IMA Conference on Mathematics of Robotics
9 – 11 September 2015, St Anne's College, University of Oxford

An Embedded Implementation of Bayesian Network Robot Programming Methods

By Mark A. Post

Lecturer, Space Mechatronic Systems Technology Laboratory.
Department of Design, Manufacture and Engineering Management.
University of Strathclyde, Glasgow, United Kingdom.

Abstract

A wide variety of approaches exist for dealing with uncertainty in robotic reasoning, but relatively few consider the propagation of statistical information throughout an entire robotic system. The concept of Bayesian Robot Programming involves making decisions based on inference into probability distributions, but can be complex and difficult to implement due to the number of priors and random variables involved. In this work, we apply Bayesian network structures to a modified Bayesian programming paradigm to provide intuitive structure and simplify the programming process. The use of discrete random variables in the network can allow high inference speeds, and an efficient programming toolkit suitable for use on embedded platforms has been developed for use on mobile robots. A simple example of navigational reasoning for a small mobile robot is provided as an example of how such a network can be used for decisional programming.

1. Introduction

One of the most pressing problems in mobile robotics is that of how to quantify and propagate certainty and uncertainty measures when making decisions for control. It is desirable for the rover itself to be able to deal with uncertainties probabilistically, since this gives a robot the ability to appropriately handle unexpected and uncertain circumstances. Bayesian Networks (BN) are well-suited for handling uncertainty in cause-effect relations, and handle dependence/independence relationships well provided that the network is constructed using valid relational assumptions. Some drawbacks of this method are that the variables, events, and values available must be well-defined from the beginning, and the causal relationships and conditional probabilities must be available initially [Kjrulff, 2008].

We approach the problem of probabilistic robotics using the Bayesian Robot Programming (BRP) methodology developed by Lebeltel, Bessiere et al [Bessiere et al., 2000] [Lebeltel et al., 2000] [Lebeltel et al., 2004], which provides a quite comprehensive framework for robotic decision-making using inference. We add to this method by formally using Bayesian networks as a knowledge representation structure for programming, which adds clarity of representation, a practical structure for constructing joint distributions dynamically, and reliance on a proven probabilistic methodology. Also, the use of recursive inference in a Bayesian network avoids the need to manually partition and decompose large joint distributions, which greatly simplifies the programming process. Our approach to using Bayesian networks for robotic programming is detailed in the dissertation by [Post, 2014], and a summary of the approach is given here.

2. Bayesian Robot Programming

A Bayesian program has been defined by Lebeltel et al. as a group of probability distributions selected so as to allow control of a robot to perform tasks related to those distributions. A "Program" is constructed from a "Question" that is posed to a "Description". The "Description" in turn includes both "Data" represented by δ, and "Preliminary Knowledge represented by π. This "Preliminary Knowledge π consists of the pertinent random variables, their joint decomposition by the chain rule, and "Forms" representing the actual form of the distribution over a specific random variable, which can either be parametric forms such as Gaussian distributions with a given mean and standard deviation, or programs for obtaining the distribution based on inputs [Lebeltel et al., 2000].

It is assumed that the random variables used such as X are discrete with a countable number of values, and that a logical proposition of random variables $[X = x_i]$ is mutually exclusive such that $\forall i \neq j, \neg(X = x_i \wedge X = x_j)$ and exhaustive such that $\exists X, (X = x_i)$. All propositions represented by the random variable follow the Conjunction rule $\mathrm{P}(X, Y|\pi) = \mathrm{P}(X|\pi)\mathrm{P}(Y|X, \pi)$, the Normalization rule $\sum_X \mathrm{P}(X|\pi) = 1$, and the Marginalization rule $\sum_X \mathrm{P}(X, Y|\pi) = \mathrm{P}(Y|\pi)$ [Bessiere et al., 2000]. Rather than unstructured groups of variables, we apply these concepts to a Bayesian network of M random variables $\wp = X_1, X_2, \ldots, X_N \in \pi, \delta$, from which an arbitrary joint distribution can be computed using conjunctions. It is assumed that any conditional independence of random variables in π and δ (which must exist, though it was not explicitly mentioned by Lebeltel et al.) is represented appropriately by the Bayesian network, thus significantly simplifying the process of factorization for joint distributions. The process for Bayesian Robot Programming, with respect to our use of Bayesian networks, is defined as follows:

(a) **Define the set of relevant variables.** This involves identifying the random variables that are directly relevant to the program desired. In a Bayesian network, this is implicit in the edges between nodes that represent dependencies. Usually, a single child node is queried to include information from all related nodes.

(b) **Decompose the joint distribution.** The original BRP methodology explicitly partitioned a joint distribution of M variables $\mathrm{P}(X_1, \ldots, X_M|\delta, \pi)$ into subsets, each one a conjunction, and then used the product of the factorization of each subset, a process called decomposition [Lebeltel et al., 2004]. We make use of the properties of the Bayesian network for implicitly including information in parent nodes when queried and apply the factorization rules to reduce $\mathrm{P}(X_1, \ldots, X_M|\delta, \pi)$ to a product of conditional distributions $\mathrm{P}(X) \prod_{m=1}^{M} \mathrm{P}(X_m|\delta, \pi)$. which are queried recursively

(c) **Define the forms.** For actual computations, the joint and dependent distributions must be numerically defined, which is done by inserting discrete values into each node. The most common function to be used is the Gaussian distribution with parameters mean $\bar{x}$ and standard deviation σ that define the shape of the distribution. A uniform distribution can also be set with $\mathrm{P}(X = x) = k$, having the same probability regardless of value. Finally, because we program continuous distributions as function calls to a function pointer defined in the node structure, a distribution that is a function into some arbitrary external process such as a map can be used.

(d) **Formulate the question.** Queries into a BRP system traditionally involve partitioning the random variables in $\wp$ into three sets: a searched set $Se \subset \wp$ for variables that contain information we want to determine, a known set $Kn \subset \wp$ for variables that contain an observed state so that $\mathrm{P}(X = x) = 1$ for $X \in Kn$, and unknown variables $Un \subset \wp$ that are only stochastically estimated. Under these sets, a "question" is formulated as the posterior probability distribution $\mathrm{P}(Se|Kn, \pi)$. Although many interpretations can be used,

IMA Conference on Mathematics of Robotics
9 – 11 September 2015, St Anne's College, University of Oxford

we will use marginal MAP queries to obtain the value of highest probability, making the form of the question $\arg\max_{Se} \sum_{Un} \mathrm{P}(Se, Un|Kn, \delta, \pi)$. The use of a Bayesian network formalizes the relationships of these sets, so that a query into a common child node of Se incorporates information from parents Kn and Un of Se. It is important to note that a "question" is functionally another conditional distribution, and therefore can operate in the same way as an additional node in the Bayesian network.

(e) **Perform Bayesian inference.** To perform inference into the joint distribution $\mathrm{P}(X_1, \ldots, X_M|\delta, \pi)$, the "Question" that has been formulated as a conjunction of the three sets Searched (Se), Known (Kn), and Unknown (Un) is posed to the system and solved as a Bayesian inference that includes all relevant information to the set Se. For our Bayesian network implementation. The "Answer" is obtained as a probability distribution, and a specific maximum or minimum value can be obtained using a value from the set Se and a Maximum A Posteriori (MAP) query as $\mathrm{MAP}(X|Y = y) = \arg\max_x \sum_Z \mathrm{P}(X \cap Z|Y)$.

Obtaining the joint distribution $\mathrm{P}(Se|Kn, \pi)$ is the goal, and requires information from all related random variables in $\{Kn, Un, \pi\}$, which in the Bayesian network are visualized as parents of Se. This distribution can always be obtained using [Lebeltel and Bessière, 2008]

$$\mathrm{P}(Se|Kn, \delta, \pi) = \frac{1}{\Sigma} \sum_{Un} \mathrm{P}(Se, Un, Kn|\delta, \pi) \tag{2.1}$$

where $\Sigma = \sum_{\{Se,Un\}} \mathrm{P}(Se, Un, Kn|\delta, \pi)$ acts as a Normalization term. To complete the inference calculation, we only need to reduce the distribution $\sum_{Un} \mathrm{P}(Se, Un, Kn|\delta, \pi)$ into marginally independent factors that can be determined. We assume that independence is denoted by the structure of the Bayesian network, so we only need be concerned with the ancestors of Se and do not need to scale by Σ. Given that inference into a Bayesian network typically involves querying a single node, we will assume that Se is the singleton $Se = \{X\}$. This can also be accomplished if Se is larger by making X a parent of all nodes in Se. Applying the chain rule for random variables to Bayesian networks, we can walk the Bayesian network backwards through the directed edges from X, determining the conditional distribution of each node from its parents as we go, and therefore breaking down the determination of the joint distribution into smaller, separate calculations.

3. Bayesian Network Implementation

For us to be able to properly organize and represent a large set of joint distributions using factorization in this way, we need a method of clearly associating random variables that are dependent on each other, in the case of our sensor example the association of W and X with Y. A Bayesian network with random variables as nodes provides a compact way to encode both the structure of a conditional factorization of a joint distribution, and also (equivalently) the independence assumptions about the random variables included in the distribution. Independent random variables have no parents, while marginally dependent or conditional random variables have parents connected as in the case of $(W \perp X|Y)$. Hence, the Bayesian network can serve as a framework for formalizing expert knowledge about how the world works. Applying the chain rule to all nodes in a Bayesian network, the probability distribution over a given network or subnetwork of nodes $\wp = \{X_1 \ldots X_M\}$ can be said to factorize over $\wp$ according to the dependencies in the network if the distribution can be expressed as a product [Koller and Friedman, 2009]

$$\mathrm{P}(\{X_1 \dots X_M\}) = \prod_{m=1}^{M} \mathrm{P}(X_m|Pa(X_m)). \tag{3.1}$$

Due to the dependency structure of the network, the conditional posterior probability of a given node X depends on all its parents $\mathrm{Pa}(X)$, so queries are a recursive operation through all parent nodes $Y \in \mathrm{Pa}(X)$ to determine their probability distributions, then multiplying them by the probability distributions of each parent node Y such that

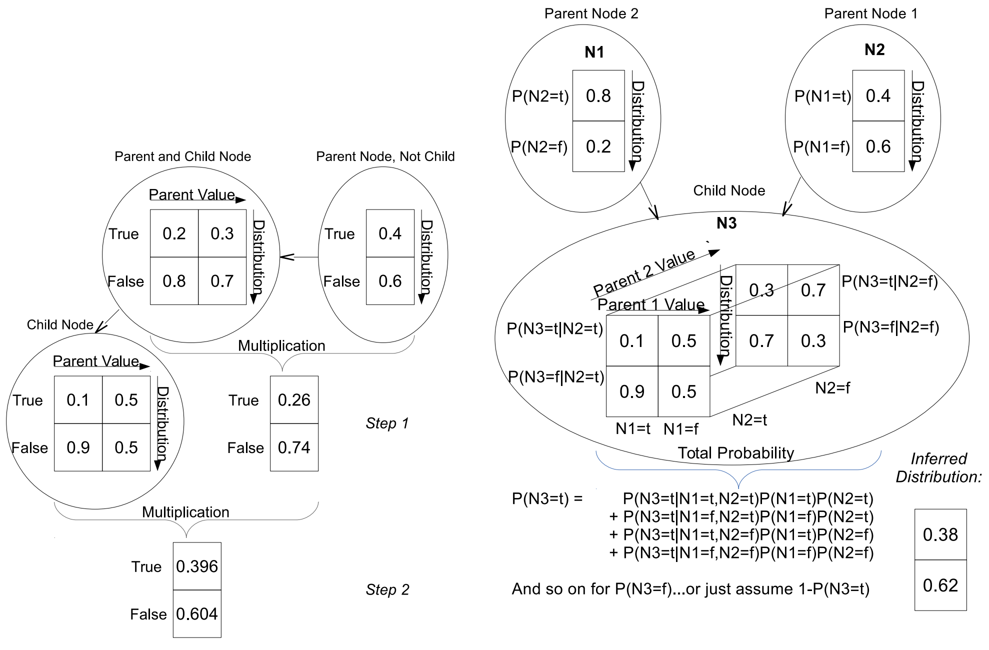
$$\mathrm{P}(X = x) = \sum_{Y \in \mathrm{Pa}(X)} \mathrm{P}(X = x|Y = y)\mathrm{P}(Y = y). \tag{3.2}$$

For single-parent random variables this is effectively a matrix multiplication, but for multiple parents, each combination of probabilities leading to a given value must be considered. Representing probability distributions by matrices, the distribution $\mathrm{P}(X|Y)$ will be an $L+1$-dimensional matrix for L parents, with the major dimension of size N for N random variable values in $\mathrm{V}(X)$. A given parent Y is assumed to have a distribution of size M, so that the distribution $\mathrm{P}(Y)$ is a $N \times 1$ matrix. Each element of the matrix represents the probability of a given combination of values from parents occurring, so to calculate the conditional distribution for X, all probability products of each combination leading to a given outcome are added together, as in Equation 3.2. This results in an $N \times 1$ matrix that is stored as a temporary posterior distribution estimate for $\mathrm{P}(X)$ which avoids frequent recalculation for determining the conditional distributions of children $\mathrm{Ch}(X)$ while traversing the network. This process is graphically illustrated in Figure 1 for a set of single parents (a) and for multiple parents (b). In this way, any node in a Bayesian network can be queried to obtain a probability distribution over its values. Considering Z to be the parents of each ancestor node Y and following the method of Equations 3.1, and Equation 3.2, and Equation 2.1, a general expression for the factorization of $\mathrm{P}(Se|Kn, \delta, \pi)$ through the Bayesian network is

$$\mathrm{P}(Se|Kn, \delta, \pi) = \sum_{Y \in \{X, \mathrm{An}(X)} \left[\prod_{Z \in \mathrm{Pa}(Y)} \mathrm{P}(Y|Z)\mathrm{P}(Z) \right]. \tag{3.3}$$

For our Bayesian network implementation, with the random variables in Se, Kn, and Un internally linked together as nodes and edges. Nodes associated with actions to be taken typically have conditional distributions that act as "questions" regarding their operational state. If for example, the question is asked what should the right-side motors do?, the network nodes related to obstacle presence and mapping, and in turn, prior sensor data, will have to be traversed to obtain the Answer, which is a posterior probability distribution that is used to select a particular value of motor speed.

At minimum, a random variable with N possible values will have a $1 \times N$ distribution matrix. A parent node with M values will create at most M additional distributions, and in general, if each parent has a distribution N_l values in size, and there are L parents, then the number of distributions M possible in the child node are $M = \prod_{l=1}^{L} N_l$, so the storage size of the node scales roughly as N^L. This can be mitigated to make processing more efficient by designing a deeper graph with more nodes and less parents per node. A parent node with an $M_l \times N_l$ distribution matrix, regardless of the number of parents and the size of M_l, will still only contribute N_l values to its child nodes. A given node X



(a) Single-parent (a multiplication) (b) Multiple Parents (total probability)

FIGURE 1. Matrix calculations for querying a discrete random variable

will then have to store a table of size $|\mathrm{V}(X \cup \mathrm{Pa}(X))|$. Total probability requires that all values in each row m sum to 1. The joint distribution P of probability values associated with a given random variable is the content actually stored.

Because the dimensionality of data stored in the node changes with number of parents, we use a single array for storing the distribution and index it with a linear index that is a function of the parent numbers of the node. To create the index, we model the array as an $L+1$-dimensional matrix for L parents and calculate an index i. For a two-dimensional row-major-order matrix with row (m) and column (n) indices, $i = n + m * columns$. By recognizing that each additional dimension must be indexed by multiplying past the sizes of all preceding dimensions, we can set index i using matrix indices m_1 for dimension 1 of size M_1 (we choose columns here for consistency), m_2 for dimension 2 of size M_1(we choose rows here for consistency), and $m_3, m_4, \ldots$ and above for additional matrix dimensions of size $M_3, M_4, \ldots$ respectively, obtaining

$$i = m_1 + m_2 M_1 + m_3 M_2 M_1 + \ldots + m_{L+1} \prod_{l=1}^{L} M_l = \sum_{n=1}^{L+1} \left(m_n \prod_{l=1}^{n-1} M_l \right). \tag{3.4}$$

4. Programming and Fixed-Point Processing

Unlike most Bayesian network implementations, our implementation is unique in that it uses fixed-point math for storage and calculation and is programmed in the C language for better portability and calculation efficiency on small-scale embedded systems and microcontrollers that do not have a floating-point arithmetic unit [Lauha, 2006]. While the use of fixed-point arithmetic does limit numerical precision depending on how the integer and fractional components are divided into numbers of bits, the imprecise nature of probabilistic estimation allows some degree of tolerance to precision limitations. An unsigned integer component with n_i bits can represent a maximum value of $2^{n_i} - 1$ in increments of 1, and a fractional component can represent up to $(2^{n_f} - 1)/2^{n_f}$ in increments of $1/2^{n_f}$, so the maximum bounded error of a fixed-point number with respect to the real number it is supposed to represent is $|1/2^{n_f}|$. Additionally, while equally-balanced *8.8* and *16.16* bit divisions for 16-bit and 32-bit words are most common, probabilistic values p are generally real positive numbers between zero and 1, and can therefore be represented suitably by fractional-only *0.16* and *0.32* bit divisions to maximize precision on a given architecture. Integer components are currently still used for probabilistic calculations due to the care required in ensuring that no part of any calculation ever exceeds 1.0, which limits the range of calculations that can be safely performed.

Mathematical operations are implemented in the conventional way for two fixed-point numbers a and b to yield a result c [Street, 2004], with addition and subtraction being unchanged from integer arithmetic ($c_{a+b} = a + b$, $c_{a-b} = a - b$) and multiplication and division done with bit shifting as well as multiplication ($c_{a\times b} = (a \times b) >> (n_f)$, $c_{a\div b} = \frac{(a<<(n_f))}{b}$). Inversion of numbers is accelerated by using only $c_{1\div a} = \frac{(1<<(2n_f))}{a}$, square root operations are performed using Turkowski's algorithm [Turkowski, 1994], and conversion to and from a floating-point number a_{float} to a fixed-point number b_{fix} must include adding 0.5 to round the number rather than truncating it. This implementation is used in all inference calculations, and the type "fix" is currently defined from "int32_t" on Intel and ARM architectures and from "int16_t" on Atmel AVR microcontrollers. Converting between word lengths and between fixed and floating point functionality is easily accomplished by changing a single C header file. The common vector and matrix operations in row-major order also were implemented in C using the fixed-point math functions for calculation. Additionally, many common constants are pre-defined in fixed-point format for efficient use such as π, 2π, e, $\sqrt{2}$, $\sqrt{3}$, etc.

The Bayesian nodes themselves are defined as *BNode* C structures with the following members that implement the methodology described above:

name: A unique name string for identification of the node
numParents: The integer number of parents the node has (L)
parent: An ordered array of integer node numbers indicating the parents of the node
numVals: The integer number of states or discrete node distribution size (N)
distSize: The total size in words of the $L + 1$-dimensional distribution matrix
conditionals: A fixed-point array for the size N inferred (conditional) distribution
distributions: A fixed-point array for the $L + 1$-dimensional distribution matrix
distFunction: A function pointer to a single-input single-output fixed-point function representing a continuous distribution to be used as an alternative to discrete arrays

Program flow starts with allocating memory for the node structures, which is generally done as a static array for reliability. Each node is then initialized using the *AddBNode()* function with a fixed-point array for the local distribution and a unique name that can be searched for with the *findBNode* function. Alternatively, *AddBFunction()* can be used to set a function pointer taking a single fixed-point argument representing a continuous distribution. The structure of the array is then defined using the *AddBParent()* function to add a node number as a parent for another node, which also implicitly resizes the internal array to add another dimension to the nodal distribution. This additional dimension is initially populated simply by copies of the original node distribution (indicating no dependency on the new parent node), but new arrays can be set using *setBNodeProbVector()* to define the index of the array to be changed with a size $L+1$ vector, or *setBNodeProbVec-*

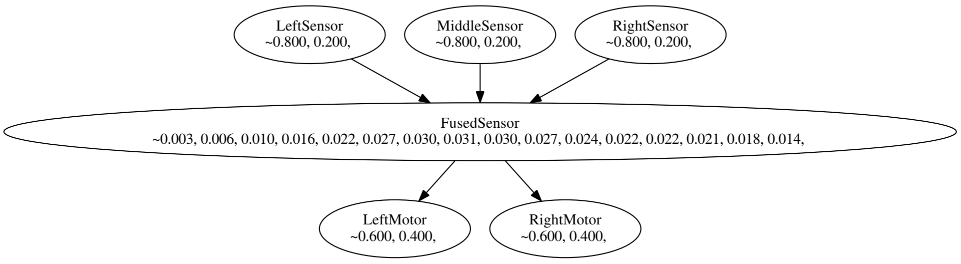


FIGURE 2. Simple Bayesian Network for Obstacle Avoidance Program

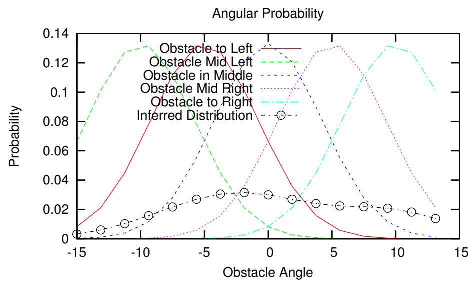
torCoords() to use $L + 1$ separate coordinate arguments for convenience. The inference operation is performed on a target node number with *inferBNode()*, which calculates the inferred conditional array that can be directly queried by *getBNodeInfVector()*. All functions return the size of the array being operated on to improve reliability. The contents of the whole network can also be printed by the *printBNetwork()* function. Graphs can also automatically be generated by the system using the DOT scripting language and the gnuplot tool to aid in visualizing the network itself, seen in Figures 2-4.

5. Sensory Reasoning Example

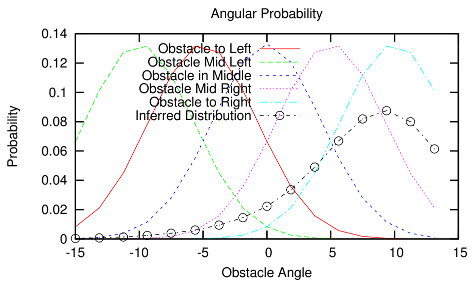
To test the concepts and implementation detailed above, a simple six-node network was set up which is graphed with base probabilities as shown in Figure 2, consisting of three nodes representing obstacle sensors with states $\{Clear, Obstacle\}$ angled 30 degrees apart in front of a mobile robot, a sensor fusion node that uses Gaussian distributions to estimate the detection angles of the three sensors, and two motor nodes for control of left and right side motors. A MAP query is used to determine the most appropriate state of the motors from $\{Forward, Reverse\}$ given inference into the sensor data.

It is assumed that each obstacle sensor has only a 20% prior probability of seeing an obstacle in practice and 80% probability of no obstacles, but this is updated when the sensor actually detects an obstacle. The sensor fusion node is pre-loaded with a set of six Gaussian functions representing an approximate likelihood of the direction of an obstacle given the six possible states of the three two-state obstacle sensors. Figure 3 shows the angular probabilities of these Gaussian functions and the resulting inferred distribution (denoted with markers) for (a) no obstacles detected, and (b) an obstacle detected by the right-hand sensor with 90% probability. The motor nodes are defined with a high probability of reverse movement for obstacles detected on the opposite side to cause obstacle avoidance behaviour, but generally a higher probability of forward movement to keep the robot moving forward if obstacles are unlikely. The functions used are shown in Figure 4 for the right and left motors

This function could be constructed in any way desired to reflect the response of the motors to likelihood of an obstacle, but a simple function is used in this case to illustrate that the system still works. The presence of an obstacle increases the inferred "probability" of the left motor reversing, which in this context would be contextualized as "probability that reversing will avoid an obstacle", and the motor will reverse when a MAP query is applied. Optimal priors can be determined by expert knowledge or by statistical learning methods which will be a focus of future work. Graphing the priors used for inference at each node shows a notable similarity to fuzzy functions, and the process of inference into discrete values that of fuzzification, but a random variable is expected

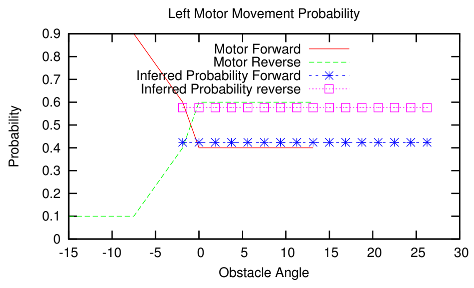


(a) No obstacles detected

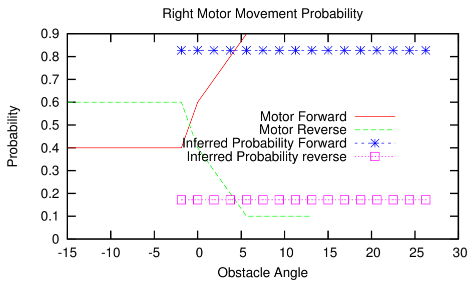


(b) Obstacle detected to right (90%)

FIGURE 3. Sensor Fusion Probability Priors and resulting Inferred Distributions



(a) Left motor movement



(b) Right motor movement

FIGURE 4. Priors for Motor Movement and Probabilities of Obstacle Avoidance for Obstacle Detected to Right (90%)

to have a single outcome while fuzzy sets represent multiple outcomes, or alternately imprecise degrees of belief.

6. Conclusions

We have described a method for making decisions using the Bayesian Robot Programming paradigm using Bayesian networks for organization of data and efficient algorithms for storage and inference into discrete random variables. This method has been implemented as a novel, efficient, structured framework for practical use of Bayesian networks and probabilistic queries on an embedded system with fixed-point arithmetic, leading to an efficient and intuitive way of representing robotic knowledge and drawing conclusions based on statistical information encapsulated in the network. The applicability of these methods to robotic programming and control is very wide, and many future applications are expected, as this method can be extended to much more complex systems simply by adding networked random variables for sensor and actuator quantities, so long as they can be probabilistically characterized. Future work includes implementations for new applications, automatic determination of network structure and optimal priors from existing hardware data, and the implementation of statistical learning methods to improve performance of the system while it operates. Building a distributed sensing and control network that shares and uses probabilistic information in the manner of a Bayesian network is one of the target goals of this work. Additionally, the use of fuzzy random variables as a more general structure to allow greater flexibility in interpreting variable information is being investigated.

REFERENCES

Bessiere, P., Lebeltel, O., Lebeltel, O., Diard, J., Diard, J., Mazer, E., and Mazer, E. (2000). Bayesian robots programming. In *Research Report 1, Les Cahiers du Laboratoire Leibniz, Grenoble, FR*, pages 49–79.

Kjrulff, U. B. (2008). *Bayesian Networks and Influence Diagrams: A Guide to Construction and Analysis.* Springer Science.

Koller, D. and Friedman, N. (2009). *Probabilistic Graphical Models, Principles and Techniques.* MIT Press, Cambridge, Massachusetts.

Lauha, J. (2006). The neglected art of fixed point arithmetic. Presentation.

Lebeltel, O. and Bessière, P. (2008). Basic Concepts of Bayesian Programming. In *Probabilistic Reasoning and Decision Making in Sensory-Motor Systems*, pages 19–48. Springer.

Lebeltel, O., Bessière, P., Diard, J., and Mazer, E. (2004). Bayesian robot programming. *Autonomous Robots*, 16(1):49–79.

Lebeltel, O., Diard, J., Bessiere, P., and Mazer, E. (2000). A bayesian framework for robotic programming. In *Twentieth International Workshop on Bayesian Inference and Maximum Entropy Methods in Science and Engineering (MaxEnt 2000)*, Paris, France.

Post, M. (2014). *Planetary Micro-Rovers with Bayesian Autonomy.* PhD thesis, York University.

Street, M. (2004). A fixed point math primer. In *OpenGL ES Game Development.* Course Technology PTR.

Turkowski, K. (1994). Fixed point square root. Technical Report 96, Apple Technical Report. Also appears in Graphics Gems V.