_k and transmit it to its ground control center before proceeding with its navigation to the neighbouring regions. Goals may be optional: the agent may reach its final goal O_f either via the prior achievement of O_1, or via O_2. Some state transition probabilities, rewards and goals are history-dependent: for instance the energy consumption of a rotorcraft never pauses during flight and its energy level A decreases over time.

An insufficient energy level can force the aircraft to *Abort* its mission and return to its base, or to land on an emergency or security crash base, which is associated with some possible penalties. Other features of the probleme are partially observable, such as the true terrain elevation map, or the wind : for instance, an estimation of both the ground relative height and speed must be made during the approach phase and memorized for a safe transition from cruise flight to landing. This can be dealt with thanks to additional variables, as it is well-known that taking into account part of the state history can compensate for partial observability. Such non-Markovian problems are commonly encountered for instance in autonomous robotic applications, but not only. Planning in non-Markovian domains is addressed in [1] by using a temporal logic in order to generate the required additional variables.

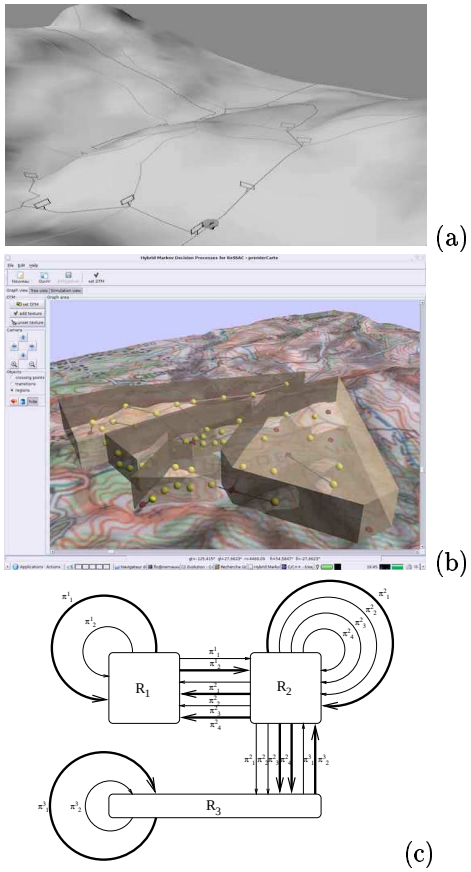


Figure 1: Navigation graph (a), navigation component of the planning problem (b) and its corresponding abstract decomposed MDP (c)

In the following, we propose an original *Symbolic Focused Dynamic Programming (SFDP)* heuristic search dynamic programming scheme. We first give a brief sketch of how our approach is related to recent work in the domain. We then describe our approach, starting with the combination of MDP decomposition and factorization techniques. We propose different algorithms implementing the *SFDP* scheme, some sub-optimal and some optimal. We present our current results of comparisons with other algorithms, and discuss optimality issues. Gridworld exploration MDPs are used for comparisons because they make it easier to generate large scale problems as in [3]. Simple problems are used in order to explain the approach and more complex ones in order to demonstrate the scalability.

2 Sketch of our approach

Our work more generally applies to Decision-Theoretic planning in large discrete state spaces, with repeated- or single-achievement goals, history-dependent rewards and costs. Such problems are generally described in factored form by using *Dynamic Bayes Nets (DBNs)* [6], for the description of probabilistic dependences between state variables as shown in Figure 2. For instance, our approach and algorithms are applicable to all the PDDL 2 problems of the probabilistic track of ICAPS'04 planning competition in which we participated. The efficiency of factored representations is well-known [5]: in order to reason at a higher level of abstraction as in [8], to deal with non-Markovian processes as in [1] or to introduce a state space hierarchical decomposition into sub-regions as in [7]. Otherwise, classical Markov Decision Processes (MDPs) [15] solution algorithms are based on an explicitly enumerated and unstructured state space. The size of the state space is an exponential function of the number of features that describe the problem. The state enumeration itself may rapidly become untractable for realistic problems. More generally [4] provide an extensive discussion on complexity and modeling issues. Other important contributions, such as the SPUDD library [13], have improved the efficiency of factored MDP solution algorithms, using decision diagrams, borrowed from the Model Checking community. The influence of the modeling structure in stochastic planning , and especially the combination of both MDP decomposition and factorization techniques has been thoroughly studied and is presented with more details in [16].

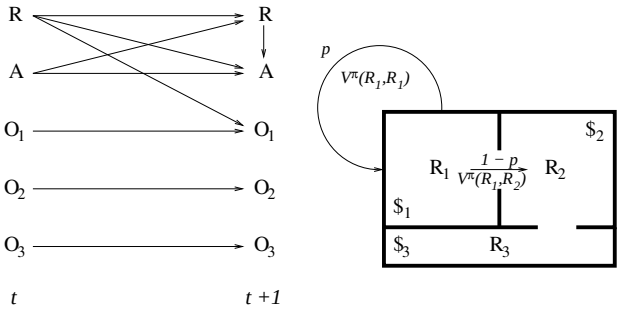


Figure 2: Action networks and transitions of the abstract MDP with the local policies of the region R_1

We furthermore propose, for autonomous robotic applications, an original way of controlling the planning optimization process, by enforcing or relaxing constraints on the specific mission goals to be achieved by the agent, thus making it possible to reach different trade-offs between a better solution quality or a better optimization time. *SFDP* borrows ideas from both symbolic *sLAO** [9] and *sRTDP* [10] algorithms, but adds the original way of controlling the optimization process by goal enforcement or relaxation. Otherwise, *SFDP* conforms, like *sLAO**, with a two-phased scheme of *planning space expansion*-and-*optimization*. This is not to be confused with the general *FIND-and-REVISE* programming scheme proposed in [2]. The *planning space expansion* is based on a reachability analysis using the current policy and a set of *planning goals*. This *expansion* phase is an deterministic implementation of the *policy execution step* applied in *sRTDP* [10]. It can be implemented exactly or approximately, for efficiency gains. In the exact implementation, every trajectory ends on a goal state.

In the approximate reachibility analysis, an earlier and easier stopping condition is added, thus leading to a sub-optimal algorithm. The *optimization* stage is a dynamic programming phase applied whithin the previously expanded planning space: it is the *expansion* phase that actually enables to *focus more or less* the search and thus to control the optimization process.

3 Decomposition

The decomposition of an MDP is based on a state space partition Π into non empty distinct *regions*. Enumerated states are thus grouped into weakly coupled regions, i.e. with fewer transitions between the exit and entrance states of the regions. An abstract MDP [7] is then constructed, on the basis of *macro-states* resulting from the decomposition into regions. For each region r, $Sper(r)$ is the set of the exit states of the region r :

$$Sper(r) = \{ s \in S - r \ / \ \exists \ s' \in r \ , \ \exists \ a \in A \ , \ T(s', a, s) \neq 0 \}$$

We found two main possible options for the definition of macro-states. In [12], macro-states are proposed to be the exit states of the regions, i.e. the state space of the abstract MDP is $\bigcup_{r \in \Pi} Sper(r)$. In [7], macro-states are the aggregate states of the regions. The first model leads to possibly sub-optimal policies, because it only considers strategies leading from a macro-state to a different macro-state. This model does not allow a local strategy to try and reach a local sub-goal by staying within the same macro-states (absorbing macro-states are not possible). This aspect of the problem lead us to choose the second model for our exploration problem (like in Figure 1). More generally, an abstract MDP [7] is a tuple $\langle S', \mathcal{A}', \mathcal{T}', \mathcal{R}' \rangle$. The actions of the abstract MDP are then the local policies generated in each region and the transitions in the abstract MDP are computed from these local policies:

- $S' = \bigcup_{r \in \Pi} r$,
- $\mathcal{A}' = \bigcup_{r \in \Pi} \{ \pi^r_1, \ldots \pi^r_{m_r} \}$,
- $T'(r, \pi^r_j, r')$ and $R'(r, \pi^r_j, r')$ depend on a *discounted macro-transition model*.

Though our decomposition model is borrowed from [7], our decomposition algorithm is based on the techniques proposed by [12] because they are more adapted to factored resolution algorithms. As a result, in our implementation, the macro-probability of transition and the macro-reward from r to r' applying macro-action π^r_j are as follows:

- $T(r, \pi^r_j, r') = \frac{1}{|r|} \sum_{s' \in Sper(r) \cap r'} \sum_{s \in r} T(s, \pi^r_j, s')$
- $R(r, \pi^r_j, r') = \frac{1}{|r|} \sum_{s' \in Sper(r) \cap r'} \sum_{s \in r} T(s, \pi^r_j, s') \cdot R(s, \pi^r_j, s')$

$T(s, \pi^r_j, s')$ and $R(s, \pi^r_j, s')$ are iteratively computed via a Gauss-Seidel value iteration algorithm [16]. As a result, the optimality theorems in [12] still apply in our case with respect to local policies. Our local MDPs $\langle S', A', T', R' \rangle^{(\lambda(s))_{s \in Sper(r)}}$ are defined as follows:

- $S' = r \cup Sper(r) \cup \{\alpha\}$, where α is an absorbing state,
- $A' = A$,
- $T(s, a, s') = \begin{cases} T(s, a, s') & \text{if } (s, s') \in r \times (r \cup Sper(r)) \\ 1 & \text{if } (s, s') \in (Sper(r) \cup \{\alpha\}) \times \{\alpha\} \end{cases}$
- $R(s, a, s') = \begin{cases} R(s, a, s') & \text{if} (s, s') \in r \times (r \cup Sper(r)) \\ \lambda(s) & \text{if} (s, s') \in Sper(r) \times \{\alpha\} \\ 0 & \text{if} (s, s') \in \{\alpha\}^2 \end{cases}$

For each combination of peripheral values, there is an optimal local policy. However, a same local policy can be optimal with different combinations of peripheral values. For each local MDP we generate the minimal set of useful local policies, in the sense that at least one local policy per region should match the global optimal policy restricted to this region. The decomposition

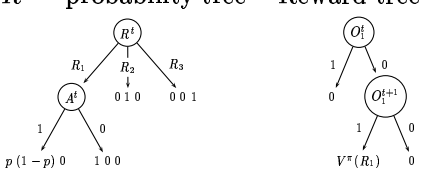


Figure 3: R^{t+1} probability and reward trees (from R_1)

of the problem is motivated by optimization time concerns. Work by [16] propose to factorize such problems through a hierarchical decomposition into an abstract factored MDP, and show the benefits of it in terms of efficiency. [12] or [14] have proposed two candidate algorithms for the computation of local policies during the MDP decomposition step. According to [16] the latter approach (LP by [14]) based on linear programming, is the one that offers the better performances and flexibility. In order to take into account the fact that rewards can only be obtained once, we have to adapt R. Parr [14] algorithm to our problem. We need to optimize the regions local policies conditionally to the status of the goals of the region: in practice this limits greatly the number of cases since the combinatorics is splitted into the regions. In our simple example, we only have one goal per region, which leads to optimize 2^1 sets of *conditional local policies* per region : one if the local *goal* has not been achieved yet by the agent, and one if it has already been. The direct benefit driven from decomposition comes from the fact that if there are k regions and one goal per region, only $2k$ local policies are computed. Without decomposition 2^k global policies should be optimized. After the decomposition phase, the navigation graph component of the gridworld exploration problem is splitted into *macro-states*, each one corresponding to a sub-region (see Figure 1.c), and combined with the other orthogonal state variables. The resulting abstract MDP is in a factored form and may be represented by Dynamic Bayes Nets as in Figure 2. The state variable R stands for the region, O_1, O_2 and O_3 stand for the *goals*, and A for the agent energy level. For simplicity, we assumed a binary energy level with constant consumption over the

regions, the function f giving the probability of "loosing the minimal energy level" between two time steps: $f(R_i, R_j, \pi) = [\ 0.65\ 0.35\ ]$ for all i,j and π.

4 The good use of Decision Diagrams

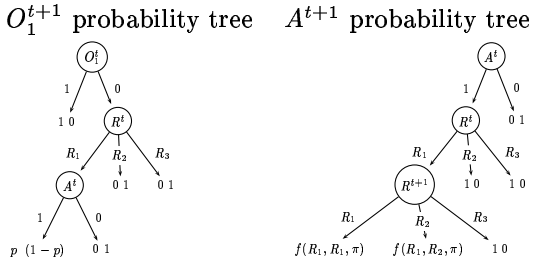


Figure 4: O_1^{t+1} and A^{t+1} probability trees (from R_1)

Our concern about encoding efficiency is related to the optimization time issue in our research context. In DBNs, the transition probabilities and rewards are represented by *Conditional Probability Tables*, i.e. large matrices, one for every post-action variables. However, these probability and reward tables are sparse in most problems and can be encoded in "decision trees" as in [5], or as "algebraic decision diagrams" (ADDs) as in [13]. Figures 3 and 4 respectively show instances of transition probability trees (vectors at the leafs) and transition rewards trees for the macro-actions of our instance of abstract MDP. For each post-action variable state, every leaf of the probability tree stores a list containing the probabilities to obtain every possible value x_i^{t+1} of this variable, knowing the values of the other variables x_i^t, x_j^t, x_j^{t+1} along the path $x_i^t \wedge (\wedge_{j \neq i} (x_j^t \wedge x_j^{t+1}))$ from the root of the tree to the considered leaf. The Figure 3 expresses the fact that, in order to obtain the reward associated with a given goal, this goal must not be already achieved and the agent must first reach the corresponding region with a sufficient energy level. Goals in the other regions cannot be achieved from "outside" and the corresponding decision tree is equivalent to a *NO-OP*. ADDs offer the additional advantage that nodes of a same value are merged, thus leading to a graph (see Figure 5) instead of a tree. After testing and comparing Decision Trees and ADDs implementations of our policy and value iteration algorithms, the conclusion was that ADDs offer a much more efficient encoding, even if they are limited to use binary conditions: some state or action variables may have to be splitted into several binary variables to fit in the model. As a matter of fact, state variables with large range of values considerably increase the computation time necessary in order to solve factored MDPs. This is either due to the width of the corresponding trees when using Decision Trees, or otherwise due to the number of binary variables required when using ADDs. It is moreover noticeable that position or navigation variables typically take a large number of possible values. This is another way of getting convinced that it is a good idea to decompose the navigation component in our exploration problem into fewer more abstract aggregate regions.

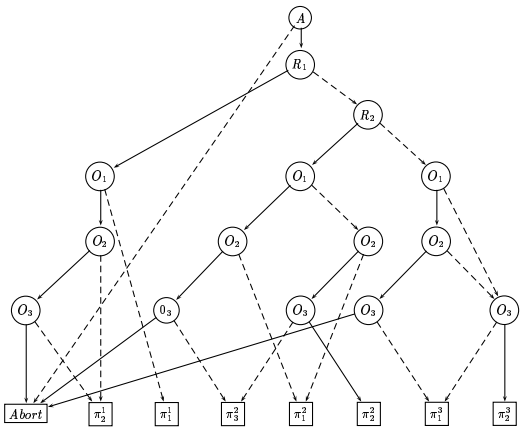


Figure 5: Optimal policy ADD (MDP of Figure 2)

Solution algorithms for factored MDPs using ADDs are based on simple algebraic and logical operations such as AND, OR, PRODUCT, SUM, etc. Some technicalities specific to ADDs are explained in [13], especially with respect to the implementation of *value iteration* in *SPUDD*, on which our own *value iteration* algorithms are based. The development of the *policy iteration* versions of the compared algorithms demanded to apply some similar technicalities: in order to improve the efficiency of our algorithms, we apply for each action a mask BDD on the complete action diagram ADD and the reward ADD of the action, representing the states where the action can be applied. Furthermore, BDDs and ADDs are restricted to the current reachable state space in order to save memory and to speed up the ADD operations.

Init
$R_0 \leftarrow Reachable(I, \mathcal{A}, G)$
$\Pi_0 \leftarrow ShortestStochasticPath(R_0 \rightarrow G)$
$S_0 \leftarrow FilterStates(R_0, P(s) < \epsilon \cdot P(I))$
$\quad \Longrightarrow (\Pi_0, S_0)$

$k \leftarrow 0$
repeat
$\quad S_{k+1} \leftarrow Reachable(S_k, \Pi_k,\ 1\ step\ lookahead)$
$\quad PolicyIteration(\Pi_k, S_{k+1})$
$\quad k \leftarrow k + 1$
until convergence over S_k

Figure 6: *sLAO* Policy Iteration*

5 Symbolic Heuristic Search Dynamic Programming

The *sLAO** algorithm [9] is the symbolic version of *LAO** [11]. We implemented the *sLAO** policy iteration algorithm shown in Figure 6, where the generic function *Reachable* is applied from S_k at each iteration, using actions from the current policy Π_k, with 1 *step lookahead*.

Note that S_k in $sLAO^*$ is always supposed to grow, which in that context gives $sLAO^*$ the same guarantee of optimality as $sLAO^*$ [11] : when the algorithm converges, every trajectory followed in S_k by applying the current policy is ending on a goal state. Symbolic heuristic search dynamic programming algorithms seem to conform to a common two-phased scheme, shown in Figure 7:

- a first **planning space expansion** and **heuristic** computation phase,
- a subsequent **dynamic programming** phase.

It constitutes our common algorithmic basis for development, testing and comparison of different heuristic search dynamic programming algorithms that conform to it such as $sLAO^*$ [9] and $sRTDP$ [10].

Init
$R_0 \leftarrow Reachable(I, \mathcal{A}, G)$
$\Pi_0 \leftarrow ShortestStochasticPath(R_0 \rightarrow G)$
$S_0 \leftarrow FilterStates(R_0, P(s) < \epsilon \cdot P(I))$
$\Longrightarrow (\Pi_0, V_0, S_0)$

$k \leftarrow 0$
repeat
$S_{k+1} \leftarrow Reachable(I, S_k, \Pi_k, G)$
$DynamicProgramming(\Pi_k, V_k, S_{k+1})$
$k \leftarrow k + 1$
until convergence over S_k

Figure 7: Heuristic Search Dynamic Programming Scheme

One strong idea is simply that the algorithm cannot apply dynamic programming on the full state space because this is untractable. The working space is thus carefully extended at each iteration, keeping it small but still sweeping the full state space. **Heuristic** computations, such as the proposed *shortest stochastic path analysis* intervene in this first phase, essentially to provide an admissible estimation of the value function or equivalently, a default policy on the border of the planning space. The "planning space expansion" phase enables to control the optimization process. $sLAO^*$ incrementally augments its working space until the expanded state spaces and the policy converge. We call $Reachable(I, \Pi_A, Stop)$ a function that takes as inputs the sets of initial and goal states I and G, uses the set of applicable actions $\Pi_A \subset \mathcal{A}$ (Π_A can be a policy or $\mathcal{A}$ itself) and computes the set R_0 of all the states that are reachable from I with successive applications of deterministic actions in $\mathcal{A}$ in an iterative loop that stops as soon as the $Stop$ condition is reached : e.g. $Stop$ can be $G \subset R_0$ or 1 *steplookahead*. The actions are made deterministic by setting the maximum transition probability to 1 and the other one to 0, which enables us to convert the ADDs into BDDs (Binary Decision Diagrams) that are more efficient. The idea here is that the planning space expansion is controlled via the $Stop$ condition, linked to the achievement of specific planning goals.

At this stage, we can at the same time compute an initial heuristic policy (or value function) and reduce – if possible – the initial reachable state space. We call $ShortestStochasticPath(R_0 \rightarrow G)$ a function that takes R_0 and G as inputs and computes a shortest stochastic path from every state in R_0, using stochastic actions from $\mathcal{A}$ without their rewards. Better simplification schemes should certainly be studied, but this heuristic seems efficient in many problems, such as navigation grid MDPs in [11] and in [3]. We call $FilterStates(R_0, P(s) < \epsilon \cdot P(I))$ a filtering function that filters the states that have a very low probability of reachability when the non-deterministic actions are applied along the shortest path trajectories. Low probability of reachability is assessed compared to the probability of the initial states. Stochastic reachability filtering seems very comparable in its results, with the effect of random sampling in $LRTDP$ [3].

During that stage the solution of the MDP is optimized on the current reachable state space . The expansion of the current reachable state space results from the function *Reachable*, applied either from I (RTDP) or from the previous reachable state space ($sLAO^*$). The $DynamicProgramming$ function of the main algorithm (see Figure 7) can be $ValueIteration$ or $PolicyIteration$. We used the latter during the competition since it seems to be more original and sometimes more efficient than the former.

6 SFDP and sLAO*

Init
$R_0 \leftarrow Reachable(I, \mathcal{A}, G)$
$\Pi_0 \leftarrow ShortestStochasticPath(R_0 \rightarrow G)$
$S_0 \leftarrow FilterStates(R_0, P(s) < \epsilon \cdot P(I))$
$\Longrightarrow (\Pi_0, S_0)$

$k \leftarrow 0$
repeat
$S_{k+1} \leftarrow Reachable(I, \Pi_k, G)$
$PolicyIteration(\Pi_k, S_{k+1})$
$k \leftarrow k + 1$
until convergence over S_k

Figure 8: *SFDP Policy Iteration*

SFDP Policy Iteration is implemented by the algorithm shown in Figure 8: the generic function *Reachable* is applied from the initial state set I at each iteration, using actions from the current policy Π_k, until G is reached. As a matter of fact, the working space S_k of *SFDP* is absolutely not guaranteed to grow : on the contrary, *SFDP* has been designed to *focus* on coherent parts of the state space. As a consequence, *SFDP* will not give the optimal solution to the problem, rather the "shortest solution", unless *SFDP* is compelled to visit all the rewards of the state space because all the rewards of the problem have been given as *planning goals constraints* prior to the optimization.

7 Experimentations

We conducted our experiments on gridworld exploration problems inspired from the example shown in figure 1.b. The number of nodes of the navigation graph is 45x45=2025 and the total number of states grows exponentially in the number of additional state variables grows with the number of goals, regions in the problem (+1 for the energy level).

We present a comparison of six algorithms that have been implemented on the basis of the SPUDD/VI *value iteration* algorithm: 1.SPUDD/VI – 2.SPUDD/PI – 3.sLAO/VI – 4.sLAO/PI – 5.SFDP/VI – 6.SFDP/PI. Note that the algorithm number 4 participated in the ICAPS'04 competition but it was not as mature as today. We present results obtained with stochastic exploration-like problems because they are closer to our research projects than the competition problems. Yet, the complexity of such exploration problems is comparable with the ICAPS'04 probabilistic track competition problems. We have compared the solution quality of *SFDP* poli-

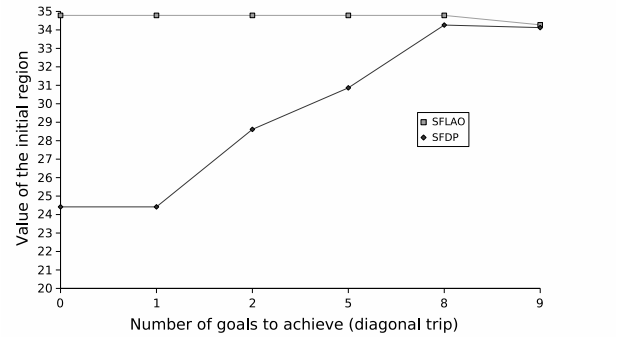


Figure 9: *SFDP* Solution quality (Value at starting point) compared with *sLAO** while increasing the number of planning goals to achieve

cies when a growing number of constraints are imposed on both algorithms concerning the planning goals. It appears that *SFDP* appears as much more sensitive to goal constraints than *sLAO**. Imposing on *SFDP* to achieve ALL the goals leads the algorithm to behave like *sLAO**, continuously extending its planning space without *focusing*, as long as the corresponding problems remains tractable. On the contrary, *sLAO** tends to try and reach all the rewards and goals of the problem even when it is not asked so. The corresponding computation time grows in proportions with the number of combinations of alternatives.

We have similarly compared the computation time for *sLAO** and *SFDP* on problems of growing complexity (varying the starting and goal points). The conclusion is that *SFDP*, still finds quite quickly (sub-optimal) solutions for the most complex problems that we could decompose in reasonable time (248s) and without swapping , which corresponds to the diagonal trip. By contrast, *sLAO** cannot give any answer after more than one hour on the fourth problem. Such comparison should be ana-

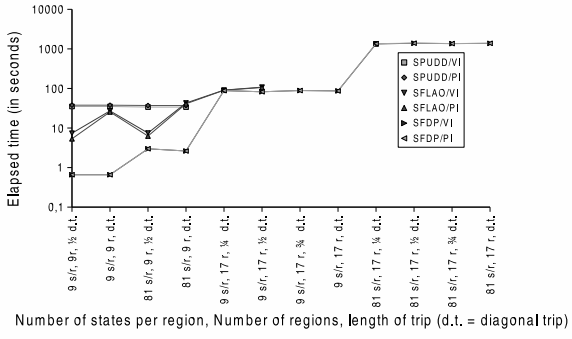


Figure 10: *SFDP* Solution time compared with *sLAO** and *SPUDD* for different starting points

lyzed carefully: *SFDP* cannot be considered as "better" nor "preferable" on the basis of this comparison. On the other hand, Figure 9 shows that it is possible to establish a quality response profile for *SFDP* on some classes of problems. The quick answer given by this algorithm could be reused in order to initiate an admissible heuristic policy, or value function for another algorithm.

8 Faster solution, weaker value

Following the previous ideas, another version of the *focused dynamic programming* scheme was developed by weakening the stopping condition of the *Reachable* function. The new stopping condition holds as soon as **at least one state is reached where the required goal conditions are achieved**. This new stopping condition is obviously weaker and the new algorithms, called *sfDP* and *sfLAO* still follow the schemes respectively presented in Figure 8 and 6, but with the new *Reachable* function. The *sfDP* and *sfLAO* algorithms are not optimal with the present stopping condition that is applied. They however show interesting properties in terms of incremental behavior, as shown in Figures 11 and 12.

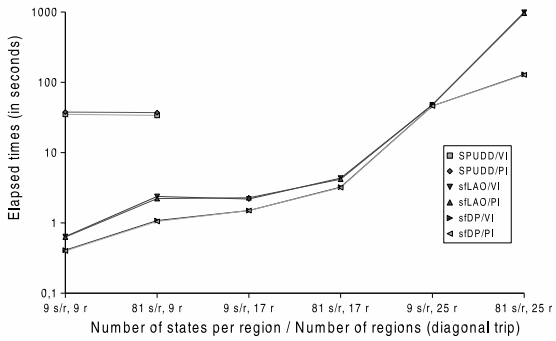


Figure 11: sfDP and sfLAO optimization time while increasing the problem size

Experimental results presented in Figure 13 show that *sfLAO* finds better solution than *sfDP*, with similar computation times. Interestingly enough, *sfLAO* still shows

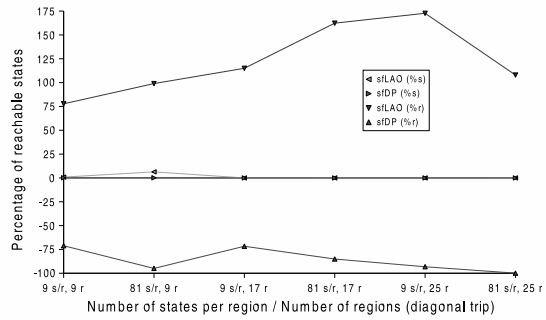


Figure 12: Percentage ($\%s$) of explored reachable states and variation ($\%r$) for sfDP and sfLAO

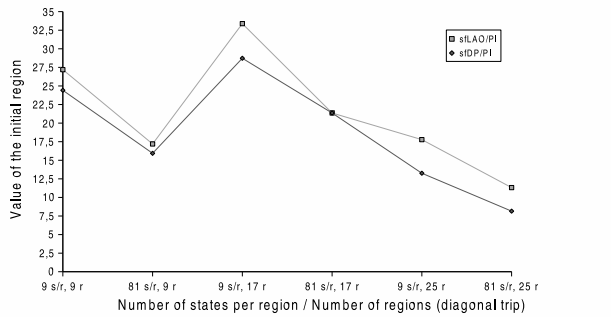


Figure 13: sfDP and sfLAO Solution Quality (Value at starting point) while increasing the problem size

the same behavior exhibited with *SFDP*: the solution quality grows with the number of goal conditions imposed up to the optimal solution. As a consequence, the computation time required in order to obtain the optimal solution is also a growing function and reaches the elapsed time obtained with *sLAO**, as shown in Figure 11. As a matter of fact, we used this idea to develop an incremental version of both *sfDP* and *sfLAO* algorithms, with respectively the *IsfDP* and *IsfLAO* algorithms that are described in Figures 14 and 15. Experimentations

$L_{sg} \longleftarrow$ List of subgoals to achieve
$S_0 \longleftarrow I$
$n \longleftarrow 1$
while L_{sg} non empty **do**
  $I_n \longleftarrow S_{n-1}$
  $G_n \longleftarrow$ head of L_{sg}
  $S_n \longleftarrow SFDP(I_n, G_n)$
  remove head of L_{sg}
  $n \longleftarrow n+1$
end while

Figure 14: ISFDP algorithm for on-line planning

shown in Figures 16 and 17 show that *IsfDP* clearly outperforms *IsfLAO*. Previous comparisons [16] between *LAO*-like* algorithms and *RTDP-like* algorithms have shown that *LAO** would be better when the problem topology is "open" and *RTDP-like* algorithms would be

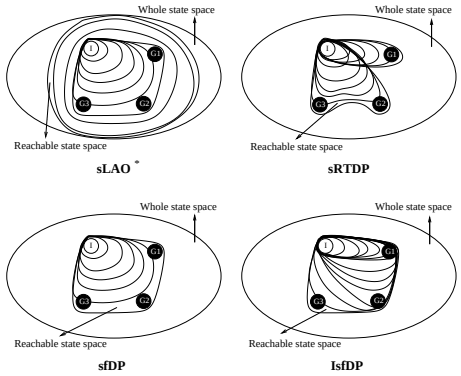


Figure 15: Reachable state spaces of sLAO*, sRTDP, sfDP and IsfDP

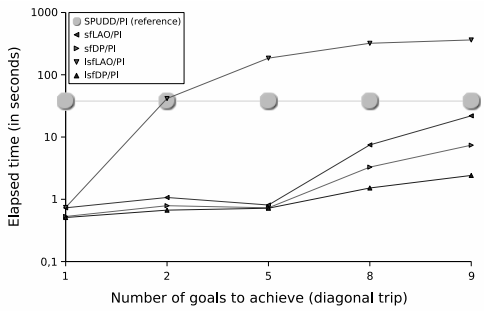


Figure 16: sfDP, IsfDP, sfLAO and IsfLAO optimization time compared with SPUDD while increasing the number of planning goals to achieve

more efficient in "corridors". Both heuristic schemes lead to limit the size of the explored state space before convergence. *SFDP* algorithms can combine both, as shown in figure 15 and rapidly improve the quality of the solution but does generally not lead to an optimal solution because of the earlier stopping condition in the reachability analysis. However, the proof of optimality of *LAO** [11] would still apply to *SFDP* if and only if : the terminal states of every trajectory in the state space resulting from the *expansion phase* ends in a goal state. This is verified **if and only if** we apply the stopping condition of *sfLAO* in the last *expansion phases* of the algorithm, which show that a true optimal algorithm can be derived from *IsfDP* with the following condition: *the expansion phase* stops when the reachable state stabilizes, independently of the reached or unreached goals.

9 Conclusion

We have presented our approach for planning under uncertainty for autonomous rotorcraft. From the point of view of real robotics implementations, we will also consider the influence of the initial phase of building of the navigation graph, since it is also possible to directly generate some of the local policies during this phase. This option could lead to efficiency gains.

We have proposed an original algorithmic scheme

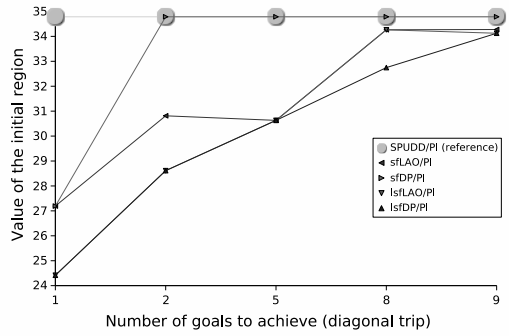


Figure 17: sfDP, IsfDP, sfLAO and IsfLAO Solution Quality (Value at starting point) compared with SPUDD while increasing the number of planning goals to achieve

SFDP for which the optimization time and the solution quality can be controlled through the definition of planning goals constraints. The design of *SFDP*, and the principles of the proposed underlying *focused dynamic programming* scheme, meet the challenges of planning under uncertainty in large state spaces for autonomous systems that rather need a current solution quite rapidly, and an optimal one if possible. Goal conditions can be adapted off-line or on-line, thus opening interesting directions for future work on decision under time and ressources constraints. This is particularly interesting in our application perspectives on autonomous aircraft. Among possible perspectives, we will consider carefully deriving an on-line version *OSFDP* that would adjust on-line the number of goal constraints to satisfy in response to the available time for finding a solution to the problem. We will also consider the coupling of *SFDP* with a higher level optimization controller in order to reuse the sub-optimal solution obtained with *SFDP* in a higher level optimization process. This was shown in the development of the incremental *IsfDP* algorithm, based on *sfDP*, an even faster, but weaker, version *sfDP* of the *focused dynamic programming* scheme. *sfDP* presented that quickly finds solution in larger problems. The incremental version *IsfDP* incrementally improves the current solution thanks to iterative calls of *sfDP* with an increasing list of planning subgoals to be taken into account. We have compared all these algorithms on a set of problems of different sizes. We will now develop an optimal version of the *focused dynamic programming* scheme, that would provide rapidly a current solution even on large state space problems, but would improve it up to the optimal solution as time is available.

References

[1] Fahiem Bacchus, Craig Boutilier, and Adam Grove. Structured solution methods for non-markovian decision processes. In *Proceedings 14th AAAI*, pages 112–117, Providence, RI, 1997.

[2] Blai Bonet and Hector Geffner. Faster heuristic search algorithms for planning with uncertainty and full feedback. In *IJCAI*, pages 1233–1238, Acapulco, Mexico, 2003.

[3] Blai Bonet and Hector Geffner. Labeled rtdp: Improving the convergence of real-time dynamic programming. In *Proceedings 13th ICAPS 2003*, pages 12–21, Trento, Italy, 2003.

[4] Craig Boutilier, Thomas Dean, and Steve Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *J.A.I.R.*, 11:1–94, 1999.

[5] Craig Boutilier, Richard Dearden, and Moises Goldszmidt. Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 121(1-2):49–107, 2000.

[6] Thomas Dean and Keiji Kanazawa**a** model for reasoning about persistence and causation.*Computational Intelligence*,5(3): 142–150, 1989.

[7] Thomas Dean and Shieu-Hong Lin. Decomposition techniques for planning in stochastic domains. In *Proceedings 14th IJCAI*, pages 1121–1129, San Francisco, CA, 1995.

[8] R. Dearden and C. Boutilier. Abstraction and approximate decision-theoretic planning. *Artificial Intelligence*, 89:219–283, January 1997.

[9] Zhengzhu Feng and Eric Hansen. Symbolic heuristic search for factored markov decision processes. In *Proceedings 18th AAAI*, pages 455–460, Edmonton, Alberta, Canada, 2002.

[10] Zhengzhu Feng, Eric A. Hansen, and Schlomo Zilberstein. Symbolic generalization for on-line planning. In *Proceedings 19th UAI*, pages 209–216, Acapulco, Mexico, 2003.

[11] Eric A. Hansen and Zilberstein Shlomo. Lao*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129:35–62, 2001.

[12] Milos Hauskrecht, Nicolas Meuleau, Leslie Pack Kaelbling, Thomas L. Dean, and Craig Boutilier. Hierarchical solution of markov decision processes using macro-actions. In *Proceedings 14th UAI*, pages 220–229, San Francisco, CA, 1998.

[13] Jesse Hoey, Robert St-Aubin, Alan Hu, and Craig Boutilier. Optimal and approximate stochastic planning using decision diagrams. Technical Report TR-2000-05, University of British Columbia, 10 2000.

[14] Ron Parr. Flexible decomposition algorithms for weakly coupled markov decision problems. In *Proceedings 14th UAI*, pages 422–430, San Francisco, CA, 1998.

[15] Martin L. Puterman. *Markov Decision Processes.* John Wiley & Sons, INC, 1994.

[16] Florent Teichteil-Königsbuch and Patrick Fabiani. Influence of modeling structure in probabilistic sequential decision problems. *RAIRO Operations Research*, to appear, 2005.