Programming Languages for Embedded Systems

Sebastian Thrun, Frank Pfenning, Sungwoo Park, and Jonathan Moody

Problem: Sensor-based embedded systems have to cope with uncertainty. Uncertainty arises from sensor limitations, environment dynamics, and lack of knowledge at the design time of embedded system.

Currently, we lack frameworks for developing software for embedded systems. Our hypothesis is that probabilistic computation and learning are key to building robust embedded software. Furthermore, we believe that these principles should be offered at the programming language level, to maximize their usability. Thus, our goal is to devise new programming languages that *seamlessly integrate probabilistic computation and learning.*

Impact: This project, if successful, could revolutionize the way we program computers. Today, computers are programmed through keyboards. Tomorrow we might teach them, or let them teach themselves through interaction with their environment. Moreover, to date we treat all information inside a computer as certain. Tomorrow's computer software might compute with multiple hypothesis, yielding more robust results in the face of uncertainty.

State of the Art: Current computer programming languages do not support any of the mechanisms studied in this project.

Probabilistic means of computing have been studied in applied statistics and AI, most notably in the context of graphical models [7]. Graphical models specify probability distributions in high-dimensional spaces that can be marginalized efficiently. They can be viewed of a declarative approach to probabilistic computation. Other relevant methods are Kalman filters [4], Hidden Markov Models [8], and particle filters [1], which are probabilistic methods for state tracking.

Robotics has developed a range of programming languages [2, 3, 5, 9] which are primarily concerned with concurrency, modular programming, and event-driven computation. Finally, methods have been proposed for learning program code from data, such as the genetic programming approach [6].

Approach: Our approach contains two basic ideas. First, we make probability distribution as usable as an elementary data types. Probability distributions can be defined over arbitrary data types. By moving from a conventional data type to a probabilistic one, a certain piece of information is turned into an uncertain one. Our belief is that computing with uncertain information is completely analogous to computing with certain information. Our approach, thus, provides arithmetic operations defined over probability densities, type conversion mechanisms, and even complex concepts such as iterations, recursion, and function calls.

Second, we enable programmers to embed function approximators in their code, such as artificial neural networks. We provide the necessary infrastructure to tune these function approximators with data, even for "distal" target signals. Function approximation offers an viable alternative to programming: Instead of programming a function by hand, the programmer can equally decide to train his software to perform the desired function. The seamless integration seeks to attain the best of both worlds: learning and programming, providing the programmer with the method of choice.

At present, we possess one implemented proto-type, which is implemented on top of C++ [10, 11]. A second proto-type is implemented on top of the programming language ML, but it currently lacks support for learning. We have successfully demonstrated that robust probabilistic robot software is two orders of magnitude more compact when developed using our new programming languages. For example, we have implemented a program for a mobile mail delivery robot with a gesture interface in only 140 lines of code

(a)

```
main(){

  // ============= Declarations ==============
  fa<vector<double>,        prob<double> > netSonar(twoLayerNeuronet, 5);
  fa<prob<vector<double> >, prob<double> > netX(twoLayerNeuronet, 5),
                                           netY(twoLayerNeuronet, 5);
  fa<vector<double>,        prob<int> >    netLeft(twoLayerNeuronet, 5),
                                           netRight(twoLayerNeuronet, 5);
  prob<double>          alpha, alphaLocal, probRotation;
  prob<double>          thetaLocal, theta, transVel, rotVel;
  prob<double>          x, xLocal, y, yLocal, probTransl;
  prob<int>             coin = {{0, 0.5}, {1. 0.5}};
  prob<int>             gestureLeft, gestureRight;
  prob<vector<double> > newSonar(2);
  double                alphaTarget, scan[24], image[300];
  double                xTarget, yTarget, xGoal, yGoal, t, v;
  struct                { double rotation, transl; } odometryData;
  struct                { double x, y, dir; } stack[3];
  int                   targetLeft, targetRight;
  int                   numGoals = 0, activeGoal;

  // ============= Initialization ==============
  alpha = UNIFORM1D(0.0, M_PI);
  theta = UNIFORM1D(0.0, M_PI);
  x = XHOME; y = YHOME;

  // ============= Main Loop ==============

  for (;;){

    // ---------------- Localization ----------------
    GETSONAR(scan);                    // get sonar scan
    alphaLocal = netSonar.eval(scan) * M_PI;
    alpha = multiply(alpha, alphaLocal);
    probloop(alphaLocal, coin; thetaLocal){
      if (coin)
        thetaLocal = alphaLocal;
      else
        thetaLocal = alphaLocal + M_PI;
    }
    theta = multiply(theta, thetaLocal);// robot's orientation
    probloop(theta; newSonar){
      int i = int(theta / M_PI * 12.0);
      int j = (i + 12) % 24;
      if (scan[i] < 300.0) newSonar[0] = scan[i];
      if (scan[j] < 300.0) newSonar[1] = scan[j];
    }
    xLocal = netX.eval(newSonar);
    yLocal = netY.eval(newSonar);
    x = multiply(x, xLocal);            // robot's x coordinate
    y = multiply(y, yLocal);            // robot's y coordinate

    GETODOM(&odometryData);             // get odometry data
    probRotation = prob<double>(odometryData.rotation)
       + NORMAL1D(0.0, 0.1 * fabs(odometryData.rotation));
    alpha += probRotation;
    if (alpha <   0.0) alpha += M_PI;
    if (alpha >= M_PI) alpha -= M_PI;
    theta += probRotation;              // new orientation
    if (theta <  0.0)        theta += 2.0 * M_PI;
    if (theta >= 2.0 * M_PI) theta -= 2.0 * M_PI;
    theta = probtrunc(theta, 0.01);
    probTransl = (prob<double>) odometryData.transl
      + NORMAL1D(0.0, 0.1 * fabs(odometryData.transl));
    x = x + probTransl * cos(theta);
    y = y + probTransl * sin(theta);
    x.truncate(0.01);                   // new x coordinate
    y.truncate(0.01);                   // new y coordinate

    // ------------ Gesture Interface & Scheduler ------------
    GETIMAGE(image);
    gestureLeft  = netLeft.eval(image);
    gestureRight = netRight.eval(image);
    if (numGoals == 0){                       // wait for gesture
      if (double(gestureLeft) > 0.5){
        stack[numGoals  ].x   = XA;    // location A on stack
        stack[numGoals  ].y   = YA;
        stack[numGoals++].dir = 1.0;
      }
      if (double(gestureRight) > 0.5){
        stack[numGoals  ].x   = XB;    // location B on stack
        stack[numGoals  ].y   = YB;
        stack[numGoals++].dir = 1.0;
      }
      if (numGoals > 0){
        stack[numGoals  ].x   = XHOME; // HOME location on stack
        stack[numGoals  ].y   = YHOME;
        stack[numGoals++].dir = -1.0;
        activeGoal = 0;
      }
    }
    else if (stack[activeGoal].dir *   // reached a goal?
             (double(y) - stack[activeGoal].y) > 0.0){
      SETVEL(0, 0);                          // stop robot
      activeGoal = (activeGoal + 1) % depth;
      if (activeGoal)
        for (HORN(); !GETBUTTON(); );  // wait for button
      else
        numGoals = 0;
    }

    else{ // ---------------- Navigation ----------------
      xGoal = stack[activeGoal].x;
      yGoal = stack[activeGoal].y;
      probloop(theta, x, y, xGoal, yGoal;
               transVel, rotVel){
        double thetaGoal = atan2(y - yGoal, x - xGoal);
        double thetaDiff = thetaGoal - theta; // location of goal
        if (thetaDiff < -M_PI) thetaDiff += 2.0 * M_PI;
        if (thetaDiff >  M_PI) thetaDiff -= 2.0 * M_PI;
        if (thetaDiff < 0.0)
          rotVel =  MAXROTVEL;         // rotate left
        else
          rotVel = -MAXROTVEL;         // rotate right
        if (fabs(thetaDiff) > 0.25 * M_PI)
          transVel = 0;                // no translation
        else
          transVel = MAXTRANSVEL;      // go ahead
      }
      v = double(rotVel);                    // convert to double
      t = double(transVel);                  // convert to double
      if (sonar[0] < 15.0 || sonar[23] < 15.0) t = 0.0;
      SETVEL(t, v);                          // set velocity
    }

    // ---------------- Training ----------------
    GETTARGET(&alphaTarget);              // these command are
    alpha.train(alphaTarget);             // only enabled during
    GETTARGET(&xTarget);                  // training. They are
    x.train(xTarget);                     // removed afterwards.
    GETTARGET(&yTarget);
    y.train(yTarget);
    GETTARGET(&targetLeft);
    gestureLeft.train(targetLeft);
    GETTARGET(&targetRight);
    gestureRight.train(targetRight);
  }
}
```

x

(b)

(c)

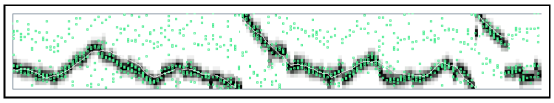

Figure 1: (a) A program in CES, our probabilistic extension of C++, for controlling the robot shown in (b) to perform mail delivery at CMU. (b) Amelia, a RWI B21 robot with sonars and cameras. (c) Tracking the orientation of the robot using probabilistic variables and learning. The solid line is the target signal, and the gray ares is the posterior probability calculated by the program.

(plus two hours of training), where comparable implementations require tens of thousands of lines of code (see Figure 1).

Future Work: In the future, we plan to solidify our implementations, develop software developing and debugging environment, and disseminate the results of our research to a user community.

Selected References:

[1] A. Doucet, J.F.G. de Freitas, and N.J. Gordon, editors. *Sequential Monte Carlo Methods In Practice*. Springer, 2001.

[2] R.J. Firby. An investigation into reactive planning in complex domains. In *Proceedings of the National Conference on Artificial Intellgence (AAAI)*, pages 809–815. AAAI, 1987.

[3] E. Gat. Esl: A language for supporting robust plan execution in embedded autonomous agents. In *Working notes of the AAAI Fall Symposium on Plan Execution*, Boston, MA, 1996. AAAI.

[4] R. E. Kalman. A new approach to linear filtering and prediction problems. *Trans. ASME, Journal of Basic Engineering*, 82:35–45, 1960.

[5] K. Konolige. Colbert: A language for reactive control in saphira. In *KI-97: Advances in Artificial Intelligence*, LNAI, pages 31–52. Springer verlag, 1997.

[6] J. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, 1992.

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

[8] L.R. Rabiner and B.H. Juang. An introduction to hidden markov models. *IEEE ASSP Magazine*, 3(1):4–16, 1986.

[9] R. Simmons and D. Apfelbaum. A task description language for robot control. In *Proceedings of the Conference on Intelligent Robots and Systems (IROS)*, Victoria, CA, 1998.

[10] S. Thrun. A framework for programming embedded systems: Initial design and results. Technical Report CMU-CS-98-142, Carnegie Mellon University, Computer Science Department, Pittsburgh, PA, 1998.

[11] S. Thrun. Towards programming tools for robots that integrate probabilistic computation and learning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, San Francisco, CA, 2000. IEEE.