Dude, Where's My Robot? A Localization Challenge for Undergraduate Robotics

Paul Ruvolo
Paul.Ruvolo@olin.edu
Olin College of Engineering

Abstract

I present a robotics localization challenge based on the inexpensive Neato XV robotic vacuum cleaner platform. The challenge teaches skills such as computational modeling, probabilistic inference, efficiency vs. accuracy tradeoffs, debugging, parameter tuning, and benchmarking of algorithmic performance. Rather than allowing students to pursue any localization algorithm of their choosing, here, I propose a challenge structured around the particle filter family of algorithms. This additional scaffolding allows students at all levels to successfully implement one approach to the challenge, while providing enough flexibility and richness to enable students to pursue their own creative ideas. Additionally, I provide infrastructure for automatic evaluation of systems through the collection of ground truth robot location data via ceiling-mounted location tags that are automatically scanned using an upward facing camera attached to the robot. The robot and supporting hardware can be purchased for under $400 dollars, and the challenge can even be run without any robots at all using a set of recorded sensor traces.

The Challenge

Robot localization is one of the most fundamental tasks in robotics, and accurately determining the location of a robot is a necessary first step for other robotics tasks such as mapping and path planning. While this topic is often part of undergraduate robotics curricula, here, I bring two important contributions. First, I provide a scaffolded assignment whereby students are guided through the process of implementing a robot localization algorithm on a real robot using the popular Robot Operating System (ROS) (Quigley et al. 2009). Second, I provide an infrastructure whereby students' algorithms can be tested quantitatively. This infrastructure enables one to structure the assignment as a challenge in which students attempt to infer, as accurately as possible, the robot's position from sensor data and motor commands.

The challenge emphasizes a rich set of AI skills, including probabilistic reasoning, approximate inference, sensor modeling, and parameter tuning. Further, the challenge places a strong emphasis on iterative design, testing, and debugging. Rather than allowing students to pursue any approach to the challenge they desire, I structure the robot localization challenge around a particular family of algorithms based on the particle filter (Ristic, Arulampalam, and Gordon 2004). This high degree of scaffolding is important due to the challenge's position in the early part of the semester, and the structure allows for a rich set of cohesive in-class activities built around explaining and successfully implementing the particle filter.

Student Audience

The challenge is designed for undergraduates who have achieved proficiency, though not necessarily fluency, with basic programming. In its current formulation, the challenge requires knowledge of Python, although the challenge could be restructured around C++ without too much effort. At Olin College, all enrolled students are required to have taken at least one full semester course in computer science (ours is roughly equivalent to CS1.5). This assignment sits in the beginning of a full-semester introduction to robotics class. Some basic knowledge of probability (e.g., distributions and conditional probability) is helpful, but not assumed.

Problem Formulation

Students are challenged to determine the current pose of a ground vehicle (location and orientation) given its approximate starting pose, a static map of the environment (represented as an occupancy grid), and a time series of sensor measurements and motor commands. Following (Thrun, Burgard, and Fox 2005), I frame robot localization as a Bayesian inference problem. First, I define the following notation.

$x_t \triangleq$ robot pose, i.e. location and orientation, at time t

$u_t \triangleq$ motor command, i.e. intended (or measured) linear and angular velocity, at time t

$z_t \triangleq$ sensory data at time t

Additionally, I use the shorthand $u_{1:t}$, for example, to indicate the sequence of variables $u_1, \ldots, u_t$. The variables $u_{1:t}$ and $z_{1:t}$ are assumed to be observable (i.e. they can be directly measured). Formally, the challenge for the students can be stated as inferring the variables $x_{1:t}$ (the time series of robot poses) given these observed variables. That is, we seek to compute the following probability distribution.

$$p(x_t | u_{1:t}, z_{1:t}) \quad (1)$$

A recursive algorithm for computing this probability distribution at time t given the same probability distribution at time $t-1$ is given by the well known Bayes filter. The recursion is started by specifying an initial distribution over the robot's pose at time 0. A detailed derivation of the Bayes filter can be found in (Thrun, Burgard, and Fox 2005). Presenting the derivation of the Bayes filter in class allows for the introduction of a number of important concepts in probability theory, including conditional probability, Bayes rule, marginalization, and the product rule. The final recursion is given as

$$p(x_t|u_{1:t}, z_{1:t}) \propto p(z_t|x_t) \sum_{x_{t-1}} p(x_{t-1}|u_{1:t-1}, z_{1:t-1}) p(x_t|x_{t-1}, u_t) \ . \quad (2)$$

The formulation above assumes that the pose, x_t, is defined over a discrete space, however, the continuous state filter can be obtained by replacing the summation with an integral.

While Equation 2 may be daunting for students without a background in probabilistic reasoning, its logic can be explained conceptually. The equation states that the degree to which one should believe the robot is in a particular state at time t is proportional to the product of the degree to which the current sensor data is consistent with the robot being in that state and the likelihood that the robot arrived in that state given the executed motor command. This formula highlights that in order to solve the robot localization challenge, students must specify two components: (1) a motion model for the robot that specifies the likelihood of the robot achieving a particular pose given its starting pose and a motor command, $p(x_t|x_{t-1}, u_t)$, and (2) a sensor model that specifies the likelihood of a particular sensor reading given a robot pose, $p(z_t|x_t)$.

At this point in the presentation of the algorithm the students are typically very excited that such a seemingly difficult problem has yielded relatively easily to the application of a few straightforward rules of probability. Unfortunately, the recursion is intractable to compute for problems with large numbers of possible poses. To get students to understand this, I ask them to determine the computational complexity of the recursion in Equation 2. The complexity of the update is $O(n^2)$ where n is the number of possible poses of the robot. In order to provide sufficient resolution over an environment, even for the simple case of a ground vehicle moving in a planar environment, one would need to use a very large n (e.g., to represent various possible combinations of x-position, y-position, and orientation).

The computational complexity of Equation 2 motivates the presentation of the particle filter algorithm. The key insight that motivates this algorithm is that the distribution $p(x_t|z_{1:t}, u_{1:t})$ is approximately zero for all but a tiny fraction of all possible poses. That is, there is only a small set of poses that have a non-negligible amount of probability mass associated with them at any point in time. Instead of tracking the entire probability distribution as suggested by Equation 2 we can instead track a small number of potential poses, or particles, over time. The particle filter algorithm gives us a process to encourage this tracked set of poses to be ones with high probability. The most common implementation of the particle filter (and the one I suggest to students) is as follows.

1. Sample the initial set of m particles $\hat{x}_0^{(1)}, \ldots, \hat{x}_0^{(m)}$ from a given initial pose distribution.
2. Sample a preliminary set of particles at time t, $\tilde{x}_t^{(1)}, \ldots, \tilde{x}_t^{(m)}$, from the motion model $p(\tilde{x}_t^{(i)}|\hat{x}_{t-1}^{(i)}, u_t)$
3. Assign each particle a weight given the sensor model $w_i = \frac{p(z_t|\tilde{x}_t^{(i)})}{\sum_{j=1}^{m} p(z_t|\tilde{x}_t^{(j)})}$
4. Sample the final set of particles at time t, $\hat{x}_t^{(1)}, \ldots, \hat{x}_t^{(m)}$, from the preliminary set of particles with probability of selecting the ith initial particle, $\tilde{x}_t^{(i)}$, given by w_i.
5. Given a new motor command, u_{t+1}, and sensor reading, z_{t+1}, go to step 2.

From a computational efficiency point of view we have made a major improvement. The new algorithm has an update time of $O(m)$ (recall that m is the number of particles). There are numerous alternatives to this specific particle filter variant, however, for this assignment I provide starter code that makes it easy to implement a version of the algorithm described above. If students desire, they can implement a more advanced version.

Given a student uses the particle filter as articulated above, the bulk of the challenge becomes specifying the motion and sensor models. Determining each of these components presents a number of difficult modeling decisions that tradeoff computational cost, accuracy, and ease of implementation.

Motion Model In the version of the challenge that I run at Olin College, I assume a differential drive robot moving around in a planar environment (see the "Robotics Platform" section). Further, the motor command at time t is given to students as a measured linear and angular velocity over some window of time (the velocities are estimated using wheel encoders). Updating the position of each particle given u_t is straightforward, however, an important part of the motion model is the noise encoded by the probability distribution $p(x_t|x_{t-1}, u_t)$. The degree of noise in the motion model serves to capture both the degree of uncertainty in our estimate of the robot's position and also to help the particle filter recover from situations in which none of the particles are close to the true pose. In practice, to achieve maximum accuracy the parameters of the motion model must be tuned empirically.

Sensor Model The Neato robotic vacuum cleaner provides a scanning planar LIDAR (the scan plane is parallel to the ground plane). Specifically, the robot provides an estimate of the distance to the closest obstacle across a range of bearings updated at 5 Hz. Therefore, the students' sensor models should specify $p(z_t|x_t)$ where z_t is the laser scan observed at time t, x_t is the robot pose, and the map of the environment (assumed to be known ahead of time) is implicitly conditioned in the probability distribution (in practice students create the map using one of the popular 2D SLAM

ROS packages. I have had the best luck with the gmapping and hector_slam ROS packages).

The basic idea of the sensor model is to evaluate the likelihood of a scan reading given the hypothesized position of the robot and the map. One can think of a scan at a particular bearing as a detected obstacle. The intuition for a reasonable sensor model is that the probability of a measurement should be high when the detected obstacle is close to an obstacle in the map and low when it is far away from any known obstacle. For robustness, allowances should be made for sensor noise / error as well as encountering previously unknown (i.e. unmapped) obstacles. In practice, each bearing from the laser scan is assigned a probability independently, and the overall likelihood of the complete laser scan is computed by aggregating the individual probabilities (there are a number of possible options for doing this aggregation). I provide students with a number of suggested approaches to filling in the details of their sensor model that follow this general intuition, including the likelihood field approach and the ray tracing approach (see https://sites.google.com/site/dudewheresmyrobot/ for more information. Also, consult (Thrun, Burgard, and Fox 2005)).

Required Hardware and Software

There are two options available for running this challenge. The first is to create a setup similar to the one I use at Olin College (the details of which are provided in the remainder of this section). The second is to eschew the robots entirely and instead base the challenge on a series of recorded test cases. This second option is possible since the challenge is one of perception and not control. Therefore, the same run can be used repeatedly to evaluate many different approaches to the challenge. I have collected a number of these test cases as ROS bag files which can be downloaded at https://sites.google.com/site/dudewheresmyrobot/.

Robotics Platform and Supporting Hardware

While the particle filter is a very flexible algorithm that can be adapted to many different robots, the provided version of this assignment assumes a robotics platform with the following capabilities.

- A two-wheeled differential drive robot.
- A planar scanning LIDAR (the greater the field-of-view, the better).
- Odometry of some type (e.g., using wheel encoders).
- (optional) An upward facing video camera (only needed for collection of ground truth validation data).

At Olin College we use the Neato XV vacuum cleaner as our robotics platform. There are many different Neato XV models available, and all of the ones that I tested have worked equally well (I have tried the XV-11, XV-12 and XV-21). The Neato is ideally-suited for this assignment as it is a two-wheeled differential drive robot, has a 360 degree rotating LIDAR, and is inexpensive (a Neato XV can be purchased for under $300). The LIDAR provides the distance to the closest obstacle with an angular resolution of 1 degree, a sampling rate of 5 Hz, and a usable range of approximately 0.5m to 5m. Additionally, the Neato has encoders to measure the rotation of each wheel, which can be integrated over time to provide the necessary odometry.

Figure 1: A Neato XV-21 with attached Raspberry Pi 2. Here, the camera is mounted forward-facing, however, for collection of ground truth data the camera should be mounted upward-facing.

The Neato is controlled via a USB serial interface, and thus needs an attached computer in order to issue motor commands and fetch sensor data. For this purpose I use a Raspberry Pi 2 (in the past I have successfully used a Raspberry Pi B+), although a laptop will also work. For the Olin setup, the Raspberry Pi acts to shuttle sensor data and motor commands between the Neato and a base station computer that does the heavy-duty computation. All of our students have a standard-issue laptop that they use as their base station computer, however, most any PC will do (provided it can run ROS and has some networking capability).

In order to allow quantitative assessment of a robot localization system one must have some way of obtaining ground truth location data (i.e. the robot's actual position). In my class students obtain ground truth location data using an upward facing video camera. For this purpose I use the Raspberry Pi camera module and ceiling mounted AR tags (Kato, Billinghurst, and Poupyrev 2000) (see Figure 2). Specifically, I use two foot by two foot AR tags and mount them to the ceiling at known locations. The number of AR tags one needs to cover a space depends on the ceiling height and the camera's field of view. Using the Raspberry Pi camera module, at Olin College a single tag can cover a 6 foot by 6 foot area.

Software

Students use the popular Robot Operating System (ROS) (Quigley et al. 2009) to run the code on the base station computer. This software provides an abstraction layer over the basic sensors and actuators of the Neato, robust visualization and debugging capabilities, and built-in support for mapping (recall that a map is needed as a precondition to the challenge). Additionally, the students use ROS to pro-

Figure 2: A ceiling mounted AR tag. These tags can provide accurate positioning information using the *ar_pose* ROS package. If the position of the tag is known within the room, one can obtain ground truth location data for a robot based on an image of the tag captured from a camera mounted on the robot.

cess the images from the upward facing camera to provide ground truth pose information using a combination of the *ar_pose* library for detecting the AR tags (see Figure 2) and a ROS node that I wrote to transform the tag detections into the actual location of the robot in the room. All software that runs on the base-station computer can be found at https://sites.google.com/site/dudewheresmyrobot/.

Students program their entry to the challenge as a ROS node. ROS supports a number of programming languages, however, full support is limited to Python and C++. In my course I use Python as the default language, and I require that all student submissions to the challenge are written in Python. C++ could also be used, and it would have some advantages over Python in terms of computational efficiency.

In addition to the software that runs on the base station computer, some software must be installed on the Raspberry Pi 2 that is connected directly to the Neato. The easiest way to install this software is to download a prebuilt Raspberry Pi 2 SD card image from the supplementary website. Once the image has been downloaded and cloned onto a micro SD card, the only tweak necessary is to change the wireless network configuration in */etc/wpa_supplicant/wpa_supplicant.conf*. The software that runs on the Raspberry Pi is part of the robot platform given to the students. The students are not expected to, nor allowed to, modify this code.

AI Programming Skills and Challenges

The challenge emphasizes a number of key AI skills. The first major area of emphasis is computational modeling and abstraction. The specification of the sensor model and the motion model require students to make modeling decisions that are justified using considerations such as accuracy, efficiency, and ease of implementation. For students that have not done this sort of thing before, the process of specifying a model that they know is wrong (or at least not fully accurate) can seem uncomfortable at first, but learning how to make effective modeling decisions is a key outcome of this assignment. Tied into these modeling decisions are a number of AI topics related to probabilistic reasoning, including Bayes rule, conditional independence, Gaussian distributions, marginalization, and the product rule of probability.

A second major area of AI emphasized by this challenge is parameter tuning and model refinement. For students accustomed to running code that can be determined to either be correct or incorrect using straightforward unit tests, evaluating a complex robotic perception algorithm is daunting. I suggest an iterative approach to the challenge. Students are encouraged to start with a simple model, debug the model to make sure it is implemented correctly, tune any model parameters, and then refine the model based on their experiments. The parameter tuning step can be approached either by evaluating the performance of the algorithm by eye, or through automatic evaluation using ground truth position data (e.g., obtained using the AR tag-based positioned system). Another useful technique for model testing and refinement is to record a series of test cases consisting of runs of the robot through an environment. These test cases can be used to refine the student's algorithm. Designing a good set of test cases is another key component of the challenge.

In order for students to be as effective as possible in debugging their system, they are encouraged to move beyond the standard print statement debugging common in computer science courses. The print statement approach becomes infeasible given the sheer amount of data that must be tracked over time. Instead (or in addition to), students are encouraged to use visualization (e.g., of particle locations, particle weights, etc.) as a key component of their debugging process. ROS's rviz package is perfect for this.

Experiences Running the Challenge

As a disclaimer, I have not yet run this assignment as a challenge (i.e. a quantitatively scored competition), however, that is on tap for the Spring 2017 semester. Further, I developed the ground truth positioning system midway through the last iteration of the course, and thus while it is well-tested, it was not available when the students did the robot localization assignment (recall that the localization assignment is at the beginning of the semester). Given these two caveats, I will provide my experiences running this as a conventional assignment during two iterations of the course. Extending this assignment to a challenge format would be relatively straightforward, requiring the specification of a performance metric (e.g., root mean squared error) and a set of representative test cases that would remain hidden to the students until the end of the challenge.

This challenge is one ?hat emphasizes skill-building, mastery, and scaffolding rather than unfettered creativity. For many students this is their first experience implementing a computer program that interfaces meaningfully with the "physical world" and all of the messiness inherent therein. Most students embrace this aspect of the challenge wholeheartedly, however, some students take a while to learn to successfully deal with the high degree of complexity that

comes with robots, unpredictable environments, and imperfect sensors. For students (and especially those that are having difficulty wrapping their heads around robotics) the particle filter algorithm provides a useful scaffolding and structure to approach the challenge. In my experience, students that are struggling can implement the scaffolded approach to the particle filter and students that are more comfortable are able to try more approaches as well as innovate their own approaches to the challenge. Student submissions vary in all sorts of dimensions. Typical twists that I have seen are simplified sensor models that allow for the usage of high numbers of particles, modifications of the standard particle filter to allow for re-initialization when none of the particles are consistent with the observed scan data, and motion models that incorporate noise that considers the physics of differential drive robots.

Student response to the assignment has been quite positive. Students appreciated the pedagogical role that the assignment plays in the course, specifically, providing a scaffolded walkthrough of the process of learning about, implementing, and testing an algorithm. The students follow this process repeatedly throughout the course in a progressively more self-directed fashion.

Conclusion

Our robot localization challenge provides a flexible framework through which many useful AI and general software engineering skills can be taught. The challenge emphasizes learning about algorithms through careful scaffolding, however, the challenge could certainly be made more open-ended. Given more time or a different student audience, students could research various approaches to localization, implement these algorithms, and evaluate them (rather than focusing on the particle filter).

Additional details, including a fun interactive class activity to explain the particle filter on a conceptual level, a detailed derivation of the Bayes filter along with an example adapted from (Thrun, Burgard, and Fox 2005), starter code, detailed diagrams, a step-by-step guide to implementing a base version of the challenge, and code for creating an occupancy grid map of a room can be found at https://sites.google.com/site/dudewheresmyrobot/.

References

Kato, I. P. H.; Billinghurst, M.; and Poupyrev, I. 2000. Artoolkit user manual, version 2.33. *Human Interface Technology Lab, University of Washington* 2.

Quigley, M.; Conley, K.; Gerkey, B.; Faust, J.; Foote, T.; Leibs, J.; Wheeler, R.; and Ng, A. Y. 2009. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, 5. Kobe, Japan.

Ristic, B.; Arulampalam, S.; and Gordon, N. 2004. *Beyond the Kalman filter: Particle filters for tracking applications*, volume 685. Artech house Boston.

Thrun, S.; Burgard, W.; and Fox, D. 2005. *Probabilistic robotics*. MIT press.