The Crunch Mobile Robot

Edward C. Kern
chris@jormungand.net
Department of Computer Science
Brown University, Providence, RI
Honors Thesis submitted 2005-05-05
Advisor: Tom Dean
Reader: Chad Jenkins

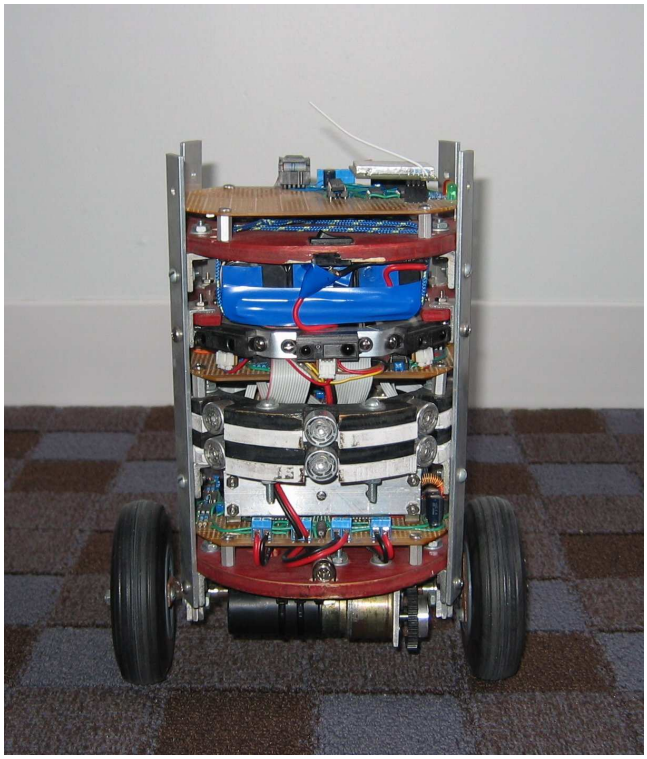

Figure 1: The Crunch Mobile Robot

ABSTRACT

Crunch is a two wheel inverted pendulum balancing robot equipped with 8 sonars and 8 infrared rangefinders and uses a FastSLAM-like[6] particle filter algorithm to build consistent maps. This paper presents a summary of the robot's design and capabilities as well as the particle filter algorithm used for consistent map building.

1. HARDWARE

1.1 Inverted Pendulum Balancing

Crunch has only two wheels and thus is inherently unstable. It is equipped with an angular rate sensor and accelerometer which, when combined using a complementary filter, measure the current pitch. The drive control microcontroller drives the wheels with a linear sum of the robot's pitch, pitch rate, wheel velocity, and error from commanded position. Forward/backward motion commands are introduced by subtracting the commanded velocity from the actual wheel velocity and moving the commanded position at the same velocity. Rotational motion commands are introduced by adding the command to one wheel velocity and subtracting it from the other; this does not alter the fore/aft balance.

1.2 Sensors

Crunch is equipped with two navigational sensor systems: sonar and infra-red. Each system has eight sensors at 45 degree angles 6 inches (sonar) and 8 inches (infra-red) above the ground and each sensor measures either a single distance or no return.

The principle difference between the sonar and infra-red is the field of detection. The sonar transducers are 40 khz piezo-electric type and have a beam pattern about 45 degrees wide. While this means they do not provide very high resolution, it also means that it is unlikely that a nearby object will not be detected by at least one sonar. The infra-red sensors, Sharp GP2D12s, have a beam width only a few degrees wide. As such they may not detect all nearby objects like the sonars but their tighter beam width allows them to resolve smaller environmental features.

There are also current and voltage sensors on the battery and charger circuits and current sensors on each of the two motor drivers.

1.3 Support Systems

Crunch is powered by a 24 cell NiMH AA-size battery pack. The balancing time constant of an inverted pendulum is proportional to the height of its center of gravity. Thus, in order to maximize stability, the batteries are located as high as possible, in contrast to traditional robot design.

On-board computations are done by two ATmega32 16MHz microcontrollers: one for inertial measurement, balancing, and movement and another to control the sonars and infra-red sensors. Both microcontroller serial ports are multiplexed into a 900MHz radio modem which transmits sensor data to and receives movement commands from a PC running Linux. The only behavior Crunch is capable of performing autonomously is balancing; all mapping and navigation algorithms are run on the PC.

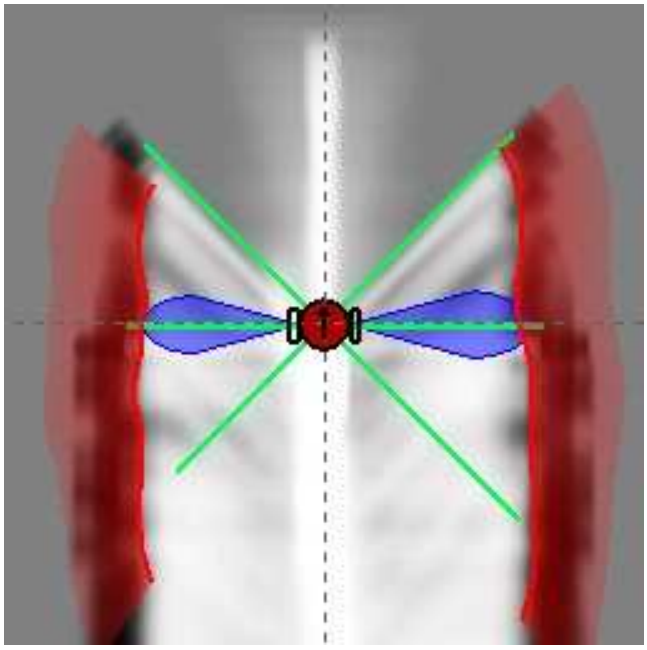

Figure 2: Sensor fusion in the local map: sonar measurements (blue cones) and infra-red rangefinder measurements (green rays) are integrated to find occupied (black) and unoccupied (white) regions which are then reduced to the location visible walls (red shading).

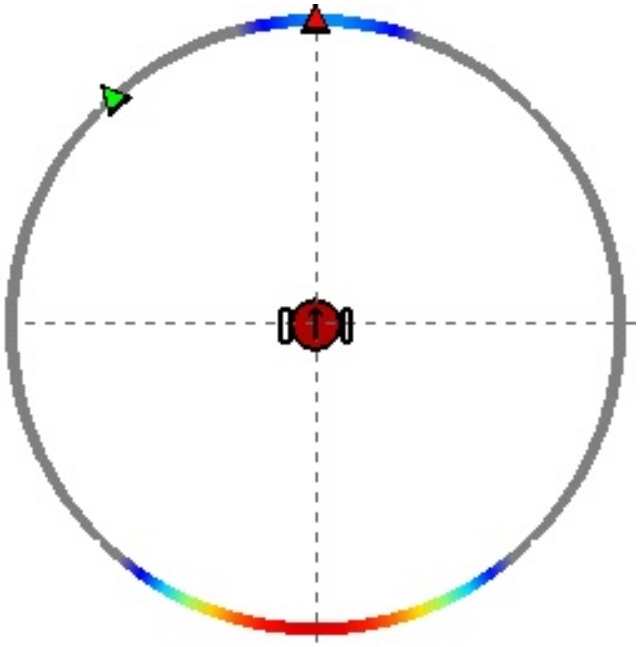

Figure 3: Vector field histogram corresponding to local map shown in Figure 2. The vector to the objective is shown as a green arrow; the valleys are identified by colored arcs on the gray circle; the commanded direction, shown as a red arrow, is selected from the valley closest to the objective.

2. MAPPING

2.1 Sensor Fusion

A local occupancy grid map is maintained for a 2x2 meter square centered on the robot. When infra-red and sonar data is received it is integrated on the local map in a pseudo-Bayesian manner. (Since the purpose of the map is to combine infra-red and sonar data, not to estimate probability of occupation based on past and current data, it allows new data to alter strong priors much more than Bayes rule would.) When dead-reckoning updates are received, the map is transformed and resampled at the new location; during resampling, areas outside the map are assumed to have neutral belief.

While an occupancy grid map works well for integrating sensor data, we eventually store the sensor data in a more compact form. Rays are traced from the center of the local map (ie. the robot) in all directions to the first point with occupancy probability greater than a threshold. These points are connected into adjacent sequences of points which represent the walls that are currently visible to the robot. We also apply the following filters: elimination of short curves, coalescing of adjacent curves separated by only a few infinite readings, and a Gaussian smoothing.

Due to the inability of the sensors to identify specific objects reliably, no effort is made to explicitly identify landmarks; the only representation is the general shape of objects.

2.2 Obstacle Avoidance

For local path planning Crunch uses the vector field histogram algorithm.[2] This is extremely easy since building the histogram is simply a radial sum of the local map and thus the extra computation needed is small.

The weights on the local map are scaled according to certainty of occupation squared and linearly decreasing from the center. The scaled weights are then summed radially, producing a distribution of obstacle density around the robot. "Valleys", contiguous arcs where the density is below a certain threshold, are identified as potential courses for the robot; a path through one of the valleys is selected based on the vector to a commanded objective waypoint.

Crunch has no global path planning and is generally operated manually.

2.3 Mapping

The reduction from occupancy grid to parametric curve is executed after every half meter of robot movement. We assume that within that distance dead reckoning error is small and can be ignored. The resulting curves are stored along with the spatial transformation the robot underwent from the last location. This results in a data structure which encodes the robot's movement path as well as the walls it observed at intervals along the path and forms the basis for our maps.

2.4 Map Likelihood

In order to build a consistent map it is necessary to apply corrections to the dead reckoning spatial transformations

along the path. In order to compute a most likely map it is first necessary to define likelihood in terms of the map representation we use. Our formulation has two components: the plausibility of the corrections and the self-affinity of the resulting map. The plausibility models confidence in the dead reckoning ability of the robot. While we know that the dead reckoning path is not correct, it is usually close to the true path; thus we use a Gaussian to compute the likelihood of a set of path corrections. The self-affinity of a map is a measure of how well the map matches itself in locations where the corrected path predicts that it crosses itself.

2.5 Map Affinity

To evaluate a candidate path it is necessary to compare the pairs of curves that the path predicts are near each other; we have experimented with two estimates for similarity of such curves.

The first map affinity function we used treats each curve as a set of individual line segments. To compare two sets of curves, each segment in the first curve is compared to each segment in the second curve. A sum is computed of the dot product of the normals of the curves multiplied by an exponentially decreasing function of the distance between them. Thus, curve segments which are nearby and oriented in the same direction contribute positively, whereas curves which are nearby and oriented in different directions contribute negatively. One serious flaw in this approach is that walls that cross at right angles produce zero affinity since the dot product is always zero; we would prefer that they be assigned a negative affinity.

The second map affinity function we used has properties more similar to traditional occupancy grid correlation functions. Test points are generated in a Gaussian pattern around the center of the two curves being compared. Each point is evaluated as being either inside or outside each of the two curves; the affinity is a function of the percent of points for which the curves agree. One advantage of this method is that the number of sample points can be reduced, increasing the variance of the affinity measurement but reducing computational time. Also it is possible to short-circuit the evaluation when it becomes clear after a small number of sample points that the affinity is very bad.

2.6 Local Search

For simple environment topologies – in general, ones without long cycles – a local search is sufficient for finding the most likely map. While the robot drives around extending the path, a background thread continuously generates random perturbations of the path and evaluates them, accepting improvements greedily.

Local search has a serious drawback in this application, however: the dimensionality of the search space is continuously growing. Thus as the path becomes longer, the amount of searching that can be done in any particular part of it given fixed processor time diminishes.

2.7 Particle Filter

The drawback of local search is that it stores only a single path hypothesis and after closing a large loop that one hypothesis will usually differ sufficiently from the true path that local search will not find the true path.

The particle set consists of several weighted hypothesis for the robot's path which together implicitly represent an arbitrary distribution. As the robot moves and its path is extended the particle set is also extended according to a probabilistic motion model consisting of uncorrelated Gaussian angle and distance errors. The variances of these errors are estimated manually and hardcoded.

Since a path hypothesis implies a map we can measure the likelihood of the path given the consistency of the map.

In our formulation the map's affinity is additive over pairs of vertices so it can be computed incrementally as particles are extended; when paths cross themselves the walls observed around each are compared and the resulting affinity is added to the particle's value.

As the particles are extended, new observations will increase the weight of some particles and decrease the weight of others until many particles have weights orders of magnitude less than the most likely particle. The power of a particle filter[1] is that it provides a framework for dividing computational time evenly over weight of belief. This is ensured by periodically resampling the particles such that each new particle has equal weight. Resampling, however, reduces diversity in the particle set. Even in cases of equally weighted particles being sampled, some particles will be lost and others duplicated, leading to particles sharing long prefixes. This collapsing of the distribution to the most likely particle, or *particle depletion*, generally leads to the failure to close a cycle properly. In order to prevent particle depletion it is necessary to resample the particle set only when observations have changed the weight distribution.

This is accomplished by making the resample phase of the particle filter conditional depending on the current particle weight distribution. Each particle in the filter that has a weight much less than average is essentially wasted. To parameterize this we compute the number of bits of entropy in the particle weight distribution using $H = -\sum_x P(x) \log_2 P(x)$. The effective number of particles representing the path belief is 2^H and when the particle distribution is uniform (such as following resampling) this is exactly the total number of particles. The particles are only resampled when the fraction of effective particles falls below a threshold.

Since particles are extended, duplicated, and deleted but never otherwise altered it is natural to represent them in a common-prefix tree structure. Specifically, we store a tree in which a node consists of a candidate path delta for a particular path vertex as well as a pointer to the node storing the delta for the previous path vertex. A particle contains a pointer to a leaf of this tree. Each node in the tree contains a reference count and is deallocated when no nodes or particles reference it. A similar tree is used in [5] and [4].

3. PF CONSEQUENCES

An interesting statistic about the particle filter is the profile of the particle tree. The profile is the number of nodes at each depth divided by the total number of particles. This

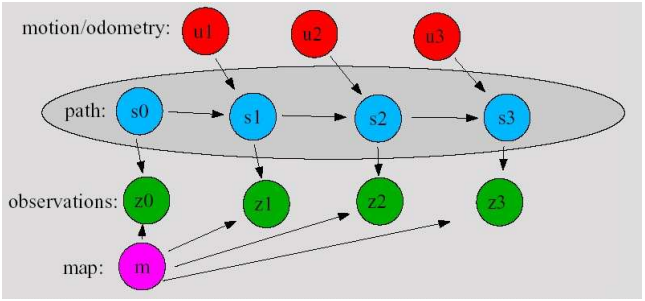

Figure 4: Rao-Blackwellization: when you sample the path (s0..s3) you can compute the probability of the implied map (m) from just the (newly) independent observations (z0..z3)

shows which sections of the path are known with relative certainty and which ones require further observations before they will be known. A specific case of this is the maximum depth with only one node; before the corresponding path vertex all particles are identical and thus that segment of the path is fixed.

3.1 Related Work

Several other roboticists have applied particle filtering techniques to the SLAM problem.

The underlying observation that makes this sort of algorithm useful is that observations, when conditioned on the robot's path, are independent. It is thus only necessary to approximate the path of the robot. This simplification is known as Rao-Blackwellization.[7]

Whereas Crunch has no sense of individual landmarks at all, most particle filter mapping systems uses the traditional SLAM model where all observations are of discrete landmarks. In such cases the data association problem must also be solved, that is, when a landmark is observed one must determine whether it is a new landmark or one of the previously observed landmarks. Particle filters offer an advantage over EKF based SLAM algorithms as each particle maintains its own hypothesis of data association. This prevents the case where an incorrect data association leads to catastrophic failure.[6] Crunch's low sensing ability makes the data association problem more difficult.

In [4] a robot equipped with a laser rangefinder is used and laser scans are matched in order to enhance dead-reckoning accuracy. In addition to providing more accuracy than odometry, scan matching provides a covariance matrix for the angular and range errors which is used to extend particles. This approach is more elegant and effective than Crunch's hardcoded variances and zero covariance but Crunch's sensors lack the range and accuracy for scan matching.

4. RESULTS

We drove Crunch manually around parts of the 4th and 5th floors of the Thomas J. Watson Sr. Center for Information Technology at Brown. The maps corresponding to the most likely particle are show in figures 5 and 6.

5. CONCLUSIONS

This paper has presented a new robot design as well as a new non-landmark FastSLAM implementation and supporting map representations and evaluation functions. While the results are not particularly impressive when compared with other FastSLAM implementations, they were obtained using a much smaller, cheaper, and less capable robot.

6. REFERENCES

[1] A. Blake and M. Isard. CONDENSATION - conditional density propagation for visual tracking, Sept. 14 1998.

[2] J. Borenstein and Y. Koren. The vector field histogram - fast obstacle avoidance for mobile robots. *IEEE Journal of Robotics and Automation*, 1991.

[3] W. Burgard, D. Fox, H. Jans, C. Matenar, and S. Thrun. Sonar-based mapping with mobile robots using EM. In *Proc. of the International Conference on Machine Learning*, 1999.

[4] D. Hahnel, W. Burgard, D. Fox, and S. Thrun. An efficient fastslam algorithm for generating maps of large-scale cyclic environments from raw laser range measurements. In *IROS-03*, 2003.

[5] M. Montemerlo and S. Thrun. Simultaneous localization and mapping with unknown data association using fastslam. Master's thesis, Carnegie Mellon University, 2003.

[6] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, Edmonton, Canada, 2002. AAAI.

[7] K. Murphy. Bayesian map learning in dynamic environments. In *Neural Info. Proc. Systems*, 1999.

[8] R. C. Smith and P. Cheeseman. On the representation and estimation of spatial uncertainty. *The International Journal of Robotics Research*, 1986.

[9] S. Thrun, M. Montemerlo, D. Koller, B. Wegbreit, J. Nieto, and E. Nebot. Fastslam: An efficient solution to the simultaneous localization and mapping problem with unknown data association. *Journal of Machine Learning Research*, 2004. To appear.

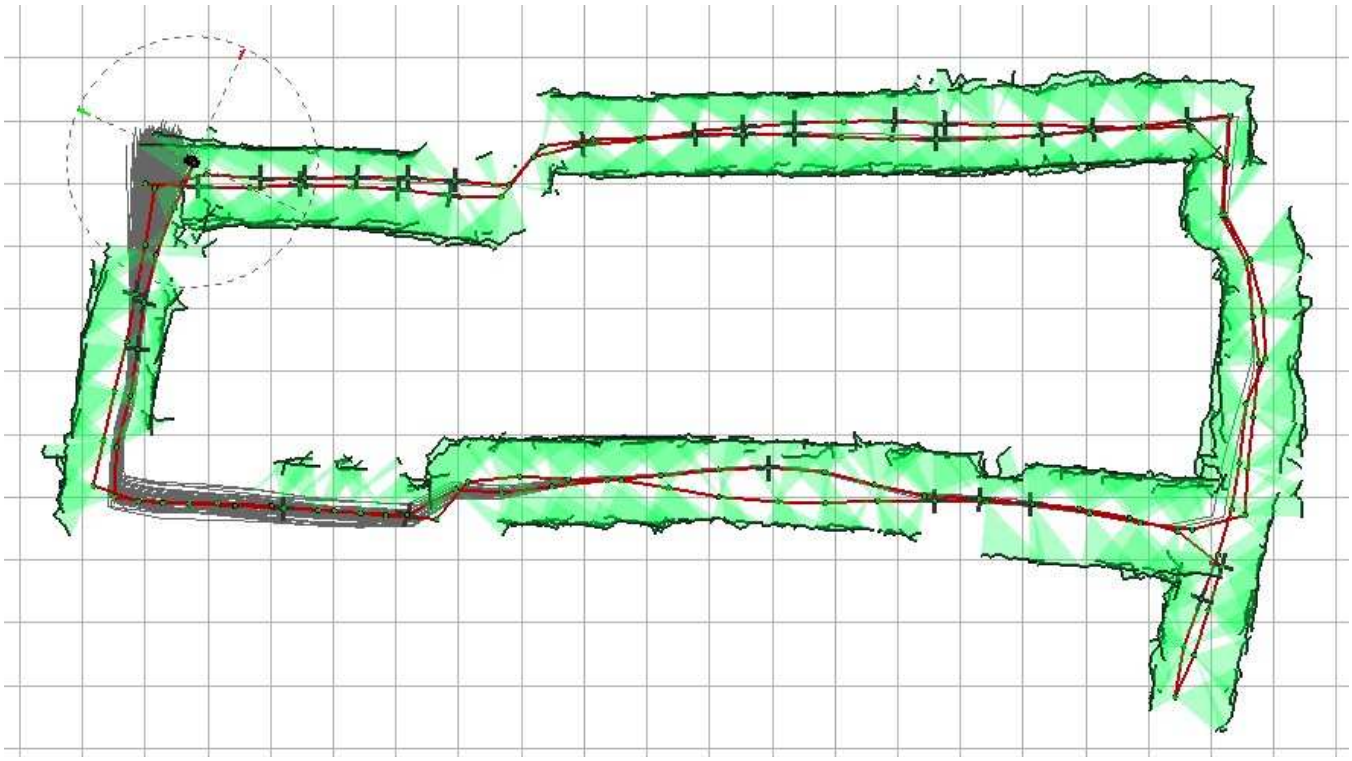

Figure 5: Two loops around an approx. 50m cycle on the CIT 5th floor. Map computed in real-time using 2000 particles. The terminal distribution of particles is shown.

Figure 6: A 250m drive from the AI Lab. Map computed offline using 5000 particles.