Vectorization of Grid Maps by an Evolutionary Algorithm

Ivan Delchev and Andreas Birk

School of Engineering and Science
International University Bremen
Campus Ring 1, D-28759 Bremen, Germany
a.birk@iu-bremen.de

Abstract. Mapping is a fundamental topic for robotics in general and in particular for rescue robotics where the provision of information about the location of victims is a core task. Occupancy grids are the standard way of generating and representing maps, i.e., in form of raster data. But vector representations are for many reasons, especially due to their compactness and the possibility to use very efficient computational geometry algorithms, highly desirable for many applications. Here a novel method for vectorization is presented that is intended to work particularly well with maps. It is based on an evolutionary algorithm that generates vector code for a so to say drawing program. The output of the evolving vector code is compared to the input grid map via a special similarity function as fitness. Experiments are presented that indicate that the approach is indeed a successful method to extract vector data out of grid maps.

1 Introduction

Mapping is a very important topic for rescue robotics for two quite different reasons. First of all, it is in general a core problem in robotics [Thr02]. Many fundamental algorithms for mobile robots simply depend on maps without which the system is restricted to be a crude tele-operated device that can hardly be called a robot. Second, maps are a fundamental added value of rescue robots over conventional systems for finding victims [BC06]. While the IUB rescue robot team [Bir05,BCK03,BKR+02] has been the first team to manage mapping in the challenging environment of the RoboCup rescue league in 2003 at the World Championship in Padua, this task meanwhile belongs to the standard challenges in this league [JMW+03,JWM03].

Probabilistic grids, also known as occupancy grids, are a well known approach to represent obstacles and free space by assigning according likelihoods to each cell in the grid [Mor88]. According to [Thr03], occupancy grids are the predominant method for generating, respectively representing maps. Occupancy grids can be easily generated, but they have their disadvantages. They represent maps as raster data, i.e., a collection of values arranged in a rectangular array. Due to its nature, it is difficult to manage and edit, requires a lot of space and contains

G. Lakemeyer et al. (Eds.): RoboCup 2006, LNAI 4434, pp. 458–465, 2007.

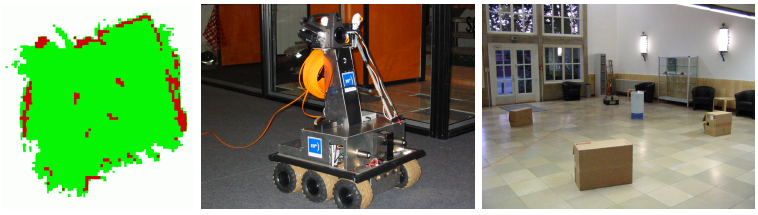

Fig. 1. A map (left) generated by an IUB rescue robot (middle) in the entrance hall of a building with a few boxes as obstacles (right)

no intrinsic semantics. Vector data is in contrast a compact format, that describes the geometry of a bar, i.e., a straight line segment with non-zero width, using a small number of attribute values, e.g., two endpoints and line width [DL99].

Many methods, differing in their precision and robustness, have been designed throughout the years for vectorization, i.e., the conversion of raster data into a vector format. The majority share the same structure - group pixels from a raster image into sets, approximate the sets with a set of vectors by some polygonal approximation method and post-process if necessary [TT00,SSTC02]. The classical approaches to vectorization use the following steps [CJ94]:

- Thin the image in order to extract its skeleton.
- Chain-code the thinned image.
- Reduce the chain codes to straight lines.

When the raster data is for example an image based on a line drawing, the standard methods work very well. But they are known to have difficulties with noisy data and "jerks" in the lines, two phenomena that are very common in robot maps. Here a vectorization technique is presented that is based on so-called reproductive perception, which tries to address this task in a non-classical way. Instead of processing the input raster data stepwise to produce its vector representation, exactly the opposite is done: namely vector code is evolved that is used to generate raster data, which is compared to the input data.

2 Overview of the Approach

Some terminology is introduced in this section that will be used in the remainder of the paper. First of all, note that grid maps can be thought of as images. Images, respectively grid maps are rectangular arrays of raster data where each pixel, respectively cell has a value that represents color, respectively the likelihood of free space. As many concepts in this paper are borrowed from image processing, the terms image and grid map are used in an interchangeable way, whatever is more appropriate in respect to its original context.

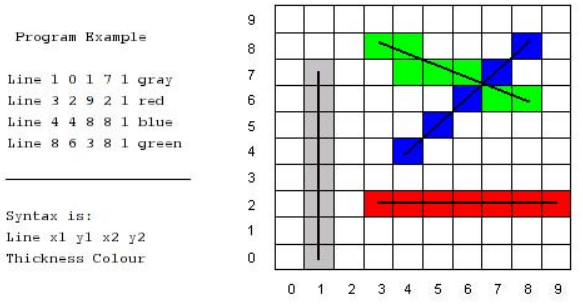

Fig. 2. A simple drawing program and its output

- A *Line* is a data structure that describes a single line - it contains x1, y1, x2, y2 of the endpoints as well as thickness and color.
- A *Program* is a data structure that contains a given number of *Lines*. Each *Program* describes an image containing straight lines and can be easily converted to a PNG or BMP format. The image described by a *Program* will be referred to as the output of the *Program*, shown in figure 2.
- *Population* is a data structure that contains a set of *Programs*.
- *Selection* is a set of *Programs* returned by a selection scheme.

The goal of the Reproductive Perception Vectorization algorithm (RPV) can be summarized in the following way. Given a raster image X like a grid map, create a *Program* Pr such that X', which is the output of Pr satisfies a similarity measure $D_{thresh}(X, X')$, i.e., Pr is the vectorization of X. The set of lines in Pr represents a compact mathematical model of the grid map X. If X' is constructed by using only lines of width 1, it is also the skeleton of X.

3 Fitness Calculation

The RPV employs an evolutionary algorithm approach to accomplish its goal. Evolutionary algorithms are a type of heuristic search techniques that incorporate principles of natural selection and "survival of the fittest". They maintain a population P that evolves at each iteration, according to rules called evolutionary operators. A fitness function $F : P \rightarrow \mathbb{R}$ assigns a value, referred to as fitness, to each individual in the population. A selection operator selects the "fittest" individuals from the current population, which are then used as input to the transformation operators. Individuals with a better fitness are more likely to yield superior ones after transformation [S.F93].

The fitness of each individual is calculated after each evolutionary step, therefore the choice of a good fitness function is vital for the performance of the algorithm. In our specific case, where each individual is a *Program*, a way to

determine the fitness of an individual is to compare its output X' with the original grid map X, and take into account its length in terms of number of Lines.

We define a measure of difference between two bitmaps called picture distance function $D : Bitmaps \times Bitmaps \rightarrow \mathbb{R}$, where $Bitmaps$ is the population of grid maps, including X. Because P may potentially be a very large population of programs and the algorithm may have to iterate through many generations, it is desirable that D is not computationally expensive. A fast image distance function, linear in the number of pixels in X', is introduced in [BJP00,Bir96]:

$$D(X, X') = \sum_c d(X, X', c) + d(X', X, c) \tag{1}$$

$$d(X, X', c) = \frac{\sum_{X[p_1]=c} min\{md(p_1, p_2) | X'[p_2] = c\}}{\#_c(X)} \tag{2}$$

where

- c is a color ($c \in \mathbb{C}$, $\mathbb{C}$ is the set of all colors)
- $X[p]$ is the color at position $p(i, j)$ in image X
- $md(p_1, p_2) = |i_1 - i_2| + |j_1 - j_2|$ is the Manhattan distance between p_1 and p_2.
- $\#_c(X)$ is the number of pixels in X having a color c

A very efficient way to compute $d(X, X', c)$ is described in [BJP00]. Finally, the fitness function F_{fit}: $Programs$ x $Bitmaps \rightarrow \mathbb{R}$, based on the picture distance is defined as $F_{fit}(Pr, X) = D(X, X', c) + 5 * len(Pr)$ where X' is the output of the drawing program Pr. The lower the fitness of Pr, the better the approximation.

4 Evolutionary Operators and Selection Scheme

The algorithm that is used here diverges from the common types of evolutionary algorithms such as genetic programming [RK94,RK92], genetic algorithms [Gol89], evolutionary programming [FOW66] by using a problem-specific set of genetic operators, which are presented below. Each operator creates a new individual, leaving the parents unchanged.

- Random Line Addition: Program → Program. Adds a Line L with randomly generated end point coordinates.
- Random Line Deletion: Program → Program. Randomly picks a Line from the set of Lines in the Program, and removes it.
- Concatenate Programs: Program × Program → Program. Concatenates the two Programs.
- Hill-climbing: Program → Program. Randomly translates the end points of a randomly picked Line from the input Program in a specified range. This procedure is performed k times and the best resultant individual is returned.

As selection operator we are using Roulette Selection - a randomized variant of fitness-proportionate selection. The chance of each individual to be selected is determined by its fitness. This is where the concept of "survival of the fittest" comes into play. However, because in our case a better individual has a smaller fitness (picture distance from the raster image), each individual X_i is assigned a value equal to X_{worst} - X_i. In this way the worst *Program* will have 0% probability of being selected and the best - the largest value compared to all other *Programs* in *Population*.

5 The Algorithm in Pseudo-Code

Algorithm *Vectorize*(*Image*, *Threshold*, *additionalargs*)
1. Create Initial Population();
2. **while** Best Individual Fitness ≥ Threshold
3. **do** Selection ←Roulette Selection();
4. Evolve(Selection);
5. Add Selection To Population;
6. Remove Extra Individuals();
7. Save Best Individual;

Algorithm *Evolve*(*Selection*)
1. Random = Random Number(0,1);
2. Operator ←Determine Evolutionary Operator(Random);
3. **for** i ←1 **to** Selection.Size - 1
4. **do** case: Operator = Random Line Addition
5. Add Random Line(Selection[i]);
6. case: Operator = Random Line Removal
7. Remove Random Line(Selection[i]);
8. case: Operator = Hill-climbing
9. Hill-Climbing(Selection[i]);
10. case: Operator = Concatenate Two Programs
11. Concatenate(Selection[i], Selection[i+1]);
12. ++i;

As a first step an initial *Population* of 50 *Programs* is created. Each *Program* contains a random number of *Lines* (between MIN and MAX specified either as input arguments to the algorithm or as global constants). Each *Line* is also randomly created in the context of a target grid map (i.e the dimensions of the grid map are known so the Line coordinates are on the grid map). At creation time the fitness of each *Program* is evaluated, taking into account both the picture distance and the number of *Lines* present in the *Program*.

The initial *Population* is evolved until a *Program* that is a good enough approximation of the target grid map is produced. At each iteration, a number of individuals called *Selection* is chosen via a selection routine, in our case Roulette Selection. An evolutionary operator that either evolves or mates individuals is applied on *Selection*, the parents remain unchanged. The frequency of each operator is determined by its probability of occurring, specified as an input to the algo-

rithm. As mentioned before the evolutionary operators are - Random Line Addition, Random Line Removal, Concatenation of two Programs and Hill-Climbing on the end points of a Line. The best performance is reached when probabilities of 30%, 25%, 15% and 30% respective to the above-mentioned operators are used.

At the end of each iteration, the same number of *Programs* that were added beforehand is removed via inverse roulette selection, i.e., by favoring worse individuals, in order to keep the population size constant. This approach retains a certain diversity in the population and at the same time quickly converges to the desired target grid map.

6 Experiments and Results

Figure 3 shows the result with an experiment within a typical office environment. The grid map contains about 10 KBytes of data that is reduced to a few hundred bytes to represent 19 vectors. Also, noisy and spurious data in the grid map has more or less disappeared in the vector representation. More important than the compression is that the vectors can serve in contrast to the raster data as basis for efficient computational geometry operations that are very beneficial for many robotic algorithms as discussed in the introduction.

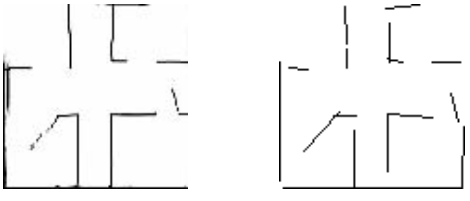

Fig. 3. On the left, a grid map from a typical office environment. Its vectorized representation is shown on the right. The EA has removed some noise and spurious data and generated out of the about 10KBytes of raster data a compact representation with just 19 vectors.

The trend of the average fitness of the whole population is shown in figure figure 4. On a standard PC with a Pentium-4 2.2GHz processor, it takes in the order of 250 msec for computing a whole generation. Analyzing the trends it can be observed that the RPV algorithm behaves like a typical evolutionary algorithm. During the first iterations, the population very quickly improves its fitness, while in the late stages it converges slowly to the target grid map. For many purposes like scan matching in SLAM, the fast rough matches after a few iterations are fully sufficient.

In the following example, a very dense number of lines is used. The purpose of this test case is to test whether the RPV algorithm is able to deal with complicated problems. It managed to evolve a good approximation of the input, although it requires a larger number of iterations. The input and output images are shown in figure 5. The EA performs quite similar to the previous case. The

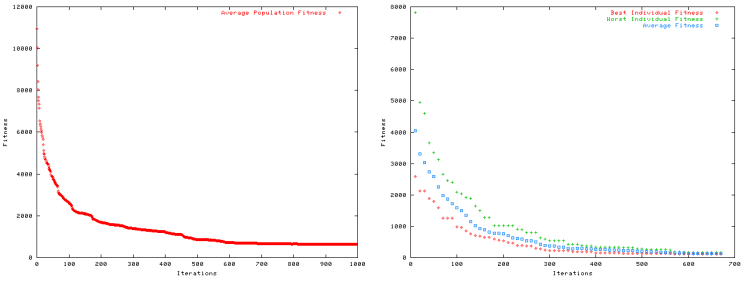

Fig. 4. On the left, the average fitness of the population when generating the vectorization of an office environment. On the right, the best, worst and average Fitness of the population in an experiment with a very crowded set of lines.

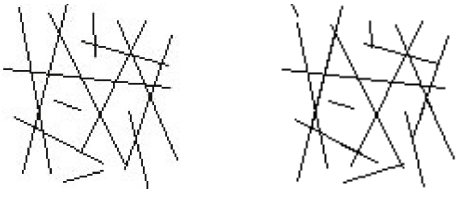

Fig. 5. An example with very "crowded" lines. The original raster data is shown on the left, a result of vectorization on the right.

population does not contain large discrepancies as can be seen from figure 4 - worst to best fitnesses ratio is roughly 2:1, which is a favorable condition, since there are hardly local minima, that could hinder the evolution.

7 Conclusion

Grid maps are widely used to generate a representation of a robot's environment as they are very well studied and as they can be easily generated. But grid maps as a form of raster data have disadvantages, especially when it comes to utilizing the data. Vector data in contrast is very compact and it can be very efficiently processed by computational geometry algorithms, for example for feature extraction or pattern recognition for SLAM or map merging. Here a novel method for vectorization was presented. It is based on an evolutionary algorithm that generates vector code, which represents the input raster data.

References

BC06. Birk, A., Carpin, S.: Rescue robotics - a crucial milestone on the road to autonomous systems. Advanced Robotics Journal 20(5) (2006)

BCK03. Birk, A., Carpin, S., Kenn, H.: The iub 2003 rescue robot team. In: Robocup 2003. Team Description Paper (TDP) (2003)

Bir96. Birk, A.: Learning geometric concepts with an evolutionary algorithm. In: Proc. of The Fifth Annual Conference on Evolutionary Programming, The MIT Press, Cambridge (1996)

Bir05. Birk, A.: The IUB 2004 rescue robot team. In: Nardi, D., Riedmiller, M., Sammut, C., Santos-Victor, J. (eds.) RoboCup 2004. LNCS (LNAI), vol. 3276, Springer, Heidelberg (2005)

BJP00. Birk, A., Paul, W.J.: Schemas and genetic programming. In: Ritter, Cruse, Dean (eds.) Prerational Intelligence, vol. 2, Kluwer, Dordrecht (2000)

BKR+02. Birk, A., Kenn, H., Rooker, M., Akhil, A., Vlad, B., Nina, B., Christoph, B.-S., Vinod, D., Dumitru, E., Ioan, H., Aakash, J., Premvir, J., Benjamin, L., Ge, L.: The IUB 2002 rescue robot team. In: Kaminka, G., Lima, P.U., Rojas, R. (eds.) RoboCup 2002. LNCS (LNAI), Springer, Heidelberg (2002)

CJ94. Jennings, C., Parker, J.R.: Vision knowledge vectorization: Converting raster images into vector form, pp. 311–315 (1994)

DL99. Dori, D., Liu, W.: Sparse pixel vectorization: An algorithm and its performance evaluation. IEEE Transactions on Pattern Analysis and Machine Intelligence 21, 202–215 (1999)

FOW66. Fogel, L.J., Owens, A.J., Walsh, M.J.: Artificial Intelligence through Simulated Evolution. Wiley, New York (1966)

Gol89. Goldberg, D.: Genetic Algorithms in Search Optimization and Machine Learning. Kluwer Academic Publishers, Dordrecht (1989)

JMW+03. Jacoff, A., Messina, E., Weiss, B., Tadokoro, S., Nakagawa, Y.: Test arenas and performance metrics for urban search and rescue robots. In: Proceedings of the Intelligent and Robotic Systems (IROS) Conference (2003)

JWM03. Jacoff, A., Weiss, B., Messina, E.: Evolution of a performance metric for urban search and rescue. In: Performance Metrics for Intelligent Systems (2003)

Mor88. Moravec, H.: Sensor fusion in certainty grid for mobile robots. AI Magazine 9(2), 61–74 (1988)

RK92. Koza, J.R.: Genetic programming. MIT Press, Cambridge (1992)

RK94. Koza, J.R.: Genetic programming II. MIT Press, Cambridge (1994)

S.F93. Forrest, S.: Genetic algorithms - principles of natural selection applied to computation. Science 261, 872–878 (1993)

SSTC02. Song, J., Su, F., Tai, C.-L., Cai, S.: An object-oriented progressive-simplification based vectorisation system for engineering drawings: Model, algorithm and performance. IEEE transactions PAMI 24(8), 1048–1060 (2002)

Thr02. Thrun, S.: Robotic mapping: A survey. In: Lakemeyer, G., Nebel, B. (eds.) Exploring Artificial Intelligence in the New Millenium, Morgan Kaufmann, San Francisco (2002)

Thr03. Thrun, S.: Learning occupancy grids with forward sensor models. Autonomous Robots 15, 111–127 (2003)

TT00. Tombre, K., Tabbone, S.: Vectorization in graphics recognition: To thin or not to thin. In: ICPR, pp. 2091–2096 (2000)