# Probabilistic Robot Action Planning/Control

John Thywissen
The University of Texas at Austin
Austin, Texas 78712-0233 USA
jthywiss@cs.utexas.edu

## I. INTRODUCTION

Robot behaviors are often designed using a set of states, with transition conditions among them, *i.e.* a finite state machine. The transition conditions often depend on uncertain inputs. Thus, transitions could fire incorrectly or fail to fire. This necessitates complicating the behavior design with additional transitions to "back out" of states judged to be erroneous.

Current robotic systems maintain probabilistic models of their environment. Deterministic state transitions are a mismatch for these models in that they require a single datum to be extracted from the probabilistic models to use as an input to the state transition decision. For example, a particle filter would be processed to produce a choice for "the" current robot pose to use as a basis for action decisions. In some sense, this "throws away" the advantages of tracking multiple hypotheses in the particle filter.

In this work, we seek to carry the probabilistic models into behavior decision making, insofar as it is possible. Clearly, a robot cannot choose to, for example, move to two different places at once, but not all decisions are mutually exclusive. For example, a robot could move its head to visually scan for objects while walking to a target location.

## II. PROBLEM STATEMENT

Given:

1) A robot in a sensed environment, with sensors that are neither omniscient nor infallible.
2) Models for drawing probabilistic conclusions from this sensor data.
3) A set of possible robot actions, along with models of their probable effects.
4) Goals for the robot, with an objective function (which supplies values for the goals and costs for actions).

Can one, in the robots' operation:

1) Use the probabilistic sensor-based beliefs to make probabilistic action planning/control decisions? For example, choosing high-cost-of-failure actions based only on highly probable beliefs.
2) Use the lack of sensor data or uncertainty in sensor data to make action planning/control decisions? For example, choosing to take actions to reduce uncertainty rather than greedily pursuing the goal.
3) Automatically select action sequences to achieve a goal in an appropriate manner?

Further, can the above be achieved with a simple software interface for action implementers and maintainers; and without requiring burdensome types or amounts of *a priori* data?

## III. PROPOSED SOLUTION

We propose the following approach to these questions:

1) Reify action sequences previously embedded in monolithic behavior code as *activities*.
2) Represent the belief state as *Dempster-Shafer belief functions*.
3) Use a planning algorithm similar to partially-observable Markov decision processes (POMDPs), but simplified and modified for Dempster-Shafer belief functions.

We have built a small framework implementing this approach.

### A. Activities

Since the problem is about actions, which are often implemented implicitly as part of large codebases, a necessary first step is to make actions addressable by the framework. We do this by reifying them as activities.

Final Project, 4 Dec 2009
C S 393R, *Autonomous Robots*
Fall 2009, Prof. Peter Stone
Department of Computer Science, The University of Texas at Austin

The term activity is used instead of action, to emphasize that they are often an sequence of actions or an ongoing repeated action – for example, scanning for an object by moving the head back and forth. Actions are often thought of as "one shot", but activities in our framework are activated and continue to run until deactivated.

Activities, in addition to supplying their implementation, a responsible for supplying data about themselves, such as the cost to perform them, and their preconditions.

## *B. Dempster-Shafer Belief Functions*

Conventionally, state machine transitions are conditional upon logical propositions. To represent uncertainty, we can assign a probability to propositions. The predominant approach to probabilistic belief representation is Bayesian, where each proposition is assigned a probability. A significant drawback of Bayesian belief appears in the absence of information. The oft-cited example is this: Suppose we have a coin that is being flipped, and our propositions of interest are "heads" $h$ and "tails" $t$. In absence of other information, the Bayesian approach would use a uniform distribution to assign probabilities, $P(h) = 0.5$ and $P(t) = 0.5$. Suppose we are told the coin is not necessarily fair. The Bayesian $P(h) = 0.5$ and $P(t) = 0.5$ now seems to represent an unfounded conclusion, but has no other effective means to approach this problem.

Dempster-Shafer belief functions [1] approach propositional questions as a *frame of discernment*, which is a set of mutually-exclusive and collectively-exhaustive propositions with respect to a question. It is always the case that, in "reality", exactly one proposition in a frame of discernment is true. Dempster-Shafer belief functions then permit probabilities to be assigned to any subset of propositions in the frame, with the restrictions that all the assignments must add to one, and that no probability is assigned to the empty set.

On this footing, we can now represent the coin example above. The basic probability assignment for that scenario is assign 1 to $\{h, t\}$, and 0 to $\{h\}$ and $\{t\}$. In other words, "It'll certainly be either be heads or tails, but I don't know which". If the coin is known to be fair, the assignment is $\mathrm{m}(\{h, t\}) = 0, \mathrm{m}(\{h\}) = 0.5, \mathrm{m}(\{t\}) = 0.5$, where $\mathrm{m}$ is the basic probability assignment function. Note that the assignment of 1 to the union of all propositions and 0 to all other combinations represents total lack of information, or total ignorance. This is called a vacuous belief function. Vacuous belief functions are an appropriate initialization value for belief functions before evidence begins arriving for or against propositions.

Belief function theory provides the belief $\mathrm{Bel}$ and plausibility $\mathrm{Pl}$ functions, which, for a given set of propositions, provide the committed belief to that set, and the maximum possible belief for that set, respectively. Belief function theory also provides *Dempster's rule of combination*, which combines two belief functions of the same frame. Belief functions and Dempster's rule of combination form a commutative monoid, with the vacuous belief function as the identity element. This means use of the combination rule is simple and flexible: there are no order dependencies, and combining with "no information" data is safe. These properties are relied upon in the state prediction step of the policy function described later.

Belief functions are a useful representation of the belief state, from which our framework can make decisions about selecting activities to execute.

## *C. POMDP-like Planning Algorithm*

Uncertainty in action selection in probabilistic robotics is handled by partially observable Markov decision processes (POMDPs) [2]. A POMDP provides a *policy*, which maps a belief state to a set of recommended activities, based on outcomes expected of executing the activities. Actions and states are assigned costs and values, leading to an objective function (also known as a payoff function). The goal of the policy is to find a feasible set of activities that maximizes the objective function. Markov decision processes and POMDPs find the policy by simulating various actions in the current state and computing the expected objective function values. This is performed recursively, for a number of steps determined by the planning horizon. This yields the expected cumulative payoff of selecting each action.

Our framework uses a process similar to Markov decision process value iteration, modified to both simplify the information required, and to use Dempster-Shafer belief functions in place of Bayesian probabilities. Importantly, a low degree of belief of a relevant fact may indicate the need to take actions to gather more sensor data, rather than proceed to the overall goal based on the most likely (but very uncertain) world state. The framework's use belief of functions enables this.

Some simplifying assumptions include:

- Value is inherent in states, not actions. This is not a strong assumption because, like the Markov assumption, it can be bypassed by expanding the state. If an action itself has value, then add a variable to the state recording "action just executed". This assumption does simplify the interface and implementation of the objective function.

- Predicted state changes by actions are independent. This assumption is made when the policy function combines multiple actions' predicted effects by using Dempster's rule. Often in practice, the assumption of independence can be an acceptable approximation. Also, this problem can be partially mitigated by state expansion again, in limited circumstances.[1]
- Instead of integrating across the entire belief space, the policy function can simply sum over the available action combinations. This assumption was inspired by QMDPs, but it needs rigorous validaition.

The policy making procedure is detailed in algorithms 1 and 2.

## IV. RELATED WORK

### A. Probabilistic Controllers

Work under the moniker "probabilistic control", for example [3], is generally part of the control theory field. This work assumes a classical control problem such as temperature control, where there is a stable goal for a single (or small number of) controlled variables. The problem we address here is how to switch among states with various goals, perhaps using classical control engineering within each state.

### B. Fuzzy Logic

Fuzzy logic is one approach to probabilistic control that is common in industrial controllers. It may appear at first glance that fuzzy logic might be adapted to the problem at hand; however, despite its heuristic usefulness in industrial controllers, fuzzy logic is unsuited for more complex reasoning about the environment in systems such as robots. For a convincing argument, see Elkan [4].

### C. Partially-Observable Markov Decision Processes

Kaelbling, Littman, and Cassandra [5] introduced partially-observable Markov decision processes (POMDPs) to the AI community from the operations research community. Pineau *et al.* [6] demonstrate a robotics application of POMDPs. Thrun, Burgard, and Fox [2] contains a well-presented development of MDP and POMDP principals and algorithms, as well as presenting QMDP and other alternative MDPs.

### D. Belief Function (Dempster-Shafer) Theory

Dempster-Shafer belief function theory has already been described. A very readable and comprehensive foundation is in [1]. Further detail, and excellent discussions on probabilistic reasoning in general, is found in [7], [8], [9].

### E. Dempster-Shafer Theory in Robotics

Dempster-Shafer belief functions are not widely used yet in robotics, however there are a few precedents. Safranek, Gottschlich, and Kak [10] used belief functions and computer vision to verify object placement. Hughes and Murphy [11] used belief functions to build an occupancy map from ultrasonic range sensor data. Murphy [12], Wu *et al.* [13], and Yu *et al.* [14] applied belief functions to sensor fusion.

## V. IMPLEMENTATION DETAILS

The framework is implemented to work as part of the Tekkotsu [15], [16] robotics platform. However, the coupling to Tekkotsu is not very extensive, so the code could be easily ported to other platforms.

### A. Behavior

The primary framework class is `ProbabilisticBehavior`, which subclasses Tekkotsu's `BehaviorBase`. Framework users should subclass `ProbabilisticBehavior` to declare their behavior classes.

### B. Activities

Each activity is implemented with three principal methods: `start`, `run`, and `stop`. When an activity is activated by the framework, `start` is called once. Then, on receipt of events, each event is dispatched all started activities' `run` method. When conditions indicate the deactivation of an activity, its `stop` method is invoked. [2]

[1]For example, if the non-independent effects can be noted in one simulation step and corrected in the next.

[2]As a convenience, activities may be declared simply by placing `ACTIVITY_DECL(BehaviorClassName, activityName);` in the behavior class declaration. This declares the activity class and defines three required housekeeping methods.

### C. *Preconditions*

Activities declare when they are available for execution by supplying two precondition-related methods. The `preconditionsMet` method uses the supplied belief state to compute the probability that the activities' preconditions are met. Depending on the activity, this may be based on Bel or Pl functions on propositions in the belief state. The `preconditionThreshold` method supplies the threshold probability value that the `preconditionsMet` must exceed in order for the activity to be considered available for possible activation.

### D. *Activity compatibility*

Multiple activities may be selected by the `ProbabilisticBehavior` class to be active simultaneously. However, not all activities can be run with each other. For example, the "walk to a point" and the "kick ball into the goal" activities may not be possible to execute at the same time. These types of constraints are expressed by the `isFeasibleActivitySet` method of the behavior class. Given a set of proposed active activities, this method should return a boolean value indicating whether the activities are mutually compatible.

### E. *Planning*

The `ProbabilisticBehavior` class selects activities for activation and deactivation every `planPeriod_ms` milliseconds (a constructor parameter). Based on the policy detailed below, `ProbabilisticBehavior` selects a feasible set of activities to be active until the next planning time. Activities newly added to this set are activated, and activities removed from the set since the last planning time are deactivated.

### F. *Policy*

A policy maps a belief state to a set of recommended activities, based on outcomes expected of executing the activities. The goal of the policy is to find a feasible set of activities that maximizes value minus cost. `ProbabilisticBehavior` does this by a process similar to a Markov decision process, modified to both simplify the information required, and to use Dempster-Shafer belief functions in place of Bayesian probabilities. The procedure is detailed in algorithms 1 and 2.

To support this policy computation, implementing behavior classes need to supply an `inherentStateValue` method implementation that maps belief states to a value of being in that state. For example, states with high "ball in goal" beliefs are assigned a high value. Similarly, activities need to supply a `cost` method implementation that maps an action in a belief state to a cost to perform that action.

Finally, activities need to supply a `predictStateChange` method implementation that maps actions in a belief state to beliefs about the changes that will occur to the state.

When combining multiple actions, the policy combines predicted state changes using Dempster's rule of combination. It also assumes the values simply add linearly.

**Algorithm 1** POLICY($b$ : belief state)

```
1:  $A \leftarrow$ empty list
2:  **for** each activity $a$ in the list of defined activities **do**
3:      **if** $a.preconditionsMet \geq a.preconditionThreshold$ in $b$ **then**
4:          $h \leftarrow \text{combine}(b, a.predictStateChange(b))$
5:          $\hat{v} \leftarrow -a.cost(b) + \text{VALUESTATE}(h)$
6:          Store $\langle s, \hat{v} \rangle$ in list $A$
7:      **end if**
8:  **end for**
9:  $\hat{v}_{\text{max}} \leftarrow -\infty$
10: **for** each combination $C$ of activities in list $A$ **do**
11:     **if** $isFeasibleActivitySet(C)$ **then**
12:         $\hat{v}_C \leftarrow \sum_{i \in C} A_{i,\hat{v}}$
13:         **if** $\hat{v}_C > \hat{v}_{\text{max}}$ **then**
14:             $\hat{v}_{\text{max}} \leftarrow \hat{v}_C$
15:             $C_{\text{max}} \leftarrow C$
16:         **end if**
17:     **end if**
18: **end for**
19: **return** $C_{\text{max}}$
```

**Algorithm 2** VALUESTATE($b$ : belief state)

```
1: **if** recursion depth limit (planning horizon) not exceeded **then**
2:     $A \leftarrow \text{POLICY}(b)$
3:     $\hat{b}' \leftarrow predictState(A, b)$
4:     **return** $inherentStateValue(b) - costActivity(A, b) + \text{VALUESTATE}(\hat{b}')$
5: **else**
6:     **return** $inherentStateValue(b)$
7: **end if**
```

### G. Belief Functions

Dempster-Shafer belief functions are represented by instances of the `BeliefFunction` class. This class maintains the state for a single frame of discernment. The current implementation assumes a small number ($\ll 32$) of propositions. For applications with a large number of propositions, such as a localization particle filter, this class will need an alternate implementation, though the interface will remain unchanged. Recall that belief functions operate on sets of propositions, not just single propositions. `BeliefFunction` supplies methods that generate proposition sets, and map a proposition set $P$ to the basic probability assignment $\mathrm{m}(P)$, belief value $\mathrm{Bel}(P)$, and plausibility value $\mathrm{Bel}(P)$. The basic probability assignments are changed using the `setProb` method. Dempster's rule of combination is provided by the `combine` method, which overwrites the receiving object's state with the combination of the two belief functions supplied as arguments.

Several `BeliefFunction` methods check the invariant of basic probability assignments:

$$\sum_{P \in \wp(\Theta)} \mathrm{m}(P) = 1,$$

where $\Theta$ is the frame of discernment. If this precondition should not be met, these methods will throw `std::logic_error`.

Two subclasses of `BeliefFunction` are provided, each which specializes belief functions for a type of proposition set. The base `BeliefFunction` class's propositions are identified simply by an integral index. `EnumFrameBeliefFunction` takes an enumeration as identifying all the propositions in a frame. `BinaryFrameBeliefFunction` represents a frame of a single proposition and its negation.

### H. Galleries

Collections of frames of discernment are called galleries. Instances of galleries are represented by instances of the `GalleryState` class. This class provides an indexed collection of belief functions and a convenience method to combine each belief function of a gallery with the corresponding belief function of another gallery. `ProbabilisticBehavior`'s belief states are represented as `GalleryStates`.

## VI. EVALUATION

Returning to the problem statement, we have demonstrated a framework that does provide the three requirements. The framework provides a means to:

1) Use probabilistic sensor-based beliefs to make probabilistic action planning/control decisions.
2) Use a lack of sensor data or uncertainty in sensor data to make action planning/control decisions.
3) Automatically select action sequences to achieve a goal.

The intent for the evaluation was to demonstrate this on a robot using vision-based action recognition. A soccer goal-keeper robot would be attempting to sense opponents' actions. The action recognizer may produce a Bayesian belief state, but it may have situations where actions are completely ambiguous. For example, as a opponent begins to kick the ball, it may be unclear initially whether the opponent is preparing to walk or to kick. Belief functions can represent this type of uncertainty. The goal-keeper's planner could select activities based on the action recognizer output. Unfortunately, the action recognizer was not completed in time for this report.

In the interim, we have evaluated the framework using a simple test harness that runs a static scenario. The belief state consists of one frame of discernment, the location of the ball, which has possible values of `inGoal`, `inFrontOfUs`, or `otherPlaceA`. The actions are `lookForBall`, `walkToBall`, and `kickBall`. The scenario is that the `lookForBall` action keeps reporting that the ball may be in front of the robot, but also fairly probably may be somewhere else, even after the robot has walked to the last reported position.

When the test is run, the resulting activities are the robot looks, walks while looking, and kicks. The ball appears to miss the goal, so the robot looks, walks while looking, looks some more, and finally has confidence enough to walk while looking, then kick again.

The test demonstrates that the planner can construct a usable policy, including steps that are simply "gather more information", and make progress towards a goal indicated by an the objective function.

This test was not designed as a generalizable valid experiment, but as a development integration test. So, the results are anecdotal. More robust evaluation remains as future work. However, it does provide evidence that the project's goals have been met,

## VII. CONCLUSION AND FUTURE WORK

### A. Conclusion

"It's a stochastic world", and robots must handle that. This work is a start down the road of removing much of the complexity of handling uncertainty in action planning/control. Ideally, this road will take us to the point where it's easier to develop probabilistic actions than deterministic actions.

### *B. Future Work*

Future work includes:

- Further validation of the policy algorithm: The algorithm needs to be checked much more extensively for stability and reasonableness of results, particularly with respect to the assumptions made.
- Performance improvements: There are many optimization opportunities that need to be taken. Currently, the performance would not be acceptable on the robot.
- Framework interface improvements: The interface for `Activity` has been engineered to be quite simple for framework users. The same needs to be accomplished for `BeliefFunction` by using an amalgamation of C++ templates and macros.

## REFERENCES

[1] R. R. Yager and L. Liu, Eds., *Classic Works of the Dempster-Shafer Theory of Belief Functions*, ser. Studies in Fuzziness and Soft Computing. Heidelberg: Springer, 2008, vol. 219.

[2] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, Mass.: MIT Press, 2006, ch. 14–16.

[3] M. Kárný and T. V. Guy, "Fully probabilistic control design," *Systems & Control Letters*, vol. 55, no. 4, pp. 259–265, 2006.

[4] C. Elkan, "The paradoxical success of fuzzy logic," *IEEE Expert*, vol. 9, no. 4, pp. 3–8, Aug 1994.

[5] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, no. 1–2, pp. 99–134, May 1998.

[6] J. Pineau, M. Montemerlom, M. Pollack, N. Roy, and S. Thrun, "Probablistic control of human robot interaction: Experiments with a robotic assistant for nursing homes," in *Proceeedings of the 2nd IARP/IEEE-RAS Joint Workshop on Technical Challenge for Dependable Robots in Human Environments*, 2002, pp. 11–19. [Online]. Available: http://www.laas.fr/drhe02/preprints.pdf

[7] J. Pearl, *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. San Mateo, Calif.: Morgan Kaufmann, 1988.

[8] G. Shafer and J. Pearl, Eds., *Readings in uncertain reasoning*. San Mateo, Calif.: Morgan Kaufmann, 1990.

[9] R. R. Yager, J. Kacprzyk, and M. Fedrizzi, Eds., *Advances in the Dempster-Shafer theory of evidence*. New York: J. Wiley, 1994.

[10] R. J. Safranek, S. Gottschlich, and A. C. Kak, "Evidence accumulation using binary frames of discernment for verification vision," *IEEE Transactions on Robotics and Automation*, vol. 6, no. 4, pp. 405–417, Aug 1990.

[11] K. Hughes and R. Murphy, "Ultrasonic robot localization using Dempster-Shafer theory," in *Neural and Stochastic Methods in Image and Signal Processing*, ser. Proceedings of SPIE, S.-S. Chen, Ed., vol. 1766, no. 2, Bellingham, Wash., USA, Dec 1992, pp. 2–11.

[12] R. R. Murphy, "Dempster-Shafer theory for sensor fusion in autonomous mobile robots," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 2, pp. 197–206, Apr 1998.

[13] H. Wu, M. Siegel, R. Stiefelhagen, and J. Yang, "Sensor fusion using Dempster-Shafer theory," in *Proceedings of the 19th IEEE Instrumentation and Measurement Technology Conference*, vol. 1, 2002, pp. 7–12.

[14] B. Yu, K. Sycara, J. Giampapa, and S. Owens, "Uncertain information fusion for force aggregation and classification in airborne sensor networks," 2004, aAAI-04 Workshop: W14: Sensor Networks. [Online]. Available: http://www.cs.cmu.edu/~byu/papers/aaai04ws-final.pdf

[15] E. Tira-Thompson, "Tekkotsu: A rapid development framework for robotics," Master's thesis, Carnegie Mellon University, Pittsburgh, Pa., May 2004. [Online]. Available: http://www.tekkotsu.org/media/thesis_ejt.pdf

[16] ——, "Tekkotsu: an open source project created & maintained at Carnegie Mellon University," Mar 2009. [Online]. Available: http://www.tekkotsu.org/