PROBABILISTIC PROGRAMMING FOR ROBOTICS

Seyed Mahdi Shamsi, Gian Pietro Farina, Marco Gaboardi, and Nils Napp
Department of Computer Science and Engineering
University at Buffalo
The State University of New York

Introduction

Autonomous robotic agents use noisy sensors and actuators to interact with a complex external world. As a result, inference engines and point estimators are essential to making modern robots work. Often, these are highly specialized and optimized to run in real-time, for online operation in autonomous agents. They are hand-crafted and make approximations such as linearization and assumed independence to gain speed or memory efficiency. While this is a reasonable approach for widely used algorithms, it makes developing new estimators both tedious and error prone.

Probabilistic programming has the potential to mitigate these two issues and make it easy to quickly build new, custom estimators that can be used throughout the lifecycle of a typical robot. When Interpreting probabilistic programs as generative processes, they can be thought of as a *specification* of the robot behavior which can then be used to tie sensor data to high-level actions.

Probabilistic Programming as a Mid-level specification language

When designing a robot, the builder, typically has a high level model of what each part of the robot is supposed to do. As a result, we propose that it is relatively easy to build a high-level behavioral model for a robot. Unknown details of the behavior might depend on unknown parameters .We would like to explore the use probabilistic programming languages for this purpose, and to design a domain specific language that makes it easy for robotics practitioners to design new generative models for their robots. For example, a code listing in *Anglican* [1] for the localization and motion task described next is given below. By changing the inputs and queries it is easy to perform a variety of practical tasks.

```
(defquery slam [obs-noise observations  motion-noise motion-commands]

    (let [obs-dist      (mvn (zero-vector 2)
                             (mul obs-noise (identity-matrix 2)))
          motion-dist   (mvn (zero-vector 3)
                             (mul motion-noise (identity-matrix 3)))
          landmark-dist (uniform-continuous 0 domain-size)
          landmarks     (repeatedly max-landmarks
                          #(vector (sample landmark-dist)
                                   (sample landmark-dist)))]

          (let [pose-seq (reduce (fn  [poses action]
                                   (conj states (add
                                      (motion-model (peek poses) action)
                                      (sample motion-noise))))
                                 [[0 0 0]] motion-commands)]

               (mapv (fn [observation pose]
                     (observe (obs-dist landmarks pose) observation)
                              observations
                              pose-seq))

                     (predict :states (last pose-seq))
                     (predict :map landmarks))))
```

Listing 1. Code for relating motion and sensor command in *Anglican*[1]. This query can be formulated from a high-level behavior description similar to the one shows in Fig.1. This program can then be used to perform various practical inference tasks.

Example Applications: Location and Map Estimation

Simultaneous localization and mapping (SLAM) is one of the basic estimation tasks performed on autonomous robots. It is a building block to more complex behaviors that require mobility and motion planning in unknown environments. The world is represented as a set of static landmarks that generate observations based on the relative position to the robot pose at time ***t***, see Fig. 1.

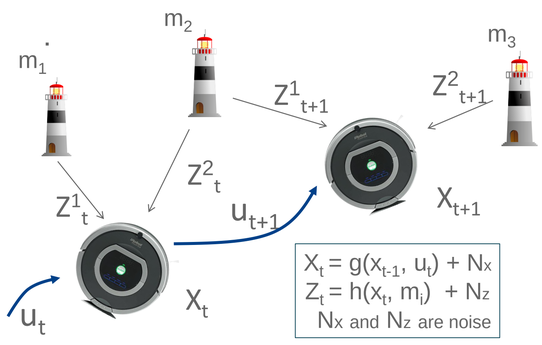

Figure 1 Schematic of SLAM problem. Landmarks and robot poses are shown at different times. The robot state is denoted by ***X***, motion inputs commands by ***u*** and observations by ***Z***.

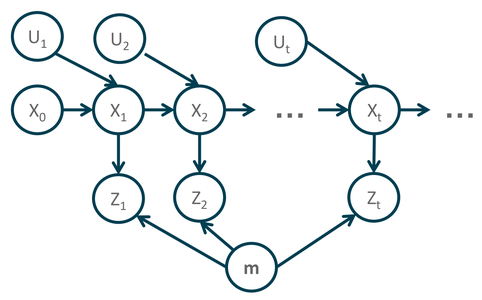

Figure 2 Graphical model for SLAM. The Bayes net shows the conditional dependence structure of the resulting PDF. The observations at different times only depend on the robot pose and the map.

Standard Filtering Approach

There are many algorithms for solving the SLAM problem. They differ primarily in the representation of the probability distributions (parametric vs. non-parametric) and whether or not they are smoothers (estimate all time steps jointly) or filters (estimate the latest time). One basic approach is filtering assuming Gaussian distributions. This results in the Extended Kalman Filter (EKF) algorithm [2].

Extended Kalman Filter ($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$)

1. $\mu p_t = g(\mu_{t-1}, u_t)$
2. $\Sigma p_t = G_t \Sigma_t G_t^T + V \Sigma_u V^T + \Sigma_\theta$
3. $K_t = \Sigma p_t H_t^T (H_t \Sigma p_t H_t + \Sigma_z)^{-1}$
4. $\mu_t = \mu p_t + K_t(z_t - h(\mu p_t))$
5. $\Sigma_t = (I - K_t H_t) \Sigma p_t$
6. return μ_t and Σ_t

Need to be calibrated

The map and pose are represented by the mean and variance (μ_t, Σ_t) of a multi dimensional Gaussian. These can be updated efficiently using the update equations given above. The insight is that Gaussian RVs are closed under affine transformations and that the motion and observation functions **g** and **h** can be linearized. However, in order to use this algorithm the noise parameters for the motion and observation need to be known *a priori*. Getting ground truth data for calibrating noise parameters is tedious since it requires external observations, e.g. through motion capture lab. Worse, the EKF algorithm behaves poorly if the parameters are chosen wrong, i.e. the matrix in step 3 is poorly conditioned.

The need for prior calibration is typical in SLAM systems, and in practice requires gathering specific datasets to perform calibration. The reason is that the inference engines are carefully constructed to perform SLAM, and not to estimate other parameters in the underlying generative process.

Using the Generative Model

The generative model in Listing 1, can be used in a variety of useful ways such as performing SLAM and calibration. It ties high-level behavior to low-level sensor values.

Performing SLAM

```
(def samples (doall (take 2000
                (doquery slam :smc
                        [+obs-noise+ +observations+
                         +motion-noise+ +motion-commands+]))))
```

Listing 2. Code for using Listing 1 to estimate the robot position given a sequence of control inputs and observations (denoted by +).

Filtering vs. Smoothing (onilne vs full)

```
(predict :states pose-seq)
```

Listing 3. Modifying the predict line for states in Listing 1 to the one listed above changes the inference problem from a filtering to a smoothing problem. These are typically implemented as completely different algorithms, e.g. EKF-SLAM[2] and GraphSLAM[3].

Using Uncalibrated Systems

```
motion-dist  (mvn  (zero-vector 3)
                   (sample (prior-for-cov-matrix-dist)))
```

Listing 4. Most SLAM systems cannot be used before calibration. However, this task is easy to accomplish with a probabilistic program specifications. Changing the motion distribution in Listing 1 to the line above puts a prior on the noise parameter.

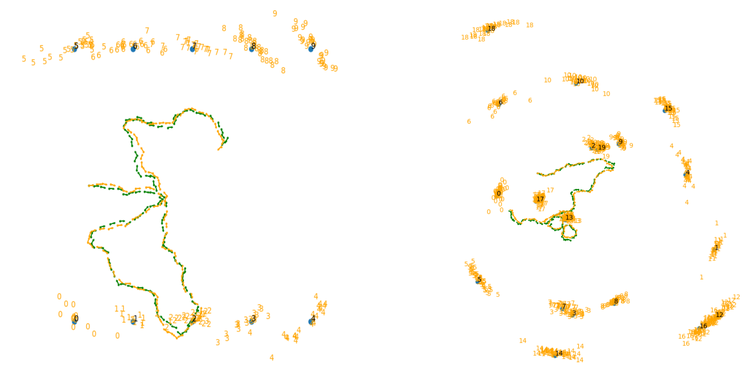

Figure 3. Two localization runs with different map layouts. In each, the green line is the ground truth of a simulated robot. The yellow line is the estimated trajectory. The blue dots are landmarks and the yellow numbers are their observed relative positions. The left run assumes the noise parameters are known and the right one places a weak prior on the noise. These data were generated with WebPPL[4].

Conclusion and Future Work

Practical inference tasks, such as SLAM can be performed using general purpose probabilistic programming languages such as Anglican[1]. While such formulations are unlikely to compete with specialized SLAM algorithms, we believe that the flexibility and adaptability of such an approach has great benefits in robotics. For example, it allows calibration of systems without additional external sensors. Next we would like to explore.

1. Explore connection between inference back ends and known SLAM implementations. For example, sequential Monte Carlo (SMC) queries on SLAM data essentially perform particle filtering techniques similar to FastSLAM[5].
2. Build a library of sensors, motors, and other components that allows roboticists to quickly build new mid-level models.
3. Use generative model to detect unexpected behavior and faults. This would allow robots to effectively use a vague description of their internal workings during runtime.

References

1. Wood, F., van de Meent, J. W., & Mansinghka, V. (2014). A New Approach to Probabilistic Programming Inference. . *International conference on Artificial Intelligence and Statistics* .
2. Thrun, S.; Burgard, W.; Fox, D. (2005). *Probabilistic Robotics*. Cambridge: The MIT Press.
3. Thrun, S. Montemerlo, M. (2006) The GraphSLAM Algorithm with Applications to Large-Scale Mapping of Urban Structures. The International Journal Of Robotics Research.
4. *N. D. Goodman , A. Stuhlmüller. Design and Implementation of Probabilistic Programming Languages.*
5. Montemerlo, M.; Thrun, S.; Koller, D.; Wegbreit, B. (2002). FastSLAM: A factored solution to the simultaneous localization and mapping problem. *AAAI National Conference on Artificial Intelligence*.