PIONEER IV

Practicum Autonome Systemen

Erick Wilts #1635085

Matthijs Dorst #1380982

10/6/2010

Contents

Introduction

The problem of localization is essential to many real world tasks a robot might have to perform. Knowing your exact position relative to the world is ever important – whether it entails placing a cap exactly on a leaking oil well, entering the correct room in an office complex or finding the correct pallet in a storage warehouse or one of the many other tasks suited for robots. It is no surprise therefore that a great many methods exist to solve this problem. We have implemented a solution based on a so called particle filter for a Pioneer robot.

By combining information on visible fixed landmarks and robot odometry, random particles are updated with the estimated position and angle of the robot at each step. If observations and odometry were perfect this would be a fairly trivial exercise, sadly they are not: odometry data contains noise and thus the robots position becomes uncertain. Furthermore, errors stack: at each consequential step the uncertainty increases.

Observing landmarks can (to a degree) solve this problem: the probability the robot would observe a certain landmark when facing the opposite direction is fairly slim. However, observing a single landmark only means the robot must be facing it – and this it can do from a great many directions. To make matters worse, landmarks are not unique: observing a single green pillar does not really help much when there are a dozen green pillars around. Multiple and frequent observations are required to maintain a low error and a high probability of the robot being in the estimated position.

Model

The Particle Filter

The particle filter is an object containing numerous particles. Particles have a position, a direction and a probability. All particles pretend to be a Pioneer robot in a certain position and facing a certain direction, and all of them have a probability that tells them what the odds are they are at the same location (that is, having the same vector) as the real Pioneer.

Somehow all particles must be able to converge on the position of the robot. Therefore, the particles with a low probability of representing the location of the Pioneer should be forgotten, while the particles with a high probability should be reproduced. This last word immediately reminds one of a genetic algorithm, and that is the idea this particle filter is based upon.

All particles are updated each time step, being replaced over the map according to the direction, angle and speed of the robot. Then, for each particle the perception of the robot is compared with a perception that particle could have (perception being a vector of perceived colored pillars). Now every particle has a probability they can be reproduced. In order to do this, a new particle field is created, and the particles are redistributed using Stochastic Universal Sampling, which will be explained in the Implementation section.

The driving mechanism

The obstacle avoidance is implemented with an alternative to the Braitenberg ‘avoid’ behavior, with an increasing weight for the sensors towards the sides of the robot.

Landmark detection

If the robot wants to know where in the map it is located, it has to know where the landmarks are, and therefore it needs to detect them. Unlike previous experiments, pillars are no longer unique and observing multiple pillars of identical color simultaneously is therefore essential. A robot mounted camera feeds an image analysis algorithm designed to detect areas of a particular color. The two largest connected areas are presumed to represent pillars of the specific color, provided they appear large enough and exist in a viable relative location.

Implementation

The Particle Filter

At the start of the program, all particles are initiated randomly. Each particle represents the position and the direction of the robot. Each time step (in the case of this program 100 ms), the particles are updated according to their direction and the speed of the robot.

The variables per particle are updated as follows:

$$x := (\, x + \sigma_{trans} \,) * \cos(\alpha)$$
$$y := (y + \sigma_{trans} \,) * \sin(\alpha)$$

$$\alpha := \alpha + \sigma_{rot}$$

Here, x and y represent the position of the particle on the map, and α the direction of the particle; σ is a noise parameter for either the translational noise or the rotational noise. In the final test those values were respectively 0.3 and 0.4, but see the Results section.

Stochastic Universal Sampling

Stochastic Universal Sampling (SUS) is a form of Fitness-Proportionate Selection (FPS) that, as the name says, samples a new generation for a genetic algorithm universally. The fitness function for this genetic algorithm consists only of the probability for each particle. To universally distribute the particles, all probabilities will have to be put in a figurative roulette wheel (the nickname for FPS is roulette wheel selection). In normal FPS, this roulette wheel would be turned *n* times, *n* being the number of particles that should be created. But in SUS, the roulette wheel is divided in *n* equal parts, starting at a random offset in the first particle and making a particle in a new particle field each 'step'. Each step has the following size:

$$size_{step} = \frac{\sum p(particle)}{n_{particles}},$$

with *p(particle)* being the probability for a single particle (so the sum is the cumulative probability for all particles) and $n_{particles}$ being the total number of particles, in our runs always 1000.

The driving mechanism

All four front left sensors are connected to the right wheel inhibitionally, and the same goes for the right sensors and the left wheel. The sonar sensors have greater weights towards the sides, so that the robot turns faster when driving next to a wall and only decreases its speed when facing a wall.

Landmark detection

Since the algorithm yields better results with false negatives (detecting nothing when actually a target is in view) than false positives (detecting a target where there is none), the color map values are set to a narrow range:

Color	Red (min / max)		Green (min / max)		Blue (min / max)	
Yellow	120	225	90	210	0	50
Pink	195	256	0	89	10	100
Green	40	119	132	256	0	97

Table 1: selected color ranges (from 0 – 255 maximum range).

Furthermore, an area of a colormap that is smaller than 60 pixels is ignored, as well as detected 'pillars' that originate above the half of the screen (bottom of the surrounding rectangle > 160 pixels). As a result, we have no trouble with people wandering around in brightly colored shirts (at least, our robot does not).

Thanks to strict color map values, this implementation did not find any false negatives on any test run. It did however occasionally observe two pillars where there was actually one (when, for example, the middle part was unrecognizably light) – however, this caused no problems.

Results

Error over parameter space

Since the goal of the exercise was to implement a successful localization algorithm, we took great care to find optimal values for the available noise parameters. A resulting contour map for error in cm over $\delta_{distance}$ and $\boldsymbol{\delta_\theta}$ (the respective noise parameters for the distance and angle odometry) is given in figure 1. Note that we have kept δ_σ, the noise in our observations, at a constant value of 0.6 for all error map observations.

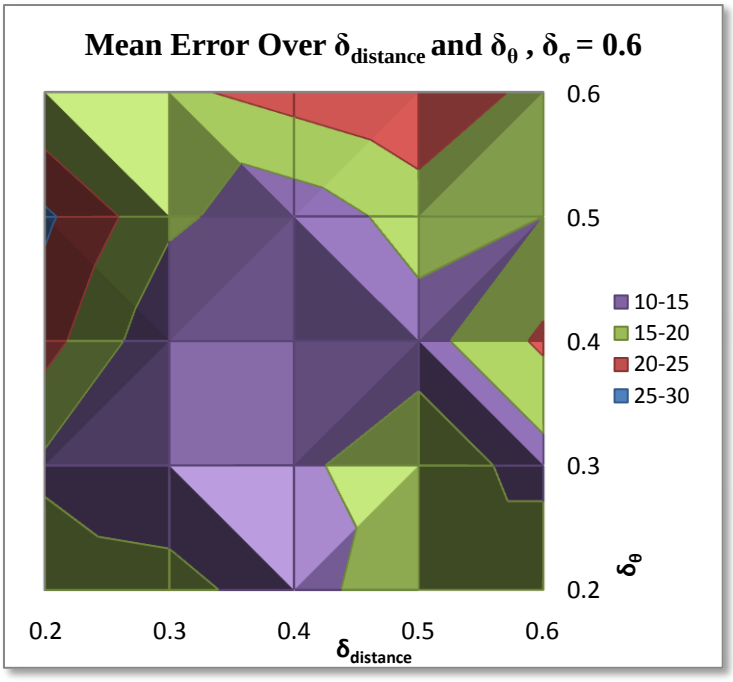

Figure 1: Mean Error over odometry noise parameters

As should be immediately clear, both noise parameters have direct influence on error. What is more, they are in a sense correlated in that one misconfigured noise parameter can cause a great error, even when the other parameter is set at an optimum value. For our final tests we have used $\delta_{distance} = 0.3$ and $\delta_\theta = 0.4$.

We observed less variation in error while adjusting δ_σ, as can be seen in figure 2. Except for very large or very small observation noise values the error remained largely constant at 12 centimeter on average.

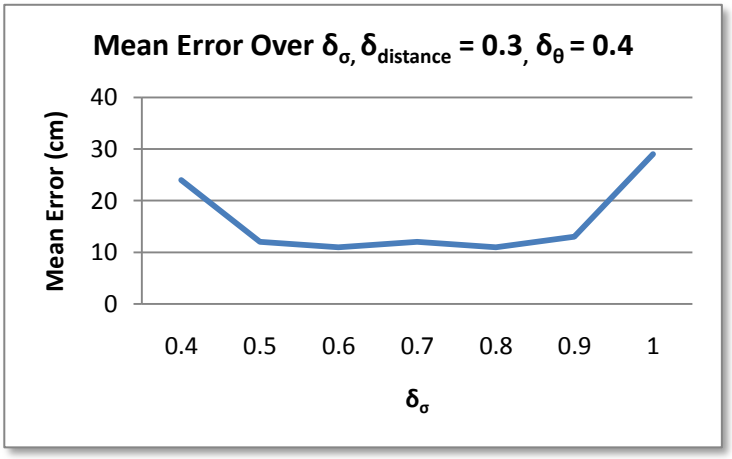

Figure 2: Mean Error over observation noise parameters

For our final runs we have used $\delta_\sigma = 0.6$ since this appeared to result in the most successful runs, while still yielding a fairly small mean error. It would appear that higher values decrease the filters ability to disperse particles during observations, which is acceptable during correct observations yet can be disastrous when the incorrect pillar is given priority (that is to say, when particles are converging on one pillar while the robot is actually heading towards a similarly colored pillar in another location).

Filter Performance

Overall the filter performed quite well, as can be derived from the obtained error scores. We noticed a typical dispersion pattern during periods with no landmark observations that followed the robot path to within a reasonable margin of error. A prototypical situation as observed by the ceiling-mounted observation camera is shown in figure 3. The green circle indicates the estimated position (as derived from odometry: the robot is currently facing the wall making no landmark observations), the white square denotes the actual robot position and each yellow dot represents a particle.

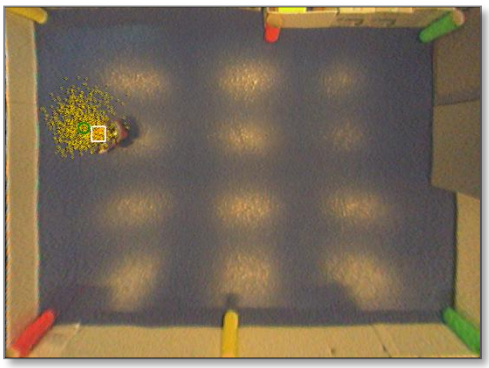

Figure 3: Particle Dispersion

Note the typical "wavering out" of particles during a period of no landmark observations: the noise in odometry causes particles to deviate from the robots actual position in a cloud surrounding the robot. The highest concentration of most-probable particles yields a position estimate roughly 20 centimeters from the robots actual position, which is quite acceptable considering the robot just drove for several meters without making landmark observations.

Discussion

Applicability & Performance

During our most successful trials, an average deviation of 11 centimeters (over 5 consecutive runs) was observed. This means the robot was, on average, no more than a hands length away from where it thought it was. Add the small uncertainty in the cameras verification algorithm, the fact that the camera observes the robot at an angle in the corners and the inherent difficulty in taking into account the effect battery voltage has on odometry error (which we estimate to account for a deviation of roughly 2 centimeters between optimal and worst-case circumstances) and the result becomes increasingly impressing. In a static world with unique landmarks on crucial locations a particle filter should be more than sufficient for nearly all applicable localization tasks.

However, we must also note that unique, easily recognizable, static landmarks are seldom encountered in the real world. Buildings might offer some help here in the form of doors, windows and corridor layouts, but these are prone to becoming blocked (with people, carts, etcetera) and not all trivial to detect. Furthermore, mapping an entire building would require considerable effort and each change to the buildings layout would mean remapping parts of it. While this could be done for a warehouse (where adding guidance markers in crucial locations is no insurmountable obstacle) it is less viable in for example a dynamic office environment (where cubicles are frequently rearranged). In short, if you have an accurate map, particle filters offer excellent performance - the trouble is getting that map! Of course, a SLAM architecture might partially solve this problem, possibly at the expense of performance.

Filter Improvements

We have debated the issue whether we should add additional functionality to the particle filter. One could make strong arguments for a converging-prevention algorithm and randomly distributed particles. We argue that no, with correct noise, there is no *need* for such measures. What is more, the benefit of preventing converging when it occurs incorrectly can just as easily be made at the expense of overall performance. This holds especially true when it is done incorrectly.

We argue that the particle filter in its proposed form is more than capable of preventing pre-optimal particle-converging – the substantial odometry and observation noise assures it. By simply adjusting noise parameters correctly the effect is both more elegant *and* more effective than adding random algorithms to prevent a problem that might not even occur!

It was possible to validate these claims to a degree: during trials comparing our implementation with 8 others, some implementing converging-prevention algorithms, some using random particles, others using both, our implementation consistently yielded lower mean errors over multiple trials than all other implementations. What is more, our implementation yielded the smallest deviation from that obtained error. That is, with this relatively simple implementation the error is small and constant, unlike other implementations yielding complex, unpredictable and not seldom far less accurate results.

In closing, it is easy for a programmer to simply start coding – but sometimes much more effective to optimize what you already have!